Addis Ababa University

College of Natural Sciences

Bug Triage Model Using Machine Learning Techniques

Meseret Andualem

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

August 2021

Addis Ababa University

College of Natural Sciences

Meseret Andualem Beyene

Advisor: Ayalew Belay(PhD)

This is to certify that the thesis prepared by Meseret Andualem, titled: *Bug Triage model using machine learning techniques* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name	Signature	Date
Advisor: Ayalew Belay(PhD)	_____	_____
Examiner: Mulugeta Libsie (PhD)	_____	_____
Examiner: Dida Midekso (PhD)	_____	_____

Abstract

A customer, analyst, or developer makes an error while using the system or generating software artifacts, which is a common chain of reaction to software bugs. This error may cause a system fault, resulting in an unexpected state. This unanticipated situation could lead to a bug, which is a visible and unwelcome event from the user's perspective. When a bug is detected, the user creates a bug report that includes the error messages issued by the software. These bugs must be dealt with in a proper manner. Bug triage is one of them. Triaging is the process of categorizing and prioritizing bug reports in order to assign priority and route them to the appropriate developer for resolution.

Many researchers have developed a number of bug triage models in recent years. However, because bugs in bug repositories aren't necessarily bugs, new strategies to identify the actuality of the bug are still needed. As a result, we present a bug triage model that uses machine learning techniques to identify actual bugs from non-bugs by assessing the severity field and then assigning to the right developer based on the developer's tossing history.

Preprocessor, feature extractor, dataset constructor, bug detector, and bug assigner are the components of the suggested model. Data tokenizing, stop word removal, and stemming are the main operations in the preprocessing component. The feature extraction component then extracts the feature vectors from the supplied bug report, while the dataset constructor splits the dataset into training (80%) and testing (20%) sets by converting the javascript object notation (JSON) file into sets. Finally, based on the recognized component retrieved from the bug report's short description, the MNB classifier is used to classify the bug report into BUG and NON BUG and automatically propose developers who have the necessary competence for processing a bug report.

The feasibility of our proposed model has been validated on Eclipse covering 315228 bug reports. We show that our techniques can detect bugs and assign them to them with a prediction accuracy of up to 95.67 percent.

Keywords: Bug Triage, Bug Detection, Bug Assignment, Multinomial Naïve Bayes

Dedication

To My Family

To my family for their love and support.

Acknowledgments

First and foremost, I want to express my gratitude to God for providing me with the chance, knowledge, and strength to carry out and complete research work. Without His blessings, none of my accomplishments would have been possible.

Then, I would like to express my heartfelt gratitude to my advisor, Dr. Ayalew Belay, whose knowledge and guidance were vital in steering this research work. Your insightful critiques, direction, and encouragement pushed me to improve my thoughts and elevate my work.

Obviously, it would not be possible without the support of my family, who have been there for me throughout my journey and beyond. I would like to express my gratitude for your prayers, love, and support, which have enabled me to attain this magnificent opportunity that I've always desired.

Thank you, Ato Yohannes Girmay, for being so humble in sharing your ideas and worries about this research work. Last but not least, I would want to thank my coworkers for keeping me going; without their unwavering support and patience, this thesis would not have been feasible. Classmates and instructors, it has been a great time and support in filling the gaps I have faced in my Postgraduate year.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

- BTS - Automated Bug Tracking System
- JSON -- JavaScript Object Notation
- KNN - K Nearest Neighbor
- LDA - Life Data Analysis
- LSTM -- Long Short-Term Memory
- ML -- Machine Learning
- MNB -- Multinomial Naïve Bayes
- NB - Naïve Bayes
- NLP -- Natural Language Processing
- NLTK --Natural Language text processing toolkit
- NR – Non- reproducible
- QA – Quality Assurance
- S2Rs -- Steps to Reproduce
- TDS -- Training Data Set
- TF-IDF -- Term Frequency Inverse Document Frequency
- TM -- Text Mining
- VDS --Verified Data Set
- VM – Virtual Machines
- VSM -- Vector Space Model
- XML -- eXtensible Markup Language
- XSLT -- eXtensible Stylesheet Language Transformations

1. Introduction

1.1 Background

Nowadays, Information Communication Technology (ICT) is the leading technology that provides technology based services for different needs of human beings. For this purpose, the software development process takes a base and a major part for the development and use of the technology need, which is developed by software engineers in collaboration with subject matter specialists.

Errors and bugs are common during software development. These bugs must be dealt with appropriately during the development phase. Bug triage is one of them. Triaging is the process of categorizing and prioritizing bug reports in order to assign priority and refer them to the appropriate developer for correction of the source of the software's malfunction [1]. In the majority of cases, the bug report is authored by the user and complemented by the software's error messages.

Bug triage is a method of prioritizing bugs based on their severity, frequency, risk, and other factors. The term "triage" is used in software testing and quality assurance to describe the severity and importance of new issues [2]. Bug triage, according to Anjali [3], is the process of assigning an appropriate developer to a bug report who can make code changes to resolve the bug.

When a software system fails, a bug report is often sent using a bug tracking system. The developers then work as swiftly as possible to validate, locate, and fix the reported bug. Bug detection and fixing in a timely manner is an important software engineering discipline. Bug tracking solutions like Bugzilla are routinely used to manage large number of bugs [4]. Most defects are assigned to developers manually, which is a time-consuming procedure, especially for large software projects. The Eclipse and Mozilla projects, for example, get hundreds of bug reports per day and allocate each one to one of the thousands of developers.

Thousands of bug reports are made to the issue tracking system for software that is publicly available and freely distributable around the world. Bugs on the Bugzilla bug repository, on the other hand, aren't always bugs. Feature requests, refactorings, and other such requests are

common. Many ways for automatically finding flaws in software have been developed over the years.

A software developer's error is a blunder or misinterpretation. As a result of an error, a flaw (defect) gets introduced into the software. It is a flaw in the software that could lead it to behave erroneously and in ways that aren't specified. Bugs are a term used to describe flaws or problems. The inability of a software system or component to accomplish its needed functions within stated performance requirements is referred to as a failure.

As a result, there is no distinction between a non-bug and a bug. However, distinguishing between bugs that are genuinely bugs and bugs that aren't is critical for bug triage success [5].

A triager is responsible for assessing bug reports and assigning them to the right developer. Selecting the appropriate developer to fix a new bug report is an important stage in the bug triaging process, and it has a substantial impact on the time it takes to repair bugs and the cost of projects. The challenge of identifying the best capable developer for the bug type [6] is one of the major reasons why bug triaging takes so long.

In this research, we use machine learning to construct a model that detects bug reports to identify real defect, and then assigns them to the most suited developer automatically.

1.2 Motivation

As the demand for ICT grows dramatically in all areas for the provision of good service for humans, software development plays an increasingly important role in the overall process.

As long as software development is ongoing, new bugs will emerge that must be addressed. This demonstrates that addressing these bugs is complex and time intensive, not to mention expensive. This is supported by a survey undertaken by the National Institute of Standards and Technology, which found that the yearly cost of software vulnerabilities is estimated to be around $59.5 billion [7]. According to some software maintenance surveys, maintenance costs account for at least 50%, and sometimes even more than 90%, of the overall expenses associated with a software product [8].

As a result, bug triaging is in continuous research attention as it severely affect IT industry. Hence, it is essential to look for solutions to monitor the bug report so as to detect the actuality of the reported bug and assign to the concerned developer.

This work is motivated based on the following two issues: no detection of the reality of incoming bug report and assignment of those real bug reports to the most appropriate developer. In general, due to the evolving nature of the bug reports they cannot be dealt with by existing triaging techniques and improvements are required. Therefore, our study has a meaningful significance on the bug triaging.

1.3 Statement of the problem

IT organizations now spend more than 45 percent of their budgets on software problem fixes [9]. Because of the rapid advancement of software technology, the quantity of software products being generated has increased dramatically, making software maintenance difficult.

The majority of software development companies' development processes have become more complicated as the number of defects generated on a daily basis has increased [10]. Detecting and fixing this defects before the deadline is the most important component of the software development lifecycle. A software fault is a condition in which a software product fails to meet a software requirement or the needs of the end user. In other words, a defect is a coding or logic fault that causes a program to fail or provide unexpected and wrong outcomes.

The concept of bug patterns is used in a simple static analysis methodology for discovering defects/bugs. A bug pattern is a code idiom that is likely to produce an error, and bug patterns develop when code does not follow standard operating procedure when using a language feature or library API [11].

Despite the fact that many researches have been conducted on automated bug assignment methods, the reality is that most defects are not found before they are assigned to developers. Based on the identified components gathered from the short description of the issue report, Saagarikha *et al*. [12] developed a strategy to automatically propose developers who have the necessary competence for handling a bug report. Their research is the first to look at the effects of several machine learning features (classifiers and training histories), as well as a ranked list of developers, on bug assignment prediction accuracy. This research is tested on Eclipse project. However, the study did not take into account whether or not the new bugs are indeed bugs.

In another study, Arudkar and pimplakar [9] proposed using text categorization and machine learning techniques to identify which developer should be assigned to an issue based on its description. Two large open-source projects, Eclipse and Mozilla, participated in the authors

experiment on data reduction for bug triage in bug repositories. Their proposed solution was built on random forest, which is a data processing methodology that reduces scalability while providing high-quality bug data in software development and maintenance. This study, however, did not involve the detection of defects before they were assigned to the developer

Furthermore, Hiwse [6] provided a bug classifier so that they could establish the type of the bug to which that bug belongs and, after applying the classification, they could assign the bug to the appropriate developer to solve it. The authors used a combination of Naive Bayes (NB) and k nearest neighbor classification techniques (KNN). They also used prepossessing to introduce a bug database reduction mechanism. However, This Study also did not involve the detection of defects before they were assigned to the developer.

As a result, the goal of this research work is to create a model for bug triage that uses machine learning techniques to forecast which engineers should be allocated to which bugs based on determining which bugs are genuinely bugs. Unlike many previous efforts that did bug detection and bug assignment separately, this work may create and implement an automatic bug triaging model which combine bug detection and bug assignment at the same time.

In a software development process, defect-detection techniques are used to identify faults before they are allocated to a developer. As a result, it is vital to evaluate the prices of errors before and after they are allocated to the developer. The goal is to cut down on bug triaging time and effort.

1.4 Objectives

General Objective

The general objective of this research work is to develop a bug triage model using machine learning technique which will help in promoting effective and timely bug triaging processes.

Specific objectives

In order to achieve and reach the ultimate purpose of this general objective successfully, the following specific objectives are set.

- Review the functional processes and states of the art of the bug triage.
- Identify and collect datasets on the existing defect repositories of bug triage.
- Identify metrics and tools for the bug triage model development.

- Develop a prototype which demonstrates the developed model
- Evaluate the performance of the developed model

1.5 Methods

An effort will be made to apply a suitable technique for the study, keeping in mind the aims. Based on that, among the various types of approaches, design science research approach is selected. The design science research approach aims to build innovative constructs, ideas, methods, and models through analysis, design, development, and evaluation procedures [13].

Purao [13], Peffers *et al*. [14], Hevner *et al*. [15], Vaishnavi *et al*. [16], Nunamaker *et al.* [17] have published a number of process models and descriptions with diagrams of design science research procedures. However, the activities to be undertaken in similar phases of design science research vary based on the design science research context, resulting in changes in the process models [18].

The design science research process model created by Peffers *et al.* [14] will be used in this study to build and assess the bug triage. The problem identification and initiation, as indicated in Figure 1, are the starting points for this study. During this phase, a survey on the current level of bug triage work will be done, and metrics will be synthesized in the meantime.

A bug triage model will be constructed based on the data gathered from various literatures. The model will be demonstrated using a prototype and assessed to determine its efficacy.

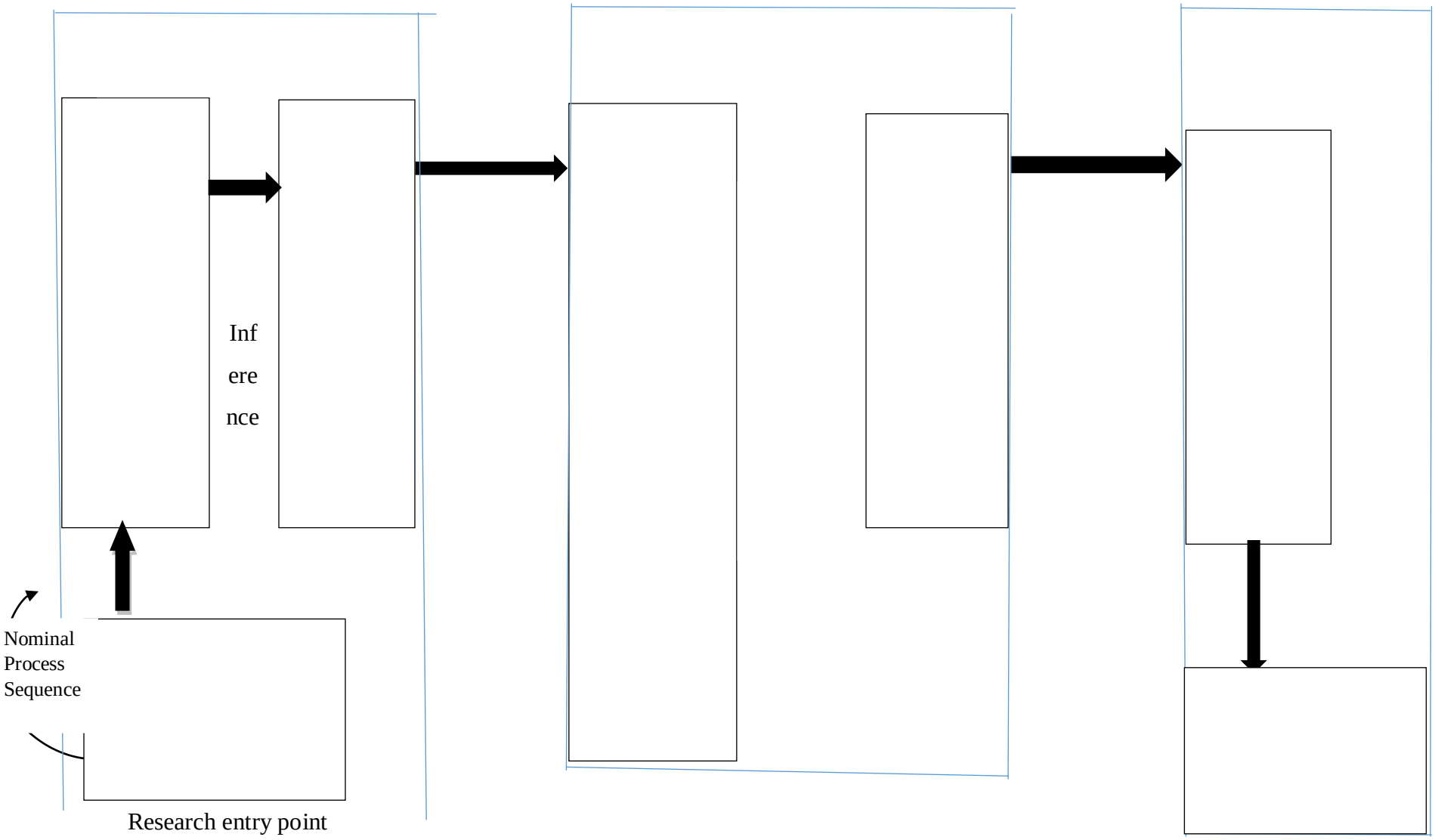


Figure 1.1 Design Science Research Process for bug triage

1.6 Scope and Limitations

- **Scope**

The research mainly focuses on detecting the actuality of bug reports in the bug repository. The aim is to be able to distinguish genuine bug reports from non-bug and assign to the most appropriate developer using machine learning approaches for the Eclipse project.

- **Limitations**

In this research, we will not study the technique to prevent or remove non-bug reports in the bug repository.

1.7 Application of results

This study's findings are expected to contribute to the enhancement of bug triage in the development processes to different beneficiaries, such as software businesses and engineers. Software development organizations can use the developed model with the specified steps for their error debug processes in their development area, which aids them in achieving an effective and improved performance of process error bug detection and fixation, thereby strengthening software development process and service management more easily. It is also used as input by other stakeholders, researchers, and students as well.

1.8 Organization of the Rest of the Thesis

The rest of this thesis is organized as follows. In Chapter Two, a variety of literatures are used to describe the topic of bug reports, bug repositories, reasons and consequences of bug report misclassification, and manual and automatic triaging. The Third Chapter examines related research on bug triage and detecting the truthfulness of bug reports. The proposed model's design is presented in Chapter 4, and the simulation results are shown in Chapter 5. The conclusion, recommendations, and future work will all be found in Chapter Six.

2. Literature Review

2.1 Introduction

This Chapter delves into the domain knowledge concepts of bug report, bug repository, reasons and consequences of bug report misclassification, and manual and automatic triaging. We believe that offering explanations in this regard is vital to help the reader understand the knowledge that was employed in the study.

2.2 Bug report

Software problems are inconvenient for developers, and they can result in catastrophic consequences [4]. Users and/or developers report bugs to issue trackers when software does not function as planned. Bug reports, which are papers that explain software flaws and are meant to contain the information needed for developers to triage and solve the faults in the product, are commonly reported [19].

A bug report, according to Anjali and Sardana [20], is "a document containing complete specification related to a bug." A bug report is made up of a number of pre-defined meta-fields as well as free-form textual content. Bug id, product, component, operating system, platform, milestone, severity, version, status, resolution, reporter, reported date and time, assigned to, and so on are among the predefined meta-fields. Keywords, summary (or tagline), description, and comments are all included in the textual content. According to Anjali and Sardana [20], the term "summary" refers to a one-line brief description of the defect. The term "description" refers to the reporter's whole detailed specification for the submitted bug. It is the major content of the bug report, and it usually includes the procedures for reproducing the problem. The open discussion by a group of people to discuss and review the solutions for the reported bug is referred to as "comments." This group of persons has a general understanding of the bug's associated area of expertise. Sentiment analysis is a technique for extracting, identifying, and characterizing a text unit's emotive content. The natural language text processing (NLTK) toolkit was used to analyze the attitudes in bug reports.

Bug reports frequently include the steps to replicate (S2Rs) the bug [21], in addition to the observed and expected behavior. The S2Rs are critical in assisting developers in reproducing and correcting issues. Unfortunately, the S2Rs are frequently vague, partial, or confusing. Low-

quality S2Rs, according to Cárdenas et al. [21], result in non-reproducible issues, unfixed issues, and a lot of manual time spent on problem triage and resolution.

Bug tracking repositories are used in large software projects, where users and engineers submit all issues they find. Developers attempt to reproduce bugs using information provided by a reporter in a bug report, and then make the necessary changes to the source code to resolve the problem. The majority of problems are caused by software faults and human error. Almost every piece of software or program will have a significant number of flaws. As a result, when users discover defects, they will report them to the developers [22].

Bug databases are now routinely analyzed to determine where bugs have happened in the past and anticipate how they may arise in the future. Different types of bug databases were described by Suo and Zou [22], including arithmetic bugs, interfacing bugs, team working bugs, syntax bugs, logic bugs, and so on. Semantic modifications to source code are always requested in bug reports. Other issue reports ask for various remedies, such as code fixes, new feature additions, documentation updates, internal refactoring, and so on. If the request does not pertain to the correction of coding flaws, it will be classified as a non-bug problem. Bug, Request for evidence, improvement, document, refactoring, and other software bug classification criteria and categories were described by Herzig *et al* [23].

BUG: is defined as outlining corrective maintenance tasks that need semantically changes to source code" is what BUG defined.

RFE is defined as issue reports documenting an adaptive maintenance task whose resolving patch(es) implemented new functionality," and is defined as "Issue reports documenting an adaptive maintenance task whose resolving patch(es) implemented new functionality."

IMPR is defined as issue reports detailing a perfective maintenance work whose resolution improved the overall handling or performance of current functionality.

Documentation Requests (DOC) are defined as "Issue reports resolved by changing external (e.g. website) or code documentation (e.g. JavaDoc)."

REFAC is defined as refactoring request and is defined as "issue reports resolved through refactoring source code." These reports were typically filed by developers."

Other Requests are defined as "any issue report that does not fall into any of the other categories." This includes requests for a back port (BACKPORT), code cleanups (CLEANUP),

modifications to specifications (rather than documentation or code; SPEC), general development tasks (TASK), and test case issues (TEST).”

An issue report will be classified as a bug if it needs corrective code maintenance, according to Herzig *et al.* [23]. Some issue reports, on the other hand, call for "perfective and adaptive maintenance, refactoring, debates, help requests, and so on." Antoniol *et al*. [24] have highlighted that certain issue reports are labeled as bugs, however they are actually non-bug issues. Herzig *et al*. [23] concluded that issue report classification is unreliable based on the categorization results:

Almost every other bug is not actually a bug: Corrective code maintenance is not mentioned in 45.1 percent of all bug reports. It is easy to get the terms "feature request" and "improvement request" mixed up: There are 51.9 percent of reports in Type-Enhancement that are Feature Requests and 27.3 percent that are Improvement Requests. Tasks, reviews, and other classifications are readily muddled: Task, Review, and Other all have 100% misclassification rates.

2.3 Sources of Non-Bugs

Because users and developers have varied perspectives and insights on bug classification, bug tracking systems contain a large number of non-bug reports, and this classification is not corrected once a bug has been handled [22]. Bug tracking systems are communication tools that allow users to submit bugs that developers will solve. However, users and developers may have differing perspectives on the project's internals. Users may lack project understanding and the ability to comprehend technical project details in many circumstances. Every problem is viewed as a bug by users. They file a bug report if the software does not meet their expectations or the documentation provided. The reporter is the one who categorizes the problems.

The developer, on the other hand, is the expert on any technical component of the software because s/he has designed and developed it. This distinction between reporter and resolver is already a topic of debate. A developer, unlike a reporter, has the capacity to discriminate between various problems and the necessary maintenance activity to resolve the issue. To categorize issue reports, the developer would be the best person to ask. Bug trackers, on the other hand, do not work in this manner. Of course, the developer has the option of changing the issue

category after it has been resolved, although this is uncommon. Once the root cause of a problem has been identified and remedied, there is often little reason to modify the issue category.

It is difficult to tell whether a problem is a bug or not. The concept of a bug, according to Herzig *et al.* [23], varies not only between consumers and developers, but even amongst developers themselves. In general, if a problem disturbs a user, the developer should repair it, whether or not it is considered a bug. The category of the issue report is irrelevant in this case. However, when creating a defect prediction model, a data miner must distinguish between bugs and non-bugs. Otherwise, the prediction model would forecast flaws rather than changes.

2.4 Bug Repository

Software repositories are large-scale databases used in modern software development to store the output of software development, such as source code, bugs, emails, and specifications. Bug repositories (also known as bug tracking systems) are used in large software projects to help developers collect information and manage bugs. A bug is stored as a bug report in a bug repository, which maintains the textual description of replicating the bug as well as changes [5].

The bug report repository [25] is an important collaborative center for many software development initiatives. A bug repository can help with the development process in a variety of ways, including recording the work that needs to be done and allowing developers who are spread out geographically to communicate about project development.

Several bug trackers [26] are built to work with distributed revision control tools. These distributed bug trackers allow bug reports to be read, added to the database, and updated even when the computer is turned off. Its query functionality, however, is not as advanced or user-friendly as that of a few other non-distributed issue trackers like Bugzilla. Bugzilla is the most widely used open source bug tracking repository, with Eclipse, Mozilla, Apache, the Linux kernel, and other significant software projects using it.

Bug tracking tools are especially crucial in open source software development, as team members may be spread out throughout the globe [27]. The bug tracking system is used to organize work among the developers as well as to keep track of problem reports and feature requests. The majority of bug-tracking systems allow for additional comments to be added to bug reports. Successful large open source projects, on the other hand, face the challenge of managing the

influx of new reports. Open source projects, such as Eclipse, contain bug repositories that are open to the public. These repositories get bug reports from users, who are typically non-technical. Effectively deciding what to do with a fresh report, according to Cubranic and Murphy [27], can be a challenge. It takes time to determine whether the report is truly a bug and which developer should be assigned to it. Antoniol *et al*. [24] brought out the issue of misclassified bug report, or reports labeled as bugs but pertaining to non-bug report. Allocating the correct bug to the correct developer is challenging for a triager who does not know the specifics of the bug [28]. Every third bug, according to Herzig *et al*. [23] findings, is not a bug. Corrective code maintenance is not mentioned in 33.8 percent of all bug reports.

Many researchers [23, 29, 30, 24] have found that many submitted reports have been labeled as bugs when they are not. This misclassification causes bias and has a significant impact on bug triage. However, in the case of huge bug reports, human classification can help reduce noise. As a result, an automatic bug classification system is required to determine whether a given bug report is, in fact, a bug. The study studied the viability of automatic bug report classification using text mining techniques in [24]. However, the work was limited to the bug reports' description section (free text).

Kukkar and Mohana [46] used a hybrid approach to determine whether a given bug report is a real bug by combining text mining (TM), natural language processing (NLP), and machine learning techniques (ML) by incorporating four fields (reporter, severity, priority, and component) with textual information such as comments, description, and summary.

2.5 Impacts of Misclassification on Bug Triage

The categorizing of bug reports aids developers in determining where problems occur in software and how to address them. The erroneous classification will have an impact on several related studies that use the data sets without first validating them. The data quality of bug databases can be reflected in misclassification rates. The cause for misclassification, according to Suo and Zou [22], is that consumers and engineers have differing perspectives on bug classification. Users who report issues frequently do not understand the distinction between an improvement, a feature request, and a bug. As a result, the primary responsibility of developers is to allocate these issue reports.

When an issue report is misclassified, it signifies we haven't worked out what the request is, and the response will most likely be ineffective and useless. “Misclassification affects the relative ranking of the most defect-prone files,” according to Herzig et al. [23].

Marking a non-issue as a bug by mistake can have major ramifications when it comes to seeking a solution. Developers typically spend a significant amount of time attempting to replicate them without success.

2.6 Bug Triaging Mechanism

Bug triage, according to Sardana et al. [32], is the process of identifying an appropriate developer for a bug report who can make code changes to resolve the bug. In the literature, various ways ranging from semi-automated to fully automatic bug assignment have been presented. Machine learning and information retrieval techniques are largely used in these approaches. The bug report assignment is formulated as a classification problem using machine learning techniques, with the various meta-fields (or non-textual contents) and textual contents acting as features and the developers acting as class labels. The predominant developer for new bug reports is predicted by the classifier based on previously fixed bug reports [2].

Bug reports are converted into feature vectors using information retrieval techniques. It is based on the idea that developers with similar experience in a particular issue category are capable of solving a new bug in that category. The time element is significant in determining a renowned developer for a specific category of issue at a specific period [4]. This is due to the fact that a person's abilities and knowledge deteriorate over time. Furthermore, the longer a developer goes without using a term, the faster that term's profile will deteriorate. Because knowledge decays with time, time factor based normalization in activity profiling could be very useful in measuring the level of competence (or knowledge) of developers. Recent research have used the information retrieval based activity profiling methodology more frequently because it achieves higher accuracy.

2.7 Manual and Automated Triaging

Users, developers, and testers of software systems report all issues they find to bug tracking repositories, which are then examined by a senior developer, referred to as a bug triager, to confirm the bug's presence. After that, the bug triager sends the verified bug reports to a developer based on the developer's experience and current workload. The assigned developer

works on the bug report to resolve and correct the problem. Manual bug assignment (also known as developer assignment) is a time-consuming, error-prone, and labor-intensive approach [33].

Bug detection and fixing in a timely manner are critical software engineering methods. Bug tracking systems like Bugzilla are routinely used to handle large numbers of bugs [34]. Most defects, on the other hand, are assigned to developers manually, which is a time-consuming procedure, especially for large software projects. The Eclipse and Mozilla projects, for example, get hundreds of bug reports per day and allocate each one to one of the thousands of developers. This is a difficult task that is prone to errors. Developers can reassign a bug report to other developers after it has been allocated; this is known as bug throwing.

According to Shaik *et al*. [35], a total of 37 percent to 44 percent of bugs have been tossed to another developer at least once. Bugs are sometimes given to developers by mistake, which is one of the most prevalent reasons for bug chucking. A developer, for example, may not own the flawed code ("not mine") or lack the expertise to remedy the fault. When tossing a bug report, it is not always clear who should address the bug next ("who should the bug go to next?"). Another popular purpose for bug dumping is to engage developers with specialized knowledge in the problem report discussion. In any case, having a lot of tossing events slows down the process of correcting issues.

2.8 Manual Bug Triage and Its Problems

When a user (or a developer) discovers a bug, they report it to the project's bug tracking system (BTS), which keeps track of all bug reports. The project's principal developers, known as dispatchers, research the bug and determine its cause in collaboration with the co-developers [27]. The dispatchers then decide whether the bug has to be corrected and, if so, how important it is to solve it.

The dispatchers then look for developers who are available and qualified for the job. Bug triage necessitates the dispatchers' participation in all of these operations. Dispatchers must choose a suitable developer to be assigned to each bug.

New defects are manually triaged by an expert developer in traditional software development, i.e., a human triage. Manual bug triage is time consuming and inaccurate due to the enormous number of daily bugs and the lack of expertise of all bugs. According to Lohagaonkar *et al*. [1], a percentage of issues are allocated by mistake in manual bug triage in Eclipse, and the average

period between opening a defect and its initial triaging is 19.3 days. Existing research has presented an automatic bug triage solution, which uses text classification techniques to forecast developers for issue reports, in order to reduce the high expense of manual problem triage. A bug report is mapped to a document in this approach, and an associated developer is mapped to the document's label. The bug triage problem is then transformed into a text classification problem, which is solved automatically using advanced text classification techniques such as Naive Bayes. A human triage assigns new bugs based on the results of text classification, incorporating his or her experience.

2.9 Automatic Bug Triage

The manual bug triaging process is opulent in time and poor in accuracy [36]. There is a need to automatize the bug triage process. In order to automate the bug triage process, machine learning techniques are applied. As long as the bug has enough information, ML can predict exactly what will happen next [37]. It can examine the data to discover any new symptoms or problems, as well as compare the current condition to previous data in the hopes of predicting the actual bug and identifying the correct developer to whom the bug should be assigned.

The goal of machine learning (ML) is to create computer systems that can execute a task and improve over time by learning from data. In AI, machine learning is very significant [38]. Because of its high performance, it is widely utilized in a variety of industries (including pattern recognition, data mining, bug detection, and so on) [39]. ML algorithms extract the data's intrinsic relationships, which can then be presented in the form of rules or models for categorization and prediction. As depicted in Figure 2.1 [40], the ML process can be accomplished in three ways: Unsupervised, supervised, and semi-supervised learning are the three types of learning.

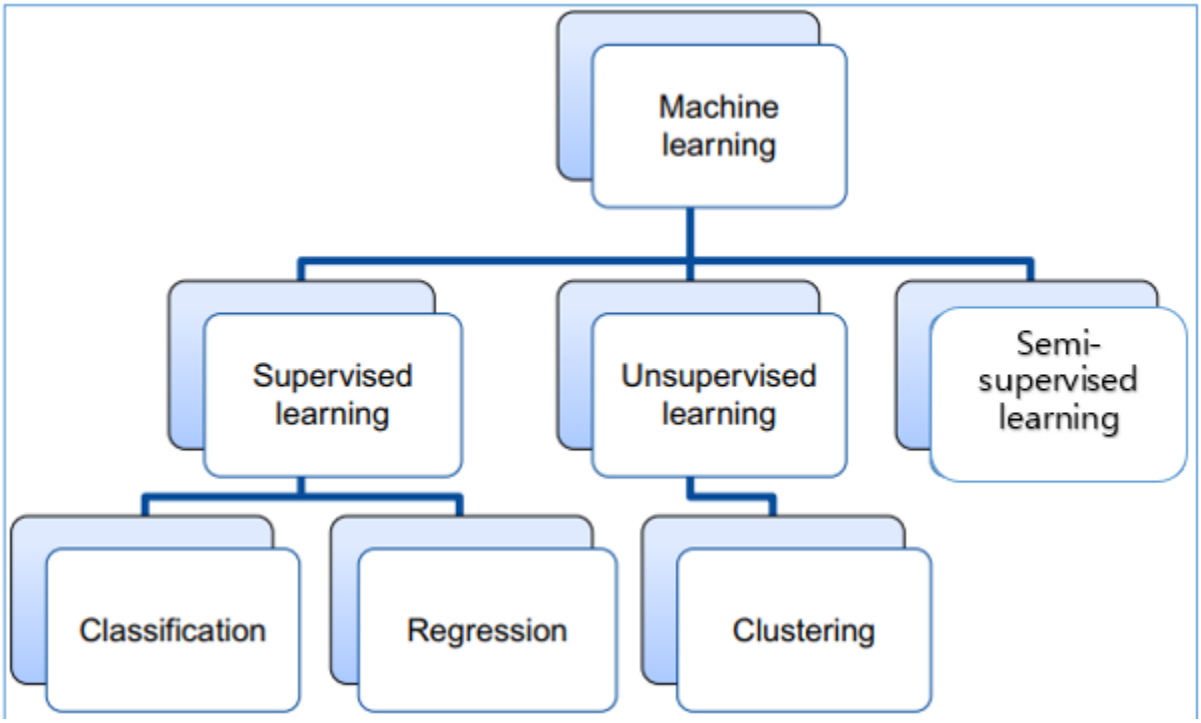


Figure 2.1 Types of Machine Learning

A. Supervised Learning

Supervised learning, often known as supervised machine learning, is an artificial intelligence and machine learning subcategory. It employs labeled datasets to train algorithms that accurately identify data or predict outcomes. As input data is fed into the model, the weights are adjusted until the model is well fitted, which happens during the cross validation phase [41].

A training set is used in supervised learning to teach models to produce the desired output. This training dataset contains both correct and incorrect outputs, allowing the model to improve over time. The loss function is used to assess the algorithm's correctness, and it is adjusted until the error is suitably minimized. The most common applications of supervised learning are classification and regression [42].

Classification is the process of accurately categorizing test data into distinct groups. It recognizes specific entities in the dataset and tries to come to some conclusions about how they should be labeled or defined [41, 42]. To explore the relationship between dependent and independent variables, regression is used. It is widely used to produce forecasts, such as for a company's sales

revenue. Popular regression algorithms include linear regression, logistical regression, and polynomial regression.

B. Unsupervised Learning

Unsupervised learning, unlike supervised learning, analyzes and clusters unlabeled datasets using machine learning algorithms [43]. In clustering problems, unsupervised learning is frequently used. Clustering is a technique for extracting inferences from datasets that lack a class label or goal values, or for exploratory data analysis to uncover hidden patterns. It creates clusters for data instances that are similar to one another and separates clusters for data instances that are highly dissimilar to one another. The K-Means algorithm, K-Nearest Neighbor algorithm, Hierarchical Clustering algorithm, and Graph Clustering algorithm are examples of unsupervised Machine Learning algorithms [55].

C. Semi-supervised Learning

Only a portion of the given input data is labeled in semi-supervised learning. It is a method that uses both supervised and unsupervised Machine Learning [44].

2.10 Classification of Algorithms

In order to automate the bug triage process, classification techniques are applied. Naïve Bayes, Decision tree (DT) and support vector machine (SVM) are the most well-known [45].

A. Naïve Bayes (NB)

Naive Bayes classifiers are typically used as a method for text categorization, which is a way of determining if a block of text belongs to one category or another. It is most widely used for document classification problem. This means checking if a document belongs to the certain category. This is done by checking the frequency the words are presented in the document and using that as a predictor. Naive Bayes is so called because it makes the strong assumption that features are independent of each other, given the label (bug, non-bug and the developer who resolved the bug) As in Figure 2.2 [45]. There are two event models that are commonly used: the multivariate Bernoulli event model and the multinomial event model [46].

MNB algorithm is a fast, easy-to-implement, almost modern text categorization algorithm [47]

The multinomial event model frequently referred to as multinomial naive Bayes or MNB. It is computationally very efficient and easy to implement.

For a given document a classes be denoted by C. and N be the size of our vocabulary. Then MNB assigns a test document ti to the class that has the highest probability$\Pr(c|t_i)$, which, using Bayes' rule, shows in equation 1 [47]:

$$\Pr(c|t_i) = \frac{\Pr(c)\Pr(ti|c}{\Pr(ti)}, c \in C \qquad (1)$$

The class prior Pr(c) can be estimated by dividing the number of documents belonging to class c by the total number of documents. $\Pr(t_i|c)$ is the probability of obtaining a document like ti in class c and is calculated using equation 2 [47]:

$$\Pr(t_i|c) = (\sum_n fni)! \prod_n \frac{pr(w_n|c)^{fni}}{fni!}, \qquad (2)$$

Where:

- f_{ni} is the count of word n in our test document t_i and
- $pr(w_n|c)$ the probability of word n given class c. the latter probability is estimated from the training document shown in equation 3 [47]:

$$\Pr(w_n|c) = \frac{1 + F_{nc}}{N + \sum_{x=1}^{N} F_{xc}}, \qquad (3)$$

Where:

- f_{xc} Is the count of word x in all the training documents belonging to class c, and
- The Laplace estimator is used to prime each word's count with one to avoid the zero-frequency problem. The normalization factor pr(t_i) in equation 1 can be computed using equation 4 [47]

$$\Pr(t_i) = \sum_{k=1}^{|c|} pr(k) pr(t_i|k), \quad (4)$$

Note that the computationally expensive terms $(\sum_n f_{ni})!$ And $\prod_n f_{ni}!$ equation 2 can be deleted without change in the results, because neither depends on the class c, and equation 2 can be calculated using equation 5 [47]:

$$\Pr(t_i|c) = \alpha \prod_n pr(w_n|c)^{f_{ni}}, \quad (5)$$

Where α is a constant that drops out because of the normalization step

The Bayes Theorem's principle of class conditional independence is used in the Naive Bayes classification approach [41]. This means that the presence of one feature in the probability of a given outcome has no bearing on the presence of another, and each predictor has an equal impact on the outcome. Multinomial Nave Bayes, Bernoulli Nave Bayes, and Gaussian Nave Bayes are the three forms of Nave Bayes classifiers [45].

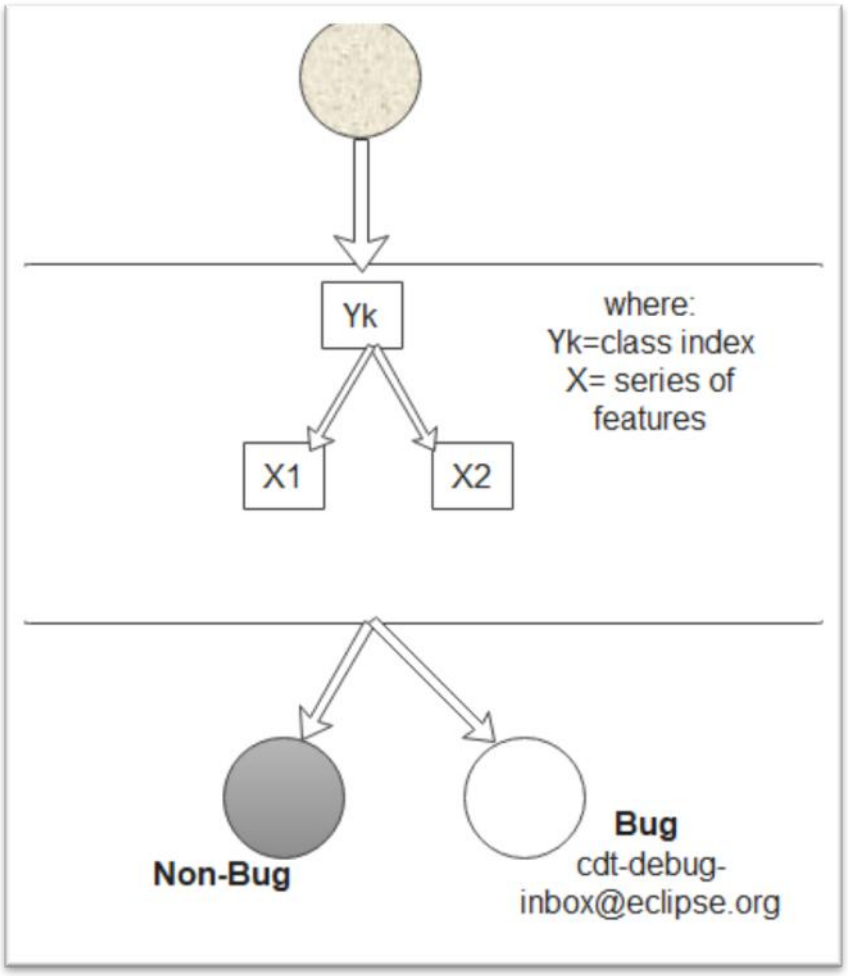


Figure 2.2 Naïve Bayes

B. Decision Tree

A supervised machine learning methodology for producing a decision tree from training data is decision tree learning (also known as a classification tree or a reduction tree) [40]. It is a predictive model in which observations about an item are mapped to conclusions about its target value. DT creates a tree that can categorize each record into multiple groups.

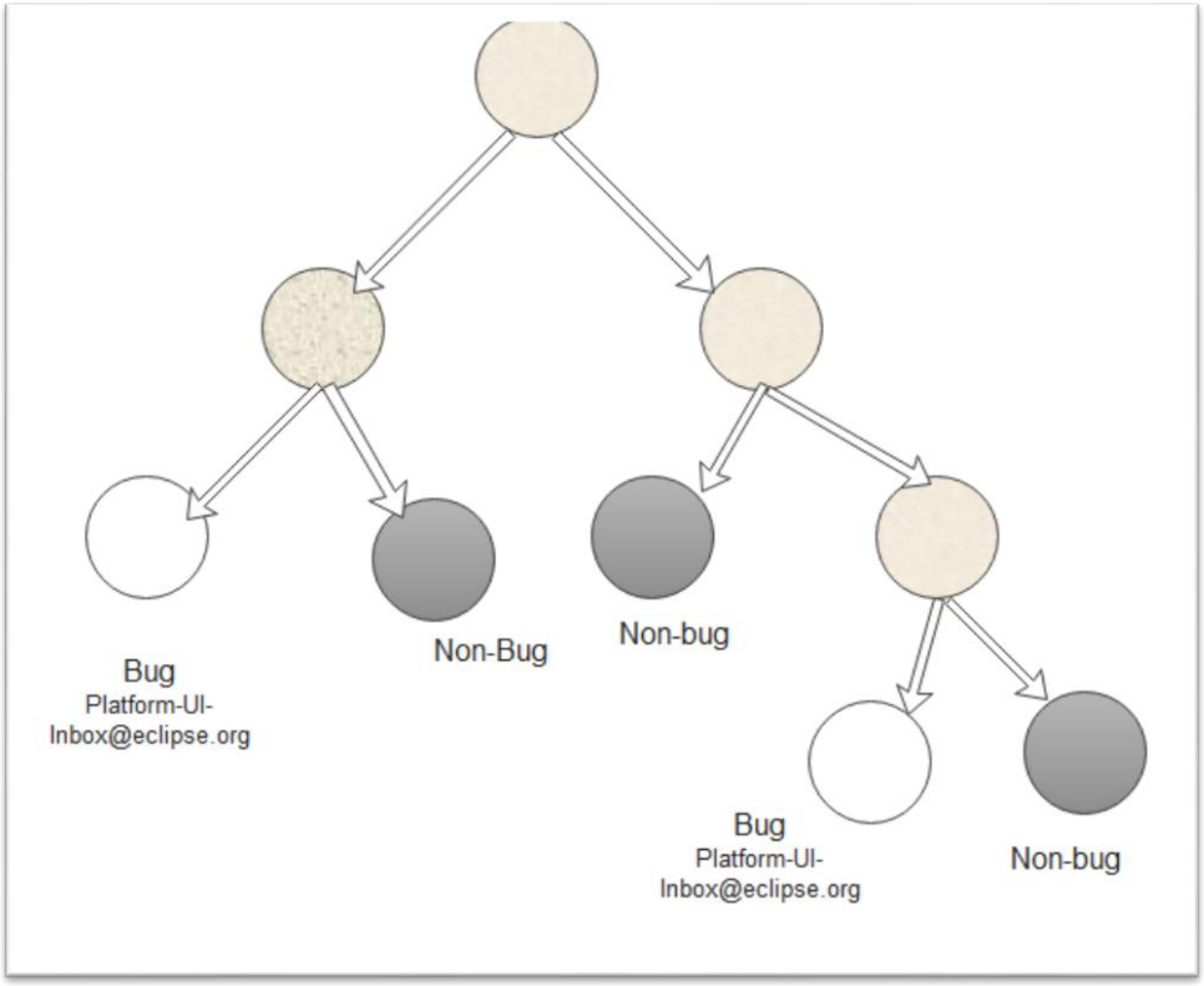


Figure 2.3 Decision Tree

Figure 2.3 [40] shows a tree structure in which leaves represent classifications (also known as labels), non-leaf nodes represent features, and branches indicate feature combinations that lead to classifications [48].

C. Support Vector Machine

A support vector machine is a popular algorithm that uses a hyperplane to divide two groups as shown in Figure 2.4 [42]. The goal is to identify an ideal hyperplane in the feature space that separates the feature points of the two classes by the greatest possible margin. The distance between two classes of data points is at its maximum on this hyperplane. The decision boundary is the line that divides the two types of data points on either side of the plane (bug vs. non-bug).

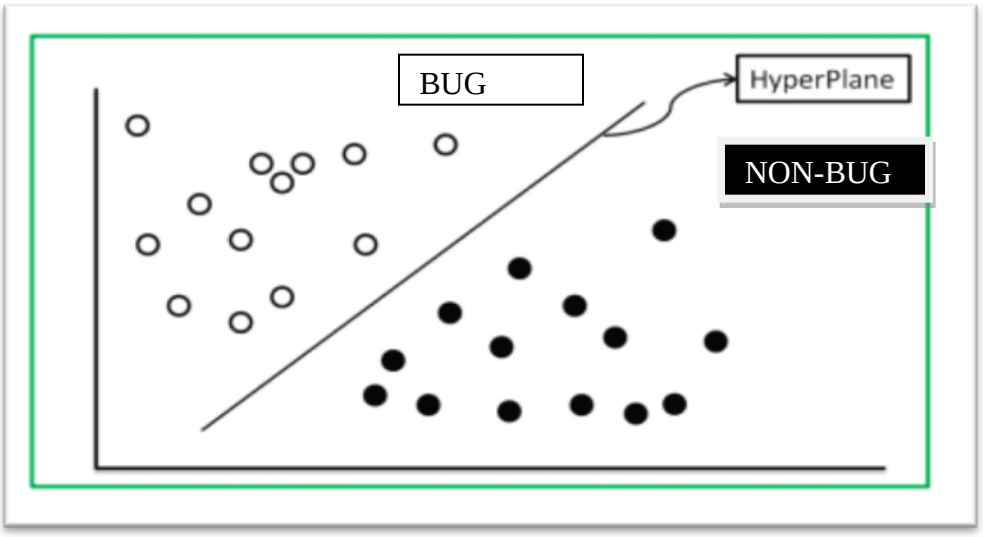


Figure 2.4 Support Vector Machine

2.11 Metrics

Highly accurate classifier puts as many instances as possible into the correct class and avoids classifying instances into the incorrect class. The metrics used for evaluating the performance of a classifier are discussed below.

2.11.1 Confusion Matrix

A confusion matrix is a 2D array that is often used to visualize the performance of a classification model (or "classifier") on a set of test data for which the correct labels are already known [49]. The confusion matrix as shown in Table 2.1 can help us to understand the evaluation metrics better. It has a total of 4 blocks: two rows and two columns which tell the values of true positives, true negatives, false positives, and false negatives [44]. The four metrics involved in a confusion matrix are discussed as follows:

- True positive (TP): represents the number of actual bug report samples correctly classified as actual bug instances.

- True negatives (TN): indicates the number of non-bug report samples correctly labeled as non-bug instances.

- False positives (FP): represents the number of non-bug report samples misclassified as actual bug instances.

- False negatives (FN): indicates the number of actual bug samples misclassified as non-bug instances

Table 2.1: Confusion Matrix

		Predicted class	
		Bug	Non-Bug
Actual Class	Bug	True positive (TP)	True negatives (TN):
	Non-Bug	False positives (FP):	False negatives (FN

2.11.2 Evaluation Metrics

To evaluate the performance of a classifier the most common and well-known evaluation metrics such as accuracy, precision and recall are used [49, 50].

- Accuracy: It gives the percentage of correctly classified instances by using Equation 6:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (6)$$

- Precision: It indicates that out of all the instances the classifier labeled as positive, what fraction were correct. It is calculated using Equation 7:

$$\text{Precision} = \frac{TP}{TP + FP} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (7)$$

- Recall (also called sensitivity or true positive rate): It indicates that out of all real positive instances, what fraction did the classifier labeled as positive. It is calculated using Equation 8:

$$\text{Recall} = \frac{TP}{TP + FN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

2.12 Summary

In this Chapter, we have introduced the basic information about bug reports including their history and lifecycle. At the same time the different bug report and sources of non-bugs is also explained. It has also been discussed the different methods of Software Bug Classification, impacts of Misclassification on Bug Triage, Bug Triaging Mechanism, the Manual and Automated Triaging with the problems of the manual triaging.

In addition, the different categories of machine learning algorithms namely supervised, unsupervised and semi supervised along with the best known classification algorithms has been discussed. Finally, classification algorithms evaluation metrics such as confusion matrix, accuracy, precision and recall are discussed.

3. Related Work

This Chapter provides an overview of related research in the field of bug triage. We will endeavor to investigate and pinpoint any gaps in past work.

3.1 Works Done on Bug Triaging

Russo *et al*. [51] used machine learning to evaluate sample data from the Google Chromium project to discover the optimal owner of a defect by analyzing bugs by title and description. They used natural language processing to look for patterns in the bug title and description in order to assign the correct developers to the bugs. They load two sets of data before implementing bug triage. The first is a triaged bug dataset that is used to train and test classifiers using cross validation. The second is an untriaged bug dataset that unsupervisedly learns from the target deep learning model.

The authour wanted to see how word2vec compared to LSTM and Naive Bayes. When compared to Naive Bayes, the results of utilizing word2vec with LSTM result in more accurate triaged problems. LSTM is a recurrent neural network suggested to regulate the gradient problem, according to NLP ideas. Its ability to detect commonplace connections. The goal of this study was to compare word2vec, LSTM, and Naive Bayes to find the best alternative for bug triage accuracy. Detecting the proper software bug from a bug repository was not included in the paper.

The subject of data reduction for bug triage was addressed by Lohagaonkar *et al*. [1] and Shaik *et al*. [35], who looked at how to minimize the scale and increase the quality of bug data. They use instance and feature selection to reduce data scale on both the bug and word dimensions at the same time. In data processing, techniques like instance selection and feature selection are commonly utilized.

Lohagaonkar *et al.* [1] used 600,000 bug reports from two large open source projects, Eclipse and Mozilla, to analyze the reduced bug data. The scale of a data set and the accuracy of bug triage were utilized as criteria. The findings suggest that data reduction can efficiently reduce data scale and enhance bug triage accuracy. Their study outlines data processing strategies for generating low-volume, high-quality bug data in software 25 development and maintenance.

Triage The system receives input in the form of a bug data set. All the specifics of software bugs are contained in the bug data collection. Each bug has a bug report as well as the name and

contact information for the developer who worked on it. The summary and description sections of the bug report are the most important. In the form of output, the suggested system provides projected results. They used two open source projects for the studies, Eclipse and Mozilla, and the results suggest that the data is decreased by using high quality bug data sets.

Shaik *et al*. [35] proposed a system that combines feature selection with instance selection to decrease the level of bug data sets by removing the redundant and non-informative bug reports as well as improve the data quality before assign to the developer. Here, LDA has been applied to get a good summary of the software history and to the commit log messages in order to see which topics are being worked on by developers at any given time.

Based on the identified component retrieved from the issue report's short description, Saagarikha et al. [12] developed a methodology to automatically propose developers who have the right competence for handling a bug report. They first looked at the effects of numerous machine learning aspects (classifiers and training histories), as well as a ranked list of developers, on bug assignment prediction accuracy, and then verified their methodology on Eclipse, which covered 2,868,000 bug reports with 253 components. Their findings showed that their strategies can predict bug assignment with an accuracy of up to 80.05 percent and drastically reduce bug assignment errors.

They used algorithms like Naive Bayes Text Classifier, Multinomial Naive Bayes, and Linear SVM to compare the prediction time for their dataset. They came to the conclusion that SVM has better accuracy and takes less time to train.

Similarly, Arudkar *et al*. [9] proposed using text categorization and machine learning techniques in bug triage to determine which developer should be assigned to the bug based on its description. They conduct a data reduction experiment for bug triage in the bug repositories of two large open source projects, Eclipse and Mozilla. Their proposed solution is based on Random Forest, which is a data processing methodology that reduces the data scale for providing high-quality bug data in software development and maintenance.

3.2 Works Done on Bug Classification

Herzig et al. [23] attempted to distinguish between Bug and non-bugs in software archives, which can have a significant impact on the accuracy of tools and studies that use such data. They claim that misclassifications have an impact on data quality and bug prediction, i.e. forecasting which applications are likely to be buggy in the future. The categorization of bug reports,

according to this study, is based on the observer's perspective. Approaches that use bug data sets should check whether the prediction model's perspective matches that of the bug creator. According to the findings, one should constantly be mindful that not all bugs are created equal. Many bugs are minor, but a few major ones, such as security or privacy flaws, can easily harm the product's reputation or even put the company's existence in jeopardy. Such repercussions cannot be assessed solely by machines. As a result, dealing with bug databases will always necessitate human effort an investment that, in the end, pays off. The goal of this project was to create a well-organized list of bug reports and features. Bug reports from Jira and Bugzilla problem reports are included in Herzig *et al* [23] datasets's. They discovered 33.8 percent of all bug reports were misclassified—that is, instead of referring to a code fix, they resulted in a new feature, an update to documentation, or an internal refactoring in a manual examination of more than 7,000 issue reports from the bug databases of five open-source projects. This erroneous classification leads to bias in bug prediction algorithms, causing confusion between bugs and features: On average, 39% of files labeled as faulty never included a flaw.

Misclassifications impair bug localization, according to Kochhar *et al*. [30]. They do so by analyzing issue reports that Herzig et al [23] have personally classified. For each issue report, use a bug localization approach to get a ranked list of candidate bugs. Then they assess whether the quality of ranked lists of bug reports is comparable to that of genuine bug reports. They used a vector space model (VSM)-based bug localization technique, which is a common information retrieval method. According to their results, further clean fing actions should be undertaken on issue reports before they are allocated.

According to Herzig *et al*. [23], more than 40% of issue reports are classified incorrectly. Kochhar *et al*. [30] study if these misclassifications have an impact on bug localization. They compared the effectiveness of a bug localization tool on issue reports that have been tagged as bugs by their reporters (Reported) vs issue reports that are true bug reports (Actual). When they compared the results for Reported and Actual, they discover that the mean average precision scores for HTTPClient, Jackrabbit, and Lucene-Java differ by -2.33 percent, 12.25 percent, and 6.98 percent, respectively. They discover that the differences are significant using the Mann-Whitney U test. They have also looked at the forms of misclassification that have the most influence on bug localization and come up with some tips for reducing that impact.

Suo and Zou [22] conducted research to see if the Herzig et al. [23] classified bug reports may be used in open-ihm. According to the researchers, Open-ihm is an open source software initiative that aids in the collection of data and information from low-income households. Its goal is to assist developers and academics in analyzing and recording family income data. The goal of constructing open-ihm is to bring information technology to every corner of the globe. Open-ihm is a data collection method that is both innovative and reliable, and it can be both open and quick. The major issue is that open-ihm software is required to assist both experts and non-experts in data analysis using specified models by incorporating the combined categorization criteria into this project, they will be placed in the same categories. The research, on the other hand, focused on concerns with bug and non-bug separation in software archives, and they described the effects of misclassification on bug localisation, data quality, and bug prediction. They did not, however, discuss the implications for bug triage.

Tran *et al*. [7] compared machine learning algorithms for determining the severity and priority of software bug reports, and then chose to use optimum decision trees, or random forest, as their strategy. Their method entails building numerous decision trees based on subsets of the current bug dataset and features, then picking the best decision trees to determine the severity and priority of new issues. They demonstrated the utility of random forest for bug report processing and conducted numerous tests using software bug datasets from open source bug tracking services. They demonstrated the utility of random forest for bug report processing and conducted numerous tests using software bug datasets from open source bug tracking services. They built decision trees and random forests using a bug dataset of 300,000 bug reports on the Windows platform (Win platform) and 300,000 bug reports on other platforms (All platform).

They compared the results of machine learning techniques for analyzing software bug reports. Their method aims to automatically determine the severity and priority of a bug report. Bug reports share some of the same characteristics as log events, such as severity and priority. The random forest's performance and accuracy are the subject of the approach's evaluation. The time consumption and accuracy of random forest for bug datasets were measured by the authors. For diverse datasets, the experimental results show that random forest beats decision tree by around 10%. Random forest, on the other hand, takes longer to process than decision tree. Their technique can currently be used to detect and forecast failures in big, complicated

communication networks and distributed systems. However, their investigation did not include detecting the actuality of the bug.

Nikita *et al*. [6] classified bugs in order to establish the bug's class and, after applying the classification, to allocate the bug to the correct developer for resolution. The authors used a mixture of two classification algorithms, Naive Bayes (NB) and K closest neighbor (KNN), as well as instance and feature selection to decrease the bug dataset. The bug detail was required for triaging the bug, which is known as the bug repository. However, they did not discussed detecting the actuality of the reported bug before bug allocation

3.3 Summary

This Chapter presented related works on bug triaging techniques along with their respective advantages and limitations. There have been several attempts to detect actual bug reports from the reported bug and assigning to the developer. But there remain certain limitations that are uncovered by these research works. The bug triaging techniques we reviewed have their own major shortcoming which is detection and assignment is done separately mean not combine first detection and then assignment those which are actual bugs. As we can observe on the limitations, lots of works should be done on providing more effective bug triaging technique. So, this study is intended to fill the gaps observed on previous works and design improved automatic bug triage model using machine learning techniques.

4. Bug Trkage Model

4.1 Introduction

The goal of this Chapter is to provide a clear picture of the proposed triaging bug model. As we discussed in Chapters 2 and 3, bug maintenance takes a long time. In the following Chapter, we'll go over the specifics of the bug detection and bug assignment approaches, as well as the architectural framework components in the proposed solution. The recommended solution architecture framework for bug triage model is described and presented in Section 4.2. The Training component of the bug triaging model is presented in Section 4.3, and the Chapter is summarized in Section 4.4 based on our proposed architectural framework for bug triage.

4.2 Overview of the Architecture

Preprocessor, feature extractor, dataset constructor, classifier, bug detector, and developer assigner are components of the proposed system architecture.

The data is tokenized, the stop-word is deleted, and is stemmed at the data preprocessing stage. All accessible features were extracted using the feature extractor. The dataset constructor is used to partition the dataset into training and testing sets after converting the JSON file into sets. The bug detector is then in charge of categorizing the bug report into one of two categories: bug or non-bug. Finally, the developer assigner is used to assign bugs to the appropriate developer.

The proposed architecture is depicted in Figure 4.1. All of the components of the architecture are discussed in depth in Section 4.3.

.

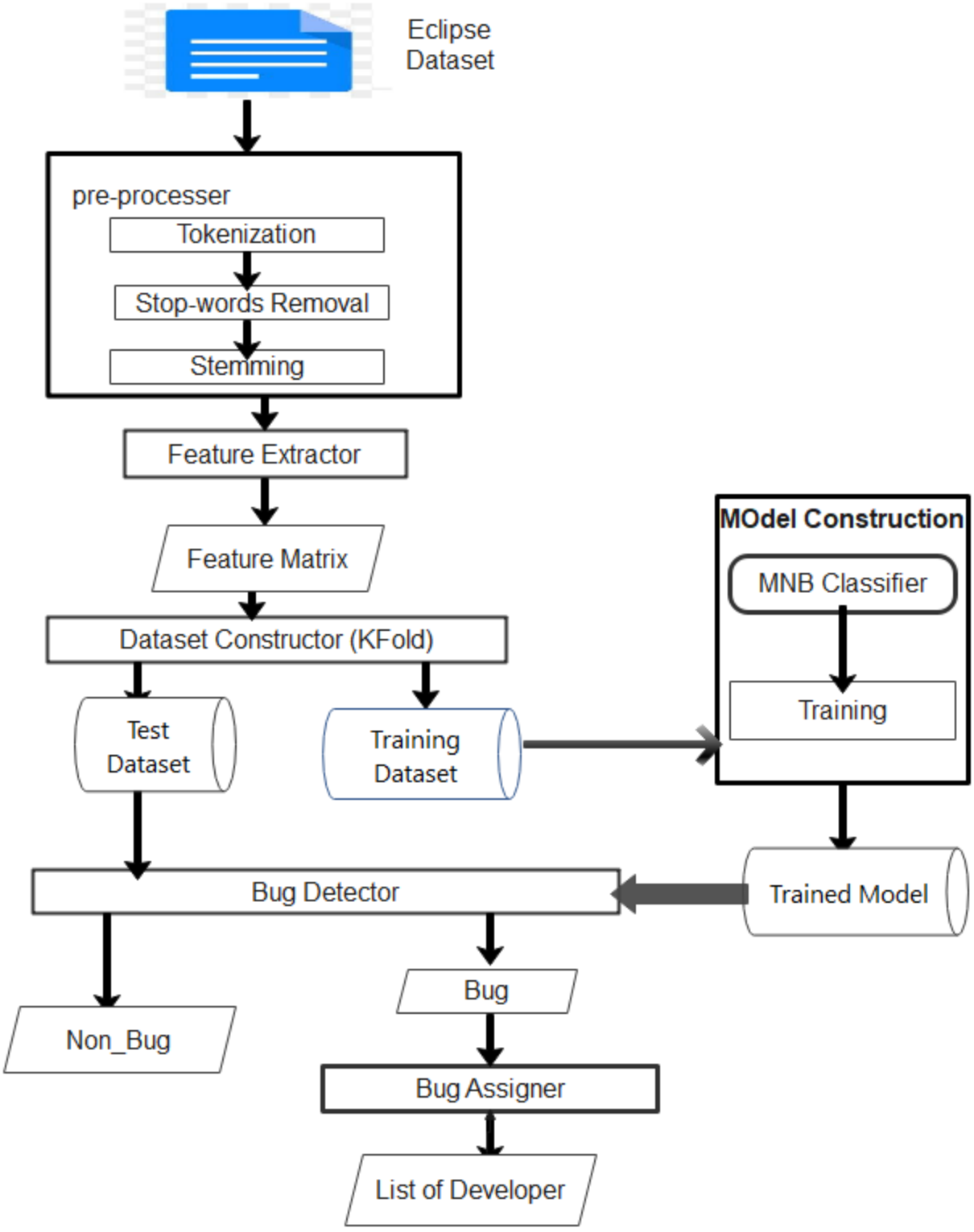


Figure 4.1. The proposed bug triage model Architecture

4.3 Description of Components of the Architecture

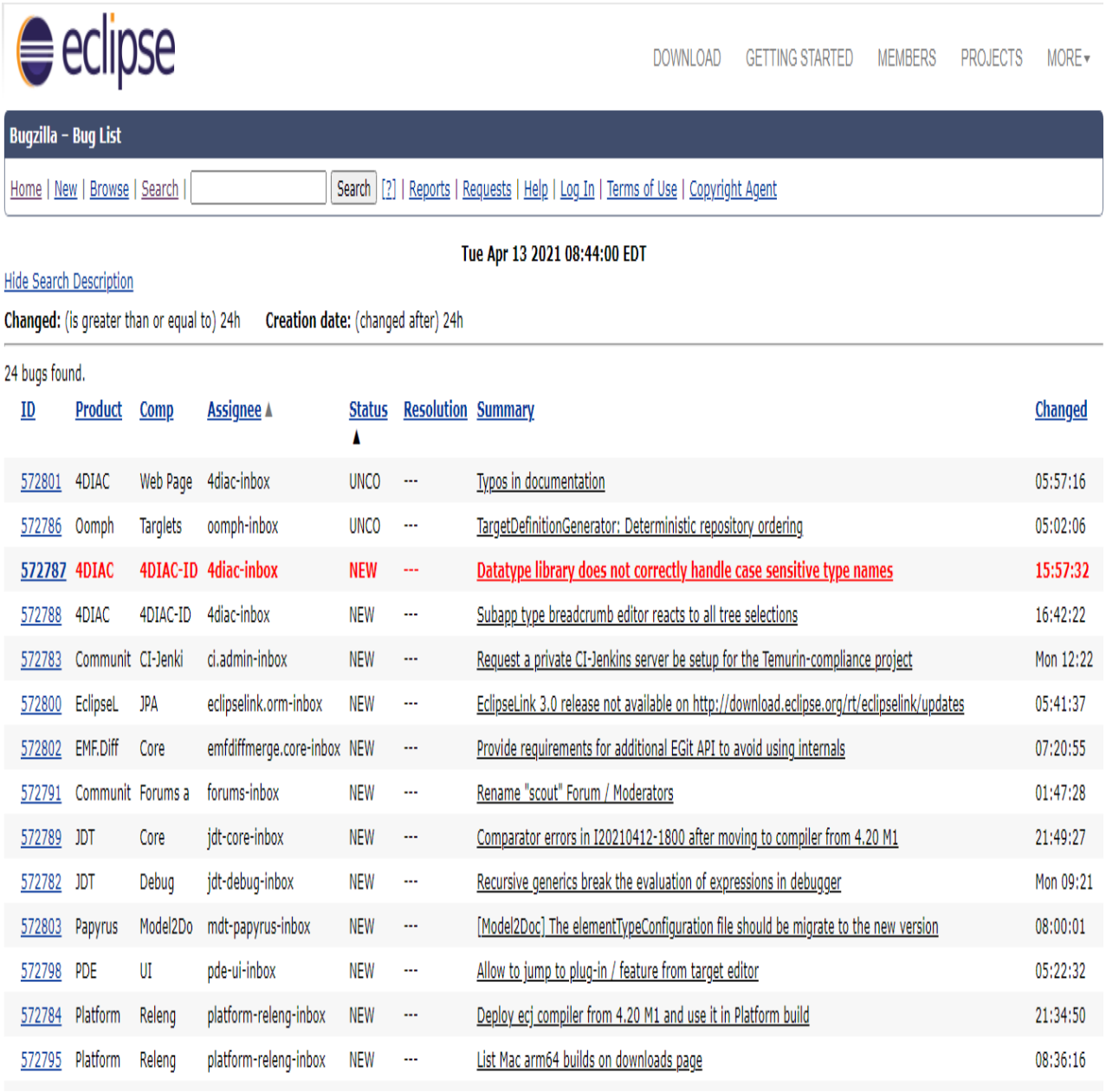
The components of the proposed solution improve the bug triage via machine learning techniques.

4.3.1 Bug Report

A bug report, generally a natural language text, submitted by the user is stored in the XML format by the bug tracker tool. Figure 4.2 illustrates the bug report structure of Eclips project on bugzilla bug repository. Information contained in a bug report [12]:

- Severity: The severity denotes how soon the bug should be fixed.
- Product: The particular software application the bug is related to.
- Component: The relevant subsystem of the product for the reported bug.
- Assigned to: The identifier of the developer to whom the bug was assigned to.
- Short_desc: Contains a natural language text embedded by the user.
- Bug status: The status of the bug at every update. NEW, ASSIGNED,RESOLVED, VERIFIED, REOPENED.
- Resolution: Tagging the bug report for maintenance. FIXED, REMIND, INVALID, WORKSFORME.

From this information the actuality of the bug is detected from severity field and the dependencies between the components, developers, and reassignment can be formed.



eclipse

DOWNLOAD GETTING STARTED MEMBERS PROJECTS MORE

Bugzilla – Bug List

Home | New | Browse | Search | Search [?] | Reports | Requests | Help | Log In | Terms of Use | Copyright Agent

Tue Apr 13 2021 08:44:00 EDT

Hide Search Description

Changed: (is greater than or equal to) 24h **Creation date:** (changed after) 24h

24 bugs found.

ID	Product	Comp	Assignee	Status	Resolution	Summary	Changed
572801	4DIAC	Web Page	4diac-inbox	UNCO	---	Typos in documentation	05:57:16
572786	Oomph	Targlets	oomph-inbox	UNCO	---	TargetDefinitionGenerator: Deterministic repository ordering	05:02:06
572787	4DIAC	4DIAC-ID	4diac-inbox	NEW	---	Datatype library does not correctly handle case sensitive type names	15:57:32
572788	4DIAC	4DIAC-ID	4diac-inbox	NEW	---	Subapp type breadcrumb editor reacts to all tree selections	16:42:22
572783	Communit	CI-Jenki	ci.admin-inbox	NEW	---	Request a private CI-Jenkins server be setup for the Temurin-compliance project	Mon 12:22
572800	EclipseL	JPA	eclipselink.orm-inbox	NEW	---	EclipseLink 3.0 release not available on http://download.eclipse.org/rt/eclipselink/updates	05:41:37
572802	EMF.Diff	Core	emfdiffmerge.core-inbox	NEW	---	Provide requirements for additional EGit API to avoid using internals	07:20:55
572791	Communit	Forums a	forums-inbox	NEW	---	Rename "scout" Forum / Moderators	01:47:28
572789	JDT	Core	jdt-core-inbox	NEW	---	Comparator errors in I20210412-1800 after moving to compiler from 4.20 M1	21:49:27
572782	JDT	Debug	jdt-debug-inbox	NEW	---	Recursive generics break the evaluation of expressions in debugger	Mon 09:21
572803	Papyrus	Model2Do	mdt-papyrus-inbox	NEW	---	[Model2Doc] The elementTypeConfiguration file should be migrate to the new version	08:00:01
572798	PDE	UI	pde-ui-inbox	NEW	---	Allow to jump to plug-in / feature from target editor	05:22:32
572784	Platform	Releng	platform-releng-inbox	NEW	---	Deploy ecj compiler from 4.20 M1 and use it in Platform build	21:34:50
572795	Platform	Releng	platform-releng-inbox	NEW	---	List Mac arm64 builds on downloads page	08:36:16

Figure 4.2 Example of bugzilla bug report

The assigned_to.xml, product.xml, component.xml, severity.xml, and short_description.xml of each bug in the data set are processed using an XSLT parser to produce a unified text file from the bug report field. Each bug's report ID is obtained from the assigned-to.xml file, and keywords like "major," "critical," "normal," "minor," "trivial," and "blocker" are obtained from severity.xml to categorize the issue as a "bug," whereas "Enhancement" is a non-bug.

The “Severity” field in Bugzilla is a field that should be used to identify bugs from non-bugs. The “Enhancement” value suggests a non-bug categorization, while the other suggests a bug classification. Finally, the “when” property of each bug update is extracted and matched with the relevant entries in short-desc.xml, product.xml, component.xml, and severity.xml, and exported to a text file in a text processing-friendly way.

When an issue is assigned to a developer for the first time, and s/he is unable to fix it, the bug is allocated (tossed) to another developer. As a result, a bug gets passed from one developer to the next until one is able to solve it. Goal-oriented tossing graphs were proposed based on these tossing paths. Suppose we have a tossing path, A → B → C → D. The bug is fixed by D, the fixer. Tossing graphs are weighted directed graphs such that each node represents a developer, and each directed edge from A to D represents the fact that bug is assigned to developer A was tossed and eventually fixed by developer D. The weight of the edge between two developers is the probability of a toss between them, based on bug tossing history. We denote a tossing event from developer A to Dj as A =⇒ Dj [12]. As depicted in Table 4.1 the list of Developers should be chosen in such a way that the probability of the bug getting reassigned must be minimum. If the bug is assigned to developer A, and developer A is unable to fix, then the bug will be tossed to developer D as it has highest probability among others which is 75%.

Table 4.1: The Goal Oriented Tossing Path

Tossing paths							
A→B→C→D A→E→D→C A→B→E→D C→E→A→D B→E→D→F							
Developer who tossed the bug	Total tosses	Developer who fixed the bug					
		C		D		F	
		#	pr	#	pr	#	Pr
A	4	1	0.25	3	0.75	0	0.00
B	3	0	0.5	2	0.67	1	0.33
C	2			2	1.0	0	0.00
D	2	1	0.5			1	0.5
E	4	1	0.25	2	0.5	1	0.25

4.3.2 Preprocessor

The preprocessor is used to make the training/testing process easier by scaling and converting the entire dataset suitably [54]. When a user discovers a bug s/he reports it to the software's bug tracker. Because the user's issue description is a natural language text, machine learning techniques such as natural language processing are utilized to extract important keywords from the bug report that will provide information about the bug the user has encountered.

The most crucial phase in data mining is data pre-processing. The data collected from bug repositories is unprocessed and cannot be utilized to train the triaging algorithm directly. To make the data relevant for training, it is initially pre-processed.

The goal of the preprocessing stage is to remove any extraneous words from the bug report's content. Words that aren't useful for data analysis are deleted because their existence can impair learning performance. The size of the feature set is reduced by deleting extraneous words,

making it easier to understand and conduct data analysis. During the data preprocessing step, we conduct three key tasks: tokenization, stop-word removal, and stemming.

A. Tokenization

Tokenization is the process of breaking down a phrase, sentence, paragraph, or even an entire text document into smaller components like individual words or phrases. Tokens are the names given to each of these smaller units. This is significant because the text's meaning may be easily deduced by examining the words in the text.

B. Stop-Word Removal

The second step in data pre-processing is stop word removal. Stop words are words that are typically filtered out before being processed by a machine. These are the most common words in any language (articles, prepositions, pronouns, conjunctions, and so on), and they don't add anything to the text. Stop words in English include "the," "a," "an," "so," and "what." We removed the low-level information from our text by deleting these words, allowing us to focus more on the crucial information. In other words, we can say that removing such phrases has no negative impact on the model we are training for our task.

Because there are fewer tokens involved in the training, removing stop words reduces the dataset size and hence reduces training time. We choose NLTK from among the several libraries.

C. Stemming

The process of reducing a word to its word stem, which affixes to suffixes and prefixes or to the roots of words known as a lemma, is known as stemming. After stemming, words like "likes," "liked," "likely," and "liking" will be reduced to "like."

4.3.3 Feature Extractor

After performing pre-processing processes, the amount of the feature set that can be generated is still large, and thus not suitable for machine learning algorithms. The extraction of the most discriminative features (terms/words) from a list of features using proper feature extraction methods is a critical step in text mining.

We employed the term frequency inverse document frequency (TF-IDF) feature extraction method to create our model. The TF-IDF (Term Frequency Inverse Document Frequency) is a

numerical statistic that measures how essential a word is to a document in a collection or corpus. This is to prevent the use of frequently occurring words throughout the document (bug reports) from lowering bug identification and bug assignment efficiency. We select “enhancement” field for non-bug class and others for bug class from severity.xml file. From “assigned-to.xml” file we select developers as a features class to perform the automatic classification

The bug reports are displayed as a feature matrix, with each row representing a bug report with "n" features (terms). The Term Frequency Inverse Document Frequency (TF-IDF) technique is used to weight each term. First term frequency is calculated using equation 9 [12]. And then invers document frequency is obtained using equation 10 [12]. By multiplying term frequency by inverse document frequency, term frequency is calculated using equation 11 [12].

$$\mathrm{tf}(t, doc) = 0.5 + \frac{0.5 \times f(t, doc)}{\max\{ f(w, doc) : w \in doc\}} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (9)$$

$$\mathrm{idf}(t, DOC) = \log \frac{N}{1 + |\{doc \in DOC : t \in doc\}|} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (10)$$

$$\mathrm{tf-idf}(t, doc, DOC) = t f(t, doc) \times \mathrm{id}\ f(t, DOC) \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (11)$$

As shown in algorithm 4.1, in scikit-learn, the TF-IDF algorithm is implemented using TFIDFTransformer. This transformer needs the count matrix which it we transform later. Hence, we use CountVectorizer first.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
count_vectorizer = CountVectorizer()
X_train_counts= count_vectorizer.fit_transform(X_train)
tfidf_transformer=TfidfTransformer()
tfidf_transformer.fit_transform(X_train_counts)
```

Algorithm 4.1 TF_IDF

4.3.4 Dataset Constructor

The dataset constructor has two major tasks in this step. The constructor will first allow us to create a dataset in JSON format. Second, it will aid in the classification of bug report data into training and testing sets. We can't use the bug reports directly in our model because it is saved in an XML file. As a result, the output from the preprocessing stage will be stemmed data in JSON format. The constructor will then use Pandas to read this JSON file and feed it to NumPy arrays. Pandas is a data manipulation and analysis tool built on top of the Python programming language. NumPy is a Python library that can process numbers, texts, and records as arrays. Finally, the dataset constructor splits the dataset after retrieving the array values from NumPy:

- Splitting the Dataset

The constructor's main task is to separate the JSON dataset into two sets: training and testing. . In machine learning, we design a model to predict the test data. As a result, the training set is used to learn the model and the testing set is used to evaluate it.

A. Training Set

The model will be trained using the training dataset, which consists of a collection of bug reports. 80 percent of the overall dataset is in our training subset.

B. Testing Set

The testing dataset is utilized to evaluate our model's classification performance. It provides a fair assessment of the final model learned from the training data. The testing subset represents 20% of the overall dataset. The dataset constructor uses the Sklearn "KFold" function [55] to divide the dataset into training and testing sets. Sklearn is a Python module with a number of data processing features, including the ability to partition a dataset into random train and test groups.

4.3.5 Classifier

In machine learning (ML), the focus is on building computer systems which should perform a task and improve themselves in this task by learning from data. We discussed different supervised based Machine Learning classification techniques earlier in Chapter 2 under Subsection 2.10, which can be used to classify a given bug report as bug or non-bug and predict

the concerned developer who fix for actual bug reports. Among the techniques, we are going to use the classification technique named Naive Bayes to build our model.

We used Naive Bayes classifier uses the following steps to find probability of an event in the training data:

- Step 1: Calculate the prior probability for given class labels

  Class for bug detection is {bug, non-bug} and class for bug assignment is { Boris_Bokowski@ca.ibm.com,jdt-ui-inbox@eclipse.org.....} which is list of developers

- Step 2: Find Likelihood probability with each attribute for each class

- Step 3: Put these value in Bayes Formula and calculate posterior probability.

- Step 4: See which class has a higher probability, given the input belongs to the higher probability class.

4.3.6 Bug Detector

To find out the actuality of software bug we used machine learning bug detector. The bug detector tackles the problem of classifying issues into two classes: bugs and non-bugs, with the goal of building automatic classification systems. Such automatic classification systems, or classifiers, can be used for checking the actuality of software bugs before assign to developers

Input: T: Training Corpus (List of bug reports)

C: List of Discriminating terms. Terms/fields that machine learning techniques use to discern bugs from other issues

Output: R: The bug report with the highest probability to be really bug

```
foreach Dicrterm C in D do
    Rc ←count bug report contain Discterm C from T
    sim (Rc with T) ← similarity calculation
    if (sim (Rc with T)>0)
        foreach bug report R in D do
            bc ←  count actual bug reports b from R
```

```
                prior(b)  |Rc|/|N|
                wordsc  ← collect all words from all bug reports
in actual bug b
          end
     end
     return b with highest probability to be actual bug
```

Algorithm 4.2: Bug Detector

4.3.7 Developer Assigner

We use machine learning developer assigner component for developer assignment. Once a bug report has been assigned, developers can reassign the bug to other developers; we call this process bug tossing. For this research work, we studied the assignment and tossing activities for 315,228 bug reports from Eclipse.

Input: Actual bug report detected by bug detector classifier

Output: The component in which the bug may potentially be, and the developer or list of developers to whom it can be assigned to. Based on the defective component and tossing history of the developers, a list of developers will be informed of the bug to solve. The list of developers should be chosen in such a way that the probability of the bug getting reassigned must be minimum. After fixing the bug, the bug report is annotated/labelled with the developer and the component related to the bug.

- **Algorithm for Automatic Bug Triaging**

Input: T: Training Corpus (List of bug reports)

C: List of Discriminating terms. Terms/fields that machine learning techniques use to discern bugs from other issues

Output: B: Bug report which contains terms in C

d: The developer with the highest probability to whom the bug maybe assigned to.

```
foreach Dicrterm C in D do
     Dc ←count bug report contain Discterm C from T
     sim (Dc with T) ← similarity calculation
     if (sim (Dc with T)>0)
```

```
        foreach bug report C in D do
            b  ← all actuall bug reports in T
            P(dj)  |b|/|total no: reports|
            foreach developer d in D do
                reports j  ← all bug reports in developer dj
                P(dj)  |reports j|/|total no: reports|

                end
            end
        end
return d with highest probability
```

Algorithm 4.3: Automatic bug triaging model

4.5 Folding

For our model we used fold based training and validation approach. In folding based training and validation approach, also known as cross validation [6], the algorithm first collects all bug reports to be used for TDS (Training Data Set) sorts them in chronological order(based on the update time of the bug) and then divides them into n folds. In the first run, fold 1 is used to train the classifier and then to predict the VDS (Verified Data Set). In the second run, fold 2 bug reports are added to TDS. In general, after validating the VDS from fold n, VDS is added to the TDS for validating fold n+1. For example, we chose n=10 and carried out 9 iterations of the validation process using incremental learning.

4.6 Summary

In this Chapter the Design of the Bug Triage model was discussed by explaining the data as well as methods used. The proposed model has sex components. This are preprocessor, feature extractor, dataset constructor, classifier, bug detector, and developer. The bug report dataset was prepared in a Tokenize, stop ward removal and Stemmed form. TF_IDF method is used for Feature Selection. The main intention is to get well-structured data that is easy to understand for machine learning. These features are used for training of the detection and classification

algorithm which is then used for triaging of bug reports. The classification algorithm used in proposed system is multinomial Naïve Bayes.

All the components in the proposed model are direct participants in the triaging process, i.e., the output of one component will be the input of the next component. In the next Chapter, we will describe the prototype detail of the proposed bug triage model.

5. Implementation and Evaluation

5.1 Introduction

This Chapter presents the development environment, programming language and tools and techniques used for prototyping of the proposed Bug Triage Model. The set of experiments conducted to validate the proposed model is evaluated by using various evaluation metrics such as accuracy, precision and recall. Section 5.2 presents the tools and programing language used in the experiment. Experimental setup is presented in Section 5.3. Following that, the dataset used is discussed in Section 5.4. Section 5.5 is about implementation process. Accuracy of the algorithm and result and discussions is presented in Sections 5.6 and 5.7 respectively. Finally, Section 5.8 is all about summarization of this chapter.

5.2 Tools and programing Language Used

Among the different programming languages, we have selected the Python programming language for developing and implementing the prototype model. Python accompanies many libraries for developing complex scientific, numeric applications and is designed with features to facilitate data analysis and visualization [52].

We used Pycharm editor which is the most popular IDE used for python scripting language. It offers some of the best features to its users and developers in code completion, inspection and advanced debugging. We also used scikit_learn, Pandas, Numpy and matplotlib libraries to build and train our model.

We used Pandas which is an open-source library that lets us easily use data structures and data analysis tools for the Python programming language. Pandas is structured around DataFrame objects. All of our data comes into one big DataFrame where we can select out some samples or other data manipulation if wanted. It can help us to load, reshape, analyze, process and adjust data. It allow as to import JSON file format and manipulate numerical tables

Numpy library was used to compute the numerical operations of the model. We also used matplotlib which is scientific plotting library required to visualize data. We used plot histograms to analyze the data.

Finally we used scikit_learn to build machine learning models. sklearn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools

for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction [53].

The great thing about Numpy, Pandas and Scikit Learn is that they all work together. A default thing to do is to load/clean/manipulate our data using Pandas. Translate our Pandas DataFrame into a Numpy array and fed it to Scikit Learn function(s).

5.3 Experimental Setup

Machine learning requires a high performance platform environment with good computational power; hence we have selected the Ubuntu Linux operating system, which has a high performance capability, for our development environment. For our purpose, our system fulfills above the minimum requirements GPUs, CPUs, RAM as well as storage capacity. The system consists of computer capacity 8 GB memory, 1 terabyte disk capacity and processor Intel r corei5 CPU 2.30GHz.

5.4 Datasets

We must train and test our model utilizing bug complaint data from a range of dataset sources to ensure that it can detect bugs and non-bugs and allocate them to developers. As a result, we acquired our dataset/bug report in JSON format/ files from the Eclipse project that Saagarikha S et al. [12] used. Our model used machine learning techniques that train themselves from their inputs dataset and performs other related tasks without the need to learn everything. It improve its performance by learning from their sample input data.

Our dataset is a collection of bug reports containing necessary information about the severity, components, developers and re-assignments.

To use these datasets for classification purposes, we used 315,228 bug reports of eclipse's project using bugzilla bug tracking tool. From the total, we used 80% (252,183 bug records) for training and 20% (63,045 records) for testing.

Using XSLT parser we convert the xml to json format. First the dataset in XML format is what we used but it has only around 10000 reports. To increase the effectiveness of our model, we used dataset which is in JSON format it has around 315,000 reports in a well-structured manner. Training data set in JSON format compares the report-id and the update("when") of each report-id in 4 the respective files namely assigned-to, severity, component, and short-desc and merges

the ”what” content present in short desc(to get the bug report),severity (to detect the actuality of the bug), component(to obtain the component) and assigned to(the developer) to a single text file. This text file is pre-processed.

The pre-processed file is converted to a feature-vector pair where the feature is the component and the vector being the severity for bug detection. The classifier learns from this feature-vector pair and predicts the actuality of bug reports.

Another feature-vector pair where the feature is the bug-report and the component it is present and the vector being the developer. The classifier learns from this feature-vector pair and predicts the accurate developer for those actual bug reports. The last feature-vector pair (component and developer) learned by the classifier is used for tossing graphs. The probability of developer solving the bug in particular component and his tossing to another developer are combined and the next probable developer who can fix the bug is determined

5.5 Implementation process

Eclipse BTS issues are downloaded as XML files. To apply machine learning techniques, we need to preprocess the XML texts of the issue (bug report). When extracting text from BTS issues, we consider and distinguish features, namely: assigned to, product, component, severity and short description of each bug. we used short description to collect keywords from the bug description in the bug report.

Using feature selection algorithm called TFIDF, we select a subset of the available features to perform the automatic classification. Finally, each indexed feature is augmented with its class [0; 1], i.e.,[non bug; bug] for bug detection and [149,105,3,9] i.e.. [Platform-UI-Inbox@eclipse.org, Boris_Bokowski@ca.ibm.com,jdt-ui-inbox@eclipse.org, daniel.megert@eclipse.org....], since bug tracking systems including eclipse use an email address as developer identification. Then this column is used by the machine learning techniques during the training phase.

5.6 Performance Evaluation

5.6.1 Evaluation Result

In this subsection, the evaluation results are presented and the performance of our model is assessed using the evaluation metrics such as accuracy, precision and recall in order to validate its performance. Getting very high accuracy somehow can be achieved by thoroughly selecting

the sample size of the dataset. So, if we use accuracy alone to evaluate the performance of our model, the result can be subjective and it cannot give us the desired performance result. For this reason, precision and recall performance evaluation measures which are not dependent on the sample size of the testing and training sets are also considered. Precision and recall are two very vital model evaluation metrics. They are to some extent inversely proportional to each other. Precision refers to the percentage of results which are relevant, whereas recall refers to the percentage of total relevant results correctly classified by our model.

In our experiments, we used a feature selection algorithm to select a subset of the available features with which to perform the automatic classification. The algorithm correctly assigns just under 95% of the bugs, when 80% of the document corpus is used as training and 20% as the test set. The accuracy slowly declines to 85% as the test sets size is increased to 50%.
Finally, in particular, we use a 10-fold cross validation, *i.e.*, dividing each set of issues in 10 sets, training the classifier on 9 of them, classifying the remaining set to test accuracy, and repeating the process to classify all the 10 sets. With 10 fold cross validation an accuracy of nearly 94% is achieved and using MNB classifier, the proposed model has an average highest detection accuracy, precision and recall rates respectively at 95.67%, 95.14% and 95.16%. Overall, after considering the results obtained, it can be concluded that the proposed system has a great performance result on MNB classifier.

5.6.2 Binary Classification

We have also evaluated our classification technique using two-dimensional (2x2) confusion matrix. Table 5.1 illustrates the confusion matrix result of our MNB classifier for the two class classification. It shows the TP, TN, FP and FN results of our classifier after tested using bug report test sets. In the confusion matrix, class Bug represent positive instances whereas NON-BUG represents negative instances.

Table 5.1: Confusion Matrix for our Classifier using the Test Sets

		Predicted class	
		NON-BUG	BUG
Actual Class	NON-BUG	TN 1,405	PF 24
	BUG	FN 66	TP 61,550

The confusion matrix in Table 5.1 shows the actual and predicted classes of the test sets. Among the total (63045) positive and negative instances in the test set, the model correctly classified 61,550 instances as positive (TP) and 66 samples as negative (TN). Accordingly, the model gained 24 FP and 1,405 FN results which is 95.65%.

5.7 Discussions

As Figures 5.1 and 5.2 shows, our model will assign those bug reports to the developer if and only if the reported bug is actually a bug. If the reported bug is maintenance bug (actual bug) the model predicted the responsible developer and also the next developer who will fix it using the developer's networks and team structures.

The proposed solution has been analyzed through experimentation to evaluate its performance in detecting actuality of the bug and assigning to appropriate developer. The experimental results show that our proposed model can successfully distinguish maintainable bug reports from non-maintainable ones with 95% detection accuracy rate.

We compared the prediction time for our dataset using various algorithms such as Naive Bayes Text Classifier, Multinomial Naive Bayes and Linear SVM. We arrived at a conclusion that Multinomial Naive Bayes provides higher accuracy.

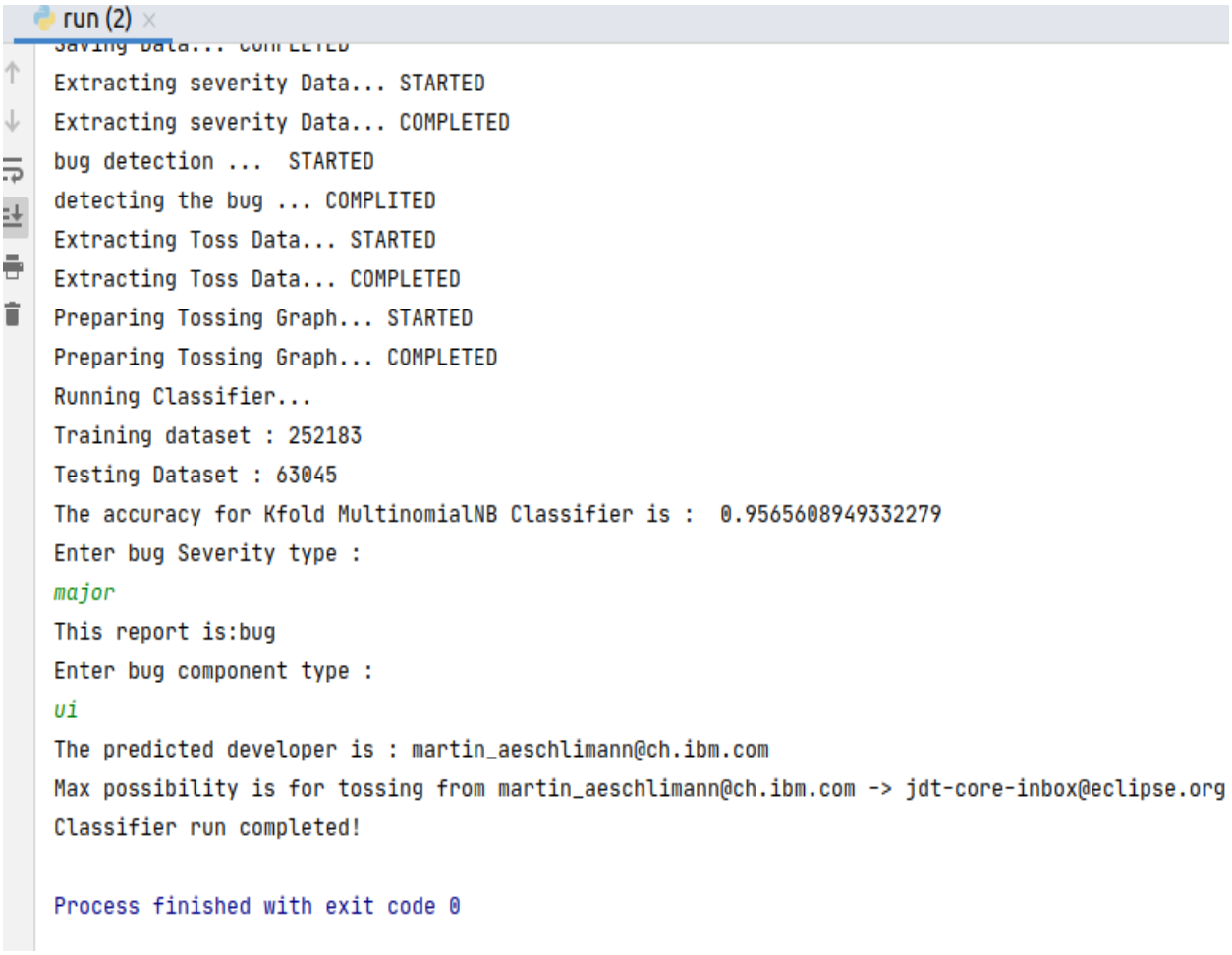


Figure 5.1 Model output for actual bug reports

```
Formatting input... STARTED
Formatting input... COMPLETED
Stemming and stop-word removal... STARTED
Stemming and stop-word removal... COMPLETED
Saving Data... STARTED
Saving Data... COMPLETED
Extracting severity Data... STARTED
Extracting severity Data... COMPLETED
bug detection ...  STARTED
detecting the bug ... COMPLITED
Extracting Toss Data... STARTED
Extracting Toss Data... COMPLETED
Preparing Tossing Graph... STARTED
Preparing Tossing Graph... COMPLETED
Running Classifier...
Training dataset : 252183
Testing Dataset : 63045
The accuracy for Kfold MultinomialNB Classifier is :  0.9565661820892453
Enter bug Severity type :
enhancement
This report is:non-bug
Classifier run completed!

Process finished with exit code 0
```

Figure 5.2 model output for no-bug bug reports

5.8 Summary

In this Chapter, we described the implementation details including discussion about how we have prepared the bug triage dataset and evaluated the bug triage model with the prepared datasets. The Bug Triage dataset includes eclipse bug reports in Surface, Stemmed, and Lemmatized forms. performance of our model is assessed using the evaluation metrics such as confusion matrix, accuracy, precision and recall The experimental results show that our proposed model can successfully distinguish maintainable bug reports from non-maintainable ones with high detection accuracy rate.

We compared the prediction time for our dataset using various algorithms such as Naive Bayes text classifier, Multinomial Naive Bayes and Linear SVM. We arrived at a conclusion that Multinomial Naive Bayes provides higher accuracy.

.

6. Conclusion and Future Work

This Chapter summarizes the major findings in this research work. Moreover, the contributions of the proposed bug triage model and future works are outlined.

6.1 Conclusion

The proposed and developed Bug Triage model can distinguish bugs from non-bug one by analyzing the severity field and then assign to the appropriate developer considering tossing history of developers using Machine Learning Techniques.

The proposed model has the following components: Preprocessor, Feature Extractor, Dataset Constructor, Bug Detector and Bug Assigner. The preprocessor component tokenized, removed stop words and stemmed the dataset. Then feature extraction component extracts the feature vectors from the given bug report and the dataset constructor converted the javascript object notation (JSON) file into sets and help us to split the dataset as training (80%) and testing (20%) sets.

Finally, the MNB classifier is utilized to classify the bug report into BUG and NON_BUG and automatically suggest developers who have the appropriate expertise for handling a bug report, based on the identified component obtained from the short description of the bug report.

From the evaluation result, the developed model is effective in detecting the actual bugs and assign to the concerned developer.

.

The feasibility of our proposed and developed model has been validated on Eclipse covering 315228 bug reports. We demonstrate that our techniques can achieve up to 95.67% prediction accuracy in bug detection and assignment of bugs. From the result, we can conclude that our proposed and developed model has great detection and assignment rate and performance and meets our objective.

6.2 Contribution

The main contributions of this thesis work include:

- The developed model incorporates existing tools and procedures that are used to detect the actuality of the bug and assign it to the right developer.

- The developed model can be used as a framework for the Bug Triage system, which includes recognizing the bug's reality as well as bug assignment using Eclipse domain datasets.

6.3 Future work

For the success of bug triage, we sought to discern between bugs that are genuinely bugs and bugs that aren't. However, even if it is truly a bug, it is often difficult to reproduce the stated bug using the information provided in a bug report, and so this bug is labeled as non-reproducible (NR). A non- reproducible (NR) bug can't be reproduced using the information supplied in the bug report. All attempts to reproduce the problem have been failed, and analyzing the system's code yields no insight into why the described behavior occurs. Following that, further work will entail separating non-reproducible bugs from those that have been determined to be true bugs, as well as predicting fixability for developer assignment for fixable NR bugs.

References

[1] Lohagaonkar D., Kharat K., Bhise O., Korhale A., and Shinde A., "Automatic Bug Triage with Software" International Research Journal of Engineering and Technology, Vol: 05, Issue: 01 Jan 2018

[2] Xuan J., Jiang H., Ren Z., Yan J., and Luo Z., "Automatic Bug Triage using Semi-Supervised Text Classification", software engineering and knowledge engineering, 2010.

[3] Anjali G., "Effective Bug Triage for Non Reproducible Bugs", IEEE/ACM 39th IEEE International Conference on Software Engineering Companion, 2017.

[4] Anvik J., Hiew L., and Murphy G., "Coping with an Open Bug Repository" https://www.researchgate.net/publication/221107928, January 2005.

[5] Lee D. and Seo Y., "Improving bug report triage performance using artifcial intelligence based document generation model" Springer open, 2020.

[6] Hiwse N., and Talmale R, "Framework for Automatic Bug Classification in Bug Triage System", International Conference on Modern Trends in Engineering Science and Technology (ICMTEST 2016), Vol: 2, Issue: 6.

[7] Tran H., Le S., · Nguyen S., and Phong T.,"Analysis of Software Bug Reports Using Machine Learning Techniques", *SN COMPUT. SCI.* **1,** 4 (2020). https://doi.org/10.1007/s42979-019-0004-129.

[8] Sayed M., Hejazi D. and Nafiseh H. "Which Factors Affect Software Projects Maintenance Cost More?" PMCID, 2013 Mar; 21(1): 63–66. doi: 10.5455/AIM.2012.21.63-66

[9] Arudkar S. and Pimpalkar A., "Design of an effective mechanism for automated bug triage System" International Journal of Research in Science & Engineering e-ISSN: 2394-8299 Vol: 3, Issue 1, January 2017.

[10] Abbineni J., Thalluri O."Software Defect Detection using Machine Learning Techniques", Proceedings of the 2nd International Conference on Trends in Electronics and Informatics (ICOEI 2018).

[11] Wagner S., Deissenboeck F., Johann M., and Schwalb W., "An Evaluation of Two Bug Pattern Tools for Java", 1st International Conference on Software Testing, Verification, and Validation, 2008.

[12] Saagarikha S., Susindaran E., and Venugopal G. “Automatic Bug triaging a Machine Learning Approach”, a project report in partial fulfillment for the award of the degree of bachelor of engineering in computer science and engineering, April 2015.

[13] Purao, S., Design research in the technology of information systems: truth or dare. GSU Department of CIS Working Paper, 2002.

[14] Peffers K. , Tuunanen T., Rothenberger M. , and Chatterjee S, "A Design Science Research Methodology for Information Systems Research," Management Information Systems, Vol: 24, No. 3, pp. 45-78, 2007-8.

[15] Hevner A., Ram S., March S., and Park J., "DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH," *Design Science in IS Research* , Vol. 28, No. 1, pp. 75-105, March 2004.

[16] Vaishnavi V., Kuechler B., and Petter S., "DESIGN SCIENCE RESEARCH IN INFORMATION SYSTEMS," Design Science Research in Information Systems. Ais, pp. 1–45, 2004.

[17] Nunamaker F., Chen M., and Purdin T., "Systems development in information sys- tems resear," *Journal of Management Information System,* Vol. 7, No. 3, pp. 89-106, Winter 1990-91.

[18] Ayalew Belay Habtie, Ajith Abraham and Dida Midekso, "Applying Design Science Research to Design and Evaluate Real-Time Road Traffic State Estimation Framework," Vol. 2, p. 331–358 , 2013

[19] Lee D. and Seo Y.,’Improving bug report triage performance using artificial intelligence based documentgeneration model’, SPRING OPEN,2020.

[20] Goyal A. and Sardana N., “NRFixer: Sentiment Based Model for Predicting the Fixability of Non-Reproducible Bugs”, e-Informatica Software Engineering Journal, Vol 11, Issue 1, 2017, pages: 103–116, DOI 10.5277/e-Inf170105.

[21] Cárdenas C., Lu J., Moran K., Mercus A., Penta M., Poshyvanyk D., and Oscar C., "Assessing the Quality of the Steps to Reproduce in Bug Reports," in *ESEC/FSE* , Tallinn, Estonia, August 26–30, 2019.

[22] Hu s. and Zou Z., “Identifying misclassified bug reports”, 2nd International Conference on Materials Science, Machinery and Energy Engineering (MSMEE 2017).

[23] Herzig K., Just S., and Zeller A., “It’s not a bug, it’s a feature: How misclassification impacts bug prediction,” in Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, pp. 392–401.S, 2014.

[24] Antoniol G., Ayari K., Penta M., Khomh F., and Eneuc G, Is it a bug or an enhancement? A text-based approach to classify change requests, in Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds, 23, pp. 304–318. 2008.

[25] Anvik J., and Murphy G., " Reducing the effort of bug report triage: Recommenders for development-oriented decisions.," ACM Transactions on Software Engineering and Methodology (TOSEM), 2011.

[26] Singhal S., and Sharma B., "Optimized Bug Report Triaging using N-Gram Features and NonReproducible Classification," *IJSRD - International Journal for Scientific Research & Development*| , Vol. 3, No. 07, 2015.

[27] Cubranic D. and Murphy G., Automatic bug triage using text categorization. Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, 2004

[28] Younus M., Neelofar E., and Mohsin H.,” An Automated Approach for Software Bug Classification”, 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems,2012.

[29] Kim. H, Zhang. R, and Gong L., “Dealing with noise in defect prediction,” in Software Engineering (ICSE), 2011 33rd International Conference on. IEEE, pp. 481–490, 2011.

[30] Kochhar P., Le T., and Lo D.,” It's not a bug, it's a feature: does misclassification affect bug localization?, in Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, pp. 296–299”, 21 September 2015.

[31] Kukkar A. and Mohana R.," A Supervised Bug Report Classification with Incorporate and Textual field Knowledge", International Conference on Computational Intelligence and Data Science (ICCIDS 2018), 2018.

[32] Sardana N. And Anjali G.., " Visheshagya: Time based expertise model for bug report assignment," in *In Contemporary Computing (IC3),Seventh International Conf on (pp.). IEEE*, 2016.

[33] Neamtiu I., Shelton .and Bhattacharya P., "Automated, highlyaccurate, bug assignment using machine learning and tossing graphs," Journal of Systems and Software, Vol. 85, p. 2275–2292, 2012.

[34] Kim S., Zimmermann T. And Jeong G., "Improving bug triage with bug tossing graphs," in In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium,2009

[35] Shaik S., Murty M., and Krishna J. "An efficient and automatic bug triage for mining software repositories" October 2016 IJSDR | Vol 1, Issue 10.

[36] Sayed F., Shah H., Kharat N., Ohal P., and Deshmukh G., "Bug Triage: An Automated Process", International Research Journal of Engineering and Technology (IRJET), Vol: 04 Issue: 06 | June -2017.

[37] Artchounin D.,"Tuning of machine learning algorithms for automatic bug assignment", Linköping University Department of Computer Science Master thesis, 2017

[38] Kulkarni N. and Bairagi V, EEG-based diagnosis of Alzheimer disease, 1st ed. Amsterdam, Netherlands: Elsevier Inc., pp. 45-47, 2018.

[39] Fang B., Wang J., Liu Q., and Cui X., "Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, pp. 1-6, 2019.

[40] Tewodros WorkuOsman,"Anomaly Based Peer-to-Peer Botnet DetectionUsing Fuzzy-Neuro Network", A Thesis Submitted to the Department of Computer Science in Partial Fulfilment for the Degree of Master of Science in Computer Science, Addis Ababa, Ethiopia,October 2020.

[41] https://www.ibm.com/cloud/learn/supervised-learning, acces 06/10/2021.

[42] Shobha G., and Rangaswamy S., "Machine Learning", Handbook of Statistics, pp. 197- 228, 2018.

[43] Talabis M., McPherson R., and Martin J., Information security analytics. Amsterdam: Elsevier, 2015.

[44] Aamir M. and Zaidi S., "Clustering based semi-supervised Machine Learning for DDoS attack classification", Journal of King Saud University - Computer and Information Sciences, 2019.

[45] Osisanwo F., Akinsola J., Awodele O., Hinmikaiye J., Olakanmi O., and Akinjobi J., "Supervised Machine Learning Algorithms: Classification and Comparison", International Journal of Computer Trends and Technology (IJCTT) – Vol 48 N0 3, June 2017.

[46] Ashraf M., Frank E., Pfahringer B., and Holmes G., "Multinomial Naive Bayes for Text Categorization Revisited", at Springer-Verlag Berlin Heidelberg, 2013

[47] Abbas M., Ali K., Memon S., and Jamali A., "Multinomial Naive Bayes Classification Model for Sentiment Analysis", Classification for Sentiment Analysis Simultaneous wireless Information and Power Transfer, March 2019.

[48] Bird C., Menzies T., and Zimmermann T., The art and science of analyzing software data. Waltham, MA: Morgan Kaufmann, 2015.

[49] Sumi M., and Narayanan A., "Improving Classification Accuracy Using Combined Filter + Wrapper Feature Selection Technique," 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, pp. 1-6, 2019.

[50] Bakhshandeh A., and Eskandari Z., "An efficient user identification approach based on Netflow analysis," 2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC), Tehran, pp. 1-5, 2018.

[51] Russo F., Raju R., Clarke C., Yang N., Escalona A., Tappert C. and Leider A., "Software Bug Triage using Machine Learning and Natural Language Processing", Proceedings of Student-Faculty Research Day Conference, CSIS, Pace University, May 8th, 2020.

[52] Rossum G., and the python development team, "python free tutorial", june 17,2020.

[53] https://scikitlearn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html, accesed may 12,2021.

[54] S. Misra, H. Li, and J. He, Machine Learning for subsurface characterization. Amsterdam, Netherlands: Elsevier Inc., pp. 129-130, 2020.

[57] Yoonsuh Jung, "Multiple predicting K -fold cross-validation for model selection", Journal of Nonparametric Statistics, November 2017

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university and that all sources of materials used for the thesis have been duly acknowledged.

Declared by:

Name: Meseret Andualem Beyene

Signature: _____

Date: _____

Confirmed by advisor:

Name: Ayalew Belay (PhD)

Signature: _____

Date: _____

Place and date of submission: Addis Ababa University, July 2021.