AN INTEGRATION OF MACHINE LEARNING ON SOFTWARE BUG REPORTING

A Capstone Project Presented to the Graduate Program
College of Engineering and Technology
Pamantasan ng Lungsod ng Maynila

In Partial Fulfillment of the Requirements for the Degree
Master's in Information Technology

By
Joane Marie F. Llamera

Dr. Khatalyn E. Mata
Thesis Adviser

August 2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

No table of figures entries found.

Chapter One

INTRODUCTION

1.1 Background of the Study

In the lifecycle of a software project, bug fixing is an essential aspect of the development and maintenance phases. A bug is a coding error that can lead to anomalous program behaviour. Within organizations, bug tracking allows for the assigning, monitoring and resolution of issues and bugs. When software testers find a bug, they will create a bug report to begin the process of fixing it. Software bugs must be dealt swiftly in large-scale software projects wherein erroneous bug report assignment to development teams can be quite costly. According to a survey conducted by the National Institute of Standards and Technology (NIST), the annual cost of software vulnerabilities is estimated to be around $59.5 billion (NIST, 2002). According to some software maintenance studies, maintenance costs account for at least 50%, and in some cases more than 90%, of total costs associated with a software product (Koskinen, 2003; Seacord et al., 2003), while other estimates place maintenance costs at several times the cost of the initial software version (Koskinen, 2003; Seacord et al., 2003). (Sommerville, 2004). According to these studies, improving the bug-fixing process will minimize evolution effort and lower software development costs.

Issue report assignment is a crucial phase in the process of locating and correcting a bug, since it is the skill of matching an open bug report to the most likely developer to handle it. With ever-larger software development systems including more workers with varying skills, it's important to consider how bugs are assigned to technical groups rather than to a single developer. The classification of defects is a crucial phase in the bug correction process as it sends the errors to a key developer who can fix them (Chauhan et. Al, 2020). Manual bug triaging can be a time-consuming operation due to the enormous

volume of bug reports sent every day. Additionally, assigning a bug to the incorrect team or developer increases the cost and time to remediate the bug.

Bug assignment, or the process of assigning defects, is hampered by several factors: it is labor-intensive, time-consuming, and error-prone if done manually; also, it is difficult to maintain track of current engineers and their competence in large projects. Growth makes it even more difficult to find the right developer to fix a new bug. For example, as projects add more components, modules, developers, and testers increase, the number of bug reports submitted daily grows, making manually recommending developers based on their expertise is difficult.

The use of recommenders for bug report triage judgments is especially significant in large software development projects, where both the frequency of reported issues and the huge number of active engineers might make it difficult to identify the appropriate developer to work on a specific issue. For example, on a daily basis, 135 reported problems are submitted to Mozilla's open-source system (Liu et al., 2013). In these huge open systems, managing a high volume of new bugs submitted every day is a demanding effort.

1.2 Statement of the Problem

Software bugs are an unavoidable part of the software development process. While software bugs must be fixed for the product to be of high quality, addressing them in a timely manner comes at a cost. Many organizations use issue tracking systems like Mozilla BugZilla and Atlassian JIRA to deal with them in a methodical way. While issue tracking systems have been proven to be effective in managing issues, many actions associated with bug resolution require a significant amount of time and effort.

According to Jeong et al.’s research, 44% of problems are assigned to the wrong developer or team. Manual defect assignment is time-consuming and error-prone, according to multiple studies (Baysal et al, 2009; Jeong et al, 2009; Bhattacharya et al,

2012), resulting in "bug tossing" (i.e., reassigning problem reports to another developer) and delayed bug remedies.

Even with issue trackers, it is still a time-consuming task to assign bug reports to programmers for fixing, since many bug reports have to be assigned (Anvik et al., 2006) and some projects have many teams of programmers.

One of key process that can be greatly improved on a Bug Reporting System is on how the previously resolved bugs can contribute on bug triaging process to reduce the time spent on identifying the appropriate resolving team and minimize bug tossing instances.

1.3 Objectives of the Study

The main objective of this capstone project is to apply Machine Learning for bug assignment on Software Bug Reporting.

Specifically, this capstone project seeks to achieve the following objectives:

1. To extract previously fixed bugs and generate keywords by using NLP (Natural Language Processing).
2. To categorize and train extracted data by development team using RBF (Radial Basis Function) - SVM (Support Vector Machines) algorithm.
3. To recommend resolving team to incoming bugs based on trained data model.

1.4 Scope and Limitations

This study will focus on creating a prototype bug reporting tool with enhanced feature on bug assignment process by utilizing machine learning techniques. The primary goal of the application is on suggesting a resolving team as the bug assignee.

Attributes of a defect report such as Title, Description, Priority, Severity, Reported Date, Reported by and Status will be manually inputted to the bug reporting system. However, the study will only use the two fields namely Title (summary) and Description for data extraction.

The study will exclude other processes in a defect triage and will only be concerned on identification of the bug assignee.

For instances wherein there is no existing keyword matches from the trained data model, manual intervention is necessary where the bug reporter shall select the appropriate assignment team.

1.5 Significance of the Study

Results obtained from this capstone project will benefit the following stakeholders:

Bug Reporter. The bug auto assignment feature will decrease the manual effort and time spent on identifying who is the appropriate team to resolve the reported software issue. Having an automated assignment feature will remove the manual error of assigning the bug to an incorrect team.

Customers/Clients. By reducing the error on incorrect bug assignment, reported issues are fixed in a timely manner as bug tossing instances are reduced.

Software Companies. Efficient bug fix process can improve not only the appeal of an app to consumers, but also the reputation of the company that created it. The reduced time spent on bug assignment will ultimately contribute om reducing the cost of bug fixing.

1.6 Definition of Terms

Software Bug. A weakness or error in a computer program or system that leads it to produce an inaccurate or unexpected result or act in unexpected ways. The majority of defects are caused by faults and errors in a program's design or source code, as well as in the components and operating systems that such programs use. Compilers creating erroneous code is responsible for a handful of them. A buggy program is one that has numerous defects and/or bugs that substantially impair its functionality (defective). Errors caused by bugs can have a cascading effect. Bugs might have modest impacts, or they can cause the software to crash or the computer to freeze. Other bugs are classified as security bugs, as they could allow a hostile person to circumvent access controls and gain unauthorized powers.

Bug Tossing. Once a bug report has been assigned, developers can reassign the bug to other developers. If the developer is unable to remedy the bug, for example because the bug was assigned incorrectly, it is passed on to another developer. Until the bug report reaches the bug resolver, the tossing procedure continues. Bug chucking not only extends the time it takes to fix bugs, but it also wastes the work of bug triagers and developers.

Bug Report. A bug report is a detailed report that details what's wrong and needs to be fixed in software or on a website. The report provides a request and/or details for how to address each issue, as well as a list of causes, or seen faults, to highlight exactly what is perceived as wrong.

Bug Reporting System. A bug tracking system, sometimes known as a defect tracking system, is a piece of software that maintains track of reported issues in software development projects. It might be considered an issue tracking system. Many bug tracking systems, including those used by most open-source software projects, allow end users to directly enter bug reports. Other systems are solely utilized within a software development company or group. Bug tracking software is frequently connected with project management software.

Chapter Two

REVIEW OF RELATED LITERATURE

This chapter presents the different research and other literatures from both foreign and local researchers, which have significant bearings on the variables included in the research. It focuses on several aspects that will help in the development of this study. The literatures of this study come from books, journals, articles, electronic materials such as PDF or E-Book, and other existing thesis and dissertations, foreign and local which are believed to be useful in the advancement of awareness concerning the study.

2.1 Related Literature and Related Studies

Bug Reporting Systems

Software defects are a problem that IT companies all around the world have to deal with software bugs. Managing programming problems consumes more than 45 percent of programming companies' budgets. Bug repositories, such as Bugzilla, a prominent and open-source bug repository, store software bugs. A bug reporting system is generally an essential element of a satisfactory software development infrastructure and regular use of a bug or issue reporting system consider one of the “sign of an honourable software team”. A large element of a bug reporting system is a database that tracks and records the information about bugs which is known (Jalbert and Weimer, 2008).

When a software bug is discovered, it is tested by a reporter, who could be a tester, developer, or deployer, and if it is determined to be a genuine bug, the occurrence is reported. A bug report is a document that contains information on a bug that can be used to reproduce it. Once a bug report is created, a human triager will allocate the bug to a developer. If the assigned developer is unable fix the bug, the bug is migrated to another developer. The process of assigning a bug report to an appropriate developer is called bug

triage (Arudkar et al., 2017). Bug triage's purpose is to review, prioritize, and assign defect solutions. The triage team must confirm the severity of the fault, make necessary changes, conclude defect resolution, and assign resources. This method is primarily utilized in agile project management (Hamilton, 2021).

The bug reports in a bug repository are called bug data. In big software development projects, the Automated Bug Tracking System (BTS) monitors bug reports and a list of developers who work on correcting them.

Several works have been done to build automatic bug triagers using machine learning algorithms. It is not surprising that issue trackers constitute a central point of focus in current software engineering empirical research such as predicting the severity of reported bug in a study conducted by Lamkanfi et al. (2010). In the study of Jonsson et al. (2016), they presented a bug triager based on a machine learning ensemble and tested it on five industrial applications. They combined well-known machine learning algorithms to increase the performance of automatic bug triage. A study by Kumari snd Singh (2018) built classifiers based on machine learning techniques Naïve Bayes (NB) and Deep Learning (DL) using entropy-based measures for bug priority prediction while considering the severity, summary weight and entropy attribute to predict the bug priority.

Support Vector Machine (SVM)

Support vector machine, according to Hearst et al. (1998), provides a classification learning model and algorithm rather than a regression model and algorithm. It manipulates the simple mathematical model $y = wx + \gamma$ to allow for linear domain division. There are two types of support vector machines: linear and nonlinear (Hastie et al. 2009). If the data domain can be divided linearly (e.g., straight line or hyperplane) to separate the classes in the original domain, it is called a linear support vector machine. Nonlinear support vector machines are used when the data domain cannot be divided linearly but can be changed to a space called the feature space where the data domain may be divided linearly to separate the classes.

The mapping of the data domain into a response set and the division of the data domain are the steps in the linear support vector machine. The mapping of the data domain to a feature space using a kernel function (Scholkopf et al. 1999), the mapping of the feature space domain into the response set, and finally the division of the data domain are the steps in nonlinear support vector machines.

Kanwal et al. (2010) suggested a bug priority recommender that is based on classification techniques such as SVM. The bug priority recommender assigns a priority rating to newly arrived problems automatically. The eclipse dataset platform product was validated by the authors. Kanwal et al. (2012) expanded on this research by comparing two classifier algorithms, namely Support Vector Machine and Nave Bayes. The results reveal that Support Vector Machine outperforms Nave Bayes for textual features while Nave Bayes outperforms Support Vector Machine for categorical data.

Supervised Machine Learning Techniques in Bug Reporting

Yuan Tian et al. (2013) made an attempt using a new framework called DRONE (PreDicting PRiority via Multi-Faceted FactOr ANalysEs). GRAY is a new classification engine developed by the authors (ThresholdinG and Linear Regression to ClAssifY). The experiment was carried out using Eclipse project training and testing data sets. The authors compared SeverisPrio and SeverisPrio+ to his planned work DRONE. Menzies et al. proposed the SEVERIS approach as a foundation for bug severity evaluation (2008). Tian et al. (2015) expanded on the research and used an automated approach known as DRONE.

It uses machine learning and information from a reported bug to forecast the priority level. The experimental findings were validated using Eclipse project bug reports, and the results suggest that DRONE can outperform a baseline method. Sharma et al. used summary attributes to assess the efficacy of various machine learning techniques, including SVM, NB, KNN, and NNet, in predicting bug priority. Except for the NB technique, the accuracy of different classifier algorithms in predicting the priority of reported problems inside and across projects is found to be above 70%.

Mani et al. (2018) introduced a deep learning system that learns a paragraph level representation while retaining word ordering and semantic relationships over a longer context. Multinomial naive Bayes, cosine distance, support vector machines, and softmax classifier were among the classifiers utilized by the authors. Bug reports from three famous open-source bug repositories - Google Chromium (383,104), Mozilla Core (314,388), and Mozilla Firefox (314,388) - were used to validate the experimental result (162,307). In all three datasets, DBRNN-A combined with the softmax classifier outperforms the bag-of-words model, improving the rank-10 average accuracy.

Natural Language Processing (NLP)

Natural language processing (NLP) is an area of AI that aids computers in interpreting and manipulating language. The goal of NLP researchers is to learn how people perceive and use language so that appropriate tools and techniques may be developed to help computers understand and manipulate natural languages to execute tasks (Chowdhury, 2003).

In a given context or domain, a word or sentence may have a specific meaning or connotation, and it may be related to a large number of other words and sentences. According to Liddy (1998) and Feldman (1999), it is critical to be able to discern between the seven interdependent levels that humans utilize to derive meaning from written or spoken languages in order to understand natural languages:

Phonetic or phonological level - deals with pronunciation

Morphological level - deals with the smallest parts of words that carry meaning, and suffixes and prefixes

- Lexical level - deals with lexical meaning of words and parts of speech analyses
- Syntactic level - deals with grammar and structure of sentences
- Semantic level - deals with the meaning of words and sentences

- Discourse level - deals with the structure of different hands of text using document structures
- Pragmatic level - deals with the knowledge that comes from the outside world, i.e., from outside the content of the document

A natural language processing system may involve all or some of these levels of analysis.

Chapter Three

THEORETICAL FRAMEWORK

Bug Report and Reporting Process

To control the quality of software products, bug tracking systems store data from bug reports. They're typically used to organize bug-reporting workflows and send them to system operators for resolution. Predefined fields, text descriptions, attachments, and dependencies are all included in a bug report for the purposes of problem management and resolution.

A bug's attributes are represented through predefined fields. Some attributes, such as the creation date and the reporter who reported the problem, cannot be changed. Other attributes, such as product, component, priority, and severity, may change over the life of the defect. Some properties, such as the assignee, the present status, and the eventual resolution, may be regularly updated. The natural language contents of a bug report, including the title of the bug report and a comprehensive description of the bug, are referred to as the text description. An example of a bug report is shown in Figure 3.1.

Figure 3.0.1 Sample Bug Report Template in JIRA

When a new bug report is submitted or reported, the manager (a senior developer or the project leader) selects a developer to address the issue. During the bug-fixing process, developers and reports will engage with one another via comments, and they will be able to provide extra resources such as screenshots to aid in the bug-fixing process. In addition, the developer has the ability to reassign the bug report to other developers. Bug tossing is the term for this type of reassignment. Bugs are frequently assigned to developers by mistake, or the developer wishes to incorporate other developers with additional knowledge in order to fix the bug. The status of the bug report will be modified after the bug has been addressed.

Manually evaluating and allocating problem reports is time-consuming and tiresome, particularly in software projects with a large number of bug reports and developers. The Eclipse and Mozilla projects, for example, receive hundreds of bug reports per day, and properly allocating each of them to one of the thousands of developers would take a long time.

Tokenization

In Natural Language Processing, tokenization is a typical activity (NLP). Both classic NLP approaches like Count Vectorizer and Advanced Deep Learning-based architectures like Transformers rely on this phase. Tokenization is the process of breaking down a large chunk of text into smaller tokens. Tokens can be words, characters, or sub words in this case. Tokenization can thus be divided into three categories: word, character, and sub word (n-gram characters) tokenization.

The most popular method for creating tokens is to use space. The tokenization of the statement yields three tokens – Never-give-up, assuming space as a delimiter. Because each token is a word, it is a Word tokenization example.

Similarly, tokens can be either characters or sub words. For example, let us consider “smarter”:

Character tokens: s-m-a-r-t-e-r

Sub word tokens: smart-er

The most often used tokenization algorithm is word tokenization. It divides a chunk of text into distinct words using a delimiter. Different word-level tokens are created depending on the delimiters. Word tokenization includes pre-trained word embeddings like Word2Vec and GloVe.

Lemmatization

Lemmatization is the process of combining a word's several inflected forms into a single item for analysis. Similar to stemming, lemmatization adds context to the words. As a result, it joins together words that have comparable meanings. Stemming and lemmatization are both part of text preprocessing. Many individuals are perplexed by these two terms. Some people confuse the two. Because lemmatization performs morphological examination of the words, it is preferable over stemming.

Applications of lemmatization are:

- Used in comprehensive retrieval systems like search engines.
- Used in compact indexing

Lemmatization is a term that relates to performing things correctly using a vocabulary and morphological analysis of words, with the goal of removing inflectional endings solely and returning the base or dictionary form of a word, also known as the "lemma." When faced with the token "saw," stemming might only yield s, whereas lemmatization might try to return "see" or "saw," depending on whether the token was used as a verb or a noun. Stemming and lemmatization may also differ in that stemming usually

collapses derivationally related words, but lemmatization usually just collapses a lemma's different inflectional forms. Linguistic processing for stemming or lemmatization is generally done via a separate plug-in component to the indexing process, and there are a number of commercial and open-source options available.

Form	Stem	Lemma
Studies	Studi	Study
Studying	Study	Study
beautiful	beauti	beautiful
beautifully	beauti	beautifully

Figure 3.0.2 Output Generated from Stemming vs. Lemmatization

Stop Words

Stop words are English words that don't add much to a sentence's meaning. They can be safely ignored without jeopardizing the sentence's meaning. Words like the, he, and have, for example. Both while indexing entries for searching and retrieving them as the result of a search query, search engines have been configured to ignore them.

Input: “This is a sample sentence, showing off the stop words filtration.”

Output: ['This', 'sample', 'sentence', 'showing', 'stop', 'words', 'filtration']

Word Vectorization

It's a method for converting a group of text documents into numerical feature vectors. There are other approaches for converting text data into vectors that the model can comprehend, but the TF-IDF method is by far the most prevalent. The term "Term

Frequency — Inverse Document Frequency" is an abbreviation that stands for "Term Frequency — Inverse Document Frequency."

The TF-IDF is widely employed in machine learning algorithms for a variety of purposes, including stop-word elimination. These are terms like "a, the, an, it" that are commonly used yet provide little information. Term frequency and inverse document frequency are the two components of the TF-IDF.

In a simple language, TF-IDF can be defined as follows:

In the TF-IDF, a high term frequency (in the provided document) and a low document frequency of the term in the entire collection of documents result in a high weight.

The TF-IDF algorithm is a combination of two algorithms.

- Term Frequency
  - Term frequency (TF) is how often a word appears in a document, divided by how many words there are.
  - TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
- Inverse document frequency
  - Term frequency is how common a word is, inverse document frequency (IDF) is how unique or rare a word is.
  - IDF(t) = log_e(Total number of documents / Number of documents with term t in it)

Support Vector Machines (SVM)

SVM stands for Support Vector Machine and is a supervised machine learning technique that can be used for classification and regression. SVMs are based on the concept of determining the optimum hyperplane for dividing a dataset into two classes.

Types of SVM

- Linear SVM: Linear SVM is used to categorize data that is linearly separable, i.e. a dataset that can be divided into two groups using a single straight line. These data points are referred to as linearly separable data, and the classifier is known as a Linear SVM classifier.
- Non-linear SVM: Non-linear SVM is used to classify data that cannot be classified using a straight line. We do this by employing a kernel approach, which places data points in a higher dimension from which they may be separated using planes or other mathematical functions. Non-linear data is referred to as such, and the classifier utilized is known as a Non-linear SVM classifier.

To begin, a set of points from each class is plotted and visualized as shown in Figure 3.3. We can efficiently separate these two classes in a 2-d space by simply applying a straight line. However, various lines can be used to classify these classes. You can choose from a variety of lines or hyperplanes (green lines). The inquiry will be: which of these lines is appropriate for classification?

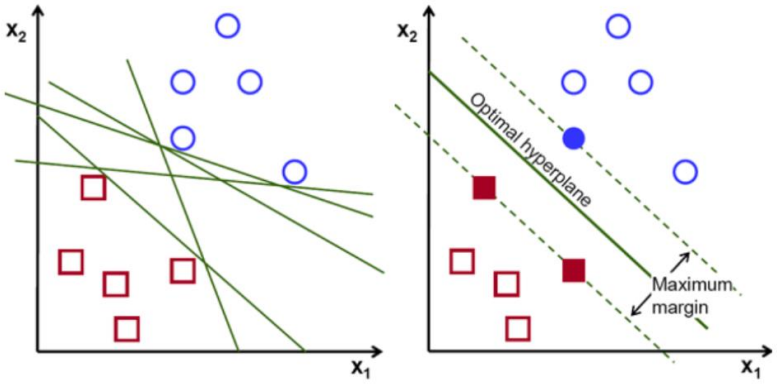


Figure 3.0.3 Linear SVM Visualization

Essentially, choose the hyper-plane that best separates the two groups. This is accomplished by increasing the distance between the closest data point and the hyper-

plane. The better the hyperplane and the better the categorization results, the bigger the distance. The hyperplane chosen has the greatest distance from the nearest point from each of those classes, as shown in the diagram below.

The support vectors of the hyperplane are the two dotted lines that run parallel to the hyperplane and cross the nearest points of each of the classes. The distance between the supporting vectors and the hyperplane is now referred to as a margin. And the SVM algorithm's goal is to maximize this margin. The hyperplane with the greatest margin is the best hyperplane.

Consider the distinction between good and harmful cells. xi is an n-dimensional feature vector that can be plotted in n-dimensional space. The class yi is assigned to each of these feature vectors. The class yi can be either a +ve or a -ve (for example, good=1 or good=-1). The hyperplane's equation is y=w.x + b = 0. The line parameters W and b are used here. The previous equation returns a value of 1 for examples in the +ve class and a value of -1 for examples in the -ve class.

The hyperplane is defined by determining the best values for w (weights) and b (intercept). The cost function is minimized to find these best values. The SVM model or the line function f(x) efficiently distinguishes the two classes once the algorithm accumulates these optimal values.

In a nutshell, the optimal hyperplane has equation w.x+b = 0. The left support vector has equation w.x+b=-1 and the right support vector has w.x+b=1.

Thus, the distance d between two parallel liens Ay = Bx + c1 and Ay = Bx + c2 is given by d = |C1–C2|/√A^2 + B^2. With this formula in place, we have the distance between the two support vectors as 2/||w||.

The cost function for SVM looks the like the equation below:

$$J(\mathrm{w}) = \frac{1}{2}\|\mathrm{w}\|^2 + \mathrm{C}\left[\frac{1}{\mathrm{N}}\sum_{\mathrm{i}}^{\mathrm{n}} \max(0, 1 - \mathcal{Y}_{\mathrm{i}} * (\mathrm{w} \cdot \chi_i + b))\right]$$

The λ parameter denotes that a larger λ provides a broader margin, and a smaller λ would yield a smaller margin. In addition, the cost function's gradient is determined, and the weights are adjusted in the direction of lowering the lost function.

Algorithm for Non-linear SVM

It is simple to create a linear hyper-plane between these two classes in the SVM classifier. However, if the data is not linearly separable, the SVM algorithm employs a technique known as the kernel trick.

The SVM kernel function translates a lower-dimensional input space to a higher-dimensional space. To put it another way, it turns a not separable problem into a separable problem. Complex data transformations are performed based on the labels or outputs that specify them.

Figure 3.4 shows non linearly separable data. The left-hand set of data points is clearly not linearly separable. When a function is applied to a set of data points, however, we receive changed data points in a higher dimension that may be separated using a plane.

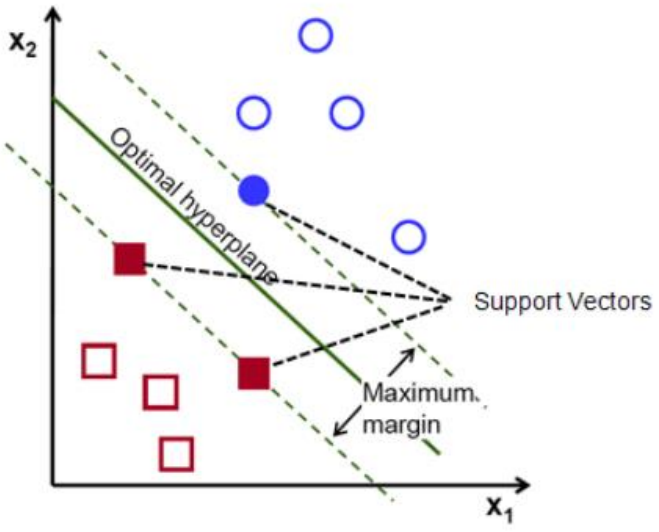


Figure 3.0.4 Nonlinearly Separable Data Visualization

We need to add an extra dimension to separate non-linearly separable data items. Two dimensions, x and y, have been employed for linear data. We add a third dimension, say z, to these data points. Let z=x2 +y2 in Figure 3.4.

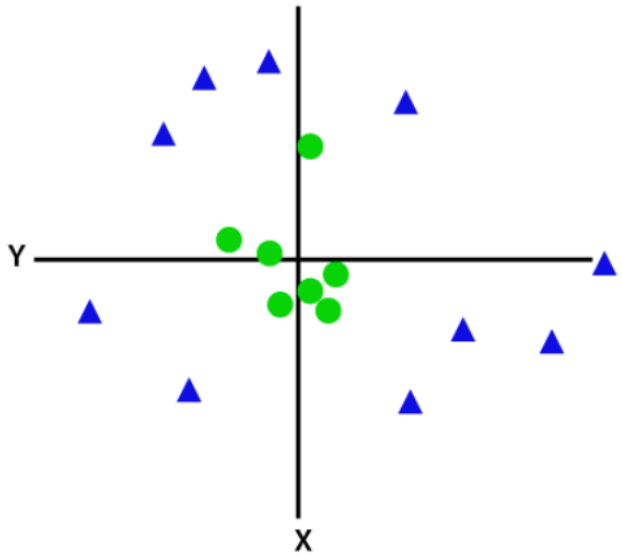


Figure 3.0.5 Data Visualization with third dimension

The sample space is transformed by the z function, or increased dimensionality, and the above image becomes as follows:

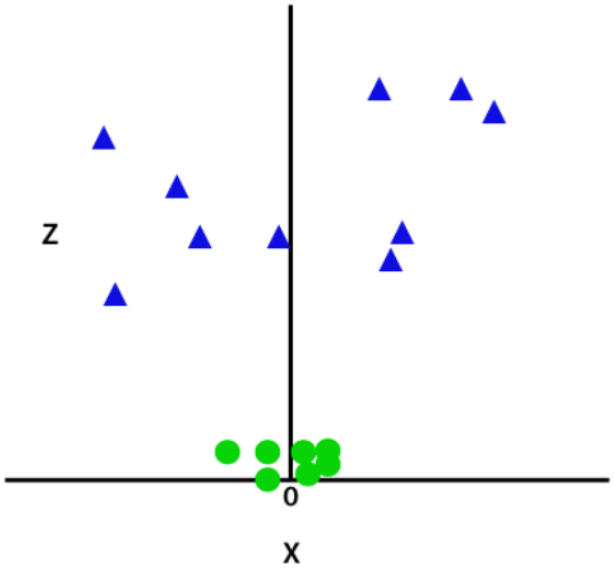


Figure 3.0.6 Transformed Data with Z function

Close examination reveals that the data points above can be split using a straight-line function that is either parallel to the x axis or inclined at an angle. Linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid kernel functions are all included.

RBF Kernel

Due to its resemblance to the Gaussian distribution, RBF kernels are the most generic form of kernelization and one of the most extensively used kernels. For two points X1 and X2, the RBF kernel function computes their similarity, or how near they are to one other. This kernel can be expressed mathematically as follows:

$$K(X_1, X_2) = \exp\left(-\frac{\|X_1 - X_2\|^2}{2\sigma^2}\right)$$

where,

1. ‘σ’ is the variance and our hyperparameter

2. $||X_1 - X_2||$ is the Euclidean (L_2-norm) Distance between two points X_1 and X_2

Let d_{12} be the distance between the two points X_1 and X_2, we can now represent d_{12} as follows:

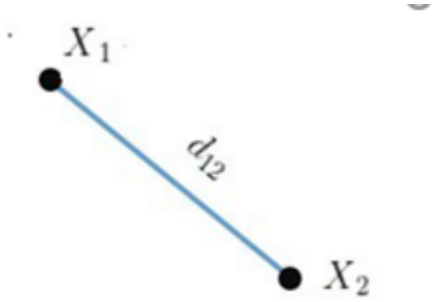


Figure 3.0.7 Distance between two points in space

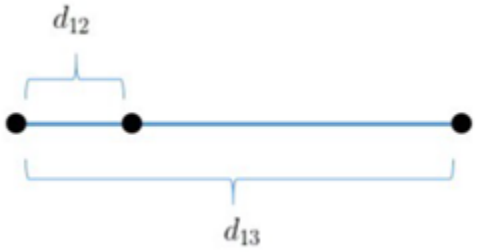
The kernel equation can be re-written as follows:

$$K(X_1, X_2) = \exp\left(-\frac{d_{12}}{2\sigma^2}\right)$$

The maximum value that the RBF kernel can be is 1 and occurs when d_{12} is 0 which is when the points are the same, i.e. $X_1 = X_2$.

- When the points are the same, there is no distance between them and therefore they are extremely similar
- When the points are separated by a large distance, then the kernel value is less than 1 and close to 0 which would mean that the points are dissimilar

Because we can see that as the distance between the points increases, they become less similar, distance can be regarded of as an analogue to dissimilarity.



$$d_{12} < d_{13}$$

$$K(X_1, X_2) > K(X_1, X_3)$$

Figure 3.0.8 Similarity decreases as distance increases

It is critical to determine the appropriate value of '' in order to determine which points should be regarded comparable, and this may be proven case by case.

a] σ = 1

When σ = 1, $\sigma^2 = 1$ and the RBF kernel's mathematical equation will be as follows:

$$K(X_1, X_2) = \exp\left(-\frac{\|X_1 - X_2\|^2}{2}\right)$$

The curve for this equation is shown below, and we can see that the RBF Kernel reduces exponentially as the distance rises and is 0 for distances larger than 4.

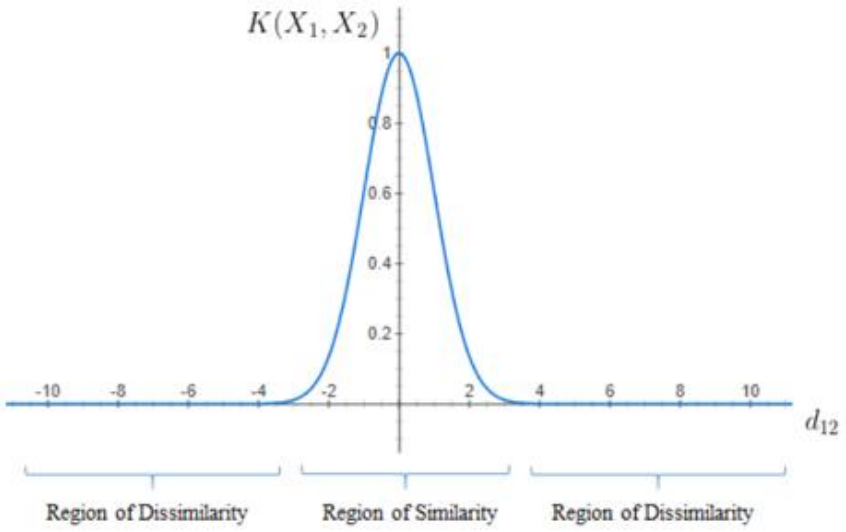


Figure 3.0.9 RBF Kernel for σ = 1

- We can notice that when $d_{12} = 0$, the similarity is 1 and as d_{12} increases beyond 4 units, the similarity is 0
- From the graph, we see that if the distance is below 4, the points can be considered similar and if the distance is greater than 4 then the points are dissimilar

b] σ = 0.1

When σ = 0.1, $\sigma^2 = 0.01$ and the RBF kernel's mathematical equation will be as follows:

$$K(X_1, X_2) = \exp(-\frac{\|X_1 - X_2\|^2}{0.01})$$

The width of the Region of Similarity is minimal for σ = 0.1 and hence, only if points are extremely close, they are considered similar.

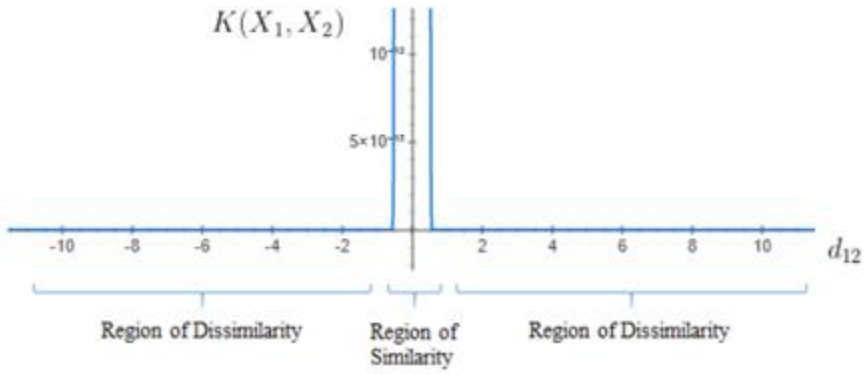


Figure 3.0.10 RBF Kernel for σ = 0.1

- We see that the curve is extremely peaked and is 0 for distances greater than 0.2
- The points are considered similar only if the distance is less than or equal to 0.2

b] σ = 10

When σ = 10, σ² = 100 and the RBF kernel's mathematical equation will be as follows:

$$K(X_1, X_2) = \exp(-\frac{\|X_1 - X_2\|^2}{100})$$

The width of the Region of Similarity is large for σ = 100 because of which the points that are farther away can be considered to be similar.

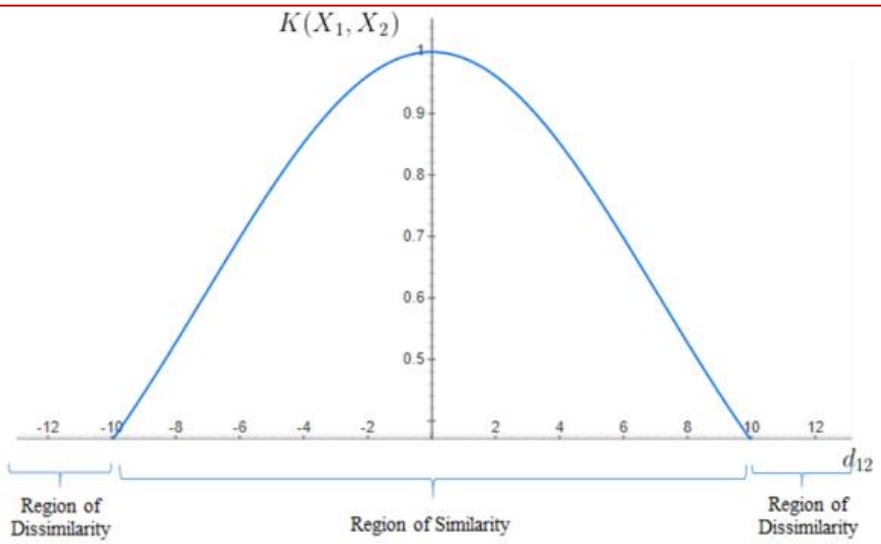


Figure 3.0.11 RBF Kernel for $\sigma = 10$

- The width of the curve is large
- The points are considered similar for distances up to 10 units and beyond 10 units they are dissimilar

3.1 Conceptual Framework

This section aims to demonstrate the overview of the final product of this capstone project. An I-P-O (Input-Process-Output) model will be used as the conceptual schema of the system It identifies relevant variables, inputs, mappings, and other components and how they will interact with each other. This includes all the underlying concepts and their associated mappings based on the system's use.

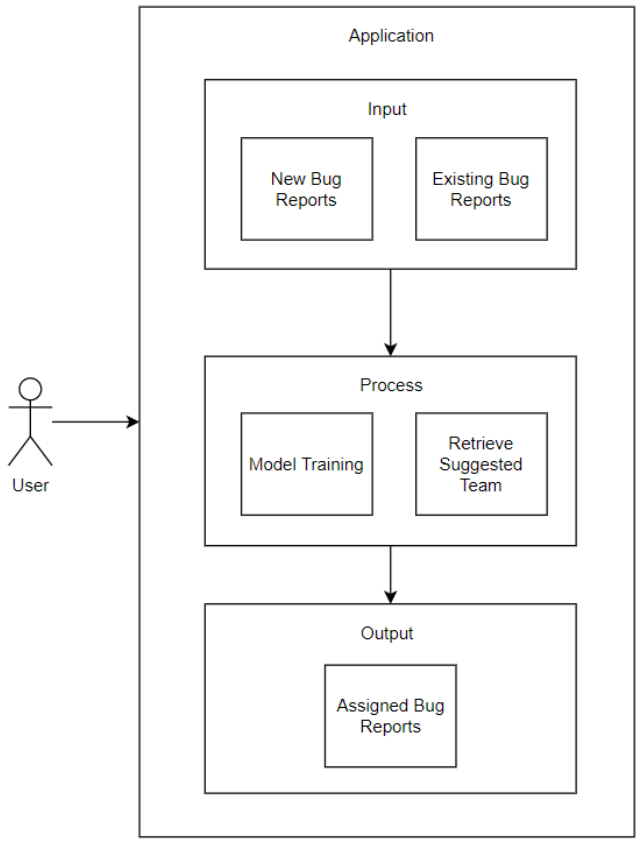


Figure 3.0.12 Conceptual Framework Diagram

Figure 3.12 illustrates the flow of the data to illustrate how the suggested resolving team are generated.

Previously resolved bug reports are used to train the data model. Labels extracted from the bug Title and Description through data pre-processing and are used to train the data model.

One of the system users are the bug reporter who will manually input the bug report attributes such as Title, Description, Priority, Severity, Reported By, Reported Date. With the algorithm therein, when a bug reporter enters the bug details, the application will be able to generate the recommended team to resolve the bug.

Chapter Four

METHODOLOGY

The proponent of this capstone project used prototype method in delivering the objectives of this project.

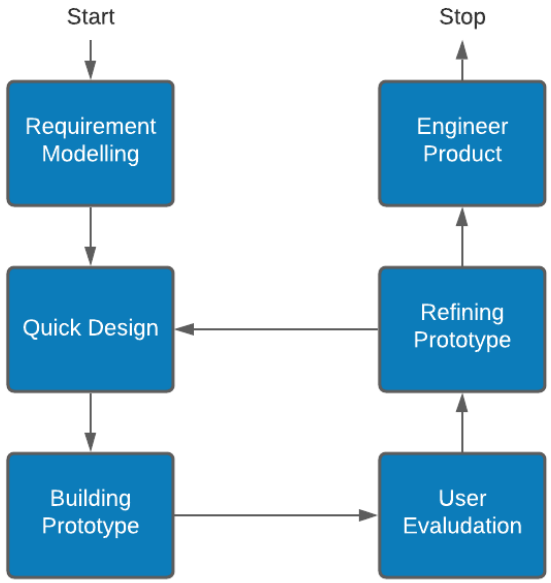


Figure 4.0.1 Prototype Model

The phases of the prototype model involve the following steps:

4.1 Requirements Modeling

In this Section, the proponent presents an approach for predicting the resolving team of each newly reported bug using resolved bug reports history obtained from the bug database. The researcher formulates the problem as a classification task.

The proponent solely uses the textual description of bug reports in this set. A bug report's textual description is divided into two sections: Title (summary) and description.

Because the description of bug reports contains many terms that are irrelevant to the functioning of bug reports, we only evaluate the summary as textual data.

Several efforts to using machine learning techniques for prediction or recommendation have discovered that prediction accuracy is dependent on the classifier used. In this study, we will use SVM (Support Vector Machines). Additionally, we will use inter-fold updates, wherein after validating the Validation Data Set (VDS) from fold n, the VDS is added to the Training Data Set (TDS) for validating fold n + 1.

4.2 Quick Design

At this stage the initial prototype is developed, where the very basic requirements are showcased, and user interfaces are provided. This stage would provide a high-level view of the application to the client.

1.2.1 Context Diagram

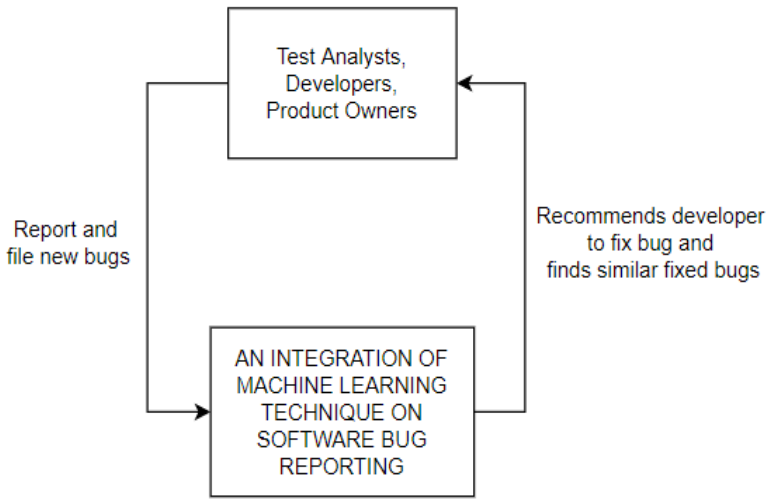


Figure 4.0.2 Context Diagram

1.2.2 Data Flow Diagram

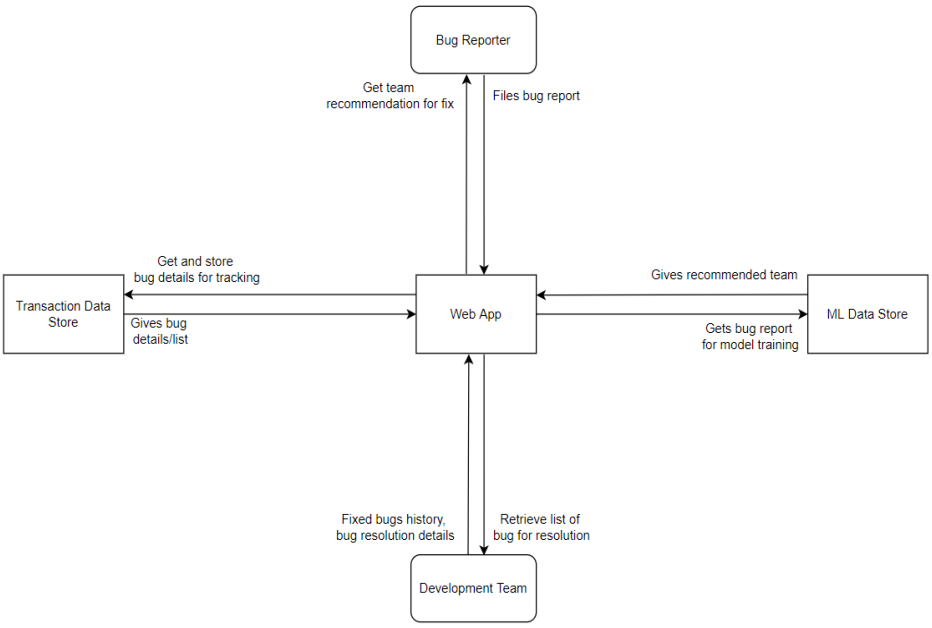


Figure 4.0.3 Data Flow Diagram

The researcher used the Data Flow Diagram, which is a dramatic representation of the information flow within a system that shows how information enters the system and leaves the system, what changes the information and where it is stored (Kendall, 2005).

1.2.3 Use Case Diagram

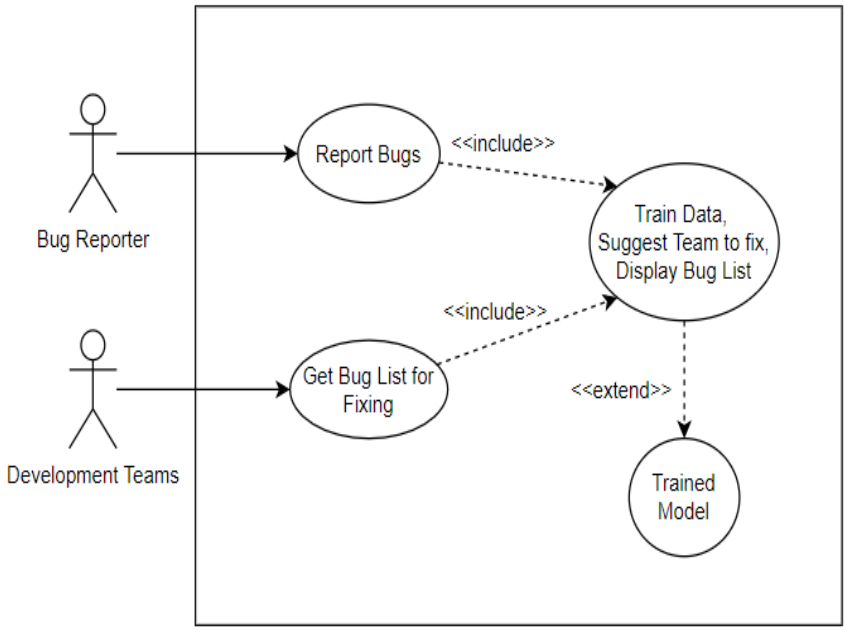


Figure 4.0.4 Use Case Diagram

4.2.4 System Flowcharts

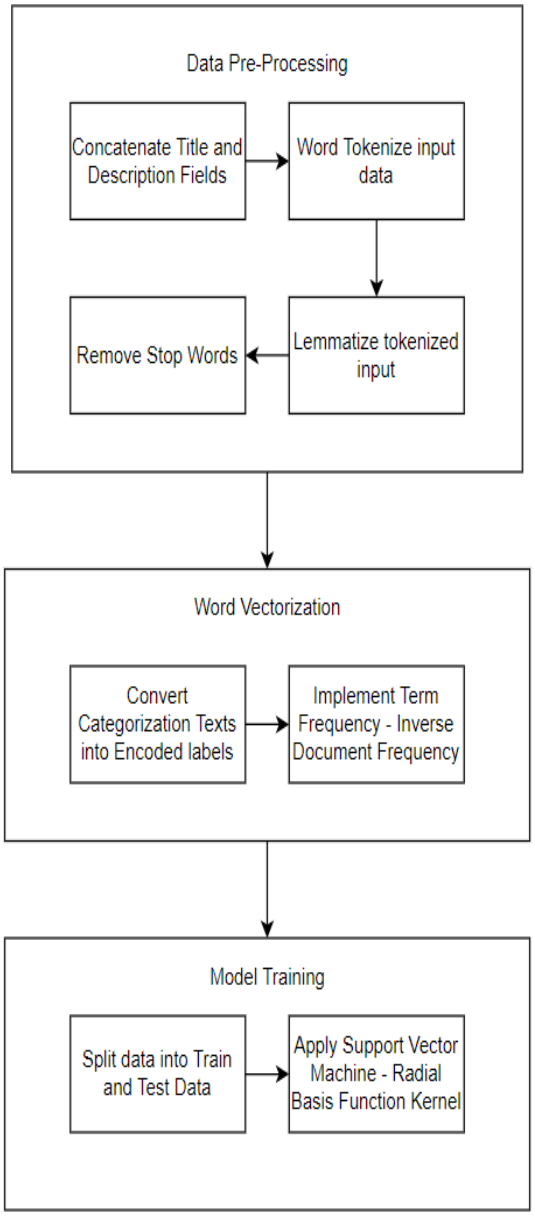


Figure 4.0.5 High Level Data Processing Flow

Bug report summaries are unstructured data that must be pre-processed before being converted to structured data. As a result, the proponent will use the usual text pre-processing method to convert the text data into a meaningful representation:

- Concatenation of Title and Description attributes
- Tokenization - split the summary of each bug report into tokens (terms)
- Lemmatization - combining a word's several inflected forms into a single item for analysis
- Removal of Stop Words

The next step is Vector space representation where each bug report is represented as a vector and each word in the bug report represents a label. We use the Term Frequency - Inverse Document Frequency (TF-IDF) of the word to get the value of each word feature.

SVM (Support Vector Machine; Boser et al., 1992), a supervised classification algorithm that finds a decision surface that maximally separates the classes of interest, will be used in this study. In SVM, the closest points to the surface on each side are as far away from the decision surface as possible. To express non-linear translations of the original input vectors, it uses kernels. It may now create very non-linear decision surfaces without having to explicitly specify the non-linear mappings.

Linear, polynomial, Gaussian Radial Basis Function (RBF), and Sigmoid are the four most frequent kernel functions. This study will use RBF functions since it has been shown to be the most successful in text classification.

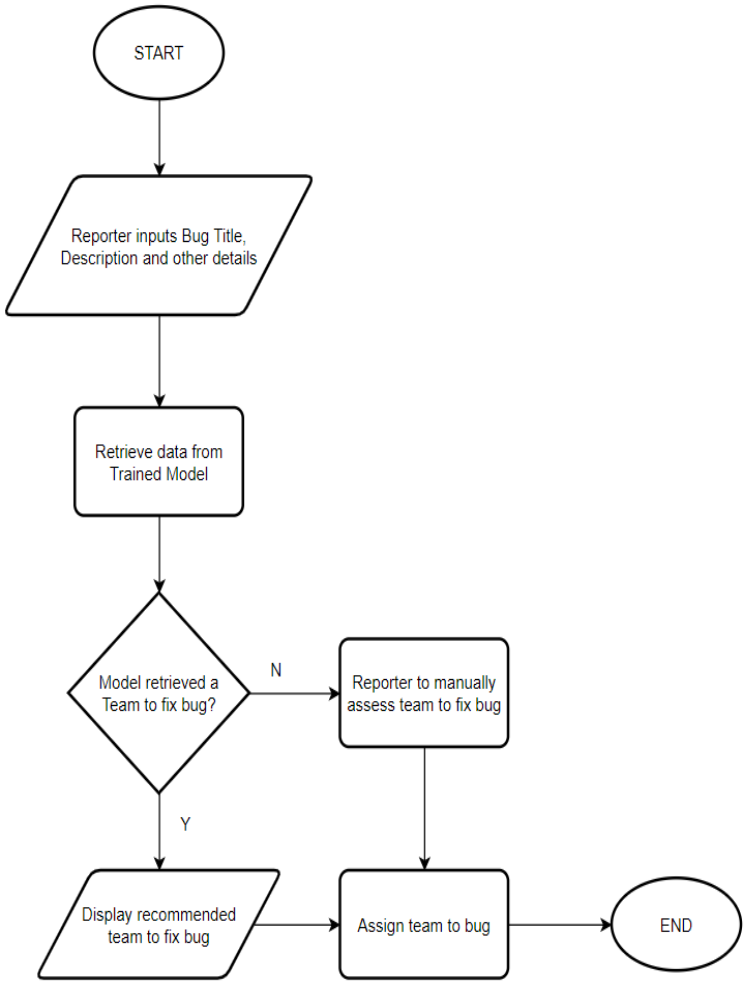


Figure 4.0.6 Bug Reporting Flowchart

Figure 4.6 illustrates the bug reporting process flow on which Bug Title, Description, and other attributes are manually inputted on the system. The system will process the inputs and retrieves the appropriate resolving team to resolve the bug using the trained model. The system will auto populate the resolving team and the bug reporter will then submit the bug report.

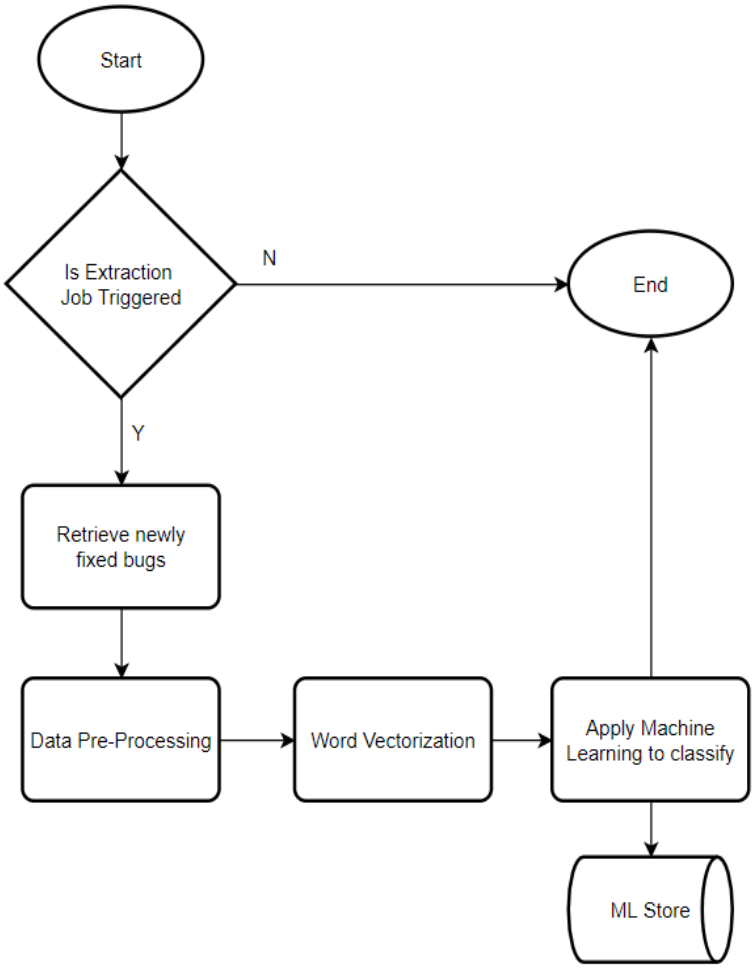


Figure 4.0.7 Data Extraction for Model Training

Figure 4.7 shows the process for extracting the data from new set of resolved bugs to be added on the ML data store.

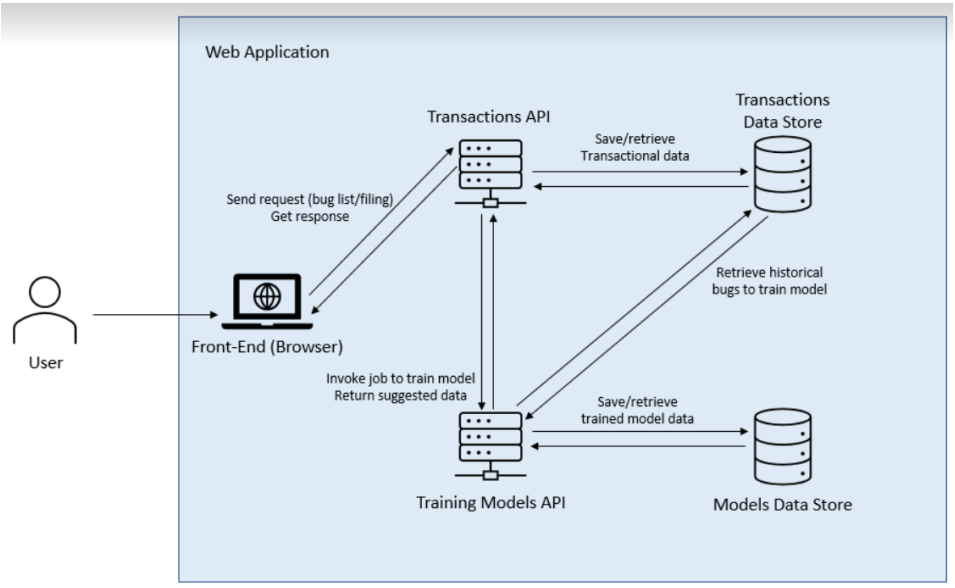


Figure 4.8 System Architecture

LIST OF REFERENCES

J. Xuan, H. Jiang, Z. Ren, J. Yan, And Z. Luo, “Automatic Bug Triage Using Semisupervised Text Classification,” In Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng., Jul. 2010, Pp. 209–214.

G. Jeong, S. Kim, And T. Zimmermann, “Improving Bug Triage With Tossing Graphs,” In Proc. Joint Meeting 12th Eur. Softw. Eng. Conf. 17th Acm Sigsoft Symp. Found. Softw. Eng., Aug. 2009, Pp. 111–120.

A. Lamkanfi, S. Demeyer, E. Giger, And B. Goethals, “Predicting The Severity Of A Reported Bug,” In Mining Software Repositories (Msr), 2010 7th Ieee Working Conference On. Ieee, 2010, Pp. 1–10.

Hamilton, 2021. Https://Www.Guru99.Com/Bug-Defect-Triage.Html

Nicholas Jalbert And Westley Weimer. Automated Duplicate Detection For Bug Tracking Systems. In International Conference On Dependable Systems & Network, Pages 52-61, 2008.

M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, And B. Scholkopf. “Support Vector Machines.” Intelligent Systems And Their Applications, Ieee, 13(4), Pp. 18–28, 1998.

T. Hastie, R. Tibshirani, And J. Friedman. The Elements Of Statistical Learning. New York: Springer, 2009.

Suthaharan S. (2016) Support Vector Machine. In: Machine Learning Models And Algorithms For Big Data Classification. Integrated Series In Information Systems, Vol 36. Springer, Boston, Ma.

B. Scholkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. R. Muller, G. Ratsch And A. J. Smola. “Input Space Versus Feature Space In Kernel-Based Methods,” Ieee Trans. On Neural Networks, Vol. 10, No. 5, Pp. 1000–1017, 1999.

Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, And Per Runeson. 2016. Automated Bug Assignment: Ensemble-Based Machine Learning In Large Scale Industrial Contexts. Empirical Software Engineering 21, 4 (2016), 1533–1578.

Kumari M., Singh V.B. (2020) An Improved Classifier Based On Entropy And Deep Learning For Bug Priority Prediction. In: Abraham A., Cherukuri A., Melin P., Gandhi N. (Eds) Intelligent Systems Design And Applications. Isda 2018 2018.

Advances In Intelligent Systems And Computing, Vol 940. Springer, Cham. Https://Doi.Org/10.1007/978-3-030-16657-1_53

Kanwal, J., Maqbool, O.: Managing Open Bug Repositories Through Bug Report Prioritization Using Svms. In: Proceedings Of International Conference On Open-Source Systems And Technologies, Lahore, Pakistan, Pp. 1–7 (2010)

Kanwal, J., Maqbool, O.: Bug Prioritization To Facilitate Bug Report Triage. J. Comput. Sci. Technol. 2(27), 397–412 (2012)

Tian, Y., Lo, D., Sun, C.: Drone: Predicting Priority Of Reported Bugs By Multi-Factor Analysis. In: Ieee International Conference On Software Maintenance, Pp. 200–209 (2013)

Menzies, T., Marcus, A.: Automated Severity Assessment Of Software Defect Reports. In: Ieee International Conference On Software Maintenance, Icsm 2008, Pp. 346–355. Ieee (2008)

Tian, Y., Lo, D., Xia, X., Sun, C.: Automated Prediction Of Bug Report Priority Using Multifactor Analysis. Empir. Softw. Eng. 5(20), 1354–1383 (2015)

Sharma, M., Bedi, P., Chaturvedi, K.K., Singh, V.B.: Predicting The Priority Of A Reported Bug Using Machine Learning Techniques And Cross Project Validation. In: Proceedings Of The 12th International Conference On Intelligent Systems Design And Applications (Isda), Kochi, India, Pp. 539–545 (2012)

Uddin, J., Ghazali, R., Deris, M.M., Naseem, R., Shah, H.: A Survey On Bug Prioritization. J. Artif. Intell. Rev. 2(47), 145–180 (2017)

Mani, S., Sankaran, A., Aralikatte, R.: Deep Triage: Exploring The Effectiveness Of Deep Learning For Bug Triaging (2018). Arxiv Preprint: Arxiv:1801.01275

Olsson, F. (2009). A Literature Survey Of Active Machine Learning In The Context Of Natural Language Processing (1st Ed.). Retrieved From Swedish Institute Of Computer Science Website: Http://Urn.Kb.Se/Resolve?Urn=Urn:Nbn:Se:Ri:Diva-23510

Chowdhury, G.G.: Natural Language Processing. Ann. Rev. Inf. Sci. Technol. 37(1), 51–89 (2003)

Liddy, E. (1998). Enhanced Text Retrieval Using Natural Language Processing. Bulletin Of The American Society For Znformation Science, 24(4), 1p16.

Feldman, S. (1999). Nlp Meets The Jabberwocky. Online, 23,62-72.