An Improved Machine Learning Approach to Text Mining for Automatic Bug Assignment

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE AWARD OF DEGREE
OF

MASTER OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted By
Bhawna Jain
2K17/SWE/18

Under the supervision of

DR. RUCHIKA MALHOTRA
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road,
Delhi-110042

JUNE, 2019

M. Tech (Software Engineering)

BHAWNA JAIN

2019

Department of Computer Science and Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Bhawna Jain, 2K17/SWE/18, student of Master of Technology in Software Engineering, hereby declare that the Major Project-II Dissertation titled "**An Improved Machine Learning Approach to Text Mining for Automatic Bug Assignment**" which is submitted by me to the Department of Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology (Software Engineering), is original and not copied from any source without proper citation. This work has not been previously formed the basis for the award of any Degree, Diploma, Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date:

Bhawna Jain

2K17/SWE/18

Department of Computer Science and Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the project Dissertation titled "**An Improved Machine Learning Approach to Text Mining for Automatic Bug Assignment**" which is submitted by **Bhawna Jain (2K17/SWE/18),** Department of Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the Degree of **Master of Technology** in **Software Engineering** in the **Department of Computer Science & Engineering**, is a record of the project work carried out by the student under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date:

DR. RUCHIKA MALHOTRA
(Project Supervisor)
Associate Professor
Discipline of Software Engineering
CSE Department
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad, Daulatpur, Bawana Road, Delhi- 110042

ACKNOWLEDGEMENT

First of all I would like to thank the almighty, who has always guided me to work on the right path of the life. My greater thanks are to my parents who bestowed ability and strength in me to complete this work.

I owe a profound gratitude to my project guide Dr. Ruchika Malhotra who has been a constant source of inspiration to me throughout the period of this project. It was her competent guidance, constant encouragement and critical evaluation that helped me to develop a new insight into my project. Her calm, collected and professionally exemplary style of handling situations not steered me through every problem, but also helped me to grow as a matured person.

I am also thankful to her for trusting my capabilities to develop this project under her guidance.

Date:

BHAWNA JAIN

M.Tech (SWE)-4th Sem

2K17/SWE/18

ABSTRACT

With the advent of digitization these days, software evolution has taken a deep rise in the computing industries. Large software development projects are being developed which, in turn, have a massive amount of data along with the bugs written in their software repositories. These bug reports are to be managed using a particular bug tracking system, and a diversified group of developers is involved in fixing those bugs. The number of bug report generated on a regular basis by frequently used and accessible systems, are generally high.

To triage the incoming reports manually consumes more amount of time. One aspect of bug triaging is to assign a particular report to the developer with the required proficiency. Automating the process of assigning the bug to the developer with suitable expertise can degrade the software evolution costs and effort.

Previous works have used various machine learning algorithms in order to automate the process of bug assignment but have incorporated limited tools which gave ineffective results with the increased size of the projects. To redress this scenario, this paper employs an improved hybrid machine learning approach, along with the bug tossing graphs to give a graph-based model that can predict the results more accurately. It also gives a comparative analysis of the machine learning algorithms that can be applied and give a solution as to which technique performs better.

The approach used will inevitably suggest developers who have the correct knowledge about dealing with a bug record, based on the identified component obtained from the short description of the bug report. The work begins with examining the impact generated by various machine learning features, including the attributes, classifiers, and the training history. Bug tossing graphs are being used along with the ranked list of developers in order to predict the accuracy of the bug assigned.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ML	Machine Learning
MNB	Multinomial Naïve Bayes
SVM	Support Vector Machine
KDD	Knowledge Discovery in Data
NER	Named Entity Recognition
RBF	Radial Basis Function

CHAPTER-1
INTRODUCTION

To create a bug is human, but to debug is exquisite. Maintenance of software, along with its evolution have high effort and cost associated with it. Some of the studies on maintaining software showed that the cost of maintaining large size projects vary from 50 to 90% or many times greater than the overall cost of the product [1,2]. If the process of fixing the bugs is efficient, then there can be a considerable reduction in the production cost and the effort. Thus, various bug tracking systems, such as Bugzilla, Mantis Bug Tracker, JIRA, FogBugz, and many more are being developed in order to facilitate bug fixing process. These trackers are utilized by outsized projects, including those of Eclipse, Mozilla, KDE, and Gnome. These systems are mainly used in open source initiatives, in which the participants of the group are distributed throughout the globe and may make a contribution in whichever way they like. These systems not only keep the record of the problem reported by the contributor, but also help the developers to coordinate among the peers whom they may see rarely, or never.

1.1 OVERVIEW

The lifecycle of Predictive Analysis involves many steps. First is having the clarity of the problem statement in hand and identification of the solution to the problem. Once we are clear with the above two points, we will be in a position to select our target variable. Second step one of the significant steps that is the identification of the dependent variables which will help to predict the target variable with the required accuracy. Accuracy to be achieved is also dependent upon the target variable selected. For example to predict the loan defaulters an accuracy of 90% is also considered as a good percentage, but if we talk about the prediction of some disease or medical case accuracy of 90% is not acceptable. In such cases, accuracy needs to be approaching 100%. Similarly, for various fields, the accuracy varies as per the stakes associated with the targeted variable.

Third steps involve data preparation, also known as pre-processing of the data set. This step is crucial in the successful building of the prediction model and almost takes 80% of the total time required to build a prediction model. Data can be gathered from various kinds of sources such as text documents, databases, files in different formats, images, and videos supporting all kinds of formatting, spreadsheets and etc. There we can be having both kind of data that is structured or unstructured data. After the data has been successfully gathered as per the problem statement in hand, next is to explore the data carefully. One should go through the data in hand and identify the type of data, types of variable, its range, identify if the variable is categorical or discrete in nature, the format of the variables, length and other such properties of the variables. Once the analyst has an understanding of the data in hand, he/she can go on with the next step of data validation.

Data validation, data can have various problems or errors that need to be dealt with before the data can be used as a training and test data set [2]. Problems like missing values, outliers, incorrect variable type, correlated variables, redundant variables, and many more need to be taken care of. There are a lot of techniques to handle the above cases. There are few modeling techniques which are self-capable to handle the above-mentioned cases, but others require separate effort to take care of the above situations. Dimension reduction technique also an essential technique for the reduction in the number of variables that can be used for building up the model. Once the data is validated thoroughly that is now the data understanding the problems with the data in hand are identified, the analyst can start with the data cleaning process.

Data cleaning step is about correcting the errors and problems identified in data exploration and validation steps. It is after this step, data is ready to be used to train the required prediction model. In this step, problems such as missing values, outliers are to be dealt with. There are no shortcuts that can be used in this step, each case to be dealt with separately keeping in mind the problem statement and type of data in hand.

Text mining is the part of Knowledge Discovery in Data (KDD), aiming at extracting all the valuable information from the unstructured text available in the form of email, text, or a data repository. Text mining process identifies valuable information available in the text, convert them into numeral indices so as to make the text accessible to multiple algorithms. Text mining maps

"text with numbers" to perform further analysis and predictions. The text mining process involves a series of steps. Initially, the text is pre-processed for cleaning unwanted information in the text, then splitting the text into white spaces resulting in the token generation, and assigning a work class to each token. Then in the next step, attributes are generated based on the words and number of occurrences of the word in the document using either of the two techniques, viz. bag of words and vector space method. The next step involves feature selection or variable section, in which a subset from the pool of important features is being selected and is used for model creation. Once the model is created, then these numeric vectors are passed to one of the existing machine learning algorithms for computing the results. This step is similar to the classic data mining techniques that are used in a structured database. The results are then analyzed and predicted for the implementation in the desired field. These steps are described in detail in section 3.1.

1.2 PROBLEM STATEMENT

The Bug tracking systems accept a plethora of bug reports at a time from various users. These bugs are, then allotted to the developer who can resolve the bug in the area of his expertise, given a restricted timeline. The method of allocating a bug to a developer to get it resolved is known as bug assignment. This process of assigning a bug is a cumbersome process, if done manually because of the extensive need of labor, time and is more prone to faults. Also, for an open source development project, it becomes difficult to keep track of the developers with the required expertise.

Recent bug tracking gives a privilege of adding additional comments along with the bugs in the bug report. This gives an edge, like a discussion forum, to the geographically separated developers in order to discuss the code design and implementation details. This also fills the niche for the project teammates, who want to give their contribution encouraging them to join in. Further, the add on modules during the software growth, may introduce new bugs in the software and rectifying those bugs daily, and manually assigning them to the developers, make the process more difficult. Jeong et al. in [3] explained that the time taken by an Eclipse project for assigning a bug, takes about forty to hundred days, varying from assigning the bug for the first time to a developer and then to reassign the bug if it can't be fixed by the first developer. This process of reassigning the

bug to another developer if the previous assignee is unable to fix it is known as "Bug Tossing". Studies indicate that the bugs from large size projects have been tossed at least once before getting resolved. This project aims at applying an improved and effective approach to automate the process of assigning the bug. In this, the process begins with validating a given bug as the real one, predict it as the new bug or the tossed one, and then look for the appropriate developer who can resolve the bug efficiently.

1.3 MOTIVATION

Text data has been growing dramatically recently, mostly because of the advance of technologies deployed on the web that would enable people to quickly generate text data. For example, every day, many web pages are being created. Emails are yet another kind of text data. And literature is also representing a large portion of text data. It is also especially imperative because of the high quality in the data. That is, we encode our knowledge about the word using text data represented by all the literature articles. A vast amount of knowledge is being represented by the text and data in these literature articles.

Now, these text data present some challenges for people. It's tough for anyone to digest all the text data quickly. So there's a need for tools to help people understand text data more efficiently. Here is also another interesting opportunity provided by such big text data, and that is it's possible to leverage the amount of text data to discover interesting patterns to turn text data into actionable knowledge that can be useful for decision making.

The software development process is emerging these days enormously. The challenges faced during the process varies significantly, majorly in the development cost and effort. The bugs so introduced during the maintenance phase of the development life cycle are assigned to the desired developer who can resolve the bug. Automating this process of assigning a bug to a developer can reduce the costs of the projects drastically. Keeping this idea into consideration, the Eclipse and Mozilla dataset of textual software repository are being used. On this dataset, a set of the classifier is being tested for performance of each of the classifier. Then, an improved approach of using the tossing graphs is also implemented, and the performance of the classifier is compared. The comparative analysis drawn for the same takes the machine learning classifier at one side and Ml

classifier with tossing graph on the other. The results are then represented graphically to have a clear picture of the efficiency of the technique used.

1.4 THESIS ORGANIZATION

The major aim of the thesis is to demonstrate the use of tossing graphs over a set of classifiers for automatic bug reassignment, to identify which classifier give best results and draw the comparative analysis for the same.

This chapter gives the brief introduction and overview of the research. Chapter 2 gives the literature review related with the research. Chapter 3 describes the process of text mining in detail. It also gives overview of the classifiers used in the research. The proposed methodology and implementation is described in chapter 4. Chapter 5 summarizes the results obtained and graphically shows the observations that are drawn from the study. Chapter 6 presents the conclusion and the future scope of the research project.

CHAPTER-2
LITERATURE REVIEW

There are various studies exist for predicting the process of bug assignment using supervised and unsupervised ML classifiers. The existing work on automating bug assignment process basically focusses on the prediction accuracy of the model using basic techniques only. The summary of these research carried over decades is discussed below.

2.1 RELATED WORK

Various researchers have contributed to bug triaging using multiple approaches varying from diversified information retrieval techniques to using tossing graphs and incremental method of learning for the classifiers. Semi-automating the process of assigning the bug was explained by Cubranic and Murphy in [4] using Naïve Bayes Approach to mine the text in bug reports. They classified the text by extracting keywords from the title and bug report description and used Id of the developer in order to train the classifier. The classifier predicted multiple developers for bug fixing when presented with a new bug report. This model gave an accuracy of up to 30% correct predictions. The major drawback with their work is the use of a small dataset of approximately eight months, ranging from January to September, 2002. This approach was improved by Anvik et al. in [5] by applying a wide range of filters, during data collection itself. They initially filtered the data based on the title of the bug report, labeled as "invalid", "wontfix" and "worksforme". Later the developers were removed who are not working on the project currently, or those who have not fixed bugs higher than nine in number. The classifier used by them includes Naïve Bayes, C45, and Support Vector Machines.

A varied approach was presented by Anvik and Gail [6], in their dissertation by building a recommender system that can automate the process of assigning the bugs. Addition to the previous work, he then used Nearest Neighbour, Conjunctive Rules and Expected Maximization as the unsupervised technique.

An incremental approach for assigning the bug automatically using probabilistic similarity in the text was proposed by Canfora et al. [7]. They tracked the reports of developers and individual modules that might have changed during the bug fixation process. The results by them showed achievement of 50% accuracy [8].

A study on classification and prioritization of software faults was done by Podgurski et al. in [9] in which the employed some machine learning algorithms for classifying the bug reports. Similar to this, Lucca et al. in [10] used classifier for classifying maintenance request for bug classification, but not for bug assignment. The results showed the accuracy of up to 84% for classifying the bugs. A set of bug assignment techniques is being implemented by Lin et al. in [11] on the SoftPM proprietary project containing a collection of 2,576 bug reports. They used the ID of the module as a feature for training the module classifier and achieved an accuracy of 77.64%, which is significant compared to one where this ID is not used.

A content analysis based approach by using the source code vocabulary was proposed by Matter et al. in [12] that can model the developer's expertise. The new report is assigned to a developer based on examining the content in the source code and then based on the extracted vocabulary words, and it is assigned to the desired developer.

An incremental model-based approach by Bettenburg et al. in [13] showed that the prediction accuracy of the classifiers could be increased by incorporating the duplicate bug reports in the training dataset for the classifier. For repetitively increasing the training dataset, the folding method is being used that gives an accuracy of about 56%. Table 1 describes chronological literature on the techniques used for bug triaging along with their interpreted results.

Machine Learning techniques have already been applied in order to mine the software repositories for detecting the debugging component and for supporting code writing in the software development. For example, Source version histories were mined by Zimmermann et al. in [22] for finding the association rules. The rules were then used in order to predict the files that change collectively in terms of program elements like variable and functions. The related work is summarized in table 2.1.

Table 2.1: Literature work on ML Techniques used in Bug Triaging.

Paper	Attributes	Extracted Features	Technique Used	Dataset Used	Results
Bhattacharya et al., 2010 [14]	title, description, keywords, product, component, last developer activity	tf-idf + bag of-word	Naive Bayes classifier + Tossing graph	Mozilla# 549,962 Eclipse# 306,297	accuracy 77.87% accuracy 77.43%
Ahmed et al., 2011 [15]	description, title	Feature terms	A fuzzy-set feature for each word	Eclipse# 69829	accuracy 68%
Anvik et al., 2011 [6]	description, title	normalized tf	Conjunctive Rules, Nearest neighbour, C4.5, EM, SVM, Naïve Bayes	Firefox# 7,596 Eclipse# 7,233	prec. 51%, recall 24% prec. 60%, recall 3%
Xuan et al., 2012 [16]	title, description	tf-idf, developer prioritizatio n	Naive Bayes, SVM	Eclipse# 49,762 Mozilla# 30,609	accuracy 53.10% accuracy 56.98%
Shokripour et al. 2013 [17]	title, description, detailed source code info	weighted unigram noun terms	Bug location prediction + developer expertise	JDT-Debug# 85 Firefox# 80	accuracy 89.41% accuracy 59.76%
Wang et al., 2014 [18]	title, description	tf	Active developer cache	Eclipse# 17,937 Mozilla# 69,195	accuracy 84.45% accuracy 55.56%
Xuan et al., 2015 [19]	description, title	tf	feature selection with Naive Bayes Classification	Eclipse# 50,000 Mozilla# 75,000	accuracy 60.40% accuracy 46.46%
Badashian et al., 2015 [21]	title, description, keyword, stackoverflow and github tags	Keywords from bug and tags	Social expertise with matching keywords	20 GitHub projects, 7144 bug reports	accuracy 89.43%
Jonsson et al., 2016 [20]	title, description	tf-idf	Stacked Generalization of a classifier ensemble	Industry# 35,266	accuracy 89%
Mani et al. , 2018 [23]	bug summary description	Bag of words	Deep bidirectional recurrent neural network with attention	Chromium, Core and Firefox	accuracy 34.47%

CHAPTER-3

TEXT MINING AND BUG TOSSING GRAPHS

In this era of computing, various digital data acquisition techniques have led to the generation of huge amount of unorganized data. These data from software repositories can is used to assign and resolve the bug to a developer. The preliminaries and definitions required for implementing this automating work is described in detail under this module. The terms explained include the process of text mining, and its underlying concepts in detail with example. Then the classification algorithms that are used in this research are defined, which include Naïve Bayes Classifier, Multinomial Naïve Bayes, Bayesian network, C4.5, linear, and RBF Support vector machines. The basics of bug tossing graph is also explained in detail.

3.1 TEXT MINING

Text mining is a process of discovering knowledge from databases (KDD) to extract useful information and identifying previously unknown facts automatically. It is also used for identifying various patterns from different text sources. It reduces information overload by focusing on a larger unit of text and defining the relationship between inter-document and intra-document information. Most of this data contains unorganized information in the form of web pages, contact center notes and transcripts, surveys, scientific literature books, legal documents, emails, collaborative system repositories, etc., which computers cannot process directly. The major problem that arises is the expression of a concept with words, but no such mapping exists. So it is essential to develop techniques and algorithms to pre-process this textual data in order to identify patterns and to extract knowledge. Text mining is the process of analyzing texts using machines, to find exciting regularities in the text and discover previously unknown facts.

The process of text mining in some aspects is similar to data mining, except the fact that data mining cannot be applied to any unstructured information. So text data needs to be pre-processed to reach a structured format, on which further data mining techniques can be applied. Data mining tools cannot handle unstructured data, so specialized tools and powerful algorithms are required

for processing and analyzing the text. Text mining process includes a series of steps as summarized in figure 3.1.

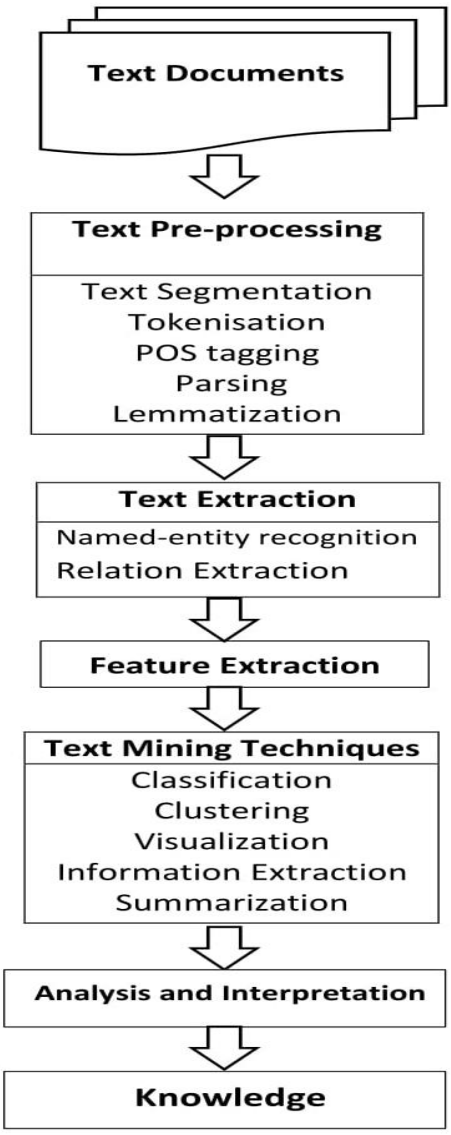

Fig. 3.1 Text Mining process

3.1.1 COLLECTION OF DOCUMENTS

The process starts with collecting the dataset including the set of documents on which text mining techniques can be applied. This dataset is collected from various resources depending upon the

type of data for which knowledge is to be extracted. The sources for the mining includes software project repositories, business documents, social media, satellite images, pdf files, website contents etc.

3.1.2 TEXT PRE-PROCESSING

Traditional databases cannot store data that is available in unstructured format as they do not follow the traditional row-column format. This makes the process of pre-processing a crucial step. Pre-processing is used to detect and remove the anomalies present in the text. The main aim is to get the real essence of the available text. It involves a series of steps:

a) Text Segmentation: Textual content segmentation is the technique of identifying the boundaries among words, terms, or some other linguistic significant devices, consisting of sentences or topics. It is able to make humans read the textual content and assist computers in performing some artificial processing. It additionally eliminates needless or undesirable statistics which includes ads on the web pages, and deals with binary layout transformed texts, tables, figures and formulas.

b) Tokenization: A token is an instance of phrase or term happening in a file. In tokenization, the input textual content is robotically divided into units known as tokens. It is executed with the aid of splitting the textual content on white areas and at punctuation marks that don't belong to abbreviations diagnosed inside the previous step. Tokenization is explained in table 3.1.

Table 3.1 Tokenization Example

A swimmer likes swimming, thus he swims.						
↓						
a	swimmer	likes	swimming	thus	he	swims

c) Part-of-speech tagging: It is also called POS tagging. Part-of-speech includes nouns, verbs, adverbs, adjectives, pronouns and conjunctions. POS tagging is the method of assigning one part of speech to a phrase. They’re useful as they provide records about a word and its friends.

d) Parsing: It is used to structure the textual content the usage of parse trees. The parse tree to be generated gets the input from the POS taggers and form the tree on the premise of speech regulations.

e) Lemmatization: Lemmatization is the mathematics system of determining a lemma for a given phrase. The principle goal is to identify proper reduction of words to dictionary headword format. This process is explain in table 3.2. It is further used to derive relative forms of a phrase to a familiar base. It gets rid off inflectional endings only and give results only of the bottom or dictionary form of a word, which is referred to as the lemma by the use of a vocabulary and doing morphological evaluation of phrases. This technique may additionally contain complex tasks including identification of context and figuring out the part of speech of a phrase in a sentence, so it may be a tough mission to enforce a lemmatizer for a new language.

Table 3.2 Lemmatization Example

A swimmer likes swimming, thus he swims.								
↓								
A	swimmer	like	swimming	,	thus	he	swim	.

3.1.3 TEXT EXTRACTION

It is the way toward changing the information into disjunctive arrangement of features. It starts with set of measured data and after that makes a progression of inferred values that are proposed to be informative. It goes for the extraction of semantic things from the reports to give a representative of their content. Disjunctive vocabulary things found in a report are relegated to the various classes by estimating the significance of those things to the archive content.

Text Content extraction can be implemented by using Named Entity Recognition (NER) and Relation Extraction. NER aims at identifying the phrases that alludes the name, place, people, location, dates and associations existing within a document. These elements are then recognised in order to match them to the predefined classes. For instance, the names of human being, will be categorised in one class, similar to which the financial qualities, time stamps, dates, etc. Relation

extraction includes the recognizable proof of connections among those named substances. For this, bag of words and support vector machine procedures can be utilized.

3.1.4 FEATURE EXTRACTION

It is a technique of choosing a subset of significant features for use in model creation. This stage principally performs expelling features which are repetitive or unimportant. The first element space of the information is mapped onto another new feature space. Feature Selection is the subset of progressively broad field of feature extraction. It is otherwise called variable selection.

Feature Vectors: Set of feature vectors available in the training dataset, is responsible for accuracy of the classifier. Keywords that form feature vectors are being extracted from software repositories in the form of the titles and the summaries so available. The extraction process is in such a way that they correspond to a specific class of bug. For instance, if the repository contain the words like “display”, “image”, “icon”, then it depicts that the report description is related to the design and layout and will correspond to that class.

3.2 BUG CLASSIFICATION USING MACHINE LEARNING

Classification is a directed artificial intelligence strategy for getting a general pattern from an informational data set. The idea behind the classification technique is to determine the category based on some rules that we design carefully to reflect the domain knowledge about the category prediction problem. The categories must be very well defined and this allows the person to clearly decide the category based on some clear rules.

Bug classification technique have one to one mapping between the developer and the class to which bug report is being assigned. The mapping between the developers and the list of bugs they have resolved in the past, make it a *supervised learning* algorithm.

3.3 CLASSIFICATION ALGORITHM USED IN TEXT MINING

The set of classification algorithms that exists for text mining varies from supervised, unsupervised and deep learning algorithms. They are applicable in diversified domains though, thus a few of them are used for implementation in this project as described below:

a) **Naïve Bayes Classifier:** Naïve Bayes classifier is a model used for differentiating the objects "independently" based on given features. It is a probabilistic model based on the principles of Bayes Theorem. Here, every group of feature that is going to be classified have independence between them. It uses conditional probability to identify the probability with which a class and an instance of it is related.

$$\mathrm{P}(\mathrm{class}|observation) = [P(observation|class) \times P(class)] \div [P(observation)] \quad \text{Eq. (1)}$$

Naïve Bayes classifier is used in bug categorization as, for instance, if a word occurs in a description of the bug report more frequently that developer D1 resolves, than in the descriptions of reports that are resolved by developer D2, then the developer D1 will be selected by this classifier as the potential developer who could resolve any new bug report containing this word concurrently.

b) Bayesian Network: Bayesian network is a model based on probabilistic features, representing the group of random variables. It represents their associated probabilities using Directed acyclic graphs (DAG) consisting of nodes and edges. Variable is represented using nodes of a graph, and the relation between the pair of variables is depicted by edges. The probability of variable given the probability of the parent is stored in the conditional probability table.

b) **C4.5:** C4.5 is a greedy version of the decision tree supervised algorithm which builds a tree using the instance attribute present in the training dataset. The results are predicted using the directed path from the root node to the leaf using the attribute value of the new instance variable. The internal nodes are broken in order to maximize the information gain value computed for each decision to be taken at that node. The leaf nodes in the tree describe the final class for which there cannot be any further distinctions

c) Multinomial Naïve Bayes: Multinomial Naive Bayes is an extended version of Naive Bayes machine learning algorithm that is used for mining the textual documents. It differs with the Naïve Bayes algorithm in the sense that, Naïve Bayes only identifies whether a particular word is present or not in the document. On the other hand, Multinomial Naïve Bayes calculates the number of words present in the document and then calculate the Tf-IDF values for the same.

d) Support Vector Machine: Support vector machines is a discriminative supervised machine learning algorithm that categorizes the sample to the respective classes by identifying a separating hyperplane. In other words, it maximizes the distance between the points and the hyperplane, by placing the points that are closer to the surface, far from the decision plane. The non-linear mapping for the input vectors so formed from one-dimensional space to others is represented using kernels. These kernels help in forming decision surfaces that are non-linear without using any form of explicit notations for such mapping. The kernel functions that are used in SVM are of four types: Sigmoid, Polynomial, Linear, and Gaussian Radial Basis Function.

3.2 BASICS OF TOSSING GRAPHS

Tossing a bug is a process of "reassigning" the bug to a new developer, if the initial developer, to whom the bug is assigned for the very first time, is unable to solve it. Thus, the tossing is bug takes place from one to another, until it gets resolved by a developer. This creates a path of tossing, starting from the initial developer to all the developers till the final developer who finally resolves it. This path is termed as tossing paths, and are depicted using tossing graphs. These graphs are weighted directed edge graphs in which each of the developers is depicted by the node, a directed edge E from a developer N_i to developer N_j defines that the bug which is assigned to the developer N_i is being tossed and now fixed by the developer N_j. The weight assigned on each edge of the graph defines the probability with which the bug is tossed between these developers, taking bug tossing history into consideration. This can be explained using an example. Here, based on the tossing history of the developer, the tossing path for each developer is described in figure 3.2.

A → E → D → C
A → B → C → D
A → B → E → D
B → E → D → F
C → E → A → D

Fig. 3.2 Tossing path history of the developer

Here, suppose a bug is tossed by a developer A to E, which is further tossed to D and then to C. Similarly, another bug from C is being tossed to E, then to A and D respectively. This is explained in table 3.3. Based on the number of tosses by each developer and the developer who actually fixed the bug, the tossing path for each developer can be made. The associated probabilities corresponding to each path of the graph is the ratio of the number of tosses made by the respective developer to the total number of tosses. This can be explained in figure 3.3.

Table 3.3 Tossing probabilities of each developer

Developer Tossing the Bug	Total number of tosses	Developer fixing the bug					
		C		D		F	
		No.	Prob.	No.	Prob.	No.	Prob.
A	4	1	0.25	3	0.75	0	0
B	3	0	0	2	0.67	1	0.33
C	2	-	-	2	1.00	0	0
D	2	1	1	-	-	1	0.5
E	4	1	1	2	0.5	1	0.25

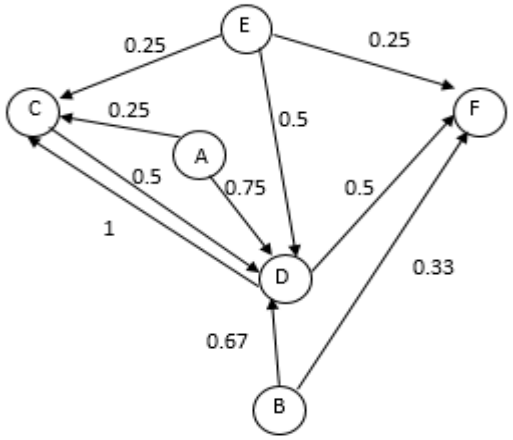

Fig. 3.3 Tossing graph with required probabilities

Here, the number of tosses done by A is four in number, out of which the developer C has fixed the bug one time. So the associated edge will be from A to C, and the associated probability with it is 0.25. In the similar manner, developer D has resolved the bug three out of four times, so there will be an associated edge from A to D. The weight of the edge will be its associated probability that is three out of four times, so 0.75 is the weight on the edge. In this way, the tossing edge graph is constructed.

CHAPTER-4

PROPOSED METHODOLOGY

In this work, the methodology for implementing the bug assigning process is being explained using the module diagram. The architecture of the dataset that is used is mapped to the classification model for clear understanding of the work. The dataset is explored over the classification algorithms and the steps are explained in detail.

4.1 MODULE FOR BUG ASSIGNMENT

A model approach for assigning the bug to the appropriate developer accurately and effectively begins with pre-processing the text data set of bug reports.

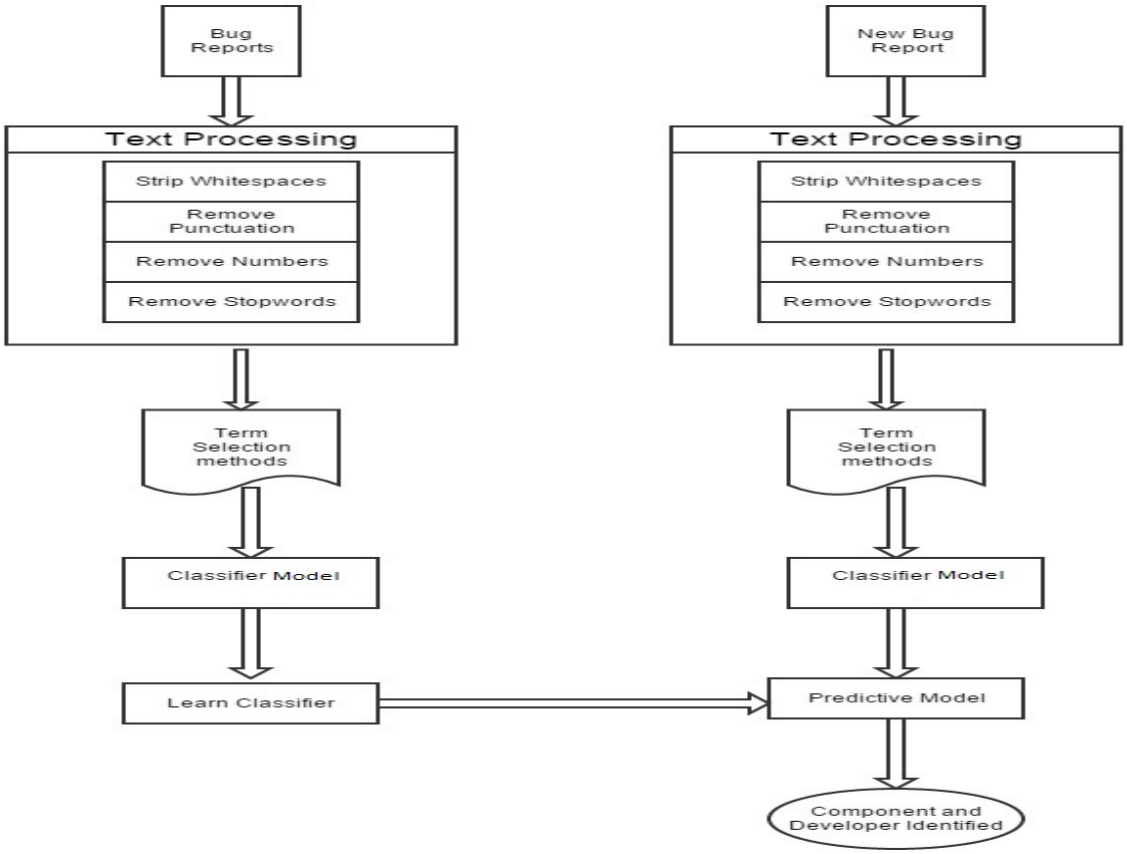

Fig. 4.1 Module diagram for bug assignment

Figure 4.1 describes the module diagram for the process. It begins with preprocessing of the description present in the reports. The sparseness present in these reports are then reduced using the term selection methods. The hybrid classifier then learns from the refined bug report and it is used to accurately identify the developer. When the new bug report is given, the system predicts the accurate developer based on the learned classifier.

4.2 PROPOSED APPROACH

Input: A bug report in natural language text submitted by the reporter briefing the problem. **Output:** The component in which the bug may potentially be, and the developer or list of developers to whom it can be assigned to.

When the user finds a bug, he/she reports the bug through a bug tracker used by the Software. Since the description of the bug submitted by the user is a natural language text, Natural Language Processing is used to extract useful keywords from the bug report that would provide information about the bug that the user has encountered. The processing involves stop-word removal and stemming from extracting useful keywords from the description of the bug report. These extracted keywords are used to identify the most probable defective component based on the dependencies that are previously learned. Then based on the defective Component and Tossing History of the developers, a list of Developers will be informed of this 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. A dependency structure is formed over time for supervised learning from the fixed bugs. The architecture of proposed methodology is depicted in figure 3.

From the available repositories, the XML files of only three sets are used for the implementation. The architecture includes the report Id, description, type of product, initial component and final component. From this design, part of it is used to train the classifier model and the remaining is used for the tossing graph model. Finally on the onset of new bug reports, they are used for predicting the results.

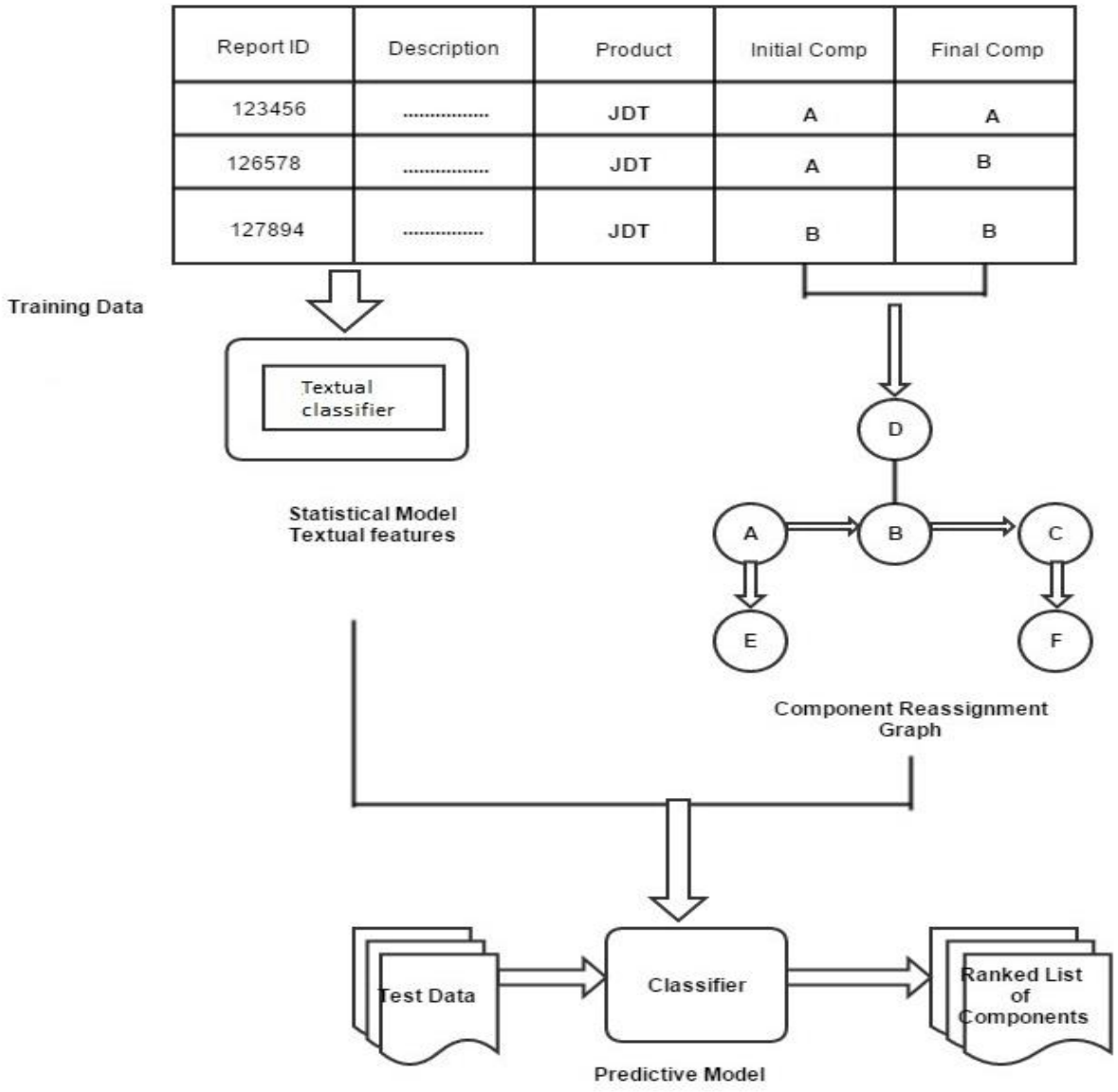

Fig. 4.2 Architecture of the proposed approach

Here the dataset from Eclipse and Mozilla repositories is taken, consisting of the Report ID along with the textual description and other attributes. This description is passed to the textual classifier for training the model. The tossing graph is drawn from the initial and final component describing the tossing history of the respective bug. This tossing graph along with trained textual classifier is now passed to the predictive model. The input to the model is the new set of repositories which give the desired rank of developers.

4.3 DATA SET

Dataset is a collection of fixed bug reports gathered from a open source software bug tracker tool containing necessary information about the components, developers and re-assignments. This is a categorized, classified, and semi-structured data. A bug report, generally a natural language text, submitted by the user is stored in the XML format by the bug tracker tool. Information contained in the dataset:

- Severity: The severity denotes how early this bug needs be fixed.
- Assigned to: The identifier of the developer to whom the bug was assigned to. The sample report from the assignedto.xml is shown in figure 4.3.

Fig. 4.3. Sample dataset from "assignedto.xml"

- Product: It defines to which software application is this bug related.
- Bug status: The status of the bug at every update. These include NEW, ASSIGNED, RESOLVED, VERIFIED, and REOPENED.

- Short Description: Contains a natural language text embedded by the user. The sample report from shortdesc.xml is shown in figure 4.4.

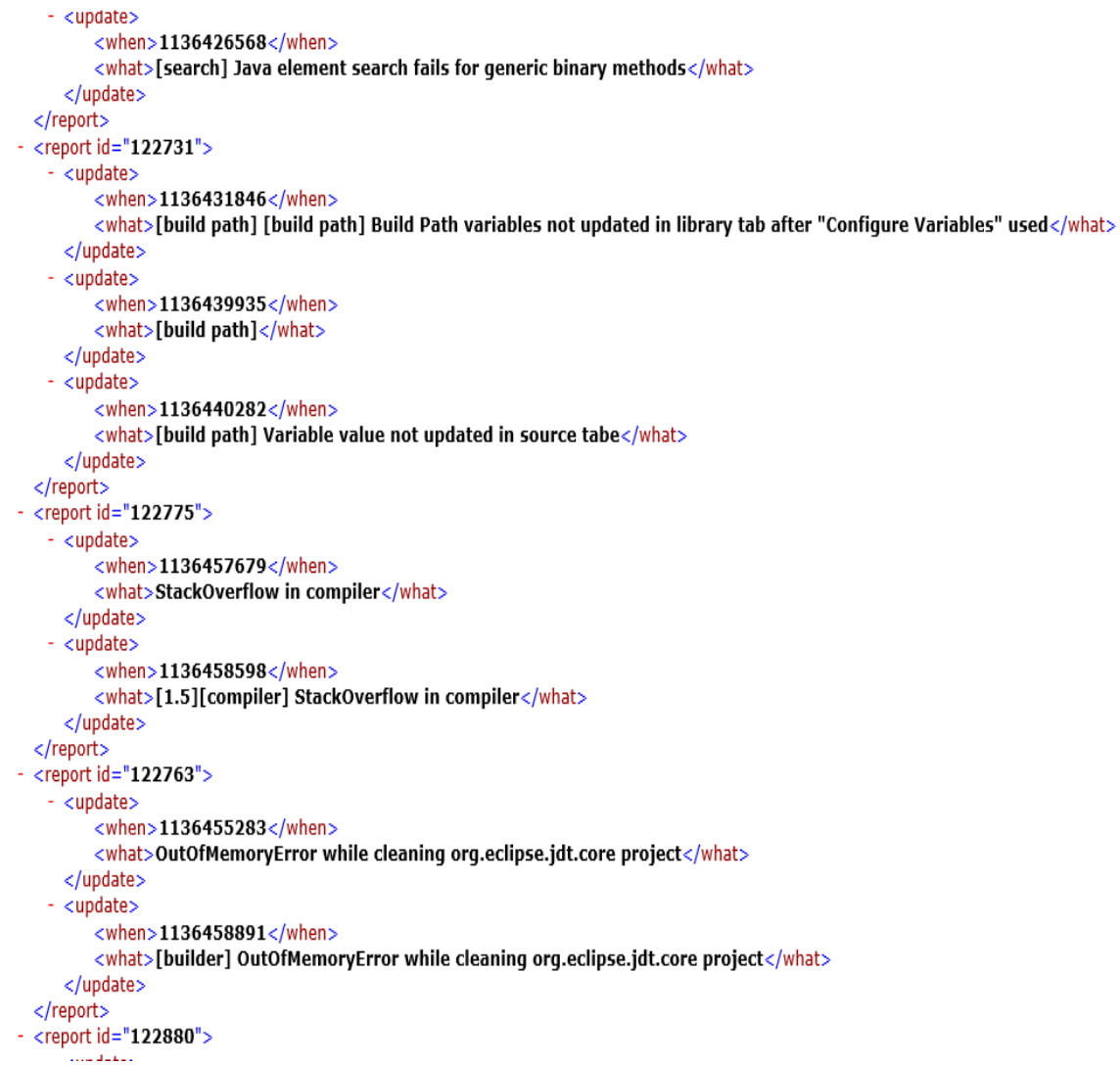

Fig. 4.4. Sample dataset from "short desc.xml"

- Resolution: Tagging the bug report for maintenance. These tags varies from FIXED, REMIND, INVALID, and WORKSFORME.
- Component: The subsystem relevant to the product for the reported bug. The sample report from the component.xml is shown in figure 4.5.

With these information the dependencies between the components, developers, and reassignment can be formed. First the dataset in XML format was used but it has only around 10000 reports. To obtain higher efficiency, dataset available in JSON format was used having around 1,60,000 reports in a well-structured manner.

Fig. 4.5. Sample dataset from "component.xml"

Training data set in JSON format compares the report-id and the update (”when”) of each report-id in the respective files and merges the “what” content present in short description (to get the bug report), 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 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 incoming bug reports. Another 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.

```
3319
New Plugin Project Wizard doesn't support nested source folders , DJ_Houghton@oti.com , Core

3320
Internal error using problem underlining , Erich_Gamma@oti.com , UI

3321
incorrect outline (i see only imports) , Erich_Gamma@oti.com , UI

3322
NPE in JavadocTreeWizardPage trying to bring up context menu , Erich_Gamma@oti.com , UI

3323
Setting a project's launch configuration type causes NPE , Darin_Wright@oti.com , Debug

3324
OpenOnSelection unable to perform in single-type import , Philippe_Mulet@oti.com , Core

3325
```

Fig. 4.6. Output after parsing the JSON file

The dataset is initially formatted to get it into a proper format. For that, the json format of the files are being used. The data is extracted to find the report id, component and the respective description in a document. The parsed output is depicted in figure 4.6.

This file is now text pre-processed for removing the stop words. Then the stemming process is being done using PorterStemmer. In this stemming the text words are stemmed to their respective stemmed trees. For instance, ‘cutting’ is stemmed to ‘cut’, removing the suffices. The output file after stemming is depicted in figure 4.7.

Fig. 4.7. Output file after Stemming

To input this file to a classifier, the file has to be converted into a feature vector format. The feature vectors form the basis for training the classifier. These feature vector pairs are depicted in figure 4.8. These feature vectors form the basis for classification of developers. The type of bug for which a developer has the required expertise is seen using these feature vectors. These are the transformation of the contextual features into the mapped binary values. These can be used by the classification algorithms along with term frequency for the categorization of the text.

Feature extraction form the crucial step in mining the text from the Eclipse repository. The number of terms in the description of the document, with respect to the number of terms in the entire repository is analyzed using term frequency-inverse document frequency. On the bass of this score, the numerical values are assigned to the type of term present in the document, and in the overall domain of the documents. This is coupled with the bug tossing graph to predict the range of developers that can finally resolve the bug based on their previous tossing histories.

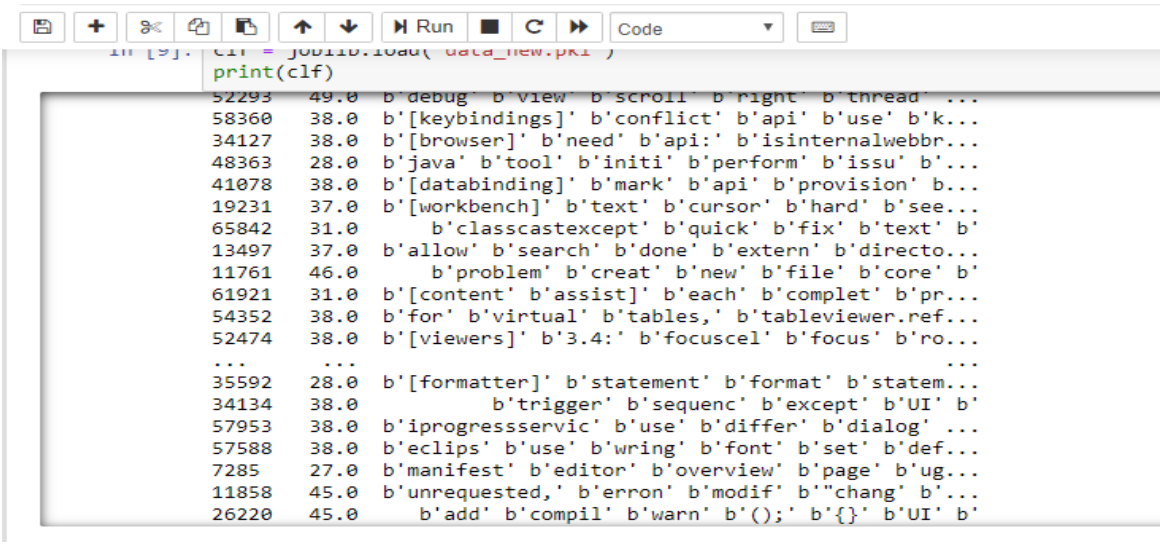

Fig. 4.8. Output file for feature vector pair

This file is then used to train the classifier and dump the result in pickle. Input from user is obtained to predict the accurate developer to whom the bug will be accurately fixed. Now the tossing history of the developer is being obtained from the dataset specifying the report id along with the number of developers to which the bug is being tossed. This result is depicted in figure 4.9.

Fig. 4.9. Tossing history of the developers

The accuracy of the classifier is now being predicted in two ways: (a) using the classifier alone and, (b) using the classifier along with tossing history. The final results are described in the next section.

CHAPTER-5
RESULTS AND ANALYSIS

In order to perform the stated experiment, Mozilla and Eclipse datasets are being used. The algorithm is analyzed over both the applications. These datasets were found to be of high quality, thus helped in reduction of noise while training the classifiers during the experiment. The accuracy of the predictor depends on the rank of the developer who fixed the bug. If the developer predicted by the classifier gets matched to the actual developer who actually fixed the bug, then the count for Top 1 developer is incremented. Similarly, if the second developer from the list of prediction gets matched to the actual developer fixing the bug, then the count for Top 2 developer gets incremented. For instance, given a set of 100 bugs in the validation data set, if 20 bugs from the actual developer are being resolved by the first developer of the prediction list, then the accuracy for Top 1 turns out to be 20%. In a similar manner, if the actual developer matches the second developer from the prediction list for 60 of those bugs, then the accuracy for Top 2 is 60%. The accuracy of the classifier is the average of the accuracy from the top 1 to top 5 accuracies.

To demonstrate the benefit of using a tossing graph with the classifier, the prediction accuracy is initially tested using the classifier alone. Table 5.1 depicts the prediction accuracy for Eclipse dataset when used with ML classifier only. It was found that the approach used gives an increased accuracy of about 8.9% to that of the previous approaches used..

Table 5.1. Bug Assignment prediction accuracy for Eclipse using ML only

Classifier for Eclipse	**Accuracy**
Naïve Bayes	67.21
Multinomial Naïve Bayes	70.58
Bayesian Network	68.91
C4.5	65.98
Linear SVM	42.92
RBF SVM	47.77

The comparative analysis of the results are being shown in figure 5.1. It depicts that Naïve Bayes classifier and Bayesian networks outperforms the other machine learning algorithms.

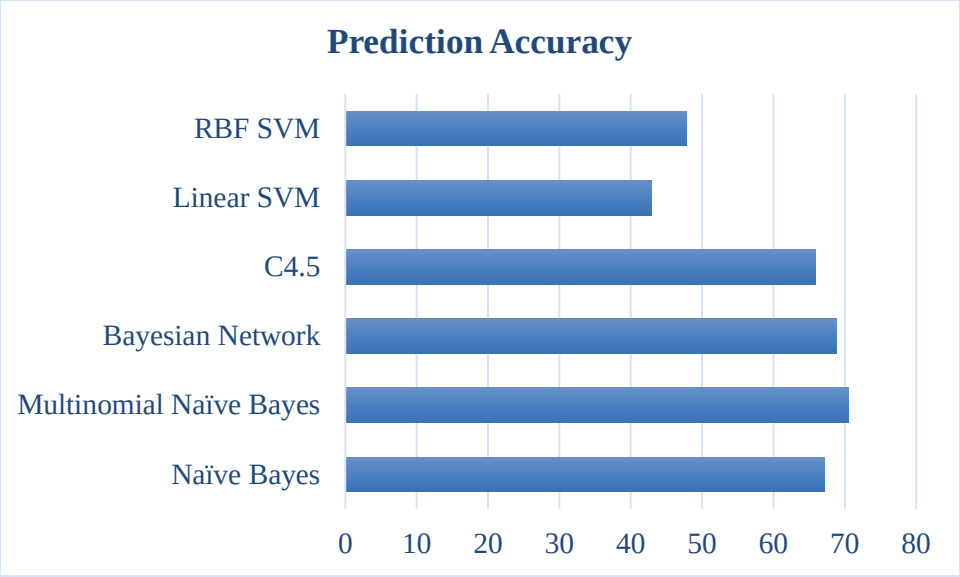

Fig. 5.1. Comparative analysis of ML Classifier Accuracy for Eclipse

Figure 5.1 shows that the prediction accuracies of classifier using the machine learning techniques only. It shows that Naïve Bayes give accuracy of 67.21%, MNB give 70.58%, Bayesian netwowk give 68.91%, C4.5 give 65.98%, but linear and RBF SVM show 42.92 and 47.77%.

Now the datasets are tested for the results over the classifier along with the use of tossing graphs. The results in table 5.2 shows the improvement in the prediction accuracy prior to the results where these graphs are not used. Figure 5.2 depicts the comparative analysis of the results by the use of ML along with classifier over Eclipse dataset.

Table 5.2. Bug Assignment prediction accuracy for Eclipse using ML and tossing graph

ML and Tossing for Eclipse	**Accuracy**
Naïve Bayes	75.43
Multinomial Naïve Bayes	76.96
Bayesian Network	74.89
C4.5	71.37
Linear SVM	51.03
RBF SVM	55.93

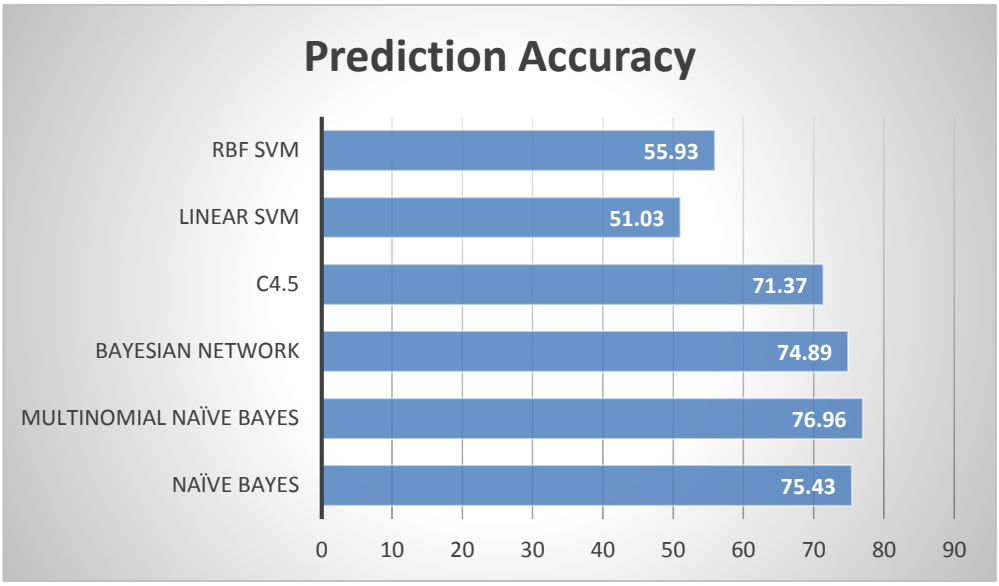

Fig. 5.2. Comparative analysis of ML and Tossing Accuracy for Eclipse

Figure 5.2 shows that the prediction accuracies of classifier using the machine learning techniques only. It shows that Naïve Bayes give accuracy of 75.43%, MNB give 76.96%, Bayesian network give 74.89%, C4.5 give 71.37%, but linear and RBF SVM show 51.03 and 55.93%. Hence MNB outperforms all the techniques.

Similar work is also being performed over Mozilla dataset. The results for Mozilla dataset is shown in table 5.3 and the comparative analysis for the same by using only the classifier is shown in figure 5.3.

Table 5.3. Bug Assignment prediction accuracy for Mozilla using ML only

Classifier For Mozilla	**Accuracy**
Naïve Bayes	65.66
Multinomial Naïve Bayes	68.55
Bayesian Network	62.19
C4.5	59.18
Linear SVM	51.17
RBF SVM	62.49

The results in the table shows that the accuracy of the classifier is comparatively low for some of the classifier in Eclipse as compared to those in Mozilla.

Figure 5.3 shows that the prediction accuracies of classifier using the machine learning techniques only. It shows that Naïve Bayes give accuracy of 66.66%, MNB give 68.55%, Bayesian network give 62.19%, C4.5 give 59.18%, but linear and RBF SVM show 51.17 and 62.49%.

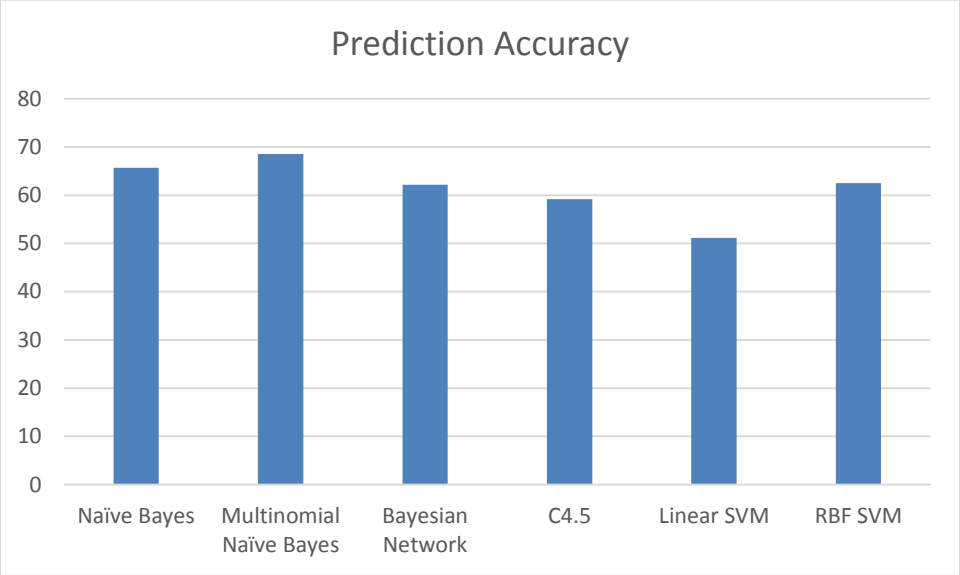

Fig. 5.3. Comparative analysis of ML Classifier Accuracy for Mozilla

The results for Mozilla dataset after using tossing graphs are comparatively higher than those of the results in Eclipse. The results are shown in table 5.4.

Table 5.4. Bug Assignment prediction accuracy for Mozilla using ML and tossing graph

Classifier	**Mozilla**
Naïve Bayes	77.87
Multinomial Naïve Bayes	80.05
Bayesian Network	68.54
C4.5	68.77
Linear SVM	50.89
RBF SVM	61.43

The improvements in the results after using tossing graphs over Mozilla dataset are depicted graphically in figure 5.4. Figure 5.4 shows that MNB along with the tossing graphs give best prediction results than the other techniques. Naïve Bayes classifier also give better results than the remaining set of classifiers.

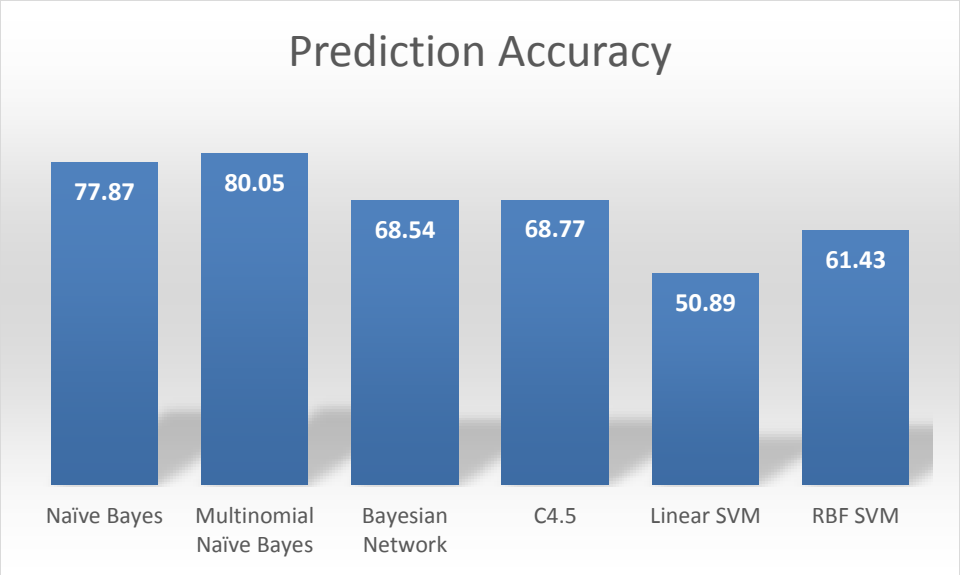

Fig 5.4 Comparative analysis of ML and Tossing Accuracy for Mozilla

The comparative analysis for the techniques used over Eclipse and Mozilla for the machine learning techniques using tossing graph is shown in figure 5.5. The results improved for all the classifier when used with tossing graphs.

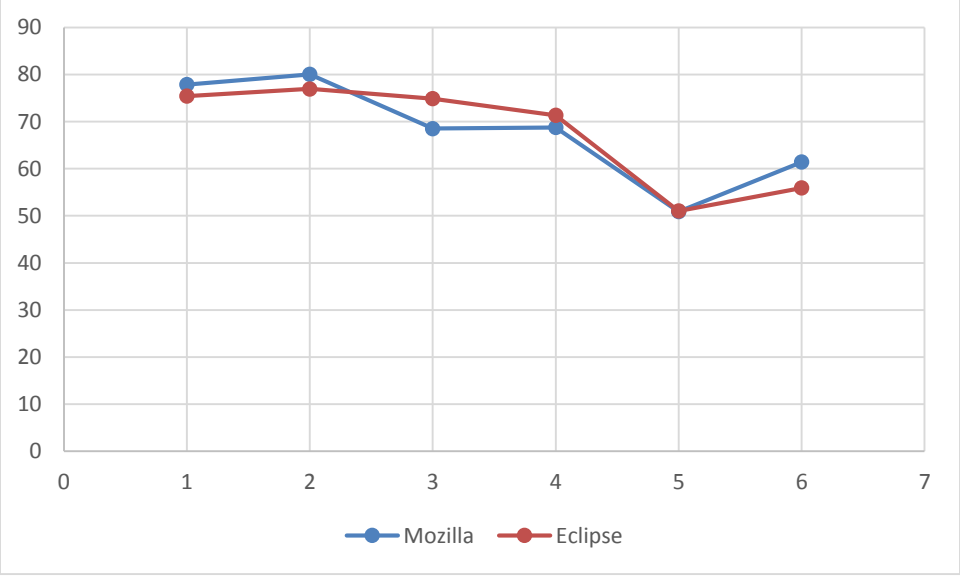

Fig 5.5 Comparative Analysis of ML and Tossing over Eclipse and Mozilla

The comparative analysis shows that tossing graphs work better for the type of datasets and give better results than using machine learning classifiers only.

CHAPTER-6
CONCLUSION AND FUTURE WORK

Multinomial Naïve Bayes Classifier is one of the powerful machine learning technique used in the process of mining the textual data. Using this approach, the automatic assignment of bugs are being predicted over Mozilla and Eclipse software repositories. The model gave the results of up to 80.05% accuracy by using the classifier along with tossing graphs. These tossing graphs provide a mathematical approach along with feature vectors, improving the accuracy of the classifier. The results so obtained are better than the previously used approaches.

To demonstrate the advantages of using the tossing graphs along with the classifiers, a comparative analysis is also being drawn between a set of classifiers. Amongst the classifier used, the Multinomial Naïve Bayes gives the accuracy better than all the other classifiers. The attributes used in this demonstration includes the report id, bug id, component id, and the description of the bug. The type of bug fall under one of the six categories, including UI, Core, Text, Debug, API, and Doc. The prediction accuracy is the average of the accuracies from top 1 to top 5 developers. Finally, the bug gets assigned to the developer based on the tossing graph and learning from the classifier.

The future task would be to use an increment learning approach for the classifier using interfold and intra fold updates. It also aims at giving a new incremental approach by using deep learning classifiers.

REFERENCES

[1] R. Seacord, D. Plakosh and G. Lewis, "*Modernizing legacy systems: software technologies, engineering processes, and business practices*", Addison-Wesley Professional, 2003.

[2] I. Sommerville, "*Software Engineering (7th Edition)"*, Pearson Addison Wesley, UK, 2004.

[3] Jeong, Gaeul, S. Kim and T. Zimmermann, "Improving bug triage with bug tossing graphs", In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 111-120, ACM, 2009.

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

[5] J. Anvik, L. Hiew and G. Murphy, "Who should fix this bug?", In *Proceedings of the 28th international conference on Software engineering*, pp. 361-370, ACM, 2006.

[6] J. Anvik and G. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions", *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20, vol. 3, 2011.

[7] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development", In *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1767-1772, ACM, 2006.

[8] G. Canfora and L. Cerulo, "How software repositories can help in resolving a new change request", *STEP 2005*, 2005.

[9] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports", In *Proceedings of 25th International Conference on Software Engineering,* pp. 465-475, IEEE, 2003.

[10] D. Lucca, A. Giuseppe, M. Penta and S. Gradara, "An approach to classify software maintenance requests", In *Proceedings of the International Conference on Software Maintenance",* pp. 93-102, IEEE, 2002.

[11] Z. Lin, F. Shu, Y. Yang, C. Hu and Q. Wang, "An empirical study on bug assignment automation using Chinese bug data", In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 451-455, IEEE, 2009.

[12] M. Dominique, A. Kuhn and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers", In *2009 6th IEEE international working conference on mining software repositories*, pp. 131-140, IEEE, 2009.

[13] B. Nicolas, R. Premraj, T. Zimmermann and S. Kim, "Duplicate bug reports considered harmful… really?", In *2008 IEEE International Conference on Software Maintenance*, pp. 337-345, IEEE, 2008.

[14] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging", In *2010 IEEE International Conference on Software Maintenance*, pp. 1-10, IEEE, 2010.

[15] A. Tamrawi, T. Nguyen, J. Al-Kofahi and T. Nguyen, "Fuzzy set-based automatic bug triaging (NIER track)", In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 884-887, ACM, 2011.

[16] J. Xuan, H. Jiang, Z. Ren and W. Zou, "Developer prioritization in bug repositories", In *2012 34th International Conference on Software Engineering (ICSE)*, pp. 25-35, IEEE, 2012.

[17] R. Shokripour, J. Anvik, Z. Kasirun and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation", In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 2-11, IEEE, 2013.

[18] S. Wang, W. Zhang and Q. Wang, "FixerCache: Unsupervised caching active developers for diverse bug triage", In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 25, ACM, 2014.

[19] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo and X. Wu, "Towards effective bug triage with software data reduction techniques", *IEEE transactions on knowledge and data engineering* 27, vol. 1, pp. 264-280, 2014.

[20] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts", *Empirical Software Engineering* 21, vol. 4, pp. 1533-1578, 2016.

[21] Badashian, A. Sajedi, A. Hindle and E. Stroulia, "Crowdsourced bug triaging", In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 506-510, IEEE, 2015.

[22] T. Zimmermann, A. Zeller, P. Weissgerber and S. Diehl, "Mining version histories to guide software changes", *IEEE Transactions on Software Engineering* 31, vol. 6, pp. 429-445, 2005.

[23] S. Mani, A. Sankaran and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging", In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pp. 171-179, ACM, 2019.