Lakehead
UNIVERSITY

Master of Computer Science Degree

Efficient Detection of Duplicate Bug Report using LDA-based Topic Modeling for Clustering and Classification

Authors:
Dhruvit Shah
Nishi Patel
Rinkal Mehta
Surya Yakkanti

Supervisor:
Dr. Thangarajah Akilan

A masters project submitted in fulfilment of the requirements for the degree of MSc Computer Science

in the

Department of Computer Science
Lakehead University
Thunder Bay, Ontario, Canada.

May 2020

Lakehead UNIVERSITY 50 1965-2015

Declaration of Co-Authorship

Dhruvit Shah, Nishi Patel, Rinkal Mehta, and Surya Yakkanti declare that this work titled, "Efficient Detection of Duplicate Bug Report using LDA-based Topic Modeling for Clustering and Classification" and the work presented in it is our own. We hereby declare that this project incorporates material derived from joint research. This report also includes the findings of the study under Dr. Thangarajah Akilan. We declare that, to the best of our knowledge, our work does not infringe anyone's copyright or violate any proprietary rights and any use of the works of other authors in any form like ideas, equations, figures, text, tables and programs are properly acknowledged at any point of their use.

We confirm that this work submitted for assessment is our own and is expressed in our own words. A list of the references employed is included in the bibliography.

Acknowledgements

First and foremost we would like to thank our supervisor Dr. Thangarajah Akilan for providing his valuable time to guide us through and having patience on our project. He always had time to discuss things and gave priority to reviewing our documents. Dr. Akilan is flexible with different directions in the field, and always gave us invaluable advice on the next step to take, he helped us shape the project by suggesting us different methods to improve our model. When we did not have a plan and did not know where our research was going, he helped us see the overall plan and helped us get through it together as a team. Also, he helped us create a quality report by contently providing feedback on the content we put out. It was really a great honour to work under Dr. Akilan who is a great supervisor and a humble person.

Then we would like to acknowledge our peers at the university who constantly gave us feedback throughout our project which immensely helped us.

Finally, we would like to acknowledge our family and friends for supporting and encouraging us to step up with the project. We would have not accomplished this without their love and support.

Contents

List of Figures

List of Tables

Abstract

A bug tracking system continuously monitors the status of a software environment, like an Operating System (OS) or applications. Whenever it detects an anomaly situation it generates a bug report and send it out to the software developer or maintenance center. However, the reported bug can be an already existing issue that was reported earlier and may have a solution in the master report repository at developer side. Such instances may occur in an overwhelming numbers. This poses a big challenge to the developer. Thus, an early detection of duplicate bug reports has become an extremely important task. This report proposes a two-tier approach of clustering and classification, whereby it uses Latent Dirichlet Allocation (LDA) for topic modeling, clustering and feature extraction techniques like Term Frequency - Inverse Document Frequency (TF-IDF), Word2Vec (W2V), FastText (FT), GloVe and multimodal features based Cosine Similarity (CS) measure is used for comparsion. The proposed model is tested on Eclipse dataset consisting of over 80,000 bug reports, which is the amalgamation of both master and duplicate reports. This work only considers the description of the reports for detecting the duplicates. The experimental results show that the proposed model achieves a recall rate of 67% for top-N recommendations. As a result of clustering, the proposed system is time efficient as it consumes only 9.378 secs per sample to process.

Keywords

Duplicate bug report, document similarity, natural language processing

Abbreviations

MR	Master Report
DR	Duplicate Report
MRR	Master Report Repository
NLP	Natural Language Processing
LDA	Latent Dirichlet Allocation
TF-IDF	Term Frequency - Inverse Document Frequency
CS	Cosine Similarity
BTS	Bug Tracking System
BoW	Bag of Words
W2V	Word2Vec
FT	FastText
RR	Recall Rate

Chapter 1

Introduction

1.1 Problem Statement

During the advancement, support and utilization of a software, there are consistently arising issues where the functionality of the whole system might get compromised due to internal defects in the software. Moreover, billions of dollars are spent by software associations for maintenance of a software [1] [2]. Whenever a technical dispute occurs in a software environment, the problem is stated as a bug. Hence, when a bug is identified in a software by a BTS such as Bugzilla, a bug report is generated. The BTS prioritize every bug report by its severity and assign them to the software developers for further assistance accordingly. The major issue in the process is there is a chance of generation of multiple bug reports that may have similar kind of concern which in turn can have an immense probability to possess an identical or correlative solution. This emerge the problem of duplicate bug reports. These bug reports are written in natural language text, which is usually ambiguous, this makes it harder to hunt down duplicate reports as different vocabulary can be used to describe the same defect in numerous reports. Due to this obstacle, the developer ends up striving repetitively for obtaining the same solutions for different bug reports. Companies spend over 45% of the cost of fixing these bugs [1]. Software testing and debugging is estimated to utilize more than 33% of the complete expense of software development [2].

If more than one bug has the same solution then the first report is being marked as a master report and the rest would be marked as duplicate reports of that particular master report. If there is an existing master report in the repository, for an incoming bug report and still that

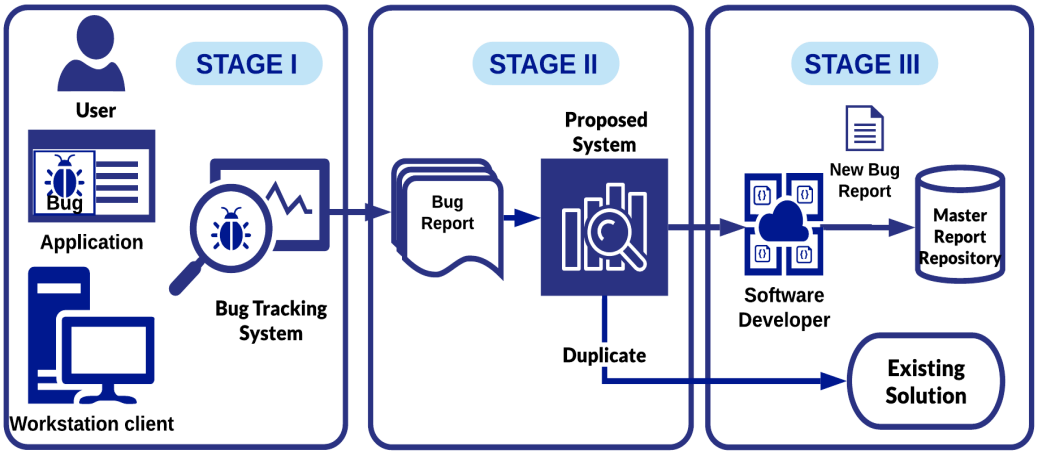

FIGURE 1.1: Bug Handling Flow: Stage I - bug report generation, Stage II - detection of duplicate bug report, and Stage III - the issue addressed at the developer side.

report is directed to the developer, it will consume lots of resources for an issue that is already or in process of being solved. As a result, detecting the duplicate bug report is extremely essential.

The proposed system is a solution to this matter which is intended to be cost-effective and time efficient. Furthermore, it will reduce the efforts of the software developers from fixing the defect which already has an existing solution by just applying the solution instead of struggling to resolve it from scratch. Figure 1.1 shows a flow from the generation of bug report to the operations carried out at the software developer side. In Stage I, the bugs are tracked and reports are generated by the bug tracking system. In Stage II, the proposed model receives the bug reports and determines if they are duplicates of already reported issues in the master report repository. In Stage III, if the new bug report is a duplicate of any master report, the solution is dispatched to the user; otherwise, it is submitted to the developer to seek out a fix and it will be stored in the master report repository as a new master report. So, this may limit the amount of bug reports reported to the developer. In 2012, approximately 7,600 bugs/month were reported to Mozilla programmers [1][2], which is essentially hard to handle. Automation can increase the productivity and reduce the time taken to seek out these duplicate bug reports [3].

1.2 Background

Most of the open source software projects like Bugzilla[1], JIRA[2] and Collabnet[3] have an open bug repository which is accessible to users of the software to report bugs. Allowing users to report problems helps to speed up the process and improve the quality of the software produced, to enable users to make these contributions most of the open source projects provide an open bug repository. A bug report has various parameters viz. title, description, priority, resolution, severity and status. Title states the issue in a single line whereas description is an elaborated explanation about the issue and priority is how fast a bug has to be fixed depending on its severity. So, basically the main issue is as addressed in section 1.1, which is, there may be a chance that a new bug reported has already existing solution in the master bug repository. This problem consumes lots of resources including finances and time, which this approach attempted to overcome by detecting the duplicate report and locating its master bug report for finding the solution for that bug. The proposed approach which is explained in section 4.1, is an efficient and effective solution for this issue.

1.3 Challenges

The proposed approach is using a dataset which has large amount of samples and its written in simple natural language. As a result, we faced certain challenges as mentioned below:

- Ambiguity on lexical, syntactic and semantic levels:
  Ambiguity is an intrinsic characteristic of humans, due to which, there are few challenges we faced in natural language processing. Ambiguity can be simply explained as sentences that have multiple alternative interpretations. Lexical ambiguity is a writing error which occurs when a sentence has a word that has more than one meaning. Syntactic ambiguity which is also called structural ambiguity, is the presence of two or more possible meanings in a single sentence. Usually syntactic ambiguity occurs due to poor word choice. Semantic ambiguity is when the words are different but the context is same. This is quite a bit of a challenge to overcome semantic ambiguity which we have tried to overcome using LDA, Word2Vec and FastText.

[1]Bugzilla: https://www.bugzilla.org/.
[2]JIRA:https://www.atlassian.com/software/jira
[3]Collabnet:https://www.collab.net/

- Word sense disambiguation:

  As we know, natural language can cause lots of ambiguity, in our case, there is a possibility that the bug might be different but the solution can be the same. It is a great challenge to find that kind of duplicate report.

- Memory problems and time constraints:

  Since we are dealing with a really huge dataset, there were a couple of issues of memory as it is really hard to process and analyze all the reports in the data. Additionally, to manage the large amount of data was significantly time consuming.

- Platform compatability:

  First, this approach utilized jupyter notebook but due to some technical glitches, we were unable to perform our model the way we wanted, so we migrated our project to google colaboratory which was really time consuming.

Chapter 2

Related Work

To release the burden on triager for detection of duplicate bug reports, many researchers have exhibited an interest in this field. In the start, efforts were focused on identifying relations between bug reports automatically. Fisher *etal.* [4] proposed an approach that discovers and visualizes the relation between features in software using the information in the bug reports. Researchers Sandusky and Gasser [5] present an approach that used duplicate and dependency relations between reports and informal references in a report to other reports to extract bug report networks to identify how large distributed open-source projects are managed also automated finding of duplicate documents is considered in other contexts also, in a large collection of documents duplicates are removed to maintain speed and productivity of search engines. In this context, the corpus of documents is relatively stable compared to bug repository in which numerous bug reports are added consistently. Dr Lyndon [6] built a java application by constructing a model of reports in the repository, it groups similar reports into centroids he used porter stemmer algorithm for preprocessing the resulting text and stemming each word after all the techniques, then the preprocessed text is converted into a document vector that is used to represent the report and used TF-IDF for weights to represent each centroid with a vector he then implemented his model on four different datasets like Firefox, Eclipse, Apache and Fedora core, for which he got the highest precision and recall rate of 29% and 50% for firefox dataset. Researchers Jalbert and Weimer [7] used Mozilla bug tracking system data set. They used Bag Of Words for textual analysis and Cosine distance between weighted word vectors derived from the documents for document similarity. Using all these techniques, they were able to find 8% of the duplicates. Based on Jalbert and Weimer's [7] experimental setup and approach Tian *et al.* [8] used Mozilla dataset for evaluation and they identified true and false positives, but for

similarity measures unlike Jalbert and Weimer [7], Tian *et al.* [8] used REP similarity measure, which is specifically designed for measuring the similarity of bug reports. They obtained true positive rate of 24.48%, which is three times increase in the TP rate compared to Jalbert and Weimer [7], In 2006, Hiew proposed an incremental clustering model using natural language processing (NLP) techniques to identify duplicate bug reports in his master thesis [4]. According to his report, the detection recall rate can achieve 20%-50% in four software projects when the recommendation list has 7 items(Eclipse: 20%, Fedora:31%, Apache: 32% and Firefox:50%). Runeson *et al.* [9] used Sony Ericsson mobile communications bug report dataset and were able to find 40% of the marked duplicates, they have made a couple of batch runs with multiple evaluation metrics like cosine, Dice and Jaccard similarity measures and they have achieved recall rate for different time frames, different stop word list, different synonyms and recall rate using project field. Based on Runeson's *et al.* [9] work Wang *et al.* [10] considered execution information. They approached the problem with two heuristics to combine two kinds of information, they have also evaluated their approach on bug reports from eclipse and Firefox repositories, they went with a calibrated approach using natural language information with class based heuristics and using only summary they were able to increase recall rates by 11-20% with eclipse and 18-26% with Firefox data sets.

Ashish and Pankaj [11] mostly used the common natural language techniques like rest of the researchers to clean their eclipse bug report, but by applying character level Ngram as low-level features to represent the title and description of the bug report using these features and techniques they were able to get a recall rate of 33.92% and 61.94% for top 50 results using 1100 and 2270 test cases. Instead of REP for measuring the similarity of bug reports like Tian *etal.* [8], Zou *et al.* [12] improved the Ngram similarity algorithm and introduced a new method called LNG which is a combination of topic model LDA and word-based Ngram model, by doing this they were able to outperform all the previous LDA and Ngram models and they achieved a higher recall rate of 11 - 18% compared to REP. Some of the recent works include researchers Phuc and Minh [13] worked on open source projects like ArgoUML, SVN and Apache, they used NLP techniques like Ngram and cluster crinkage, used vector space model to represent bug reports like most of the researchers. Firstly they extracted tokens of words and then used character level Ngram to find similarity between lexical words in detail and then they have used word level Ngram to find sequence relation between words, for similarity computation they have tried other similarity measures but decided to go with cosine similarity as it has the best performance.

Chapter 3

Methodologies

3.1 Latent Dirichlet Allocation (LDA)

LDA is an abbreviated name for Latent Dirichlet Allocation whereupon, "Latent" stands for hidden and "Dirichlet" stands for probability distribution. LDA is an unsupervised statistical model that allows class of observations to be interpreted by unseen groups which explain why some parts of the data are similar. The underlying concept is that each document is represented as distribution of latent topics, where each topic is characterized by a distribution of words. By implementing LDA on BoW corpus, a document-topic matrix is generated. As a result, for each document, LDA results a probability distribution of topic within that document. Figure 3.1 illustrates a probability distribution of topics across a single document.

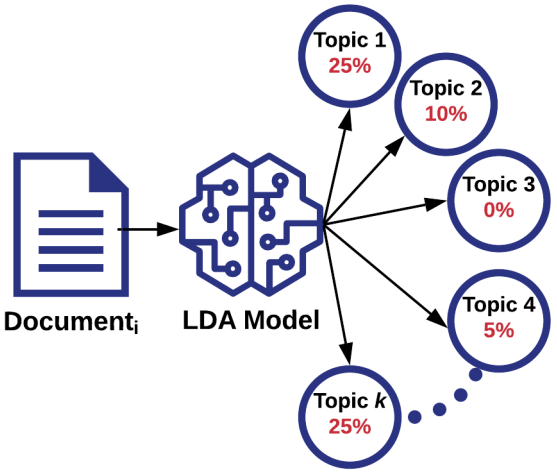

FIGURE 3.1: Example of Probability Distribution of Topic in a Document$_i$ using LDA.

To construct the topic model, Gensim library from NLP toolkit is used. LDA[1] works by first making a key supposition on the manner in which a document is produced by choosing a set of topics and afterwards for every topic, choosing a set of words. There are a few steps for demonstrating how LDA works internally and assign topics to a document d:

1. t topics can be given to LDA as a parameter across all the documents.
2. It randomly assigns each word v in document to one of the t topics.
3. This random assignment gives both topic distribution of all the documents and word distributions of all the topics.
4. Nevertheless, to improve these assignments, it will again go through each word v in document d and determine two things :

   The amount of words in document d that are at present allocated to topic t.

   The amount of assignments to topic t over all the documents that come from this word v.

   After checking this, it will reassign word v to a new topic.
5. So, basically it is assuming that all topic assignments except for the current word are correct, and then updating the assignment of the current word using the model.
6. The above steps will iterate until all the assignments seems appropriate. By using these assignments, probability distribution of k topics in document d and the words assigned to each topic can be accomplished.

LDA suppose that every one document have t topics. Assuming that the t dimensional vector θ is the parameter of the prior data distribution of the subject, $k \times z$ matrix ,δ is the parameter of the distribution of the word (δ is sum of words) within the topic, i.e., $\delta_{ij} = \mathrm{p}(x_j|y_i)$ is that the probability of the word x_j within the i_{th} topic. Thus, we generate a document topic distribution of N topics. The probability of the N word is expressed as:

$$p(\alpha, y, x) = p(\alpha|\theta) \prod_{n=1}^{N} p(y_n|\alpha)(x_n|y_n, \delta), \tag{3.1}$$

[1]LDA:https://towardsdatascience.com/lda-topic-modeling-an-explanation-e184c90aadcd

where α is the topic distribution vector of the document, y is the topic vector of N dimensions, and x is the vector composed of the N words. Since α and y are latent variables which cannot be observed, they are eliminated through marginal distribution from the left:

$$p(y|\theta,\delta) = \int p(\alpha|\theta) \prod_{n=1}^{N} p(y_n|\alpha)(x_n|y_n,\delta)d\alpha, \tag{3.2}$$

To corpus c which has m documents,

$$p(c|\theta,\delta) = \sum_{c=1...m} p(x_c|\theta,\delta), \tag{3.3}$$

Thus,

$$p(c|\theta,\delta) = \prod_{d=1}^{m} \int p(\alpha_d|\theta)(\prod_{n=1}^{N_d} \sum_{y_{dn}} p(y_{dn}|\alpha_d)p(x_{dn}|y_{dn},\delta))d\alpha_d, \tag{3.4}$$

As shown above, the process of constructing the LDA model is to achieve maximized parameters θ and δ of $p(c|\theta,\delta)$.

$$p(\alpha,y|x,\theta,\delta) = \frac{p(\alpha,y,x|\theta,\delta)}{p(y|\theta,\delta)}. \tag{3.5}$$

We obtain p(x|y), θ and δ after building a topic model, and then deduce the topic of the new unlabeled text by the trained topic model and predict its corresponding topic distribution which is also the conditional probability distribution of the document in the topic space.

3.1.1 Selection of Optimum Number of Topics in LDA

To select the optimum number of topics in LDA, Coherence Score[2] is used. Topic coherence is the combination of intrinsic measure UMass and extrinsic measure UCI.

$$coherence = \sum_{i<j} Score(w_i, w_j) \tag{3.6}$$

[2] http://qpleple.com/topic-coherence-to-evaluate-topic-models/

UCI measure uses a score function the PMI (Pointwise mutual information).

$$score_{UCI}(w_i, w_j) = log\frac{p(w_i, w_j)}{p(w_i)p(w_j)}, \tag{3.7}$$

where $p(w_i, w_j)$ is the probability of seeing both w_i and w_j co-occurring in a random document, $p(w)$ represents the probability of seeing w_i in a random document.

UMass uses score function which is the empirical conditional log-probability

$$log(w_i|w_j) = log\frac{p(w_i, w_j)}{p(w_j)}, \tag{3.8}$$

smoothed by adding 1 to $D(w_i,w_j)$.

$$Score_{UMASS}(w_i|w_j) = log\frac{D(w_i, w_j) + 1}{D(w_i)}, \tag{3.9}$$

where $D(w_i,w_j)$ is the count of documents containing both words w_i and w_j, D is the total number of documents in the corpus, $D(w_i)$ is the count of documents containing the word.

3.2 Formation of Features Vectors using Single Modality and Multi-Modality Feature Extraction

3.2.1 TF-IDF

TF-IDF stands for term frequency-inverse document frequency. It is an algorithm mainly used in information retrieval, summarization and text mining. Different variations of TF-IDF algorithms are being used in search engines to find the relevance or score of a given input. TF-IDF is composed of two terms they are

- TF - *Term Frequency* : It can be represented as the number of times a word appears in a document divided by the total number of words present in the documents.

- IDF - *Inverse Document Frequency* : It measures how important a term is, Basically tf is the occurrence count of a term in particular document but idf is number of different documents the term appears, idf of a term is the number of documents present in the corpus divided by document frequency of a term. Finally both the tf and idf is combined to form the score of a term t in document t.

$$TF - IDFScore = TF_{xy} \times IDF = TF_{xy} \times \log \frac{N}{df}, \tag{3.10}$$

where TF_{xy} is the frequency of key phrase X in the article Y, N is the number of document in the corpus. df is the number of document containing key phrase X.

3.2.2 Word2Vec

Word2Vec is a two layer neural network that processes text by "vectorizing" words. Word2Vec contructs linguistic context of words which preserves the semantic similarity of the words. A large corpus of text can be fed to Word2Vec as an input, which will produce a vector space of several hundred dimensions in which each and every unique word will be allocated an associated vector in the space. It is used to produce word embeddings. Word embedding is one of the most popular representation of document vocabulary. It is proficient for capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. So, the word embedding can be obtained by using two methods of Word2Vec: Skip gram and Continuous Bag of Words (CBoW).

- CBoW : In this architecture, the model will predict the current word from a window of neighbouring context words. The sequence of context words does not impact the prediction.
- Skip gram : In this architecture, the model uses the current word to assume the adjoining window of context words. The skip-gram architecture weighs close by context words more heavily than more distant context words.

3.2.3 FastText

FastText is a framework for learning word representations and also performs robust, rapid and precise text classification. It is basically an extention of Word2Vec. But unlike Word2Vec, FastText considers each word as a Bag of Character n-grams. This is also called as a subword model. Taking the word "horse" and n=2 (bi-grams) as an example, it will be represented by the character n-grams: <ho, hor, ors, rse, se> and the special sequence <horse> representing the whole word. After that, word vectors of each n-gram is generated and all the n-gram word vectors are summed up to get an embedding for a whole word. Similar to Word2Vec, FastText also preserves the semantic similarity of the words.

3.2.4 GloVe

GloVe[3] stands for Global Vectors for Word Representation which uses global corpus statistics. It is an unsupervised algorithm used to obtain word embedding. Generally, training is performed on aggregated global word-word co-occurrence statistics from a corpus. But, the proposed model is trained on our word-word co-occurrence matrix.

3.2.5 Multi-Modality Feature Extraction

The proposed approach is using multi-modality feature extraction by integrating multiple feature vectors to enhance performance. Consequently, four multi-modal feature extraction techniques are introduced.

1. *FG* - Fusion of FastText and GloVe
2. *GW* - Fusion of GloVe and Word2Vec
3. *FW* - Fusion of FastText and Word2Vec
4. *FGW* - Fusion of FastText, GloVe and Word2Vec

Equation (3.11) depicts the fusion of multiple feature vectors formed by different feature extraction techniques.

$$\bar{v} = f_n(v_1^{R^{100}}, v_2^{R^{100}}), \tag{3.11}$$

[3] https://nlp.stanford.edu/pubs/glove.pdf

where v_1 and v_2 are the vectors formed by different feature extraction techniques, n can be a non-zero integer between 1 to 4 and R is the dimension of the vectors. The operations employed for amalgamation of the feature vectors are revealed in the following equations:

$$f_1(a, b) = a \oplus b, \tag{3.12}$$

where b is concatenated with a, which in this case would be v_1 and v_2. Also,

$$f_2(a, b) = avg(a, b), \tag{3.13}$$

where the average of a and b is acquired. Besides, this approach utilizes the PCA technique. PCA stands for Principal Component Analysis which is a method for decreasing the dimension of the data, enhance the interpretability but simultaneously minimizing data loss. It does so by creating new unlinked variables that successively maximize variance.

$$f_3(a, b) = PCA(f_1(a, b)) \tag{3.14}$$

In the above equation, PCA is applied on the concatenated a and b to perform dimensionality reduction. Moreover,

$$f_3(a, b) = PCA(f_2(a, b)) \tag{3.15}$$

describes PCA applied on the average obtained from equation (3.13). These were the few techniques we used for the aggregation of the feature vectors formed using multiple feature extraction techniques.

3.3 Cosine Similarity

Cosine Similarity is a similarity measure between two non-zero vectors of an inner product space. It substantially evaluate the cosine angle between two vectors that are projected to the multidimensional space. The cosine value ranges from 0 to 1. The bigger the angle, the smaller the cosine value. The cosine similarity work as follows: the given documents are projected into a vector space and generates vectors for each document. Later, the cosine angle between two documents can be measured and it can be analysed as, the closer the documents by the angle, the higher the cosine similarity.

$$Sim(d_1, d_2) = \frac{\sum_i (d_{1i} \cdot d_{2i})}{\sqrt{\sum_i (d_1^2 i \cdot d_2^2 i)}}, \tag{3.16}$$

where d_{1i} and d_{2i} are the vectors of document d_1 and d_2 in i_{th} dimension topic space.

3.4 Euclidean Similarity

It is derived using Euclidean distance as mentioned in equation (3.18). Euclidean distance is just a straight-line distance between two points in euclidean space. It can be referred to as a Pythagorean metric. The Euclidean distance can be measured between two normalized vectors in euclidean space.

$$Distance_{euc}(d_1, d_2) = \sqrt{\sum_{i=1}^{n} (d_{1i}^2 - d_{2i}^2)}, \tag{3.17}$$

where d_{1i} and d_{2i} are the vectors of document d_1 and d_2 in euclidean n-space.

$$Sim_{euc}(d_1, d_2) = \frac{1}{1 + Distance_{euc}}, \tag{3.18}$$

the above equation is to derive euclidean similarity from euclidean distance.

Chapter 4

Proposed Framework

4.1 Proposed Model

The proposed model is an approach for enhancing the bug handling process to be time-efficient and cost-effective which will ease the tasks of software developer. When a new report enters, this approach will seek out the top-N similar master reports for the new report, which will preserve a great deal of time of the programmer to find the solution for the bug as it may already exist. By receiving the top-N recommendations, the programmer can discover the most matching issue and can apply the existing solution.

As depicted in the fig. 4.1, the proposed model is consists of two significant stages: Clustering and Classification. So, this approach is using LDA model for topic modeling and clustering purposes and feature extraction techniques for the similarity measures to identify the top-N recommendations. The basic overview of the two stages is given below:

- Stage 1: Clustering
  - LDA is trained on the preprocessed master reports to form clusters.
  - Pre-trained LDA is applied on the preprocessed duplicate report to find the most relevant cluster in which associated master report may exist.
- Stage 2: Classification
  - Home cluster: The duplicate report jumps into the selected cluster to find the most similar master report.

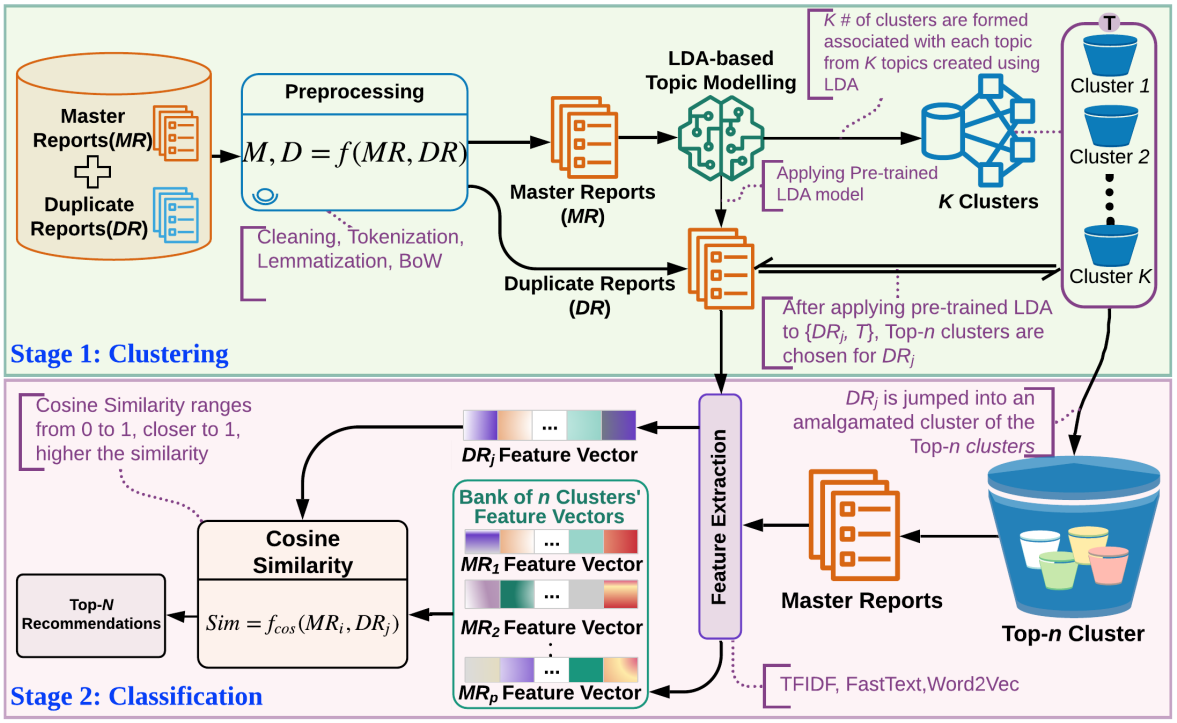

FIGURE 4.1: The Functional Flow of the Proposed Model.

- Finding the MR:
  - Cosine similarity would be measured between feature vectors of the duplicate report and the master reports in the corresponding cluster individually.
  - Top-N cosine similarities would be selected which would result in top-N recommendations.

As this was just an overview of both the stages, they are deeply elaborated in later sections.

4.2 Clustering

4.2.1 Preprocessing

The bug reports are ambiguously written in simple natural language. Consequently, the data requires sufficient amount of preprocessing. The goal of the proposed work is to get data in a clean and standard format for analysis. As a result, there are five main steps involved in data preprocessing. They are :

(a) Cleaning Text

(b) Stop word Removal

(c) Tokenization

(d) Lemmatization

(e) Bag Of Words (BoW)

4.2.1.1 Cleaning Text

Usually, natural language is written ambiguously. So, it requires adequate cleaning. Cleaning of the bug reports dataset includes removal of unnecessary part of the text and elimination of invalid bug reports. Removal of unnecessary parts includes elimination of punctuations which involved question mark (?), exclamation mark (!), comma (,), semicolon (;), colon (:), hyphen (-), parentheses((,)), brackets ([,]), braces ({,}), apostrophe (’), quotation marks (”), and ellipsis(...). In elimination of invalid bug reports, this approach initially discards the reports with empty description as description field is the one which is being used for the analysis. 129 reports with empty description were found, which were eliminated from the data. Invalid bug reports also includes few descriptions consists of some redundant fixed sentence patterns: “Fixed in HEAD” which means that the particular issue is been revised in previous versions and “Has been marked as read-only” which means this bug report is been marked as read only. So, the bug reports with these fixed patterns are eliminated as it has nothing to do with the bug itself but it can impact the similarity measures.

4.2.1.2 Stop Word Removal

There are many terms, for example, “the”, “ that”, “a”, “in” etc., that are referred as stop words, which does not carry much significance but can adversely affect the similarity analysis [9]. These words generally occurs with almost same frequency but are not associated with context of the text. These words are being removed so that more emphasis can be granted to the words with higher significance. After stop words removal, all the text is been converted to lower case.

4.2.1.3 Tokenization

Tokenization is the process of splitting text into minimal considerable elements called "tokens". For example, if a sentence to be tokenized is "Apple is a healthy fruit." then, after data cleaning and stop words removal, there would be 3 terms left, which would be tokenized and 3 tokens would be obtained as follows: "apple", "healthy" and "fruit". All the bug reports are tokenized and the list of tokens for each report will become input for all the further processing steps.

4.2.1.4 Lemmatization

Initially, stemming was performed on the tokenized data. But the outcome of stemming was not satisfactory as it derives a word to its root form by removing its suffix, applying stemming on our data, some words changed its meaning and some words turned meaningless, for eg., "Title" changed into "Titl" and "Eclipse" changed into "Eclips". This can affect the context of the word. Therefore, lemmatizing is choosed over stemming in this approach. Lemmatizing derives a word to its root form preserving the context, for eg., "fixing" will become "fix" , "requires" will become "require" etc.

4.2.1.5 Bag Of Words (BoW)

Bag of words is a natural language technique used to extract features from text documents. It is basically a portrayal of text which will depict the occurrence of a word within a document. Additionally, it is used to make vocabulary of all the unique words occurring within the documents. A text is represented as bag of its words, ignoring the grammar and even word sequence but keeping multiplicity. There are two steps involved in this process: determination of vocabulary and vectoring the words according to its presence in a document. This approach created a bag of words for the bug reports dataset.

4.2.2 LDA-Based Clustering

As displayed in the figure 4.1, the dataset is an amalgamation of master bug reports and its duplicate bug reports. So, this work splits master bug reports from the duplicate bug reports as this model is later tested on the duplicate reports for the quantitative analysis.

Now, the LDA model is trained on the BoW corpus created using the master reports. As mentioned in section 3.1, LDA outcomes the probability distribution of the topics within each document. Hence, LDA will provide the probability distribution of each topic in each and every master report. The number of clusters would be the same as the number of topics. If LDA creates t topics, then we will initiate clustering by creating t empty clusters. After analysing the topic distribution in a master report, it will be assigned to the number of cluster in which it has the highest topic distribution. Now, the following steps shows the cluster formation in accordance of the topic modeling by an example:

- Let's suppose, LDA creates five topics. Hence, five empty clusters will be created.
- As shown in figure 4.2, we will analyse the topic distribution of Master Report$_i$ (MR_i) after applying trained LDA on it.
- As we can see, topic 3 covers the highest probability distribution of MR_i, consequently, MR_i will be allocated to cluster 3 and will now belong to cluster 3.

This process will reiterate on each master report until all the master reports are allocated to their associated cluster.

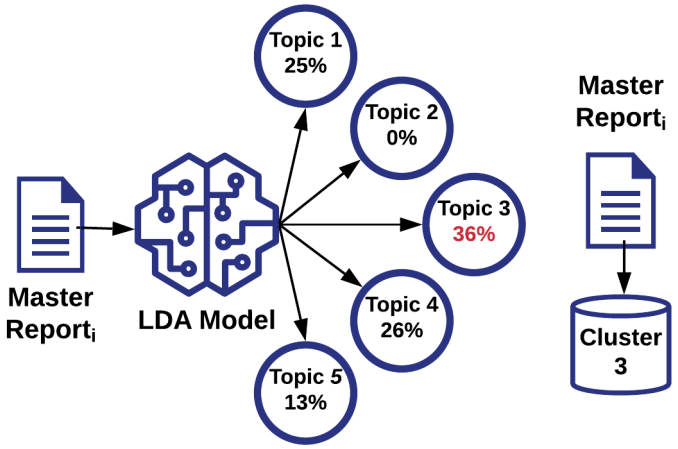

FIGURE 4.2: Cluster Formation based on Topic Modeling.

4.3 Classification

4.3.1 Selection of Top-n Clusters

As shown in figure 4.1, when a duplicate report enters, a pre-trained LDA model is applied on that report. The word pre-trained means the LDA model is already trained on the master reports, so it knows the distribution of words in each topic already. As stated in section 3.1, LDA performs to outcome the probability distribution of topic in each document. By applying pre-trained LDA on the duplicate report, it will result the probability distribution of each topic in that report.

Now, by adopting the same technique as performed in section 4.2.2 for the duplicate report, the topic with the highest probability distribution is identified and it will lead to the cluster in which its master report may reside. But unlike inserting the report into that cluster, the duplicate report is compared to that cluster which will be discussed in section 4.3.2. If the size of data is huge, there might be a possibility that the associated master report may not exist in the selected cluster. For overcoming this limitation, this approach selects top-n clusters. Top-n means the n clusters whose associated topics has the maximum probability distribution. So, in case, if the corresponding master report fails to exist in the top-1 cluster, it will also consider another cluster to find the appropriate one.

4.3.2 Analogy of Reports for Classification

After performing the process of selection of Top-n clusters as referenced in section 4.3.1, the corresponding master report for the duplicate report has to be identified. The most essential phase in this process is to find the similarity between the reports. To find the similarity between the reports, the average of cosine similarity metric and euclidean similarity that is explained in section 3.3 and section 3.4 respectively, is used. Cosine similarity measures the cosine angle between two non-zero vectors in inner product space. Consequently, to find the cosine similarity, feature vectors are required, which are formed in section 3.2. Euclidean similarity uses the same vectors to find euclidean distance which is being converted into similarity. The duplicate report is being compared to every master report existing in the Top-n clusters. As a consequence, feature vectors of the duplicate report and each master report from the Top-n clusters are

created followed by the average calculation of cosine similarity and euclidean similarity between both feature vectors.

4.3.3 Top-N Recommendations

After measuring the similarities as mentioned in the above section, the last step to do is to seek out the top-N recommendations. For that, we just have to identify the top-N recommendations which has the highest similarity measure with the duplicate report.

Chapter 5

Results

5.1 Dataset

This approach is conducted on Eclipse Dataset [1] which was collected from the year 2001 to 2013. It consists of 85,156 bug reports. After all the preprocessing steps performed in section 4.2.1, we were left with 81618 bug reports, in which 69,535 are marked master reports and 12,083 are marked the duplicates of the master reports, which means there is 85.19% master reports and 14.81% duplicate reports. This makes a master report to duplicate report ratio of about 85:15. A bug report is generally a structured document that consists of different fields as shown in table 5.2. The table 5.1 can be seen depicting duplicated reports indicated by DR and MR being the master reports. Here we can see that report sixteen is the master report of 2956 and report 29645 is duplicate of report 20.

Issue_id	**Title**
16	$[MR]$ Automerge button
2956	$[DR]$ Missing automerge
20	$[MR]$ Workspace files
29645	$[DR]$ need workspace to save repo connections

Table 5.1: Example of Master Report (MR) - Duplicate Report (DR) Pair.

[1] https://bugs.eclipse.org/bugs/.

Domain	**Description of the domain**	**Sample**
Issue_id	Unique ID for each and every bug report	184467
Priority	Priority of the report on the basis of its severity	P3
Component	In which component category would the issue of the report belong to?	UI
Duplicated_issue	The issue id of its master report	184421
Title	A short summary about the issue	[Key Bindings] Warnings in log file
Description	A detailed description of the issue	build i0427-0010; ; Just noticed the following 2 warnings in my log file. No steps... dont know what I did to deserve this. ; ; Warning; Fri Apr 27 11:50:51 EDT 2007; A conflict occurred for CTRL+W: [Binding(CTRL+W;Parameterized Command (Command(org.eclipse.help.ui.-closeTray;Close User Assistance Tray;Close the user assistance tray containing context help information and cheat.
Status	Whether the issue is been resolved, closed or verified	RESOLVED
Resolution	Whether the issue is been fixed, won't fix, invalid or duplicate	DUPLICATE
Version	The version of the product for which the issue is about	3.3
Created_time	The time of creation of the bug	2007-04-27 11:57:00 AM
Resolved_time	The time when the solution of the bug is obtained	2007-04-27 12:19:00 PM

Table 5.2: Structured Information of Bug Report.

5.2 Experimental Setup

The experimental analysis is carried out on the eclipse dataset discussed in section 5.2. Apparently, there are multiple experiments conducted using two distinct amount of data. The first experiment was using TF-IDF as a feature extraction technique, which was conducted on 10,000 master reports and tested on 200 duplicate reports. The second experiment was using W2V and FT as feature extraction techniques individually, which was performed on 70,629 master reports and tested on 200 duplicate reports.

The experimental analysis was conducted on a computer with an Intel Core i7-8550U processor running at CPU speed 1.8 GHz using 12 GB of RAM, on windows version 10.

5.3 Evaluation Metric

To evaluate the accuracy of the detection strategy, this approach has used the recall rate measure. So, the recall rate is basically used to evaluate how effective the proposed model is. It can be defined as the ratio of the duplicates which can accurately find its corresponding master report in the top-N recommendations.

$$RecallRate = \frac{N_{true}}{N_{total}}, \tag{5.1}$$

Equation 5.1 indicates how to calculate the recall rate, where N_{true} is the number of duplicate reports that can correctly find its associated master report, and N_{total} is the total number of duplicate reports.

5.4 Quantitative Analysis

5.4.1 Coherence Score

Firstly, the LDA model is trained on the master reports for 20 passes and 100 iterations. After training the model, coherence score[2] is calculated, which is elaborated in section 3.1.1 as a measure to evaluate the optimum number of topics. A sanity analysis was performed to determine the optimum number of topics along with the coherence score. As shown in table 5.3, for 10 topics, we get highest coherence score as well as the highest recall rate compared to higher number of topics. Consequently, 10 is selected as the optimum number of topics for our dataset. In this analysis, the RR is obtained using FT model as a feature extractor.

No. of Topics	**Coherence Score**	**RR Using** *FG*
10	0.621	67%
20	0.593	64.5%
30	0.582	62.5%
40	0.509	63.5%
50	0.476	60%
60	0.472	68.5%
70	0.455	67.5%
80	0.445	65.5%

TABLE 5.3: Coherence Score and RR for Different Number of Topics.

5.4.2 Number of Master Report in each Cluster

In section 4.2.2, shows the process of clustering. As shown in table 5.3, the LDA model works better for 10 number of topics, consequently, the proposed system formed 10 number of clusters. Each cluster has different number of MR based on the topics which they are related to. The table 5.4 shows the number of MR in each 10 clusters. The MRs in each cluster are contextually related to each other based on topic distribution.

[2] http://qpleple.com/topic-coherence-to-evaluate-topic-models/

Cluster No.	**No. of MR**
0	6975
1	9986
2	2832
3	7264
4	3467
5	1106
6	10288
7	8624
8	10538
9	8455

TABLE 5.4: Number of Master Reports in each Cluster.

5.4.3 Performance Analysis

The performance of the proposed approach is measured using recall rate (RR) as mentioned in section 5.3. The table 5.5 depicts the experiments of four different feature extraction techniques M1-W2V, M2-FT, M3-GloVe and M4-fusion of FastText and Glove i.e., FG and acquired the RR for top-N recommendations. In this approach, CBoW is used for both W2V and FT as it is giving better results for our experiments than Skip-gram. Both the models are trained for 100 iterations and 5 workers. While GloVe model is trained for 1000 epochs. We can analyze that FT perform better than the other two models. For Top-2.5k using Clustering, FT obtained 64.5% RR. On the other hand, W2V and GloVe obained 62% and 58% respectively. So, it can be analyzed that for both FT and W2V, the highest RR is achieved for feature vectors of size 100. Similarly, FT performed better then other two even without clustering.

-

		With Clustering					Without Clustering				
Top-N		**100**	**500**	**1000**	**2000**	**2500**	**100**	**500**	**1000**	**2000**	**25000**
M1	**RR**(%)	29.5	44.0	54.0	61.5	62.0	33.0	48.5	55.0	62.0	65.0
	Time(sec)	7.359	7.359	7.401	7.353	7.443	25.704	25.269	25.515	25.704	25.911
M2	**RR(%)**	30.5	45.5	55.5	63.0	64.5	35.0	50.0	57.5	64.0	67.0
	Time(sec)	7.360	7.390	7.410	7.420	7.460	25.944	26.667	26.694	26.850	26.055
M3	**RR**(%)	23.5	37.5	46.5	55.0	58.0	27.5	42.0	49.5	59.0	62.0
	Time(sec)	5.286	5.322	5.346	5.400	5.409	17.355	17.451	17.424	17.517	17.436
M4	**RR**(%)	31.5	49.5	57	64.5	67.0	27.5	42.0	49.5	59.0	62.0
	Time(sec)	9.438	9.318	9.3462	9.468	9.380	17.355	17.451	17.424	17.517	17.436

TABLE 5.5: RR and Per Sample Processing Time (PSPT) Comparison of M1-W2V, M2-FT, M3-GloVe and M4-FG.

		f_1	f_2	f_3	f_4
RR(%)	**M4-***FG*	65.5	66.5	66.5	67.0
	M5-*GW*	62.0	63.5	64.0	65
	M6-*FW*	64	62.5	66	65.5
	M7-*FGW*	64	62.5	65.5	66.5

TABLE 5.6: RR of model M4, M5, M6 and M7

Table 5.6 depicts the outcome of the multi-modality feature extraction techniques using different kind of fusing approaches. Overall, it can be analysed that by aggregation of multiple models, the performance of the system is increasing unlike using the single modality feature extraction. It can be seen that fusion FG results 65.5%, which is slightly higher than the results of other models for all the four fusing techniques. Besides, the technique f_2 is better performing than f_1 for FG as the performance is increased by 1%. In fact, it can be considered that the performance of each model is highest for technique f_4 that is dimensionality reduction using PCA for average of the vectors. By this study, it can be concluded that the combination of FastText and GloVe (FG) by f_4 is performing the best among all the models.

In figure 5.1, it can be certainly analysed that among model M1, M2 and M3, model M2 gives the best performance while model M3 got the lowest recall rate. Later, comparing the result of M2 with the fused models M4, M5, M6 and M7, performance of the fused model is better than the performance of the single models. Overall, it can be seen, model M4 is the best performing model among all the models.

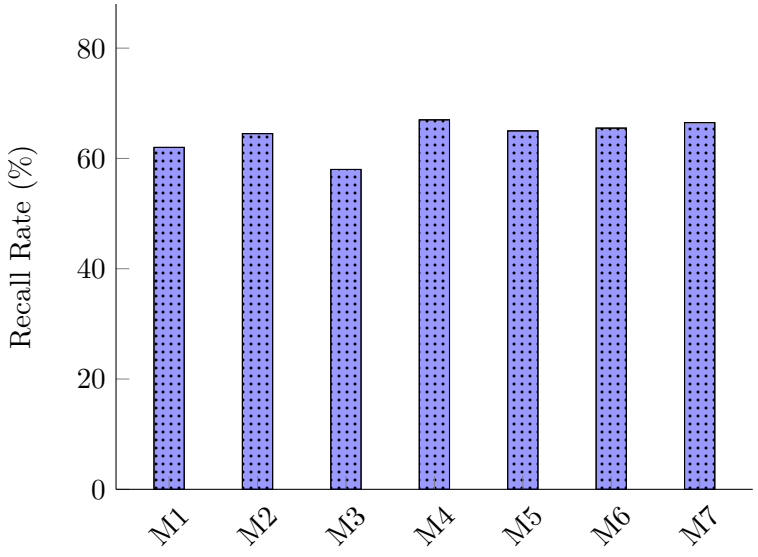

FIGURE 5.1: Performance Analysis across all the Models for Top-2.5k.

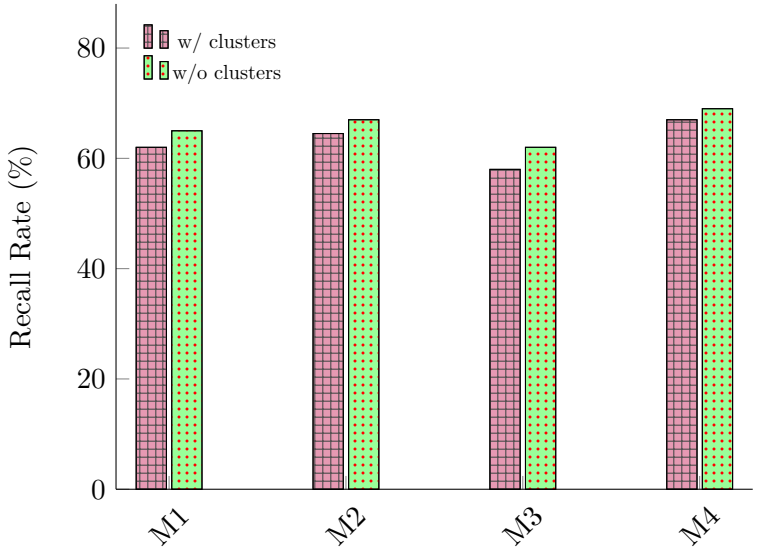

FIGURE 5.2: RR for models M1, M2, M3 and M4 for Top-2.5k.

This work is using clustering on the basis of topic modeling. So, to analyze the effectiveness of this approach, the analysis is performed with and without using clustering. By figure 5.6, it can be examined that the performance of the model M1, M2, M3 and M4 with clustering is acquiring a diminutively less recall rate than the one without clustering which can be unconsiderable against the amount of time being saved which is elaborated in next section.

5.4.4 Time Analysis

The proposed approach is using clustering based on topic modeling as mentioned in section 4.2.2, which makes this approach legitimately efficient. As, if clustering is not used, the comparison has to be performed on a one-to-one basis, while using clustering, just the reports which are residing in the top-n clusters have to be compared. This process saves a great deal of time with nearly a similar recall rate.

The table 5.5 shows the per sample processing time in seconds(sec) for single model feature extraction techniques Word2Vec, FastText and GloVe with clustering and without clustering. As mentioned in section 5.4.3, FastText performes better in terms of recall rate among all single modal techniques. But in terms of time, if we compare all three single model, we can see that GloVe performes better. We can analyze from table 5.5 that for top-2.5k recommendations, GloVe using clustering consumes 5.409 secs per report. As mentioned in section 5.4.3 and table 5.6, the recall rate of model FG is highest using f_4. But in terms of time, compared to single feature extraction techniques, FG is bit time consuming.

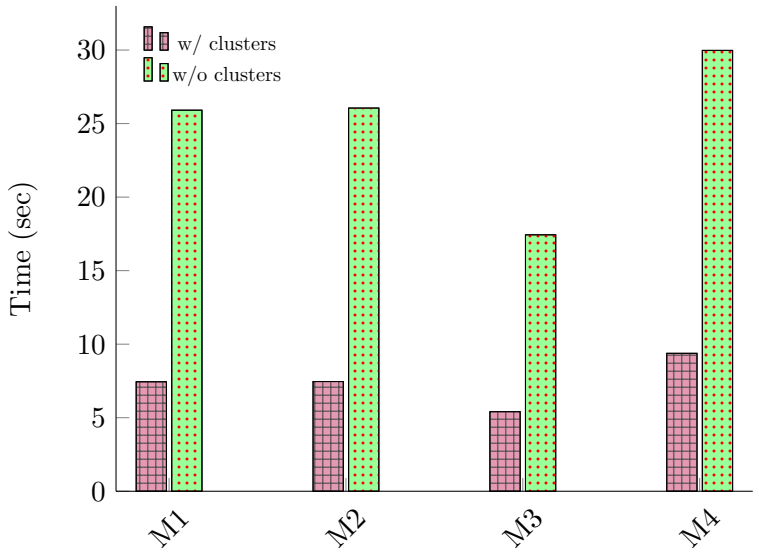

FIGURE 5.3: Per Sample Processing Time for Model M1, M2, M3 and M4 for Top-2.5k.

On the other hand, when we compare the results of both single and multimodal techniques using clustering and without clustering, we can see that the models without clustering consumes a great deal of time per report. If we analyse the efficiency of M2, then it consumes 18.595 sec more when used without clustering. Similarly, M4 with clustering consumes 20.595 sec less in comparison without clustering which is depicted in figure 5.3.

Initially, we attempted the analysis using TF-IDF as a feature extraction technique but it can be observed from table 5.7 that it is time consuming as it takes approximately 41.58 seconds per sample. Considering, that TF-IDF is experimented on just 10,000 reports which is just 15% of the whole dataset. While FT, W2V and GloVe is performed on whole dataset and acquired 7.460, 7.443 and 5.409 seconds respectively. As a result, we decided to not to perform TF-IDF in our approach as its very time consuming.

Per Sample Processing Time using TF-IDF		
Top-N	**RR(%)**	**Time(mins)**
10	12.5	0.654
100	34.0	0.654
500	44.0	0.567
1000	46.5	0.573
2000	52.0	0.573
2500	50.0	0.693

TABLE 5.7: RR using TF-IDF for Top-N Recommendations.

Chapter 6

Conclusions

We successfully achieved the results for W2V, FT and GloVe recall rate for top-N recommendations while TF-IDF is consuming a great deal of time for forming the feature vectors for just 15% of the whole data. It can be concluded that in terms of performance, FT works better than W2V and GloVe as higher recall rate is obtained and in terms of time complexity, GloVe is time efficient compared to FT and GloVe.This was all about single model. In multi-modal, as mentioned in section 5.4.3, the table 5.6 states FG attains highest accuracy of 67.Additionally, the idea of LDA-based clustering is functioning fairly well, as we do not have to compare the new report with each and every report, instead we just have to compare it to the reports residing in the top-n clusters which is proved to be significantly time efficient compared to the comparison without clustering.

6.1 Advantages

The proposed approach is using LDA-based clustering with classification. As stated in section 5.4.4, the proposed system is porved to be time efficient. Using LDA-Based clustering, the time for comparison of number of master reports with duplicate report is decreased. The table 5.4 in section 5.4.2 shows the number of MR in each cluster. The proposed system is selecting top-3 clusters. Consequently, the comparison with MR is limited. The time consumed with clustering and without clustering is shown in table 5.5. So, we can conclude that the proposed approach is time efficient.

Additionally, we have overcome the semantic ambiguity by using LDA-based topic modeling, which keeps the words with the same context in similar topic which will maintain the context of the topics which will lead to contextually similar clusters. Also, for feature extraction we have used W2V and FT which also preserves semantic similarities as they are trained to reconstruct linguistic contexts of words.

6.2 Limitations

The proposed approach is found to be time efficient though the recall rate can be improved. In chapter 2, the recall rate referred for other approaches is for lower number of N in the top-N recommendations while the highest results for our approach is for top-2.5k which is a relatively higher figure. In fact, there was a challenge which we wanted to overcome was word sense disambiguation, in which the bug from the bug reports are different but the solution might be the same, this is the challenge which we are still facing because it is really tough to detect this kind of reports.

Appendix A

IEEE Permission to Reprint

In reference to IEEE copyrighted material which is used with permission in this project, the IEEE does not endorse any of the Lakehead University products or services. International or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to https://www.ieee.org/publications/rights/reqperm.html

Appendix B

Submission of SMC Paper

We have written a research paper and also submitted the research paper in 2020 IEEE International Conference on Systems, Man and Cybernetics (SMC) Organizing Committee. Submission no.-259, Efficient Detection of Duplicate Bug Report using LDA-based Topic Modeling and Classification.

Appendix C

Code Samples

The code samples are available at the github link https://github.com/Detection-of-Duplicate-Bug-Repor
Detection-of-Duplicate-Bug-Report.git Or you are welcome to reach the authors at npatel38@lakeheadu.ca, rpatel30@lakeheadu.ca, dshah12@lakeheadu.ca and syakkant@lakeheadu.ca.

Bibliography

[1] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, pp. 429–489, 2002.

[2] J. Sutherland, "Business objects in corporate information systems," *ACM Computing Surveys (CSUR)*, vol. 27, no. 2, pp. 274–276, 1995.

[3] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pp. 35–39, ACM, 2005.

[4] M. Fischer, M. Pinzger, and H. Gall, "Analyzing and relating bug report data for feature tracking," in *WCRE*, vol. 3, p. 90, 2003.

[5] R. J. Sandusky, L. Gasser, and G. Ripoche, "Bug report networks: Varieties, strategies, and impacts in a f/oss development community.," in *MSR*, pp. 80–84, IET, 2004.

[6] L. Hiew, *Assisted detection of duplicate bug reports.* PhD thesis, University of British Columbia, 2006.

[7] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pp. 52–61, IEEE, 2008.

[8] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *2012 16th European Conference on Software Maintenance and Reengineering*, pp. 385–390, IEEE, 2012.

[9] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering*, pp. 499–510, IEEE Computer Society, 2007.

[10] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 45–54, ACM, 2010.

[11] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *2010 Asia Pacific Software Engineering Conference*, pp. 366–374, IEEE, 2010.

[12] J. Zou, L. Xu, M. Yang, M. Yan, D. Yang, and X. Zhang, "Duplication detection for software bug reports based on topic model," in *2016 9th International Conference on Service Science (ICSS)*, pp. 60–65, IEEE, 2016.

[13] P. N. Minh, "An approach to detecting duplicate bug reports using n-gram features and cluster chrinkage technique," *Int. J. Sci. Res. Publ.(IJSRP)*, vol. 4, no. 5, pp. 89–100, 2014.