Mining GitHub Issues for Bugs, Feature Requests and Questions

by

Marvi Jokhio

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Mining GitHub Issues for Bugs, Feature Requests and Questions

by

Marvi Jokhio

Supervisory Committee

Dr. Neil Ernst, Supervisor
(Supervisor)

Dr. Jens Weber, Departmental Member
(Departmental Member)

Supervisory Committee

Dr. Neil Ernst, Supervisor
(Supervisor)

Dr. Jens Weber, Departmental Member
(Departmental Member)

ABSTRACT

The maintenance and success of software projects highly depend on updated and bug-free code. To effectively process hundreds of daily new issues in big software projects, tools like issue tracking systems (ITS) play an important role but the critical aspect for issue processing and triaging needs assignment of accurate labels to determine their type (e.g., bug, feature, question and so on). This labelling is a time-consuming and tedious task and hence needs automated solutions. Automatic classification of issues is a challenging task due to semantically ambiguous text which contains code, links, package and method names, commands etc.

In this work, we propose supervised and unsupervised mining techniques for GitHub issues using text only. In the supervised machine learning technique, we show that our model can classify issues in bug, feature, and question classes with 86.7% AUC scores. We also proposed a technique to extract topics from GitHub issues using Latent Dirichlet Allocation (LDA) to analyse the type of development issues faced by developers.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

First and foremost, I would like to praise Allah the Almighty, the Most Gracious, and the Most Merciful for His blessing given to me during my study and in completing this work.

I would like to thank:

Dr. Neil A. Ernst, for his support, mentoring, patience, and guidance throughout this work. Without his advice, encouragement, and guidance, it would not be possible to complete this work.

Dr. Jens Weber, for his support and funding during the initial times of my degree and allowing me to work for a funded project which helped me a lot to learn new things.

My family and especially my beloved husband Salahuddin, for their support and all sacrifices they have made.

DEDICATION

My son M. Daniel Jokhio!

and my beloved elder sister who passed away in this pandemic.

Chapter 1

Introduction

The widespread use of open-source repositories, version control systems, communication forms and defect tracking systems have generated a huge amount of valuable data regarding software development and evolution [1]. Mining Software Repositories is an active area of research nowadays and helps to analyze the rich data to uncover interesting and actionable information about software projects [2]. Software repositories such as source control systems, archived communications between project personnel, and defect or issue tracking systems are used to manage the progress of software projects. Software engineering researchers are recognizing the benefits of mining these repositories to support the maintenance and evolution of software systems, improve software reusability, and empirically validation of novel ideas and techniques [3]. The advancements in natural language processing (NLP) techniques and high computing capabilities have introduced new ways to conduct empirical studies in software engineering. Many state-of-the-art (SOTA) NLP techniques are proved to be useful in project management activities. For example, the text classification techniques could be applied to data extracted from software repositories and discussion platforms to design language models for efficient information detection.

The optimization of QA activities needs timely and appropriate management of every software artifact and information but paying in-depth attention to each file is very time consuming and unaffordable especially in the case of big projects. Current tools used for information handling need improvements because of their inability to process and extract useful information from unstructured data like human language.

The management and tracking of software issues play an important role in software maintenance and evolution to meet users' incipient needs [4]. The users raise their problems by reporting issues faced during its usage or maintenance. These

issues may report bugs, feature requests, change requests, documentation updates, code refactoring and simply the discussion or request for help. The efficient processing and resolving of these issues requires filtering, issue labelling, prioritization, and classification so that they could be assigned to the right developer or team accordingly.

Issue tracking systems (ITS) support these tasks by providing functionalities such as issue registration and filtering, time tracking, tickets priority management, workflow oversight etc. Thus, ITS tools help keep the software maintenance cost low [5]. However, especially in popular projects, tens or hundreds of issues are reported daily. A Mozilla developer, as quoted by Anvik et al., mentions that "Every day, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" [6]. This complicates the issues management activities, resulting in heavier workloads for developers [7][8][9].

1.1 Motivation and Problem Statement

In GitHub, user behaviors that are related to issues are of high importance relative to all user behaviors, which indicates that issues play a very important role in open-source project development [9]. The efficient response and resolution of opened issues not only determined the level of project maintenance but also attract passionate external contributors. GitHub provides a straightforward process of submitting new issues which only requires a summary, including a title and a short optional description. This easiness however fascinates new inexperienced contributors to participate in open-source projects but also causes a bulk of newly opened issues every day for certain projects. Most of the time external contributors who open issues assign a specific category to the report but due to lack of in-depth project understanding or contribution guidelines, they assign wrong and vague categories. Therefore, in many projects, the core team of project maintenance has to manually reassign the correct label or category to incoming issues so that they can be assigned to appropriate developers on time.

"Developers are faced with the challenges of semantically understanding and classifying GitHub issues for projects in production which can slow down the process. Therefore, there is an extreme need for an automated method to analyze the contents of issue reports and automatically assign a label to them".

1.2 Research Questions

RQ1: How do the GitHub issue reports differ for bugs, features, and questions based on their content features?

Rationale: The intent is to discover, compare and analyze the content for the 'bugs', 'feature request' or 'questions' category.

RQ2: Can we automatically classify software issues in a bug, feature, or question category only using title and description? If yes, which algorithm or families of algorithms perform the best?

Rationale: The title and description are two main features in every issue tracking system. The three labels bug, feature, and question are the most established and frequent categories used in every next project [10]. The focus is to discover a technique that only uses natural language as input to extract features for classifying the issue reports with the best possible accuracy in order to find a very generalized solution to the discussed problem. Therefore, the interest is to compare and analyze the performance of different machine/deep learning algorithms. Also, find out what combination of technique and model produces the best results.

RQ3: What type of issues are faced by software developers?

Rationale: Our intent here is to discover latent topics from the text of GitHub issues using topic modeling techniques like LDA.

1.3 Related Work

In [11], it is shown that the categorizations of issue types have a huge effect on defect prediction. From five projects, 7041 issues were validated manually and the analysis of the classification was given. The findings show that every third issue is mislabeled as a defect in the issue tracking systems. This incorrect labeling causes a bias in the defect prediction models because 39% of the files are misclassified as having a defect due to misclassified issues that are linked to updates in the version control systems. The work in [12] confirms the results presented in [11]. In addition to this, [12] also proves that misclassifying issues can negatively impact defect prediction data. Similarly, impacts of mislabeling the defect predictions on bug localizations have been studied in [13], [14] and [15]. Researchers use supervised and unsupervised machine learning methods to address the problem of mislabeling in issue systems.

The approach proposed in [16] suggests using the issue description as input. The

input is preprocessed by tokenization, spacing the camel case characters, and stemming followed by building a Term Frequency Matrix (TFM), which includes the raw term frequencies for every term of issue. The TFM does not directly describe the features which are used as input for the classification techniques. Instead, this work first selects asymmetrical uncertainty attribute for identifying the related features. The classification algorithms used are Naive Bayes (NB), Alternating Decision Trees (ADT), and Logistic Regression (LR). Several other researchers use the TFM to define the features. The technique discussed in [17] proposes an automated approach based on fuzzy set theory for labeling an issue as a bug or a request. The proposed approach is the same as the NB classifier but a different scoring function has been utilized. The TFM of the issue titles have been used as input for Support Vector Machine (SVM), NB, or LR classifiers as proposed in [18]. A TFM variant with a fixed word set is proposed in [19], in which 15 non-bug issue-related keywords list (e.g., improvement, enhancement, refactoring) is used. The term frequency is calculated based on the description and title of the issue. This variant of TFM has been provided as an input to classifiers such as Random Forest (RF), SVM, and NB. The work in [20] did not use TFM frequencies, rather they proposed to use a Classification Association Rule Mining (CARM) based approach, using 60 keywords as binary features. In [21], the title and description of the issues as input were used for LR and RF classifiers by using an Inverse Document Frequency (IDF) of n-grams based technique. An approach combining the issue title TFM with structured issue information, e.g., severity, priority, reporter, assignee, and component is proposed in [22]. The approach uses an NB classifier trained on the TFM to classify the titles into three classes namely low (i.e., non-bug), middle (i.e., uncertain decision), and high (i.e., a bug). A Bayesian Network (BN) is trained by using the features generated by combining structural information and NB prediction. The BN predicts two classes i.e., a bug or not a bug. A Latent Dirichlet Allocation (LDA) based topic modeling approach was proposed in [23]. The Decision Trees (DT), LR, and NB algorithms are proposed for classification. The title, description, and discussion are used to calculate the topic-membership vectors through LDA. The topic-membership vectors are then used as a feature to train the classification algorithms. A Hierarchical Dirichlet Process (HDP) or LDA-based topic modeling approach to derive the features is proposed in [24]. The case study discussed in [24] suggests that LDA performs better as compared to HDP. A similar process as discussed in [23] is used to find the topic-membership vectors to be used as features is used. The classification algorithms used are ADT, LR, and NB. Word

embedding of the title and description of the issues as features for training a Long Short-Term Memory (LSTM) network is proposed in [25].

Ticket Tagger [26] is proposed as a recommendation system for issues classification on GitHub. This tool works on the fastText Facebook AI Research algorithm [27] and can be directly integrated into GitHub. The fastText algorithm calculates the n-grams based feature representation from the input text. The features are then used for training a neural network, which performs the text classification task.

The work in [28] considers the difference in structure between title and description of the issue and train different models. The issues containing the null pointer exception (which can indicate an issue as a bug) are detected manually. The prediction performance is improved slightly because of the structural information about the issues used in the model training. The results also show that a null pointer-based description of the issue is not useful in improving the accuracy.

Table 1.1 shows the summary of whole literature review in a systematic way along with performance measures.

Ref #	Classification Type and classes	Data	Technique and classifiers	Performance Measures
[16]	Binary (bug, non-bug)	GitHub (Mozilla, Eclipse, JBoss)	TFM with 20, 50, and 100 term features (LR, ADTree, NB)	Accuracy =>Mozilla -77% , Eclipse – 82%, JBoss – 82%
[17]	Binary (bug, non-bug)	HTTPClient, Jackrabbit, and Lucene	Fuzzy set theory , (LDA)	Accuracy =>87%, 83.5% and 90.8% and F1 =>0.83, 0.79 and 0.84
[18]	Binary (bug, non-bug)	HTTPClient, and Lucene	TFM from only titles (summaries) from JIRA (LR, NB, SVM, LDA)	Accuracy =>71.4%, 72% and F1 =>78.4%, 64%
[19]	Binary (corrective or perfective)	AspectJ, Tomcat, SWT	The feature set of 15 keywords , (NB, SVM, Random Trees)	Avg. Accuracy of all projects=>89.6% to 92.9%
[20]	Binary (bug, non-bug)	HTTPClient, AspectJ	Classification Association Rule Mining (CARM) with 60 binary terms	Accuracy=>84% to 89%
[21]	Binary (bug, non-bug)	HTTPClient, Jackrabbit, Lucene, Cross Project	N-gram IDF, Topic Modeling (N-gram IDF-based models)	F1=>0.67 - 0.69, 0.62 - 0.65, 0.68 - 0.73, and 0.65 - 0.67

[22]	Binary (bug, non-bug)	Mozilla, Eclipse, JBoss, Firefox, OpenFOAM	Data grafting algorithm (combining title TFM with other structured data from the issue)	F1=>81.7% for Mozilla, 80.2% for Eclipse, 93.7% for JBoss, 79.5% for Firefox, 85.9% for OpenFOAM.
[23]	Binary (bug, non-bug)	HTTPClient, Jackrabbit, and Lucene	Topic modeling on title, description, and discussion using LDA to build a set of topic membership vectors , (DT, NB, and LR)	F1 score varies between 0.66-0.76, 0.65-0.77, and 0.71-0.82 for HTTPClient, Jackrabbit, and Lucene project respectively
[24]	Multiclass (BUG, IMPR, RFE, TASK, TEST)	Combined the HTTPClient, Jackrabbit, and Lucene from JIRA	Topic modeling using Hierarchical Dirichlet Process (HDP) and LDA	ROC =>0.768 (HDP), 0.807 (LDA) F1=>0.601 (HDP), 0.641 (LDA)
[25]	Binary (bug, non-bug)	HTTPClient, Jackrabbit, and Lucene and cross-project	Word embeddings , (LSTM)	F1=>0.771, 0.717, 0.757, 0.746
[26]	Multi-class (Bug, Enhancement, Question)	Closed issues from 12, 112 Heterogeneous GitHub projects	Word n-gram methods with max n=3, (FastText)	F1=>83.1% (Bug) , 82.3% (Enhancement), 82.5% (Question) and AvgF1=>82.6

[29]	Binary (bug, non-bug)	HTTPClient, Jackrabbit, and Lucene	DTM from summary , (NB, LDA, KNN, SVM, DT, RF)	W-AvgF1=>∼63% , ∼76% and ∼79%
[28]	Binary (bug, non-bug)	HTTPClient, Jackrabbit, and Lucene, Rhino, Tomcat5	Issues with "null pointer exception" labeled as bugs, (NB, RF, FastText (FT), FastText + Autotune (FTA))	F1=>CV_{ALL} 0.809 and CV_{BUG} 0.643, FTA+UV 0.783

Table 1.1: Literature review synthesis matrix.

1.4 Report Organization

- **Chapter 1** provided a brief introduction about mining software repositories, software issues tracking systems. The motivation of the project, problem statement, research questions, and objectives are discussed. This chapter also provides the related work done in the same domain.
- **Chapter 2** discusses the methodology and experiments design. The exploratory data analysis is thoroughly discussed.
- **Chapter 3** provides detailed discussion and performance comparison of the machine and deep learning algorithms using precision, recall, F-score, and AUC score.
- **Chapter 4** discusses the results of topics extracted from GitHub issues.
- **Chapter 5** concludes the report, discusses threats to validity and provides suggestions for the future work.

Chapter 2

Methodology

Figure 2.1 shows the methodology used to build a machine/deep learning model, starting from data acquisition and analysis to model validation. The results from one phase are used as input for the next phase. After the text normalization phase, we perform content feature analysis in order to find the answer for ***RQ1*** but is not used for any other phase. In the data encoding phase, three different types of experiments based on each technique are performed after the tokenization. In each type of experiment, we tested a couple of machine and deep learning models. The result of each encoding technique (TF-IDF, Word Integer Encoding, and GloVe word embeddings) is used as input for the specific ML/DL learning models. Finally, the performance measure is recorded for each experiment using precision, recall, f1-score, and area under curve values. In the following sub-sections, we will discuss each experiment in detail.

2.1 Data Acquisition

The dataset is extracted from GitHub issues from heterogeneous projects and is prelabeled by GitHub users. It contains a trainset with 450,000 rows and a test set of 30,000 rows. The trainset has 3 attributes *title, body* and *label*. The title attribute contains the short summary of the issue, the body attribute contains the description of the issue, and the label attribute is the target class attribute having 3 classes i.e., 0-Bug, 1-Feature, 2-Question.

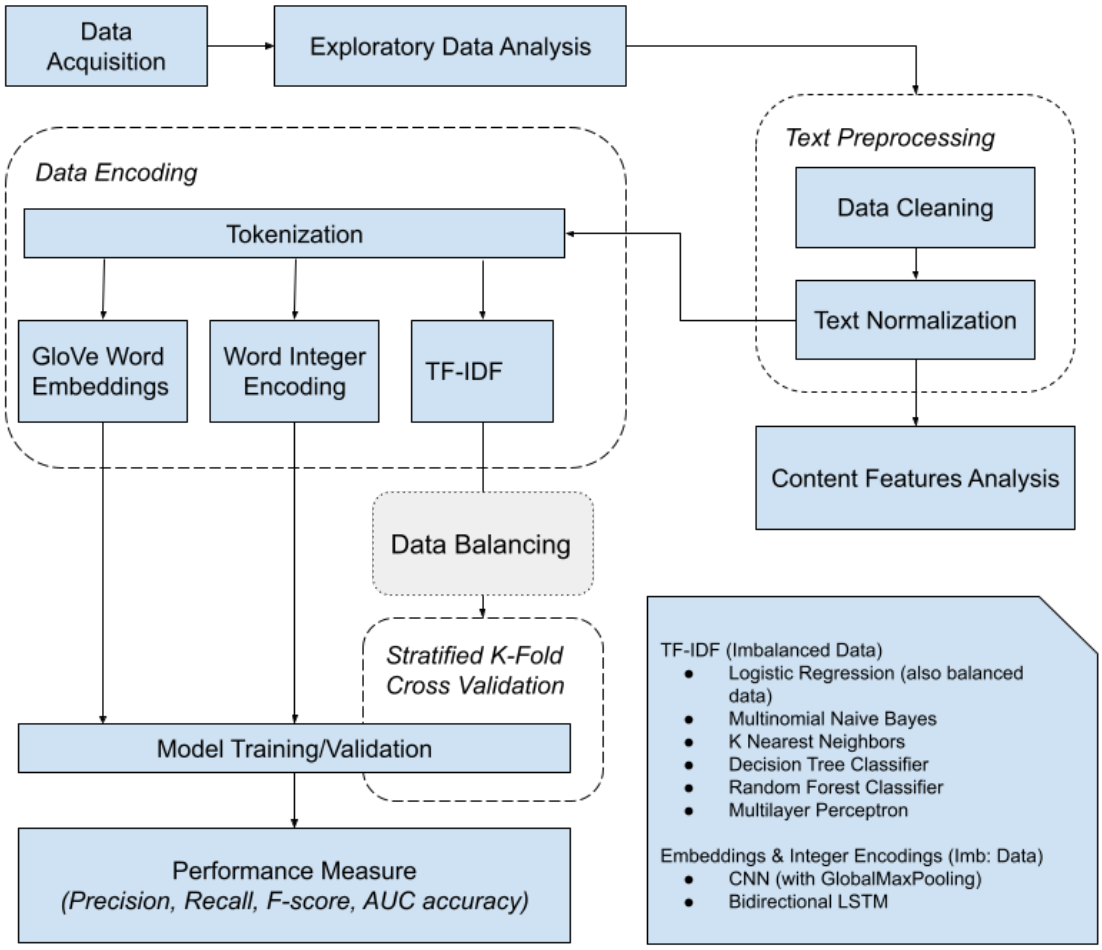

Figure 2.1: Methodology for detecting type of issue reports.

2.2 Exploratory Data Analysis

In this phase, we performed various data exploration techniques which are thoroughly discussed in the following subsequent sections.

2.2.1 Distribution Analysis

Target Class Distribution

The results of supervised machine learning are highly dependent on the quality of input data. The quality includes the nature of data either structured or unstructured, the correct labeling, and symmetrical distribution for all classes. In our case, the data

is imbalanced as well as unstructured as we are using only textual content of issue reports. The trainset contains 20,0481 data samples for bugs, 20,7318 for features, and 42,201 for questions as shown in Figure 2.2. The distribution for bugs and features classes is almost the same but on contrary, the amount of data for the questions class is very low as compared to the rest of the two classes. It should be noted here that in

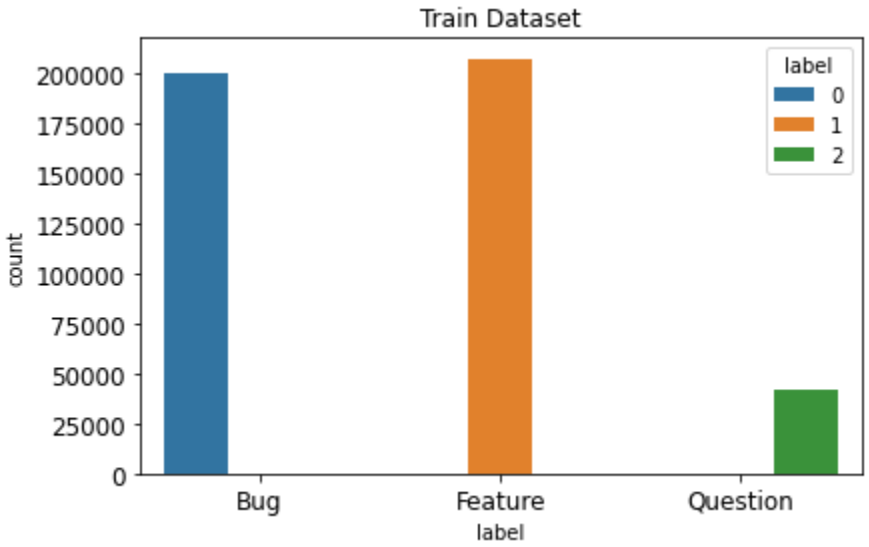

Figure 2.2: Target class distribution for dataset.

all next phases of the research experiments we combined two columns *title* and *body* in one column named as *text*.

Distribution of Top Frequent Words

Table 2.1 shows the frequency count of the top 10 common words for each target class along with their frequency count. The words *'git'* and *'hub'* occurred with extremely high frequencies in all three classes are not included in the list. The list of top words shows that there are some words that are common in all three classes like the word *'version'* because of the fact that the dataset is extracted from the GitHub platform which is mostly used for version control for the software repositories. However, there are some words that are common in two of the target classes. For example, the words *'get'* and *'issue'* are both present in bug and question class. The words *'use'*, *'like'*, *'data'* and *'name'* are common in feature and question class.

Moreover, the word *'error'* is the top most common word for both bug and feature categories for the fact that developers use this word more often in issues either for

S#	Bug	Features	Question
1	error: 129473	add: 74666	error: 19501
2	java: 123669	use: 43293	get: 16936
3	version: 89338	need: 35644	version: 16439
4	test: 81272	like: 35299	use: 14798
5	line: 77120	new: 34946	data: 14013
6	com: 74846	data: 33343	using: 13322
7	get: 68226	name: 32911	code: 12875
8	issue: 65483	support: 32214	name: 12207
9	tat: 65369	version: 32050	issue: 11952
10	content: 62490	code: 31216	like: 11595

Table 2.1: Frequency count of top 10 common keywords in target classes.

reporting a bug or for asking a question about a bug.

For the target class *'bug'*, we can see that the second top word is *'java'* which could mean that the dataset contains more issues about projects developed in java or it could be possible that new developers when programming in java language, face most bugs due to the fact that it is a tightly coupled programming language.

The list of top common words for a bug, feature, and question category seem to be very close enough to a common perspective about words that can generally define these classes. Furthermore, the word clouds in Figure 2.3, provide the abstract reflection of words in each target class of the dataset.

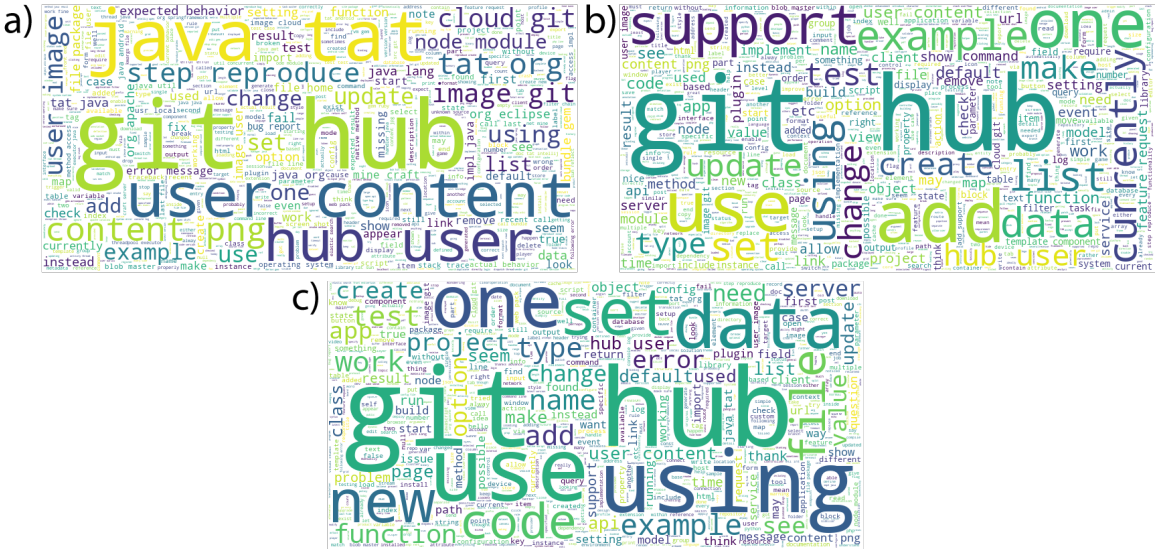

Figure 2.3: Word Clouds for top frequent words for a) bugs, b) features and c) questions in dataset.

The words *'http'*, *'git'*, and *'hub'* were filtered out in further phases due to their extremely high frequency and presence in all three classes and therefore were considered as noise.

Meta-Morphological Distribution

Words play a critical role in sentences and their meaning. The analysis of morphological features is important to see their proportion in the dataset. We have tried to find the meta-morphological symmetry between target classes in a given training set and also tried to find the same facts between train and test set.

Figure 2.4 depicts the proportional distribution of word count amongst each class in trainset versus train and test sets. The plot left side shows that data samples with

word count approximately up to ∼200 have a higher proportional density in each target category. However, this distribution decreases with an increase in word count more than ∼200. Similarly, the plot on the right side shows the same proportional fact for data samples in the train versus test set.

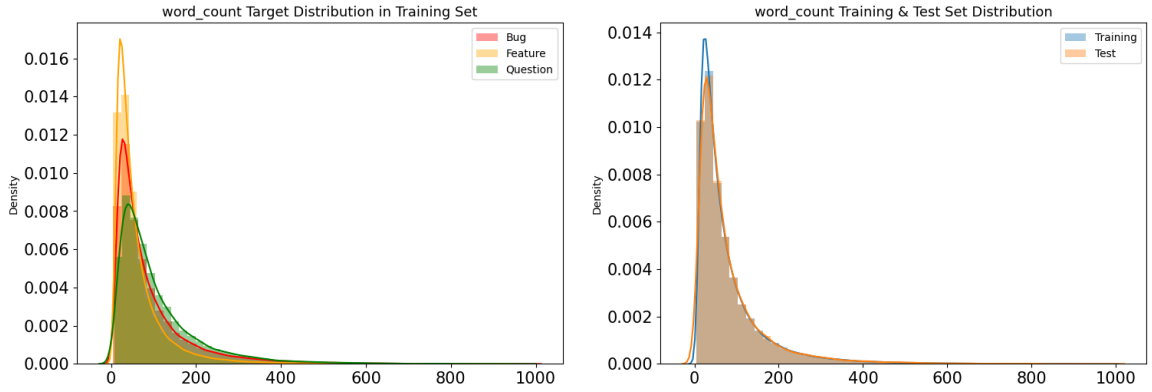

Figure 2.4: Distribution of words count.

Figure 2.5 shows the proportion of unique words amongst all three classes and between train and test sets. Since the distribution of unique words between train and test set is much more symmetrical as compared to the target classes in the left-side graph having a very small imbalance that could be neglected here.

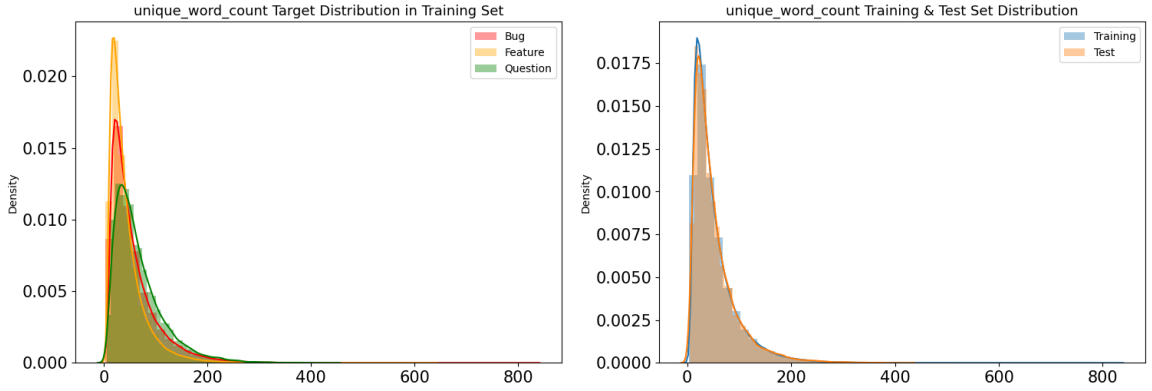

Figure 2.5: Distribution of unique words.

In Figure 2.6, the left plot shows that samples having stop word count up to ∼100 have higher proportional frequency distribution in each target category but it suddenly decreases if the count jumps over 100 approximately. Similarly, the same fact holds for a train versus test set in the right side plot.

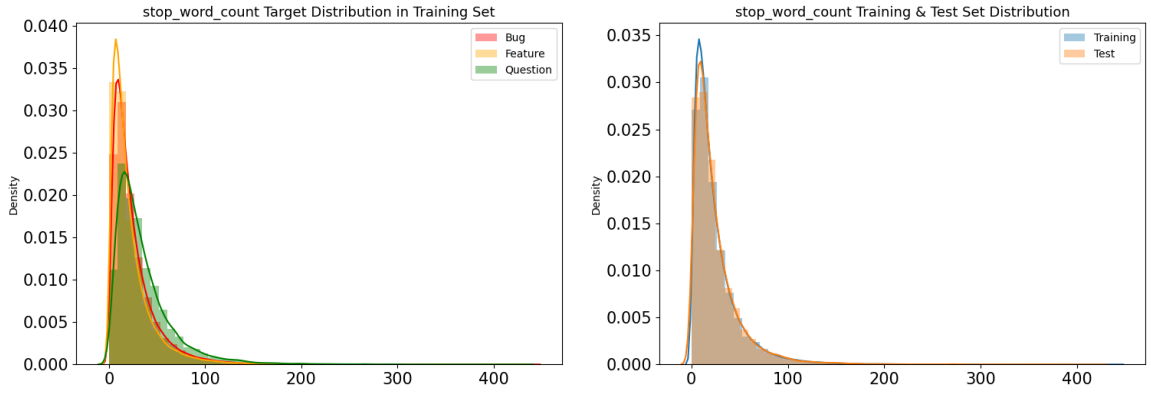

Figure 2.6: Distribution for stop words count.

The meta-morphological features for mean word length and character count are shown in shown in Figure 2.7 and 2.8 respectively. It is clear from plots that the data mostly contains samples with their *mean word length* between ∼60 to ∼120 irrespective of category and the same holds for data samples in train and test sets. Similarly, talking about the distribution of *character count* we can see that plots show the higher distribution for data cases with character count ranging from ∼50 to ∼400.

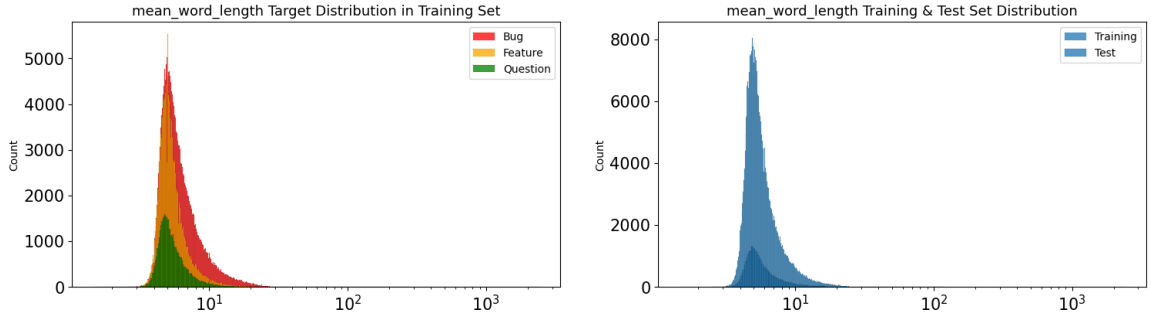

Figure 2.7: Mean word length distribution.

From the figures above we can see that both plots show good and somehow identical symmetry for features either in target classes or in train/test sets.

2.3 Text Preprocessing

We are dealing with unstructured data in the form of written natural language so phases of data preparation for model training are extremely critical and will directly

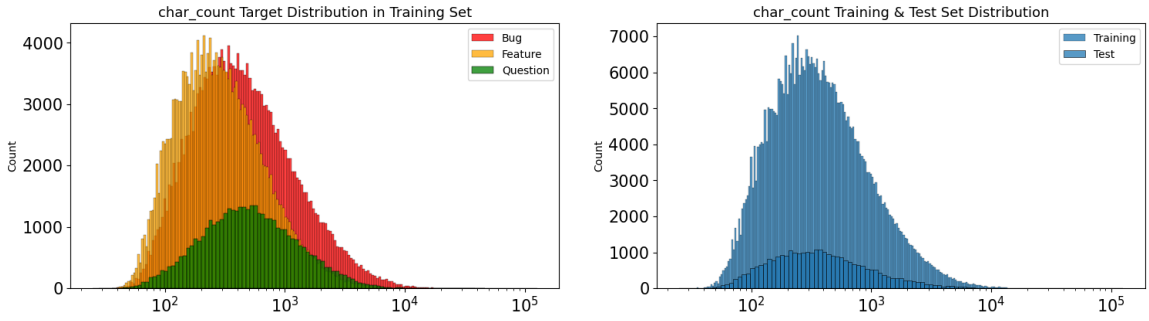

Figure 2.8: Char count distribution.

affect the final phase of model evaluation.

2.3.1 Data Cleaning

The data from GitHub issues contain noise as it is collected through web scraping and is in unstructured form. The cleaning of this data is challenging because it contains punctuations, special characters, resource links, code snippets, comments in code, namespaces, function names, execution commands, etc. Table 2.2 shows a few samples of data before it is cleaned.

To clean the noise from text we use regular expressions to filter out unwanted data. We performed following operations on data to cleanse it;

- Lowercase
- Removed hash tags
- Removed forward / to split URLs for preserving words inside urls
- Removed $<!--$ and $-->$ but keeping anything inside it to keep important tokens
- Removed punctuations
- Removed newlines (\n)
- Removed returns (\r)
- Removed numbers
- Removed words containing numbers

index	text	label
0	y-zoom piano roll a y-zoom on the piano roll would be useful.	1
1	buggy behavior in selection ! screenshot from 2016-02-23 21 27 40 https://cloud.githubusercontent.com/assets/9442944/13 260424/cf086b72-da74-11e5-8584-68534b3fd9a0.png \r	0
2	auto update feature hi,\r \r great job so far, @saenzramiro ! : \r \r an auto update feature would be nice to have.\r or alternatively a menu button to check for the latest version manually.	1
3	filter out noisy endpoints in logs i think we should stop logging requests to:\r - __health \r - __gtg \r - favicon.ico \r	1
5	script stopped adding video's a recent change in the youtube layout broke the script.\r probably caused by element names being altered.	0
14	collectionprovider support pagination it would be very helpful if support for pagination could be added. https://laravel.com/docs/5.2/pagination we'd like to be able to use eloquent models without needing to load the full dataset from the table.	2
21	app breaks when changing months after the actual i can browse months from before the actual. like from august to july, june..\r changing from june to july and august works normal. if i try to advance to september, the app freezes.\r \r logcat.txt https://github.com/simplemobiletools/simple-calendar/files/ 421272/logcat.txt \r	0
71	printing members in network \r \r import scala.collection. mutable.arraybuffer;\r \r \r class network {\r class member val name: string {\r var contacts = new arraybuffer member \r }\r \r private val _members = new arraybuffer member \r \r def join name: string = {\r val m = new member name \r _members += m\r m\r }\r \r \r override def tostring = {\r s\contacts in network ${_members}\\r }\r }\r \r val chatter = new network\r val myface = new network\r \r \r val fred = chatter.join \fred\\r val wilma = chatter.join \wilma\\r fred.contacts += wilma\r \r println chatter \r \r	2

Table 2.2: Sample issues before data pre-processing.

- Removed extra spaces

The cleaning process above didn't remove the text contained inside code snippets for example namespaces, class names, method names, and also keep text inside code comments and URLs. In programming practices, the words used in the method names, namespace, and the commands reflect the purpose of the task being resolved. Therefore, removing them can cause an ML model to lose important features. Therefore, we used the cleaning technique to keep such data as much as possible.

2.3.2 Normalization

In this step, we used the word segmentation technique for segmenting name identifiers, camel case method names by using *wordsegment* python library. Next, the stop words are removed using the Natural Language Toolkit (NLTK) list of default stop words. Finally, to convert the words to their base form we did the lemmatization through *WordNet* which is an open-source lexical database for the English language.

Table 2.3. shows obtained results for data samples in Table 2.2. after applying cleaning and normalization techniques discussed above.

index	text	label
0	zoom piano roll zoom piano roll would useful	1
1	buggy behavior selection screenshot http cloud git hub user content png	0
2	auto update feature great job far saenzramiro auto update feature would nice alternatively menu button check latest version manually	1
3	filter noisy endpoint log think stop logging request health gtg favicon ico	1
5	script stopped adding video recent change youtube layout broke script probably caused element name altered	0
14	collection provider support pagination would helpful support pagination could added http ravel like able use eloquent model without needing load full dataset table	2
21	app break changing month actual browse month actual like august july june changing june july august work normal try advance september app freeze log cat txt http git hub com simple mobile tool simple txt	0
71	printing member network import scala collection mutable array buffer class network class member val name string var contact new array buffer member private val member new array buffer member def join name string val new member name member override def tostring contact network member val chatter new network val face new network val fred chatter join fred val wilma chatter join wilma fred contact wilma println chatter	2

Table 2.3: Sample issues after data pre-processing.

2.4 Content Feature Analysis

To discover and study the common issue publishing behavior that the developers use while opening the new issues on GitHub, we manually analyzed some of the data samples to find the most common factors present in every next issue. These factors include usernames, URL/URI, HTML tags, HTML comments, code snippets, images.

We found the average percent frequency of these content features used in all classes of issues in the given dataset. Figure 2.9 depicts this in more visually understandable form. Figure 2.9 clearly shows that URL is most frequently used feature in almost

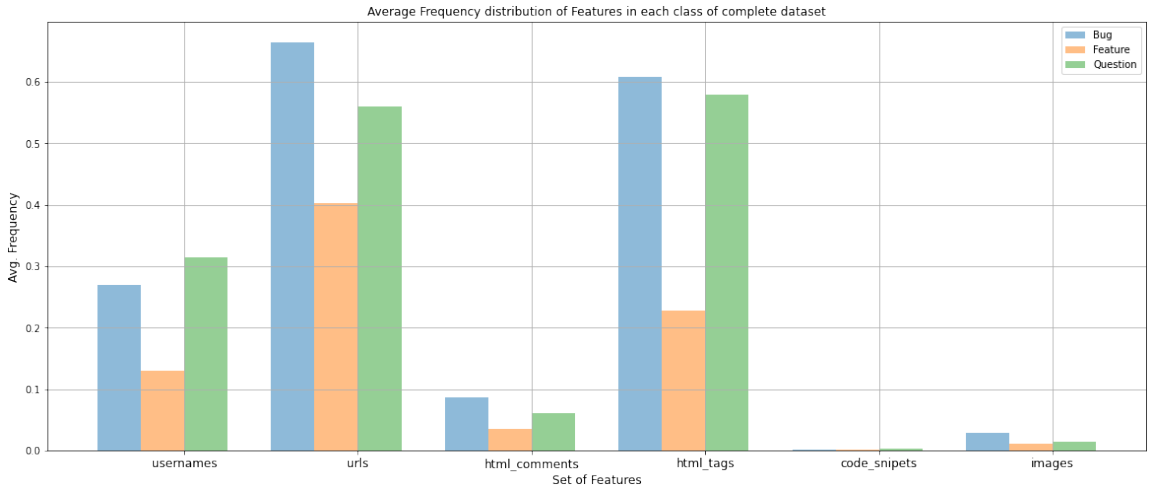

Figure 2.9: Average frequency distribution of content features in all dataset classes.

every issue report because it is the backbone of this cloud-based version control system where every single repository is identified by a unique URL. The HTML tags are the second most prominent factors used in issue reports and the reason for this may be that the users use HTML tags to format the text, insert URLs links, images, or even code snippets as needed. The third most common factor is usernames because users also tag other contributors or developers while opening an issue. The use of images and code snippets seems to be very low as compared to rest of the factors which is apparent because the use of screenshots and code snippets is main factor in platforms where developers ask for coding problems such as Stake Overflow etc.

RQ1: How do the GitHub issue reports differ for bugs, features, and questions based on their content features?

The distribution analysis of features discussed above concludes that users use almost the same pattern for opening an issue representing a bug or a question. Whereas,

the pattern of posting an issue for the request of a new feature, improvement, or change doesn't necessarily follow the same style of patterns as for a bug or a question.

The distribution analysis of features discussed above concludes that pattern of opening issues for bugs and questions is almost same

2.5 Data Encoding

To extract the meaning of words in a sentence is a very overwhelming task since the machines cannot understand the words. We need to encode the sentences in a form that is suitable for machines to process and extract the sense. This needs some operations to be done to convert sentences into number sequences. There are various techniques available in natural language processing (NLP) to accomplish this task.

We used the following encoding techniques for our experiments to see which one performs best to our problem and dataset.

2.5.1 Tokenization

The first step required before transforming the text into machine understandable form is to split it into small units called tokens using a delimiter and the process is called tokenization. The tokens can be words, characters, or sub-words. It is the backbone for building the vocabulary from the corpus as the set of unique tokens in the corpus or by considering top N frequently occurring words. We performed word-level tokenization on our normalized corpus using space as a delimiter.

The following subsection will discuss how tokenization is further used in each case of the encoding technique used in our experiments.

2.5.2 Term Frequency Inverse Document Frequency(TF-IDF)

TF-IDF defines the importance of a term by taking into consideration the importance of that term in a single document and scaling it by its importance across all documents. We used this technique to build a matrix of TF-IDF features with a maximum 100K vocabulary size which includes uni-grams, bigrams, and trigrams. The vocabulary didn't take into account the terms which occurred in less than 10 documents and more than 80% of all documents. In this encoding scheme we conducted two separate streams of experiments;

1. **with original imbalanced data** on following ordinary machine learning algorithms;
   (a) Logistic regression
   (b) Multinomial naïve bayes
   (c) K- Neighbour classifier
   (d) Decision tree classifier
   (e) Random forest
   (f) AdaBoost classifier
   (g) Multilayer Perceptron or simply artificial neural networks
   (h) Ridge Classifier
   (i) SGD Classifier
2. **with re-sampled data** using four oversampling techniques i) Random over sampling, ii) SMOTE (Synthetic Minority Over-Sampling Technique), iii) ADASYN (Adaptive synthetic sampling) and iv) SMOTE Tomek to handle imbalanced data. We only used the logistic regression for training in this case.

2.5.3 Integer Encoding

This is done by assigning a unique number to every word in extracted vocabulary from the data. A sentence is then converted into a vector of fixed length number sequences and specific dimensions. The meaning is reflected by the direction of the vector in one of the defined number of dimensions. For example, for the data sample “add applicable tax section provide date satisfied applicable option”, we have the integer encoding as [11, 1656, 2124, 365, 274, 164, 5195, 1656, 66] and the corresponding vector of length 200 and post padding as;

[11 1656 2124 365 274 164 5195 1656 66 0]

We performed 4 experiments with different combinations of input vector length and embedding dimensions and vocabulary size but only used the LSTM model for training.

#	Model	Vector Length	Train/ Val: Size	Voc: Size	Emb: Dim	%Voc Matched
1	CNN	200	360K/90K	97196	200	69.17
2	LSTM					
3	CNN				300	81.12
4	LSTM					

Table 2.4: Parameter settings for experiments with GloVe pretrained embedding models.

2.5.4 Word Embeddings

Word embedding is the representation of words in the form of a dense vector of floating-point values that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning [30]. An embedding is with trainable parameters which are weights learned by the model during the training process, in the same way, a model learns weights for a dense layer. The dimensions of embeddings are specified depending on the size of the dataset. The dimensions of embeddings range from 8 for smaller datasets and can go up to 1024 for large-sized datasets. A higher dimensional embedding can capture fine-grained relationships between words but takes more data to learn. Some well-known word embedding models include Tomas Mikolov's Word2vec, Stanford University's GloVe, GN-GloVe, Flair embeddings, AllenNLP's ELMo, BERT, fastText, and Gensim.

We used Stanford University's GloVe word embeddings in the embedding layer in Convolutional Neural Networks and LSTM with 200 and 300 embedding dimensions. These predefined word embeddings are converted to a corresponding embedding matrix with an entry at index 'i' is the pre-trained vector for the word in index 'i' in our vectorizer's vocabulary.

The results were pretty good which will be discussed in the upcoming chapter. The Table 2.4 shows the model input parameter settings for 4 experiments. The column 'Vocabulary Size' shows the vocabulary extracted from the trainset whereas '%Voc Matched' contains the percentage of extracted vocabulary matched with the vocabulary of GloVe's pre-trained word embeddings either in 200 or 300 dimensions.

2.6 Model Training and Validation

The method for model training and validation depends on the data encoding technique used for the experiments. The following sub-sections discuss how models' training and validation are done in each technique we used in this work.

2.6.1 TF-IDF

For training the models with TF-IDF, the stratified k-folds cross-validation technique is used to avoid the overfitting of models. The stratified k-folds cross-validation is a type of K-Fold that returns stratified folds to generate a test set so that it contains the equal distribution or as close as possible. We used default 5 k-folds with random shuffling in all experiments and each fold calculated the 'macro' average for precision, recall, and f1 score for individual target classes i.e., Bug, Feature, and Question. Finally, we calculated the mean accuracy score of 5 k-folds and also the mean of 'macro average' scores for precision, recall, and f-score after every kth-fold.

2.6.2 Integer Encoding

We constructed the LSTM model with sequentially stacked layers with "adam" optimizer and "sparse_categorical_crossentropy" for loss detection. The model has two hidden bidirectional LSTM layers and each layer returns sequences. Finally, the two dense layers with the 'Softmax' activation function in the last dense layer are used. Figure 2.10 shows the model design with stacked layers.

We trained the LSTM model with different combinations of input parameters (i.e., input vector size, dimension size) and datasets. In one set of experiments, we combined the given train and test set in one big dataset and split it into 80-20 for train and test sets, while on the other hand we only used the given trainset for training. The total no: of epochs were 10 in all of the experiments with deep learning models. Finally, the Area Under Curve (AUC) accuracy was calculated after 10 epochs along with training and validation accuracies for performance comparison of models.

2.6.3 Word Embedding

In this stream of experiments, we trained CNN and LSTM models with predefined GloVe word embeddings. The construction of CNN model is depicted in Figure

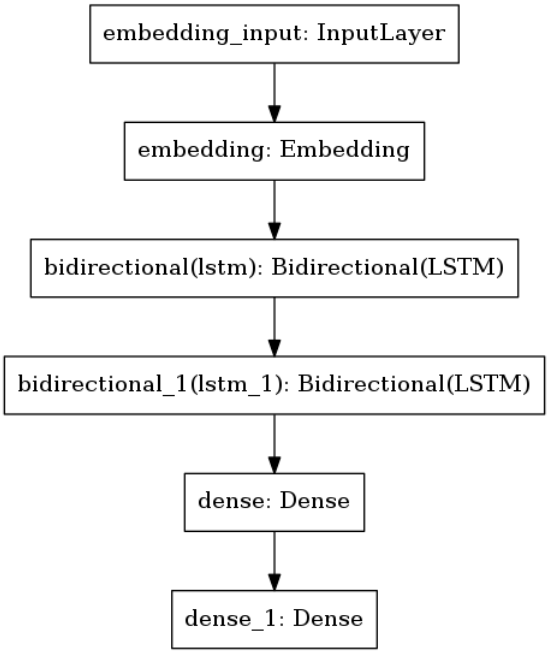

Figure 2.10: LSTM model layers for integer encoding experiments.

2.11 where as the LSTM model construction is exactly same as in Figure 2.10. The performance of each experiment is compared in terms of AUC-score after 10 epochs.

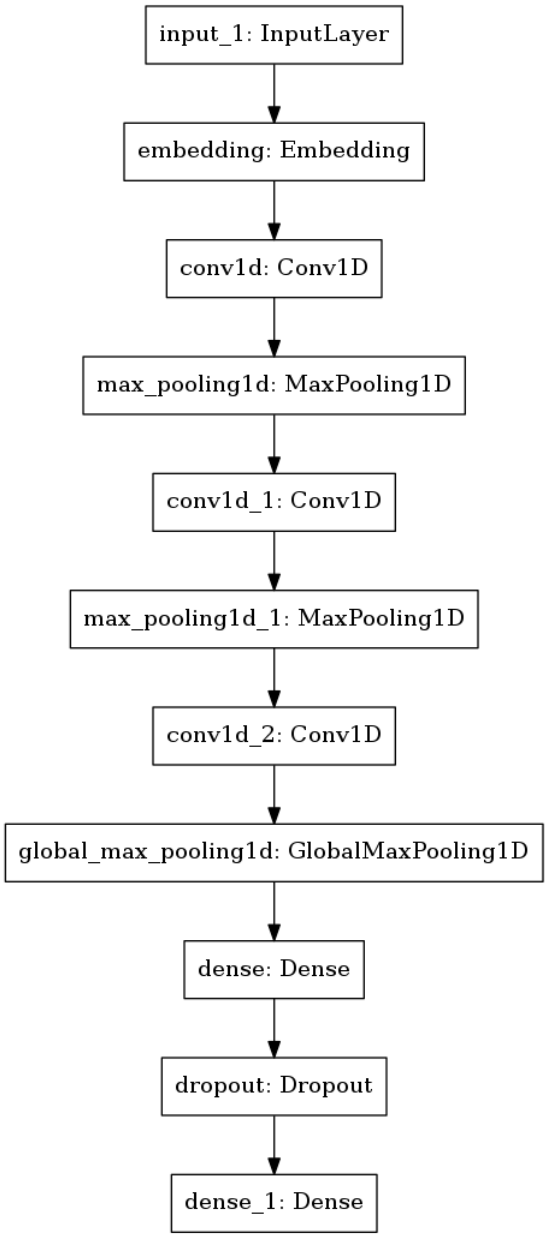

Figure 2.11: CNN model analysis.

Chapter 3

Performance Comparison of Machine Learning Algorithms

This chapter discusses and compares the results obtained from ML/DL models used in the experiments conducted under each technique as discussed in the methodology chapter.

3.1 Results and Discussion

3.1.1 TF-IDF

Imbalanced Data

Before discussing the actual results of the experiments, let's briefly discuss the measures used to evaluate the performance of a machine learning model for a classification problem.

Accuracy is the most common parameter to evaluate the model's performance but since it is just the ratio of correctly classified samples to the total no: of samples and is the best measure when we have a symmetrical dataset. However, for an uneven dataset, it is more useful to look at other parameters to evaluate the performance of the model. High precision means a classifier has a low false-positive rate or in a simple way it can correctly classify samples to their true classes amongst the total no: of predicted samples in those classes. Whereas, recall is the ratio of correctly predicted samples to total samples available in those classes [31].

Both precision and recall have values in the range [0-1]. The F1-score i.e., the

weighted average of precision and recall scores is considered as best for evaluating the model's performance. If any of the precision or recall values are closer to 0, the F1-score drops in a huge amount because more weight is given to lower values in harmonic mean. However, in the case of the multiclass problem each class should be included in the F1-score through computing the macro average precision and macro average recall for each class separately. To accomplish this, we first compute precision and recall for the individual class through the same formula as in the binary case. For example, for a given class k we have

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{3.1}$$

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{3.2}$$

Then, the macro average precision and recall are computed as the arithmetic mean for every single class using the following formula

$$MacroAveragePrecision = \frac{\sum_{k=1}^{K} Precision_k}{K} \tag{3.3}$$

$$MacroAverageRecall = \frac{\sum_{k=1}^{K} Recall_k}{K} \tag{3.4}$$

Eventually, Macro F1-Score is the harmonic mean of Macro-Precision and Macro-Recall i.e.

$$MacroF1 = 2 * (\frac{MacroAveragePrecision \times MacroAverageRecall}{MacroAveragePrecision^{-1} + MacroAverageRecall^{-1}}) \tag{3.5}$$

The good thing about this macro average method is that it provides the same importance to all classes irrespective of their sizes. Therefore, macro F1-score evaluates the algorithm from a class standpoint: high Macro-F1 values indicate that the algorithm has good performance on all the classes, whereas low Macro-F1 values refer to poorly predicted classes [32].

We tried a total of 9 machine learning algorithms using the TF-IDF technique using the default parameter settings and found a few of them performed pretty well in classifying the issue reports. Table 3.1 shows the mean accuracy and also the mean of the macro average precision, the mean of the macro average recall, and the mean of the macro average F1 score for each corresponding classifier.

#	Classifier Name	Mean Accuracy	Mean Precision	Mean Recall	Mean F1-Score
1	Logistic Regression	**77.26**	**71.93**	**63.52**	**65.34**
2	K Neighbors Classifier	50.83	50.86	37.59	34.72
3	Multinomial Naive Bayes	72.45	68.95	54.64	53.77
4	Decision Tree Classifier	66.16	55.19	54.32	54.64
5	Random Forest Classifier	74.00	74.81	54.60	52.25
6	AdaBoost Classifier	68.93	63.30	55.36	56.39
7	Multi-Layer Perceptron	73.29	48.86	53.91	51.26
8	SGD Classifier	**76.79**	**74.25**	**59.81**	**60.44**
9	Ridge Classifier	**76.39**	**72.47**	**61.90**	**63.64**

Table 3.1: Mean accuracy, precession and recall of classifiers for TF-IDF encoding with imbalanced data.

The dataset in our case is a multi-class unbalanced dataset, therefore, we will use the F1-score for classifiers' performance comparison in general but we will also discuss and compare their precision and recall scores for in-depth analysis.

From Table 3.1 we can see that there are three classifiers whose F1-score is above 60 whereas the rest of all are below 57 scores. It shows that these classifiers have a good tendency to accurately classify issue reports in their corresponding classes. The F1-score above 60 is 65.34, 63.64, and 60.44 for Logistic Regression, Ridge, and SGD Classifier respectively. The highest F1-score for logistic regression tells that it performs best when classifying issue reports in their appropriate classes.

Although the SGD and Ridge classifiers have higher precision values i.e., 74.25 and 72.47 respectively, at the same time have lower recall values i.e., 59.81 and 61.90 compared to logistic regression which makes their F1-scores a bit lower. This shows that these two classifiers are very fastidious in classifying the samples in a particular class to which they belong but at the same time missing a good number of other samples from those classes. The scores of logistic regressions reflect the good performance of the classifier. The performance of the rest of the 6 classifiers is clear from their scores in Table 3.1. Figure 3.1 makes the comparison of accuracy, precision, recall, and F1-score of all 9 classifiers clearer using grouped bar plot.

Balanced Data

We applied some oversampling techniques to balance the distribution of classes and see the effect on performance results. Therefore, we tested the best performing classifier

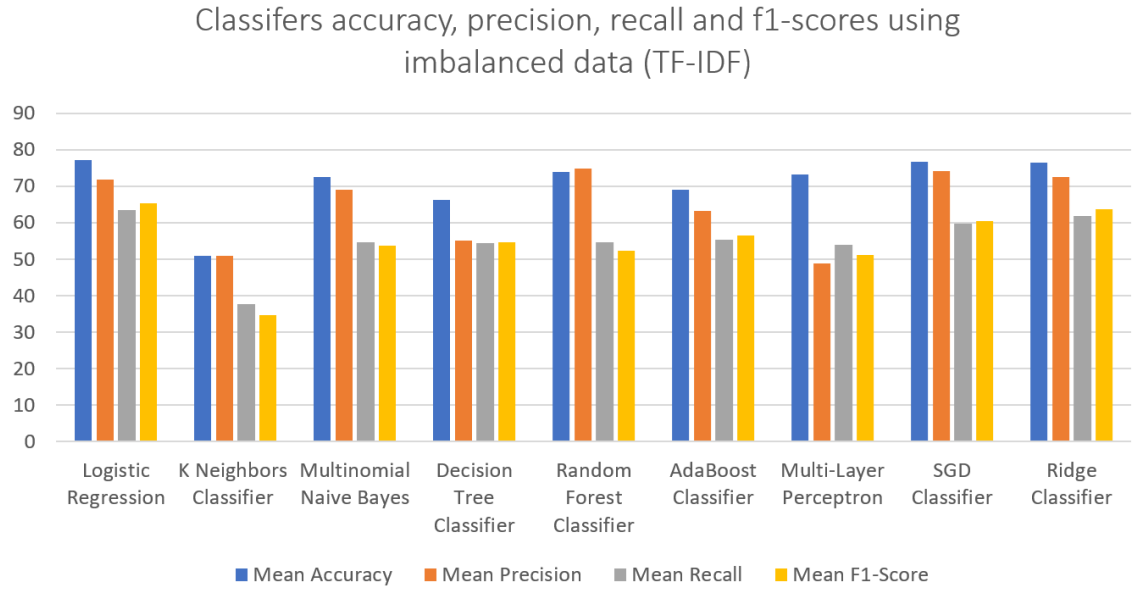

Figure 3.1: Results comparison for TF-IDF encoding technique.

from imbalanced data experiments and we obtained results shown in Table 3.2. The scores in small braces with a positive and negative sign show the difference of measure compared to logistic regression results from imbalanced data in Table 3.1. For all experiments of resampling, the scores for recall and F1 are increased but at the same time, the accuracy and precision have decreased. The resampling using Random Over Sampler has increased recall and F1-score by 5.62% and 0.58% respectively but at the cost of a decrease in precision for 6.91%. However, this difference of precision value is smaller in Random Over Sampler compared to SMOTE, ADASYN, SMOTE

#	Classifier Name	Sampling Method Used	Mean % Accuracy	Mean % Precision	Mean % Recall	Mean % F1-Score
1	Logistic Regression	Random Over Sampler	73.06 (- 4.2%)	65.02 (- 6.91%)	69.14 (+ 5.62%)	65.92 (+ 0.58%)
		SMOTE	72.91 (- 4.35%)	64.80 (- 7.13%)	68.85 (+ 5.33%)	65.62 (+ 0.28%)
		ADASYN	73.15 (- 4.11%)	64.93 (- 7.15%)	68.87 (+ 5.35%)	65.78 (+ 0.44%)
		SMOTE Tomek	72.90 (- 4.36%)	64.78 (- 7.15%)	68.72 (+ 5.2%)	65.56 (+ 0.22%)

Table 3.2: Mean accuracy, precession and recall of classifiers for TF-IDF encoding with balanced data.

#	Model	Dataset	Dictionary Size	Emb. Dim	Vector Size	Train. Acc.	Valid. Acc.	AUC Score
1	LSTM	Combined	497272	20	100	0.9520	0.6771	0.768
2				64	200	0.9732	0.6644	0.766
3		Train	415542	20	100	0.9652	0.7008	0.789
4				100	100	0.9865	0.7053	**0.795**

Table 3.3: Training, validation and accumulative accuracies for Integer Encoding experiments after 10 epochs.

Tomek which shows that the model is better than the rest of the techniques.

3.1.2 Integer Encoding

Table 3.3 shows the results obtained through the integer encoding technique along with data and parameter settings used in experiments. The experiments with only trainset were better than combined data. Experiment no: 4 with embedding dimension and vector length of 100 gave the best accuracy results with AUC accuracy ∼79.5% amongst all of the four experiments.

3.1.3 Glove Word Embeddings

The technique of using pre-trained word embeddings in the embedding layer of deep learning models seems to work better than the rest of the two techniques due to it's better validation and AUC scores.

In Table 3.4, we can see that the LSTM model performed better than CNN in both cases of embeddings dimension length. It is interesting to note that the AUC score for LSTM, in either case, is the same and is equal to 86.7%. To say that both models are the same in performance is doubtful here because there is a difference in training and validation accuracies which show that one is better than the other. Hence, from validation accuracy, we can say that the LSTM model with validation accuracy equal to 77.37% is slightly better than one with validation accuracy of 76.85% even though the percentage of vocabulary matched in the case of 300 dimensions is greater than that of 200. The plots in Figure 3.2 and 3.3 provide the in-depth performance evolution of models in table 3.4 by drawings the trend lines for corresponding accuracies and losses throughout the 10 epochs. Since the AUC score is the same for the two experiments we need to further look at models' performance through these plots.

#	**Train/ Val. Size**	**Voc. Size**	**Emb. Dim**	**Model**	**Train Acc.**	**Val Acc.**	**AUC Score**
1	360K/90K	97196	200	CNN	0.8008	0.7489	0.843
2				LSTM	0.8458	0.7737	**0.867**
3			300	CNN	0.8219	0.7374	0.835
4				LSTM	0.8623	0.7685	**0.867**

Table 3.4: Experiments and their training, validation and accumulative accuracies after 10 epochs.

In Figure 3.2, the accuracy plot for CNN shows the validation accuracy line slightly stable around ∼0.75 but validation loss jumps upwards after 3rd epoch from ∼0.65 and keeps the same uptrend to the 8th epoch with the value of ∼0.80 showing that model is not best at prediction on new data. However, the LSTM is better than CNN because its validation loss line is slightly upwards and reaches ∼0.60 at the 10th epoch which is very less than CNN loss after 10 epochs. Additionally, the validation accuracy of LSTM seems stable around ∼0.77 which is also greater than CNN. Similarly, in

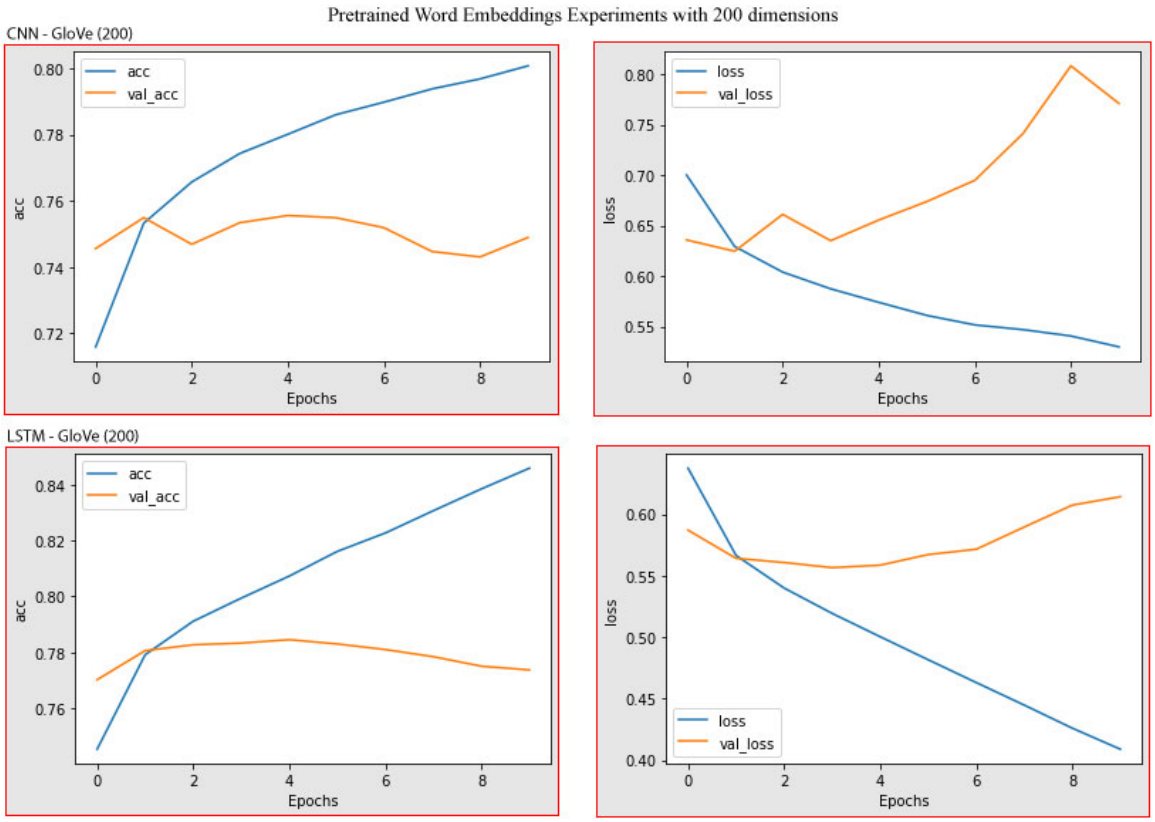

Figure 3.2: Accuracy and loss trends for CNN and LSTM for 200-dimensional GloVe pretrained model.

Figure 3.3, the CNN plot shows the trend of validation accuracy line is slightly going downwards during the period of 10 epochs along with the validation loss line jumping upwards after the 6th epoch and keeping the same trend throughout the rest of epochs. This behavior of plots shows that the model is going towards overfitting. The LSTM also shows almost similar behavior in plots for accuracy and loss lines as CNN. Therefore, both of the models with 300 dimensions are not a good choice based on their results showing an overfitting pattern.

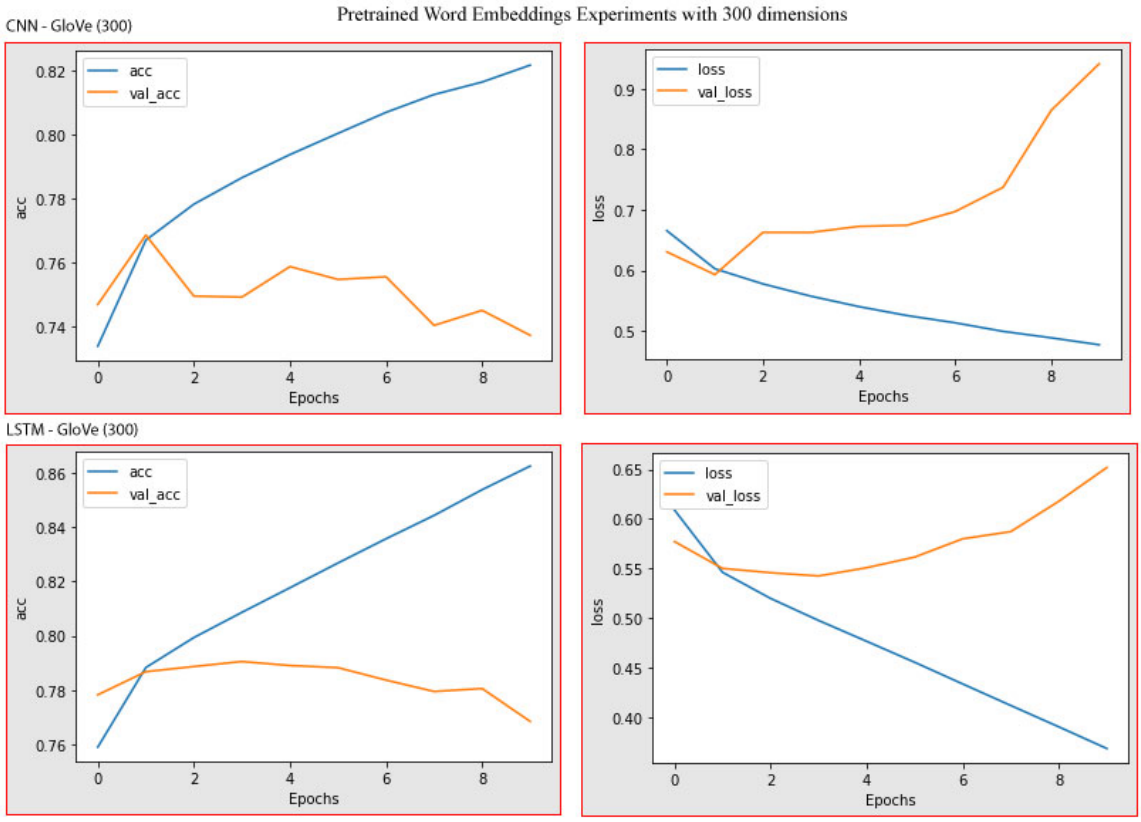

Figure 3.3: Accuracy and loss trends for CNN and LSTM for 300-dimensional GloVe pretrained model.

From all four pairs of plots, the model with GloVe-200 dimensions is best due to its validation accuracy and loss lines tend to remain consistent within the small range of values. All experiments in every technique gave pretty good results but the technique of using GloVe embeddings with LSTM gave the best results because it's a proven model for unstructured sequential data like natural language and the pre-trained GloVe embeddings added in the model to learn the relation amongst similar words. The results of the integer encoding technique are not good as compared to the TF-IDF technique mainly because the issues contain various words from programming languages that are not commonly used in discussions and can generate very unusual vectors. Whereas TF-IDF is our second recommended technique due to fact that the experiments were conducted in ordinary default settings for all classifiers.

RQ2: Can we automatically classify software issues in a bug, feature, or question category only using title and description? If yes, which algorithm or families of algorithms perform the best?

Yes, the software issues reports can be automatically classified in the bug, feature, and question category using the text from title and description only. The experiments show that Recurrent Neural Networks (RNNs), specifically LSTM model with pre-trained word embeddings perform best in solving this problem with AUC accuracy ranging from ∼84% to ∼87% in just 10 epochs.

Although there are many types of issues reported daily for a project on GitHub, the most common of all are three i.e., bug, feature, and question [10]. The majority of previous work was done on the binary classification of issue reports i.e., to identify the bug reports from the non-bugs but a few have worked on classifying them in three of the classes above. In addition to this, the results of most of the work previously done are project-specific and the validity of their results is within the scope of the individual project used for conducting the study. However, there are a few studies that tested their techniques on cross-projects or combined data from all projects which would be better in the scope of their validity. The following section briefly discusses and compares previous work with our work.

3.1.4 Comparison with past works

The authors in [18] have achieved the accuracy of 71.4% and 72% on a binary classification problem to detect bug reports in issues from open-source projects HTTPClient and Lucene respectively which is a bit less than our mean accuracy i.e., ∼77.3% for LR in our experiments with the TF-IDF technique. Although compared to our results, the work in [19], achieved a pretty good average accuracy score between 89.6% to 92.9% for issues from three open-source projects AspectJ, Tomcat, Standard Widget Toolkit (SWT) using 15 keywords feature set from previous works. The reason for higher accuracy could be the fact that the samples were manually classified and the size of dataset was very small as compared to our dataset i-e 593 (AspectJ), 1056 (Tomcat), and 4151 (SWT) samples. The work in [25] obtained an F1 score of 0.746 for cross-project experiments for binary classification of bugs vs non-bugs. For classifying issues in bugs, enhancements, and questions, authors in [26] used the word n-gram method with a maximum n =3 for input in FastText model to classify reports from the collection of closed issues extracted from 12,112 heterogeneous GitHub projects and achieved F1 scores of 0.831 (Bug), 0.823 (Enhancement), 0.825 (Question), and the average as 0.826. The reason for high scores is that their train set was balanced and build from large no: of heterogeneous GitHub projects. The

authors in [29] were able to achieve the weighted average F1 scores of 0.63, 0.76, and 0.79 for HTTPClient, Jackrabbit, and Lucene projects respectively to detect bugs from issues. The work [28] on the binary classification of issues achieved an F1 score of 0.809 for bugs vs non-bugs whereas 0.643 specifically for bugs which are less than our mean F1 score for LR in the TF-IDF technique. However, in our deep learning experiments, instead of using F1 measures, we used the AUC score for evaluating LSTM and CNN models ranging between $\sim$0.77 to $\sim$0.87.

Chapter 4

Topic Modeling on GitHub Issues

This chapter provides a brief introduction of topic modeling, the technique we used for extracting topics from GitHub. Additionally, it also discusses the methodology, experiments design, and results obtained for topic modeling on GitHub issues.

4.1 Topic Modeling

Topic modeling is an advanced method to discover hidden patterns in unstructured textual data. It is an unsupervised learning technique that enables models to learn topics (represented as a set of important words) in a large collection of unlabelled documents. It provides the structure to unstructured textual data. Therefore, topic modeling can be defined as; Topic models are a suite of algorithms that uncover the hidden thematic structure in document collections. These algorithms help us develop new ways to search, browse, organize and summarize large archives of texts [33]. There are many well-known techniques to extract topics from documents such as Latent Semantic Analysis (LSA), Probabilistic Latent Semantic Analysis (PLSA), Latent Dirichlet Allocation (LDA), and Correlated Topic Model (CTM) but the most widely used technique is Latent Dirichlet Allocation (LDA).

4.2 Latent Dirichlet Allocation (LDA)

LDA takes documents as discrete distributions over topics, and topics are regarded as discrete distributions over the terms in the documents [34]. In LDA topic modeling the vocabulary is built using the words in a given collection of documents which is

then used to extract hidden topics in the corpus. Figure 4.1 provides the overview

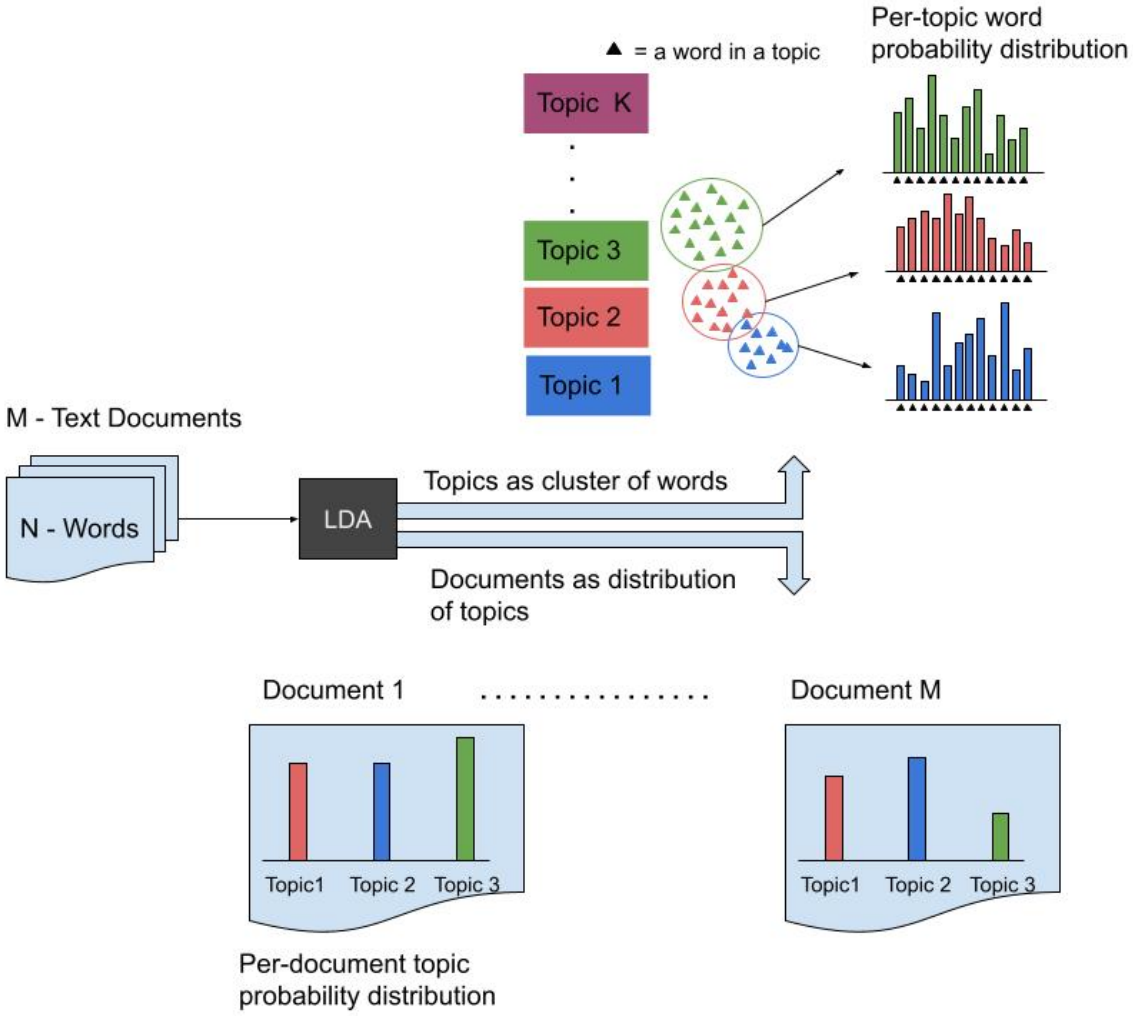

Figure 4.1: Schematic diagram of LDA topic modeling.

of LDA topic modeling process [35]. The input for LDA model is large corpus of M text documents each having N words. Let's say d is a document with $d \in 1, \ldots, M$, Nd represents collection of words in a document d, t is a topic with $t \in 1, \ldots, K$, K is total number of topics and w_n is a word in a document d with $n \in 1, \ldots, N_d$.The general steps of LDA process are as follows

- Start with random K topics
- Randomly assign a topic t to each word in all documents.
- For each document d compute $p(w_j|t_k)$ and $p(t_k|d_i)$ (where $p(w_j|t_k)$ is the pro-

portion of all documents assigned to a topic t_k for a given word w_j and $p(t_k|d_i)$ is the proportion of words in document d_i that are assigned to topic t_k.)

- Update the $p(w_i|t_k, d)$ such that $p(w_j|t_k, d_i) = p(t_k|d_i) * p(w_j|t_k)$
- Loop through each word in each document, reassign the topic for currently selected word based on maximum value for $p(w_j|t_k, d_i)$
- Repeat till end of iterations

Finally, the trained model outputs two different probability distributions θ and ϑ; where θ is the probability distribution of words for each topic t and ϑ is the probability distribution of topics for each document d [35].

Alpha and Beta parameters of LDA

Both alpha α and beta β parameters are Dirichlet prior concentration parameters and their values determine the corresponding densities. The α controls the density of topics in a document, so the higher the value of alpha, the higher the number of topics in a document. Similarly, β determines the density of words in a topic. In other words, a higher value of beta means broader topics, and smaller values result in narrow topics.

4.3 Mallet Package for Topic Modeling

We used the MALLET (A Machine Learning for Language Toolkit) package with python wrappers available in *gensim* to extract the topics from issue reports. Mallet is a java-based package to perform statistical natural language processing tasks including topic modeling. It provides sampling-based efficient implementations of LDA, Pachinko Allocation, and Hierarchical LDA and provides better results than gensim topic modeling libraries [36].

To extract latent topics from our given corpus of GitHub issues, we performed four streams for experiments. First one for issues related to bugs class, second for a feature class, third for question class, and final on combined issues data.

4.4 Methodology

To extract topics from GitHub issues we used the following methodology.

	Bugs	Features	Questions	Combined
Initial Vocabulary	53202	45715	26952	69612
Final Vocabulary	13120	10704	5675	18664

Table 4.1: Initial and final dictionary sizes for bugs, features, questions and all issues reports.

1. Data Preprocessing and Transformation
2. Generating Dictionary and Corpus
3. Building LDAMallet model
4. Analyzing model results

4.4.1 Data Preprocessing and Transformation

In this phase, we used preprocessed data from earlier experiments of machine learning and removed the rest of the columns except id and the text. We performed a few additional steps here to generate bigram and trigrams and also filter out all words except the nouns, adjectives, verbs, and adverbs by using spaCy's default pipeline tagger [37].

4.4.2 Generating Dictionary and Corpus

A dictionary and a corpus are needed to model for extracting topics that are generated from the given dataset. Dictionary is a list of all the unique words from the given dataset along with indices, whereas, a corpus is a list of words and their frequencies. For creating a dictionary and corpus, first, we extracted the initial vocabulary and built a corpus to check the frequency count of words to remove words with high frequencies. For example, the word 'use' with the frequency of 31578, file (20749), error (16076), get (15357), the user (14725), http (13476), version (13221) are a few samples to name just in questions category. Therefore, the words with frequencies greater than 5000 were removed. Similarly, we also removed words that occurred in less than 20 documents and more than 50% of all the documents to obtain the final vocabulary. Table 4.1 shows the vocabulary size of datasets before and after these filtrations.

4.4.3 Building LDA Mallet Model

In this phase, we already have all requirements fulfilled to create and train the model including the ldamallet package. To build the model, the wrapper named LdaMallet in gensim needs a few parameters passed mallet_path (path to ldamallet package), corpus (the corpus built from given dataset), id2word (the dictionary built from given dataset), and num_topics (no: of total topics to be extracted).

4.4.4 LDA Model Evaluation

It is impossible to know before training the LDA model about how many topics exist in our given corpus. Therefore, to estimate the optimal number of topics we used topic coherence measure. We trained LDA on different no: of values for K and compute their coherence values to find out the optimal no: of topics in the corpus. Taking the highest value will mostly give very granular topics with repeated keywords in many topics.

The plot in Figure 4.2 shows coherence scores against the no: of topics. The coherence score continues to increase to the point where no: of topics is 22 with coherence value ∼0.53 and then smoothly decreases with an increase in no: of topics. We know from the plot that our optimal no: of topics is 22 for all combined corpus of issues.

Similarly, we found the optimal no: of topics separately for bugs, features, and questions data and plots in Figure 4.3 show their coherence scores against no: of topics. The plot from left to right is for bugs, features, and questions respectively. The plots for features and questions show the coherence value decreasing after reaching its peak value but for bugs, it jumps up suddenly under 10 topics but remains up in higher values for around 50 topics.

When the coherence score seems to keep increasing in plot with the number of topics, it may make better sense to pick the model that gave the highest CV before flattening out or a major drop [38]. In the case of bugs, we picked k = 17.

- **Bugs-** Num Topics=17 with Coherence Value of 0.5121
- **Features-** Num Topics = 12 with Coherence Value of 0.5247
- **Questions-** Num Topics = 12 with Coherence Value of 0.5907

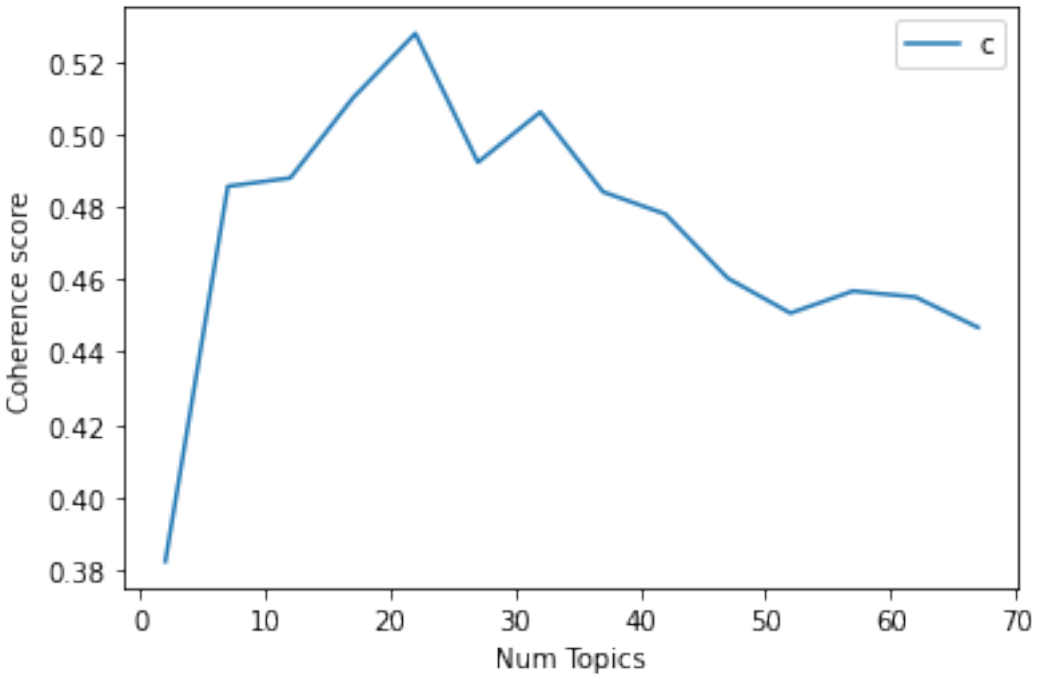

Figure 4.2: Topic coherence w.r.t number of topics for combined data (All).

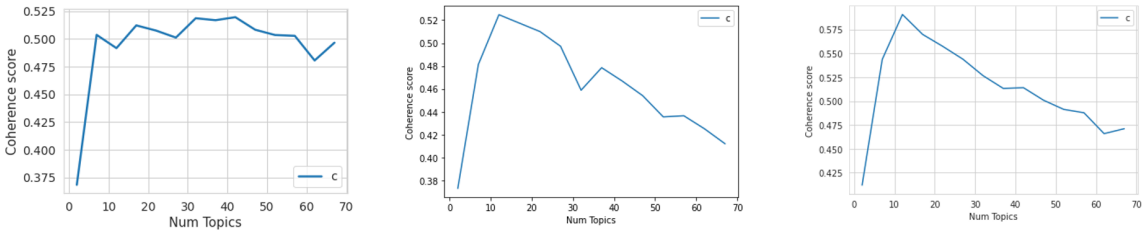

Figure 4.3: Topic coherence w.r.t number of topics for Bugs, Features and Questions.

4.5 Results and Discussion

First, we will discuss the results for extracted topics for combined data in order to get the answer for our ***RQ3***, then for bugs, features, and questions.

4.5.1 Topics Visualizations

There are many other ways to visualize topics and their top terms such as the widely used way is through word clouds but sometimes it is hard to guess the ranking of words based on their weights to their topics.

Figure 4.4 shows the radial dendrogram of extracted topics from combined data

with the top 10 keywords in each topic. This visualization of topic modeling shows the circles with their size defined by the weights of the words contributing to the corresponding topic. The bigger the size of the circle, the higher the weight of the word in the topic. The visualization through this dendrogram keeps the information of weight ranking and also provides a list of some words. There are also some limitations to this dendrogram visualization that it is not dynamic and could not be suitable if no: of extracted topics is larger enough for a corpus.

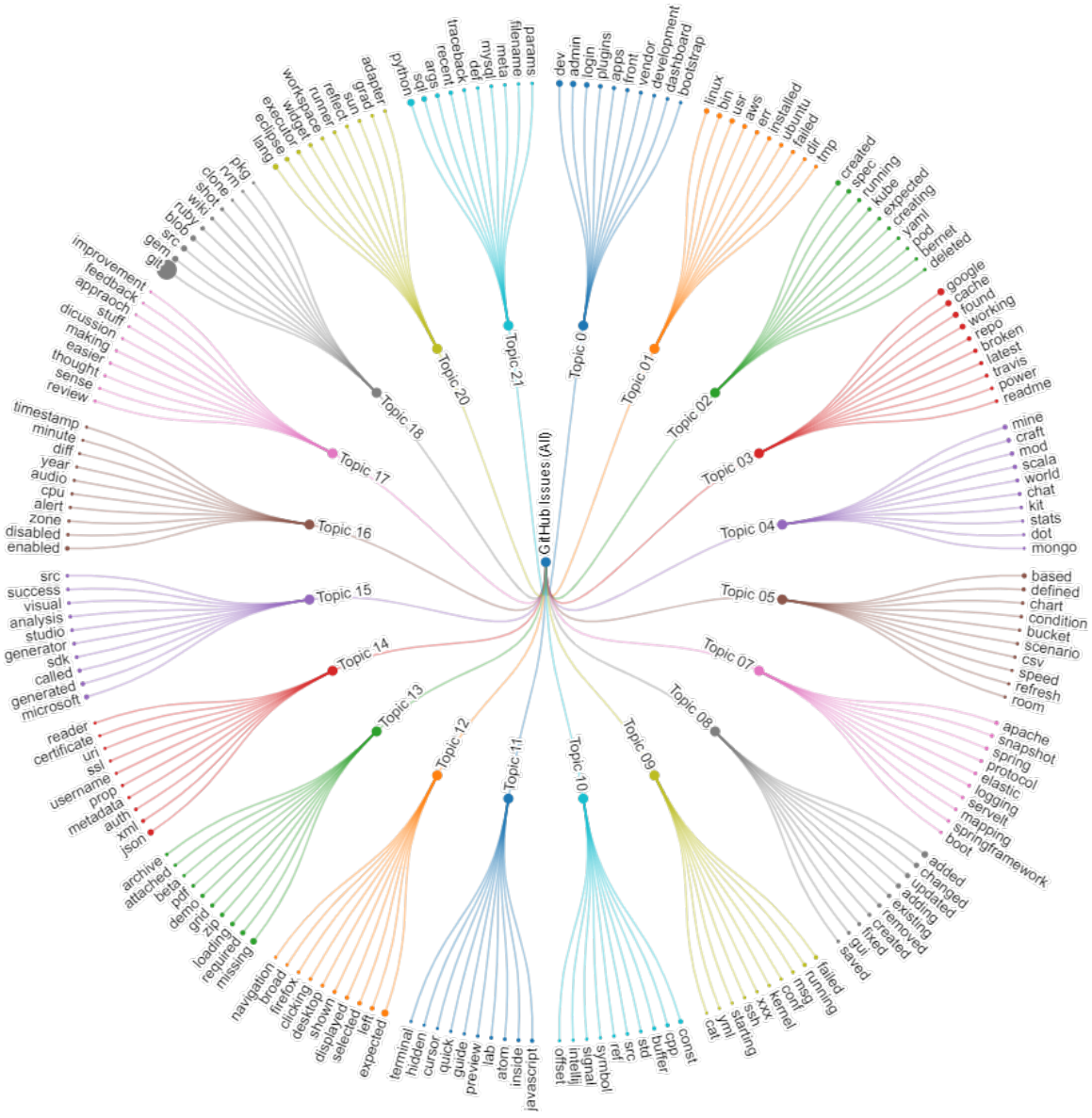

Figure 4.4: Radial Dendrogram - Topic Modeling Visualization.

4.5.2 Topics Visualization using pyLDAvis

The most popular library for visualizing the topics obtained using LDA is pyLDAvis which provides interactivity to better understand the topics, terms, and the relationship amongst them. Selecting a bubble will populate the right side with the top 30 words with their overall and estimated frequency in that selected topic. It also provides the facility to explore relationships amongst topics through exploring the distance between topic bubbles on the left side. If bubbles are fewer, bigger in size, and widespread in all sides of four quadrants instead of congested within one point shows that the model has extracted very general but distinct topics with good validity but the opposite is true for a model with too many topics.

Figure 4.5 shows the pyLDAvis visualization for topics extracted from combined data. The visualization shows that the model was able to extract mostly unique topics but few having repeating keywords as their bubbles are overlapping.

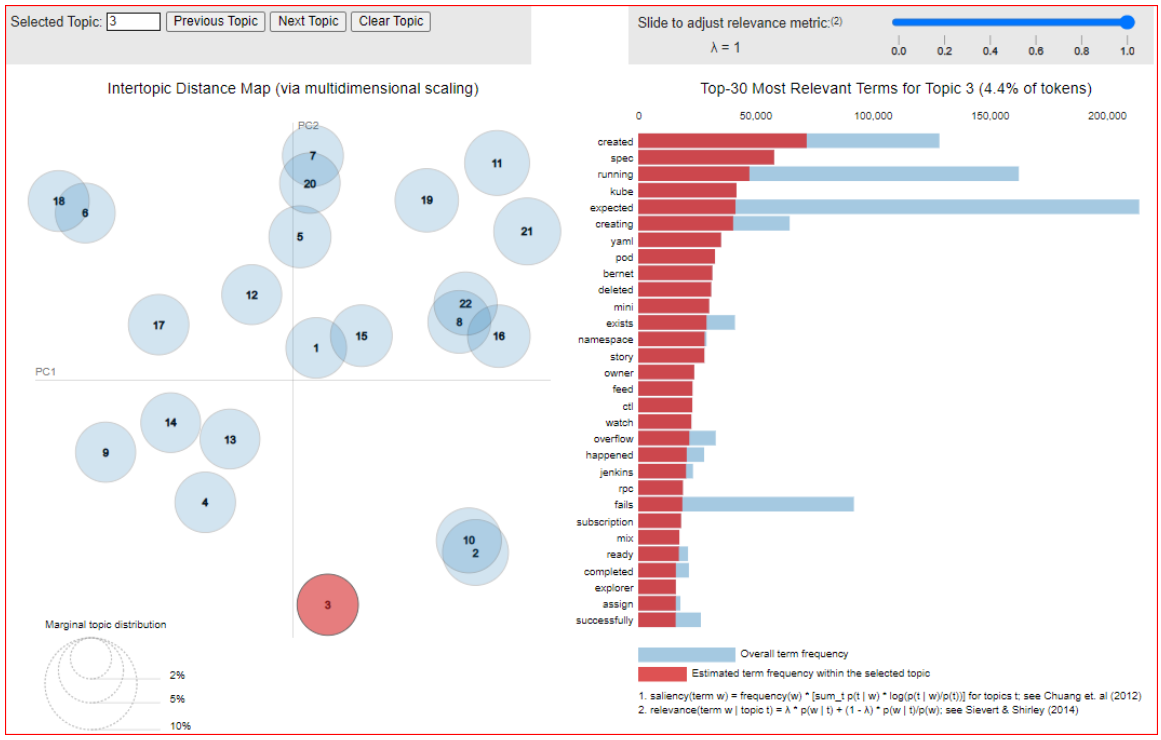

Figure 4.5: pyLDAvis - Topic Modeling Visualization for combined data.

Table 4.2 shows the topics with relevant terms for combined data. The keywords like dev, admin, login, plugins, apps, front, vendor, development, dashboard, bootstrap in **Topic1** most likely represent issues in **plugins or app configuration** for a

blogging platform. Similarly, keywords for **Topic2** most probably represent an issue about **AWS resource access** on Linux os and so on.

Topic #	Top 20 Terms per Topic
Topic1	dev, admin, login, plugins, apps, front, vendor, development, dashboard, bootstrap, www, router, production, article, logged, includes, community, blog, logo, wordpress
Topic2	linux, bin, usr, aws, err, installed, ubuntu, failed, dir, tmp, opt, building, found, cmd, region, dist, running, bash, ode, vue
Topic3	created, spec, running, kube, expected, creating, yaml, pod, bernet, deleted, mini, exists, namespace, story, owner, feed, ctl, watch, overflow, happened
Topic4	google, cache, found, working, repo, broken, latest, travis, power, readme, promise, shell, software, running, forum, instruction, license, win, reading, baz
Topic5	mine, craft, mod, scala, world, chat, kit, stats, dot, mongo, chunk, tick, inventory, fun, bus, forge, age, fire, actor, spawn
Topic6	based, defined, chart, condition, bucket, scenario, csv, speed, refresh, room, vim, center, scan, zoom, fact, linked, implemented, individual, sensor, marker
Topic7	max, min, width, height, random, rate, tile, customer, fit, light, price, magen, payment, white, shape, viewer, blue, relative, catalog, segment
Topic8	apache, snapshot, spring, protocol, elastic, logging, servlet, mapping, springframework, boot, management, processing, transport, monitor, processor, registration, nested, aaaa, consumer, stash
Topic9	added, changed, updated, adding, existing, removed, created, fixed, gui, saved, author, mark, priority, migration, workflow, updating, saving, longer, tracker, made
Topic10	failed, running, msg, conf, kernel, xxx, ssh, starting, yml, cat, peer, rancher, dns, connected, received, tcp, started, pid, timer, mount

Table 4.2: Top 10 topics with top 20 relevant terms for combined dataset.

The visualization in figure 4.5 shows that topic 10 overlaps with topic 2 in a bigger portion of its area. This is because the top two keywords of topic 10 i.e., *failed* and *running* are also found in the top 10 keywords of topic 2. Additionally, the words in both topics are semantically very close and most probably represent **remote resource access**.

4.5.3 Dominant Topics Distribution Across Corpus

Figure 4.6 shows the distribution of dominant topics throughout the whole corpus. The x-axis shows topics and the y-axis shows their frequency in total documents (or issues). The plot shows that a few topics like 1, 7, 13, and 18 are very dominant with a frequency count of more than 25K whereas some of them are less dominant such as topics 5, 6, 12, and 20. Topic 21 is the least dominant among all of the topics.

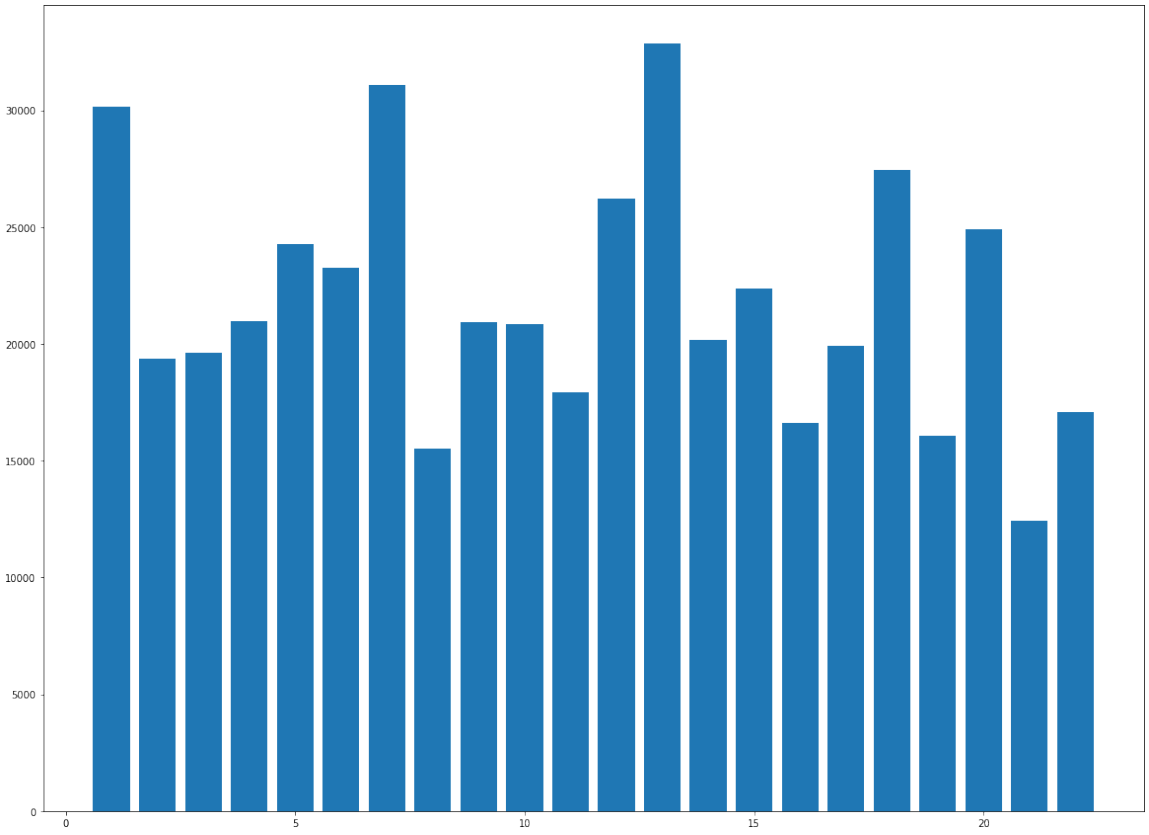

Figure 4.6: Dominant topic distribution across corpus for combined data.

4.5.4 Bugs, Features and Questions

The optimal number of topics extracted in each of these three categories separately are depicted in word clouds in Figure 4.7.

Figure 4.7: (Top to bottom) Topic-terms word clouds for bugs, features and Questions.

The Table 4.3, 4.4, and 4.5 present top 10 topics with top relevant terms related to bugs, features and questions respectively. Each Topic contains top 20 keywords with weights increasing in ascending order.

Topic #	Top 20 Terms per Topic
Topic1	abstract, eclipse, lang, apache, executor, xml, sun, runner, factory, reflect, snapshot, servlet, concurrent, parent, springframework, threadpool, grad, spring, execution, processor
Topic2	auth, kube, timeout, failed, volume, elastic, shift, pod, xxx, bernet, sending, mini, deploy, ssl, policy, loaded, street, kind, endpoint, ctl
Topic3	setting, changed, password, enabled, msg, running, shell, expected, reset, power, ssh, sign, extra, username, disable, disabled, summary, paste, fails, alert
Topic4	png, git, admin, login, graph, shot, displayed, permission, www, broken, shown, showing, pipeline, hash, aaaa, img, website, baz, logged, stash
Topic5	linux, failed, ubuntu, tmp, zip, flow, running, dir, kernel, total, cpu, archive, disk, vim, pid, cat, starting, software, utc, sec
Topic6	tab, drop, selected, expected, theme, atom, clicking, panel, background, selection, visible, press, area, preview, gif, focus, bottom, cursor, blank, mouse
Topic7	side, bot, chat, card, jpg, mobile, left, track, contact, hit, phone, effect, term, large, lot, small, turn, audio, big, amount
Topic8	int, channel, const, byte, cpp, buffer, std, static, void, symbol, signal, pdf, char, free, rust, mongo, proto, gcc, netty, struct
Topic9	cache, sql, scala, parser, statement, builder, mysql, pool, queue, join, lock, insert, uri, scope, dot, batch, topic, syntax, operator, prefix
Topic10	microsoft, aws, trace, terraform, framework, async, asset, studio, success, azure, boolean, previous, visual, region, sdk, thrown, exe, debugger, dll, book

Table 4.3: Top 10 topics with top 20 relevant terms for Bugs.

Topic #	Terms per Topic
Topic1	expected, browser, question, language, actual, reproduce, related, video, describe, additional, behaviour, angular, chrome, demo, summary, live, platform, latest, fill, visual
Topic2	cache, token, login, day, auth, transaction, register, sign, topic, meta, scope, authentication, total, stats, year, customer, join, refresh, credential, active
Topic3	doc, html, google, entry, developer, storage, team, global, readme, external, font, website, sync, guide, backend, usage, dashboard, hard, bundle, javascript
Topic4	python, directory, home, running, root, debug, linux, program, env, machine, installed, bin, shell, binary, installation, txt, apps, lib, found, zip
Topic5	warning, net, src, notification, cli, exception, static, stack, pack, plugins, loading, framework, lib, upgrade, logging, asset, common, helper, dynamic, generator
Topic6	setting, php, auto, admin, export, controller, provider, manager, cluster, disable, role, aws, enabled, permission, security, management, csv, yaml, policy, bounty
Topic7	structure, pattern, hash, lot, break, lab, avoid, extra, clean, ignore, reason, rename, checking, backup, exists, existing, batch, detect, removed, alias
Topic8	array, rule, int, count, memory, flow, range, operation, word, frame, chart, metric, byte, convert, buffer, operator, raw, matrix, dataset, cpu
Topic9	json, attribute, var, flag, argument, false, entity, schema, null, empty, foo, optional, print, character, max, console, const, callback, override, prefix
Topic10	label, limit, background, performance, large, small, speed, effect, random, high, rate, fast, scale, algorithm, low, score, turn, hit, increase, light

Table 4.4: Top 10 topics with top 20 relevant terms for Features.

Topic #	Terms per Topic
Topic1	content, cloud, png, link, tag, stream, folder, video, load, download, player, upload, source, demo, medium, show, zip, track, play, jpg
Topic2	case, multiple, feature, based, solution, part, interface, specific, single, implementation, implement, unit, testing, separate, current, side, structure, existing, approach, functionality
Topic3	app, android, library, plugin, google, device, push, platform, application, sdk, sample, notification, framework, working, activity, plugins, apps, phone, swift, manager
Topic4	window, order, variable, bit, extension, language, product, entry, foo, bar, transaction, working, limit, missing, match, supported, print, visual, studio, sharp
Topic5	docker, container, instance, resource, host, info, aws, running, root, driver, storage, machine, environment, cluster, agent, network, configuration, che, volume, var
Topic6	input, date, size, number, read, output, format, write, frame, byte, memory, buffer, max, flow, range, length, int, camera, real, rate
Topic7	point, element, box, label, level, tool, chart, editor, space, top, display, height, width, drop, position, layer, left, figure, word, original
Topic8	install, python, lib, local, directory, home, linux, bin, usr, command, installed, warning, found, program, failed, dir, target, include, src, cpp
Topic9	api, url, post, response, html, header, template, send, site, custom, email, route, body, rest, grid, put, mail, working, website, chat
Topic10	module, node, component, import, react, script, const, err, export, native, angular, require, index, prop, lib, app, load, child, vue, javascript

Table 4.5: Top 10 topics with top relevant terms for Questions.

Figure 4.8 provides the comparison of documents percentage for top 20 dominant topics for each category of issues. Topic 6 and Topic 11 seem to have a high percentage of documents in all three categories as their percentage is close to 50%. Additionally, Topic 4, 6, and 7 have the highest no: of documents in the bug category. The majority of topics have more than 30% of documents in all categories except Topic 1, 12, 16, and 18. Topic 11 and 18 are the most influential topics for questions and features respectively as compared to the whole corpus.

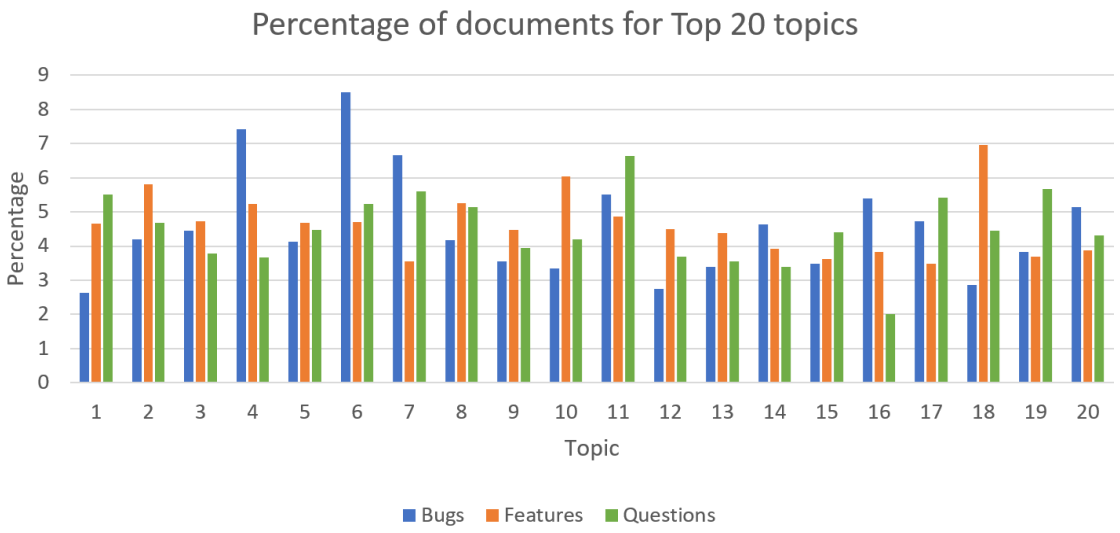

Figure 4.8: Percentage of documents for top 20 dominant topics in bugs, features, and questions.

To get the answer for ***RQ3***, we will analyze the keywords of highly influential topics for combined data. The keywords for Topic 1, 5, 6, 12, and 18 would help us in getting information about the most frequent type of issues the developers have to face during software development and maintenance. We already analyzed the keywords for Topic 1 and it turned to be for **plugins or app configuration for a blogging platform**. The keywords for topic 5 *(mine, craft, mod, scala, world, chat, kit, stats, dot, mongo, chunk, tick, inventory, fun, bus, forge, age, fire, actor, spawn)* most probably represent topic about **mine craft game global chat**. For Topic 7 the keywords *(max, min, width, height, random, rate, tile, customer, fit, light, price, magen, payment, white, shape, viewer, blue, relative, catalog, segment)* most probably show shipping and payment modules of e-commerce platform. Similarly, keywords *(javascript, inside, atom, lab, preview, guide, quick, cursor, hidden, terminal, lint, blank, electron, calendar, ctrl, markdown, esl, quote, proto, typescript)* represent is-

sues regarding opensource **text editor 'Atom'**. Finally, Topic 18 with keywords *(review, sense, thought, easier, making, discussion, stuff, approach, feedback, improvement, svg, counter, general, feel, answer, proposal, alternative, addition, easily, made)* represents **in-depth analysis and improvement** for anything.

RQ3: What type of development issues are faced by software developers?

From the analysis of topics and their dominancy we come to know that the developer mostly faces problems regarding remote resource access, integration and configurations issues, issues regarding game design and development, payment transactions, front-end design issues, etc.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Software issues classification has been a hot topic for many years. There is a lot of work done by previous researchers to find out the best methods and techniques to automate the task of labeling software issues. Many empirical studies have achieved good results for the solution to this problem using various natural language processing techniques. The previous work was mostly done on detecting issue reports for bugs because bugs are the main hindrances in software project development and maintenance. In the open-source community, issues regarding feature requests, enhancements, improvements, and general questions by contributors play a vital role in the overall progress of the project. Hence, we have tried to find a general technique to classify issues in the three most common classes Bugs, Features, and Questions as a multiclass classification problem using state-of-the-art natural language processing techniques using machine learning and achieved good results as compared to many of previous works. Our results may not beat the accuracy results because we only used unstructured natural language data and the problem was multiclass. Our research has opened paths for further empirical experiments which could help achieve the future goal of automatic software issue classification in general.

We also have extracted the latent topics in GitHub issues using the LDA-based topic modeling technique. We tried to assess the big picture of the type of issues faced by the software development team during the maintenance.

5.2 Application

A tool such as the GitHub application can be developed using the proposed machine learning technique which can also be integrated with any GitHub project.

There can be two ways to use this application for a GitHub project. First, it can be used as a recommendation tool to suggest the labels based on their likelihood values. Second, it can assign a label based on the model's final detection result. The app could have an option to switch between modes of usage to let developers choose one based on their circumstances of workload and time.

The model proposed using the deep learning model with GloVe word embeddings outputs three probabilities in order of bugs, features, questions. For example, a text is given as input to the model and it provides this output string ([[0.0622307 0.9040217 0.03374764]] feature). The one with the highest probability is the final result as *feature* in the above example.

Finally, it's on the maintenance team how they want to use this model to assist in issue labeling.

The proposed technique for extracting latent topics in issue report can help us in figuring out the key areas of problems faced by developers. The maintenance team can use this information to improve the documentation.

5.3 Threats to Validity

Construct

The dataset was obtained from Kaggle and there was no information about it except its meta-data and it was also highly unbalanced. The lack of getting enough information about the dataset used has raised issues of construct validity.

Internal

We didn't used any well defined criteria for choosing hyper parameters but instead used the default ones and the experiment and verification method which leads to internal validity threat. The conclusion of claims for best combination of technique could possibly change in future considering this fact.

5.4 Future Work

For future work, here are a few possible suggestions.

- The work could be further improved by using combined features from content and meta-morphological information of issue reports. One technique could be using features from content since we discovered the difference among bugs, features, and questions (for example features like 'Has Code', 'Has Username', 'Has HTML', 'Code Snip Length' etc) and also meta-morphological features as we found some in our study are 'Word Count', 'Word Length', 'Unique Word Count', 'Stop Word Count' etc.
- We did not test other pre-trained language models like BERT, XLNet, OpenAI's GPT-2 etc.
- For better classification accuracy, advanced techniques of hyperparameter tuning for deep learning methods can be implemented.
- To ensure the scope and external validity of techniques this work can be applied on some well-known datasets used in previous studies.

Bibliography

[1] M. D'Ambros and R. Robbes, "Effective mining of software repositories.," in *ICSM*, p. 598, 2011.

[2] A. E. Hassan, "The road ahead for mining software repositories," pp. 48–57, 11 2008.

[3] K. K. Chaturvedi, V. Sing, and P. Singh, "Tools in mining software repositories," in *2013 13th International Conference on Computational Science and Its Applications*, pp. 89–98, IEEE, 2013.

[4] A. Di Sorbo, G. Grano, C. Aaron Visaggio, and S. Panichella, "Investigating the criticality of user-reported issues through their relations with app rating," *Journal of Software: Evolution and Process*, vol. 33, no. 3, p. e2316, 2021.

[5] P. Floris and H. Vogt Harald, "How to save on software maintenance costs," *Omnext white paper*, vol. 2, 2010.

[6] 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, 2005.

[7] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*, pp. 188–197, IEEE, 2013.

[8] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes," in *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 251–260, IEEE, 2014.

[9] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24003–24015, 2018.

[10] J. Wang, X. Zhang, and L. Chen, "How well do pre-trained contextual language representations recommend labels for github issues?," *Knowledge-Based Systems*, p. 107476, 2021.

[11] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *2013 35th international conference on software engineering (ICSE)*, pp. 392–401, IEEE, 2013.

[12] S. Herbold, A. Trautsch, F. Trautsch, and B. Ledel, "Issues with szz: An empirical assessment of the state of practice of defect prediction data collection," *arXiv preprint arXiv:1911.08938*, 2019.

[13] C. Mills, J. Pantiuchina, E. Parra, G. Bavota, and S. Haiduc, "Are bug reports enough for text retrieval-based bug localization?," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 381–392, IEEE, 2018.

[14] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *11th working conference on reverse engineering*, pp. 214–223, IEEE, 2004.

[15] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source code retrieval for bug localization using latent dirichlet allocation," in *2008 15th Working Conference on Reverse Engineering*, pp. 155–164, IEEE, 2008.

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

[17] I. Chawla and S. K. Singh, "An automated approach for bug categorization using fuzzy logic," in *Proceedings of the 8th India Software Engineering Conference*, pp. 90–99, 2015.

[18] N. Pandey, A. Hudait, D. K. Sanyal, and A. Sen, "Automated classification of issue reports from a software issue tracker," in *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, pp. 423–430, Springer, 2018.

[19] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in *Proceedings of the 9th International Conference on Information Communication and Management*, pp. 17–21, 2019.

[20] M. S. Zolkeply and J. Shao, "Classifying software issue reports through association mining," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 1860–1863, 2019.

[21] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? bug report classification using n-gram idf," in *2017 IEEE international conference on software maintenance and evolution (ICSME)*, pp. 534–538, IEEE, 2017.

[22] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016.

[23] N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *2013 20Th asia-pacific software engineering conference (APSEC)*, vol. 2, pp. 13–18, IEEE, 2013.

[24] N. Limsettho, H. Hata, and K.-i. Matsumoto, "Comparing hierarchical dirichlet process with latent dirichlet allocation in bug report multiclass classification," in *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1–6, IEEE, 2014.

[25] H. Qin and X. Sun, "Classifying bug reports into bugs and non-bugs using LSTM," in *Proceedings of the tenth Asia-Pacific symposium on internetware*, pp. 1–4, 2018.

[26] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 406–409, IEEE, 2019.

[27] F. A. Research, "FastText: A library for efficient learning of word representations and sentence classification."

[28] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5333–5369, 2020.

[29] N. Pandey, D. K. Sanyal, A. Hudait, and A. Sen, "Automated classification of software issue reports using machine learning techniques: an empirical study," *Innovations in Systems and Software Engineering*, vol. 13, no. 4, pp. 279–297, 2017.

[30] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall series in artificial intelligence, Pearson Prentice Hall, 2009.

[31] J. Zhou, A. H. Gandomi, F. Chen, and A. Holzinger, "Evaluating the quality of machine learning explanations: A survey on methods and metrics," *Electronics*, vol. 10, no. 5, p. 593, 2021.

[32] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

[33] D. M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[34] S. Lazarina, "Topic modelling: A deep dive into lda, hybrid-lda, and non-lda approaches." Lazarina Stoy, July 2021, Accessed on: Sept. 5 2021 [Online]. Available: https://lazarinastoy.com/topic-modelling-lda/.

[35] Ipshita, "Topic modelling using lda." Medium, July 15, 2021, Accessed on: Sept. 5 2021 [Online]. Available: https://medium.com/analytics-vidhya/topic-modelling-using-lda-aa11ec9bec13.

[36] A. K. McCallum, "Mallet: A machine learning for language toolkit." http://mallet.cs.umass.edu, 2002.

[37] M. Honnibal and I. Montani, "Spacy trained models & pipelines." spaCy, 2016 - 2019, Accessed on: Sept. 9, 2021 [Online]. Available: https://spacy.io/models.

[38] S. K., "Evaluate topic models: Latent dirichlet allocation (lda), a step-by-step guide to building interpretable topic models." Medium, Aug. 19, 2019, Accessed

on: Sept. 9, 2021 [Online]. Available: https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0.