Summarizing Developer Chat Conversations

by

Lakshmi Prasanna Kumar

A Thesis submitted to
the Faculty of Graduate and Postdoctoral
Affairs in partial fulfilment of
the requirements for the degree of
Master of Computer Science
with
Specialization in Data Science

Carleton University
Ottawa, Ontario

©2022
Lakshmi Prasanna Kumar
The undersigned recommend to the Faculty of Graduate and Postdoctoral Affairs acceptance of the Thesis

**Summarizing Developer Chat Conversations**

Submitted by **Lakshmi Prasanna Kumar**

in partial fulfilment of the requirements for the degree of

**Master of Computer Science**

---

Dr. Olga Baysal, Thesis Supervisor

Dr. Diana Inkpen, Internal Examiner

Dr. Majid Komeili, External Examiner

Dr. Olga Baysal, Chair of Defence

Dr. Michel Barbeau, Department Chair

Carleton University

2022
Abstract

In recent years there is an unprecedented growth in online communication and collaborative platforms like Slack, Discord, Microsoft Teams, Gitter, etc. These platforms facilitate communication among developers all over the world and allow distributed software development. Software developers rely on these platforms to discuss their projects and to seek technical help. These discussions are vital source of information that would assist researchers and tool makers to develop tools and services like chatbots, automated virtual assistants, chat summarization techniques, Q&A thesaurus, etc. Summarizing developer discussions allows users to quickly grasp the highlights of a conversation without going through the entire content. It is challenging to summarize these chat messages due to their short size, unstructured, colloquial format with abbreviations and emojis.

This thesis is an attempt to tackle the problem of summarizing chat conversations by applying topic modeling techniques to generate discussion summaries. Topic modeling approaches are proven to be successful in identifying the high level functionality of source code in source code summarization [1]. We use a similar approach in this thesis to extract short summary based on topics to have a high level understanding of the conversations. In this work, we use a dataset extracted from the Discord chat conversations and evaluate four topic modeling techniques to identify the primary topics from the discussions. Additionally, we evaluate different embedding models and study their impact on the performance of the topic modeling technique. We
perform an extensive analysis of the topics per month to have a better understanding of discussions. We also study evolution of the topics over a period of one year to understand the common, emerging, and disappearing topics.
I dedicate this thesis to my beautiful daughters, Jiya & Tanvi, and my husband Jitu, who supported me throughout this journey.
Acknowledgments

I, Lakshmi Prasanna Kumar, would like to express my sincere gratitude to my esteemed supervisor, Professor Olga Baysal, for her continuous guidance, advice, and support. I am sincerely thankful for her attention, immense knowledge and valuable feedback for the successful completion of this research and without her, I would not have made it through my Master's degree.

I am thankful for the valuable advice and suggestions from Prof. Preetha Chatterjee (Drexel University) in proposing the dataset DISCO: A Dataset of Discord Chat Conversations for Software Engineering Research which is an integral part of this research.

I am very grateful to my husband and my beautiful daughters Jiya and Tanvi who were constantly encouraging, motivating, and supporting me throughout this journey. I am also thankful to my parents, in-laws and siblings for their endless support and keeping me optimistic at all times.

I would like to acknowledge the help of my friend and my peer Keerthana Muthu Subhash, for participating in the manual labeling process of this work. Without her help and time the evaluation of topic models would not have been possible.

Last, but not least, I would like to acknowledge the advice and encouragement of several professors: Prof. Diana Inkpen and Prof. Herna Victor from the University of Ottawa, and Prof. Majid Komeili from Carleton University.
# Table of Contents

Abstract ........................................ iii

Acknowledgments ................................ vi

Table of Contents ................................ vii

List of Tables ..................................... x

List of Figures .................................... xi

Nomenclature ..................................... xiii

## 1 Introduction ................................. 1

1.1 Summarizing Developer Chat Conversations .......... 1

1.2 Motivation ..................................... 3

1.3 Research Questions .............................. 5

1.4 Contributions .................................. 6

1.5 Thesis Structure ............................... 7

## 2 Background and Related Work ................. 8

2.1 Background .................................... 8

2.1.1 Text Summarization ......................... 9

2.1.2 Topic Modeling ............................. 10
4 Results

4.1 Performance of Topic Modeling Techniques ........................................... 57
4.2 Impact of Embedding Models on the Performance of BERTopic ............. 61
4.3 Extracted Topics for python#python-general ......................................... 66
4.4 Evolution of Topics ............................................................................. 67

5 Discussion ............................................................................................. 72

5.1 Findings ............................................................................................... 72
5.2 Implications .......................................................................................... 74
   5.2.1 Implications for Software Developers .............................................. 74
   5.2.2 Implications for Researchers ............................................................ 75
   5.2.3 Implications for Python Community Moderators ............................ 76
5.3 Threats to Validity ............................................................................... 76

6 Conclusion .............................................................................................. 79

6.1 Summary of Contributions ................................................................... 79
6.2 Future Work ......................................................................................... 81

List of References ..................................................................................... 82

Appendix A Appendix ................................................................................ 97

A.1 Representative Documents for Topics ................................................... 97
A.2 Evaluation of Topic Models .................................................................. 99
# List of Tables

1. Dataset of disentangled Discord conversations.................. 33
2. Dataset statistics................................................. 38
3. Default parameters for LDA.................................... 40
4. Default parameters for NMF.................................... 42
5. Default parameters for CTM.................................... 46
6. Pre-trained models for sentence transformer..................... 52
7. Examples of the topic misc..................................... 55
8. Dataset size....................................................... 58
9. Performance evaluation of topic models.......................... 59
10. Performance evaluation of BERTopic with embedding models... 62
11. Manual rating for three embedding models for each month...... 63
12. Number of meaningful topics per month.......................... 66
13. Topics extracted for python#python-general channel............ 68
14. General topics.................................................... 69
List of Figures

1. Graphical model representation of LDA ....................... 12
2. Non-negative matrix factorization ............................ 13
3. High-level architecture of Combined Topic Model ........... 15
4. Main components of BERTopic ................................ 17
5. Workflow diagram ............................................. 29
6. Overview of the data collection and disentanglement process .. 31
7. Format of the chat conversation data .......................... 35
8. Workflow of the OCTIS framework ............................. 39
9. The overall workflow of BERTopic ............................. 47
10. SBERT architecture with classification objective function .... 51
11. NPMI score for topic models for June 2020 .................. 60
12. C\textsubscript{v} score for topic models for June 2020 ....... 60
13. Diversity score for topic models for June 2020 ............... 61
14. NPMI score for BERTopic with different embeddings-December 2019 ... 64
15. C\textsubscript{v} score for BERTopic with different embeddings-December 2019 ... 64
16. Diversity score for BERTopic with different embeddings-December 2019 ... 65
17. Total number of conversations per month ....................... 70
18. Total number of utterances per month ........................ 70
19. Heatmap of topics between November 2019 and October 2020 ... 71
20. NPMI score for topic models for December 2019 ............. 99
21 C_v score for topic models for December 2019 . . . . . . . . . . . . . . . . . . . . . 100
22 Diversity score for topic models for December 2019 . . . . . . . . . . . . . . . . . . . . 100
23 NPMI score for topic models for March 2020 . . . . . . . . . . . . . . . . . . . . . . . . . . 101
24 Diversity score for topic models for March 2020 . . . . . . . . . . . . . . . . . . . . . . . . . . 101
25 C_v score for topic models for March 2020 . . . . . . . . . . . . . . . . . . . . . . . . . . 102
26 NPMI score for BERTopic with different embeddings - March 2020 . . . . . . . . . . . . . . . . . . . . 102
27 Diversity score for BERTopic with different embeddings - March 2020 . . . . . . . . . . . . . . . . . . . 103
28 C_v score for BERTopic with different embeddings - March 2020 . . . . . . . . . . . . . . . . . . . 103
29 Sample topic labeling for GloVe-SO and GloVe . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104
Nomenclature

Abbreviations

This thesis uses some common abbreviations existing in the domain of computer science. The following table lists the abbreviations and their meaning:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO</td>
<td>Stack Overflow</td>
</tr>
<tr>
<td>LDA</td>
<td>Latent Dirichlet Allocation</td>
</tr>
<tr>
<td>NMF</td>
<td>Non-Negative Matrix Factorization</td>
</tr>
<tr>
<td>CTM</td>
<td>Combined Topic Model</td>
</tr>
<tr>
<td>MMR</td>
<td>Maximal Marginal Relevance</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The first chapter introduces our research vision of summarizing developer chat conversations (Section 1.1) which is the primary goal of this thesis by highlighting the importance of mining data from online collaboration platforms like Discord and applying topic modeling techniques to extract summaries of the data. Section 1.2 provides a brief motivation behind this work, followed by the research questions (Section 1.3). This is followed by the contributions of this thesis (Section 1.4), and lastly, a thesis structure presented in Section 1.5 concludes this chapter.

1.1 Summarizing Developer Chat Conversations

Recent years have seen the growth of online collaboration platforms such as Discord, Slack, IRC, Gitter, and Zoom in the software development industry. Developers rely on these platforms for collaboration and communication on their projects. These communication channels can be used to share knowledge, receive technical help, and also for real-time conversations between users. These chats are rich in valuable data which can help software development communities.
Chatterjee et al. [2] [3] have mined Slack chat conversations and their studies highlight the importance of valuable information in these chats. Code snippets’ description and APIs, bug debugging techniques, best programming practices, and causes of common errors/exceptions are a few examples. Similarly, Gitter data is studied by Esteban et al. [4] to help new developers get familiar with software products. Another communication platform gaining popularity is Discord. Originally developed for the online gaming community, this platform appeals to over 150 million active monthly users, as of 2021 [5]. Mining Discord data with millions of messages is an indispensable activity that provides numerous research opportunities thereby helping software development community. Summarizing these conversations is a potential research area which has not been explored much in the software engineering research.

Automatic software summarization is a growing field within the software engineering research that generates a succinct representation of software artifacts. This process provides the required information needed by a software stakeholder in a concise form that helps to perform a particular software engineering task [6]. The presence of source code makes software summarization different from text summarization. Moreno et al. [6] categorizes the summarization techniques into (i) text-to-text summarization: the textual artifacts such as bug reports or user reviews are summarized to a textual data. (ii) Code-to-text summarization: textual summaries are generated from source code such as classes, methods, test cases, code changes, etc. (iii) Code-to-code summarization: source code based summaries are generated from code fragments or code usage examples, and (iv) mixed artifact summarization: summaries are generated from heterogeneous software artifacts that contain text data and source code such as programming forum posts. This thesis focuses on the fourth category that is a mix of both text and source code that has not been explored much by the software engineering research community. In this work, we consider the source code that is part of the general discussions. We do not consider any source code that is added as
an attachment to the conversations.

In this thesis, we study the data from a Discord public channel, python#python-general, for a period of one year from November 2019 to October 2020. The aim is to summarize the developer discussions during this time period to gain a comprehensive idea of the discussions. Application of traditional summarizing techniques such as extractive and abstractive methods is highly challenging to generate meaningful summary from the chat data. The highly unstructured, informal, colloquial structure of chat data poses challenge in the summarization task. We try to overcome these challenges by applying topic modeling techniques to extract the topics discussed. Finally we also explore the evolution of topics to understand the emerging topics and the disappearing topics.

1.2 Motivation

Online chat conversations are a rich source of knowledge since a substantial amount of information is exchanged between the participants. Summarizing the chat conversations helps to distill the vital information into a comprehensive form. One of the advantages is that it helps the users to capture the highlights of the conversation without reading the entire text. The usage of online communication platforms has been on the rise since the pandemic. Due to this reason there is an information overload that results in a need for a summarizer.

Software development communities also rely on online communication and collaborative platforms such as Slack, Gitter, Discord, IRC, etc. for their communication. The information exchanged between user is extremely resourceful including code snippets’ description and APIs, bug debugging techniques, best programming practices, and causes of common errors/exceptions [7]. Mining such data and summarizing it
can be very useful for software developers, researchers, and moderators of the community and ecosystem. Though there exists a considerable amount of research in summarizing software artifacts such as bug reports, source code, mailing lists, API related documents, etc., summarizing developer discussions have not been explored much. Developer discussions are heterogeneous in nature because of the presence of text and source code data. It is challenging to summarize the conversations because they are shorter, unstructured, and they contain spelling mistakes, hyperlinks, acronyms, and emojis.

There are two approaches in text summarization namely extractive and abstractive summarization. This thesis focuses on summarizing Discord data by extracting the topics discussed. Due to the challenges in summarizing unstructured conversations using traditional text summarization techniques, we use topic modeling based summarization that generates short summaries to get a high level understanding of the conversations. Discord chat conversations also follow informal, unstructured, and asynchronous format. It has a number of participating users, and the conversation length can range from two messages to hundreds. These conversations are entwined with each other. Previous research have used topic modeling techniques for getting a sense of unstructured data mined from software repositories [8] [9] [10]. In this work, we focus initially on disentangling the conversations and then grouping them together based on the conversation ID so that all the messages in a thread are clubbed together. This helps in overcoming one of the challenge that is the data sparsity problem in short text and also helps in identifying meaningful topics.

Following this, we perform a comparative study of state-of-the-art topic models like LDA [11], NMF [12], CTM [13], and BERTopic [14] and identify the best performing topic model along with the best embedding model in extracting meaningful topics. We also study and analyze the topics discussed in each month and see how
these topics evolve over time. This helps us to understand the changes in the discussions that are happening over a one year period. Identifying the emerging and diminishing topics help stakeholders in many ways. Support documents for Python language can be prepared based on these findings. Also developers can search for a particular topic and the corresponding conversations effortlessly using this technique.

1.3 Research Questions

The research questions addressed in this work are listed below. Figure 5 illustrates the overall workflow that helps to answer the research questions.

- **RQ1: How effective are topic modeling techniques in extracting summaries from developer conversations?**

  In this research question, we are trying to identify which topic modeling technique is most efficient in summarizing developer conversations. We leverage four topic models such as LDA, NMF, CTM, and BERTopic. The models are evaluated based on topic coherence metrics such as $C_w$ & NPMI and Topic Diversity metrics along with manual rating of the quality of the topic models. This is an important question as it is the primary step in investigating how well the topic models are able to summarize the conversations.

- **RQ2: What is the impact of different embedding models on the performance of BERTopic?**

  This research question investigates the impact of various embedding models on the topics extracted by the BERTopic, the topic model identified from RQ1. We conduct a comparative analysis of word + document embeddings provided by BERTopic. We choose three pretrained embeddings from sentence transformers.
along with a custom word embedding (GloVe-SO) trained on Stack Overflow data dump for the experiments.

- **RQ3**: What are the topics discussed in the python#python-general Discord channel?

Here, we try to extract different topics discussed on the Discord channel python#python-general. These topics reflect the summary of the discussions that happen among the Python development community. Two annotators perform a manual labeling of the topics.

- **RQ4**: How do discussion topics evolve over the one-year period?

In this research question, we study the evolution of topics in the channel. We study when a new topic emerges and the duration of it. We also study the topics which disappears after few months. The result of this research question can help the relevant stakeholders in many ways. For example, for those who create support document for Python language can give more emphasis on the popular topics. They can give more importance to emerging topics and vice versa.

### 1.4 Contributions

The contributions of this thesis are as follows.

- Creating and sharing a dataset — DISCO¹, A Dataset of Discord Chat Conversations, consisting of the one-year public DIScord chat CONversations of four software development communities for software engineering research. This dataset is a collaborative work of a team of five members.

¹[https://zenodo.org/record/5909202](https://zenodo.org/record/5909202)
• Conducting a comparative study of state-of-the-art topic models in summarizing developer chat conversations.

• Investigating the impact of embedding models on the performance of the BERTopic model for extracting topics.

• Extracting topics related to various discussions for each month between November 2019 and October 2020.

• Analyzing the evolution of topics in the Python’s General Discord channel.

1.5 Thesis Structure

The structure for the rest of the thesis is as follows: Chapter 2 offers background (Section 2.1) and discusses related work (Section 2.2); followed by the presentation of our methodology in Chapter 3. Results of our experiments are presented in Chapter 4, followed by discussion of key finding, implications, and limitations in Chapter 5. The thesis concludes with Chapter 6 by summarizing key contributions in Section 6.1 and discussing future work in Section 6.2.
Chapter 2

Background and Related Work

This chapter describes the background knowledge for summarization and topic modeling (Section 2.1) and the related work corresponding to summarizing developer chat conversations (Section 2.2).

2.1 Background

Automatic text summarization is an inevitable task to process and understand vast amount of textual data which is available in the form of web contents, scientific papers, legal documents, news articles, medical documents, data from software repositories, etc. Automatic text summarization allows users to grasp the important highlights of a document without reviewing the entire document. It can help in increasing the productivity of users as they spent less time reading through the documents or searching for some specific topics. One of the most powerful techniques in text mining and summarization named *topic modeling* is used to extract a general idea of the document or any article by identifying hidden structure present in it. It can also be used for summarizing text documents. The hidden structures in the documents are called *topics* which is a collection of recurring words. There are a number of topic modeling approaches which we discuss next along with a brief explanation of various
text summarization techniques.

2.1.1 Text Summarization

According to [15], a summary is defined as a text that is produced out of one or more texts, that contains a significant portion of the information of the original text(s), and that is no longer than half of the original text(s). It is a difficult task for computers to understand the entire context of a document and find the significant data in it as compared to humans [16]. Scientists have started the research in text summarization as early as 1950, and they are still seeking new techniques to achieve a summary as close to human summary [17].

Based on the summarization approach, automatic text summarization can be divided into extractive and abstractive summarization. Extractive approach extracts the most important sentences from the input text and generates the summary by concatenating them, whereas abstractive approach generates a summary by paraphrasing the contents of the input text [18]. Most of the time, extractive summaries are simpler and more accurate than the latter. Maximal Marginal Relevance (MMR) algorithm suggested by Carbonell et al. [19] is one of the initial methods for sentence selection in extractive text summarization followed by Integer Linear Programming (ILP) [20], submodular based approaches [21], etc. Neural network based models for both extractive and abstractive paradigms became popular afterwards. Nallapati et al. [22][18] proposed abstractive and extractive techniques using recurrent neural networks (RNN) based sequence model for generating multi-sentence summaries. The authors proposed novel models like hierarchical encoders to capture the sentence-word hierarchy, feature-rich encoders for keyword modeling and switching generator-pointer to handle out-of-vocabulary (OOV) words. In order to overcome the repetition of words and to increase the accuracy, See et.al. [23] suggested a hybrid pointer-generator network with coverage mechanism. There is a boom in the use
of pretrained language models for text summarization tasks leveraging transformer
based sequence-to-sequence models \cite{24}. The transformer architecture enables trans-
fer learning for NLP by training language models on unlabelled raw corpus and then
fine-tuning for specific downstream tasks. Pretrained language models like Bert \cite{25},
GPT \cite{26}, Bart \cite{27}, and Pegasus \cite{28} have been very successful in this domain.

Topic modeling based text summarization is another category which is very pop-
ular in multi-document summarization. Latent Dirichlet Allocation (LDA) is a gen-
erative probabilistic model for extracting topics from a text corpora \cite{11}. LDA based
summarization is the most widely used method where the sentences of a document
are clustered based on latent topics. As part of summarization we can select salient
sentences from each cluster which in turn will generate a summary \cite{29} \cite{30}. To get a
high quality summary, the source document should have high topic diversity. There
exists a number of topic modeling techniques which will generate high quality topics.
These techniques are explained in Section \ref{sec:topic-modeling}.

\section{2.1.2 Topic Modeling}

The availability of large amount of information in our day to day life demands Natural
Language Processing (NLP) techniques to organize, understand, and summarize the
unstructured data. Topic modeling methods can be used for this purpose. Topic
modeling identifies latent topics from the collection of documents which reflect the
meaning of the documents. Some of the applications of these unsupervised machine
learning models include analysis of bioinformatics data, social data, environmental
data \cite{31}, data from software artifacts \cite{10}, software engineering research \cite{32}, etc.
A number of topic modeling approaches have been proposed since 1980. However,
we cover four of the topic models in this section which is the focus of this thesis.
Conventional topic models such as Latent Dirichlet Allocation (LDA) \cite{11} and Non-
Negative Matrix Factorization (NMF) \cite{12} along with neural topic models such as
Combined Topic Model (CTM) \cite{13} and Bertopic \cite{14} are explained next.

### 2.1.2.1 Latent Dirichlet Allocation (LDA)

LDA, one of the most popular methods in topic modeling was introduced in 2003 by Blei, Ng and Jordan in 2003 \cite{11}. It is an unsupervised generative probabilistic method for a text corpora. Here the documents that form the corpora are considered as a mixture of latent topics and each topic as a distribution over words. This model also uses Bayesian inference methods on the models. The words with highest probability in a topic gives a good idea of the topic \cite{9}. The words in a document are assumed to be independent of each other and they are represented as Bag of Words (BOW) format. The topic distribution in all documents share a common Dirichlet prior. The word distributions in a topic also share a common Dirichlet prior. In addition to the assumptions that each document is a mixture of topics and each topic is a collection of probability distribution of words, the third assumption is that the number of topics in an LDA should be fixed by the user.

As a first step in implementing LDA, the document needs to be cleaned and preprocessed and represent them as Document Term Matrix where every row is a document and every column is a term (word). LDA converts this into document topic matrix and topic word matrix. These matrices can be processed again to generate the weighted list of topics for each document. The graphical model representation of LDA is given in Figure 1. According to the authors Blei, Ng, and Jordan \cite{11} the LDA model has three hierarchies: corpus-level, document-level, and word-level. The corpus level parameters $\alpha$ and $\beta$ are sampled once while generating a corpus. The document-level variables $\phi_d$ are sampled once per document. The word-level variables $z$ and $w$ are also once for each word in each document.

The generative process of LDA can be explained as follows. Assume that there is a corpus $D$ with $M$ documents and each document $d$ in $M$ has $N_d$ words. $T$ is the
Figure 1: Graphical model representation of LDA. The boxes are “plates” representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document \[\Pi\].

The number of topics the user expects. The hyper parameters \(\alpha\) and \(\beta\) are generated from a Dirichlet allocation distribution on a random base and they control per document topic distribution and per topic word distribution. The generative process consists of the following:

- For a topic \(t\) where \(t \in \{1, \ldots, T\}\), choose a multinomial distribution \(\phi_t\) from a Dirichlet distribution with parameter \(\beta\)
- For a document \(d\) where \(d \in \{1, \ldots, M\}\), choose a multinomial distribution \(\theta_d\) from a Dirichlet distribution with parameter \(\alpha\)
- For a word \(w_n\) where \(n \in \{1, 2, \ldots, Nd\}\) select a topic \(Z_n\) from \(\theta_d\) and a word \(w_n\) from \(\phi_{z,n}\).

The probability of the corpus \(D\) can be written based on the equations before as follows:

\[
p(D \mid \alpha, \beta) = \prod_{d=1}^{M} \int p(\theta_d \mid \alpha) \left( \sum_{n=1}^{Nd} p(z_{dn} \mid \theta_d) p(w_{dn} \mid z_{dn}, \beta) \right) d\theta_d \tag{1}
\]
Figure 2: Non-negative matrix factorization: $D \approx UV$, with $U$ and $V$ element wise non-negative. [36]

The final objective is to find the most optimum document-topic distribution and topic-word distribution. In order to find the parameters in LDA various methods have been proposed such as Gibbs sampling [33], Expectation Propagation [34], and Variational Bayes Inference [11].

### 2.1.2.2 Non-negative Matrix Factorization (NMF)

Non-negative Matrix Factorization (NMF) [35] algorithm belongs to linear algebraic optimization algorithms which is used for dimensionality reduction of input data using factor analysis methods. Here the high dimensional vectors are decomposed to non-negative lower dimensional vectors. NMF, which is an unsupervised algorithm can also be used for topic modeling where the input text corpus is represented in an encoded TF-IDF term-document matrix format. This input is decomposed into two matrices namely term–topic matrix and topic–document matrix. These matrices are randomly initialized and the NMF algorithm can be run iteratively to find these matrices with minimum cost. The cost function is calculated by using Frobenius norm. The NMF can be shown diagrammatically as shown in Figure 2.

The optimization algorithm can be explained as follows:

$$\min \|D - UV\|_F^2$$  \hspace{1cm} (2)
where $D$ is the term-document matrix with $M$ terms and $N$ documents, $U$ is the term-topic matrix with $M$ terms and $K$ latent topics and $V$ is the topic-document matrix with $K$ topics and $N$ documents. The Multiplicative Update learning algorithm is represented by the equations as follows.

\begin{align*}
U & \leftarrow U \frac{DV^T}{UVV^T} \\
V & \leftarrow V \frac{U^T D}{U^T UV}
\end{align*}

These equations can be iterated until convergence to obtain the final $U$ and $V$ matrix.

### 2.1.2.3 Combined Topic Model (CTM)

Topic coherency is an important factor in deciding the quality of a topic model. Syntactic and semantic analysis can improve the coherency of topic models and Bag of Words model do not take these factors into consideration. The topics generated by the Combined Topic Model (CTM) model has high coherency and diversity compared to traditional Bag of Words (BoW).

There are two components in this model which is a neural topic model ProdLDA and Sentence-Bert (SBERT) embeddings. ProdLDA is based on Variational AutoEncoder. The choice of neural topic model and the pre-trained embedding representation can be agnostic assuming the neural topic model is based on auto encoder and pre-trained embeddings are capable of embedding documents. The variational framework trains a neural network to obtain a latent representation from the BoW. The decoder reconstructs the BoW from the latent document representation.
frame work uses Gaussian distribution to approximate Dirichlet prior. This architecture is extended by adding contextualized embeddings from SBERT which is an extension of BERT that generates sentence embeddings.

The high level architecture of CTM is presented in Figure 3. Here the document embeddings from SBERT are projected to a hidden layer having the same dimension as the vocabulary size. It is further concatenated with BoW representations. A decoder reconstructs this BoW from document-topic representation learned by the ProdLDA model using the parameter $\mu$ and $\sigma^2$ of a Gaussian distribution. In their work [13], the authors have evaluated the model using two metrics for topic coherence: normalized pointwise mutual information (NPMI) & a word-embedding based coherence measure. They also used topic diversity as another evaluation metric. Upon
evaluation this model proved to be very competitive compared to its contemporary topic models.

2.1.2.4 BERTopic

BERTopic is a topic modeling technique that leverages transformers and class based TF-IDF to generate interpretable topics from clusters of the documents \[14\]. Recently, researchers started taking advantage of the contextual embeddings generated by transformers \[25, 40, 41\] and its variations in topic modeling \[42\]. Some of the recent topic models like Top2Vec \[43\] and model by Sia et al. \[42\], the documents are clustered and the words close to the centroid of the cluster are extracted as the topics. However, this centroid-based perspective for topic modeling can lead to misleading topics \[14\]. This is because the clusters may not always fall within a sphere around a centroid. This problem is overcome in BERTopic by using \textit{c-TF-IDF}. The working of BERTopic can be explained in three steps as shown in Figure 4.

The first step is to create a document level embeddings using pre-trained language model like Sentence-BERT (SBERT) \[39\]. The second step is to perform a dimensionality reduction of these embeddings using Uniform Manifold Approximation and Projection (UMAP) \[44\] and then further cluster it to semantically similar clusters using Hierarchical Density Based clustering (HDBSCAN) \[45\]. The third step is to apply class based TF-IDF to extract topics from these clusters. To get coherent diversified topics, a Maximal Marginal reference algorithm can be applied as an optional step. These steps are explained in detail below.

1. **Document embeddings.** As a first step the model generates document embeddings using Sentence-BERT (SBERT) framework. This framework allows the embeddings of sentences and paragraphs to be represented as dense vectors using pretrained language models. The basic assumption is that the documents which are semantically similar in the embedding space may have the
same topics. This framework allows to choose any sentence-transformer model for generating the embeddings. Similar to CTM, this model is also agnostic to the embedding model. In addition to sentence-transformer models, the implementation of pretrained embeddings in BERTopic can also be done with the help of NLP libraries like Flair [46] and Gensim [47]. The default embedding models available in BERTopic are from sentence-transformer models such as all-MiniLM-L6-v2 and paraphrase-multilingual-MiniLM-L12-v2. The first model is an English language model trained for semantic similarity tasks whereas the second one is used for multilingual models. BERTopic also provides an option to use custom embeddings as per the user’s requirements.

2. **Document clustering.** To overcome the curse of dimensionality, a dimensionality reduction technique UMAP [44] is performed on the embeddings. UMAP
is selected due to two reasons (a) it preserves local and global features of high-dimensional data even after dimensionality reduction (b) there is no restriction on the dimension of embedding space. As a result it can be used to different language models with varying embedding dimensions. After the dimensionality reduction the clustering of documents is done with the help of HDBSCAN [48] an extension of density based clustering, which uses a soft clustering method and also identifies the outliers which are present.

3. **Topic extraction.** Topic extraction is performed with the help of doing a class-based TF-IDF on the clusters of document. In a traditional TF-IDF [49], the importance score of words is calculated across the documents. In BERTopic, the importance scores of the words within a cluster is calculated by considering a cluster as a single document. All documents in a cluster are simply concatenated together and considered as a single document. Then the class based TF-IDF is calculated as per the equation given below.

$$W_{t,c} = t f_{t,c} \cdot \log \left( 1 + \frac{A}{t f_t} \right)$$

(5)

Here $t f_{t,c}$ represents the frequency of term $t$ within a cluster (class) $c$. The inverse document frequency is the next term which is the logarithm of the average number of words per class $A$ divided by the frequency of term $t$ across all classes. Finally add one to the division within the logarithm to get only positive output values. To get a user specified number of topics iteratively merge the least common topic to its similar one.

BERTopic also leverages the idea of dynamic topic modeling to model the evolution of topics over time. The experiments conducted by the authors [14] shows that this topic model is very competitive to its contemporaries and is stable.
2.2 Related Work

The related work section is divided into two areas — discussion of summarization of software engineering artifacts and analysis of chats in online discussion forums like Slack, Gitter, GitHub, etc. which are presented in Section 2.2.1 and Section 2.2.2 respectively.

2.2.1 Summarizing Software Engineering Artifacts

Summarizing software artifacts focuses on bug reports, source code, mailing lists and developer discussions. Dialogue summarization is a domain which is gaining popularity due to the availability of high volume conversational data obtained when people use digital platforms and smartphones to exchange information. The dialogue domain can contain emails, meetings, online chats, customer service interactions, medical conversations between doctors and patients, podcasts, etc. [50].

Due to the COVID-19 pandemic, software companies are focusing on online communication platforms for their project related discussions. Summarizing developer conversations can be advantageous to get an understanding of the discussions happening between software developers. Summarization task allows developers to search and extract specific information from the artifacts rapidly. It also allows to organize information effectively thereby saving time and resources. Recent work on summarization of software artifacts focuses on summarization of bug reports, source code, mailing lists, and developer discussions.

2.2.1.1 Summarization of Bug Reports

Bug report summarization helps developers to save time when they are trying to solve a problem at hand. This task allows the developers to search for a similar problem
from the past and identify its solution. Majority of the work on bug report summarization focuses on extractive summarization [51]. Rastkar et al. [52] [53] investigated the possibility of summarizing bug reports which allows the developers to consult the generated summary instead of the entire artifact. Their results suggested that a summary generator trained specifically on bug reports can produce better summary. They also suggested that the existing conversation-based extractive summary generators can produce better summaries compared to a random classifier. The machine learning algorithm is logistic regression, and they calculated the probability of each sentence to be part of the final extractive summary. Final set of sentences are selected based on sorted score of the probability values. Sentences were selected until 25 percentage of the bug report word count was reached. Jiang et al. [54] proposed a new supervised algorithm which resulted in a better bug report summarization named Logistic Regression with Crowd-sourced Attributes (LRCA). They developed a new tool named Crowd-sourcing Software Engineering Platform to infer new effective attributes from the crowd-generated data.

Mani et al. [55] suggested unsupervised approach for bug report summarization with noise reduction. The presence of noise reduce the quality of summaries considerably. The methodology used here can be explained in two simple steps. Initially the bug report is passed through a noise reducer module where the classification of a sentence into question, investigative sentence, code fragment and others is performed. These classified and filtered sentences are passed to an unsupervised summarizer module to select useful sentences. The sentence filtering process filters out the selected class of sentences for multiple experiments. The unsupervised summarizer consists of four different techniques for summarization: Centroid based [56], MMR [19], DivRank [57], and Grasshopper [58].

Yang et al. [59] have extended the model AUSUM suggested by Mani et al. [55] by
adding two new classes in the noise module: anthropogenic and procedural information. There are two layers in this Two-layer Semantic Model (TSM) where the first layer is a semantic filtering model to filter out informative sentences and the second layer is a bug report classifier based feature extraction. Based on these features the training of summarizer is performed. DeepSum is the first deep learning based summarizer with a stepped auto-encoder network. This model incorporates bug report characteristics into a deep neural network. The basic idea behind this summarization is that the input features of the document will be represented in a compressed form in the hidden layers of a deep learning module. A sentence selecting algorithm can then be used to select salient sentences based on the weightage of the sentences. The bug report characteristics can be integrated using evaluation enhancement and predefined fields enhancement modules. Jindal et al. proposed a new unsupervised approach based on keyword-based features and sentence-based features. Two methods term frequency-inverse document frequency (TF-IDF) and Rapid Automatic Keyword Extraction (RAKE) are used for keyword extraction.

2.2.1.2 Summarization of API Related Data

Summarizing API reviews is another useful summarization task for developers. Uddin et al. have investigated the advantages of summarizing API reviews which can be concise, informative and provide a quick insight about the APIs. The authors presented two algorithms - statistical and aspect-based to summarize opinions about APIs. An online opinion summarization engine, Opiner is developed as a search engine to present the summaries of opinions. In addition to the proposed algorithms, Opiner uses 6 off-the-shelf techniques to generate the summaries of opinions. These algorithms are (a) topic-based summarization based on LDA (b) contrastive viewpoint summary based on technique proposed by Kim and Zhai (c) extractive summarization algorithms such as Luhn, Lexrank, and Textrank, and
(d) abstractive summarization using Opinosis [68], a domain-independent abstractive opinion summarization engine. In the paper [69], the authors proposed an unsupervised API summarization using extractive summarization algorithm TextRank [67]. Here, Stack Overflow posts are considered as an unofficial documentation for these APIs.

KG-APISumm [70] is a hybrid extractive summarizer for API classes based on a query. The summaries generated are task-specific as the methods and sentences generated are related to the query. This means that for a single class different summaries can be generated based on the queries. Here the summarizer is hybrid because it can extract sentences from API reference document as well as the method names. The input to the summarizer are a natural language user query $Q$, describing the developer task, a class $C$ from an existing library $L$, and the API knowledge graph of the library $L$ (API KG($L$)).

2.2.1.3 Summarization of Source Code

To make people understand a program easily the explanation of the logic and functions of source code in natural language is needed [71]. For program comprehension and maintenance a high quality source code summarization is required. Initially, the code summarization was performed using keyword extraction from source code and by building Bag of Words model (BoW). Recently the code summarization task has been focusing on deep learning models including Recurrent Neural Network (RNN), Long Short Term Memory model (LSTM), Gated Recurrent Unit (GRU), Convolution Neural Network (CNN), transformer based models, etc. Attention mechanism can also aid in generating efficient summaries [72]. We can categorize source code summarization in (a) Manually-Crafted Templates-Based, (b) IR-Based, and (c) DL-Based automatic source code summarization Generation [72].

Manually-crafted templates based source code summarization is one of the earliest
methods and this model uses heuristics based approach for generating summaries \cite{73, 74}. The Information Retrieval (IR) based approach searches for keywords of source code or comments of similar code. This approach checks the correlation between the target code and the similar code. The best matching code’s summary will be returned as the target summary. Some of the approaches of IR-based modeling are Latent Semantic Indexing (LSI), Vector Space Model (VSM), and Latent Dirichlet Allocation (LDA) \cite{75–79}.

Some of the important models using DL-based source code summarization is explained below. CODE-NN \cite{80} is one of the benchmark model for many code summarization papers. This model could generate a description of C# and SQL sequences using LSTM and attention mechanism. One such model which follows CODE-NN as benchmark model is Code2Vec \cite{81} and it represents source code using Abstract Syntax Tree (AST) path sets. Woo et al. \cite{82} has proposed an algorithm called CBAM which improved the representation ability of CNN. The models DeepCom and hybrid Deep-Com \cite{83, 84} are seq2seq models proposed to generate Java method summarization based on Attention mechanism. To overcome the disadvantages of seq2seq models, recent code summarization models started leveraging transformer based models \cite{85, 87}. Reinforcement learning framework also became popular in this domain \cite{88, 89}. Wang et al. \cite{89} used a hierarchical attention network along with an actor-critic reinforcement learning.

2.2.1.4 Summarization of Developer Discussions

The relevant works based on developer communications on online discussion platforms like Slack, Gitter, Stack Overflow are discussed here. Collabot – a chat assistant service \cite{90} is a personalized group chat summarizer that implicitly learns users interests and social ties within a chat group. For a collaborative work in a team, tools like Slack, Microsoft Teams, etc. are crucial. Now, people from all around the world can
work together remotely using these communication platforms. Due to the information overload, it is possible that some users may miss parts of the discussions. Collabot can be considered as a solution for this by offering personalized group chat summary. The first step in this process is creating a user’s profile by learning the topics of interests of user and their social ties from the chat history. Next, the transcripts are divided into conversations by conversation disentanglement which is heuristics based. Using user’s profile the conversations are scored and lowest scoring conversations are filtered out. Next step is clustering of the conversations with the help of topic modeling and each conversation is given a tag (title) based on the highest probability words in the topic. Finally, the important sentences having either the title words or actionable items such as questions and commitments are chosen from the conversations according to some predefined length.

AnswerBot [91][92], an answer summary generation tool based on Stack Overflow is yet another important work under summarization. AnswerBot automatically generates an answer summary for a technical problem. The summarization task is a query-focused multi-answer-posts summarization. AnswerBot is implemented as a three-stage framework: 1) relevant question retrieval where a ranked list of relevant questions are retrieved based on a query; 2) useful answer paragraph selection - in this stage all the answer paragraphs are chosen from the repository. They are ranked based on the query and salient paragraphs are chosen. These paragraphs are chosen based on three features - query related features, paragraph content features and user oriented features; 3) diverse answer summary generation - from the selected paragraphs a Maximal Marginal Relevance (MMR) algorithm is applied to chose the relevant answers. The evaluation of AnswerBot is conducted by building a repository which includes a large number of Java questions and their corresponding answers from Stack Overflow.

GitterAns [93] is an answer bot for troubleshooting technical questions asked in an
online Gitter chat and provides relevant answers from Stack Overflow posts. The three parts in GitterAns is question detection, searching for answers on Stack Overflow, and answer processing. The question detection consists of a machine learning classifier which classifies the question as a trouble shooting question or not. Three machine learning algorithms, namely Naïve Bayes, Random Forrest, and Stochastic Gradient Descent are used in this stage. The next stage is answer search done with Google CustomSearch API where the relevant Stack Overflow posts are returned as a JSON object for the query. In the final step, the answers are parsed and returned as a dictionary of links and post titles.

Ren et al. [94] in their paper investigated controversial discussions in Stack Overflow. Their studies revealed that many answers in Stack Overflow are not optimal, wrong or out-of-date. The authors designed an automatic open information extraction approach for systematically discovering and summarizing the controversies in Stack Overflow. To understand the controversies they have exploited the official API documentation. The empirical study of these controversial discussions was conducted in Java/Android-tagged Stack Overflow questions and answers. The approach consists of discovering controversies including controversial answers and critique posts, explain API-related controversies, and summarize controversies as salient, concise, semi-structured warnings. Käfer [95] has proposed a research plan for summarizing different communication sources into one big summary using and improving existing text summarization approaches which includes chat messages. Coata [96] proposed that the knowledge found in developer instant communication can be reused to help developers. The author also suggested techniques for identifying and summarizing this knowledge from unstructured data.

Information retrieval (IR) approaches are used in many software engineering problems like summary, link recovery, and software reuse [97]. The methods used for IR are
based on vector space model (VSM), Latent Semantic Indexing (LSI), Latent Dirichlet Indexing (LDI), TF-IDF (term frequency-inverse document frequency) methods etc. This thesis focuses on summarizing chat data using topic modeling approaches. Additionally it focuses on summarizing developer conversations on a Discord channel which is first of its kind. Most of the research work on developer discussion summarization is focusing on Slack, Gitter, or Stack Overflow data. Summarizing short conversation text is highly challenging because of its unstructured nature and can be one of the contributing factors behind the lack of extensive research work in this area.

2.2.2 Analysis of Chat Data

In this section, recent works exploring chat data from online discussion platforms are explained. Software engineering researchers have analyzed chat data from online discussion platforms like Gitter, Slack, GitHub, etc. to understand the interaction of developers, the topics discussed, the communication style of developers etc. Some of the works have studied Slack data to understand their use in software engineering. Chatterjee et al. [3, 98–100] assessed Slack public Q&A chat as a mining source for supporting software maintenance and evolution tools. They studied the potential usefulness and challenges of mining such conversations. The authors found that Q&A chats from Slack contain same information as in Q&A posts on Stack Overflow but in lesser quantities. The Slack Q&A chats contain more information on API mentions than in the Stack Overflow Q&A posts. Chatterjee et al. [2] also presented a dataset of software-related chat conversations extracted from three open Slack communities that led to many research works. Another interesting work from Chatterjee et al. [100] is extraction of opinion Q&A from online developer chats by developing automatic identification of opinion-asking questions and extraction of participants’ answers. The online developer chats were extracted from Slack and IRC channels. Wang et al. [101] have also studied the Slack channel groups to analyze the communication style and
the relation between communication style and team performance.

Shi et al. [102] have explored the live chat of developers from eight Gitter communities and offered an understanding of developer communication profiles, community structures, discussion topics, and interaction patterns. Their studies have identified the days where the developers communicated the most, the most discussed topics such as API usages and errors and six dialog interaction pattern. Sahar et al. [103] have performed a study of issue reports and the resolution time of issues from 24 open source Gitter project chat rooms. Their findings show the importance of Gitter chat rooms and identifies them as a rich data source for information about the issue resolution process in open source system.

Shihab et al. [104] have studied IRC meetings to understand the usefulness of meeting data. Their study focused on three dimensions: content, meeting participants, and their communication style. Panichella et al. [105] have also explored IRC chat logs, mailing lists, and issue trackers and investigated collaboration links between them. Alkadi et al. [106, 107] have explored the rationale behind decisions during software development by exploring chat messages. They investigated the frequency of rationale and performed content analysis and machine learning techniques on chat messages from three software development projects. This thesis is analysing chat data from a discord channel which has not been explored before.
Chapter 3

Methodology

This chapter describes the methodology followed for summarizing the developer conversations. The overall workflow of this thesis is presented in Figure 5. The first part of this is detailed explanation of data collection from four Discord server public channels followed by data cleaning and preprocessing that is explained in Section 3.1. The data selected for this thesis consists of conversations from the channel python#python-general starting from November 2019 to October 2020. The dataset is cleaned and preprocessed as the next step which is explained in Section 3.2. After the preprocessing step the conversations are grouped together based on the conversation ID created as part of the disentanglement process. In the next step, as an initial experiment three months of data are selected randomly which is for the months December 2019, March 2020, and July 2020 and topic models such as LDA, NMF, CTM, and BERTopic are evaluated. The basic idea behind this process is to identify the topic model which produces high quality topics and use this model to extract topics from the data in a monthly basis. This evaluation process is described in Section 3.3. After this process, BERTopic is selected and evaluated with various sentence embedding techniques which are explained in Section 3.3.4.1. Finally, the metrics used for evaluating topic models and the manual labeling of topics are explained in Section 3.4 and Section 3.5 respectively.
3.1 Dataset: DISCO

Recently software development communities are depending on online chat platforms such as Discord, Slack, IRC, Gitter, Microsoft Teams, etc. for their project collaboration and related communications. These collaborative platforms are a rich source of technical knowledge which can provide technical help for developers. These platforms help the developers to share knowledge with their peers and facilitate real-time conversations among community members. Even with these advantages, there are only limited studies on mining these chat conversations compared to the studies on mining emails and bug reports [108], tutorials [109], and Q&A forums [110–113]. Some of the recent works are by Chatterjee et al. [2, 3] who have mined and studied Slack chat conversations; their results show that these conversations contain valuable information such as code snippets’ description and APIs, bug debugging techniques, best programming practices, and causes of common errors/exceptions. Another related
work is by Esteban et al. [4] who have studied Gitter data to help new developers get familiar with software products. There have not been any studies related to mining of Discord server data.

Since Discord server is a public chat platform with thousands of users all over the world, it contains a wide variety of discussions which have not been exploited yet. Mining Discord conversation would provide numerous research opportunities to help software communities. We have collected and curated a dataset called DISCO [7] consisting of the one-year public DIScord chat conversations of four software development communities. We have collected the chat data of the channels containing general programming Q&A discussions from the four Discord servers, applied a disentanglement technique [114] to extract conversations from the chat transcripts, and performed a manual validation of conversations on a random sample (500 conversations). The dataset consists of 28,712 conversations, 1,508,093 messages posted by 323,562 users. The dataset collection was a collaborative effort of a team of five members.

There are over 150 million active monthly users [5] in Discord in 2022 and 78% participants claim to use Discord for non-gaming activities [115]. The Discord channel conversations are short text and they follow an informal, unstructured, and asynchronous format. The conversation length might range from 2 messages to 100s spanning with numerous participating users. Since the conversations are not always continuous and are entwined with each other, the conversations need to be subjected to a technique to separate or disentangle them. The overall process of data selection, collection, and chat disentanglement is shown in Figure [6].

Initially the chat transcripts are downloaded from the selected channels in JSON format using a date range. From this only helpful information such as timestamp, user name, and message content are retained. It was followed by conversion into XML format and later by anonymizing the usernames in XML. This is to ensure
the privacy of the users that eliminates the possibility of identifying the original Discord users. Since the messages are entwined, the disentanglement algorithm [2] was leveraged to extract disentangled Discord conversations (in XML format). The final dataset includes an additional computed attribute, \texttt{conversation id}, which explains the conversation ID for each utterance. For this thesis work, the data is selected from the channel \texttt{python\#python-general} which consists of twelve months of data starting from November 2019 to October 2020.

### 3.1.1 Data Selection

The reason behind choosing Discord over other chat platforms and the channels selected for data selection are explained in this section. The channels selected are public server channels as the source in creating the dataset. The data from these channels can support interesting research opportunities and tool development. This data complements Slack data extracted by Chatterjee et al. [2] to foster further research on studying distributed software development communities, communication among community and team members, informal documentation, etc.

Due to the ease of setting up a Discord server and the unlimited preservation of
historical chat data, many software development communities have started to migrate their communication from Slack to Discord [116]. Discord is more popular than Slack because it has 14 million daily active users whereas Slack has around 10 million [117, 118]. One of the main reasons for migrating from Slack to Discord is that Slack’s free plan supports only 10,000 of the most recent messages to be searched and viewed. Even though many started migrating, some communities continue to maintain both communication mediums [119, 120]. Another popular platform is Gitter which is an instant messaging and chat platform designed for GitHub and GitLab users where the discussions are happened on specific projects. Due to the popularity and amount of messages in Discord it is highly important that the conversation data from 150 million active Discord users is collected and available to researchers.

The Discord servers selected for creating the dataset are for four programming languages such as Python, Go (or GoLang), Racket, and Clojure. These channels demonstrate a good daily activity and a substantial number of members (e.g., Python Discord server has a total of 300,919 members) compared to other available Discord programming servers. Joining these servers are comparatively easy as anyone with a Discord user ID can join these servers as they are publicly visible. Discord users can start asking general help or technical questions on these channels without any delay. We identified the following server channels that follow a Q&A format and offer general technical help including python#python-general, gophers#golang, racket#general, and clojurians#clojure for our data collection and conversation disentanglement process. The channels selected were similar to the datasets curated by Chatterjee et al. to allow triangulation with their studies.

3.1.2 Data Collection and Preprocessing

The data is exported using an open-source application, Discord Chat Exporter [121] in JSON format within a specific date range. The data is collected for a year from
Table 1: Dataset of disentangled Discord conversations.

<table>
<thead>
<tr>
<th>Channel</th>
<th>#Conversations</th>
<th>#Utterances</th>
<th>#Users</th>
<th>Avg CL</th>
</tr>
</thead>
<tbody>
<tr>
<td>python#python-general</td>
<td>19,155</td>
<td>1,254,362</td>
<td>300,919</td>
<td>57.49</td>
</tr>
<tr>
<td>gophers#golang</td>
<td>8,860</td>
<td>247,179</td>
<td>19,983</td>
<td>27.47</td>
</tr>
<tr>
<td>racket#general</td>
<td>538</td>
<td>4,975</td>
<td>917</td>
<td>8.95</td>
</tr>
<tr>
<td>clojurians#clojure</td>
<td>159</td>
<td>1,577</td>
<td>1,743</td>
<td>9.99</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>28,712</strong></td>
<td><strong>1,508,093</strong></td>
<td><strong>323,562</strong></td>
<td></td>
</tr>
</tbody>
</table>

November 2019 to October 2020 for three channels (Python, Clojure, Racket) and for gophers#golang the date range is November 2019 to September 2020 due to our University’s Fair Dealing Policy in using public copyrighted data for research purposes [122].

Next, the collected Discord chat transcripts in JSON format are then converted to XML files. All the unnecessary information such as the user-related details, reactions on the messages, etc. are filtered out from the resulting XML file during the conversion. Three tags are maintained in each message such as a timestamp, the ID of the user and the message text. To preserve the privacy of the users, the user IDs are then anonymized using the randomly selected person names. The dataset size is reported in Table 1 in terms of metrics such as the number of conversations, utterances, users, and average conversation length (i.e., Avg CL) for each channel. The average conversation length is measured in terms of sentences.

### 3.1.3 Chat Disentanglement

In multiparty conversational forums, the conversations, either formal or informal, happen between the users simultaneously. To use this data for applications such as question answering, response selection, or while creating a dataset for dialogue systems [123], the intertwined chat data needs to be disentangled. Chat disentanglement process identifies separate threads in the conversation. An example of a preprocessed
Discord XML file is shown in Figure 7. In this XML snippet there are two separate conversations which are entangled with each other. While the 1st conversation is in progress, the 2nd question is asked. And before a relevant reply to 2nd question is given, the 1st conversation continued. This conversation needs to be disentangled to ease the process of chat mining.

Some of the previous works on the disentanglement techniques are for IRC [124], Slack [2, 125] and Gitter [4]. One of the earliest work in chat disentanglement is by Elsner and Charniak [114] who used a supervised model (maximum-entropy classifier) that considers the time frame and features between the message pairs. In addition to this, the user similarity between the message pairs, cue words, similar word usage, and technical expressions while disentangling the chats were also considered. Chatterjee et al. [2] leveraged the well-known Elsner and Charniak disentanglement technique with some modifications in their research work for Slack data. Here, the modification is done on the feature computation between the message utterances when compared to the original method.

The modifications are as follows. Gratitude words such as “thanks”, “this works”, “makes sense” are added in the modified algorithm. When the time frame of $\leq 1477$ (1.518) seconds is observed between the message utterances, or when the utterance is within the last 5 messages from one another, the features were calculated. With these modifications, the classifier is then trained on 500 manually disentangled Slack conversations. The disentanglement technique by Chatterjee et al. [2] is used for the Discord data since both Slack and Discord channels follow the same conversation format.

Micro-averaged F-score is considered to be a better metric to evaluate the disentanglement process than the standard F-measure as the annotators can have disagreement while creating the gold set of disentangled chat conversations [114]. For this, two annotators [7] have calculated the micro-averaged F-score by selecting a random
Figure 7: Format of the chat conversation data.

block of 500 Discord messages extracted from the Python channel and then manually
disentangling it. The average F-score is 0.79 which is higher than the F-score of 0.66
reported by Elsner and Charniak and also similar to the one reported by Chatterjee
et al. [2] for Slack disentanglement (i.e., F-score of 0.80). This result further supports
our observation that Slack and Discord follow similar chat conversation patterns, and
hence the same disentanglement algorithm used on Slack could be applied for Discord.
3.2 Data Preprocessing for Topic Modeling

The initial step of a topic modeling pipeline is data preprocessing of the datasets. The evaluation of topic models which includes running the experiment, validating the results, and preprocessing the data is conducted with the help of an open-source python package, OCTIS (Optimizing and Comparing Topic models is Simple) [126]. From the DISCO dataset, only the Python channel is selected for this work, and a CSV file is created from the XML file filtering out the unnecessary fields. In the interest of this thesis, only relevant fields such as conversation_id and text are retained filtering out the remaining fields. Using the conversation_id, each thread of conversation can be identified, and the respective conversations are clustered together by a simple groupby function in Python. The data preprocessing is applied only to the text field which is relevant for the evaluation. OCTIS framework includes utilities for the preprocessing steps. Additionally some extra preprocessing steps were also applied such as removing (a) the user name mentioned in the conversation, (b) emojis, and (c) added more conversation specific stop words such as “lol”, “folks”, “guys”, etc. The preprocessing steps are explained below.

1. Converting the text to lower case. Converting the text to lower case is an important step to avoid the duplicate problem. For example the words “Android” and “android” are considered as two different words by the machine. This is one of the simplest and efficient method in preprocessing which can improve the accuracy of results.

2. Punctuation removal: Punctuation marks are used to divide the text into phrases or sentences or even paragraphs. Since they appear frequently in the text, it can affect the results of any text processing approach which depends on the frequencies of words or phrase occurrence. The punctuation can increase the noise in the text and affect the quality of training.
3. **Stop word removal**: In the text document there are a group of words which occur frequently such as articles, determiners, and prepositions. These words are called *stop-words* which are meaningless, and by removing them the NLP model can focus on the important words in the text. Stop-word removal can reduce the size of the dataset considerably, hence reduce the training time, and also improve the efficiency of whole model. Additional stop-words can also be specified depending on the dataset and the NLP application.

4. **Lemmatization**: OCTIS also gives an option to lemmatize the sentences. It is a technique that reduces a word to its base root mode called *lemma*. This process is similar to stemming, but the meaning of the word is preserved in this process.

5. **Removal of short documents**: In this step, the conversations which has less than five words are removed.

The dataset after preprocessing contains an average vocabulary size of 24,146 words and average number of tokens per document as 440. Vocabulary denotes the number of unique words in a corpus. Here, the tokens refer to the number of words per document and the average value is calculated across 12 months. The statistics of the dataset are represented in Table 2.

### 3.3 Evaluation of Topic Models

The overall workflow of this thesis is shown in Figure 5. The next process in this pipeline is evaluating the topic models such as LDA, NMF, and the neural topic models such as CTM and BERTopic. A detailed explanation of these topic models is given in Section 2.1.2. The evaluation of the topic models is performed for three months of data with the aim of selecting one topic model which can provide high quality topics. The framework for this process is a modified version of OCTIS by Grootendorst.
by adding the functionality for BERTopic model to it. The open-source evaluation framework OCTIS [126] can be used for a comparative study of state-of-the-art topic models. This framework allows the user to perform a hyper-parameter optimization for an unbiased comparison. The main functionalities of this package is preprocessing, training topic models, estimating evaluation metrics, hyperparameter optimization, and interactive web dashboard visualization [126]. However, following the footsteps of Grootendorst, a hyper parameter optimization was not performed in this thesis. LDA and NMF are run through OCTIS with default parameters, while “all-mpnet-base-v2” is used as the sentence embedding model for both CTM and BERTopic. The overall workflow of OCTIS is shown in Figure 8.
3.3.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is implemented using Python’s Gensim package [47] within the OCTIS framework. The LDA model is trained with the default parameters set in the OCTIS model. LDA and all other topic models are implemented with topics ranging from 10 to 50 with steps of 10. The evaluation metrics topic coherence and topic diversity are calculated for each step. This is continued for three runs and the final results are obtained by taking the average across 3 runs. The two important inputs to the LDA model are the dictionary and the corpus. The corpus is a mapping of word id and word frequency. The word id parameter is the unique ID for each word in the text document. This model also requires number of topics to be specified in advance. The number of topics can be decided based on the dataset and the application. The default parameters and their values are reported in Table 3. The explanation of each parameter is presented below.
Table 3: Default parameters for LDA.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_topics</td>
<td>100</td>
</tr>
<tr>
<td>distributed</td>
<td>False</td>
</tr>
<tr>
<td>chunksize</td>
<td>2000</td>
</tr>
<tr>
<td>passes</td>
<td>1</td>
</tr>
<tr>
<td>update_every</td>
<td>1</td>
</tr>
<tr>
<td>alpha</td>
<td>symmetric</td>
</tr>
<tr>
<td>eta</td>
<td>None</td>
</tr>
<tr>
<td>decay</td>
<td>0.5</td>
</tr>
<tr>
<td>offset</td>
<td>1.0</td>
</tr>
<tr>
<td>eval_every</td>
<td>10</td>
</tr>
<tr>
<td>iterations</td>
<td>50</td>
</tr>
<tr>
<td>gamma_threshold</td>
<td>0.001</td>
</tr>
<tr>
<td>random_state</td>
<td>None</td>
</tr>
</tbody>
</table>

1. **num_topics**: denotes the number of requested latent topics to be extracted from the training corpus.

2. **distributed**: defines whether a distributed computing is required or not to accelerate training.

3. **chunksize**: controls how many documents are processed at a time in the training algorithm.

4. **passes**: defines the number of passes through the corpus during training.

5. **update_every**: denotes the number of documents to be iterated through for each update. For batch learning set, it is as 0, for online iterative learning set, it is $> 1$.

6. **alpha**: defines an a-priori belief on document-topic distribution. The default is “symmetric” which means it uses a fixed symmetric prior of $1.0 / \text{num\_topics}$.
7. *eta*: is a-priori belief on topic-word distribution. The hyperparameters *alpha* and *eta* affect the sparsity of the topics.

8. *decay*: a number between [0.5, 1] to weight what percentage of the previous lambda value is forgotten when each new document is examined.

9. *offset*: is a hyper-parameter that slows down the early iterations of the Online Variational Bayes LDA algorithm [127].

10. *eval_every*: the log perplexity is estimated every that many updates.

11. *iterations*: defines the maximum number of iterations through the corpus when inferring the topic distribution of a corpus.

12. *gamma_threshold*: is a minimum change in the value of the gamma parameters to continue iterating.

13. *random_state*: is a parameter for setting a randomState object or a seed to generate one.

After the training is completed, the results are obtained in a dictionary format with three entries such as “topics”, “topic-word-matrix”, and “topic-document-matrix”.

### 3.3.2 NMF

The implementation of NMF is similar to LDA in the OCTIS framework. This model is ran for three iterations where the topic size ranges from 10 to 50 with a step size of 10. The metrics are evaluated at each step and the final results are the aggregated results of three runs. The parameters and their default values for this model are presented in Table [4].

Each parameter is defined as follows.
Table 4: Default parameters for NMF.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_topics</td>
<td>100</td>
</tr>
<tr>
<td>chunksize</td>
<td>2000</td>
</tr>
<tr>
<td>passes</td>
<td>1</td>
</tr>
<tr>
<td>kappa</td>
<td>1.0</td>
</tr>
<tr>
<td>minimum probability</td>
<td>0.01</td>
</tr>
<tr>
<td>w_max_iter</td>
<td>200</td>
</tr>
<tr>
<td>w_stop_condition</td>
<td>0.0001</td>
</tr>
<tr>
<td>h_max_iter</td>
<td>50</td>
</tr>
<tr>
<td>h_stop_condition</td>
<td>0.001</td>
</tr>
<tr>
<td>eval_every</td>
<td>10</td>
</tr>
<tr>
<td>normalize</td>
<td>True</td>
</tr>
<tr>
<td>random_state</td>
<td>None</td>
</tr>
<tr>
<td>use_partitions</td>
<td>True</td>
</tr>
</tbody>
</table>

1. *num_topics*: denotes a number of requested topics to be extracted from the training corpus.

2. *chunksize*: denotes the number of documents to be used in each training chunk.

3. *passes*: defines the number of full passes through the corpus during training.


5. *minimum_probability*: It depends on the value of the parameter *normalize*. If *normalize* is “True”, topics with smaller probabilities are filtered out.

6. *w_max_iter*: It defines the maximum number of iterations to train \( \mathbf{W} \) per each batch.

7. *w_stop_condition*: It gives the condition to stop the training for the current batch. If error difference gets less than that, training of \( \mathbf{W} \) stops for the current batch.
batch

8. $h_{\text{max\_iter}}$: gives maximum number of iterations to train $h$ per each batch.

9. $h_{\text{stop\_condition}}$: gives the stopping condition for training for current batch. If error difference gets less than that, training of $h$ stops for the current batch.

10. $\text{eval\_every}$: Number of batches after which 12 norm of $(v - Wh)$ is computed. Decreases performance if it is set too low.

11. $\text{normalize}$: denotes whether to normalize the result or not

12. $\text{random\_state}$: is the parameter for setting a randomState object or a seed to generate one.

The final results of topic modeling with NMF returns dictionary with up to 3 entries, “topics”, “topic-word-matrix”, and “topic-document-matrix”.

### 3.3.3 CTM

CTMs are a family of topic models, which contains CombinedTM and ZeroShotTM, each having different use cases. CTMs combine BERT embeddings which are contextual with the unsupervised capabilities of topic models to get topics out of documents. Preprocessing is the main key to get accurate results. To get a good representation out of a contextual model like BERT, we need to give not preprocessed text for BERT embeddings. The preprocessing class of CTM can take care of this. CTM uses SBERT which in turn allows to use any type of embedding model from sentence transformers. If the size of the bag of words has been restricted to a number of terms less than or equal to 2,000 elements, CTMs work better. The preprocessing pipeline of CTM has two outputs – preprocessed and non preprocessed text. The non-preprocessed texts are not disregarded, they are used as input for obtaining the contextualized document
representations. The `TopicModelDataPreparation` object takes care of creating bag of words and then obtaining the contextualized BERT representations of documents. The training dataset is created by this operation. Similar to the previous models NMF and LDA, CTM is also run for a number of topics ranging from 10 to 50 with step size of 10. Here, the `contextual_size` is set as 768, while the number of epochs as 10 for the training. The other parameters are listed in Table 5. The explanation of each of the parameter is provided below:

1. `num_topics`: is the number of topics. The default value is 10.

2. `model_type`: selects from either of the two models LDA or `prodLDA`. The default is `prodLDA`.

3. `activation`: selects the required activation function from `softplus`, `relu`, `sigmoid`, `swish`, `tanh`, `leakyrelu`, `rrelu`, `elu`, `selu`. The default is `softplus`.

4. `num_layers`: selects the number of layers, default value is 2.

5. `dropout`: is the dropout value to use; default is 0.2.

6. `learn_priors`: is a boolean value and make priors a learnable parameter.

7. `batch_size`: is the batch size for training.

8. `lr`: is the learning rate to use for training; default is 0.99.

9. `momentum`: is the momentum to use for training.

10. `solver`: is the optimizer `adam` or `sgd`; default is `adam`.

11. `num_epochs`: is the number of epochs for training; default is 100.
12. *num_samples*: is the number of times theta is needed to be sampled; default is 10.

13. *use_partitions*: If this value is set as True, the model is trained on the training set and evaluated on the test set.

14. *reduce_on_plateau*: reduces learning rate by 10x on plateau of 10 epochs.

15. *inference_type*: selects the type of the CTM model.

16. *bert_path*: is the path to store the document contextualized representations.

17. *bert_model*: the name of the contextualized model to be given (default: \texttt{bert-base-nli-mean-tokens}).

After model training, we can explore the topics and use visualization graphs to view the results using respective functions.

### 3.3.4 BERTopic

According to Grootendorst [14], the topic modeling technique BERTopic leverages BERT embeddings and c-TF-IDF, and uses clustering algorithm such as HDBSCAN to create dense clusters of similar documents. Using this, BERTopic can create easily interpretable topics while keeping important words in the topic descriptions. Since the working of BERTopic is explained in Section 2.1.2.4, we only offer the methodology followed and implementation details in this section. BERTopic implementation follows the OCTIS framework where the data is preprocessed using OCTIS’s preprocessor class. The preprocessed data is loaded using the dataloader function in OCTIS.

The training data is obtained by combining the tokens in each document. This data can be passed to the selected sentence transformer model and generate the
Table 5: Default parameters for CTM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_topics</td>
<td>100</td>
</tr>
<tr>
<td>model_type</td>
<td>prodLDA</td>
</tr>
<tr>
<td>activation</td>
<td>softplus</td>
</tr>
<tr>
<td>dropout</td>
<td>0.02</td>
</tr>
<tr>
<td>learn_priors</td>
<td>True</td>
</tr>
<tr>
<td>batch_size</td>
<td>64</td>
</tr>
<tr>
<td>lr</td>
<td>2e-3</td>
</tr>
<tr>
<td>momentum</td>
<td>0.99</td>
</tr>
<tr>
<td>solver</td>
<td>adam</td>
</tr>
<tr>
<td>num_epochs</td>
<td>100</td>
</tr>
<tr>
<td>reduce_on_plateau</td>
<td>False</td>
</tr>
<tr>
<td>prior_mean</td>
<td>0.0</td>
</tr>
<tr>
<td>prior_variance</td>
<td>None</td>
</tr>
<tr>
<td>num_layers</td>
<td>2</td>
</tr>
<tr>
<td>num_neurons</td>
<td>100</td>
</tr>
<tr>
<td>use_partitions</td>
<td>True</td>
</tr>
<tr>
<td>num_samples</td>
<td>10</td>
</tr>
<tr>
<td>inference_type</td>
<td>zeroshot</td>
</tr>
<tr>
<td>bert_path</td>
<td>-</td>
</tr>
<tr>
<td>bert_model</td>
<td>bert-base-nli-mean-tokens</td>
</tr>
</tbody>
</table>

embeddings. In the next step, training can be performed with given parameters. As explained in other models, the required metrics are evaluated for topics ranging from 10 to 50 with steps of 10. To obtain the final results, these results are averaged across three runs for each step. The overall workflow using BERTopic is illustrated in Figure 9.

The parameters which are set for the training are as follows:

1. \textit{nr\_topics}: the number of topics are set as ranging from 10 to 50 in steps of 10.
Figure 9: The overall workflow of BERTopic.

2. \textit{min\_topic\_size}: is the minimum size of the topic which is set as 15 for the training. The higher the value, the lower the number of clusters/topics.

3. \textit{diversity}: A value is set to use Maximum Marginal Relevance function (MMR) \cite{9}. MMR is a widely used ranking algorithm for diversity modeling tasks. MMR considers the similarity between keywords in a document along with the similarity of selected keywords. This algorithm helps to diversify the resulting topic representations. The values can range from 0 and 1 with 0 being not at all diverse and 1 being very diverse. Here it is set as \texttt{None}.

4. \textit{verbose}: changes the verbosity of the model.

In addition to this set of parameters, there are additional parameters for hyper parameter tuning. Each of them is explained below.

1. \textit{top\_n\_words}: defines the number of words to be extracted per topic. This should be preferably between 10 and 20.

2. \textit{n\_gram\_range}: sets the n grams in a topic representation. By default, it is 1.

3. \textit{min\_topic\_size}: is the minimum number of words needed in a topic.

4. \textit{low\_memory}: setting this to \texttt{True} allows UMAP to be run with low memory.

5. \textit{calculate\_probabilities}: allows to calculate the probability of each document to a topic.
Additionally hyperparameter tuning can be done for certain parameters of UMAP and HDBSCAN. For this thesis, parameter tuning was performed only for reducing the number of outliers. This can be done by setting the parameter `min_cluster_size` to a lower number in HDBSCAN. Since setting this to a lower value was not very effective in reducing the number of outliers in our data, this approach was not continued for the remaining data.

### 3.3.4.1 Embedding Models

Word embeddings are a type of word representation that allows words to be represented as real-valued vectors in a predefined vector space. In the vector space, similar words will be clustered together whereas different words will be far apart. To make NLP tasks easier, document embeddings can also be used. Document embeddings create vector representations which are of fixed length from the given documents. BERTopic creates embeddings for documents in vector space and identifies semantically similar sentences before clustering. This topic model leverages pre-trained embeddings which allows to create topic from the user’s data. BERTopic allows users to select any embedding model from libraries like Flair [46], Gensim [47], Huggingface transformers [128], Spacy [129] etc. The default embedding model used in BERTopic is by sentence-transformers. The embeddings created by sentence-transformers work really well in generating topics. Since these embeddings are used only to cluster semantically similar documents, any custom embedding model which suits user’s requirements can be used. Next, we offer an explanation for each model.

**Flair**: It is a framework [46] for state-of-the-art NLP techniques such as named entity recognition (NER), part-of-speech tagging (PoS), sense disambiguation and classification, etc. The simple interfaces Flair provides allows users to use and combine different word and document embeddings including Flair embeddings, BERT
embeddings, and ELMo embeddings. Any embedding model that is publicly available can be selected using Flair. It also allows to choose any huggingface transformer model that is publicly available. Flair library allows to create document embeddings by simply averaging word embeddings and this can be easily passed to BERTopic.

**Gensim**: is a very popular Python library that offers an efficient suite of NLP tools that focuses on topic modeling. This library provides implementation of word2vec for learning word vectors. Additionally, it also supports other embeddings such as Glove, Doc2vec, and FastText. Gensim supports a set of pre-trained embeddings and has libraries to load these vectors in a specific format. It also supports querying these embeddings. Using `gensim.downloader` module supported by BERTopic, users can download any word embedding model supported by Gensim.

**Spacy**: It is another open source library which can be used for advanced NLP techniques. Embeddings can be created using Spacy’s transformer or non-transformer models and can be passed to BERTopic.

**Universal Sentence Encoder**: It is another document embedding model which encodes text into high-dimensional vectors. This pre-trained embedding is available on TensorFlow Hub. This model is ideal for clustering and semantic similarity and it is optimized for sentences, phrases, or short paragraphs.

**Word + Document Embeddings**: BERTopic’s backend supports a word plus document embeddings format where a user can specify the word embedding and document embedding model of their choice. For example, FastText word embeddings can be combined with a sentence-transformer based embedding model. In BERTopic, the word embeddings are used only in the final step where Maximal Marginal Relevance algorithm is calculated to select the top 10 keywords in a topic. This algorithm helps to select diverse keywords thereby avoiding redundancy. To add this feature in the implementation, one would need to instantiate BERTopic with a diversity value
between 0 and 1.

**Custom Embeddings**: Custom embeddings can be implemented in BERTopic by passing the document embeddings created by the user through `fit_transform` method of BERTopic.

**Custom Backend**: If a user wants to create a backend embedding model which is custom based, `bertopic.backend.BaseEmbedder` class can be used to create the backend.

**TF-IDF**: The TF-IDF matrix can be implemented in BERTopic using the `fit_transform` method in BERTopic. The dimensionality reduction technique UMAP (Uniform Manifold Approximation and Projection for Dimension Reduction) typically uses cosine distance metric which is not suitable for a TFIDF matrix. BERTopic will use Hellinger distance metric once it recognises the sparse matrix created by TFIDF.

**Sentence-Transformers**: One of the most popular Python framework for cutting-edge sentence, text, and image embeddings is sentence-transformers. This framework provides embeddings for more than 100 languages. It is also based on PyTorch and transformers and provides a set of pre-trained embedding models which are fine tuned for specific tasks. The initial model is sentence BERT [39] which is based on BERT network that uses Siamese and triplet network structures. The BERT network model is modified to procure semantically meaningful sentence embeddings. Comparison of these embeddings is done with the help of cosine-similarity. The applications of this model are for large-scale semantic similarity comparison, clustering, and information retrieval via semantic search.

The authors suggest that the embeddings from SBERT have higher performance than the sentence embeddings from the original BERT model where the sentence embedding is obtained by pooling from BERT output layers or by taking the embedding of the first output token ([cls] token). The basic idea behind this model
Figure 10: SBERT architecture with classification objective function.

is to use a Siamese network architecture to generate fixed-size vectors for the input sentences. After this, similar sentences can be obtained by using either cosine similarity or Manhatten/Euclidean distance. This method is very effective for semantic similarity search and clustering.

The SBERT architecture is presented in Figure 10. In this model pre-trained BERT/ROBERTa model is used which is fine tuned on SNLI [130] and MNLI [131] datasets. To get the output embeddings a pooling operation is applied. There are three pooling operations- using the output of the CLS-token, using mean of all output vectors and using a max-over-time of the output vectors. However the default option is
Table 6: Pre-trained models for sentence transformer.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>Model</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Model</td>
<td>microsoft/mpnet-base</td>
<td>nreimers/MiniLM-L6-H384-uncased</td>
<td>distilroberta-base</td>
</tr>
<tr>
<td>Max seq length</td>
<td>384</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>Dimensions</td>
<td>768</td>
<td>384</td>
<td>768</td>
</tr>
<tr>
<td>Pooling</td>
<td>mean pooling</td>
<td>mean pooling</td>
<td>mean pooling</td>
</tr>
<tr>
<td>Size</td>
<td>420 MB</td>
<td>80 MB</td>
<td>290 MB</td>
</tr>
</tbody>
</table>

mean. Here, these models have tied weight. Using Siamese and triplet networks \[132\], these weights are updated to produce semantically meaningful sentence embeddings. The experiments is conducted using three objective functions-classification objective function, regression objective function, and triplet objective function. This is the basic architecture of SBERT, and there are various pre-trained models provided by the sentence transformers. Among these models, “all-mpnet-base-v2” model provides the best quality, while “all-MiniLM-L6-v2” is five times faster and still offers good quality. The models used in this thesis are the following:

1. all-mpnet-base-v2;
2. all-MiniLM-L6-v2; and
3. all-distilroberta-v1.

The description of each of these models is offered in Table 6.

**GloVe-SO:** GloVe, Global Vectors for word representation, is an unsupervised algorithm to generate word embeddings by capturing global and local statistics of a corpus \[133\]. The word embedding is obtained by aggregating word co-occurrence matrix from a corpus. The co-occurrence matrix gives an idea of how often a pair of words occur together. The results of these aggregations show interesting linear substructures of the word vector space. For this thesis, custom GloVe vectors trained
on Stack Overflow data dump \cite{134} are used along with sentence transformer models. GloVe embeddings trained on regular English text cannot capture the actual context of words from text in developer chats and other software development communication. There is a difference in terms of vocabulary and semantics compared with regular English text. The Stack Overflow data dump is collected as of June, 2020. The training was performed after standard tokenization and preprocessing and trimming of rarely occurring words. This embedding model contains 123,995 words and each word is represented by a 200-dimensional word vector.

In the methodology for BERTopic evaluation, a word+document embedding model is used where the word embeddings are based on GloVe vectors trained on Stack Overflow (GloVe\_SO) loaded using Flair word embeddings library and document embeddings are procured using the three sentence transformers: all-mpnet-base-v2, all-MiniLM-L6-v2, and all-distilroberta-v1.

### 3.4 Evaluation Metrics

The performance of topic models in this thesis is evaluated by two widely used metrics — topic coherence and topic diversity. The topic coherence metric measures the association between a topic and a set of top n words that form a topic. Röder et al. \cite{135} made a study on topic coherence measures and created hybrid coherence metrics by different combinations of existing coherence metrics. According to the framework proposed by authors, there are four steps in the coherence calculation pipeline which are the following: 1) segmentation of subsets of word, 2) word probabilities estimation, 3) confirmation measures computation, and 4) aggregation of confirmation measures to form final coherence score. The authors have suggested four coherence metrics that match human judgement which are UCI, UMass, NPMI, and $C_v$. The coherence metrics used for evaluation of the topic models in this work are NPMI and $C_v$. 
NPMI is a metric based on Normalized Point-wise Mutual Information (PMI) \[136\]. Considering the words’ empirical frequency in the original corpus, it measures how related the top ten words of a topic are to each other \[7\]. This metric has a reasonable performance and emulates human judgement \[138\]. The coherence score ranges from -1 to 1 where the perfect association is indicated by 1. In the equation \[7\] \(w_i\) and \(w_j\) refer to the pair of words, and \(N\) is the top number of words of a topic.

The metric \(C_v\) is the default metric provided in the Gensim topic modeling pipeline. This metric uses the co-occurrence of the words to create content vector and, after that, calculates the score using NPMI and the cosine similarity. The next important metric used is the topic diversity \[6\] which is calculated as the percentage of unique words for all topics \[139\]. The diversity score can range from 0 to 1, where 1 indicates more diversified topics and 0 as redundant topics. The parameters \(t_{\text{unique}}\) stands for unique words in a document, and \(t_n\) stands for total number of words in a document.

\[
\text{TopicDiversity} = \frac{t_{\text{unique}}}{t_n} \tag{6}
\]

\[
\text{NPMI} \left( w_i \right) = \frac{N-1}{\sum_j - \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}} \tag{7}
\]

### 3.5 Manual Labeling of Topics

The topics generated by the topic modelling techniques are manually labeled for human interpretation/validation and further analysis. Two coders have performed the manual labeling of the topics in the following steps.

1. **Reviewing the corpus together to get a general understanding of the discussions.**

2. **Independent reviewing of the most frequent words of topics for three months of**
Table 7: Examples of the topic \textit{misc}.

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>random, of, in, so, yield, with, on, more, not, time</td>
<td>misc</td>
</tr>
<tr>
<td>of, python, in, with, on, not, or, so, are, learning</td>
<td>misc</td>
</tr>
<tr>
<td>channel, for, otc, been, time, user, answered, know, your, program</td>
<td>misc</td>
</tr>
<tr>
<td>site, tos, or, this, for, rules, they, amazon, in, about</td>
<td>misc</td>
</tr>
</tbody>
</table>

3. Calculating inter-rater agreement score using Kappa score measurement \cite{140}. We obtained the Kappa’s score of 0.8659, indicating a strong agreement.

4. Disagreements on the topics were discussed and resolved.

5. Labeling is continued for the three months of data with the selected topic model and three different embedding models.

6. Once the topic and embedding models are decided the final labeling for the remaining months is performed.

7. There are two category of the labels: respective topics, and \textit{misc}. The label \textit{misc} is assigned for topics that have no meaning. Few examples are shown in the Table 7.

3.5.1 Manual Rating of Topic Models

For evaluating the quality of topic models, a manual rating of the models in a three point scale is performed. The rating is based on the percentage of meaningful topics over total number of topics. We can obtain the total number of meaningful topics by counting the labeled topics that are not \textit{misc} after the manual annotation. If the
percentage is above 80, the rating is given as excellent. The rating good is given if the percentage of labeled topics is between 50 and 80. The third rating poor is given if the percentage falls below 50.
Chapter 4

Results

In this chapter, we present the results of experiments conducted and provide answers to all research questions stated in Section 1.3.

4.1 Performance of Topic Modeling Techniques

In this section, we answer the first research question “How effective are topic modeling techniques in extracting summaries from developer conversations?”. The training dataset consists of conversations as shown in Table 8. The experiments performed can be divided into two. In the initial experiment, comparison of topic modeling techniques such as LDA, NMF, CTM, and BERTopic were performed on three months of data. Selection of months were random and the months selected are December 2019, March 2020, and June 2020. The embedding used in BERTopic is the default embedding, i.e., all-MiniLM-L6-v2.

The final results are obtained by taking the average of three months and reported in Table 9. The performance evaluation is conducted with the metrics NPMI, $C_v$, and Topic Diversity. The NPMI score for all the topic models is negative for this dataset. The NPMI value can range between $-1$ to $1$, where $1$ is the perfect association and $-1$ is the zero association between the word pair. The maximum values for each
Table 8: Dataset size.

<table>
<thead>
<tr>
<th>Month</th>
<th>#of Utterances</th>
<th>#of Conversations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov2019</td>
<td>33,911</td>
<td>1,086</td>
</tr>
<tr>
<td>Dec2019</td>
<td>30,939</td>
<td>1,125</td>
</tr>
<tr>
<td>Jan2020</td>
<td>46,312</td>
<td>1,278</td>
</tr>
<tr>
<td>Feb2020</td>
<td>38,291</td>
<td>1,203</td>
</tr>
<tr>
<td>Mar2020</td>
<td>56,762</td>
<td>1,322</td>
</tr>
<tr>
<td>Apr2020</td>
<td>76,488</td>
<td>1,500</td>
</tr>
<tr>
<td>May2020</td>
<td>97,710</td>
<td>1,693</td>
</tr>
<tr>
<td>Jun2020</td>
<td>120,647</td>
<td>1,817</td>
</tr>
<tr>
<td>Jul2020</td>
<td>165,006</td>
<td>1,981</td>
</tr>
<tr>
<td>Aug2020</td>
<td>209,358</td>
<td>2,224</td>
</tr>
<tr>
<td>Sep2020</td>
<td>173,066</td>
<td>2,114</td>
</tr>
<tr>
<td>Oct2020</td>
<td>205,860</td>
<td>2,321</td>
</tr>
</tbody>
</table>

of the metrics are shown as bold values in Table 9. As per the evaluation metrics, BERTopic achieves the highest NPMI score of $-0.02$, while CTM offers the highest values for $C_v$ and Topic Diversity which is 0.39 and 0.59 respectively.

Grootendorst [14] suggests that these metrics can only give an indication of highest performing model. These evaluation metrics are proposed as the substitutes of human evaluation. Since human evaluation is a very subjective process, the judgement of topic coherence and diversity vary from one person to another. Hoyle et al. [141] proposed in their studies that neural topic models can give good npmi score without giving an adequate explanation of corpus to the user. According to the authors npmi score works well with classical topic models than neural topic models. For this reason, we conducted a manual evaluation of topic models by adding a human rating to the topic model. The rating is done in a three point scale - poor, good, and excellent based on the quality of the topics.

Out of the labeled topics, we calculated the percentage of meaningful topics over
Table 9: Performance evaluation of topic models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dec2019</th>
<th>March2020</th>
<th>June2020</th>
<th>Average</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>npmi</td>
<td>C_v</td>
<td>div.</td>
<td>npmi</td>
<td>c_v</td>
</tr>
<tr>
<td>CTM</td>
<td>-0.26</td>
<td>0.43</td>
<td>0.64</td>
<td>-0.26</td>
<td>0.39</td>
</tr>
<tr>
<td>LDA</td>
<td>-0.03</td>
<td>0.32</td>
<td>0.09</td>
<td>-0.03</td>
<td>0.31</td>
</tr>
<tr>
<td>NMF</td>
<td>-0.03</td>
<td>0.34</td>
<td>0.31</td>
<td>-0.03</td>
<td>0.33</td>
</tr>
<tr>
<td>BERTopic</td>
<td>-0.03</td>
<td>0.34</td>
<td>0.35</td>
<td>-0.03</td>
<td>0.33</td>
</tr>
</tbody>
</table>

the total number of topics. If the percentage falls below 50, the rating is given as *poor*, and if it is above 80 percentage, the rating is *excellent*. The rating *good* is given if the percentage of labeled topics is between 50 and 80. According to our manual evaluation as reported in column Rating in Table 9, BERTopic model offers an overall good rating compared to other models. The human rating of the topic model is performed for each month - December 2019, March 2020, and June 2020, respectively, and the final average rating for each topic model is decided by the majority vote.

The evaluation metrics for the month of June are shown in Figure 11, Figure 12 and Figure 13. The graph consists of the metrics evaluated for a number of topics ranging from 10 to 50 in steps of 10. For the month of June, the number of utterances is 120,647 and the number of conversations is 1,817. The NPMI score for BERTopic is close to zero compared to other models and $C_v$ and Topic Diversity is higher for the model CTM. According to the manual rating CTM has a poor performance as it is able to have only 16 meaningful topics out of 50, while BERTopic offers 13 meaningful topics out of 24.
Figure 11: NPMI score for topic models for June 2020.

Figure 12: $C_v$ score for topic models for June 2020.
Answer to RQ1:

We evaluated the performance of four topic models in extracting meaningful summaries from developer monthly chat conversations. Two models, BERTopic and CTM, have offered comparable performance. With respect to the NPMI score, BERTopic performed well, while CTM performed better with respect to the $C_v$ and Diversity metrics. However, our manual evaluation of topics quality demonstrates that the topics generated by BERTopic are of higher quality compared to the other models (LDA, NMF, and CTM).

4.2 Impact of Embedding Models on the Performance of BERTopic

In this section, we provide the answer to the second research question “What is the impact of different embedding models on the performance of the best topic model?”.
Table 10: Performance evaluation of BERTopic with embedding models.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe.SO+MiniLM</td>
<td>-0.01</td>
<td>0.33</td>
<td>0.66</td>
<td>-0.03</td>
<td>0.33</td>
<td>0.39</td>
<td>-0.03</td>
<td>0.33</td>
<td>0.33</td>
<td>-0.02</td>
<td>0.33</td>
<td>0.46</td>
<td>good</td>
</tr>
<tr>
<td>GloVe.SO+distilroberta</td>
<td>-0.01</td>
<td>0.34</td>
<td>0.44</td>
<td>-0.03</td>
<td>0.34</td>
<td>0.45</td>
<td>-0.05</td>
<td>0.40</td>
<td>0.55</td>
<td>-0.03</td>
<td>0.36</td>
<td>0.48</td>
<td>good</td>
</tr>
<tr>
<td>GloVe.SO+mpnet</td>
<td>-0.02</td>
<td>0.34</td>
<td>0.36</td>
<td>-0.03</td>
<td>0.33</td>
<td>0.34</td>
<td>-0.02</td>
<td>0.34</td>
<td>0.30</td>
<td>-0.02</td>
<td>0.33</td>
<td>0.33</td>
<td>good</td>
</tr>
</tbody>
</table>

From the first experiment, it is concluded that BERTopic performs best compared to other topic models. In the second part of the experiment, BERTopic model is evaluated with various embedding models. For this experiment, different embedding models are used with BERTopic. BERTopic makes this possible with the help of different libraries like Flair, Gensim, SentenceTransformer, etc. Even with the options to use a variety of embedding models, Grootendorst [14] suggests using sentence-transformer embedding model based on the results of their experiments. BERTopic model provides an option to combine a word embedding model of user’s preference with a document embedding model.

In this work, various embedding models were implemented and the use of combined word & document embeddings offers good quality topics compared to individual embedding models. This selection is performed on the basis of manual assessment of the topics generated. The word embedding model is based on GloVe trained on Stack Overflow data dump (GloVe.SO) [134] and is implemented with the help of Flair library. The sentence-transformer models selected are all-mpnet-base-v2, all-distilroberta-v1, and all-MiniLM-L6-v2.

The results of this experiment are reported in Table 10. This experiment is performed on three months of data, similar to our RQ1. The final results are averaged across three months. We also report a label of our manual assessment of the topic quality where the rating is in a three-point scale (poor, good, or excellent). From the results in Table 10, we can observe that the three models are achieving
comparable performances. In terms of NPMI, GloVe_SO+all-mpnet-base-v2 and GloVe_SO+all-MiniLM-L6-v2 offer the same score of −0.02, and in terms of $C_v$ and Topic Diversity, GloVe_SO+all-distilroberta-v1 attains a better performance with values 0.36 and 0.48 Though these metrics give the user an indication of best performing model, the manual rating of topics give us the actual picture of the best performing model. The manual assessment shows that all three models offer a good rating for the extracted topics. Checking for each month individually as shown in Table 11, the models GloVe_SO+all-mpnet-base-v2 and GloVe_SO+all-MiniLM-L6-v2 perform slightly better than GloVe_SO+all-distilroberta-v1.

Table 11: Manual rating for three embedding models for each month.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dec 2019</th>
<th>March 2019</th>
<th>June 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe_SO+MiniLM</td>
<td>excellent</td>
<td>good</td>
<td>poor</td>
</tr>
<tr>
<td>GloVe_SO+distilroberta</td>
<td>good</td>
<td>good</td>
<td>poor</td>
</tr>
<tr>
<td>GloVe_SO+mpnet</td>
<td>excellent</td>
<td>good</td>
<td>poor</td>
</tr>
</tbody>
</table>

Figure 15, Figure 16, and Figure 14 illustrate the performance of the three embedding models for the month of December 2019. By analyzing these plots, we can see that each model performs differently for each of the three metrics. For example, the model GloVe_SO+all-MiniLM-L6-v2 has the highest topic diversity score but has the lowest $C_v$ score compared to other two models. Similarly, the model GloVe_SO+all-distilroberta-v1 has the highest NPMI and $C_v$ score and an average Topic Diversity score compared to other two models. Additional images for the performance evaluation are reported in Section A.2.
Figure 14: NPMI score for BERTopic with different embedding models for December 2019.

Figure 15: $C_v$ score for BERTopic with different embedding models for December 2019.
Figure 16: Diversity score for BERTopic with different embedding models for December 2019.

**Answer to RQ2:**

The results demonstrate that the combination of word embedding and document embedding models is powerful in generating good quality topics. Out of the three sentence-transformer models, the two models such as all-mpnet-base-v2 and all-MiniLM-L6-v2 perform consistently good with the word embedding model GloVe_S0. Among these two, the model with the sentence-transformer model all-MiniLM-L6-v2 offers a slightly better diversity than the other one. However, the performance of these models varies for each month. We argue that human evaluation of the topics quality is critical and must complement traditional metrics when selecting the best embedding model.
Table 12: Number of meaningful topics per month.

<table>
<thead>
<tr>
<th>Months</th>
<th>No. of labelable topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 2019</td>
<td>18</td>
</tr>
<tr>
<td>Dec 2019</td>
<td>26</td>
</tr>
<tr>
<td>Jan 2020</td>
<td>22</td>
</tr>
<tr>
<td>Feb 2020</td>
<td>25</td>
</tr>
<tr>
<td>March 2020</td>
<td>18</td>
</tr>
<tr>
<td>April 2020</td>
<td>12</td>
</tr>
<tr>
<td>May 2020</td>
<td>15</td>
</tr>
<tr>
<td>June 2020</td>
<td>15</td>
</tr>
<tr>
<td>July 2020</td>
<td>7</td>
</tr>
<tr>
<td>Aug 2020</td>
<td>11</td>
</tr>
<tr>
<td>Sep 2020</td>
<td>6</td>
</tr>
<tr>
<td>Oct 2020</td>
<td>11</td>
</tr>
</tbody>
</table>

4.3 Extracted Topics for python#python-general

Here, we answer our third research question “What are the topics discussed in the Discord channel python#python-general?”. The performance across the evaluation metric NPMI and manual rating suggests the combined word document embedding model with all-MiniLM-L6-v2 as the best performing embedding model. As a result, this combined word document embedding model is selected as the primary embedding model for BERTopic. The second embedding model is also implemented and the additional topics identified is also included in the final topics list. The statistics of the number of meaningful topics are reported in Table 12. There is also a category of topics labeled as misc by the annotators to identify topics which do not convey any particular meaning. The total number of misc labels is 127 for the primary embedding model.

The topics discussed in each month are presented in Table 13. In general, the
topics discussed are related to various installations, Python IDE, different Python libraries, API calls, concurrent programming, Python learning resources, gaming in Python, Android programming, etc. The number of topics discussed in each month varies and some of the topics are repeated.

**Answer to RQ3:**


### 4.4 Evolution of Topics

In this section, we address our last research question which is related to the evolution of topics discussed in the `python#python-general` channel over the one-year period. The total number of conversations and utterances over a one year period is shown in Figure 17 and Figure 18 respectively. The graph shows that the number of conversations and utterances increases steadily after March 2020. To understand the evolution of topics, a heatmap is constructed as shown in Figure 19. In the heatmap, we can see the number of posts each topic appears for every month starting from November 2019 to October 2020. For the construction of a heatmap, some of the topics in the individual months are grouped under a more general term as shown in Table 14. After this step, we finalized a list of 23 topics as show in Figure 19.

There are certain topics which are common across a minimum of 11 months. Those labels are Installation Related Topics, Python IDEs, Learning Resources, Python Functions, List Operations, and Web Framework. There are some topics which are uncommon and these topics are discussed only for few months. They
<table>
<thead>
<tr>
<th>Month</th>
<th>Topics</th>
</tr>
</thead>
</table>
Table 14: General topics.

<table>
<thead>
<tr>
<th>General Name</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>Class Functions, Print Functions, Input Output Functions, Log Functions,</td>
</tr>
<tr>
<td></td>
<td>Map Reduce Functions, Package Publishing Functions, Loop Functions,</td>
</tr>
<tr>
<td></td>
<td>Iterable Objects, Node Data Structure Functions, Python Generator Functions.</td>
</tr>
<tr>
<td>Learning Resources</td>
<td>ML Learning Resources, Self Learning, Python Training, Learning Materials,</td>
</tr>
<tr>
<td></td>
<td>New Project Ideas, Course Recommendation, Beginner Resources.</td>
</tr>
<tr>
<td>Errors and Exceptions</td>
<td>Bugs, Exceptions.</td>
</tr>
<tr>
<td>Installation</td>
<td>Install Python in Windows, Install Postman, Darwin Package manager, Dist-</td>
</tr>
<tr>
<td></td>
<td>Packages, Windows Reinstallation.</td>
</tr>
<tr>
<td>Python IDE</td>
<td>Pycharm, Vim Text Editor, Visual Studio Installation, Git Code and Visual-</td>
</tr>
<tr>
<td></td>
<td>Studio.</td>
</tr>
<tr>
<td>Gaming</td>
<td>Pygame, Game Development, Game Project, Game Platform, 3D Python-</td>
</tr>
<tr>
<td></td>
<td>Scripting in Blender.</td>
</tr>
<tr>
<td>Data Structures</td>
<td>Dictionary, Set, List.</td>
</tr>
<tr>
<td>Concurrent Programming</td>
<td>Asyncio, Erlang.</td>
</tr>
<tr>
<td>API Calls</td>
<td>API, XML in Web.</td>
</tr>
<tr>
<td>Android App Framework</td>
<td>Kivy, Pydroid.</td>
</tr>
</tbody>
</table>

are GUI Apps, Libraries, Concurrent Programming, Web Scraping, Errors and Exceptions, File Processing, Android App Framework/Programming, Discord Channel Related Topics, DB Connection, API Calls, GitHub Commands, File Storage, Regular Expression Handling, Pandas Operations, and Code Help. The topic Code Help is the only topic which appears from July onwards till October. Certain topics like Pandas Operations, Regular Expression Handling, File Storage, DB Connection, and Android App Framework/Programming are discussed only in the initial months of the study period.
Answer to RQ4:
When we analyze the topics over a period of one year, we found that topics such as Installation, Python IDE, Python Functions, Web Framework and Learning Resources continuously appear in the discussions. The topics such as Android App Programming/Framework, Concurrent Programming, Web Scraping, DB Connection, File Storage, Regular Expression Handling, and Pandas Operations are discussed only in the earlier months of the studied period. The Code Help topic emerges in July 2020 and continues to be discussed onwards. Topics like Libraries and Gaming are discussed in the earlier months and not discussed between May 2020 and September 2020; yet they reappear in the discussions in October 2020.
Figure 19: Heatmap of topics between November 2019 and October 2020.

### Answer to RQ4 (contd):

All other topics such as **Data Structures**, **GUI Apps**, **Errors and Exceptions**, **File Processing**, **API Calls**, **Discord Channel Related**, and **GitHub Commands** are discussed sparsely throughout the year. We also observed that **Learning Resources** and **List Operations** are the most discussed topics compared to others.
Chapter 5

Discussion

This chapter presents the overall findings in Section 5.1, discusses the implications of this work in Section 5.2, and addresses potential threats to validity in Section 5.3.

5.1 Findings

In this thesis, our goal is to summarize developer chat conversations in the Discord channel: Python#General collected over a period of one year starting from November 2019 to October 2020. To accomplish this goal, we address and answer four research questions (Section 1) by applying the NLP and topic modeling techniques (introduced in Section 3). In this section, we discuss key results and findings.

The objective of this thesis is to summarize developer chat conversations that happen in Discord. Summarization techniques such as extractive and abstractive techniques are not very effective in summarizing chat data due to their unstructured nature and the presence of colloquial terms. Due to these reasons, topic modeling techniques are used to identify the most common topics in these discussions which in turn help in summarizing the conversations effectively.

For our study, we have extracted data from four Discord public channels and performed a chat disentanglement technique to identify the utterances which are part
of a conversation. The data used for this work is from the Python#General channel which has a higher volume of conversations than the other three channels. As an initial step, the data is cleaned and the different utterances were grouped together into conversations based on the conversation id. As illustrated in Figure 17, we can observe that the number of conversations is gradually increasing from March 2020 with October 2020 having the highest number of conversations (2,321).

The experiment as explained in Section 4.1 was the step in identifying the best topic model for extracting topics from our chat data. We compared four topic models, two classic models and two neural topic models which are LDA, NMF, CTM, and BERTopic. Based on the results of the first experiment, the model BERTopic performed well compared to the other models. The models were compared against the evaluation metrics such as topic coherence ($C_v$ and NPMI) and topic diversity. The most interesting finding here is that the selection of topic models cannot be made based on the metrics alone. Though the model CTM performed better than BERTopic in terms of $C_v$ and Topic Diversity, BERTopic performed better in terms of topics quality. The topics quality is measured based on human assessment of the quality of the generated topics.

As explained in Section 4.2, the next step was to identify the best performing embedding model for BERTopic. The BERTopic model can be experimented with different types of embedding models [14]. After experimenting with multiple embedding models and assessing the quality of topics, the combined word document embedding model is finalized and the results are reported in Section 4.2. The findings suggest that word document embedding model with a word embedding model trained on Stack Overflow data and a document embedding model with sentence-transformer offered better performance in generating quality topics.

The Section 4.3 reported the different types of topics discussed in the
python#python-general channel. The most number of labelable topics are generated on December 2019 with 26 topics. The topics for each month are listed in Table 13. The topics discussed are typically related to Installations, Python Functions, Data Structures, Web Framework, Python IDE, Libraries, Android App Programming, Learning Resources and Recommendations, Pandas Operations, Gaming, and General Code Help.

In Section 4.4, we study the evolution of discussion topics over the one year period. The most important finding here is that certain topics are discussed throughout an year. They are Installation, Python IDE, Python Functions, Web Framework, and Learning Resources. But there are certain topics which are discussed only in the earlier months of the studied period. Some of them are Android App Programming/Framework, Concurrent Programming, Web Scraping, DB Connection, File Storage, Regular Expression Handling and Pandas Operations. This can be due to the introduction of new help channels in the Python community. There are separate channels in the Python community which are #async-and-concurrency, #databases, #game-development, #Discord channel related, #web-development, etc. Interestingly, during the last four months, the topic Code Help is discussed more.

5.2 Implications

Our work offers numerous implications which can help software developers, researchers, and chat moderators of the Python user community.

5.2.1 Implications for Software Developers

Software developers can benefit from this work in many ways. They can use the insights from this work in searching for a particular discussion topic from the chat
discussions. If software developers are new to some channels or have missed some chat messages, they can obtain the gist of the discussions by running a chat summarizer that can be built on a topic modeling technique (e.g., BERTopic). Such chat summarizer can extract the different topics discussed in the channel. BERTopic provides an option to display representative documents for each topic. If the developer finds a topic of their interest, they can revisit the relevant conversations and find what they are looking for easily. Our recommendation for software developers is to consider integrating topic modeling techniques for summarizing chat conversations. Chat summarizers can increase the effectiveness and efficiency of software developers by reducing the time required for searching conversations on a particular topic. Chat summarizers can also reduce the information overload problem [142]. We can extend this work by developing approaches and tools for extracting personalized chat summaries for each developer based on a user profile (e.g., interest in specific discussion topics, libraries, APIs, etc.).

5.2.2 Implications for Researchers

Our work can be seen as preliminary research towards summarizing software developer chat conversations. Researchers can continue on developing novel and more advanced techniques for summarizing developer conversations. A possible extension of this work is to generate a personalized summary based on a user profile. Additionally, our work can be extended by using different chat disentanglement techniques and also experimenting with other state-of-the-art topic modeling techniques such as Top2Vec. Based on our work, we would like to advocate for the more extensive use of human assessment of the quality of topics extracted by topic modeling techniques. Our research shows that these models are not stable in terms of metrics since conversations and thus, corpus, vary from month to month. Our research focuses on summarizing conversations on a monthly basis. As the number of conversations increase very month,
future research work can also focus on weekly conversations. Studying the right granularity for chat summaries in terms of daily, weekly, or monthly conversations can be beneficial for the research community.

5.2.3 Implications for Python Community Moderators

Python community moderators can benefit from this work by using this technique to identify the topics where more help can be provided. If the number of posts for a particular topic is very high, they can consider creating a dedicated channel for that topic. For example, we noticed that the topic Learning Resources has been in demand throughout the year. Users ask for resources for self study, recommendations for courses, and ideas for beginner’s projects. Identifying these useful resources, community moderators can provide a list with the links to these learning resources and share it on the community portal or as pinned message on the right channel. By identifying the most discussed topics in each month, maintainers and moderators can produce additional support documentation for the greater Python community users by providing specific tutorials (e.g., installing Python packages), design documents, cheat sheets, guides, FAQ documents, online learning courses, etc.

5.3 Threats to Validity

Our research is subject to several threats to the validity; we discuss them next.

Disentanglement technique. First, we used chat disentanglement technique by Preetha et al. to disentangle the entwined conversations from the Discord channel. Elsner and Charniak’s disentanglement algorithm is modified to take into account the features specific to Slack. Even though the Discord data is very similar to Slack, there is a minor possibility of error in the disentanglement process. The authors evaluated the accuracy of the disentanglement process by performing a
manual disentanglement of 500 Discord messages and calculated the micro-averaged F-score. However, to ensure the accuracy further manual post-processing of the disentangled conversations can be performed. A comparative study with other state-of-the-art chat disentanglement techniques can also be explored to identify the best performing disentanglement technique. We have shared the replication package including our code implementing the disentanglement technique, and our dataset [7].

**Dataset.** Second, the data we used only cover general technical Q&As for Python programming language. The topics extracted provide a general idea of the discussions. Thus, if researchers are interested in mining information on specific topics, the dataset needs to be extended by collecting data for specific channels of interest. Our DISCO dataset offers data for four technical help channels: python#python-general, gophers#golang, racket$general, and clojurians#clojure. As the primary objective of this work is to summarize the developer chat conversations, the scope of our study is limited to the python#python-general channel. However, as a future work we can extend this approach to other channels.

**Human bias.** Third, there is a possibility of human error during annotation process for selecting the topic labels. The annotators can make a mistake by giving different names for the same topic. To avoid this, we reviewed the accuracy of the topics manually and corrected any errors in labeling by discussion and mutual agreement. We have reported the percent agreement during the manual annotation process in Section 3.5.

**Model configuration.** Fourth, is the quality of the BERTopic model in generating topics. BERTopic takes into account that a document contains only a single topic [14]. It is possible that a document may contain multiple topics. One possibility to avoid this problem is by dividing the data into smaller segments which is not ideal. The usage of HDBSCAN in BERTopic can resolve this issue to some extent.
Another limitation which affects the quality of topics generated is the maximum sequence length for the sentence-transformers. A common value for this is 512 which is around 300 to 400 English language words. Longer sequences are truncated in the sentence-transformer model. For this purpose, we have calculated the average number of tokens for 12 months which is 440 and is well under 512 tokens.
Chapter 6

Conclusion

The thesis concludes the work with a summary of the contributions in Section 6.1 and future research directions that are outlined in Section 6.2.

6.1 Summary of Contributions

This study investigates topic modeling techniques for summarizing developer chat conversations by comparing a wide variety of models on the task at hand. As a result, several contributions are made to the research community.

Firstly, we curated a dataset called DISCO consisting of the one-year public Discord chat COnversations of four software development communities. We have collected the chat data of the channels containing general programming Q&A discussions from the four Discord servers. After this we applied a disentanglement technique \cite{2} to extract conversations from the chat transcripts. A manual validation of conversations on a random sample (500 conversations) is also performed. This dataset consists of 28,712 conversations, 1,508,093 messages posted by 323,562 users. DISCO dataset with disentangled conversations along with the modified Elsner and Charniak’s algorithm code, as well as the JSON to XML conversion script are shared with the research community and made publicly available online \cite{7}.
Secondly, the results of applying topic modeling techniques to the chats, which are highly unstructured with a large number of colloquial terms, proved to be effective in summarizing the conversations to an extent. This can be confirmed by checking the quality of topics extracted from these chats. Also we tried to identify which of the topic models were successful in generating high quality topics. The experiments were conducted using four topic models such as LDA, NMF, CTM, and BERTopic. The results suggested that the neural topic model, BERTopic performed comparatively better than other models.

Thirdly, the most important contribution is that we cannot rely blindly on evaluation metrics alone to identify a best performing model. Human intervention is highly recommended to assess the quality of the extracted topics. After the topic labeling by two annotators, the performance of each of the topic model was assessed by a manual rating using a three-point scale.

Fourthly, we identified that the choice of embedding model can also affect the quality of topics. The results of our experiments suggest that a combined word+document embedding model with sentence-transformer performed better than other embedding models. The word embedding model trained on Stack Overflow data dump proved to be the best.

Fifthly, we performed an analysis of the evolution of topics in the Python channel and identified the newly emerged topics, the most common topics, the popular topics and the topics which disappeared from the discussions. As a result, the most popular topics are Learning Resources & List Operations, and the emerging topic is Code Help. There are some topics which were discussed in the initial months and disappeared later such as Concurrent Programming, Android App Programming, Web Scraping, etc. This information can be helpful for researchers, software developers, and community moderators.
6.2 Future Work

In this study, we identified a number of extensions that can be added as a future work. We recommend extending this work to other channels in Discord to get an in-depth idea of the discussed topics. For example, there are specific channels in the Python’s Discord server for concurrent programming, gaming, web development, etc. Yet we found many of these topics being still discussed on the General channel. This work can be extended to other online collaboration and communication platforms like Gitter that follow the same Q&A format like the python#python-general channel.

BERTopic is successful in extracting meaningful topics regardless of the programming language and custom word embedding model. However, to get diverse topics a custom word embedding model can be added to the model. Our primary goal is to summarize the developer chat conversations which is achieved to an extent. A more useful approach to summarization can be personalized summarization based on user profile [90]. One possible extension is to add other neural topic models for comparison like Top2Vec and also to experiment more with other embedding models. We can also consider other clustering algorithms such as $K$-means. In this work, we have explored the evolution of topics by checking the topics individually for each month. BERTopic offers an option to perform dynamic topic modeling that allow sequentially-organized documents to be modeled. Another possible extension of this work is by exploring the dynamic topic modeling feature of BERTopic to study the evolution of topics.
List of References


Appendix A

Appendix

A.1 Representative Documents for Topics

BERTopic provides the users with an option to view three most representative documents to a corresponding topic. Two examples of a topic and the corresponding representative documents are listed below.

- **Topic : Web Scraping**
  
  **Document** : 'what is selenium used for’, 'does anyone in here chromium with selenium’, 'is web scrapping hard hmm can you recommend me what web scraper i should use'

- **Topic : Class Functions**
  
  **Document** : 'pep says class, iirc ah i am trying convert keras model tf model im getting error yolo head missing hello so trying understand init what does for example self guitarist guitarist actually do guitarist alone is argument you assign object property named guitarist so if going something like if i would do def player guitarist then i would have in init you would define a class player then a def init self guitarist and when you create object you could like legend player jimmy hendrix legend would a player object with a guitarist property
that you can access with legend guitarist object other people can explain better
than me i think help channels are better suited for this kind of questions like
a def is object a function in class quick google about objects python https
www.programiz.com/pythonprogramming class object are more things they can
do stuff with function methods or just contains data or both user there is no
requirement at all participate in code jam except that you give a try pass our
qualifier the qualifier itself is a small fun challenge after you passed you will
participating in code jam with a team even if you have no prior knowledge
of working as a team working with kivy okay i will read this thanks it will a
valuable learning experience do give a try what is time investment required for
this codejam ok how often do u arrange this competition user twice a year a
winter a summer code jam for more information about code jam in general you
can read them here https.pythondiscord.com.pages.codejams',

'does keeping or not keeping this in class have any effect why would you do it
saw this in a codebase that i am working on would override init inherited y so
instead of executing y init when you instantiate an x object nothing happens ah
thanks lemon user dont do that this channel is for discussing python if any knew
pls either ping me or msg here maybe python cookbook and related websites
should easy find if you want some python challenges then mr hemlock sent
some bit of time before discordapp.com channels its in pinned messages of this
channel i just thought would very nice if i could my mouse while a virtual mouse
is used by python program perfect thanks person who made this',

'whats difference fundamentally between instance class variables when i have
following i get that class variables are shared between all instances or whatever
but following code proves otherwise as you can see when i initialized my classes
class_var started as one for both but changing in in did not propogate over in
self class_var will access an instance variable base class_var would access a class variable ohhh so class variables are more static variables in other languages and i effectively just overrode class variable with an instance variable in this case pretty much well not overrode just made a new instance one with same name as class one so how would i access class variable in at that point base class_var so if i passed in in or something didnt know its type but knew had a class variable id have do type in class_var ye wild i dont know what do with this information but is here’

A.2 Evaluation of Topic Models

In this section, the evaluation metrics for various topic models are illustrated for the months of December 2019 and March 2020. The evaluation metrics for BERTopic for March 2020 are also given. In this section one sample of topic labeling using GloVe-SO and GloVe is given in Figure 29. Here the pretrained sentence transformer model used is all-MiniLM-L6-v2.

![Figure 20: NPMI score for topic models for December 2019.](image-url)
Figure 21: $C_v$ score for topic models for December 2019.

Figure 22: Diversity score for topic models for December 2019.
Figure 23: NPMI score for topic models for March 2020.

Figure 24: Diversity score for topic models for March 2020.
Figure 25: $C_v$ score for topic models for March 2020.

Figure 26: NPMI score for BERTopic with different embedding models for March 2020.
Figure 27: Diversity score for BERTopic with different embedding models for March 2020.

Figure 28: $C_v$ score for BERTopic with different embedding models for March 2020.
<table>
<thead>
<tr>
<th>Model</th>
<th>Topic Keywords</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe-SO</td>
<td>in</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>what</td>
<td>alright</td>
</tr>
<tr>
<td></td>
<td>python</td>
<td>of</td>
</tr>
<tr>
<td></td>
<td>python</td>
<td>import</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>class</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>true</td>
<td>python</td>
</tr>
<tr>
<td></td>
<td>as</td>
<td>apex</td>
</tr>
<tr>
<td></td>
<td>of</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>file</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>web</td>
<td>scraping</td>
</tr>
<tr>
<td></td>
<td>file</td>
<td>vim</td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>ml</td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td>input</td>
<td>prime</td>
</tr>
<tr>
<td></td>
<td>python</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>break</td>
<td>loop</td>
</tr>
<tr>
<td></td>
<td>mysql</td>
<td>or</td>
</tr>
</tbody>
</table>

| GloVe | of | not | its | python | an | or | code | class | on | dont | misc |
|-------|python | of | book | or | learn | its | on | my | dont | so | python learning |
|       | mxr | cats | intrigue | yess | constructive | powering | eye | gents | mp | yall | misc |
|       | discord | bot | token | help | this | channel | removed | free | please | anyone | discord related |
|       | flask | of | django | python | or | im | backend | it | data | are | web framework |
|       | python | install | windows | it | pycharm | error | of | my | run | bit | python installation |
|       | of | file | vim | timestamp | its | not | pyc | an | my | ide | Python ide |
|       | python | scraping | selenium | this | want | bs | are | using | pages | browser | web scraping |
|       | file | pi | of | python | on | seek | write | dont | open | so | file operations |
|       | hack | python | user | so | on | im | testing | of | was | google | misc |
|       | key | of | with | python | on | android | its | so | phone | im | android programming |
|       | python | engine | cython | not | of | general | programming | at | written | dont | cython |
|       | of | on | learning | so | my | python | ml | tensorflow | am | code | python learning(ML) |
|       | thanks | gpt | coco | axes | sent | wow | alright | cool | me | this | misc |
|       | regex | columns | html | match | so | mysql | parser | tags | num | yeah | regex |
|       | main | ups | script | imports | module | package | not | manually | dont | directory | python packages |

Figure 29: Sample topic labeling for GloVe-SO and GloVe