

Automated Text Classification

Group:

Desy Natasha Haloho a1941816

Yu-Chieh Liao a1915688

**The University of Adelaide**

4533\_COMP\_SCI\_7417\_7717 Applied Natural Language Processing

Lecturer: Dr. Orvila Sarker

**Table of Contents**

[1. Abstract 3](#_Toc196039283)

[2. Introduction 4](#_Toc196039284)

[3. Data Collection 5](#_Toc196039285)

[3.1. API URL and Parameters 6](#_Toc196039286)

[3.1.1. API URL 6](#_Toc196039287)

[3.1.2. Parameter 7](#_Toc196039288)

[3.2. Process Workflow 9](#_Toc196039289)

[3.3. Variables in the collected data 10](#_Toc196039290)

[3.4. Statistics of the collected data 10](#_Toc196039291)

[4. Data Preprocessing 15](#_Toc196039292)

[4.1. Regex Cleaning 15](#_Toc196039293)

[4.2. Lowercasing 15](#_Toc196039294)

[4.3. Lemmatisation 16](#_Toc196039295)

[4.4. Stopword Removal 16](#_Toc196039296)

[4.5. Overall Process 16](#_Toc196039297)

[5. Data Visualisation 17](#_Toc196039298)

[5.1. Word Cloud 17](#_Toc196039299)

[5.2. Word Frequency 18](#_Toc196039300)

[6. Automated Text Classification 20](#_Toc196039301)

[6.1. Methodology 20](#_Toc196039302)

[6.1.1. BERTopic 20](#_Toc196039303)

[6.1.2. POS Tagging 20](#_Toc196039304)

[6.1.3. Dependency Parsing 21](#_Toc196039305)

[6.1.4. Rule-based Method 22](#_Toc196039306)

[6.1.5. Cosine similarity with embeddings 22](#_Toc196039307)

[6.1.6. Evaluation Metrics 24](#_Toc196039308)

[6.2. Implementation 25](#_Toc196039309)

[6.2.1. Defining Category Dictionary 25](#_Toc196039310)

[6.2.2. Result and Discussion 28](#_Toc196039311)

[6.3. Final workflow 35](#_Toc196039312)

[7. Conclusion 38](#_Toc196039313)

[8. References 39](#_Toc196039314)

# Abstract

This study focuses on collecting and analyzing data from a well-known forum centered around computer-related questions. The main goal is to explore and cluster posts based on custom-defined categories, using various features extracted from each entry. Throughout the process, a range of data preprocessing techniques will be applied and tested in different combinations to identify the most effective approach. Evaluation will rely on widely used metrics in unsupervised machine learning to determine the optimal method for organizing and interpreting the data. To achieve this, we will apply two main clustering strategies: a traditional rule-based model and a modern BERT-based pretrained model. By leveraging these techniques, the research also aims to uncover trends and shifts in user interests over time, offering deeper insights into the evolving landscape of discussion topics.

# Introduction

Online forums play a huge role in how people share knowledge, especially in fields like programming and computer science. Platforms like Stack Overflow have become well known places for developers to ask questions, find solutions, and learn from each other. With so much content being generated every day, it’s become increasingly important—and interesting—to figure out what kinds of topics people are talking about and how those interests change over time.

Manually organizing and making sense of this content is time-consuming and not always consistent. That’s where automated approaches like clustering come in. By grouping similar posts based on their content, we can start to see patterns and trends that would be hard to spot otherwise.

In this study, we’re diving into that process. Using a mix of traditional rule-based methods and a more modern approach with BERT, a powerful pre-trained language model, we’ll explore how different clustering strategies perform. Along the way, we’ll look at which preprocessing techniques work best and what they reveal about user behaviour and topic popularity over the years.

# Data Collection

The data used in this system will be collected from Stack Overflow using the Stack Exchange API, focusing on challenges and solutions related to NLP. In Stack Overflow, each post consists of a question, tags, and an identifier indicating whether it has answer and whether an answer has been accepted. An accepted answer means that the author who posted the question received an answer that worked for them personally [2]. Often, this means the answer provided by the community successfully resolve their problem. In addition, each post also has one or more tags, which are words or phrases that describe the topic of the question [3]. We will use tags to focus on collecting post that related to NLP topic.

To build the system, we aim to collect posts that have accepted answer as a priority, as this ensures our system is credible and useful for improving efficiency in solving common challenges by developers. We want to collect posts that include the title, description of the post, tags, answer. The process will be divided into two types of data requests. The first will request information related to the post question which include the title, question body, tags, and answer ID. The second will request information related the post answer that includes the answer body.

In our system, we included 20,000 posts focused on collecting post that have accepted answer and are related to NLP. To achieve this, we started the data request from NLP collective and NLP tag for posts with an accepted answer. Collectives on Stack Overflow are a newer feature, serving as dedicated spaces where developers can find content organized around specific area of technical practice by connecting a group of related tags [4]. We retrieved posts from the NLP collective and tags, which were later evaluated by ID to ensure there was no duplication in our system.

We obtained 2,500 posts from the NLP collective posts with an accepted answer and 9,077 posts from the NLP tag with an accepted answer. To ensure the system remains focused on the NLP topic, we also collected posts from the NLP tags that had at least one answer and we obtained 6,535 posts. To enrich the dataset, we extended our data request by collecting posts with an accepted answer from more general tags that is machine learning, deep learning, and python that still relate to the NLP topic. We used 1,888 posts from these tags to maintain the focus on the NLP topic.

Each data request resulted in a different JSON file. Below are the detailed component of the data request that we performed to construct our dataset.

**Table 3.1** Data request used to construct the system dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| **Priority** | **Data Request** | **Count** | **Output File** |
| 1 | NLP collective with accepted answer | 2,500 | stack\_overflow\_posts\_nlp\_collective.json |
| 2 | NLP related tags with accepted answer | 9,077 | stack\_overflow\_posts\_nlp.json |
| 3 | NLP related tags that have answer | 6,535 | stack\_overflow\_posts\_nlp\_not\_accepted\_answer.json |
| 4 | Broader tags with accepted answer | 1,888 | stack\_overflow\_posts\_nlp\_general.json |

## 3.1. API URL and Parameters

To request data from the Stack Exchange API, we needed to define the API URL and input parameters to obtain the desired data. As mentioned, we performed four rounds of data request, each consisting of two types to retrieved question and answer. Below, we provide detailed information about the URL and parameters used for each data request.

### 3.1.1. API URL

There are two components of Stack Overflow post that we retrieved that is the question and answer. We needed to specify different URLs for different purposes, and we also utilized the advanced search feature provided by Stack Exchange API. Advanced search gives us flexibility to specify our criteria and provides personalized URL to match our requirements.

In our system, we used the advanced search to retrieve questions posts that have accepted answers. This allowed us to obtain only questions from posts that have accepted answer. Below is the primary API URLs used in our system.

**Table 3.2** Primary API URLs used to perform data request in the system.

|  |  |
| --- | --- |
| **Usage** | **API URL** |
| Retrieve question information from the post | https://api.stackexchange.com/2.3/questions |
| Retrieve answer information from the post | https://api.stackexchange.com/2.3/answers |
| Retrieve question information from posts with accepted answer | https://api.stackexchange.com/2.3/search/advanced?accepted=True |

Following that, here are the detailed API URLs we used for each data request along with the usage.

**Table 3.3** API URLs for each data request.

|  |  |  |
| --- | --- | --- |
| **Data Request** | **Usage** | **API URL** |
| **(1)** NLP collective with accepted answer  **(2)** NLP related tags with accepted answer  **(4)** Broader tags with accepted answer | Retrieve question information from posts that has accepted answer sorted by activity in descending order. From this step, we will retrieve the answer\_id from the accepted answer. | https://api.stackexchange.com/2.3/search/advanced?order=desc&sort=activity&accepted=True |
| Retrieve answer information from posts by answer\_id. Thus, we could obtain the body of the accepted answer. | https://api.stackexchange.com/2.3/ answers /{ids} |
| **(3)** NLP related tags that has answer | Retrieve question information from posts. From this step, we will retrieve the answer\_id from each post. | https://api.stackexchange.com/2.3/questions |
| Retrieve answer information from post by answer\_id that was obtained in the previous step. The answer that will be retrieved only the one with the highest score. | https://api.stackexchange.com/2.3/questions/{ids}/answers |

### 3.1.2. Parameter

There are several parameters used to ensure the collected data aligns with our needs. Each parameter defines different criteria, as described below:

**Table 3.4** Description of parameters used in the Stack Exchange API.

|  |  |
| --- | --- |
| **Parameter** | **Usage** |
| site (string) | Specifies the particular to connect on the Stack Exchange Network. |
| filter (string) | Determine which field are included in the response. We use this to include the body content of question and answer requests. |
| key (string) | A unique API key used to increase the request quota. This can be created from [Stack Apps](https://stackapps.com/apps/oauth/register). |
| tagged (string) | Filter result based on specific tag. |
| collective (string) | Filter result based on specific collective |
| page (int) | The page number to retrieve the results. |
| pagesize (int) | The number of items per page with maximum of 100 items. |
| order (string) | Specifies the order of the results. |
| sort (string) | Specifies the sorting field of the result. |

Below are the detailed parameters used for each data request:

**Table 3.5** API parameter for each data request.

|  |  |  |
| --- | --- | --- |
| **Data Request** | **Parameter** | **Value** |
| **(1)** NLP collective with accepted answer | site | stackoverflow |
| filter | withbody |
| collective | nlp |
| pagesize | 100 |
| order | desc |
| sort | activity |
| accepted | True |
| **(2)** NLP related tags with accepted answer | site | stackoverflow |
| filter | withbody |
| tagged | nlp, tokenize, recurrent-neural-network, python;text-classification, python;regex, machine-learning;text, python;lstm, python;visualisation |
| pagesize | 100 |
| order | desc |
| sort | activity |
| accepted | True |
| **(3)** NLP related tags that has answer | site | stackoverflow |
| filter | withbody |
| tagged | nlp, tokenize, recurrent-neural-network, python;text-classification, python;regex, machine-learning;text, python;lstm, python;visualisation |
| pagesize | 100 |
| order | desc |
| sort | votes |
| **(4)** Broader tags with accepted answer | site | stackoverflow |
| filter | withbody |
| tagged | machine-learning, deep-learning, python |
| pagesize | 100 |
| order | desc |
| sort | activity |
| accepted | True |

## 3.2. Process Workflow

In this section, we provide the detailed explanation of the data collection to better understand the overall workflow. As mentioned earlier, each data request consists of two types that is to retrieve the question and answer. The process follows these steps:

1. Define the parameters to be included in the API URL.
2. Perform a data request for the question, including the question\_id. If the request is for questions with an accepted answer, it will also include the accepted\_answer\_id.
3. Perform a data request for the answer using the accepted\_answer\_id. to retrieve the answer body for the specific ID. If the question does not have an accepted answer, we request the answer body for the answer with the highest score for that specific question\_id.
4. Saved the result into a JSON file.
5. Repeat steps 1-4 for each data request specified in Table 1.
6. Merged the obtained JSON files and convert into a single DataFrame.
7. Remove the unnecessary columns to create an effective and clean dataset that is ready to use.

A black background with a black square

Description automatically generated with medium confidence

**Figure 3.1** Workflow of retrieving NLP related posts from Stack Overflow.

## 3.3. Variables in the collected data

The dataset consisting of our retrieved Stack Overflow posts is stored in a variable called tidy\_posts\_df that will be used throughout our analysis. The complete list of columns in this dataset is shown below.

**Table 3.6** Overview of the variables in the data.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| title | The title of the question posted on Stack Overflow. |
| body | The description of the question. |
| tags | A list of tags that describe the topic related to the question. |
| answer\_body | The content of the accepted answer or the answer with the highest score. Each post has at least one answer, as demonstrated in the workflow where we only retrieved posts with either an accepted answer or at least has one answer. |
| is\_answered | A boolean indicating whether the question has an accepted answer. In our data, posts with value of False indicate that the post does not have an accepted answer but does have answer. |
| creation\_date | The timestamp when the question was posted. |
| score | The total number of upvotes minus downvotes on the question. This score reflects how the community evaluates the post, indicating whether it contains useful or incorrect information. |
| answer\_count | The number of answers the question has received. |
| view\_count | The total number of times the question has been viewed. |
| link | The URL link to the post on Stack Overflow. |

## 3.4. Statistics of the collected data

Before using the data in our system, we want to explore key aspects of the data to better understand its distribution and structure. To achieve this, we will extract summary statistics and create visualizations to analyze the patterns and distribution within the data.

1. Statistics of score, view\_count, and answer\_count

Here, we want to evaluate the distribution of score, view\_count, and answer\_count by extracting the summary statistics, as shown below:

**Table 3.7** Summary statistics of score, view\_count, and answer\_count.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **score** | **view\_count** | **answer\_count** |
| **Count** | 20000 | 20000 | 20000 |
| **Mean** | 10.26 | 13517.08 | 1.98 |
| **Std** | 93.30 | 118302.85 | 2.81 |
| **Min** | -8.00 | 5.00 | 1.00 |
| **25%** | 0.00 | 129.00 | 1.00 |
| **50%** | 1.00 | 581.00 | 1.00 |
| **75%** | 2.00 | 2319.25 | 2.00 |
| **Max** | 5920.0 | 6959711.00 | 108.00 |

The statistics shows that the median of views, votes, and answers are relatively low, while the mean numbers are significantly higher. This suggest that there are some posts that significantly higher and get a ton of attention. These super popular posts raise the averages number well above the median, as reflected by the high variance in these metrics. This indicates that generally developers may not be actively using Stack Overflow as their main platform for solving problems.

1. Correlation between metrics

Following that, we also want to evaluate the correlation between score, view\_count, and answer\_count. In addition, we want to see if text length has any correlation with these metrics. To achieve this, we will use a heat map to easily interpret the results.

A screenshot of a graph

Description automatically generated

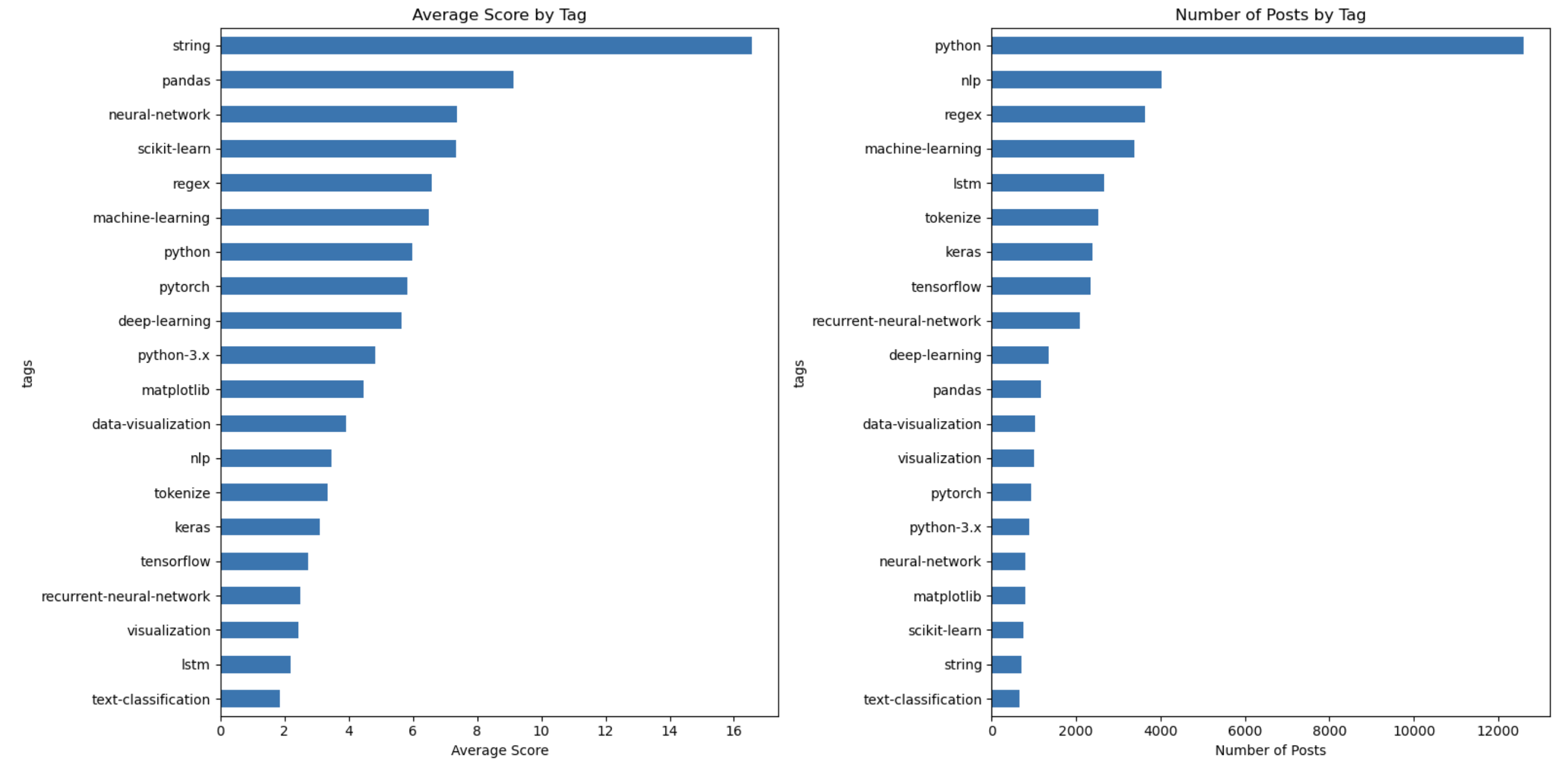
**Figure 3.2** Correlation matrix between metrics

The correlation matrix shows that score, view\_count, and answer\_count are strongly related to each other, indicating that popular posts tend to be reflected across these metrics. In contrast, features like title length, body length, and answer length show weak correlations with engagement, indicating that the text's length has little impact to the popularity or success.

From this result, although the score, view\_count, and answer\_count appear to be important indicator for identifying popular posts that could be used to indicate post with more credible answer, we decided not to use them as features due to their significantly high variance. In addition, these metrics also accumulated overtime, meaning that older posts naturally have higher values regardless of the quality, which may affect fairness. Thus, we will be focusing on the linguistic feature to perform the text classification.

1. Analysis by Tags

Furthermore, we aim to analyze the distribution of our data by the tag. Here, we evaluate which tags are associated with more useful answers by looking at the average score for each tag. In addition, we also want to analyze the number of posts per tag in our data.



**Figure 3.3** Plot analysis by tags

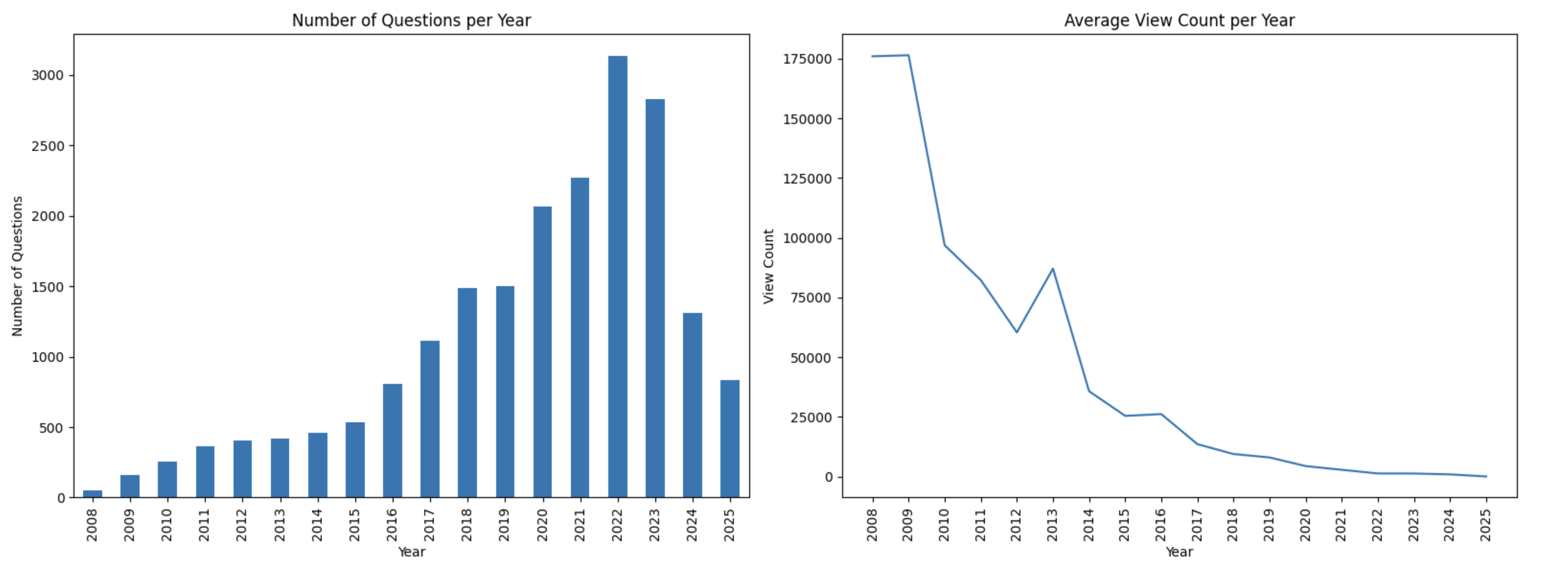
From the plots, we found that general tags like 'python', 'pandas', and 'string' tend to get higher average scores. This indicates that questions with widely applicable or foundational topics often receive more useful solution for developers and attract more attention. On the other hand, specialized tags like 'lstm', 'tokenize', and 'text-classification' are less common and usually score lower.

Although our data collection focuses on posts tagged with ‘nlp’ tag, we found that tagswith the highest number of posts is 'pyhton'. The 'nlp' tag ranks second, with a significant difference compared to ‘python’ tag.

However, it also important to note that each post in Stack Overflow has more than one tag, ranging from general to more specific core concepts tags. Since 'python' is a broader tag, thus, 'python' is likely to accompany almost every other tags, as it forms the foundation of most problems. Apart from that, we can observe that there are some tags that may not always be directly related to NLP topics such as 'machine-learning', 'deep-learning', and 'data-visualization'. This point will be taken into account in the following part.

1. Time-based Trends

Finally, we want to evaluate the distribution of our data over the years, as the data we collected starts from when Stack Overflow was created in 2008.



**Figure 3.4** Plots time-based analysis

From both plots we found that over the years, the total number of questions asked each year has been increasing, having the highest number in 2022 with over 3,000 questions. However, in 2023, there was a noticeable decline where the number of questions continued to decrease until 2025.

On the other hand, the average view count per question has been significantly decreasing, from 2008 to 2025. The number has dropped significantly, especially in recent years.

These trends indicate that the number of developers using Stack Overflow to share their problems has been increasing, shows by the high number of questions. However, the number of developers using Stack Overflow to find their answer has been decreased (decline in engagement) from the low number view counts. This also shows that, as engagement declines, the more recent answers collected in our system might not be the best option, as they may not have sufficient community validation.

# Data Preprocessing

The collected data follow HTML format, which consists of various HTML tags. Therefore, we implement several data preprocessing steps to ensure the text used to classifying each post is optimized and cleaned from noisy character, which will helps improve classification performance.

In this system, we apply four types of preprocessing techniques, that is regex cleaning, lowercasing, lemmatisation, and stopword removal. We later experiment with multiple combinations of these preprocessing techniques to evaluate their effectiveness. Additionally, we apply these preprocessing combinations to three text columns that is the title (title), the question (body), and the concatenation of the title and question (title\_body). This approach aims to determine which part of the post is most effective for accurate classification, as well as which preprocessing techniques are best suited to the text column. This will allow us to improve the effectiveness of the text classification and eliminate unnecessary preprocessing, which enhances the overall runtime process.

## 4.1. Regex Cleaning

Regex cleaning removes unnecessary characters, HTML tags, and links from the text. This helps prevent confusion during the classification and assists in retrieving the main idea of the text by keeping only the natural language content. The regex cleaning process follows these steps:

1. Remove newline characters (\n).
2. Remove URL links in text (https://).
3. Remove double quote tag in text (&quot).
4. Remove code block from the text that follows this pattern <pre><code> text </code></pre>.
5. Remove the rest of text format tags while preserving the content inside that follow this pattern <tags> text </tags>.
6. Remove any non-alphanumeric characters. The final text will contain only letters, numbers, and spaces.

We define this process in a function called regex\_text. This implementation of regex cleaning is lightweight and helps us achieve results faster, which important given the large size of our dataset and the length of the text.

## 4.2. Lowercasing

Lowercasing is implemented to normalize text that helps to make text more consistent and reduce the vocabulary size. This method ensures that words with different capitalization treated as a single word.

## 4.3. Lemmatisation

Lemmatisation is another text normalization technique that we implement to reduce words to their base or root form [5 p.05]. This allows the classifier to interpret different word forms as a single base word rather than separate entities that will improve the consistency in the classification. In our system, we use the Spacy library for lemmatization, as it is known for the accuracy and speed. Compared to the NLTK library, Spacy has significantly faster performance, which is especially important given the large volume of data we are processing [6].

## 4.4. Stopword Removal

Stopwords are high-frequency words that carry little semantic weight and often do not contribute much to the meaning of the text [5 p.274]. Therefore, removing these words help reduce noise and allows us to focus on meaningful semantic words, which can also improve processing time. In our system, we remove stopwords using NLTK library, which widely used as it provides a comprehensive list of common English stopwords.

## 4.5. Overall Process

The text preprocessing is defined within the text\_preprocessing method, which receives boolean arguments of regex, remove\_stop\_word, lemmatization, and lower\_case. These arguments specify which techniques should be applied to the text. In the previous part, we mentioned that the regex cleaning defined in regex\_text function, which is called within the text\_preprocessing method to integrate all techniques into a single method for easier control in the future.

A close-up of a questionnaire

Description automatically generated

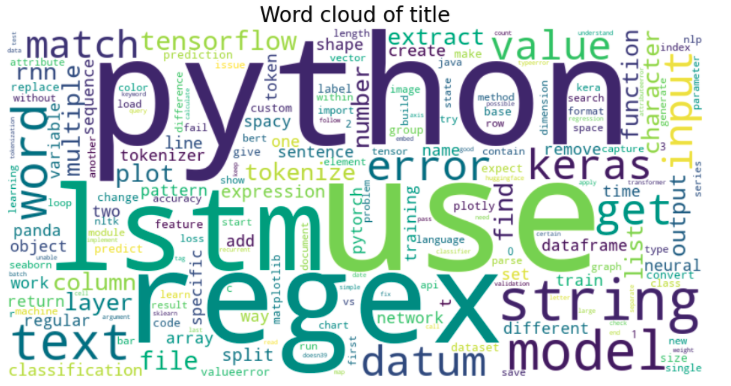
**Figure 4.1** Example of the data preprocessing result.

# Data Visualisation

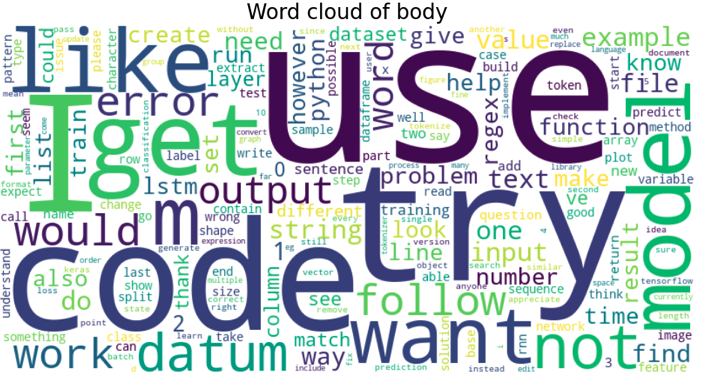
In building our system, we aim to experiment with three different text columns that is title, body, and title\_body to determine which one provides better separation for our classifier. In this section, we explore the linguistic distribution within each column to identify the most frequently occurring words. The report uses word clouds and bar charts to visualize the word frequency.

## Word Cloud

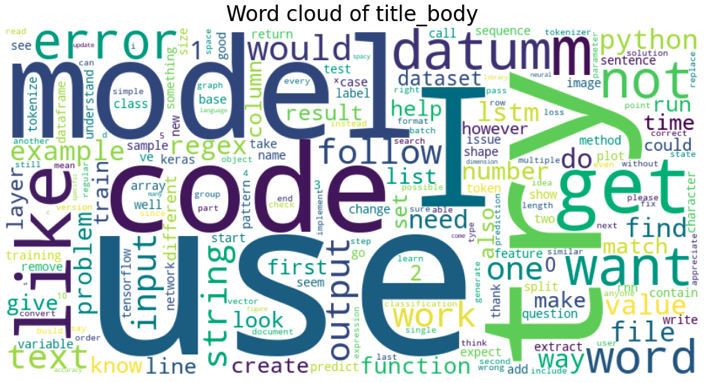
First, we generated the cloud chart based on three text columns: title, body, and title\_body.



**Figure 5.1** Word cloud of ‘title’ column.



**Figure 5.2** Word cloud of ‘body’ column.

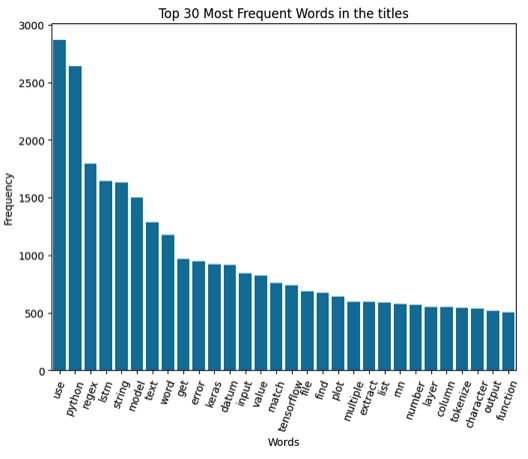


**Figure 5.3** Word cloud of ‘title\_body’ column.

From the word cloud, it is apparent that the title column contains more meaningful words with high frequency. For example, words like 'lstm', 'regex', and 'string' appear frequently and reflect the core topics of the posts. However, body and title\_body contains more noise that can be seen from the frequent occurrence of more general and less meaningful words, such as 'use', 'try', 'code', and 'model. This indicates that the title column offers concise summary of the post that will be powerful for our text classification.

## 5.2. Word Frequency

Following that, we display the bar chart of word frequency from the title column to further support our argument.



**Figure 5.4** Word frequency plot of ‘title’ column.

From the plot, although some of the high frequency words may be less meaningful, many represent technical terms related to the core problems described in the posts. These terms are valuable for task-based categorization. With these technical terms such as 'regex'. 'string', 'keras', 'tensorflow', and 'tokenize, we could easily categorise our data based on the NLP tasks using `title`. Thus, these results suggest that the title column contains strong semantically meaningful words that are more likely to provide better separation for our classification task.

# Automated Text Classification

This section will explain the algorithms and evaluation metrics used to build and assess the performance of the unsupervised model, the implementation of each algorithm, and outlines the final workflow of the classifier.

## Methodology

In this section, we provide the underlying theory of the algorithms used to perform various tasks in building the classification system.

### BERTopic

To better understand the semantic relationship within our text data, we utilize BERTopic to generate cluster of semantically similar texts, which can be implemented with a simple line of code. BERTopic generates topic representations that has the ability of semantic understanding through the use of an embedding component to produce meaningful topics understandable to a human (Axelborn & Berggren 2023, p. 27).

BERTopic operates in three main steps. First, each document is converted to its embedding representation using a sentence-transformer (SBERT). Then, the dimensionality of the resulting embeddings is reduced using the UMAP algorithm to optimize the clustering process with HDBSCAN algorithm. Lastly, topic is generated from the clusters of documents using a custom class-based variation of TF-IDF to extract representative keywords for each topic (Grootendorst 2022, p. 2).

Although BERTopic assumes that each document only contains a single topic, which does not reflect the reality that documents may contain multiple topics, this aligns with our needs as we intend to assign each post to a single dominant category. It has no limitation on how large datasets it can handle and has been shown to achieve competitive performance compared to other state-of-the-art topic modeling algorithms, such as LDA (Axelborn & Berggren 2023, p. 40), making it suitable for our system, especially since we are working with a large data with long texts.

To build the classifier in our system, we observed the result from BERTopic to explore the latent structure of our dataset that provides insights into contextual similarities across documents. The result from BERTopic will be used as a reference point for us to manually define our dictionary of category and keywords to perform the categorisation.

### POS Tagging

Part-of-speech (POS) tagging assigns a grammatical label to each word in a sentence. This method begins with breaking text into individual words or tokens, followed by analyzing the lexical information and context of each word to determine the appropriate tag. In the context of Stack Overflow posts, which often contains technical descriptions, code explanations, or problem explanations, we can filter the key ideas by extracting words with specific POS tags that helps to ensure that our text retains only the most meaningful information. Here, we aim to explore whether filtering text in this way can improve the classifier ability to accurately capture the meaning of each post. We retain only words tagged as NOUN, PROPN, VERB, and ADJ, as explained below:

**Table 6.1** Description of POS tags used for text extraction

|  |  |
| --- | --- |
| **Tag** | **Description** |
| NOUN and PROPN | Stack Overflow posts frequently mention technical concepts, tool or library names, and programming languages. These tags capture domain-specific terminology, objects, and concepts in the problem, such as function, python, error, or syntax. |
| VERB | Question often describes actions developers trying to solve in the problem, such as run, install, compile. |
| ADJ | It captures information that often used to describe the condition of problem, such as slow or missing. |

### Dependency Parsing

Dependency parsing analyzes the grammatical structure of a sentence by identifying dependencies between words. It can be illustrated using a dependency tree, with words represented as nodes and the dependencies as directed edges, as shown below.

A diagram of a flight

Description automatically generated

**Figure 6.1** Example of dependency parsing tree

Stack Overflow posts often contain irregular grammar or partial sentences, which we can filter using dependency parsing. Similar to POS tagging, we use dependency parsing to retain only the syntactically important parts of the sentence. We retain only words with the following dependency labels ,which are ROOT, nsubj, dboj, compound, amod, as explainedbelow:

**Table 6.2** Description of dependency relation used for text extraction

|  |  |
| --- | --- |
| **Relation** | **Description** |
| ROOT | Represents the central concept in a sentence. This element is essential as it captures the primary focus in the discussion. |
| nsubj | Identifies the subject of the sentence that particularly important for capturing which component is experiencing an issue or performing an action, for example “the library crashes…” or “the function returns …”. |
| dboj | Identifies the direct object being acted upon, capturing the target of the operation, for example “clean the text”. |
| compound | Captures multi-word technical terms that are often used to describe many technical terminologies, for example “data frame”. |
| amod | Describes adjectival modifiers of nouns that provide information about the technical elements, for example “missing value”. |

### Rule-based Method

The rule-based method works by using a set of predefined rules and keywords to classify text into different groups. Unlike machine learning models that learn patterns from data, this approach uses straightforward logic that is if certain words appear in the text that match the keywords, a score is assigned to the matching category. It's easy to understand and gives you full control over the classification process. The category with the highest score is then used to label the text. If the text does not match any of the keywords, it is labelled as 'other'. This method does have limitations, as it can struggle with complex or ambiguous text that does not included in the defined keywords. Also, it may overlook relevant content if certain keywords never appear in the dataset. However, it serve as a simple and interpretable method for categorisation that allow us to categorise text without training process.

### Cosine similarity with embeddings

The cosine similarity method begins by converting the post and category dictionary into embeddings, which are numerical representations that capture the semantic meaning of text. Each post is embedded using a language model, while the keywords associated with each category are first concatenated into a single sentence string and then embedded. The embedding vector of the post is then compared with each category's embedding vector using cosine similarity to determine which category is semantically closest to the text. In this project, we will explore the performance of cosine similarity with different embeddings using sentence-transformer (SBERT) and CodeBERT.

#### Embeddings

SBERT is a modification of BERT to improve performance on sentence similarity tasks. Standard BERT generates contextual embeddings for each token and requires comparing sentence pairs individually, which is computationally expensive and inefficient for large-scale similarity tasks (Axelborn & Berggren 2023, p. 13). SBERT addresses this limitation by using a Siamese network structure to produce fixed-size sentence embeddings that can be compared using cosine or Euclidean similarity. In our project, we use the SentenceTransformer library that provides implementations of SBERT and its variants. The `all-MiniLM-L6-v2` is a pre-trained transformer-based language model that has small size and offers strong performance compared to the other pre-trained transformer, which will be useful for our large dataset.

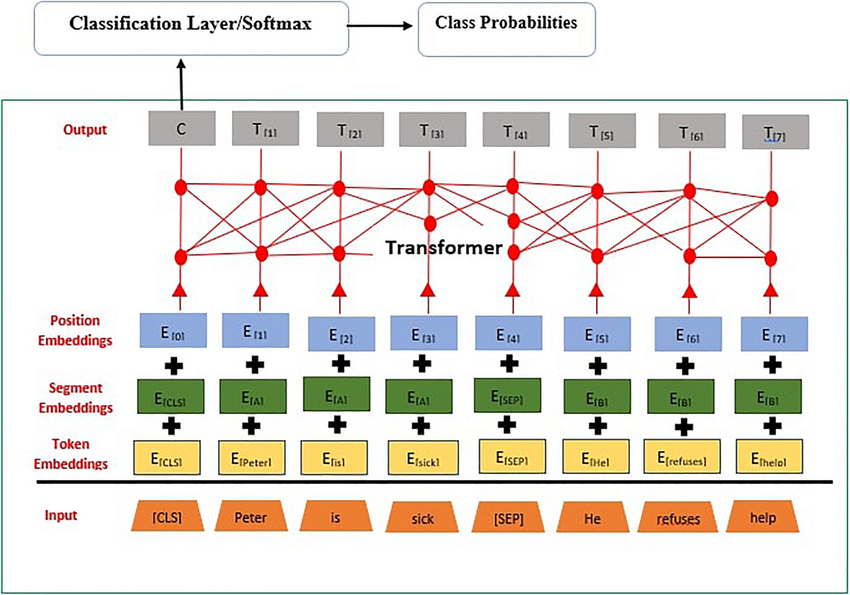


Figure 6.2 Bert model architecture

Above is a simple illustration of the BERT model architecture to review how the algorithm works. BERT starts by breaking the input text into smaller units called tokens. Each token is then combined with two other types of information that is segment embeddings, which help the model distinguish between different parts of the input, and position embeddings, which indicate the order of the tokens in the sentence. This combined information is passed through multiple layers of transformers, allowing the model to understand the context and relationships between words. This architecture forms the foundation of SBERT’s ability to generate meaningful sentence-level embeddings.

CodeBERT is a bimodal pre-trained model developed to support natural language and programming language understanding. It also extends the BERT architecture and is trained on paired data of natural language (NL) and programming language (PL) from GitHub repositories (Feng et al. 2020, p. 1). This design allows CodeBERT to perform well on code-related tasks such as code search and code summarization (Feng et al. 2020, p. 8). As we know the posts text in Stack Overflow often contains programming texts (such as library names or code block) that SBERT may not be capture semantically. Thus, we also want to explore the performance of CodeBERT to embed our text.

#### Cosine Similarity

Cosine similarity is a measure of the degree of similarity between two vectors. It takes values between 0 and 1, where value of 0 indicates that there is no similarity between the documents and a value of 1 indicates that the documents are identical (Novotný et al. 2020, p. 2). Thus, as we want to categorize each post, we measure the similarity between the post text and category, expressing as follows:

**( 6.1 )**

where and represent the embedding vector of post text and category, respectively.

### Evaluation Metrics

As our algorithm is unsupervised, we use three metrics to evaluate how well our classifier performs, which are silhouette score, intra-cluster distance, and inter-cluster distance.

Silhouette score gives us a sense of how well each piece of data fits into its assigned cluster. It tells us whether a point is closer to its own cluster or to another one. The score ranges from -1 to 1, and the higher the score indicates better clustering, meaning the data points are grouped in a meaningful way (Joshi, T. 2021).

**( 6.2 )**

where is the average distance between point and all other points in the same cluster and is the smallest average distance between point and all points in the nearest neighboring cluster.

The overall Silhouette Score is the average of all individual scores.

**( 6.3 )**

Intra-cluster Distance looks at how close the points in the same cluster are to each other. If this distance is small, it means the points are tightly grouped, which usually means the clustering is doing a good job.

**( 6.4 )**

This is Centroid-based Intra-cluster Distance (Yadav, S. 2023), where denotes Cluster , denotes the centroid of , and K denotes the number of clusters.

**( 6.5 )**

This is Pairwise Intra-cluster Distance (Yadav, S. 2023).

Inter-cluster Distance on the other hand, measures how far apart different clusters are from each other. The bigger this distance, the better—it means the clusters are more clearly separated and distinct (Yadav, S. 2023).

**( 6.6 )**

where denotes cluster i and j, and K denotes the number of clusters.

## Implementation

In this section, we implement the algorithm outlined in the methodology on our data. It begins with defining our dictionary of category with associated keywords, followed by implementing the two methods, rule-based and cosine similarity with embeddings. Finally, we evaluate the result to determine which method best suits our data.

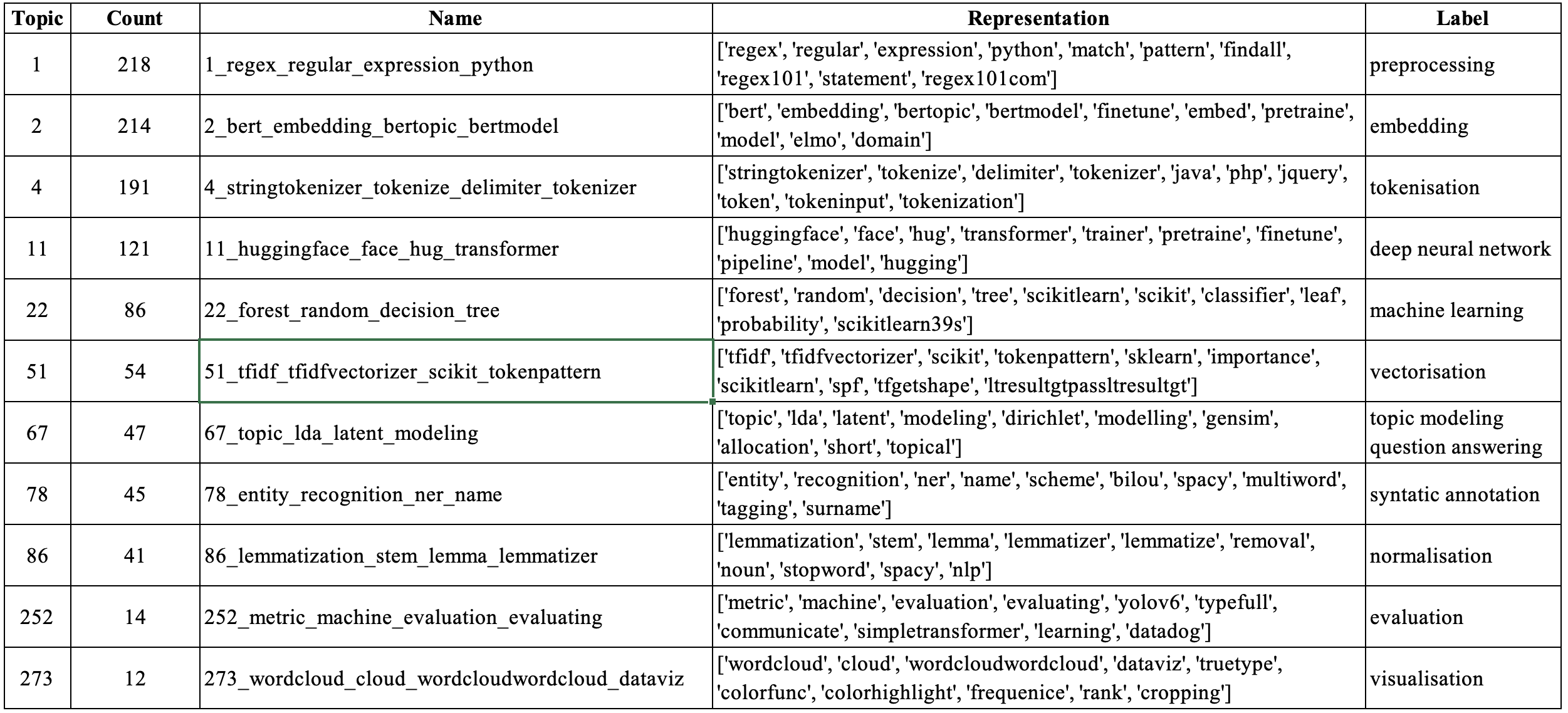
### 6.2.1. Defining Category Dictionary

Since we have a large dataset, we aim to create a custom categorization using our own category dictionary. Our goal is to build a system that consistently categorizes posts around NLP tasks, as we observed that the tags on Stack Overflow often mix task and library name. By focusing on tasks, we also align more closely with how developers typically follow a sequence of steps when working on language modeling project. Thus, this approach improves efficiency and clarity of insights.

To define our custom categories, we build a dictionary where each task label is a key, and the associated keywords related to the task are the values. This gives us more control over the category separation that ensure alignment with our objectives and offers easier interpretability.

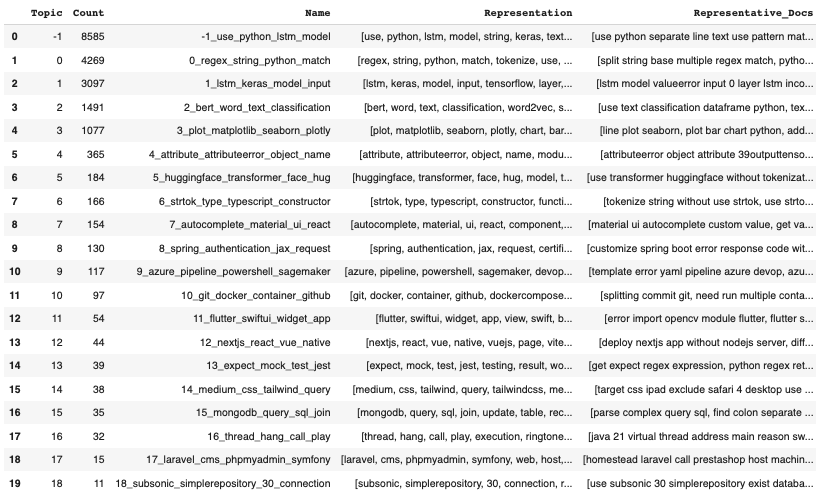
In defining our own category dictionary, we used BERTopic to evaluate the latent structure and contextual similarities across documents. It served as a reference point for manually defining our category dictionary and also allowed us to assess whether the separation based on task is suitable for our data.

From the data visualization above, we obtained that title column serve as a concise summary of the post, containing semantically meaningful words. Thus, we used title column to generate cluster using BERTopic. The result produced 298 clusters, which we manually evaluated to filter topics related to NLP. Below, we provide the preview of the detailed results we obtained:



**Figure 6.3** BERT-Topic result with manual labelling

Apart from that, we also reduced the BERTopic results to observe how topics are grouped at a broader level, as shown below:



**Figure 6.4** Reduced BERTopic results

We found that some of the topic combine task and tool/library names (1\_lstm\_keras\_model\_input), which support our understanding that most posts have mixed tags. Although, we can see there is topic that very specific to library names (5\_huggingface\_transformer\_face\_hug), more of the topic are broader and more closely related to NLP tasks (0\_regex\_string\_python\_match, 2\_bert\_word\_text\_classification, and 3\_plot\_matplotlib\_seaborn\_plotly). This indicates that classifying our data based on task is a suitable approach. In addition, some of the other topic are not related to NLP tasks, which shows that our data also contains noises from unrelated NLP topic. But, we can see that the number of posts falling under the NLP-related topic is larger. Therefore, we will use the more detailed output to construct the keyword for our dictionary.

After evaluating the result from BERTopic, we come up with 12 categories to classify the text based on NLP tasks. From there, we collected keywords that related to the category based on our domain knowledge, as well as terms that are semantically similar from BERTopic result. The manually defined dictionary of category and keywords are specified below.

**Table 6.3** Manually defined dictionary of category and keywords used for text classification

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Category** | **Keywords** | **Description** |
| -1 | other | - | Categorising posts that do not match any predefined category. |
| 0 | preprocessing | 'regex', 'string', 'match', 'character', 'expression', 'pattern', 'replace', 'space', 'matching', 'substring' | General text preprocessing using regular expressions and string manipulation. |
| 1 | tokenisation | 'tokenize', 'tokenizer', 'token', 'tokenization', 'delimiter', 'ngram', 'bigram', 'wordtokenize', 'streamtokenizer', 'punkt' | Tokenising text, including n-grams and tokenisation tools. |
| 2 | normalisation | 'remove', 'spacy', 'nltk', 'stop', 'stanford', 'stopword', 'stem', 'lemmatization', 'corenlp', 'wordnet' | Normalising text through techniques, which are stopword removal, stemming, and lemmatisation. |
| 3 | syntatic\_annotation | 'tag', 'entity', 'ner', 'recognition', 'parser', 'dependency', 'noun', 'pos', 'location', 'adjective' | Syntactic annotation such as POS tagging, dependency parsing, and named entity recognition. |
| 4 | vectorisation | 'vector', 'tfidf', 'countvectorizer', 'tf', 'tfidfvectorizer', 'sparse', 'bag', 'vectorize', 'vectorization', 'bow' | Representing text as vectors using sparse vector such as TF-IDF or bag of words. |
| 5 | embedding | 'bert', 'vector', 'embed', 'word2vec', 'embedding', 'bidirectional', 'fasttext', 'dense', 'cbow', 'embeddings' | Representing text into dense embeddings such as Word2Vec, FastText, and BERT. |
| 6 | deep\_neural\_network | 'lstm', 'keras', 'tensorflow', 'rnn', 'layer', 'pytorch', 'tensor', 'huggingface', 'transformer', 'recurrent' | Deep learning models and algorithms used in NLP, including RNNs and transformers. |
| 7 | machine\_learning | 'regression', 'validation', 'sklearn', 'scikitlearn', 'tree', 'random', 'cluster', 'linear', 'decision', 'forest' | Machine learning algorithms used in NLP modelling. |
| 8 | topic\_modeling\_question\_answering | 'api', 'gensim', 'question', 'topic', 'answer', 'gpt2', 'openai', 'lda', 'prompt', 'gpt3' | Topic modelling and question answering, including transformer-based approach. |
| 9 | evaluation | 'accuracy', 'matrix', 'score', 'metric', 'confusion', 'evaluate', 'evaluation', 'precision', 'recall', 'roc' | Evaluation metrics to evaluate NLP models. |
| 10 | visualisation | 'plot', 'matplotlib', 'chart', 'seaborn', 'axis', 'visualize', 'tick', 'cloud', 'barplot', 'wordcloud' | Plot for data visualisation. |

We compiled all relevant keywords and selected the top 10 keywords with the highest occurrence in our text. We choose this approach to ensure consistent and fair comparison between categories. By using the same number of keywords per category, we aim to maintain balanced semantic richness and ensure clear separation between keywords. This helps avoid redundancy in meaning between categories that supports better interpretability and separation. It also reduces the chances of overlooking important keywords that might otherwise be forgotten.

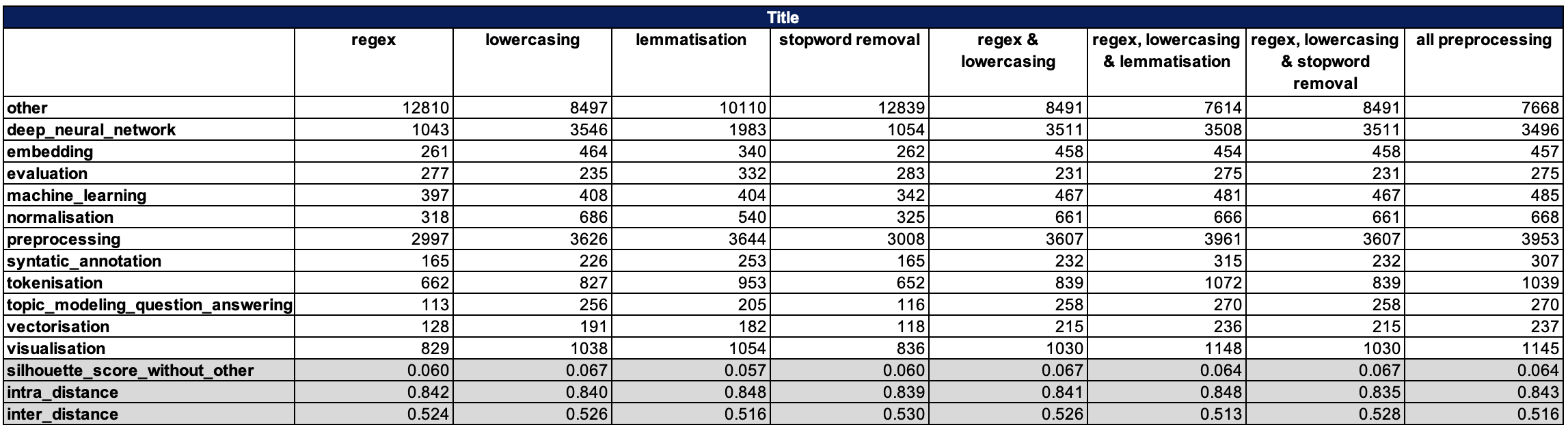
Later, we will categorise our data using this dictionary with two methods. In the rule-based method, maintaining a constant number of keywords simplifies interpretation and avoids the potential bias due to some categories having more keywords matches than others. Similarly, in the embedding based approach, we want to prevent excessive semantic signals that could lead to length-based bias, where longer strings might have denser semantic representation because it provides more context.

### 6.2.2. Result and Discussion

In this section, we discuss the performance result of both methods using different combination of preprocessing techniques. In addition, we assess the result between three text columns, which are title, body, and title\_body.

#### Rule-based method

The result and evaluation metrics obtained from the rule-based method are presented below.



**Figure 6.5** Result and evaluation metrics from the rule-based method using ‘title’ column

A screenshot of a spreadsheet

Description automatically generated

**Figure 6.6** Result and evaluation metrics from the rule-based method using ‘body column

A screenshot of a computer

Description automatically generated

**Figure 6.7** Result and evaluation metrics from the rule-based method using ‘title\_body’ column

From ***Figure 6.7***, we can see that:

1. The result using title has the best result compared to body and title\_body.
2. The result of title\_body improves compared to body but lower compared to title.
3. The result using regex, lowecasing, and stopword removal in `title` has the best silhoutte score with the lowest intra-cluster distance and highest inter-cluster distance.

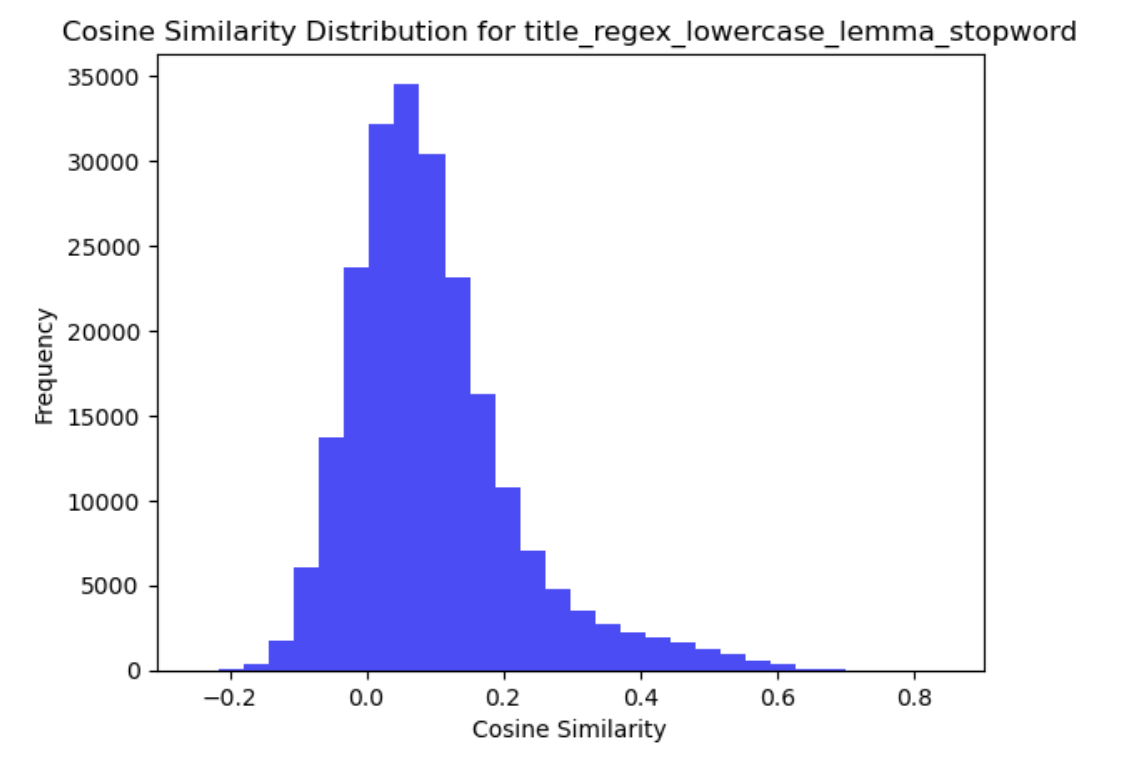
Since we obtain that the title yields better model performance compared to the other text column, it shows that title provides a concise summary of each post containing the key terms about the core concept of the post, making them highly discriminative. It contains very little amount of unnecessary text that does not appear in our dictionary keywords. While body and title\_body often include detailed explanation of the problem and tool or library names, which can introduce noise into the result. This also supported by the visualisation that we performed where we also This findings is further supported by the word cloud visualizations we performed, which show that title displayed high frequency of semantically meaningful words.

We also obtain that title\_body performs better than body which further support our argument that title carries a strong signal that contains key terms related to the post's concept. However, when combined with body, the strong signal in title gets diluted resulting in performance that is lower than using title alone. It is also common for the body to include explanation of the problem and potential answer that can blur the category separation compared to title

Lastly, we found that the best performing preprocessing techniques combination are regex, lowercasing, and stopword removal. This shows that with lowercasing it helps normalise the text, which is important for rule-based that require exact match to the keywords to obtain scores. While the regex and stopword removal helps in removing semantically low value words that creates the method to focus on distinctive vocabulary. This aligns well with rule-based methods, as it helps concentrate attention on meaningful terms and reducing less informative words.

#### Cosine similarity with embeddings

First, we evaluate the semantic consistency of the embeddings by presenting a cosine similarity distribution plot to observe how similar the text embeddings to the category keywords embeddings generated by both models.

A graph of a number of objects

Description automatically generated with medium confidence

**Figure 6.8** Cosine similarity distribution comparison between SBERT and CodeBERT embeddings.

A table with numbers and text

Description automatically generated

**Figure 6.9** Result and computation time comparison from cosine similarity method with SBERT and CodeBERT embeddings.

From ***Figure 6.8***, it is apparent that CodeBERT embeddings produce almost perfect cosine similarities, which shows the CodeBERT embedding does not provide clear distinction between different text categories and possibly create embeddings that do not affectively differentiate between different concepts. SBERT provides better nuanced embeddings that are better to distinguish categories that is more useful to classification tasks. This can further be seen by the number of 'other' in CodeBERT result, which is significantly lower compared to SBERT, showing that CodeBERT create overly similar embeddings that cause incorrectly group dissimilar texts.

Other than that, we discovered that CodeBERT has a significantly longer computation time, approximately 91 times slower than the SBERT. Thus, as we also has a large size of data, we will proceed with using SBERT as our embeddings algorithm.

Following that, we want to evaluate the threshold between three text columns using SBERT. To begin, we will examine the distribution plot of the three text columns after applying all preprocessing techniques.

A blue graph with black text

Description automatically generatedA blue graph with black text

Description automatically generatedA blue graph with black text

Description automatically generated

**Figure 6.10** Cosine similarity distribution comparison between three text columns using the SBERT embeddings

Based on ***Figure 6.10*** , all distributions show similar right-skewed patterns with most values falling between -0.2 and 0.6, and most concentrated around 0.0-0.2 before tapering off after 0.3. Thus, we will use 0.3 as the threshold for labelling across the three columns, as it sits near the right tail of these distributions, where the frequency of values significantly decreased but before it completely tapers off. We can also observe that using 0.3 as the threshold creates a reasonable separation between likely matches and the "other" category, which aligns with our initial statistics that many posts contain tag combinations outside 'nlp' alone.

Since CodeBERT did not demonstrate suitability for our data, we conducted the comparison using SBERT. The result and evaluation metrics obtained from the cosine similarity with embeddings are presented below.

A screenshot of a spreadsheet

Description automatically generated

**Figure 6.11** Result and evaluation metrics from the cosine similarity method using ‘title’ column

A screenshot of a computer

Description automatically generated

**Figure 6.12** Result and evaluation metrics from the cosine similarity method using ‘body column

A screenshot of a spreadsheet

Description automatically generated

**Figure 6.13** Result and evaluation metrics from the cosine similarity method using ‘title\_body’ column

From *Figure 6.11* , we can see that:

1. Similar to rule-based, using title yields the best performance compared to body and title\_body. The result of title\_body also improves compared to body, but lower compared to title.
2. The silhouette score results show only slight differences across various preprocessing techniques combination for title, with the highest silhouette score achieved using lemmatisation only. But, stopword removal results in a lower intra-cluster, distance even though it has a slightly lower silhouette score. Therefore, the full combination of all preprocessing techniques produces a slightly lower silhouette score than lemmatisation alone but result in a lower intra-cluster distance.
3. Although the result for body show a significantly lower silhouette score, it achieves a lower intra-distance compared to title.

We found that the title column performs best across both rule-based and cosine similarity with embedding methods. This confirms that title contains meaningful terms that produces a strong signal for classification.

In this method, we found that title produces only slight differences in result across different preprocessing techniques combination. This shows that preprocessing does not provide significant benefits in generating embeddings. As we know, embeddings have the ability to capture semantic meaning and contextual relationship. Thus, the subtle difference in result imply that the preprocessing is less crucial for title, which aligns with the fact that title contains fewer low semantic words compared to body. Also, we observed that body shows more significant differences across preprocessing techniques as it has higher text complexity. The title is written to be specific and concise, making them excellent candidates for embedding classification.

The lower intra-cluster distance in body indicates that once posts are grouped, the full text contain consistent vocabulary within categories. However, the lower silhouette scores indicate more overlap between categories that shows poor separation between categories. This happens likely due to shared technical terms across different categories that make it harder to distinct categories. The body is more likely to mention the technical terms than titles, making the separation between categories less distinct.

Overall, we will proceed with title using lemmatisation alone, as it shows highest silhouette score while maintaining a reasonable intra-cluster and inter-cluster distance. This approach also offers efficiency, as it required only one preprocessing technique to work well and more practical.

When comparing the two methods, we found that the cosine similarity with embedding clearly outperform the rule-based method. It has higher silhouette score that indicates better cluster quality, lower intra-cluster distance that indicates more coherent categorisation, and higher inter-cluster distance that indicates better separation. The embedding ability in contextual understanding helps distinguish terms that might appear across multiple categories and it also require less preprocessing, which shows it superiority.

#### Incorporating POS Tags and Dependency Parsing

A table with numbers and words

Description automatically generated with medium confidence

**Figure 6.14** Result and evaluation metrics from the cosine similarity method with POS tags and dependency parsing for word extraction using the ‘title’ column.

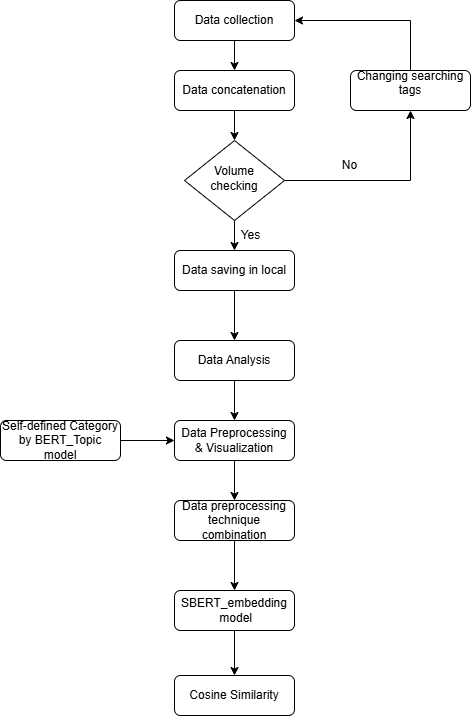
From ***Figure 6.14*** , we found that extracting words with relevant POS tags and dependency parsing could improve the performance of the model. It shows better categorisation with lower intra-cluster distance and higher inter-cluster distance compared to the previous model. Thus, this indicates that it effectively supports embedding to focus on essential vocabulary by removing more generic elements that add noise and further helps in creating better classification.

It also shows that the selection of POS tags and dependency parsing also relevant for our data where Stack Overflow post are highly task oriented that are often structured around tools and library names (`NOUN`, `PROPN`), problems or challenges that being faced (`VERB`, `ROOT`, `dobj`), and technical description (`ADJ`, `amod`).

Therefore, we can conclude our experiment with taking conclusion that the best performing model that suitable for our data is cosine similarity with SBERT embeddings, using lemmatisation, POS tags, and dependency parsing.

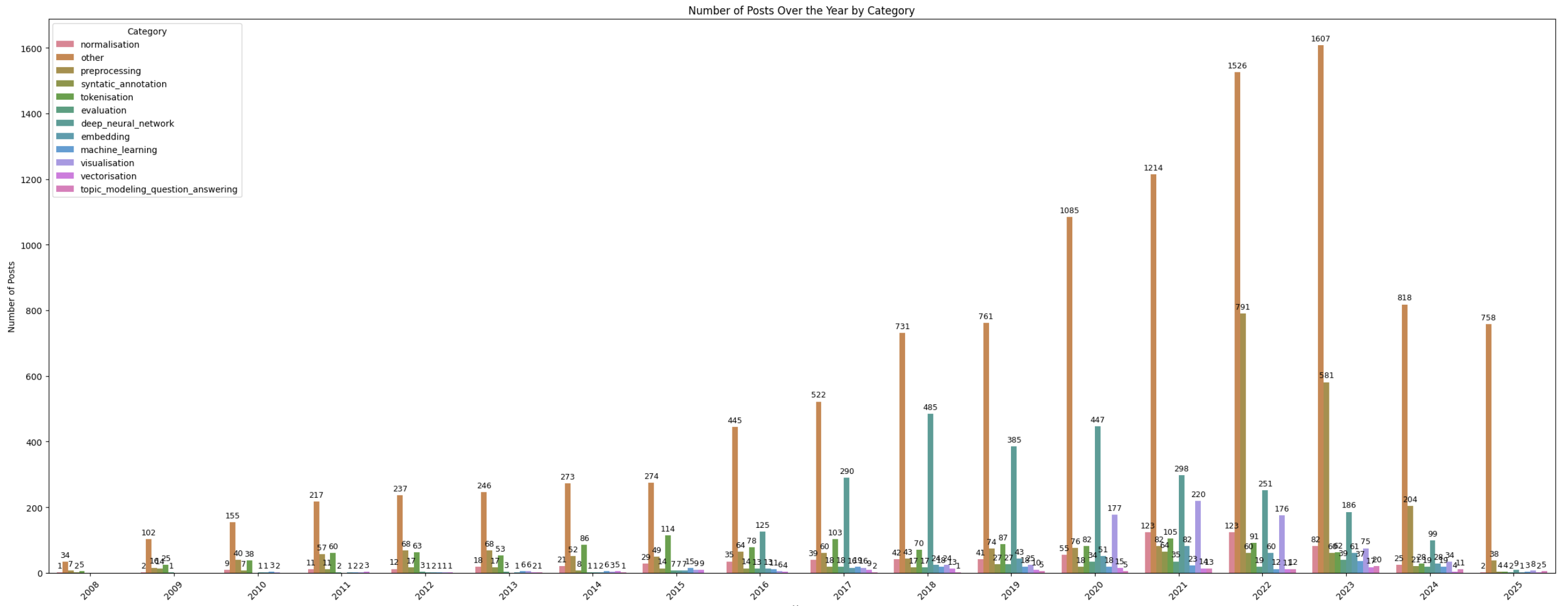
## Final workflow

In this final section, we consolidate the end-to-end process that will be implemented to our classification task. Based on the experiment, we found that the best performing model for our data is cosine similarity with SBERT embeddings, using lemmatisation, POS tags, and dependency parsing as the preprocessing techniques. To make the workflow easy to understand, we present below the overall process of the text classification system we implemented.



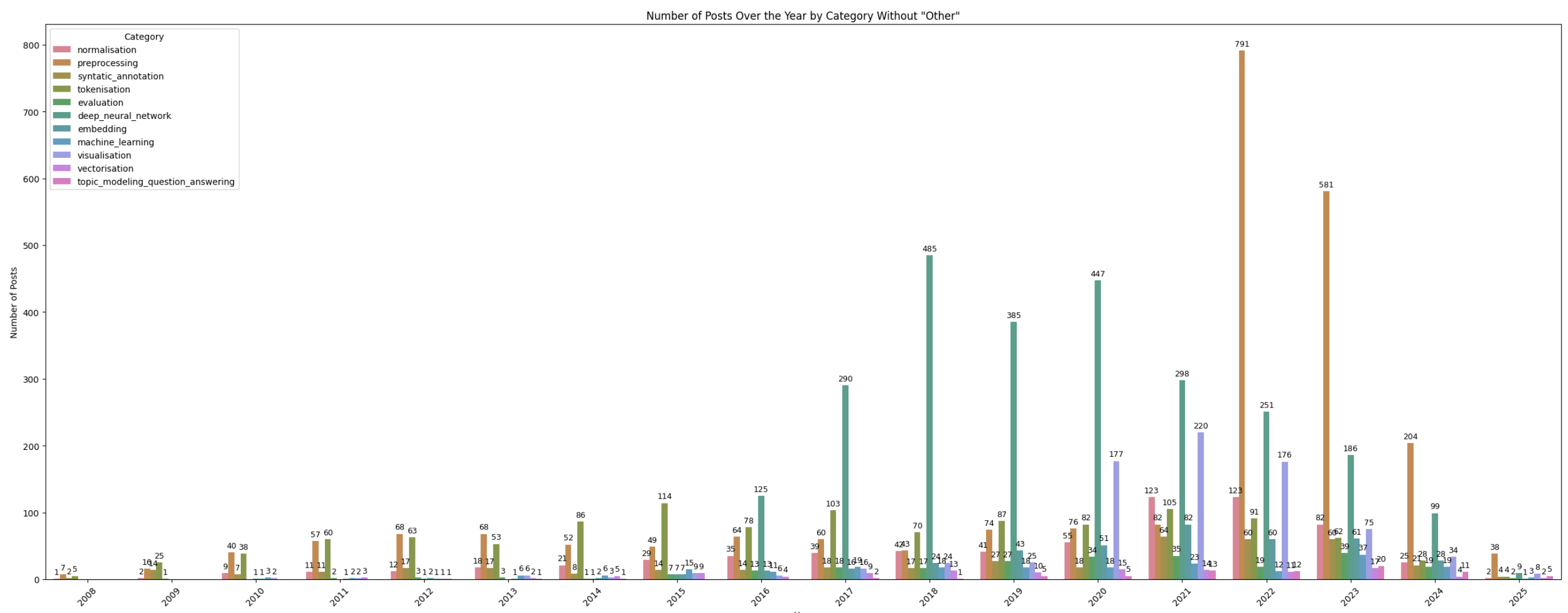
**Figure 6.15** Final workflow

Below, we also present a visualization to better understand the distribution of posts by category throughout the year.



**Figure 6.16** Number of posts by category over the year 6.17

From ***Figure 6.16***, it is apparent that the ‘other’ category consistently dominates the distribution. As shown in Figure 3.3., many of our posts contain tags that are not directly related to NLP but are mainly associated with Python. This indicates that the classification result is effectively identified topics that do not specifically discuss NLP concepts. Following this, we focus our discussion on the distribution of NLP task-based categories. To provide a clearer view, we present a visualization that excludes the ‘Other’ label.



**Figure 6.17** Number of posts by category over the year without ‘other’ label6.18

From ***Figure 6.17***, we obtained these insights:

1. In the early years, from 2008 to 2013, the number of posts related to NLP was relatively low. This is expected, as Stack Overflow had only recently launched, and the developer community was still growing. During this period, there was minimal activity in advanced algorithm categories and more dominated with foundational NLP tasks, which are normalization, preprocessing, syntactic annotation, and tokenization.
2. In the following years, from 2014 to 2015, the pattern remained largely similar to the previous period, with discussions related to advanced algorithms still limited and more focused on foundational NLP tasks. However, the total number of posts increased compared to the earlier years, indicating growth in the developer community using Stack Overflow.
3. From 2016 to 2021, posts related to deep neural network showed a significant increase and overtook in dominance. This aligns with the explosive breakthroughs and growth of deep learning in NLP during this period, such as the transformer revolution. Discussion related to advanced algorithm apart from deep neural network, that is embedding and machine learning, also showed significant and more consistent growth. This shows that beyond deep learning, the entire NLP ecosystem has continued to evolve. In addition, discussions around the foundational NLP task have shown steady growth, not only reflected community growth but also indicated that these concepts remained relevant to support more advanced algorithm.
4. Continuing to 2022 to 2023, the preprocessing category showed significant growth and overtook deep neural network in dominance. This more likely reflects that as advanced algorithm created greater interest and were widely adopted around the previous years, the community increasingly began focusing on optimizing preprocessing to improve the model performance. On the other hand, posts related to deep neural network gradually declined, which suggest a shift interest in NLP from model innovation to improving data quality.
5. In 2024, preprocessing still takes the lead, followed by deep neural network, indicating similar interest as in previous years. However, the total number of posts has significantly decreased, which indicates that Stack Overflow might not be used as much anymore for NLP related discussion. This reflect the increasing popularity of AI chatbots that developers can use to assist them in solving their programming challenges. Since AI chatbots provide immediate answers compared to Stack Overflow, this could explain the platform declining engagement.
6. Finally, in 2025, as the year has not ended yet, our argument might not be solid. However, we found that the proportion of category resemble 2024 and is more likely to decrease compared to the previous year. As AI chatbots knowledge grows and become stronger, the community might find more comfort working with AI to solve their challenges. Thus, our system may need to explore alternative data sources to capture more recent and relevant challenges in NLP.

# Conclusion

In summary, after the experiment on different combination of preprocess techniques and unsupervised model, we find out the best performance is produced by …., The reason could be

# References

1. Stack Exchange 2025, *Stack Exchange API v2.3*, Stack Exchange, viewed 13 April 2025, <https://api.stackexchange.com/docs>.
2. Axelborn, H & Berggren, J 2023, 'Topic modeling for customer insights: a comparative analysis of LDA and BERTopic in categorizing customer calls.', MA thesis, Umeå University, Sweden, <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1763637&dswid=-9433>.
3. Feng, Z, Guo, D, Tang, D, Duan, N., Feng, X, Gong, M, Shou, L, Qin, B, Liu, T, Jiang, D & Zhou, M 2020, 'Codebert: A pre-trained model for programming and natural languages.', *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp.1536-1547.
4. Grootendorst, M 2022, ‘BERTopic: Neural topic modeling with a class-based TF-IDF procedure’, *arXiv:2203.05794*, DOI:10.48550/arXiv.2203.05794.
5. Vithanage, Dinithi & Yu, Ping & Wang, Lei & Deng, Chao. (2024). Contextual Word Embedding for Biomedical Knowledge Extraction: a Rapid Review and Case Study. Journal of Healthcare Informatics Research. 8. 1-22. 10.1007/s41666-023-00157-y.
6. Novotný, V, Ayetiran, EF, Štefánik, M & Sojka, P 2020, ‘Text classification with word embedding regularization and soft similarity measure’, *arXiv:2003.05019*.
7. Prakash, A. 2023. *Understanding Cosine Similarity: A key concept in data science*. Medium. April 2025. https://medium.com/@arjunprakash027/understanding-cosine-similarity-a-key-concept-in-data-science-72a0fcc57599.
8. Joshi, T. 2021. *Silhouette Score*. Medium. April 2025. https://tushar-joshi-89.medium.com/silhouette-score-a9f7d8d78f29.
9. Yadav, S. 2023. *Understanding Intra-Cluster Distance, Inter-Cluster Distance, and Dun-Index: A Comprehensive Guide*. Medium. April 2025. https://medium.com/@Suraj\_Yadav/understanding-intra-cluster-distance-inter-cluster-distance-and-dun-index-a-comprehensive-guide-a8de726f5769.

Messy:

1. <https://stackoverflow.com/help/accepted-answer>
2. <https://stackoverflow.com/help/tagging>
3. <https://api.stackexchange.com/docs/advanced-search#order=desc&sort=activity&accepted=True&tagged=nlp&filter=default&site=stackoverflow&run=true>
4. <https://api.stackexchange.com/docs/paging>
5. <https://api.stackexchange.com/docs/answers-by-ids>
6. https://medium.com/@davidlfliang/intro-getting-started-with-text-embeddings-using-bert-9f8c3b98dee6
7. Jurafsky ebook