THE UNIVERSITY of ADELAIDE

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

This project develops an unsupervised text classification designed for Stack Overflow posts. The platform tag system classifies content based on technology content, making it important to have more detail tags that focused on purpose to improve effectivity in browsing solutions. We evaluate two main clustering strategies: a traditional rule-based model and a modern cosine similarity with embeddings. Furthermore, different text components and combination of data preprocessing techniques will be explored to identify the most effective approach. We find that the cosine similarity with SBERT embeddings achieves the best clustering quality when using title text with lemmatization and word extractions. This demonstrates the advantage of embedding-based methods in capturing semantic relationships and contextual meaning, with titles offering more meaningful vocabulary than body text, and word extraction enhancing classification by focusing attention on the key terms.

2. Introduction

Software developers frequently solve their challenges with the help of question and answer online forum. Stack Overflow is an online forum widely used by developers to solve their development challenges through asking questions, finding solutions, and learning from each other. The platform leverages tags to support developers search and browsing activities. The tags mainly aim at classifying posts based on technological content but not based on their purpose (Beyer et al. 2018, p. 1).

In building language models, there is a well-defined sequence of steps that need to be performed. The large volume of content in the platform makes it important to have more domain-focused categorisation that will improve efficiency for developers in finding their solutions. Therefore, the capability of categorising posts based on specific NLP tasks become necessary to help developers more effectively navigate content aligned with the specific steps of language model development.

The goal of this project is to build a system that consistently categorizes posts by NLP tasks, as well as evaluating different text components and preprocessing techniques to improve classification quality. We applied automated approaches to achieve more consistent organization by identifying and grouping similar posts based on their content and patterns.

Our approach compares a traditional rule-based method with a modern approach using cosine similarity with embeddings. We examine the impact of different clustering strategies and preprocessing, including the use of part-of-speech tagging and dependency parsing. The result contribute to more efficient browsing and offer valuable insights into optimizing text classification for programming related content.

3. 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 (Stack Overflow 2025). 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 (Stack Overflow 2025). 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 (Stack Overflow 2025). 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.

3.1. Process Workflow

In this section, we provide the workflow to illustrate the overall process of data collection, as shown below.

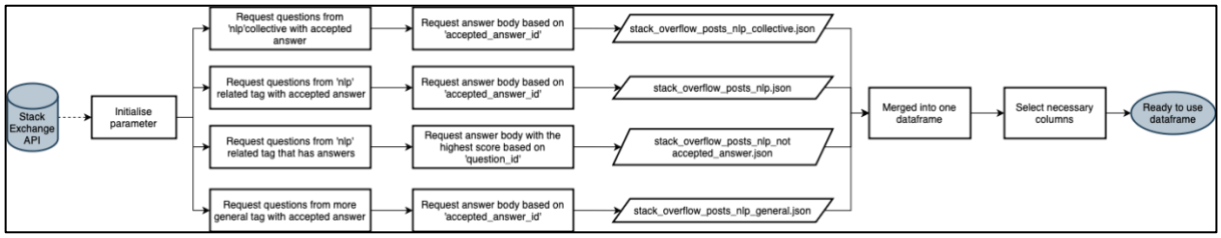


Figure 3.1 Workflow of retrieving NLP related posts from Stack Overflow.

3.2. 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`

	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	5920.00	6959711.00	108.00

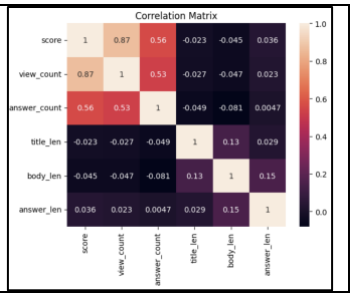


Figure 3.2 Statistics of score, view_count, and answer_count.

***Figure* 3.2** shows that the median of views, votes, and answers are relatively low, while the mean numbers are significantly higher. Following that, 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` has strong correlation 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.

2. Analysis by Tags

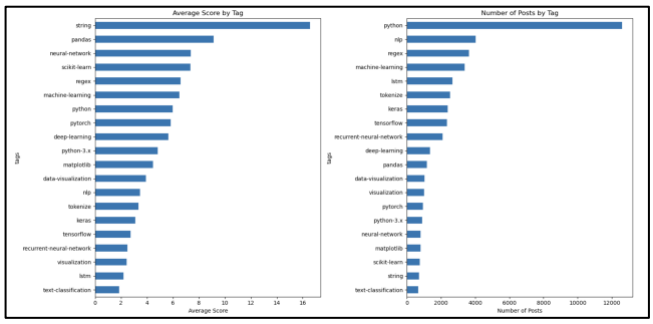


Figure 3.3 Plot analysis by tags

From ***Figure* 3.3**, 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 tags with 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.

4. Data Preprocessing

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.

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.

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 (Jurafsky 2023, p. 5). 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.

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 (Jurafsky 2023, 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.

5. Data Visualisation

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.

5.1. Word Cloud

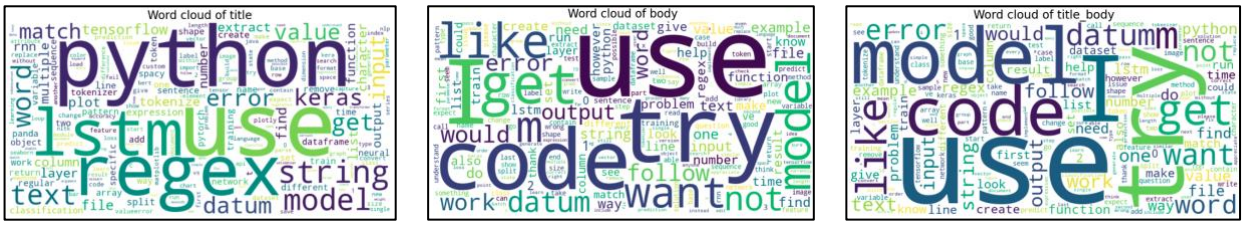


Figure 5.1 Word cloud of 'title','body', and 'title_body' column.

***Figure* 5.1**, 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

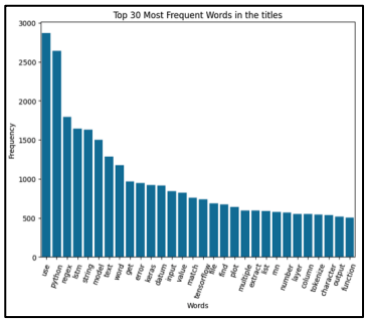


Figure 5.2 Word frequency plot of 'title' column.

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.

6. 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.

6.1. Methodology

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

6.1.1. 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).

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.

6.1.2. POS Tagging

Part-of-speech (POS) tagging assigns a grammatical label to each word in a sentence. 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. 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.

6.1.3. Dependency Parsing

Dependency parsing analyzes the grammatical structure of a sentence by identifying dependencies between words. Stack Overflow posts often contain irregular grammar or partial sentences, which we can filter using dependency parsing. We retain only words with the following dependency labels, which are ROOT, nsubj, dboj, compound, amod, as explained below.

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”.

6.1.4. Rule-based Method

The rule-based method works by using a set of predefined rules and keywords to classify text into different groups. 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. 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. However, it serve as a simple and interpretable method for categorisation that allow us to categorise text without training process.

6.1.5. 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 `all-MiniLM-L6-v2`, 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.

Furthermore, 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:

$$Cosine(\theta) = \frac{\overrightarrow{text} \cdot \overrightarrow{category}}{\|text\| * \|category\|} \quad (6.1)$$

where $\overrightarrow{text}$ and $\overrightarrow{category}$ represent the embedding vector of post text and category, respectively.

6.1.6. 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 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).

$$s(i) = \frac{b(i)-a(i)}{\max(a(i),b(i))} \quad (6.2)$$

$$S = \frac{1}{n} * \sum_{i=1}^{n} s(i) \quad (6.3)$$

where $a(i)$ is the average distance between point i and all other points in the same cluster and $b(i)$ is the smallest average distance between point i and all points in the nearest neighboring cluster.

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 (Yadav, S. 2023).

$$Intra = \frac{1}{K} \sum_{k=1}^{k} (\frac{1}{n_k} \sum_{x_i \epsilon C_k} \|x - \mu_k\|^2) \quad (6.4)$$

where X_i denotes Cluster C_k, μ_k denotes the centroid of C_k, and K denotes the number of clusters.

Inter-cluster Distance, 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).

$$Inter = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \|u_i - u_j\| \quad (6.5)$$

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

6.2. 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 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 represents as 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 section, 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:

Topic	Count	Name	Representation	Label
1	218	1_regex_regular_expression_python	['regex', 'regular', 'expression', 'python', 'match', 'pattern', 'findall', 'regex101', 'statement', 'regex101com']	preprocessing
2	214	2_bert_embedding_bertopic_bertmodel	['bert', 'embedding', 'bertopic', 'bertmodel', 'finetune', 'embed', 'pretraine', 'model', 'elmo', 'domain']	embedding
4	191	4_stringtokenizer_tokenize_delimiter_tokenizer	['stringtokenizer', 'tokenize', 'delimiter', 'tokenizer', 'java', 'php', 'jquery', 'token', 'tokeninput', 'tokenization']	tokenisation
11	121	11_huggingface_face_hug_transformer	['huggingface', 'face', 'hug', 'transformer', 'trainer', 'pretraine', 'finetune', 'pipeline', 'model', 'hugging']	deep neural network
22	86	22_forest_random_decision_tree	['forest', 'random', 'decision', 'tree', 'scikitlearn', 'scikit', 'classifier', 'leaf', 'probability', 'scikitlearn39s']	machine learning
51	54	51_tfidf_tfidfvectorizer_scikit_tokenpattern	['tfidf', 'tfidfvectorizer', 'scikit', 'tokenpattern', 'sklearn', 'importance', 'scikitlearn', 'spf', 'tfgetshape', 'ltresultgtpassltresultgt']	vectorisation
67	47	67_topic_lda_latent_modeling	['topic', 'lda', 'latent', 'modeling', 'dirichlet', 'modelling', 'gensim', 'allocation', 'short', 'topical']	topic modeling question answering
78	45	78_entity_recognition_ner_name	['entity', 'recognition', 'ner', 'name', 'scheme', 'bilou', 'spacy', 'multiword', 'tagging', 'surname']	syntatic annotation
86	41	86_lemmatization_stem_lemma_lemmatizer	['lemmatization', 'stem', 'lemma', 'lemmatizer', 'lemmatize', 'removal', 'noun', 'stopword', 'spacy', 'nlp']	normalisation
252	14	252_metric_machine_evaluation_evaluating	['metric', 'machine', 'evaluation', 'evaluating', 'yolov6', 'typefull', 'communicate', 'simpletransformer', 'learning', 'datadog']	evaluation
273	12	273_wordcloud_cloud_wordcloudwordcloud_dataviz	['wordcloud', 'cloud', 'wordcloudwordcloud', 'dataviz', 'truetype', 'colorfunc', 'colorhighlight', 'frequenice', 'rank', 'cropping']	visualisation

Figure 6.1 BERT-Topic result with manual labelling

***Figure* 6.1** show topics separation that organize associated vocabulary that correlates with functional aspects in NLP pipelines. This indicates that classifying our data based on task is an achievable approach.

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
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

Rule-based method

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

Title	regex	lowercasing	lemmatisation	stopword removal	regex & lowercasing	regex, lowercasing & lemmatisation	regex, lowercasing & stopword removal	all preprocessing
silhouette_score_without_other	0.060	0.067	0.057	0.060	0.067	0.064	0.067	0.064
intra_distance	0.842	0.840	0.848	0.839	0.841	0.848	0.835	0.843
inter_distance	0.524	0.526	0.516	0.530	0.526	0.513	0.528	0.516

Body	regex	lowercasing	lemmatisation	stopword removal	regex & lowercasing	regex, lowercasing & lemmatisation	regex, lowercasing & stopword removal	all preprocessing
silhouette_score_without_other	0.018	0.017	0.015	0.013	0.027	0.025	0.025	0.023
intra_distance	0.881	0.850	0.854	0.845	0.876	0.875	0.850	0.845
inter_distance	0.429	0.417	0.408	0.409	0.435	0.426	0.427	0.418

Title + Body	regex	lowercasing	lemmatisation	stopword removal	regex & lowercasing	regex, lowercasing & lemmatisation	regex, lowercasing & stopword removal	all preprocessing
silhouette_score_without_other	0.022	0.024	0.019	0.017	0.034	0.033	0.034	0.032
intra_distance	0.868	0.842	0.847	0.842	0.863	0.865	0.845	0.842
inter_distance	0.435	0.427	0.416	0.418	0.449	0.444	0.448	0.440

Figure 6.2 Result and evaluation metrics from the rule-based method.

From ***Figure* 6.2,** we can observe the following: (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`; and (3) the result using regex, lowecasing, and stopword removal in `title` has the best silhouette score with the lowest intra-cluster distance and highest inter-cluster distance.

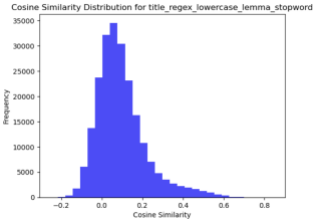
Since we obtain that the `title` yields better model performance compared to the other text column, it shows that `title` provides a concise summary of each post containing the key terms about the core concept of the post, making them highly discriminative. It contains very little amount of unnecessary 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 previous word cloud visualizations, which show that `title` displayed high frequency of semantically meaningful words.

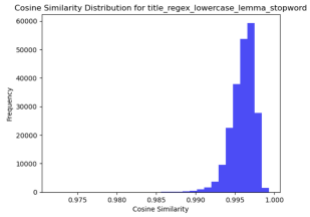
We also obtain that `title_body` performs better than `body` which further support our argument that `title` carries a strong signal that contains key terms related to the post's concept. However, when combined with `body`, the strong signal in `title` gets diluted resulting in performance that is lower 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.





Title	SBERT	CodeBERT
other	8633	30
deep_neural_network	3337	539
embedding	517	2677
evaluation	246	130
machine_learning	281	2003
normalisation	857	2025
preprocessing	2818	9904
syntatic_annotation	318	1546
tokenisation	1534	2
topic_modeling_question_answering	85	396
vectorisation	134	414
visualisation	1240	334
computation time in seconds	13.16	1541.96

Figure 6.3 Comparison of cosine similarity distribution and results between SBERT and CodeBERT embeddings.

From ***Figure* 6.3**, 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 have a large size of data, we will proceed with using SBERT as our embeddings algorithm. The evaluation metrics obtained from the cosine similarity with embeddings are presented below.

Title								
	regex	lowercasing	lemmatisation	stopword removal	regex & lowercasing	regex, lowercasing & lemmatisation	regex, lowercasing & stopword removal	all preprocessing
silhouette_score_without_other	0.090	0.091	0.093	0.090	0.090	0.092	0.089	0.091
intra_distance	0.811	0.810	0.812	0.809	0.811	0.812	0.807	0.809
inter_distance	0.619	0.621	0.619	0.614	0.619	0.617	0.610	0.610

Body								
	regex	lowercasing	lemmatisation	stopword removal	regex & lowercasing	regex, lowercasing & lemmatisation	regex, lowercasing & stopword removal	all preprocessing
silhouette_score_without_other	0.068	0.074	0.073	0.071	0.068	0.068	0.063	0.064
intra_distance	0.821	0.777	0.778	0.772	0.821	0.816	0.798	0.787
inter_distance	0.589	0.603	0.604	0.596	0.589	0.588	0.570	0.573

Title + Body								
	regex	lowercasing	lemmatisation	stopword removal	regex & lowercasing	regex, lowercasing & lemmatisation	regex, lowercasing & stopword removal	all preprocessing
silhouette_score_without_other	0.072	0.071	0.073	0.072	0.072	0.073	0.071	0.071
intra_distance	0.812	0.779	0.779	0.776	0.812	0.809	0.795	0.788
inter_distance	0.594	0.592	0.595	0.594	0.594	0.594	0.585	0.585

Figure 6.4 Evaluation metrics from the cosine similarity with SBERT embeddings.

From ***Figure* 6.4** , we can observe the following: (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 show only slight differences across various preprocessing techniques combination for `title`, with the highest silhouette score achieved using lemmatisation only; and (3) although the result for `body` show a significantly lower silhouette score, it achieves a lower intra-distance compared to `title`.

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. Also, comparing the two methods, we found that the cosine similarity with embedding clearly outperform the rule-based method. 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

Title		
	pos tags	pos tags & dependency parsing
silhouette_score_without_other	0.093	0.103
intra_distance	0.806	0.798
inter_distance	0.614	0.625

Figure 6.5 Evaluation metrics from the cosine similarity method with POS tags and dependency parsing for word extraction.

From ***Figure 6.5*** , we found that extracting words with relevant POS tags and dependency parsing could improve the performance of the model. 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. Also, it 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`).

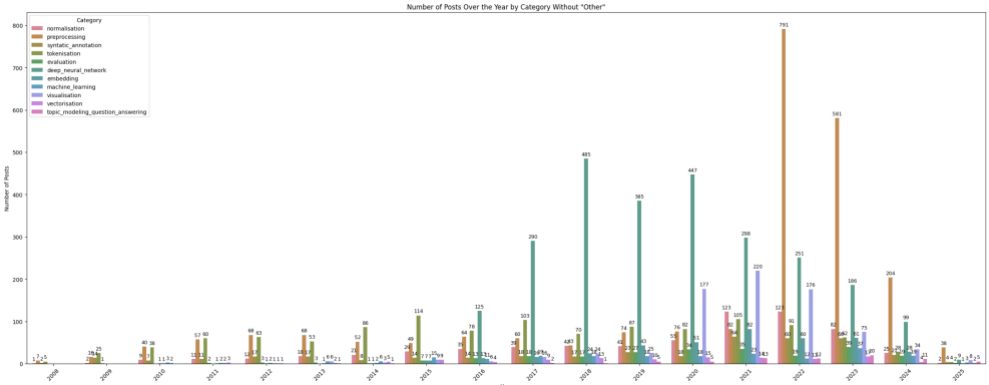
Therefore, we can conclude our experiment with taking conclusion that the best performing model that suitable for our data is **cosine similarity with SBERT embeddings, using lemmatisation, POS tags, and dependency parsing**.

6.3. 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 the best performing model for our data. To make the workflow easy to understand, we present below the overall process of the text classification system we implemented.

Figure 6.6 Final workflow

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



other	11005
deep_neural_network	2585
preprocessing	2370
tokenisation	1154
visualisation	784
normalisation	670
syntatic_annotation	393
embedding	393
evaluation	240
machine_learning	211
vectorisation	119
topic_modeling_question_answering	76

Figure 6.7 Number of posts by category over the year without 'other' label6.18

From ***Figure 6.7***, we can see the results organized into 12 distinct categories that do not overlap as we assign labels based on category with the highest cosine similarity score. These categories are particularly meaningful as they align with the sequence steps in NLP development that increase efficiency for developers to find solutions compared to technology-focused tags alone. Our approach of using title text embeddings provide efficiency by requiring fewer preprocessing steps for concise text while still effectively capturing semantic relationship between posts. In addition, we recognized the large number of posts in the 'other' category that suggest potential further exploration to identify additional meaningful categories. However, this also reflect the presence of many posts with tags not directly related to NLP, as shown in ***Section 3.2.***, which causes them to fall outside the defined NLP-related category.

This results also help provide information about the trends as shown in the bar plot. Starting with 2008 to 2015, the community activity was minimal with focused on foundational NLP tasks, which are normalization, preprocessing, syntactic annotation, and tokenization. The post volume steadily increased over time, indicating community growth in Stack Overflow. Between 2016 to 2021, deep neural network showed a significant increase and overtook in dominance that aligns with the major advances in NLP such as the transformer revolution. In addition, the foundational NLP task have shown steady growth, showing

their continued relevance to support more advanced algorithm.

Next, from 2022 to 2023, the preprocessing category overtook deep neural network in dominance that suggest a shift in focus from model innovation to optimizing model performance. In 2024, preprocessing remained dominant, followed by deep neural network. However, the post volume significantly decreased likely due to the increasing popularity of AI chatbots that provides faster answers. This declining trend indicates that our system may need to explore alternative data sources to capture more recent and relevant challenges in NLP.

7. Conclusion

In this project, we have built a text classification system using an unsupervised approach that was evaluated using different combination of preprocess techniques and text columns from StackOverflow posts. In summary, we found that out of all methods evaluated, the cosine similarity method using SBERT embeddings performed best, with the highest silhouette score. The method works best using the title of the post with lemmatization and was further improved by incorporating POS tags and dependency parsing for word extraction. Our result provides insights that embedding-based methods outperform rule-based method by capturing semantic relationship and contextual meaning. Titles provided more meaningful words than body text, and preprocessing had a greater impact on embeddings when applied to longer text. Furthermore, extracting relevant terms using POS tags and dependency parsing enhance classification by focusing on meaningful vocabulary. These findings demonstrate the effectiveness of combining semantic embeddings with preprocessing techniques for classifying domain-specific text, particularly in programming related content. The development of our system not only saves time and effort but also improves the accuracy and quality of the categorization.

8. References

[1] 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>.

[2] Beyer, S, Macho, C, Pinzger, M, & Di Penta, M 2018, 'Automatically classifying posts into question categories on stack overflow' In *Proceedings of the 26th Conference on Program Comprehension,* pp. 211-221.

[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] Jurafsky, D, & Martin, JH 2023, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.*

[5] Joshi, T 2021, *Silhouette Score*, Medium, viewed in 15 April 2025, <https://tushar-joshi-89.medium.com/silhouette-score-a9f7d8d78f29>.

[6] Novotný, V, Ayetiran, EF, Štefánik, M & Sojka, P 2020, 'Text classification with word embedding regularization and soft similarity measure', *arXiv:2003.05019*.

[7] Stack Exchange 2025, *Stack Exchange API v2.3*, Stack Exchange, viewed 13 April 2025, <https://api.stackexchange.com/docs>.

[8] Stack Overflow 2025, *What does it mean when an answer is "accepted"?*, Stack Overflow, viewed 12 April 2025, <https://stackoverflow.com/help/accepted-answer>.

[9] Stack Overflow 2025, *What are tags, and how should I use them?*, Stack Overflow, viewed 12 April 2025, <https://stackoverflow.com/help/tagging>.

[10] Yadav, S 2023, *Understanding Intra-Cluster Distance, Inter-Cluster Distance, and Dun-Index: A Comprehensive Guide*, Medium, viewed 15 April 2025, <https://medium.com/@Suraj_Yadav/understanding-intra-cluster-distance-inter-cluster-distance-and-dun-index-a-comprehensive-guide-a8de726f5769>.

Group Contribution

Desy Natasha Haloho:

1. Defined the category and key words for each category by the key words produced by BERT Topic for the first half of the topics result.
2. Using BERT embedding model to embed each self-defined category ‘s key words and text contents from posts data, then use cosine similarity to label the category.

Yu-Chieh Liao:

1. Defined the category and key words for each category by the key words produced by BERT Topic for the second hald of the topics result.
2. Use Rule-based model to define the category based on the frequency of specific key words occurring in each content of post data.

GitHub URL

https://github.com/desy-natasha/nlp_text_classification.git

https://github.com/Rainman4301/NLP_Practice.git