Master Thesis

MSc. Data Science and Marketing Analytics

Enhancing Tag Recommendation in Software Q&A Forums: A Comparative Study of LDA and Doc2Vec Approaches

Author: Nicole Liao (Chia-Yi Liao)

Student number: 575586

Supervisor: Prof. Dr. (Bas). ACD Donkers

Second assessor: Dr. (Carlo) C Cavicchia

Date final version: April 14, 2023

Table of Content

1 Introduction

Over the recent years, software Question-Answering (Q&A) sites have become popular and reliable avenues by software developers to ask questions, communicating new techniques, share knowledge and receive answers to technical questions. Tagging is one of the common problems and has been attracting much attention on software Question-Answering (Q&A) sites. Users on software questions forum often ask complex questions related to programming. Tags are popular as they describe the most important feature of the software objects. However, when the website is not well-tagged, it can be difficult for developers and managers to quickly and easily gain an insight into a certain theme. Addressing these issues can lead to improved users' satisfaction. Tag recommendation can help developers to better tag new content by effectively reusing existing tags on software Question-Answering (Q&A) sites.

The tag recommendation approach presented in this research consists of several stages. The overall architecture of tag recommendation are text pre-processing, feature extraction, and tag classification using a multi-label classifier. We propose two methods for feature extraction: LDA and Doc2Vec. Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is a generative probabilistic model of a corpus to extract the hidden structure and topics, and is considered the standard topic modeling approach widely used in NLP tasks. Doc2Vec (Le & Mikolov, 2014a) is a state of the art neural-network-based approach for generating fixed-length vector representations of a given document. These models have been shown to be effective in improving the accuracy of tag recommendations by taking into account the content of the documents. By capturing the underlying topics and semantic meaning of the documents, LDA and Doc2vec are able to provide more relevant and specific tag recommendations. LDA has been a popular method for solving the tagging problem in software-related domains, while there are limited research comparing LDA and Doc2vec for tag recommendation model on Software Q&A site. This leads to the following

research questions: How can a tag recommendation model be built using topic modeling and word embedding approaches to enhance tagging quality?

Sub Question 1: How can feature be extracted to automatically recommend a set of tags from Q&A forum?
Sub Question 2: How to find interpretable insights in this analysis?
Sub Question 3: Which approach can best used for tag recommendation? Can some conclusion be reached based on the results from different approaches?

In this research, we aim to investigate and compare LDA and Doc2vec methods to enhance the performance of tag recommendations for Software Q&A forums. Our study will focus on evaluating the performance of these models in providing accurate and relevant tag recommendations for new questions in a Software Q&A forum setting. By studying the strengths and weaknesses of different approaches and investigating how these approaches can be improved in terms of performance, we aim to assist in choosing appropriate methods specifically for software Q&A forum types of data.

Throughout this paper, these questions will be answered. This research will begin with exploring the current literatures about tag recommendation on software Q&A forum (Chapter 2). Following the literature review, an overview of the dataset is provided (Chapter 3). Chapter 4 contains the technical aspects of building the tag recommendation system, presenting the all the stages for tag recommendation. Subsequently, Chapter 5 presents the result of model's performance, the effects of various pre-processing methods on performance, and a discussion of the models' interpretations for future improvement of the models. Finally, Chapter 6 concludes the research, offering suggestions for further studies and drawing overall conclusions.

2 Literature Review

In this section, we begin by reviewing previous work on the importance of tags in software developer Q&A websites and the challenges developers encounter when tagging their questions. Subsequently, we examine the literature on the evolution of tag recommendation models utilizing various techniques. Among these techniques, we review the literature on two popular approaches for building content-based recommendation systems, namely topic modeling and document embedding.

2.1 Background and motivation

The rapid growth of software technologies has led to a surge in the use of Q&A websites by software developers. These online platforms provide developers with the ability to search, share, and learn from the experiences of others, as well as to obtain solutions, troubleshoot errors, and access open-source projects. As software complexity continues to grow, developers increasingly rely on these Q&A communities to seek expert support. When faced with technical queries, developers frequently turn to forums within these communities due to their prompt response times and wider visibility. (Squire, 2015) With continuously increasing information and users on online Q&A communities poses a challenge for users in quickly and efficiently obtaining the information they need on a given topic. Software information sites rely users to classify contents their contents with tags, for example, the programming languages or the software objects. Tags are relevant keywords associated with a piece of information on web pages (Chirita et al., 2007), which aid in efficient information retrieval and management. Tags are very useful for the Community Question-Answering forum because tags can help categorizing, browsing, and managing the questions. By facilitating the detection of similar queries, tags enable users and community experts to conduct more efficient searches and identify questions that align with their areas of expertise. Tags play an important role in software Q&A sites as it can bridge the gap

between social and technical aspects and it improves team-based software development practices (Treude & Storey, 2009).

Figure 1. An example question in Stack Overflow and its tags

Tag recommendation suggests a group of tags for a question post, making it easier for a user to manually tag the question. The development of such models for software Question-Answering (Q&A) sites is motivated by three key factors. First, Inconsistency: tags are decided by users or developers themselves and tags are not restricted to a certain vocabulary guideline. Tags can be inconsistent and vary in frequency of use due to users' personal terminology as well as due to the different purposes of tags (Golder & Huberman, 2006a). Words that convey the same meaning can differ in various ways, including the presence or absence of spaces, capitalization, abbreviations versus their full spellings, and the presence of hyphens. These differences pose challenges for software developers attempting to search for existing tags, leading them to resort to their own word choices (Liu et al., 2018). Inconsistency among the terms used in tagging can make it very difficult to make sure all the relevant subjects have been found. Synonymy or words that have the same or similar meanings make tags tend to be noisy and sparse (Golder & Huberman, 2006b). For example, (Xia et al., 2013) noticed tags "zombie" and "zombies" both describe the zombie process in Unix. Second, labor-intensive: tagging manually is also time-consuming and labor-intensive. Choosing suitable tags may not be an easy task since developers might just choose the terms that were mentioned in the question, instead of the most relevant tags that represent the most important features. Sometime low-quality of tags are selected and they may have a negative effect on the organization of the website. Lastly, organized and avoid duplicates: tags have been shown to increase the capabilities of searching, organizing and managing content. If

recommended tags are highly relevant, it will be easier for users to search for the questions they are looking for from tags. Additionally, the use of tags helps to prevent the occurrence of duplicate content, thereby streamlining the organization of information within the Q&A community.

2.2 Tag recommendation

Automatic tag recommendation provides a viable solution for these challenges by suggesting the existing high-quality tags. Many studies have been done on tag recommendation for software Q&A sites. (He et al., 2022; Wang, S., Lo, Vasilescu, & Serebrenik, 2018a; Xia et al., 2013). Existing tag recommendation techniques can be broadly categorized into two different categories, namely user-centered approaches (collaborative filtering methods) and document-centered approaches (content-based methods). User-centered approaches, rely on the past interactions and preferences of users to recommend tags for a given document. The goal is to recommend tags to a user based on the tagging behaviors of similar users or user groups, as well as their historical interactions with items, such as ratings or clicks. User-centered approaches are less effective than document-centered approaches. One of the reasons is that user-centered approaches can suffer from the "cold start" problem, which refers to the difficulty in recommending tags for a new user or a new document that has not yet been interacted with by any user. With content-based approaches, tags can be treated as class labels for document, or summarizations of documents, thus document-centered approaches are more flexible to apply any sophisticated machine learning algorithms. Additionally, content-based approaches are more robust because of the rich information contained in the documents (Song et al., 2011). Content-based approaches have been widely used in the field of tag recommendation for Q&A forums. The main aim of these approaches is to analyze the content of the questions and answers and find associations between the themes present in the documents and their tags. These approaches are particularly useful in cold-start settings where either the user or the content does not have any tagging history. (e.g., new users or new documents) (Kataria & Agarwal, 2015). Research (Xia et al., 2013) proposed

TagCombine, an automatic tag recommendation analyzing objects in software information sites. The method contains three different components: multi-label ranking component by adapting multi-label learning one-versus-rest Naive Bayes classifiers, similarity-based ranking component by applying TD-IDF, and tag-term based ranking component.

In the field of information retrieval, content-based tag recommendation models have been widely used to provide recommendations for documents. However, despite their benefits, these models also have limitations such as the inability to capture the context of the document and the relevance of tags to the document. LDA and Doc2vec are popular content-based tag recommendation models that have been developed to overcome these limitations. LDA is a generative probabilistic model that uses latent topics to represent the content of the document and the tags. Doc2vec, on the other hand, is a deep learning model that uses neural networks to capture the semantic meaning of documents. These models have been shown to be effective in improving the accuracy of recommendation by taking into account the content of the documents. By capturing the underlying topics and semantic meaning of the documents, LDA and Doc2vec are able to provide more relevant and specific tag recommendations.

2.3 Topic modeling

The field of information retrieval has become increasingly vital due to rapidly increasing sources of information on the internet. Topic modeling stands out as one of the most effective techniques for text summarization and information retrieval （Chauhan & Shah, 2021). It is a powerful tool for identifying the key topics or themes that are discussed within a given corpus of text, and has been applied to a wide range of applications, including sentiment analysis (Jeong et al., 2019; Rao, 2015) and recommender systems (Ansari et al., 2018; Zhao et al., 2016; Zoghbi et al., 2016). Topic modeling infers the underlying topics in a corpus of text documents by analyzing the distribution of words across documents. The key idea is that words that appear together

frequently in documents are likely to be related, and can be used to infer the latent topics present in the corpus. Once the topics have been identified, each document can be represented as a probability distribution over the topics. This means that a given document may be associated with multiple topics, each of the topic represents a different aspect or theme of the document. These topics can be used to identify semantically similar documents and to explore the relationships between the topics and the documents in the corpus.

The most widely used algorithm for topic modeling is Latent Dirichlet Allocation (LDA), due to its highest performance when dealing with long documents and interpreting identified topics (Chiru et al., 2014). LDA has been frequently implemented to extract topics and has shown to be effective and popular method for solving software engineering problems (Asuncion et al., 2010; Somasundaram & Murphy, 2012; Wang, T. et al., 2014). Tag recommendation using LDA is also seen in software related website. To overcome cold start problem, (Krestel et al., 2009) implemented LDA approach which provides a probability value for each topic and tag. The probabilities assigned by LDA to each tag for each latent topic is used to generate a probability value for each tag. To control the number of recommended tags, the system recommends tags with a threshold so the system can recommend only those tags that meet a certain level of probability, allowing for adjusting the balance between recall and precision. This approach of the paper achieves significantly better precision and recall and more specific for a particular resource compared to an approach using association rules. In (Ramage et al., 2009), the authors note that traditional topic models like Latent Dirichlet Allocation (LDA) do not effectively handle labeled data, therefore they introduce Labeled LDA (L-LDA), a supervised topic model which constrains the topics of a document to the tags attached to that resource. Its tags are predefined and are trained together with a set of documents, so that it will calculate the probability distribution of topics. (Wang, S., Lo, Vasilescu, & Serebrenik, 2018b) proposed ENTAGREC++, a method for recommending tags to software objects that are untagged. It builds on the previously proposed

ENTAGREC approach, which uses a probabilistic model to assign tags to software objects based on the words that appear in them. However, the basic approach suffers from two major problems: the presence of unrelated words and data sparsity. To address the issues of unrelated words, it extended Labeled LDA model by removing unrelated words using Part-Of-Speech (POS) Tagger to identify the unrelated words. Only nouns and noun phrases are retained in this study.

2.4 Document embedding

In recent years, learning embeddings that learn distributed representation of words and documents from large text has become popular for many natural languages processing tasks, including document tagging. Document embedding has been a very active research area in various applications, such as sentiment analysis (Bilgin & Şentürk, 2017; Thongtan & Phienthrakul, 2019) and identify abnormal comments (Chang et al., 2018). However, there has been limited research on using document embeddings for multi-label learning.

Doc2Vec is sentence embedding unsupervised text learning model introduced by Google (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). It is an extension to Word2Vec model. In addition to learning semantic representations, the paragraph vectors also contribute to the prediction task alongside the word vectors. Doc2vec performs really well in the case of representing longer documents (Lau & Baldwin, 2016a). Tag recommendation model build in (Chen et al., 2017) was the first embedding based multi-label learning approach to NLP tasks. Its DocTag2Vec extends Doc2Vec by adding tag embedding. (Liu et al., 2018) propose a tag recommendation model for software Q&A site. The model's architecture is similar to continuous bag of words model (CBOW) and it creates a look up table that only contain words that appear in the software information site.

Recent research use Doc2Vec on other recommendation method has achieved good result. (ZhengWei et al., 2022) use Doc2vec to represent bibliographic text. It is a new journal

recommendation approach combing with XGBoost algorithm to classify bibliographic information after extracting text representation from Doc2Vec. (Nandi et al., 2018) builds a content-based news recommendation model using Doc2vec and compare the model with LDA and LSA.

Doc2vec and LDA are two popular approaches for building content-based recommendation system, ranging from extending the approaches by adding more components to applying the model on larger dataset. To the best of our knowledge, there is limited research on comparing both methods for tag recommendation models in the context of software Q&A forums. The expected outcomes of this study include a deeper understanding of the effectiveness of Doc2Vec and LDA for tag recommendation models in software Q&A forums. We will use pre-processed methods and evaluation metrics that have not been studied in previous research and examine the interpretability of the models. The study will identify the strengths and limitations of each approach and provide recommendations for improving the performance of these models.

3 Data sets

To address the research questions, we compare different methods on one recent dataset. The dataset consists of manually tagged questions from Stack Overflow, a leading community for developers and technologists to ask and answer programming-related questions. The forum allows app developers and experts to submit questions and answers, with multiple tags tagged to each questions.

The dataset from Kaggle https://www.kaggle.com/datasets/imoore/60k-stack-overflow-questions-with-quality-rate contains 60,000 Stack Overflow questions from 2016-2020. Users submitting questions label each question with one or more tags, and platform moderators can adjust these tags if necessary. The dataset includes the question ID, question title, question content, tags associated with each question, creation date, and quality score of the questions. Tag, question title and question itself are used in this research. The question title of a post summarizes the question, while the question body offers additional details, including the question description and code snippets. Tag, question title and question body are text data, but after pre-processing, they will be converted to numeric values, as the models can only accept numerical inputs.

3.1 Tags

In total there are 219880 tags and 10704 unique tags. The three most popular tags are 'javascript', 'python', and 'java'. The following figure (Figure 2.) shows distribution of the 100 most frequent tags in this dataset. 100 most frequent tags account for 39.30% of total amount of tags.

From Figure 2 we can see that the tags are imbalanced, 10 most popular tags account for more than 20% of total numbers of tags. In Table 1 shows the percentage of top 10 tags occurs in total number of tags, and the percentage of top 10 tags occur within top 100 tags, and Table 2. Shows the accumulated percentage of top 10 tags. The majority of the most frequent tags are programming language.

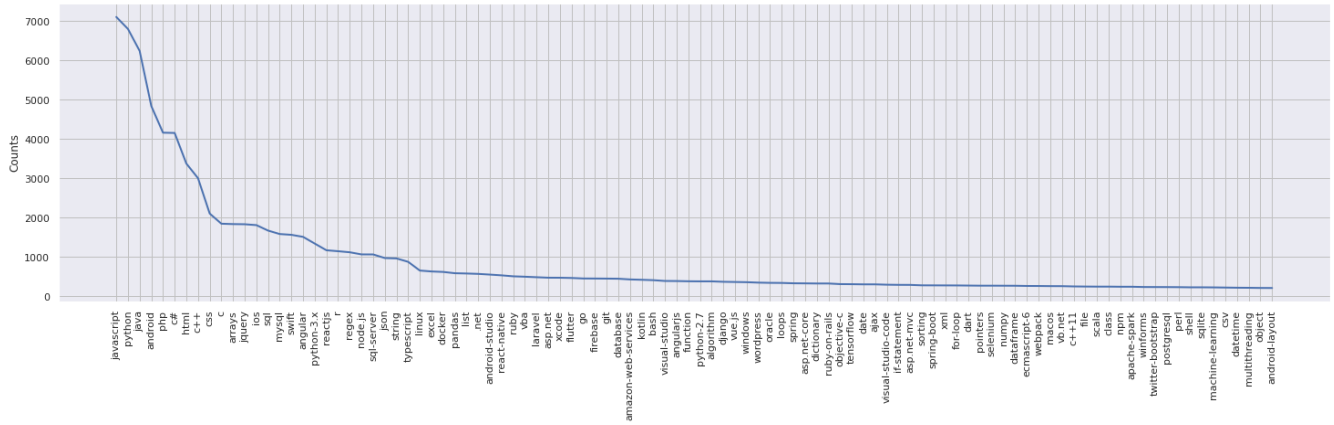


Figure 2. Distribution of 100 most frequent tags

Tags	javascript	python	java	android	php	c#	html	c++	css	c
of total tags	3.23%	3.09%	2.84%	2.20%	1.89%	1.89%	1.54%	1.37%	0.96%	0.84%
of top 100 tags	8.22%	7.87%	7.22%	5.60%	4.82%	4.81%	3.91%	3.48%	2.43%	2.14%

Table 1. The occurrence of top 10 tags

Tags	javascript	python	java	android	php	c#	html	c++	css	c
of total tags	3.23%	6.32%	9.16%	11.36%	13.25%	15.14%	16.68%	18.05%	19.01%	19.85%
of top 100 tags	8.22%	16.09%	23.31%	28.91%	33.73%	38.54%	42.45%	45.93%	48.36%	50.50%

Table 2. The accumulated occurrence of top 10 tags

3.2 Questions

The majority of the word length of each question is shorter than 200 words. The average word length is 200 and median is 184.54. (Figure 3)

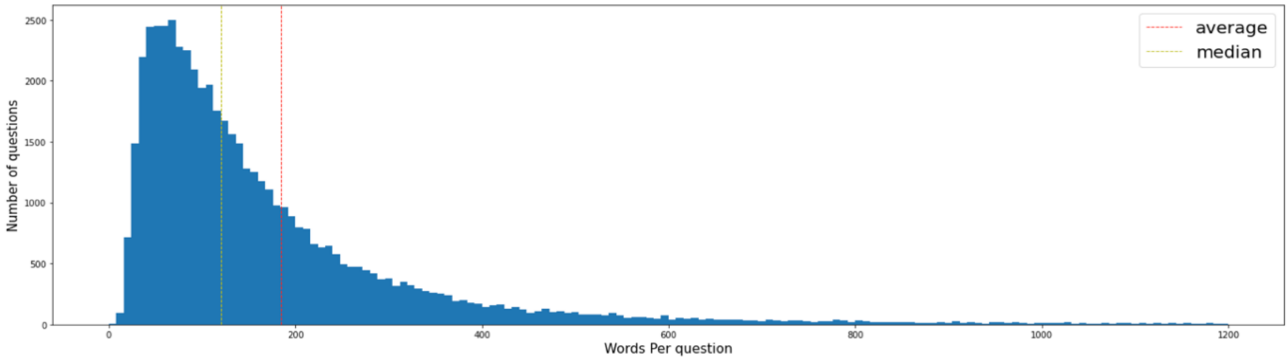


Figure 3. Distribution of Question Lengths

There are few outliers clearly seen in boxplot (Figure 4) These outliers are removed since very short documents may not contain enough information to be useful for training, and very long documents may contain too much noise or irrelevant information. Therefore, 72 questions are removed.

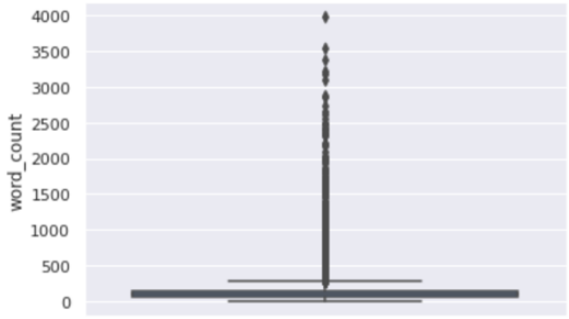


Figure 4. Boxplot of Question Length

4 Methodology

In this section, the two tag recommendation models used in this research will be discussed in more detailed. Besides introducing the models and their algorithms, three main stages for both tag recommendation model will also be explained. The three stages are text-preprocessing, feature extraction and multi-label classification. Once the features were extracted from two models, we trained two multi-label classifiers from the extracted features then assign tags to the corresponding questions. In the last part of this section contains evaluation metrics that are used to evaluate and compare these models.

To evaluate the two models, the dataset is split into a training set (70%) and a testing set (30%). The tags that occur more than 250 times are selected because the goal of the tag recommendation model is to recommend general tag sets for new posts (questions).

4.1 Pre-processing

Before comparing methods, pre-processing need to be performed on both the question body and tags so that the texts are in desirable format. (Lau & Baldwin, 2016b) use both the question title and body as document content for evaluating the effectiveness of Doc2Vec on forum question duplication. Additionally, result from (He et al., 2022) demonstrated that question title is the most significant components for tag recommendation in Stack overflow posts. Therefore, the input text combines both title and questions body content.

Few tags belonging to the same category are combined to avoid inconsistency problem mentioned by (Golder & Huberman, 2006b). pyth*on-2.7, python-3.x* are changed and combined into *python*. *c++11* is changed into c++.

Text preprocessing takes as raw text input and returns cleansed tokens. The approach used for pre-processing a document for topic modeling includes the following steps:

1. Convert all letters into lowercase
2. Remove HTML formatting using Beautiful Soup library
3. Remove special characters, punctuations, and undesirable marks like “@#$%^&*(){}:”<>?/.,’;][“ , unless they are part of tags (e.g. c#, c++)
4. Tokenize words
5. Perform lemmatization
6. Remove stop words

First, text cleaning, also called noise removal is performed. It includes remove all of the punctuation and converting uppercase letters to lowercase. To remove punctuation, regular expression *re.compile* is used to identify the targeted regular expression pattern, allowing for the removal of the characters that are not whitespace. However, punctuations that are not related to important tags (such as c++, c#) are remained in the text body. In an HTML web page, tags such as and define paragraphs. The Beautiful Soup library can remove these tags generated from HTML files. Some contracted forms are converted during the data cleaning process; for example, ‘what’s’ is converted into ‘what is’ and ‘I’ve’ is converted into ‘I have’, and ‘can’t is converted into ‘cannot’.

Next, text normalization is performed, which involves converting the text to a more convenient, standard form. Text normalization includes tokenizing (segmenting) words and normalizing word formats (Stemming and lemmatization) (Jurasfky & Martin, 2000). Tokenization divides a large quantity of text into smaller parts called tokens. Stemming and lemmatization are common pre-processing steps that generate the root form of the family word

into a single word. However, lemmatization process is more elaborate because replaces actual words with dictionary words, while stemming only changes words into root forms. As a result, lemmatization requires more resources (Jivani, 2011). Lemmatization determines words that share the same root; for example, the words is, am, and are share the same lemma be; the word function and functions both share the same lemma function.

After lemmatizing words, we use the list of stop words (e.g., a, an, the, etc.) provided by the NLTK library. Removing stop words should improve the model because they represent low-level information in a document's content. By applying these pre-processing steps, the text data is prepared for further analysis using the chosen methods.

4.2 LDA

Topic modeling is an unsupervised machine learning technique used to discover latent themes and generate hidden topics from unstructured data. The output of this process consists of topic vectors that can be used as feature vectors in supervised classification models to identify "similar" documents without prior knowledge regarding their content. This approach differs from traditional methods of manually identifying groups of interest by annotating a subset of documents. Although topic models cannot replace human interpretation of text, they do offer a viable means of making educated guesses about how words cohere into different latent themes by identifying patterns in their co-occurrence within documents. However, one of the limitations of topic modeling is its underlying assumption that each word has a singular meaning, which may not always be accurate.

One popular technique for topic modeling is Latent Dirichlet Allocation (LDA), which is a probabilistic soft-clustering method where clusters may overlap, it is different from hard-clustering methods such as Hierarchical Clustering and k-means, where each element belongs to

one cluster. Given the software related questions do not fit only into one single category, LDA is a preferred method for this study. The term “Latent” in LDA refers to the hidden topics present in the data, while “Dirichlet” represents a form of probability distribution that helps to explain the similarity of data by grouping features. “Dirichlet” is a "distribution of distributions" as it represents the probabilities of possible outcomes of another distribution. In the context of LDA, “Dirichlet” indicates the probability distribution of topics in documents and the probability distribution of words in each topic.

Latent Dirichlet Allocation (LDA) is a probabilistic generative model that allows us to discover the underlying topics in a corpus of text documents. LDA assumes that each document in the corpus is a mixture of K latent topics, where K is a user-defined parameter. The model also assumes that each word in a document is generated from one of the K topics, and the probability of a word belonging to a topic is determined by the topic-word distribution and the topic proportions in the document. LDA is guided by two fundamental principles: each document represents a combination of topics, and every topic consists of a combination of words. Through LDA modeling, two sets of probability distributions are produced (Figure 5):

- The set of probability distributions of topics for each document
- The set of probability distributions of words for each topic

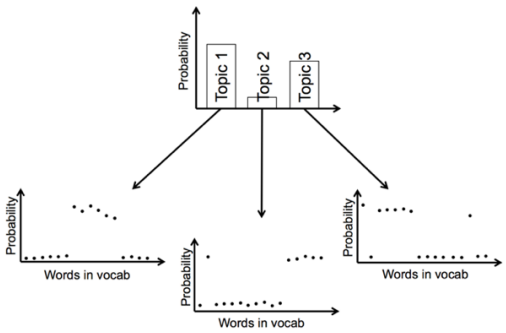


Figure 5. An example of LDA probability distributions (Sam Tazzyman, 2018)

The learning process in LDA starts by randomly assigning each word in the corpus to one of the K topics. Subsequently, for each word in each document, the model calculates the probability of that word belonging to each topic, based on the current topic distribution for the document and the topic-word distributions learned from the corpus. These probabilities are used to assign the word to one of the topics, by sampling from the distribution over topics. The probability of words belonging to a topic is $P(w_i|d)$, the probability of the i_{th} word for a given document d. It is calculated by multiplying two following probabilities.

- $P(w_i \mid z_i = j)$ represents the probability of w_i inside the topic j
- $P(z_i = j \mid d)$ represents the probability of the word stemming from topic j given the document

The latent topic z_i is a hidden variable that represents the assigned topic for the i_{th} word in the document. z_i is inferred probabilistically based on the learned topic distributions for each document. The probability of word w_i belonging to any topic is the sum of these probabilities over all possible topics. $P(w_i|d)$ can be formalized as follows:

$$P(w_i|d) = \sum_{j=1}^{z} P(w_i \mid z_i = j)\, P(z_i = j \mid d)$$

This process is repeated for all words in the document, updating the topic distribution for the document based on the newly assigned topic for each word. The process then moves on to the next document and repeats until we obtain a set of topic distributions that best explain the observed word frequencies in the corpus.

In Latent Dirichlet Allocation (LDA), the parameters alpha and beta are hyperparameters that control the sparsity of the topic distributions for the documents and the word distributions

for the topics, respectively. These two hyperparameters are experimented and then adjusted to fit this data. The alpha parameter influences the topic distribution for each document by controlling the degree to which topics are expected to be present in a document. A smaller alpha value results in sparser topic distributions, where each document is expected to have fewer dominant topics. A larger alpha value results in more uniform topic distributions across all documents. The beta parameter influences the word distribution for each topic by controlling the degree to which words are expected to be present in a topic. A smaller beta value results in sparser word distributions, where each topic is expected to have fewer dominant words. A larger beta value results in more uniform word distributions across all topics.

Perplexity is an intrinsic evaluation that measures of how well a probability model predicts a sample. It indicates how well the model describes a set of documents; To compute perplexity of LDA model, a test corpus and a trained LDA model is needed, which output is topic-word probability and document-topic probability. The probability of each word in each test document given the topic assignments of all the words is calculated, and then the result takes the exponential of the negative average log-likelihood of the sample. The formula is shown as:

$$\text{Perplexity of set of documents} = \exp \frac{-\log\left(\Pr[all\ words\ in\ docs]\right)}{Total\ numbers\ of\ words}$$

4.2.1 Random Forest to determine the most informative words using Part-of-speech tagging

In the process of text pre-processing, although stop words have been removed, many non-stop words remain unrelated to software object tags and thus need to be eliminated. Part-of-speech (POS) tagging is a technique that identifies and labels words as specific parts of speech, such as nouns, verbs, adverbs, or adjectives, based on their context and definition. This enables a deeper understanding of a sentence's grammatical structure, resulting in more accurate and

effective language processing. In a typical NLP pipeline, tagging is the second step after tokenization. To determine the types of words that are informative for tag recommendation in this specific type of task, texts are labeled using part-of-speech tagging to classify words into categories such as nouns, verbs, adverbs, or adjectives, among others. (Wang, S. et al., 2018b) extended the L-LDA model by employing a POS Tagger to remove unrelated words, retaining only the nouns. In our study, we utilize the Random Forest model to analyze the most important word types. Once the POS tags associated with each word are extracted, the importance plot generated by the Random Forest model reveals the most informative word types.

4.2.2 Convert clean text into numerical representation

In order to run the LDA model, the next step is to convert the cleaned text into a numerical representation. TF-IDF helps to penalize too frequent words and provide better features space. *Countvectorizer* or *TfidfVectorizer* can be used to transform the Document Term Matrix (DTM) into numerical arrays. According to (Blei & Lafferty, 2006) the Tf-IDF score can be useful for LDA. *TfidfVectorizer* from Scikit-learn converts documents into a matrix of TF-IDF features. TD-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure that evaluates how relevant a word is to a document within a collection of documents. These documents can be sentences, dialogues or even long texts. Term frequency refers to the occurrence of a given word divided by the total number of words appearing in the document, in other words, how often a word occurs in a document. However, calculating term frequency alone is insufficient to solve our problem, since some words may appear several times without providing meaningful insights, such as the words "the" and "and". To identify special words in each document, Inverse Document Frequency is calculated. This involves taking the log of the number of documents divided by the number of documents with the term (t). For example, the word “the” would appear in all

documents, and then value of the denominator will be the same as the number of documents. With the log in front of the value, which is log of 1, it will turn the value to 0.

Lastly, by multiplying Term Frequency and the Inverse Document Frequency, we obtain the TD-IDF score. The higher the score, the more relevant the word is in that document. For the mathematical notation of TD-IDF, some notations are introduced. The corpus (D) represents a collection of documents, d represents a document, t represents a term (word), and N is the count of corpus.

$$tf - idf\ (t, d, D) = tf(t, d) * idf(t, D) = tf(t, d) * log \frac{\mathrm{N}}{\#(\mathrm{d} \in \mathrm{D}: \mathrm{t} \in \mathrm{d})}$$

Within Scikit-learn's TfidfVectorizer, tokens that have less than 2 characters are removed to exclude single characters which may not be informative enough to be useful in the analysis.

4.2.3 Feature extraction from topic distribution

Treating individual words as features yields a vast number of features set (Joachims, 1999). One way to reduce this feature set is to use topic distributions for dimensionality reduction. After processing the training set of documents and extract say N topics, we get the topic distribution of N topics for all the labeled documents. We use these topic distributions to recommend tags for a given text by identifying the most probable tags associated with those topics. Topic distribution is a measure of the probability of each topic present in a document, representing the likelihood that each topic generated the words in the document. After fitting the LDA model to the training data, the topic distribution for each document is extracted. During the tags classifying phase, multi-class classifiers are built using these vector representing topic proportions as input features, with binarized tags serving as the target variable.

4.3 Doc2Vec

Doc2Vec (Mikolov, Chen et al., 2013) is an unsupervised model for learning embeddings of documents or sentences, also known as document or sentence embeddings. It is a neural network-based approach for representing documents in a continuous, low-dimensional vector space.

4.3.1 Word2Vec

Doc2Vec is an extension of Word2Vec that attempts to determine an adequate continuous vector for a paragraph. Word2Vec includes two algorithms, CBOW and Skip-Gram model. Continuous bag-of-words model (CBOW) predicts the middle word based on surrounding context words, while Skip-Gram model predicts words within a certain range before and after current word in the same sentence. The whole idea of the CBOW model is to maximize average log probability of the focus word given the context words, p ($focus\ word$ | $context\ words$), while the idea of Skip-Gram model is to maximize the average log-probability of predicting context words given the focus word, p ($context\ words$| $focus\ word$), which is an inverse task of CBOM model. Given a sequence of training words w_1, w_2, $w_3 \cdots .. w_T$, where t is the position and k represents the window size. The size of the window can affect the training accuracy and computation time. Larger size of windows leads to higher accuracy but higher computational cost as the model has more information about the words in the context. The average log probability is written as

$$\frac{1}{T}\sum_{t=k}^{T-k} \log\ p\ (w_t | w_{t-k}, \dots, w_{t+k})$$

In the first layer, each word within the window will be encoded into one-hot vector representations. Then this one hot encoding input vector is fed into the simple neural network that has only one hidden layer and one output layer. (Figure 6.) The hidden layer constitutes a lookup matrix, containing weights for the word vector. The weights in the lookup matrix determine the strength of the connections between the neurons and the importance of the features captured by the word vectors. In the output layer, the word w_t can be predicted using multiclass classifier

Softmax function, where the terms y_w, is the unnormalized log-probability for each output word w_i. w_i captures the similarity between two-word vectors. The higher the similarity, the higher the probability. If the words do not appear in each other's context, the resulting representations of the words will be distinct, leading to a small value in the numerator of the probability calculation. Denominator normalizes and the values of probability obtained in the output layer to one. The equation is written as:

$$p(w_t|w_{t-k},\dots,w_{t+k};W) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

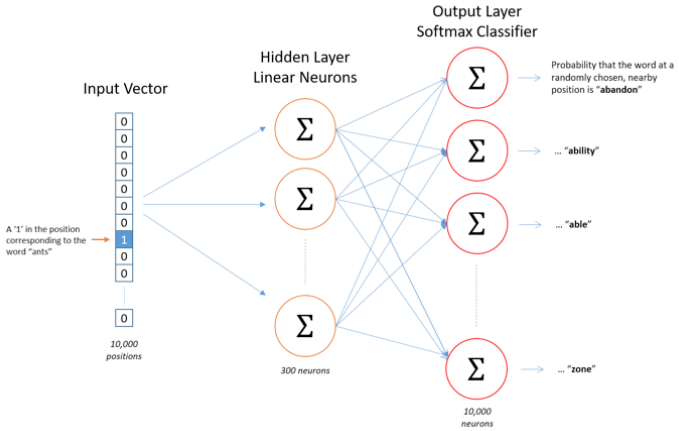


Figure 6. the global overview of Word2vec Skip-Gram model architectures (McCormick, 2016)

4.3.2 Feature extraction from paragraph vector

The motivation behind Doc2Vec was to better understand the unstructured document as a whole, comparing to individual words. Unlike Word2Vec, Doc2vec is trained to represent the meaning of entire sequences of sentences. This is achieved through joint training with the word vectors during the language modeling process. Instead of using words to predict the next word compare to Word2Vec, there is another document token, paragraph vector, added to the training set. The paragraph vector is unique among the documents, while the word vectors are used among all the documents. There are two algorithms, the first algorithm is Distributed Memory Model of Paragraph Vectors (PV-DM), which is similar to CBOW in Word2Vec. The training is done by

passing a sliding window over the sentence, trying to predict the next word based on the previous words in the context and the paragraph vector (or the Document ID in Figure 7). The second algorithm is Distributed Bag of Words version of Paragraph Vector (PV-DBOW), which is similar to Skip-Gram in word2vec model. It is a simpler model that ignores word order. The model only requires to store the Softmax weights, instead of both Softmax weights and word vectors.

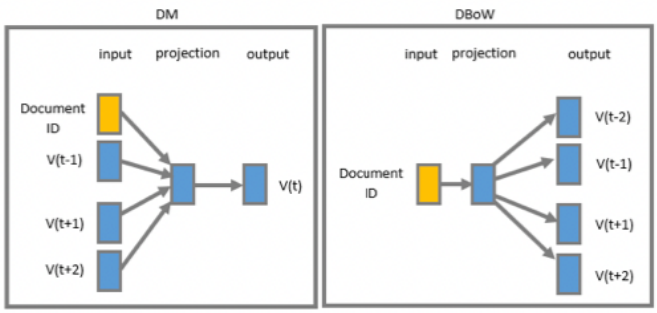


Figure 7. Distributed Memory (PV-DM) and Distributed bag of words (PV-DBoW) (Bilgin & Şentürk, 2017)

The difference between the DM model and DBOW model is that the DM model takes into account the word order, while the DBOW model does not. The DBOW model does not preserved the semantics of the words because it does not use word vectors, therefore, it is harder to detect similarities between words. The PV-DBOW model captures the most important features of a document, including its overall content and meaning, and represents them in a condensed and efficient form. Meanwhile, the PV-DM model focuses on the meaning or the context of the document, using the hidden layer to encode the context information. In this study, we experimented with both and PV-DBOW and PV-DM. The results revealed that PV-DBOW has poor performance compared to PV-DM. These results are consistent with the outcomes reported in the paper (Le & Mikolov, 2014b), where PV-DM consistently performed better than PV-DBOW. Consequently, the PV-DM model was adopted for the analyses in this study.

Doc2Vec is especially useful when trying to determine similarity of content based on the entire content and not only the hot spot or the focus words. The method is based on the assumption that more frequent co-occurrence of two words in a small neighborhood of a document implies higher semantic similarity between them. The paragraph vectors are updated by learning the vector representation of each word and a vector representation of each document. Doc2Vec is trained on the pre-processed question text. The training process begins by converting each sentence and word in the corpus of text to a one-hot representation. The model then selects a random window of words around a specific word from the corpus on each iteration. The center word in the window is the target word that the model attempts to predict, based on the context of the surrounding words. The model computes a feature vector for every document in the corpus. After completing the training process, Doc2Vec generates a vector representation for new documents. The resulting vector, referred to as the paragraph vector (or document vector), serves as an embedding for the new document, which can then be used as input for a supervised classifier to predict the recommended tags. All the weights of the model are fixed except for the weights of the document vector, which are updated at every step.

4.3.3 Training Doc2vec

In the final step of training a Doc2Vec model, the predicted probabilities from the Softmax function are compared with the actual words from the selected context words. This comparison is done using a cross-entropy loss function, which sums up the products of the actual one-hot encoded vector (t) and the predicted probability vector ($\hat{t}$) obtained from the Softmax function. The loss function only takes into account the prediction error for the words in the context, and stochastic gradient descent (SGD) is used to minimize the loss function by updating the model's parameters. The objective of the SGD is to gradually reduce the loss function, minimizing the resulting difference with every iteration. The SGD algorithm achieves this by computing the

gradients of the loss function on the matrices D (document vectors) and U (word vectors). These gradients indicate how much we need to adjust the weights that connect the neurons in the model to reduce the loss function. The gradients are obtained by taking the derivative of the loss function with respect to the components of matrices D and U. To update the weights, we pass the prediction error (subtract the actual one-hot encoded vector (t) from the predicted probability vector ($\hat{t}$) obtained from the Softmax function) back through the network, which involves computing the output prediction error o and the embedding layer error h.

$$\text{cross entropy loss function} = E\left(\vec{t}, \hat{\vec{t}}\right) = -\sum_{j} t_j \log \hat{t}_j$$

The gradient of the loss function is computed for each of the words in the context and summed up before being passed back through the network. We then use the sum of gradients of the loss function to obtain the update values for D and U, which are multiplied by a learning rate *alpha* to limit the size of the weight adjustments. A common choice is to start with alpha=0.025, and then gradually reducing it to 0.001. Finally, we repeat these steps for all the documents in the corpus to complete one epoch of the training process.

4.3.4 Increase efficiency

However, Softmax is computationally very expensive. Computing the denominator of the probability distribution, which serves as a normalizing factor over the entire vocabulary, poses a significant computational challenge since the size of the vocabulary can comprise hundreds of thousands or even millions of words and has a tremendous number of weights. To address this challenge, two popular techniques, negative sampling (Mikolov et al., 2013) and hierarchical softmax (HS) (Morin & Bengio, 2005) are employed instead of using the naive softmax to make the computation feasible.

Negative sampling addresses this by having each training sample only modify a small percentage of the weights. Instead of minimizing the similarity for every word in the vocabulary that is not a context word, negative sampling utilizes a randomly selected subset of words, which are also samples of words that are not neighbors. The negative term is one that the output of network is 0 in this context. The negative samples are chosen using a unigram word distribution, with more frequent words being more likely to be chosen as negative samples. In each training step, one positive example and its associated negative examples is taken and the dot products of the input embedding with its context embeddings are calculated. The result of the scores is processed using sigmoid operations to turn all values between zero and one. The errors are calculated by subtracting the sigmoid scores from the target labels. Then theses errors are used to update the model parameters so that the result would be closer to the target scores in the next calculation.

Hierarchical Softmax is performed by representing the Softmax layer as a binary tree where the words are leaf nodes of the tree, and the calculation of probabilities is performed by navigating from the root of the tree to the relevant leaf.

4.4 Multi-label Classification

The third stage of the tag recommendation model is to classify all the features or topics generated by two models. Since a Stack Overflow question can belong to multiple tags or topics at the same time, we consider the task as a multi-label learning problem.

One-vs-Rest (OvR) is a common strategy for multi-label classification problems, where the goal is to predict the label for each sample. In OvR, a separate classifier is trained for each class, with the samples from that class as positive samples and all other samples as negative samples. During prediction, each classifier makes a prediction for a sample and the class with the highest predicted probability is chosen as the final prediction. It allows the model to predict multiple tags

for a given input text, instead only predicting one single tag. OvR is used with algorithms that do not natively support multi-class classification, in this analysis logistic regression and support vector machines are used.

Logistic regression is often used for classification problem and it is a statistical method for modeling the probability of a binary outcome. When the model that uses logistic regression for tag prediction, the model trains a separate logistic regression classifier for each tag in our dataset. Instead of regular y= {0,1}, in multi-class classification, the description is extended to y= {0,1⋯n}, where y is the tag. It considers one class as 1 and the rest as 0. With one-vs-Rest, it creates the same number of models as the number of tags, with each classifier inside each model is a classifier that separates tags into binary classifier. After models are created, the testing input pass into each model to receive output scores that give different possibility of each model. The final output of the model will be the model that gives the highest score for this input.

A Support Vector Machine is a class of supervised learning algorithms that can be used for classification task. Among supervised classification models like Bayesian classifier, logit model, SVM (support vector machine), k-nearest-neighbor, decision trees and neural networks, SVM is better for Doc2vec model due to its flexibility and its robustness against errors (Truşcă, 2019). Support Vector Machine uses kernel functions that handle non-linear separable data to find the optimal hyperplane in a high-dimensional feature space that maximize and separate two classes. In this analysis, the two classes would be the relevant and non-relevant tags. SVM finds the best hyperplane by looking for the maximum margin between the data points and the hyperplane. The hyperplane depends only on the support vectors, which are the closest data points to the hyperplane and can influence the hyperplane's position. Consider a random point (x) vector and a vector (w) that is perpendicular to the hyperplane. The optimal hyperplane is calculated as

w.x+b=0, meaning that if the result is large then 0 or smaller 0, the point fall into the side of either relevant tags or non-relevant tags. The hard margin can be formulated as:

$$\text{Margin} = \frac{2}{||w||}$$

Since the SVM are designed to perform binary classification, it is not able to perform multiclass and multilabel classification natively and efficiently as a multilabel model does. This is because hyperplane must be an equidistant from the classes to ensure the margin is maximum.

4.5 Evaluation Metric

To evaluate previously described methods, evaluation metric is an important method to structurally comparing the different models in an appropriate manner so that fair conclusion can be made. Popular evaluation metrics for evaluate the models are *Precision, Recall,* and *F-measure* (F1-score), and *Average predicted probability*. *Precision, Recall,* and *F-measure* (F1-score) evaluation metrics are widely used in in previous researches on Software Q&A site for evaluating and comparing with previous tag recommendation models. (He et al., 2022; Khezrian et al., 2020; Liu et al., 2018; Xia et al., 2013)

Accuracy measures the number of correct classifications, which are true positive and true negative, in relation to the total number of all observations. The metrics is calculated as follows:

$$\text{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + True\ Negative + False\ Negative}$$

However, accuracy is not a good measure in some cases, it is not an ideal metric especially for imbalanced data because it does not take into account the distribution of classes. For example, consider a binary classification problem where 95% of the samples belong to class A and only 5% of the samples belong to class B. If the classifier always predicts class A, it would still achieve an

accuracy of 95%. However, this accuracy would not reflect the fact that the classifier is not doing a good job of identifying class B. In common practical scenario example would be credit card fraud detection. Fraud transaction are typically a small minority of all transactions. A model that has high accuracy will not actually be effective on preventing fraud. The dataset in this study may be the case since certain tags are much more common than others.

Precision calculates the number of observations that are correctly classified divided by the total number of observations that are predicted as positive. A high precision means that the classifier is making very few false positive predictions, meaning that the model is predicting a small number of irrelevant tags.

$$\text{Precision} = \frac{True\ Positve}{True\ Positive+False\ Positive} = \frac{True\ Positive}{Total\ Predicted\ Positive}$$

Recall measures the samples that were correctly predicted by the model divided by the sum of true positive predictions and false negative. In this study, it measures the proportion of relevant tags that were correctly recommended by the model. A high recall means that the model is predicting a large number of relevant tags. In other words, the tags that the user is interested in are likely to be included in the list of predicted tags. This is more useful in the tag recommendation context because if the model misses relevant tags, the user may not see the results they were looking for.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive+False\ Negative} = \frac{True\ Positive}{Total\ Actual\ Positive}$$

F-score represents a summarization of both two measures. With F1 measure, a high F1 score has both a high precision and high recall. Therefore, it can give a more accurate result of how well the models perform. When the classes in a dataset are imbalanced, precision and recall can be misleading. Since some tags occur much more frequently than others, we also evaluate Micro F1

score. Micro F1-score is calculated by counting the total number of true positives, false positives, and false negatives across all classes and then calculating the F1-score based on those counts.

$$F1 = \frac{2 \times recall \times .precision}{recall + precision}$$

However, precision, recall, and F1 score measures depend only on the tag that receives the highest probability, and therefore, do not reflect the quality of predictions for tags that do not receive the highest probability. For example, suppose a question has two relevant tags: 'python' and 'pandas'. If the model predicts 'python' with a probability of 0.8 and 'pandas' with a probability of 0.2, the precision, recall, and F1 score will be based only on the prediction of "python", and the prediction of 'pandas' will be considered a false negative. In contrast, average predicted probability provides information on how high the probability of the outcome is when it is not highest. It takes into account the probabilities of all the predicted tags, not just the one with the highest probability. The benefits of using average predicted probability as a performance measure is that it provides a more nuanced view of model performance than previous measures such as precision, recall, and F1 score.

Unlike the outputs of logistic regression can be interpreted as probabilities, SVM (support vector machine) does not provide probabilistic outputs directly. To address this limitation, a technique called Calibrated Classification classifier model is used to obtain probability estimates from the SVM classifier. Calibrated predicted probabilities are probabilities that have been adjusted or transformed to better match the expected distribution of probabilities for each class. In this study, we utilize Platt scaling, a method that fits a logistic regression model (sigmoid function) to the output of the SVM classifier to map the SVM outputs to calibrated probabilities. Platt scaling is trained on the decision function values produced by the SVM classifier, transforming the SVM outputs into representations of the true class probabilities. Average predicted probabilities are calculated by taking the mean of the calibrated probabilities.

5 Results

In this chapter, the results of the analysis are presented. In the first sections, the result of text pre-processing is presented. In the second and the third sections present the result of building each model, including model interpretability. These two sections focus on achieving the highest accuracy by identifying the best performance of the models and examining the results of employing different multi-label classifiers. In the final section, we compare two models by different performance metrics.

5.1 Text pre-processing

To verify the text pre-processing procedure, word cloud (Figure 8.) is used to get a visual representation of most common words across all the documents.

Figure 8. Word cloud after text pre-processing

The Table 3. below provides an example of the title and question before and after text-preprocessing phase. The table present the 20th question in the dataset, which the question is tagged with '*php*' and '*mysql*'.

Before text preprocessing	After text pre-processing

I cant UPDATE datetime to MySQL
 I need a little help. I'm try to UPDATE a datetime to MySQL, but it didn't work.
 The declaration is like this:
<pre><code> $startDate = time();
  $time = date("Y-m-d H:i:s", strtotime('+7 days', $startDate));
</code></pre> After this i want to UPDATE, but in MySQL is still blank always.
 UPDATE:
<pre><code>mtquery("UPDATE table SET end_time = ".$time." WHERE id = ".$table['id']."");
</code></pre>
 If i use NOW() instead of ".$time.", it works perfectly.
 If someone can help, please write the solution.
 Thanks,
 KoLi
	update datetime mysql need help try update datetime mysql work declaration startdate time time date day startdate want update mysql still always update update table set end time time table use instead time work perfectly help write solution thank koli

Table 3. An example of original text and pre-processed text

5.2 LDA

In this section, the results of the LDA-based tag recommendation model are presented. The LDA model with 90 topics was trained on the term frequency-inverse document frequency (TF-IDF) representation of the training data. To achieve the best performance, several pre-processing steps were undertaken, including the selection of an optimal number of topics, lemmatization, and removal of unrelated words using Part-of-Speech (POS) tags. Then we utilize multi-label classification task and classifiers were trained using the topic distributions from the training set and their corresponding multi-label binarized tags. The performance of the model was assessed using micro-averaged precision, recall, and F1 scores. Furthermore, the average predicted probability was calculated. The following sections will detail the pre-processing steps taken to achieve the best performance before the training process, the tuning process and the determination of an optimal threshold.

5.2.1 Numbers of topics

For determining the numbers of topics for LDA model, initially, we considered using perplexity as an evaluation metric to determine the best number of topics. However, the perplexity scores increase as the number of topics increases. This result prompted us to use alternative evaluation metrics for selecting numbers of topics. We decided to consider the balance between F1 score and average predicted probability. This will help ensuring the model is both accurate and confident in its predictions, which should lead to better performance when applied to predicting new tags. The chart in Figure 9 presents the number of topics in an LDA model and its corresponding F1 score (black line) and average predicted probability (dark blue line). The numbers of topics where both F1 score and average predicted probability are relatively high is at topics 90, yielding an F1 score of 0.3855979 and an average predicted probability of 0.0198371. The exact value of Figure 9 is shown in Figure 13 (Appendix).

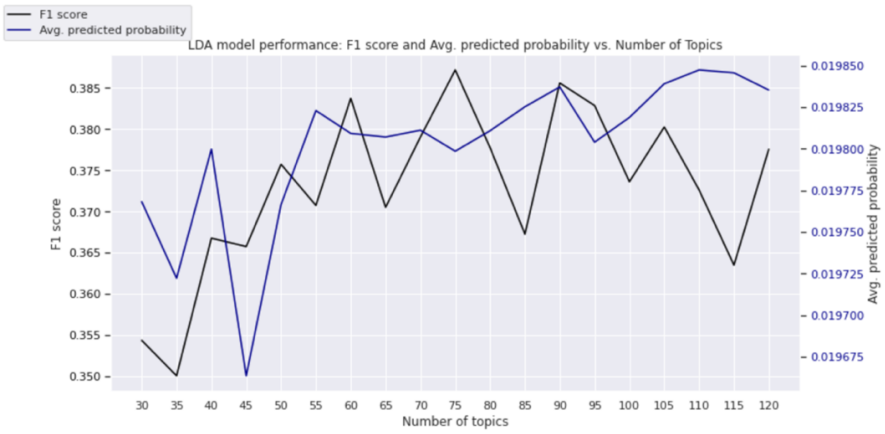


Figure 9. LDA model performance: F1 Score and average predicted probability vs. number of Topics

5.2.2 Lemmatization on different words

Lemmatizing only specific parts of speech, such as nouns and verbs, yielded different results compared to lemmatizing all parts of speech. In particular, we discovered that when lemmatizing only a subset of words, the model might not capture the full context of the text, leading to less accurate results. Conversely, when we lemmatized all words, the model was better able to

comprehend the context and thus produced more accurate results. Based on these results, we decided to lemmatize nouns, verbs, adverbs, and adjectives for further analysis.

Lemmatization is a process of reducing words to their base or root form, often referred to as the lemma. The lemma is the word that appears in the dictionary, and it can be a noun, verb, adjective, or adverb. Lemmatizing only specific word types such as nouns and verbs, yielded different results compared to lemmatizing all word types. When lemmatizing all four common word types, the model was better able to comprehend the context and thus produced more accurate results. These findings provide insights into the importance of pre-processing techniques in natural language processing and their potential impact on model performance. Based on these results (Table 4.), we decided to lemmatize nouns, verbs, adverbs, and adjectives for further analysis. The highest result for each performance metric is indicated in bold.

Numbers of most informative word types	Multi-label Classifier	Precision	Recall	Micro F1	Average predicted probability
Noun, Verb	LDA + Logistic Regression	0.31407	0.48109	0.38004	0.01983
	LDA + SVM	0.38671	0.47592	0.42670	0.02412
Noun, Verb, Adverb	LDA + Logistic Regression	0.29801	0.46907	0.36447	0.01981
	LDA + SVM	0.37938	0.45701	0.41459	0.02362
Noun, Verb, Adverb, Adjective	LDA + Logistic Regression	0.31783	0.47925	0.38220	0.01985
	LDA + SVM	**0.38885**	**0.48128**	**0.43016**	**0.02426**

Table 4. Comparison of Lemmatization by Parts of Speech on model's Performance

5.2.3 Removing Unrelated Words with random forest

After text pre-processing, including lemmatization on noun, verb, adverb, and adjective, there are still many unrelated words to the topics. To investigate what types of words are more informative, this study chooses to remove unrelated words with random forest and POS Tagger.

We start with converting all words from the questions text into the word-vector format with binary values, as well as their corresponding tags. Subsequently, these columns are trained using

the OneVsRestClassifier combing with a Random Forest model. Random forest models are well-suited for feature selection, which can be used to identify the most informative POS tags in this case. Random forest models use an ensemble of decision trees to make predictions, which can improve the stability and generalizability of the model. From the Random Forest model, we extract feature importance and associate them with their respective feature names (words). Next, we classify words into their respective parts of speech (nouns, adjectives, verbs, and adverbs). Finally, to find what word types are more important, we compute the sum of each word type by summing their importance values (Table 5.)

nouns	0.652844
verbs	0.146715
adjectives	0.137607
adverbs	0.026481

Table 5. Importance of word type from random forest

In Table 6 below, the performance metrics of the model are presented while retaining varying combinations of the most informative word types, ranging from the most important to the least important (noun, verb, adjective, adverb). The highest score for each performance metric is highlighted in bold. The optimal performance is achieved by retaining nouns, verbs, and adjectives in the text, while removing other word types.

Keeping the top most informative word types	Multi-label Classifier	Precision	Recall	Micro F1	Average predicted probability
Only noun	LDA + Logistic Regression	0.363600	0.357764	0.360658	0.019834
	LDA + SVM	0.361034	0.426239	0.390937	0.023144
Noun and verb	LDA + Logistic Regression	0.3623054	0.367199	0.364736	0.019809
	LDA + SVM	0.359467	0.412497	0.384161	0.022496
Noun, verb and adjective	LDA + Logistic Regression	0.378879	0.413398	0.395387	0.019819
	LDA + SVM	0.385115	**0.468013**	**0.422537**	**0.023824**
Noun, verb, adjective and adverb	LDA + Logistic Regression	0.369840	0.392529	0.380847	0.019788
	LDA + SVM	**0.387406**	0.464137	0.422314	0.023487

Table 6. Comparison of informative words on model's Performance

5.2.4 Hyperparameters tuning and threshold setting

After text- preprocessing and removing unrelated words, next step of the model was to extract feature from LDA and use these features in the classification task. Topic distribution tells us how much each topic was presented in the document. Each question receives a topic distribution for the number of 90 topics which are used as features in the classification task.

From the result of Table 7, we can see that the highest topic distribution is 0.1367993 at topic 40, indicating that the 20th question is significantly associated with this topic. A few other topics, such as 0.09325721, 0.05174439, 0.0342548, and 0.02415846, also have higher proportions than the rest, suggesting that the question may be related to these topics as well.

Topic 1	Topic 2	Topic 3	[…]	Topic 40	[…]	Topic 89	Topic 90
0.02023364	0.00965164	0.00965016	[…]	0.1367993	[…]	0.00964976	0.00964926

Table 7. An example of a topic distribution for 20th question

Before tuning the hyperparameters, the majority of the topic proportions are around the same numbers, meaning that the LDA model is not capturing distinct topics effectively or that the document is not strongly associated with these topics. We examine topic distribution of each document and discovered that when the alpha and beta were set to default, topic distribution present almost the same score across all the topics. After experimenting the hyperparameters we set alpha =0.1 and beta =0.01. We set beta lower than default to avoid not overly focused on specific words. While experimenting, we discovered that higher alpha leads to higher precision but lower recall. This means that when alpha is higher, each question is more likely to contain a more diverse mixture of topics, leading to a more uniform distribution of topic probabilities across documents. However, the model may not be able to capture the specific nuances of each topic and may not assign high probability to a relevant topic if it is not a dominant theme in the document. By setting the hyperparameters lower, it increases sparsity and it makes the model more interpretable by identifying the most important topics and words for each document.

In the context of tag recommendation models, it is common to establish a threshold for determining which tags should be recommended for a given document. Setting an appropriate threshold leads to a higher F1-score, which represents the harmonic mean of precision and recall. The threshold that yields the highest F1-score is selected as the optimal value for recommending tags in both the LDA and Doc2Vec models. This approach ensures that the recommendations provided by the models are as accurate and relevant as possible, and finds an appropriate balance between false positives and false negatives. For LDA model, the threshold value of SVM is set to 0.1, and logistic regression is set to 0.13, meaning that if the predicted probability of a tag being associated with a question is greater than or equal to 0.1, that tag will be assigned to the question.

5.2.5 Visualization with importance of topic keywords plots

In the Word Count and Importance of Topic Keywords plots (Figure 11.) in Appendix, word count refers to the frequency of each word in the entire dataset (light blue). A high word count indicates that a word appears often in the dataset. Weights (blue) represent the importance or contribution of each word within a specific topic. A high weight for a word in a topic indicates that the word is more significant in defining that topic. Topic 4 is related to programming in C and C++ languages, and the important words in this topic include common elements of C and C++ code, such as 'cout' and 'printf'. Topic 10 is associated with JavaScript and jQuery, while Topic 12 appears to be related to error handling, debugging, and task management in programming. The important words, such as "message," "task," "letter," "structure," "invalid," and "error," suggest that this topic focuses on issues that arise during the development process. Topic 19 appears to be related to data manipulation and organization, specifically focusing on array and string operations, sorting, and indexing. From the inspection of these topics, most of the topics seem to make sense, and there are only a few overlapping or redundant topics. Moreover, words with high word counts

but low weights, which might be common across multiple topics and can potentially be added to the stop words list to refine the LDA model, such as words ‘use’, ‘user’, ‘get’, and ‘file’.

5.3 Doc2Vec

After the same text pre-processing step as LDA, except for removing POS tag, the pre-processed texts are fed into Doc2Vec model. The model is fine-tuned by trying different combinations of hyperparameters. The vector space is set to a dimension of 300, and it is trained for 20 epochs with a window size of 5. The learning rate decreases by 0.002 for each epoch. The adjustment to vectors become smaller and smaller with each epoch. Negative sampling (the parameters hs=0 and negative is tuned to 5) is compared with hierarchical Softmax (hs=1). Negative sampling has a slightly higher precision, 0.0067 higher with logistic regression, but has slightly lower precision, 0.0027 higher with SVM. The performance of negative sampling and hierarchical Softmax does not differ significantly. Negative sampling is chosen due to its lower computational complexity. Additionally, since the tags are imbalanced, negative sampling may be more effective in addressing this issue.

The optimal threshold that yields the highest F1 score for Logistic regression and SVM differ. For Logistic Regression, the threshold value was set to 0.09, while for SVM, it was set at 0.06. When the predicted probability of a tag being associated with a question is greater than or equal to these respective thresholds, the tag will be assigned to the question. Setting different threshold ensures that the most appropriate tags are selected for each question based on the performance of the individual classifiers.

	Precision	Recall	Micro F1	Average predicted probability
Doc2Vec+ Logistic Regression	0.13196	0.25602	0.17415	0.01977
Doc2Vec + SVM	0.18070	0.268130	0.18070	0.01235

Table 8. The performance of Doc2Vec model

5.3.1 Visualization with t-SNE

To examine the model, we use t-SNE (Figure 10.) to fit multiple dimensions of the model into a simple plot. t-SNE (t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique that is often used to visualize high-dimensional data visualize the vector representations adjust earlier in the Doc2Vec model. It can help understanding which words are most similar to a given tag and insights into the relationships between different tags. The main parameter controlling the fitting is *perplexity*. A low perplexity value (e.g. 5) will result in a t-SNE that mainly preserves the local structure of the data, and therefore will tend to produce clusters or tight groups of similar data points. A high perplexity value (e.g. 50) will result in a t-SNE that mainly preserves the global structure of the data and will tend to produce a more dispersed or evenly distributed visualization. We set the perplexity value to a low number (perplexity =5) to help identify clusters of similar questions. We have annotated the plot with the tag names and used as label to give a sense of what type of questions are clustered together. The position of each question in the plot is determined by the similarity of its content to other questions. If two questions are close together in the plot, it means they have similar content, and if they are far apart, their content is dissimilar. For instance, in the plot, we observe that the 'mysql', 'sql' and 'sql-server ' tag names are close together, suggesting that questions the contents are similar.

However, in some cases both words are related, but they have different usage patterns and are used differently in a sentence, which are reflected in their embeddings. As a result, the t-SNE plot may place them in different regions of the space, depending on their specific patterns of co-occurrence and usage. For instance, the tags names 'ios' and 'xcode' are far, but the tag 'ios' is closed to 'android', although xcode is more related to iOS. This means that the model has identified document similarities between iOS and Android, such as both being mobile operating systems, sharing app development concepts, or having a more significant number of questions

and discussions around them. iOS is an operating system used in Apple devices, and Xcode is an integrated development environment used for developing applications for iOS, macOS, and other Apple platforms. Although they are closely related, questions around iOS might focus more on its features, and user experience, while discussions about Xcode might focus on its tools and debugging. Similarly happens to 'javascript'. 'javascript' is closer to its popular library 'jQuery' and far from 'typescript'. TypeScript is a superset of JavaScript, meaning that it extends JavaScript by adding optional static typing and other features. The discussions around TypeScript might focus more on its unique features, such as type checking, interfaces, and classes. In contrast, questions about JavaScript might be more focused on its core features, libraries, and frameworks.

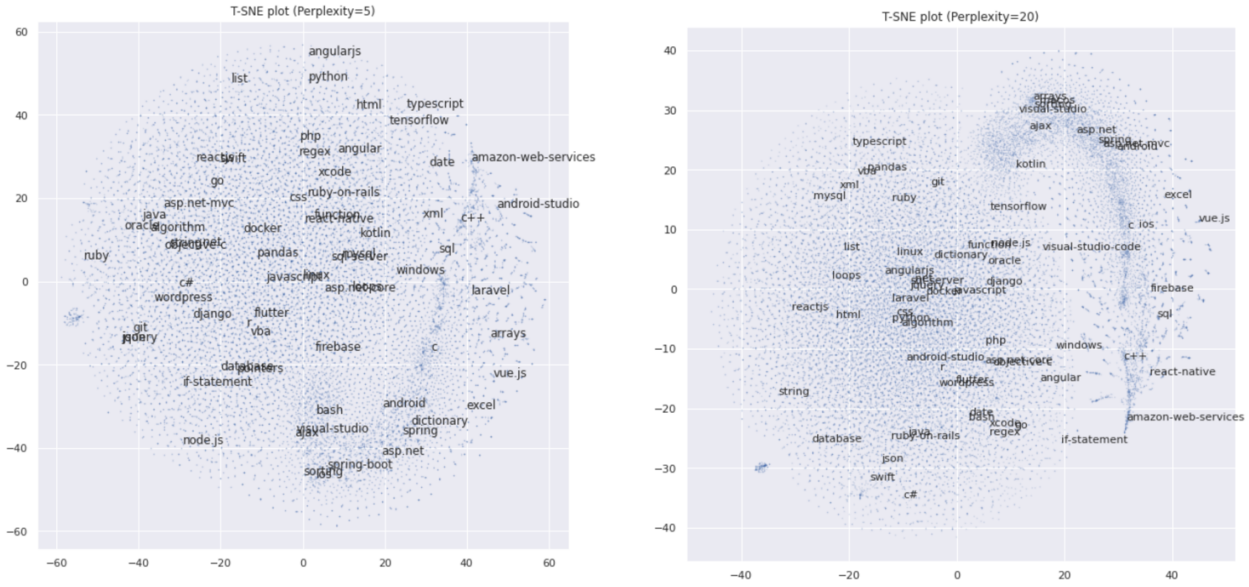


Figure 10. t-SNE plot of perplexity =5 and perplexity =20

5.3.2 Classification report

The classification report (Table 8.) provides an overview of how well the model in assigning tags to the questions. Tags are listed in descending order according to their frequency presented in the dataset. The results demonstrate that the model is more accurate in predicting tags with higher frequency, while its performance diminishes for tags with lower frequency. The table also

reveals that categories associated with data structures, algorithms, and general programming constructs generally exhibit poor performance. For instance, data structures and operations such as 'string' (F1 score: 0.0000), 'dictionary' (F1 score: 0.0000), 'list' (F1 score: 0.0200), 'arrays' (F1 score: 0.0251), and 'dataframe' (F1 score: 0.0000) demonstrate low F1 scores. Similarly, general programming constructs like 'if-statement' (F1 score: 0.0000), 'for-loop' (F1 score: 0.0000), 'loops' (F1 score: 0.0000), and 'function' (F1 score: 0.0155) also underperform. Finally, algorithm and technique-related tags such as 'sorting' (F1 score: 0.0000), 'algorithm' (F1 score: 0.0157), 'regex' (F1 score: 0.0051), and 'pointers' (F1 score: 0.0000) yield low performances. These categories may struggle due to their broad applicability across multiple programming languages, which can make it challenging for the model to predict them accurately based on context.

SVM can predict more tags that have smaller size, or less popular, for example, xml (470 tags) and windows (357tags), with F1 score 0.0230 and 0.0923 respectively, while logistic regression performed 0.000. Logistic regression is a linear model, and it may not be able to fully capture the complex relationships between the dense, high-dimensional document vectors produced by Doc2Vec.

Precision, Recall and F1 score require a predicted probability for the actual tag to be the highest. To determine whether the models are capable of highlighting that a tag is more likely in one document compared to the average document, we compare the average predicted probability for each tag with the average presence of each tag across all the tags selected in this tag recommendation model (tags appear more than 250 times). Average predicted probability for a tag can be retrieved by averaging the predicted probabilities of that tag across all the documents in the test set. If the average predicted probability is higher than the average presence, it means that the model is able to identify tags that are more likely in one document compared to the average document. The average predicted probabilities and average presence shown in Table 8.

Since the average predicted probabilities of all the tags are lower than the average presence, it suggests that the model is not making confident predictions for these tags. The tags that have low performance, such as 'string', '.net', 'react-native', 'angularjs', 'vue.js', 'wordpress', 'oracle', 'loops', 'dictionary', 'ajax', 'visual-studio-code', 'asp.net-mvc', 'sorting', 'for-loop', 'pointers', and 'dataframe' exhibit low F1 scores (0.0000), and their average predicted probabilities are considerably lower than their average presence.

Tags	Number of tags	Precision	Recall	F1 score	Average presence	Average predicted probability
python	8517	0.1400	0.9617	0.2444	0.1347	0.08498
javascript	7106	0.1391	0.9707	0.2434	0.1335	0.08535
java	6242	0.1207	0.9494	0.2141	0.1179	0.07143
android	4838	0.2155	0.2015	0.2082	0.0919	0.05408
php	4161	0.1024	0.0621	0.0773	0.0771	0.05075
c#	4154	0.1527	0.0965	0.1183	0.0773	0.04808
html	3376	0.1266	0.0489	0.0705	0.0610	0.04188
c++	3003	0.1769	0.0808	0.1109	0.0531	0.03553
css	2104	0.1310	0.0325	0.0521	0.0363	0.02675
c	1845	0.1610	0.0352	0.0578	0.0335	0.02210
arrays	1835	0.1127	0.0141	0.0251	0.0353	0.02140
jquery	1831	0.1111	0.0116	0.0211	0.0373	0.02113
ios	1809	0.2938	0.0882	0.1356	0.0331	0.02060
sql	1670	0.1462	0.0363	0.0582	0.0325	0.01894
mysql	1582	0.0896	0.0133	0.0232	0.0279	0.01903
swift	1562	0.3304	0.0787	0.1271	0.0292	0.01765
angular	1510	0.2903	0.0612	0.1011	0.0274	0.01732
reactjs	1169	0.2619	0.0324	0.0577	0.0210	0.01351
r	1145	0.2623	0.0455	0.0775	0.0218	0.01340

regex	1118	0.0278	0.0028	0.0051	0.0222	0.01307
node.js	1064	0.0811	0.0101	0.0180	0.0184	0.01279
sql-server	1063	0.1373	0.0211	0.0366	0.0206	0.01197
json	971	0.0417	0.0034	0.0063	0.0183	0.01067
string	962	0.0000	0.0000	0.0000	0.0187	0.01115
typescript	875	0.1667	0.0197	0.0352	0.0157	0.01014
linux	653	0.0909	0.0116	0.0205	0.0107	0.00803
excel	630	0.1667	0.0099	0.0186	0.0126	0.00723
docker	618	0.2000	0.0372	0.0628	0.0116	0.00725
pandas	585	0.2222	0.0112	0.0214	0.0110	0.00671
list	578	0.2000	0.0105	0.0200	0.0118	0.00662
.net	568	0.0000	0.0000	0.0000	0.0096	0.00658
android-studio	550	0.1176	0.0110	0.0201	0.0113	0.00600
react-native	531	0.0000	0.0000	0.0000	0.0105	0.00589
ruby	506	0.2353	0.0242	0.0440	0.0102	0.00573
vba	496	0.1250	0.0066	0.0126	0.0094	0.00570
laravel	483	0.3077	0.0267	0.0491	0.0093	0.00568
asp.net	471	0.0714	0.0062	0.0114	0.0100	0.00497
xcode	470	0.2000	0.0205	0.0373	0.0090	0.00524
flutter	464	0.2500	0.0496	0.0828	0.0087	0.00527
go	451	0.0000	0.0000	0.0000	0.0079	0.00532
firebase	450	0.1364	0.0229	0.0392	0.0081	0.00507
git	448	0.0000	0.0000	0.0000	0.0073	0.00561
database	445	0.0000	0.0000	0.0000	0.0073	0.00548
amazon-web-services	427	0.0909	0.0156	0.0267	0.0079	0.00483
kotlin	418	0.1429	0.0155	0.0280	0.0080	0.00467
bash	408	0.0870	0.0169	0.0284	0.0073	0.00480
visual-studio	388	0.0714	0.0087	0.0155	0.0071	0.00118
angularjs	386	0.0000	0.0000	0.0000	0.0069	0.00458

function	380	0.1000	0.0084	0.0155	0.0073	0.00452
algorithm	378	0.0526	0.0093	0.0157	0.0067	0.00488
django	367	0.1250	0.0081	0.0152	0.0077	0.00407
vue.js	362	0.0000	0.0000	0.0000	0.0061	0.00446
windows	357	0.2143	0.0508	0.0923	0.0073	0.00390
wordpress	346	0.0000	0.0000	0.0000	0.0070	0.00402
oracle	340	0.0000	0.0000	0.0000	0.0057	0.00388
loops	338	0.0000	0.0000	0.0000	0.0057	0.00409
spring	329	0.2353	0.0435	0.0734	0.0057	0.00351
asp.net-core	327	0.0417	0.0102	0.0164	0.0060	0.00358
ruby-on-rails	324	0.1818	0.0189	0.0342	0.0065	0.00355
dictionary	324	0.0000	0.0000	0.0000	0.0064	0.00360
objective-c	308	0.1333	0.0233	0.0396	0.0053	0.00341
tensorflow	306	0.2857	0.0952	0.1429	0.0052	0.00361
ajax	302	0.0000	0.0000	0.0000	0.0061	0.00337
date	302	0.0000	0.0000	0.0000	0.0059	0.00358
visual-studio-code	293	0.0000	0.0000	0.0000	0.0060	0.00333
if-statement	289	0.0000	0.0000	0.0000	0.0047	0.00365
asp.net-mvc	288	0.0000	0.0000	0.0000	0.0049	0.00331
sorting	277	0.0000	0.0000	0.0000	0.0051	0.00313
spring-boot	276	0.4167	0.0625	0.1087	0.0049	0.00296
xml	275	0.1429	0.0125	0.0230	0.0049	0.00317
for-loop	274	0.0000	0.0000	0.0000	0.0055	0.00316
dart	271	0.3000	0.0353	0.0632	0.0052	0.00299
pointers	268	0.0000	0.0000	0.0000	0.0051	0.00325
selenium	268	0.1429	0.0244	0.0417	0.0051	0.00309
numpy	267	0.1875	0.0375	0.0625	0.0049	0.00303
dataframe	266	0.0000	0.0000	0.0000	0.0059	0.00282
webpack	260	0.1538	0.0263	0.0449	0.0047	0.00297

macos	257	0.1000	0.0141	0.0247	0.0044	0.00302
vb.net	256	0.0714	0.0143	0.0238	0.0043	0.00290

Table 8. Classification report of the best tuned Doc2Vec model

5.3.3 Challenges and Limitations of Doc2vec Model

Contextual variations can impact the performance of a model, as the usage of a tag may differ depending on the context of the question or answer. For instance, the tag 'loops' depends on the context in which it appears, as different types of loops, such as 'for-loops,' may be mentioned. If the text preprocessing step removes the word 'for' as a stopword, it can make it more difficult for the model to learn associations with the 'loops' tag. Another example is the tag 'sorting', which might be used in various contexts, such as sorting a list of numbers or a table of data, but the exact implementation might be different in each programming languages. The tag 'date' might be used in the context of parsing or formatting dates in different programming languages, this could make it more difficult for the model to learn a consistent association with the '*date'* tag.

Moreover, the cleaning and pre-processing steps can affect the quality of the model. Several function names and programming terms that might be removed when using the NLTK library's built-in English stopword list. Removing certain words (e.g. 'if', 'else', 'while', 'for', 'return', 'from') can make it more difficult for the model to learn associations with certain tags, for example the tag *'if-statement'*, '*loops*' and '*for-loop'*

5.4 Model Comparison

	Precision	Recall	Micro F1	Average predicted probability
LDA + Logistic Regression	0.378879	0.413398	0.395387	0.019819
LDA + SVM	**0.385115**	**0.468013**	**0.422537**	**0.023824**
Doc2Vec+ Logistic Regression	0.13196	0.25602	0.17415	0.01977
Doc2Vec + SVM	0.18070	0.26813	0.18070	0.01235

Table 9. Performance Comparison of the best LDA and Doc2Vec

Table 9. shows that the LDA + SVM model demonstrates the best overall performance in terms of precision, recall, Micro F1 score and average predicted probability. The LDA + Logistic Regression model shows moderate performance, while the Doc2Vec-based models (Doc2Vec + Logistic Regression and Doc2Vec + SVM) exhibit relatively weaker performance in the given tag recommendation task.

6 Discussion and Conclusion

6.1 Main conclusion

Tag recommendation in software Question-Answering (Q&A) forums is crucial for organizing questions and enhancing user engagement. This study compares the performance of Latent Dirichlet Allocation (LDA) and Doc2Vec methods to improve tag recommendations for Software Q&A forums. The research aims to answer the main question: How can a tag recommendation model be built using topic modeling and word embedding approaches to enhance tagging quality?

The methodology consists of text preprocessing, feature extraction, and multi-label classification using Support Vector Machines (SVM) and Logistic Regression. The performance of both methods is evaluated based on precision, recall, F1-score, and average predicted probability. Results show that LDA combined with SVM outperforms the Doc2Vec model with an F1 score of 0.422537. The finding emphasizes the importance of pre-processing stage and the need for a customized stop word list. This main research question was explored through the following sub-question:

Sub Question 1: How can feature be extracted to automatically recommend a set of tags from Q&A forum?

In Chapter 4 methodology section, how two models extracted their features are explained. To extract features for automatically recommending a set of tags from Q&A forums, the text of questions and their titles should be preprocessed. The next step is to convert clean text into numerical representation. Additionally, before extracting features for LDA, the model was extended by removing unrelated words types using POS tagging and random forest for determining the importance of the word types. LDA calculates the probability of a word belonging

to a topic. Topics are defined as probability distributions over the vocabulary. The generated topic vectors are then used as feature vectors in supervised classification models.

Feature extraction in Doc2Vec was achieved by training the model on the pre-processed questions text, which allows the model to learn the patterns and relationships between the documents and their tags. In the final step of training a Doc2Vec model, the predicted probabilities are compared with the actual words from the selected context words using a cross-entropy loss function. The resulting document vector generated by the model is serve as an embedding for the new document, which was then used as input to a multi-label classifier to predict the recommended tags.

Sub Question 2: How to find interpretable insights in this analysis?

To find interpretable insights in this analysis for LDA, we focus on the LDA model and its performance with different preprocessing techniques and visualizations. The study investigated the impact of lemmatizing different word types on the performance of the LDA model and found that lemmatizing all type of word types (Noun, Verb, Adverb, Adjective) produced more accurate results. Additionally, we used random forest to visualize the importance of each POS Tagger then remove the less important word types to improve the performance of the LDA model. By adjusting the alpha and beta values, we have enhanced the model's ability to identify distinct and meaningful topics associated with each document. The optimal hyperparameter settings allowed for a sparser topic distribution, resulting in facilitating the identification of the most important topics and words for each question. Topics distributions can be extracted to examine whether the model is tuned well. Additionally, selecting an appropriate threshold for recommending tags is essential to ensure that the tag recommendations provided by the models are accurate and relevant. The Word Count and Importance of Topic Keywords plots offers interpretable insights

by identifying significant words that define each topic. To further refine the LDA model, we suggested to add the words with high word counts but low weights to the stop words list.

For interpreting Doc2Vec, we use t-SNE plot to gain insights different usage patterns and are used differently in a sentence when mentioning certain tags. The classification report provides a comprehensive evaluation in analyzing the limitations and strengths of the model by providing insights into the model's performance on different tags. The model also has low performance on some library-related tags and programming logic and objects. The comparison of multi-label classifier shows that the Doc2vec model may not be well-suited for linear models like logistic regression. Then we address the limitation of Doc2vec model including difficulty in capturing the context of specific terms in the text, sensitivity to contextual variations, and the preprocessing steps. Removing certain stop words in the text during preprocessing, such as common programming terms and function names, can limit the model's ability to learn associations with certain tags.

Finally, considering the design of the tag recommendation model in this study, which allows users to manually choose tags from the recommended tags after typing their questions, and our main goal is to avoid tag inconsistency and tag synonymy, we want the model to identify as many relevant tags as possible even if it means including some irrelevant tags. Irrelevant tags will be filtered out by users, but it would be helpful if the more relevant tags are shown to the users. Therefore, Recall is more important than Precision since we care about more relevant tags to be suggested so that so each question is well-organized.

Sub Question 3: Which approach can best used for tag recommendation? Can some conclusion be reached based on the results from different approaches?

The result of this study demonstrated that the combination of LDA with SVM is the most effective approach for tag recommendation as it has higher Precision, Recall, Micro F1 score, and Average predicted probability. Furthermore, SVM has outperformed Logistic Regression for both models, suggesting that it is better suited to handle the complexity of the problem. Due to model complexity, such as the neural network architecture, stochastic gradient descent for optimization and more hyperparameters, Doc2vec has longer training and prediction times than LDA.

6.2 Limitations and future research

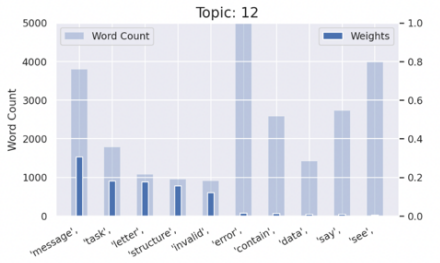
Both LDA and Doc2Vec have their own strengths and limitations. In the case of Doc2Vec, the limitations stem from contextual variations. For both models, the cleaning and pre-processing steps impact on the model's quality, and therefore in the future research, the stop word list of both models needs to be customized.

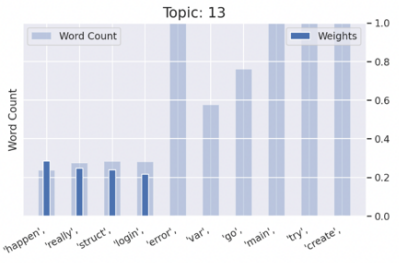
The presence of code snippets in the dataset may favor LDA's ability to capture word co-occurrence patterns as code snippets might contain tokens that frequently co-occur with certain programming concepts or tags. While Doc2Vec attempts to capture the meaning of the entire text, including title, question descriptions, and code snippets. This can be challenging as code snippets may have different characteristics, which could impact the model's ability to learn meaningful associations with tags. Future research could explore separating the code snippets and other text or eliminating them. Since code snippets are an essential component of software Q&A questions, future research could also investigate the effectiveness of specialized code representation methods for tag recommendation, such as pre-trained language model codeBERT.

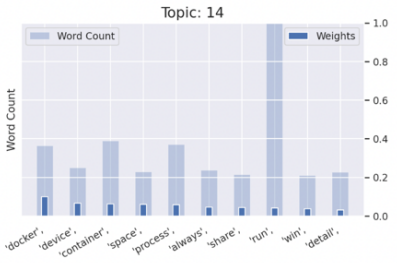
Additionally, this study focused on a single dataset. To obtain more robust and reliable results, we are interested in conducting using various datasets and different software Q&A site. Addressing this issue is an essential aspect of future research.

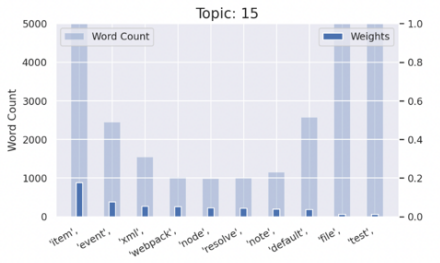
7 Appendix

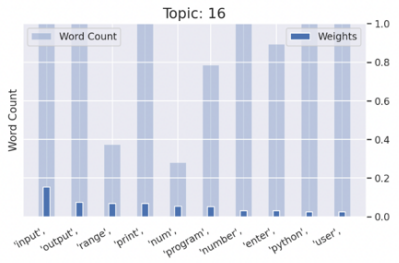
A. Word Count and Importance of Topic Keywords plots

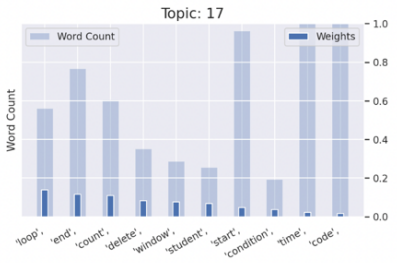

Topic: 12
Word Count
Weights
'message', 'task', 'letter', 'structure', 'invalid', 'error', 'contain', 'data', 'say', 'see',

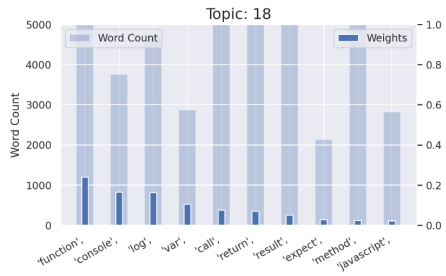

Topic: 13
Word Count
Weights
'happen', 'really', 'struct', 'login', 'error', 'var', 'go', 'main', 'try', 'create',

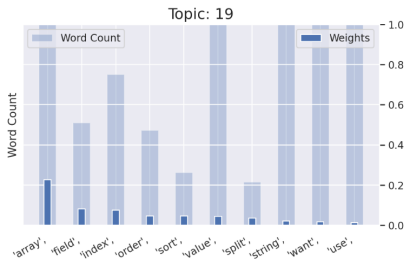

Topic: 14
Word Count
Weights
'docker', 'device', 'container', 'space', 'process', 'always', 'share', 'run', 'win', 'detail',

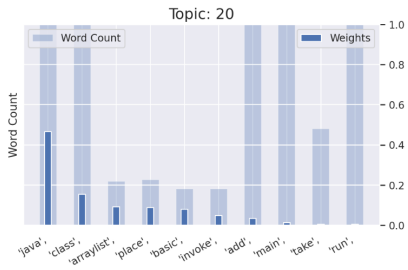

Topic: 15
Word Count
Weights
'item', 'event', 'xml', 'webpack', 'node', 'resolve', 'note', 'default', 'file', 'test',

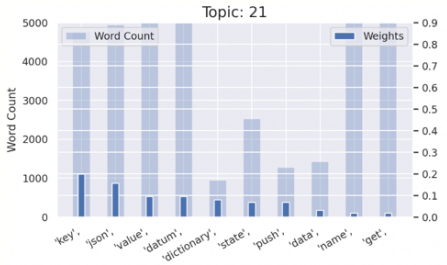

Topic: 16
Word Count
Weights
'input', 'output', 'range', 'print', 'num', 'program', 'number', 'enter', 'python', 'user',

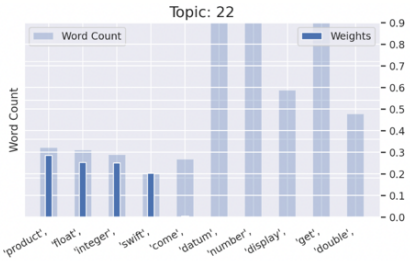

Topic: 17
Word Count
Weights
'loop', 'end', 'count', 'delete', 'window', 'student', 'start', 'condition', 'time', 'code',

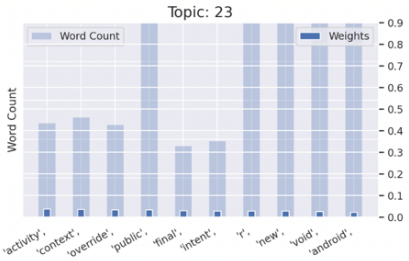

Topic: 18
Word Count
Weights
'function', 'console', 'log', 'var', 'call', 'return', 'result', 'expect', 'method', 'javascript',

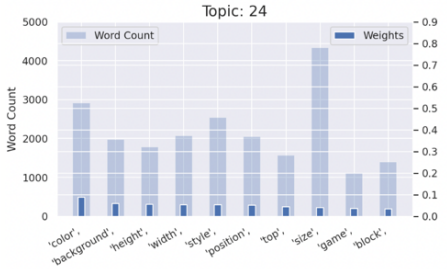

Topic: 19
Word Count
Weights
'array', 'field', 'index', 'order', 'sort', 'value', 'split', 'string', 'want', 'use',

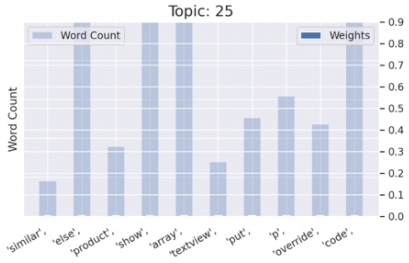

Topic: 20
Word Count
Weights
'java', 'class', 'arraylist', 'place', 'basic', 'invoke', 'add', 'main', 'take', 'run',


Topic: 21
Word Count
Weights
'key', 'json', 'value', 'datum', 'dictionary', 'state', 'push', 'data', 'name', 'get',


Topic: 22
Word Count
Weights
'product', 'float', 'integer', 'swift', 'come', 'datum', 'number', 'display', 'get', 'double',


Topic: 23
Word Count
Weights
'activity', 'context', 'override', 'public', 'final', 'intent', 'r', 'new', 'void', 'android',


Topic: 24
Word Count
Weights
'color', 'background', 'height', 'width', 'style', 'position', 'top', 'size', 'game', 'block',


Topic: 25
Word Count
Weights
'similar', 'else', 'product', 'show', 'array', 'textview', 'put', 'pi', 'override', 'code',

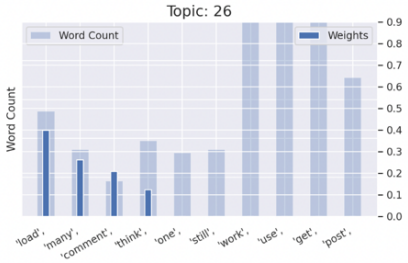

Topic: 26
Word Count
Weights
'load', 'many', 'comment', 'think', 'one', 'still', 'work', 'use', 'get', 'post',

Figure 11. Example of LDA topics (Word Count and Importance of Topic Keywords plots)

B. The exact values of LDA model performance

Number of topics	F1 score	Average predicted probability
30	0.354355172789708	0.0197681562390663
35	0.350048190190619	0.0197221543033582
40	0.366758107863478	0.0197997488380264
45	0.365744675018503	0.0196634697028316
50	0.375730045425048	0.0197664754722426
55	0.370736350021694	0.0198229539558468
60	0.383739789448851	0.0198092187506876
65	0.370498992009653	0.0198070268209410
70	0.379036760600628	0.0198111432275565
75	0.387184751631783	0.0197985222110629
80	0.377747584541062	0.0198108116932629
85	0.367248843110912	0.0198252278502389
90	0.385597923799696	0.0198371289571306
95	0.382865241309946	0.0198039415219056
100	0.373611568046873	0.0198188114255589
105	0.380253905628559	0.0198390500537000
110	0.372612545897744	0.0198473636291307

Figure 12. The exact values of LDA model performance from Figure 9

8 References

Ansari, A., Li, Y., & Zhang, J. Z. (2018). Probabilistic topic model for hybrid recommender systems: A stochastic variational bayesian approach. *Marketing Science, 37*(6), 987-1008.

Asuncion, H. U., Asuncion, A. U., & Taylor, R. N. (2010). Software traceability with topic modeling. Paper presented at the *2010 ACM/IEEE 32nd International Conference on Software Engineering, , 1* 95-104.

Bilgin, M., & Şentürk, İ F. (2017). Sentiment analysis on twitter data with semi-supervised Doc2Vec. Paper presented at the *2017 International Conference on Computer Science and Engineering (UBMK),* 661-666.

Blei, D. M., & Lafferty, J. D. (2006). Dynamic topic models. Paper presented at the *Proceedings of the 23rd International Conference on Machine Learning,* 113-120.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research, 3*(Jan), 993-1022.

Chang, W., Xu, Z., Zhou, S., & Cao, W. (2018). Research on detection methods based on Doc2vec abnormal comments. *Future Generation Computer Systems, 86*, 656-662.

Chauhan, U., & Shah, A. (2021). Topic modeling using latent dirichlet allocation: A survey. *ACM Computing Surveys (CSUR), 54*(7), 1-35.

Chen, S., Soni, A., Pappu, A., & Mehdad, Y. (2017). Doctag2vec: An embedding based multi-label learning approach for document tagging. *arXiv Preprint arXiv:1707.04596,*

Chirita, P., Costache, S., Nejdl, W., & Handschuh, S. (2007). P-tag: Large scale automatic generation of personalized annotation tags for the web. Paper presented at the *Proceedings of the 16th International Conference on World Wide Web,* 845-854.

Chiru, C., Rebedea, T., & Ciotec, S. (2014). Comparison between LSA-LDA-lexical chains. Paper presented at the *Webist (2),* 255-262.

Golder, S. A., & Huberman, B. A. (2006a). Usage patterns of collaborative tagging systems. *Journal of Information Science, 32*(2), 198-208.

Golder, S. A., & Huberman, B. A. (2006b). Usage patterns of collaborative tagging systems. *Journal of Information Science, 32*(2), 198-208.

He, J., Xu, B., Yang, Z., Han, D., Yang, C., & Lo, D. (2022). PTM4Tag: Sharpening tag recommendation of stack overflow posts with pre-trained models. *arXiv Preprint arXiv:2203.10965,*

Jeong, B., Yoon, J., & Lee, J. (2019). Social media mining for product planning: A product opportunity mining approach based on topic modeling and sentiment analysis. *International Journal of Information Management, 48*, 280-290.

Jivani, A. G. (2011). A comparative study of stemming algorithms. *Int.J.Comp.Tech.Appl, 2*(6), 1930-1938.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. Paper presented at the *Icml, , 99* 200-209.

Jurasfky, D., & Martin, J. H. (2000). An introduction to natural language processing, computational linguistics, and speech recognition.

Kataria, S., & Agarwal, A. (2015). Distributed representations for content-based and personalized tag recommendation. Paper presented at the *2015 IEEE International Conference on Data Mining Workshop (ICDMW),* 1388-1395.

Khezrian, N., Habibi, J., & Annamoradnejad, I. (2020). Tag recommendation for online Q&A communities based on BERT pre-training technique. *arXiv Preprint arXiv:2010.04971,*

Krestel, R., Fankhauser, P., & Nejdl, W. (2009). Latent dirichlet allocation for tag recommendation. Paper presented at the *Proceedings of the Third ACM Conference on Recommender Systems,* 61-68.

Lau, J. H., & Baldwin, T. (2016a). An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv Preprint arXiv:1607.05368,*

Lau, J. H., & Baldwin, T. (2016b). An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv Preprint arXiv:1607.05368,*

Le, Q., & Mikolov, T. (2014a). Distributed representations of sentences and documents. Paper presented at the *International Conference on Machine Learning,* 1188-1196.

Le, Q., & Mikolov, T. (2014b). Distributed representations of sentences and documents. Paper presented at the *International Conference on Machine Learning,* 1188-1196.

Liu, J., Zhou, P., Yang, Z., Liu, X., & Grundy, J. (2018). FastTagRec: Fast tag recommendation for software information sites. *Automated Software Engineering, 25*, 675-701.

McCormick, C. (2016). Word2vec tutorial-the skip-gram model. *Apr-2016.[Online].Available: Http://Mccormickml.Com/2016/04/19/Word2vec-Tutorial-the-Skip-Gram-Model,*

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv Preprint arXiv:1301.3781,*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems, 26*

Morin, F., & Bengio, Y. (2005). Hierarchical probabilistic neural network language model. Paper presented at the *International Workshop on Artificial Intelligence and Statistics,* 246-252.

Nandi, R. N., Zaman, M. A., Al Muntasir, T., Sumit, S. H., Sourov, T., & Rahman, M. J. (2018). Bangla news recommendation using doc2vec. Paper presented at the *2018 International Conference on Bangla Speech and Language Processing (ICBSLP),* 1-5.

Ramage, D., Hall, D., Nallapati, R., & Manning, C. D. (2009). Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. Paper presented at the *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing,* 248-256.

Rao, Y. (2015). Contextual sentiment topic model for adaptive social emotion classification. *IEEE Intelligent Systems, 31*(1), 41-47.

Somasundaram, K., & Murphy, G. C. (2012). Automatic categorization of bug reports using latent dirichlet allocation. Paper presented at the *Proceedings of the 5th India Software Engineering Conference,* 125-130.

Song, Y., Zhang, L., & Giles, C. L. (2011). Automatic tag recommendation algorithms for social recommender systems. *ACM Transactions on the Web (TWEB), 5*(1), 1-31.

Squire, M. (2015). " Should we move to stack overflow?" measuring the utility of social media for developer support. Paper presented at the *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, , 2* 219-228.

Thongtan, T., & Phienthrakul, T. (2019). Sentiment classification using document embeddings trained with cosine similarity. Paper presented at the *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop,* 407-414.

Treude, C., & Storey, M. (2009). How tagging helps bridge the gap between social and technical aspects in software development. Paper presented at the *2009 IEEE 31st International Conference on Software Engineering,* 12-22.

Truşcă, M. M. (2019). Efficiency of SVM classifier with Word2Vec and Doc2Vec models. Paper presented at the *Proceedings of the International Conference on Applied Statistics, , 1*(1) 496-503.

Wang, S., Lo, D., Vasilescu, B., & Serebrenik, A. (2018a). EnTagRec : An enhanced tag recommendation system for software information sites. *Empirical Software Engineering, 23*, 800-832.

Wang, S., Lo, D., Vasilescu, B., & Serebrenik, A. (2018b). EnTagRec : An enhanced tag recommendation system for software information sites. *Empirical Software Engineering, 23*, 800-832.

Wang, T., Wang, H., Yin, G., Ling, C. X., Li, X., & Zou, P. (2014). Tag recommendation for open source software. *Frontiers of Computer Science, 8*(1), 69-82.

Xia, X., Lo, D., Wang, X., & Zhou, B. (2013). Tag recommendation in software information sites. Paper presented at the *2013 10th Working Conference on Mining Software Repositories (MSR),* 287-296.

Zhao, F., Zhu, Y., Jin, H., & Yang, L. T. (2016). A personalized hashtag recommendation approach using LDA-based topic model in microblog environment. *Future Generation Computer Systems, 65*, 196-206.

ZhengWei, H., JinTao, M., YanNi, Y., Jin, H., & Ye, T. (2022). Recommendation method for academic journal submission based on doc2vec and XGBoost. *Scientometrics, 127*(5), 2381-2394.

Zoghbi, S., Vulić, I., & Moens, M. (2016). Latent dirichlet allocation for linking user-generated content and e-commerce data. *Information Sciences, 367*, 573-599.