ANALYSIS OF THE IMPACT OF TAGS ON STACK OVERFLOW QUESTIONS

by

Von Ithipathachai

May, 2022

Director of Thesis: Mark Hills, PhD

Major Department: Computer Science

User queries on Stack Overflow commonly suffer from either inadequate length or inadequate clarity with regards to the languages and/or tools they are meant for. Although the site makes use of a tagging system for classifying questions, tags are used minimally (if at all). To investigate the impact of tags in the quality of results returned by the queries, in this research we propose a new query expansion solution. Our technique assigns tags to queries based on how well they match the queries' topics. We evaluated our technique on eight sets of queries categorized by overall length and programming language. We examined the retrieval results by adding varying numbers of tags to the queries, and monitored the recall and precision rates. Our results indicate that queries yield considerably higher recall and precision rates with extra tags than without. We further conclude that tags are a particularly effective means of enhancement when the original queries do not already return sufficient yields to begin with.

ANALYSIS OF THE IMPACT OF TAGS ON STACK OVERFLOW QUESTIONS

A Thesis

Presented to The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Computer Science

by

Von Ithipathachai

May, 2022

ANALYSIS OF THE IMPACT OF TAGS ON STACK OVERFLOW QUESTIONS

by

Von Ithipathachai

APPROVED BY:

DIRECTOR OF THESIS:

Mark Hills, PhD

COMMITTEE MEMBER:

Venkat Gudivada, PhD

COMMITTEE MEMBER:

Rui Wu, PhD

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:

Venkat Gudivada, PhD

DEAN OF THE GRADUATE SCHOOL:

Paul J. Gemperline, PhD

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Whenever we have a problem, but do not have the solution to it ourselves, we frequently turn to search engines to find one. These search engines often require users to input their own queries to act as search criteria. Unfortunately, due to not fully understanding how to express their needs, user queries may sometimes be underdeveloped and insufficient for finding their answers, necessitating further enhancements on the part of the system so that they can more effectively retrieve satisfactory ones. Typically, when a search engine receives a query from a user, it pre-processes the query and divides it into distinct terms, then refers to those terms to retrieve documents that it judges to be relevant. However, retrieving relevant documents is challenging for underdeveloped queries, which may return results that are overly broad. In addition, the documents retrieved may not even satisfy the user's needs, particularly when the user doesn't know the precise terminology for what it is he/she is looking for. Automatically expanding the query with more terms can remedy the problem, but not just any terms can be chosen if improvement is to be noticeable and misguided users are to be put back on the right track.

Stack Overflow is a popular website to refer to for help with difficult coding problems [6, 9, 16, 18, 15, 13]. It is frequented by programmers in training and seasoned software developers alike, and presents a classic example of this issue. Among the site's numerous features is its own search engine [1], to which users can feed queries in order to find answers that might solve their problems, often attached to question threads from users experiencing similar problems. However, like with all search engines, not all users know how to write queries that are strong enough to make optimal use of it [15, 13]. There are a couple of notable recurring problems with queries on

Stack Overflow. One problem is that queries are often short, giving the search engine relatively little material to work with for finding matching documents. Another problem is that queries are too general, being worded vaguely or failing to specify whichever of the many programming languages, libraries, and/or other tools in existence their users are struggling with. Phrases like ”nested for loop”, ”linux apache2 flask folium”, ”encapsulate executable”, ”two html drop down list event”, etc., are just a few examples of these kinds of deficient queries that will probably fail to help users find useful answers, taken from the Stack Overflow dataset itself.

A number of studies [17, 14, 9] have already been performed on possible ways to fix these inadequate Stack Overflow queries. These typically entail automated query expansion/reformulation techniques that derive extra terms to add from different sources depending on the study. To the best of our knowledge, no studies appear to have incorporated the tags attached to questions and answers into the process. In addition to its search engine, Stack Overflow also includes its own tagging system [1], which enables questions to be tagged for easier classification and location by the search engine. The search engine even allows users to search for posts directly by tags [1], though it is difficult to articulate a user problem to search for entirely through tags alone. We feel that these tags are underutilized and have potential utility for query improvement.

Therefore, to address the currently limited usability of tags for search purposes and show how they can be used to strengthen user queries, in this paper, we present a query expansion solution that incorporates both the base tags from questions and additional tags derived from LDA (Latent Dirichlet Allocation) topics into user queries for the purposes of retrieving answers[1]. To test this approach, we perform an experiment on questions and answers from the Stack Overflow dataset for multiple programming languages, in which initial queries composed of question titles and their associated tags are assigned topics and used to retrieve answer documents, with gradually more tags from the assigned topics being added with each experiment iteration. The results of this experiment demonstrate that the extra tags chosen and added to queries always result in the queries having greater levels of document recall. In addition, the results also demonstrate that the addition

[1]The source code is available at https://github.com/24karatDVNO/StackOverflow-Query-Improvement-with-Tags

of these tags leads to increased precision, with the top most relevant documents having greater cosine similarity with their queries. Our approach is thus shown to enhance query results, which should lead to more satisfied users. The rest of this paper explains the process and results in greater detail.

The remaining sections are as follows. Section 2 provides additional background information from other studies using Stack Overflow for query expansion and/or question analysis, as well as from other related studies. Section 3 explains our approach, including our reasoning for opting to use LDA for tag generation. Section 4 describes details of the experiment used to evaluate our approach, including the research questions used to guide the experiment. Section 5 presents the results of our study. Section 6 discusses what the results indicate including their practical implications. Section 7 discusses possible threats to the validity of our study, and Section 8 discusses conclusions and future work.

Chapter 2

BACKGROUND AND RELATED WORK

2.1 Background

This section of the paper discusses pre-existing studies with the Stack Overflow dataset that are similar to ours. These primarily include other studies related to query expansion and/or reformulation. They also include studies related to the analysis of Stack Overflow questions and how useful they may be considered for users.

2.1.1 Query Expansion/Reformulation Studies

Over the past decade, several studies have been conducted to investigate ways to improve queries in terms of either the quality of retrieved code snippets [17, 14] or the clarity and/or grammatical correctness of the queries themselves [9]. However, none of them discuss retrieval results in terms of non-code text content. Most importantly, they also make minimal use or no use at all of Stack Overflow tags for enhancement in the way that our study does.

A study by Nie et al. [17] describes a method of Query Expansion based on Crowd Knowledge (QECK), i.e. document-extracted software development knowledge useful for answering questions. The documents, in this case question-approved answer pairs, are used as pseudo-relevance feedback for identifying useful terms to add to a query based on its initial ones. Jason Liu et al. [14] describe a method of Neural Query Expansion (NQE) which works in conjunction with Facebook's pre-existing Neural Code Search (NCS) approach that retrieves relevant code snippets from Stack Overflow for natural language queries. After extracting the query's keywords, NQE is

able to predict new keywords to add based on their co-occurrence in the document corpus with the original keywords. Both of these studies demonstrate the effectiveness of their solutions through use in conjunction with pre-existing code search algorithms.

In another study, Cao et al. [9] describe a deep learning-based query reformulation solution which makes use of Stack Overflow query logs for improving deficient user queries. The solution extracts existing original-reformulated query pairs from the query logs and derives reformulation patterns from them. Using these patterns, it can then receive an original query to fix and generate multiple possible improved versions of that query to replace the original. The study puts its solution to the test against common evaluation metrics and other enhancement tools and performs beyond their standards to varying degrees. Here, improvement is measured primarily in terms of typographical error correction, increase in query clarity, and reduction of time spent on manual query reformulation, rather than what the queries themselves return.

2.1.2 Stack Overflow Question Analysis

In addition to these query enhancement-related studies, numerous studies on the analysis of Stack Overflow questions themselves have also been performed within the past decade. These studies discuss question analysis in terms of either the questions themselves [4, 7, 10] or the code snippets they contain [11, 16]. Although our solution does not take question quality into account for topic generation, it will be helpful to consider for future revisions.

The more general question quality studies, such as the ones by Arora, Baltadzhieva, Correa, etc. [4, 7, 10] evaluate quality based on numerous different factors. The factors found to reliably indicate whether or not questions are of good quality include (but are not limited to) their composition [4, 7], the scores they have received [4, 7], the number of answers they have received [7], and their similarity to other questions that have received high/low scores [4]. It is also important to consider whether or not a question has been closed, as questions can be closed if they have serious flaws such as being a duplicate, off-topic, subjective, overly-localized, or not real questions [4, 10]. The more code snippet-centric studies, such as the ones by Nasehi, Dujin, etc. [6, 9, 16, 18] ac-

knowledge the fact that Stack Overflow is heavily relied upon by programmers of all skill levels as a source of helpful code examples and therefore reasonably conclude that good quality questions will often include at least some amount of reusable code. As such, they include code quality among the other factors that determine whether or not a question is good. Some metrics used in these studies for measuring code quality include (but are not limited to) readability/conciseness [11, 16], style compliance [11], and number of constructs [11].

2.2 Related Work

The studies presented in this section do not fit neatly into either query expansion/reformulation or question analysis. However, they either use similar methodology to our own to answer different research questions or offer additional insights into how the Stack Overflow user experience can be improved. Because of this, the knowledge they offer may also be useful to take into consideration for future revisions of our solution.

2.2.1 Studies with LDA Topic Modeling

Allamanis and Sutton [3] performed a study in which they used LDA to classify Stack Overflow questions according to views of concept (concepts being subjects such as ”applets”, ”games”, ”languages”, etc.) and type of information (such as how to fix a problem or locate helpful resources). The resulting topics generated for these viewpoints were able to provide useful insights, such as which questions lend themselves best to the use of code snippets, the universality of questions between programming languages, how the orthogonality of different tools and technologies can be evaluated, and how concepts and types can be connected to each other to solve user problems. Zou et al. [20] also performed a study with LDA in which they analyzed Stack Overflow’s non-functional requirements (NFRs) to develop a more comprehensive understanding of the site’s development activities. Much like our study, they extracted questions and bodies from the Stack Overflow dataset and used them to generate development-related topics. The topics were then labeled with whichever NFRs fit them the best, or with none at all if no NFRs fit. After validating

the topics against another corpus, they found that the developers focused most greatly on usability and reliability, with maintainability and efficiency being secondary concerns.

2.2.2 Other Studies

Yang et al. [19] performed a study analyzing Stack Overflow code snippets and their usability rates. Like our study, this one also categorized extracted code snippets by programming languages (Python, Java, C#, and JavaScript). Snippets could be classified as parsable and/or either compilable depending on the language. Code snippets for Java and C# were found to have significantly lower usability rates by comparison, even after additional repairs and the removal of one-word snippets.

Bhat et al. [8] conducted a study in which they investigated potential factors influencing response time to new Stack Overflow questions. Tag-related factors included the amount of popular tags used, the specificity (co-occurrence rate) of the tags, and the quantity of subscribers to those tags. These tag-related factors were found to be more effective in eliciting fast response times than non-tag-related factors such as length of the body and titles, the amount of code and images present in the question, grammar and punctuation, and even whether the question was posted on a weekend. This study further reinforces our findings that the use of tags plays an important role in finding answers and also notes at the time of its writing the lack of exploration of tag-based features from previous studies.

Sengupta and Haythornthwaite [18] conducted a study in which they analyzed comments on Stack Overflow posts in order to gain an understanding of their contribution to user learning. They found that comments fell into a number of different categories, (improvements, code and/or explanation, limitations, etc.), with each category serving some sort of purpose contributing to both learning and community-building. Like tags, comments were noted not to have received much attention as subjects of analysis, and subsequently demonstrated to be a valuable asset in enriching discussions.

Li et al. [12] performed a study with query reformulation which was similar to that of Nie et. al.

in that it made use of crowd knowledge for query reformulation. Their solution focuses on locating high-quality open-source software, using list and detail pages for software featured on Oschina as its documents. Although the solution does not use documents from Stack Overflow, it does use the site's tags for building a lexical database with which is used to aid in locating terms in documents that may be related to the user's query to some degree.

Mingwei Liu et al. [15] propose their own solution for addressing the knowledge gap between user needs and query construction. Their approach for searching Stack Overflow questions entails an initial keyword-based query being elaborated upon with multi-faceted categorization from the candidate questions retrieved by the query with the use of natural language processing and data mining. The approach is implemented as an interactive tool called MFISSO (Multi-Faceted and Interactive Searching of Stack Overflow) which facilitates iterative refinement of search results until suitable questions are found. Their approach uses their own defined facets of Stack Overflow questions, which includes, among other things, topics but not tags.

Jiakun Liu et al. [13] carried out a study in which they sought to characterize developer search activities on Stack Overflow. Their analysis made use of data from real-world search logs, which few contemporary query reformulation solutions had at the time. Some of their findings included immediate reformulation of an immediately preceding query as the most common search activity, that the majority of Stack Overflow reformulations adhered to known general reformulation strategies, and that a small but noticeable portion of search sessions searched only for code snippets without specifying a programming language or other key element.

Chapter 3

APPROACH

3.1 Overview of the Approach

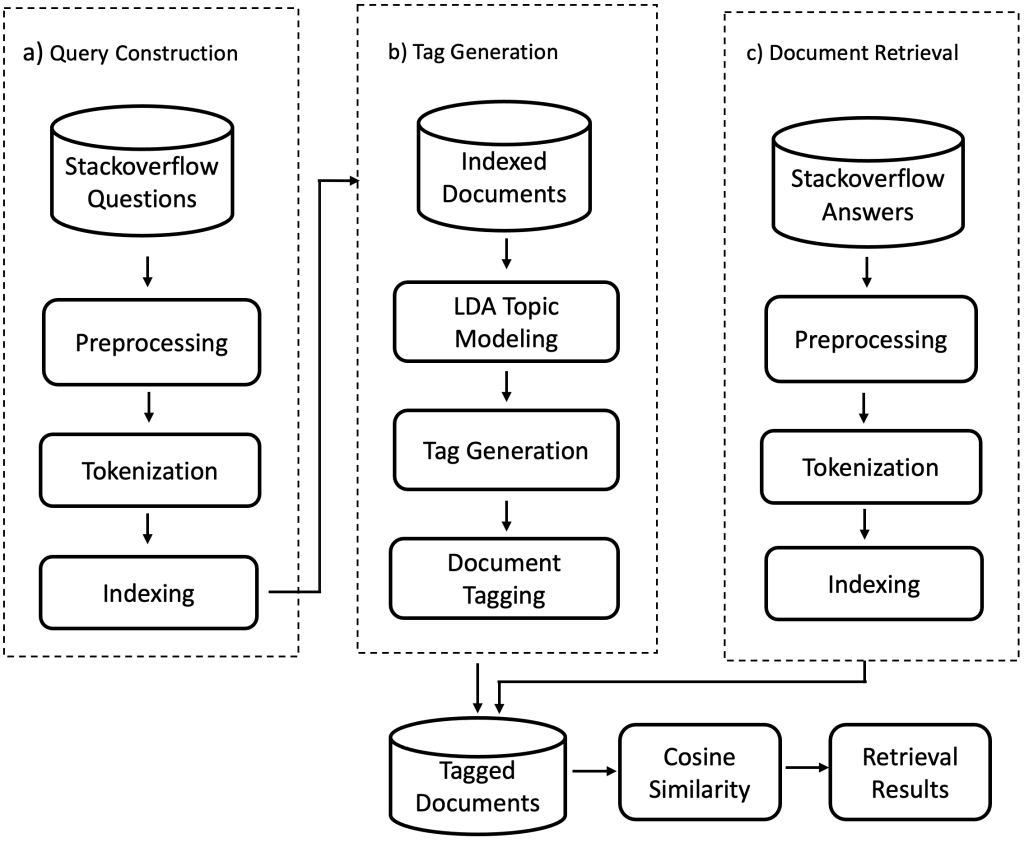

Figure 3.1: An Overview of the Framework

Our query expansion solution is divided into four basic phases, illustrated in Figure 1. First, initial queries are constructed from Stack Overflow question titles and their initial tags. Next, LDA topics containing additional tags are generated from question bodies using the Gensim library

for Python; these topics are then assigned to the queries that are determined to fit them the best. In addition, answer documents are generated (also from the Stack Overflow dataset), which can potentially be matched to queries. Finally, the queries are used to retrieve matching answers, and the compatibility of the queries with each of their found answers is measured by their cosine similarities. The rest of this section elaborates on each step in greater detail.

3.2 Query Construction

In order to prepare the queries for further analysis, the raw questions and tags retrieved from the Stack Overflow dataset need to be preprocessed. Preprocessing entails normalizing text data such that non-alphabetic characters and other unwanted elements are not present and all of the data follows a consistent format. Our standard preprocessing algorithm for questions involves lowercasing every word, removing punctuation, isolating distinct words from one another with NLTK's word tokenizer, stemming words to basic forms with NLTK's Snowball stemmer, and removing stop words (extremely common words that carry insufficient meaning or value to be worth consideration). Question titles and bodies are retrieved separately from each other and used for different purposes; here, the titles and their initial tags will form the initial queries to use for retrieval. The titles are able to be preprocessed with the standard algorithm, but their tags need a special version due to how they are stored in the Stack Overflow dataset. Each of these tagsets is stored as a single large string, with tags being separated by vertical bars, e.g. "python — matplotlib — label — legend". Since the tags themselves are already lowercased, each tagset string simply needs to be broken apart into individual tokens at the vertical bars, with the tags themselves being stemmed afterwards.

3.3 Tag Generation

3.3.1 Latent Dirichlet Allocation

Adding extra tags to queries helps make them less vague and provides the search engine with more clues about what kind of documents to search for. However, manually choosing extra tags in sufficient quantities to add to a query is highly time-consuming. To make matters worse, these manually decided tags can vary widely based on what different individuals feel is best, which can lead to inconsistencies in tagging logic [5]. To address this issue, we opt to use LDA to guide and partially automate the process. The question bodies contain enough information that we can gather which ones fall under the same or similar topics as well as what those topics might be. In addition, LDA is also known to perform well for a variety of tasks, such as classification [3] and discovering trends [20].

When LDA generates topics for a document, each topic z for the document d is described based on the probabilities of certain terms occurring in the document, given by the formula

$$P(t_i|d) = \sum_{j=1}^{Z} P(t_i|z_i = j)P(z_i = j|d)$$

where $P(t_i|d)$ is the probability of the ith term for the document and z_i is a latent topic for that document. $P(t_i|z_i = j)$ is the probability of the ith term of the latent topic occurring within another topic j, while $P(z_i = j|d)$ is the probability of a term shared by the other topic and the document being selected for the latent topic [5]. Every document receives latent topics, with each one having some percentage likelihood of contributing to the overall document. In this context, the documents receiving LDA topics are the initial queries, which will be expanded so that they include both the initial tags of the question titles taken directly from the Stack Overflow dataset and extra tags from discovered topics.

3.3.2 Topic Generation

Since the question bodies will be used for generating topics, special measures must be taken when preprocessing them. One such measure includes the removal of all program code and associated code and formatting tags from the bodies. We opt to remove program code in preprocessing due mainly to the inherently idiosyncratic nature of variable names and naming conventions making them highly unlikely to constitute recurring and noteworthy items of discussion. In addition, a substantial amount of Python code snippets from Stack Overflow have been found to suffer from style violations, making them poorly suited for use in topic generation [6]. To preserve neutrality in our study and protect the integrity of the results, program code is removed from question bodies for all languages.

The other measure that has to be taken is a broader definition of stop words, i.e. any words like ”code”, ”program”, ”bug”, etc. which occur frequently enough to carry little meaning on their own in the context of programming, in addition to the typical stop words like ”my”, ”it”, ”the”, etc. used merely for purposes of grammatical correctness. Our solution opts to treat the 2% most frequently occurring words in the question bodies as stop words; a higher percentage of frequently occurring words being removed is more likely to result in potentially useful words for topic identification being lost. Once the question bodies have finished preprocessing, they are ready to be used for generating LDA topics. Each topic consists of 20 tags (terms) along with their probabilities of appearing together expressed as decimal values between 0 and 1. Furthermore, to make query tagging, answer retrieval, and cosine similarity calculation easier later, the terms and values are also isolated from one another.

3.3.3 Query Tagging

Queries to use for answer retrieval are selected at random from the preprocessed question titles available. Each chosen title is then combined with its respective tagset to form an initial query that will be expanded later. Next, each query is manually assigned a topic containing the extra tags it will be expanded with. When possible, topics were assigned to queries that shared at least

one word with them, with the understanding that the query sharing a tag with a topic indicates that the topic's other tags are also usually relevant or searched for alongside the shared tag. In cases where the query shares no tags with any topics, it is instead assigned a topic containing tags that are thought to be similar in meaning and/or appearance.

3.4 Document Construction

Like the questions and their tags, the answers are also taken from the Stack Overflow dataset. To ensure better compatibility with the queries used to retrieve them, they are preprocessed with the exact same methods used for the question bodies; like the bodies, the answers sometimes contain code snippets and related formatting artifacts that will interfere with accurate retrieval and thus need to be removed. The unaltered answers are also written to text documents identified by their ID numbers so that they can be quickly referred back to when analyzing the queries' retrieval results.

3.5 Document Retrieval

Within a corpus of documents, each distinct word has term frequencies and a document frequency. The document frequency for a term is simply the number of documents it appears in, while its term frequencies are how many times it appears in each document where it is present. Together, they form tf-idf weights for each term in a document, indicating how important each term is in the context of that document. A term's tf-idf weight is simply equal to its term frequency multiplied by its inverse document frequency. The latter is itself derived from the document frequency using the formula

$$idf = log_2(x/df)$$

where df is the document frequency, idf is the inverse document frequency, and x is the total number of answer documents present. Once a document has been weighted, its vector length can be calculated. The vector length of a document is equal to the square root of the sum of the squares

of all of its tf-idf weights. When calculating cosine similarity values, tf-idf weights and vector lengths must be calculated for both the query and the answer.

Cosine similarity is a commonly used measure of how similar a pair of documents are to each other. It represents each document as being part of the same vector space, with document vectors that are closer to each other in the space (based on the cosine of the angle between them) being more similar. We opt to use this measure as our main metric for document relevance because queries tend to retrieve many documents and directly comparing every document to its query is a highly time-consuming and error-prone process. Cosine similarity values are calculated for every pair consisting of a query and one of the answer documents it retrieves. This also requires tf-idf weights and vector lengths for both the query and answer to be found.

When answers are retrieved for a query, any answer that shares at least one term with the query is considered a match, however minimal the resemblance may be. While these loose requirements may sound problematic in theory, they are acceptable in practice because the most relevant answers will share many more terms with the query than just one, and search engine users will realistically stop considering documents returned past a certain quantity threshold as they gradually decrease in relevance. For this reason, when determining how cosine similarity values for each query-answer pair have changed in quality as more extra terms are added to the query, we only consider at most the top 50 documents in the likely event that more than 50 documents are retrieved.

With tf-idf weights and vector lengths in hand, the inner product of the query and the document can be found. This is equal to the sum of the squares of the weights of every term they share; terms present in either the query or the document but not both have weights of zero. Finally, the cosine similarity value is equal to the inner product divided by the product of the query and document's vector lengths. The closer a cosine similarity value is to 1, the better the query and document match.

Chapter 4

EMPIRICAL STUDY

To evaluate our proposed technique, we performed an empirical study considering the following research questions:

- RQ1: Is our proposed technique effective in identifying relevant answers for Stack Overflow queries?
- RQ2: To what extent does our proposed technique improve the performance of Stack Overflow queries as measure by precision and recall?

4.1 The Dataset

The Stack Overflow dataset is publicly accessible through Google Cloud using Google's BigQuery API and includes detailed tables of Stack Overflow question and answer posts. Often included in the tags for question posts is the programming language and/or related tools for which the question is intended. Since programming languages are not always designed with the same purposes in mind, we felt it necessary when preparing our experiment to organize the questions by the programming languages they are intended for. We performed the experiment on four languages: Python, Java, C++, and JavaScript, which are among the most commonly used and therefore provide many questions to use.

4.2 Variables and Measures

4.2.1 Independent Variables

The independent variables for this experiment are query length, programming language, and number of tags added. In addition to the four languages mentioned above, queries were also organized by preprocessed length of question titles without any tags at all, even their initial ones. Queries comprised of titles eight words or longer after preprocessing were considered long, while all others were considered short. In total, the experiment was performed on eight different sets of queries, with each query set consisting of either long or short queries for each language. Each experiment iteration increases the number of extra tags added to each query from its assigned topic, starting with no extra tags and incrementing by five every iteration until twenty tags have been added.

4.2.2 Dependent Variable and Measures

The dependent variables for this experiment are precision and recall. Precision is typically defined as a ratio of retrieved documents that are relevant to all documents that are retrieved, while recall is typically defined as a ratio of number of documents retrieved to total number of documents in the corpus. When discussing our results, we opt to use modified definitions of these two due to the probabilistic rather than deterministic nature of the problem, as well as the lack of true/false labeling. For precision, we opt to select the top 50 most relevant documents by cosine similarity value that are retrieved by a query and calculate the average of their values to determine the overall precision rating for the query. This is because our definition of what constitutes a ”relevant” query is relatively loose, resulting in queries often retrieving hundreds of documents with only a small fraction likely to be examined by the typical search engine user. For recall, we opt to simply state the number of documents retrieved by the query out of all of the documents in the corpus, since the typical user will think of recall in terms of the raw number of answers retrieved and not the percentage out of all answers that is retrieved.

4.3 Data Collection and Experimental Setup

From the dataset, we retrieved 100,000 question posts each for Python, Java, C++, and JavaScript, with questions for each language being retrieved based on the presence and/or absence of specific language tags. Retrieving Python and C++ questions was fairly straightforward, but special care needed to be taken when retrieving the Java and JavaScript questions. Java and JavaScript are often conflated with one another, and the latter language is also often referred to as ”js” for short. Therefore, while the Python and C++ questions only needed to include ”python” or ”c++” respectively in their tags, Java questions were required to include ”java” but not ”javascript”, and JavaScript questions could have either ”javascript” or ”js” in their tags. We also retrieved 200,000 answer documents to use as potential answers to retrieve for queries; this set of answers was used for all experiment iterations.

The experiment was carried out using a Python program running on a Windows 10 machine with Python 3.9.5 installed. This program performs a single experiment iteration per execution and requires three integer command line parameters to be entered. The first parameter indicates which language the program will be run for (0 for Python, 1 for Java, 2 for C++, or 3 for JavaScript). The second parameter indicates how many extra tags needed to be added to each query, i.e. 0, 5, 10, 15, or 20 extra tags. The third parameter indicates whether long (0) or short (1) queries will be used. In addition, because the topics LDA produces when provided data are not always consistent, a separate program was used to produce files containing topic data in advance. These files are then read by the main program and used in all iterations to produce more consistent results. The specific extra tags to add to each query set are also hardcoded into the main program for this reason.

When an experiment iteration is finished executing, for each query a list of every retrieved query-answer pair along with its cosine similarity value is produced. The results for each query are sorted in descending order of the latter, with average cosine similarity for that query’s top 50 results appearing at the bottom of its results list. There are eight sets of results in total for the experiment, all of which can be stably replicated given pre-generated topic data and extra tags to add chosen in advance.

Chapter 5

DATA AND ANALYSIS

RQ1 Analysis: Our first research question investigates whether the proposed technique effective in identifying relevant answers for Stack Overflow queries or not. Tables 5.1 and 5.2 show average recall and precision values respectively for each query set given some number of extra tags added. Long queries are those for which the question titles used to create them are eight words or longer after preprocessing, while short queries are those for which they are seven words or shorter afterwards. For recall, Table 5.1 shows clear evidence of significantly improved retrieval rates with extra tags added, as every five extra tags for all of the queries in each set results in a consistent increase of roughly 300-500 documents retrieved per experiment iteration. This is to be expected, as every extra term broadens the search criteria to provide more reach. For precision, Table 5.2 shows less dramatic increases in precision with each increment of five extra tags; average cosine similarities initially show increases of about 0.05 to 0.09 from 5 to 10 tags added, then increases of about 0.03 to 0.04 for later increments. Though the changes in precision are not as impressive, the most important thing to note from these results is that adding more extra tags always results in an increase for both precision and recall values, meaning that adding more tags to queries causes them to retrieve results that are both more numerous and better matches. By the time 20 tags have been added, all query sets consistently retrieve about 1,500 documents (versus about 100 without them) and average cosine similarities of about 0.8 (versus about 0.4 to 0.52 without them). Thus, for the purposes of our research, we can say that our technique is effective in identifying relevant answers for Stack Overflow queries.

Query Set	# Extra Tags Added			
	5	10	15	20
Python (Long)	343.1	686.2	1096.4	1461.5
Python (Short)	350.1	773.2	1168.2	1444.4
Java (Long)	509.8	945.4	1296.7	1646.2
Java (Short)	522.3	890.2	1279.9	1585.3
C++ (Long)	411.4	795.5	1172.2	1456.4
C++ (Short)	280.6	701.4	1140.3	1577.6
JavaScript (Long)	346.4	688.9	1085.9	1397.9
JavaScript (Short)	342.4	793.6	1123.7	1428.7

Table 5.1: Average quantities of documents retrieved by each query set given some number of extra tags added to each query.

Query Set	# Extra Tags Added			
	5	10	15	20
Python (Long)	0.633	0.723	0.761	0.795
Python (Short)	0.661	0.735	0.777	0.805
Java (Long)	0.704	0.760	0.787	0.822
Java (Short)	0.714	0.769	0.803	0.832
C++ (Long)	0.653	0.747	0.782	0.807
C++ (Short)	0.639	0.730	0.770	0.792
JavaScript (Long)	0.694	0.751	0.792	0.822
JavaScript (Short)	0.665	0.741	0.779	0.809

Table 5.2: Average cosine similarity of query-answer pairs for each query set given number of extra tags added to each query.

Query Set	Extra Tags Added?		% Improvement
	No	Yes	
Python (Long)	112.4	896.8	698
Python (Short)	116.5	934.0	702
Java (Long)	135.5	1099.5	711
Java (Short)	111.2	1069.4	862
C++ (Long)	107.9	958.9	789
C++ (Short)	88.9	925.0	940
JavaScript (Long)	125.4	897.8	602
JavaScript (Short)	91.3	922.1	910

Table 5.3: Average quantities of documents retrieved by each query set with vs without extra tags extra tags.

Query Set	Extra Tags Added?		% Improvement
	No	Yes	
Python (Long)	0.524	0.728	39
Python (Short)	0.431	0.745	73
Java (Long)	0.494	0.768	55
Java (Short)	0.471	0.780	65
C++ (Long)	0.395	0.748	90
C++ (Short)	0.420	0.747	89
JavaScript (Long)	0.521	0.765	47
JavaScript (Short)	0.403	0.748	86

Table 5.4: Average cosine similarity of query-answer pairs for each query set with vs without extra tags.

RQ2 Analysis: Our second research question investigates the extent to which our proposed technique improves the performance of Stack Overflow queries as measured by precision and recall. Tables 5.3 and 5.4 show average recall and precision values respectively for each query set without any extra tags added versus with extra tags; the latter is the average of all the values obtained for each experiment iteration as seen in Tables 5.1 and 5.2. For recall, the baseline performance observed without the addition of extra tags to queries is incredibly poor, with all query sets

returning only roughly 100 documents on average. When extra tags are added, these amounts increase drastically, ranging anywhere between 600% to 950% more documents being returned than without. For precision, the baseline performance without extra tags is also rather mediocre, with average cosine similarity values often falling below 0.5, indicating poor matches that are unlikely to be relevant. With extra tags, these values increase at broadly varying rates from about 40% to 90%, but always exceed 0.7, indicating that the documents retrieved are likely to at least be passable matches with a fair degree of relevance to the user. As before, the results clearly indicate that the addition of extra tags always causes queries to return both more numerous and more relevant results. Thus, we can say that our technique offers a significant level of performance improvement for identifying relevant answers for Stack Overflow queries.

Results by Category: In general, our results indicate that adding extra tags to queries affects their search performance independently of programming language or length. Although some data to the contrary exists, it is inconclusive and insufficient for definitively proving bias towards a particular language or length class.

5.0.1 Programming Language

When viewed in terms of programming languages, all four languages display roughly equal rates of improvement for recall and precision. For raw recall and precision rates, Python, C++, and JavaScript continue displaying similar results to one another, but Java achieves higher rates than all three, with Java queries returning 200 more documents on average and having average cosine similarity values 0.2 to 0.4 points higher given a nonzero amount of extra tags added.

The higher numbers for Java queries can most likely be attributed to overlap between Java and JavaScript answers. Due to their similar names, they are often conflated with one another or otherwise believed to be directly related when in reality neither is the case. Because of this misconception, it is possible that some number of questions intended to be JavaScript-related were wrongly tagged as being Java-related, resulting in some answers retrieved by Java queries actually being for JavaScript. When selecting questions to use as queries for the Java query sets, it was

necessary to filter out questions tagged as being for JavaScript in order to ensure more trustworthy results, i.e. selecting questions with ”java” in their tags, but not ”javascript” or ”js”. However, the possibility of some Java questions being erroneously tagged as such remains, and may explain the relatively high recall and precision values obtained for the Java query sets. As such, the higher values for Java as seen in the tables should not be taken to mean that queries for Java benefit more from our solution than queries for other languages.

5.0.2 Long vs. Short Query Sets

Performance differences between long and short query sets for the same language are generally negligible, although one type may outperform the other at different extra tag intervals. For raw recall rates, the difference is only a few documents at the lowest and usually does not exceed 100 documents at the highest. For raw precision rates, the difference in cosine similarity values is usually between 0.01 and 0.03 points. However, for both recall and precision rates, unusually high performance differences between long and short query sets can be observed at specific extra tag intervals. For recall rates, the two C++ query sets display high performance differences at 5 and 20 tags added, and the two JavaScript query sets display high performance differences at 10 tags added. For precision rates, the Python and JavaScript query sets display high differences at 0 tags added, and the C++ query sets display high differences at 5 tags added.

The explanation for these abnormalities in the data lies in the performance of individual queries within the sets. Due to certain terms being more commonly encountered than others, performance changes for individual queries moving from one interval of extra tags added to the next may vary widely depending on the tags that have been added. Thus, performance differences between long and short query sets may become especially high if the queries in one set mostly grow quickly in performance and the queries in another set mostly grow slowly in performance. Regardless of whether the performance differences between query sets of a certain language in the same interval are large or small, they ultimately do not detract from the consistent upward trends in recall and precision rates seen as queries receive more extra tags.

Chapter 6

DISCUSSION

6.1 Performance for Individual Queries

Of the 80 total queries used in the experiment, 30 queries (37.5% of all queries) failed to retrieve at least 50 documents prior to receiving extra tags; of those 30, 15 queries (18.75% of all queries) failed to retrieve any documents. The reasons for certain queries returning zero or extremely few documents are unknown, but such queries are highly likely to have contributed to the very low recall and precision rates observed for all query sets with no extra tags added. Conversely, as more extra tags were added to each query, they would return increasing numbers of documents that had cosine similarity values of 1.0. Such documents would often be comprised mainly of code with very little non-code text or simply be very short in length. Even without taking these 1.0 cosine similarity documents into account, each addition of 5 extra tags to queries would consistently result in the queries retrieving some number of new documents with relatively high cosine similarity values which the previous iteration of extra tags failed to retrieve. Thus, regardless of whether query-document pairs with 1.0 cosine similarity values are to be accounted for, extra tags are always shown to have positive effects on recall and precision when added to queries.

6.2 Effects of Chosen Topic on Retrieval Results

The nature of the answer documents retrieved by a query appears to be heavily dependent on the topic it is assigned. This is best observed in queries that fail to retrieve any documents at all when they are not given any extra tags, such as queries 6 and 7 in Table 6.1. A few pairs of such queries

that were used in the experiment ended up returning identical results when they did receive extra tags, and continued to retrieve the same documents throughout their usage period. These queries had been assigned the same topic before the experiment began, as said topic was judged to fit them both. With these observations in mind, the possibility arises that manually assigned topics may cause the documents their queries retrieve to deviate from the intent of the queries themselves.

6.3 Limitations of Adding Extra Tags

Query ID	Topic ID	# Docs Returned		Avg. Cos. Sim.	
		+0 tags	+5 tags	+0 tags	+5 tags
1	47	71	440	0.547	0.711
2	35	32	270	0.423	0.601
3	12	221	668	0.598	0.720
4	58	123	645	0.619	0.733
5	62	232	487	0.631	0.677
6	40	0	216	0	0.627
7	59	0	53	0	0.373
8	13	0	447	0	0.684
9	20	375	694	0.705	0.753
10	3	25	194	0.424	0.647

Table 6.1: Data gathered for long C++ queries with no extra tags added versus with five extra tags added.

Table 6.1 shows retrieval results for each of the long C++ queries with 0 and 5 extra tags added. The queries themselves and the extra tags added to them can be found in Tables 6.2 and 6.3. While extra tags have been observed to be useful for enhancing the recall and precision rates of queries, there are some scenarios where they may not be necessary and could in fact be a detriment. As noted in the previous subsection, adding too many extra tags to a query can cause the retrieved documents to more closely reflect the topic they came from and not the original query. This can be especially harmful if the query is mistakenly assigned the wrong topic, similarly to how Java queries can have abnormally high recall and precision rates if they mistakenly include JavaScript tags in addition to regular Java tags. To mitigate this issue, potential solution could be to enlist a human expert to review the extra tags assigned to queries and verify that they fit adequately.

Table 6.2: Original Long C++ Queries

Query ID	Query
1	How to connect to a ZMQ_REQ socket to another endpoint, once sending to the original address has timed-out?
2	is there a way to disable message map handlers during runtime, in mfc?
3	Core profile vs version string? Only getting GLSL 1.3/OGL 3.0 in mesa 10.0.1
4	boost::iostreams::copy - sink - ENOSPC (No space left on device) error handling
5	Is emitting a signal from C++ to QML for reading a Q_PROPERTY a synchronous event?
6	Best way to store string of known maximum length in file for fast load into vector<string>in C++
7	Overloading operator<<for template class. Impossible to get private members even with friend keyword
8	Grabbing file name from a command line argument in C++ console app
9	I am getting a 'base(const std::ios_base)' is private error in my code. What is wrong with it?
10	Why does GetProcessImageFileName return null instead of the address of the process?

Table 6.3: Experiment Objects and Associated Data.

Query ID	Extra Tags
1	portion, discov, challeng, dispatch, capabl
2	str, plus, simpler, ultim, blob
3	illustr, qstring, factor, restart, expens
4	intersect, lack, vec, stage, uniform
5	emit, belong, chanc, sphere, among
6	bracket, sizeof, worri, concret, repli
7	getter, setter, pod, toolchain, crc
8	scroll, cpprefer, impli, grab, highest
9	iostream, deleg, filesystem, deploy, abort
10	act, smallest, dereferenc, proxi, worth

Recall and precision rates for queries that receive no extra tags are also important to consider. Although some queries used in the experiment returned fewer than 50 documents with no extra tags added, others were able to return as many as 200 with no extra tags. Although these recall rates are lower than they would be without extra tags, they can still be acceptable in some cases, especially when users have limited time budgets and cannot look at every document their query retrieves.

Additionally, although it was not implemented in our experiment program due to time constraints, complexity issues, and a desire to focus on the use of LDA topics for query enhancement, substituting base tags from queries that don't appear in any documents for similar ones could help with enhancing recall and precision rates. This could be accomplished by retrieving questions from the Stack Overflow dataset that contain such tags and deriving associated tags from them. An associated tag could then be substituted in for the base tag, and a new search could be performed with it. It may also be necessary to set some sort of limit on the number of associated tags usable for base tag substitution if the amount of associated tags available for use is too large. Finally, it

would be prudent to determine a threshold for the maximum number of extra tags to add to a query so that the topic for that query does not influence the retrieval results too heavily. This threshold could be chosen based on a minimum probability for a term in a topic also appearing in the query.

6.4 Parameter Sensitivity Analysis

For the experiment, 50 topics were created for each of the four programming languages to ensure a wide pool of topics to choose from when assigning them to queries. Additionally, the topic sizes were doubled from the normal 10 tags each [2] to 20 tags each to allow for more easily measurable increments of extra tags to add. Besides these, none of the parameters for creating the LDA model itself were altered from their defaults, except for the number of passes performed through the corpus during training, which was increased from the normal 1 pass [2] to 2 passes to improve the reliability of generated topics without potentially having drastic, unforeseen effects elsewhere. The effects of higher numbers of passes cannot be reliably tested at this time due to Gensim's implementation of LDA models not always producing the same topics with every use.

Chapter 7

THREATS TO VALIDITY

As observed in earlier sections of this paper, the main threats to validity are the numbers chosen for increments of extra tags and amount of topics and the interpolation parameter used for system tagging. The reason for this is that the results observed may vary depending on the chosen parameter values. We are also uncertain how the size of the training datasets would affect the results. The external validity refers to the generalization of our findings. In order to address this threat, we selected four different programming languages, Python, Java, C++, and JavaScript. A larger set of queries and larger dataset would strengthen the results from this perspective. Other threats to the validity of our solution include topics being sub-optimally generated (i.e. due to not performing enough passes through the corpus) and queries receiving tags and/or topics that are not necessarily the best fits for them. Both of these are attributable to human error in some capacity and would likely be alleviated by the assistance of an expert knowledgeable in a wide variety of programming languages and their related tools to verify the queries are receiving tags that match their language.

Chapter 8

CONCLUSIONS AND FUTURE WORK

With our research, we have proposed and demonstrated a new approach for query expansion that utilizes question tags and LDA topics generated from question bodies to enhance the retrieval performance of Stack Overflow queries. To evaluate the effectiveness of our approach, we conducted an experiment on long and short query sets for the Python, Java, C++, and JavaScript programming languages in which they were used to retrieve answer documents, adding greater numbers of extra tags from topics to each query with each experiment iteration. The consistent increases in recall and precision rates observed for all query sets throughout the experiment indicate that the addition of extra tags has a significant positive impact on query performance, and that tags should not be overlooked as a means of enhancement.

One possible goal to pursue next for our research is determining how our solution performs across multiple different sets of answers, with the same sets of queries being used for each answer set. Although we could not investigate this ourselves due to time constraints, it would be prudent to do so in order to verify its practical applications. Other possible future work to pursue includes investigating ways to simultaneously automate the assignment of topics to queries and reduce the need to rely on a human expert to do so accurately, determining the best threshold for how many extra tags should be added to a query, and further experimentation with the various optional parameters used by Gensim's LDA model implementation to determine how they may affect generated topics for query expansion.

BIBLIOGRAPHY

[1] https://stackoverflow.com/. [Accessed: July 14, 2021].

[2] https://radimrehurek.com/gensim/models/ldamodel.html. [Accessed: September 10, 2021].

[3] ALLAMANIS, M., AND SUTTON, C. Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In *Working Conference on Mining Software Repositories (MSR)* (2013), IEEE.

[4] ARORA, P., GANGULY, D., AND JONES, G. J. The good, the bad and their kins: Identifying questions with negative scores in Stack Overflow. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (2015), IEEE.

[5] AZIZI, M. A tag-based recommender system for regression test case prioritization. In *International Conference on Software Testing, Verification and Validation Workshops* (2021), IEEE.

[6] BAFATAKIS, N., BOECKER, N., BOON, W., SALAZAR, M. C., KRINKE, J., OZNACAR, G., AND WHITE, R. Python coding style compliance on Stack Overflow. In *International Conference on Mining Software Repositories (MSR)* (2019), IEEE.

[7] BALTADZHIEVA, A., AND CHRUPAŁA, G. Predicting the Quality of Questions on Stack Overflow. *Proceedings of Recent Advances in Natural Language Processing* (Sept. 2015).

[8] BHAT, V., GOKHALE, A., JADHAV, R., PUDIPEDDI, J., AND AKOGLU, L. Min(e)d your tags: Analysis of question response time in Stack Overflow. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (2014), IEEE.

[9] CAO, K., CHEN, C., BALTES, S., TREUDE, C., AND CHEN, X. Automated query reformulation for efficient search based on query logs from Stack Overflow. In *International Conference on Software Engineering* (2021), IEEE.

[10] CORREA, D., AND SUREKA, A. Fit or unfit : Analysis and prediction of 'closed questions' on Stack Overflow. In *ACM Conference on Online Social Networks (COSN)* (2013), ACM.

[11] DUJIN, M., KUCERA, A., AND BACCHELLI, A. Quality questions need quality code: Classifying code fragments on Stack Overflow. In *Working Conference on Mining Software Repositories (MSR)* (2015), IEEE.

[12] LI, Z., WANG, T., ZHANG, Y., ZHAN, Y., AND YIN, G. Query reformulation by leveraging crowd wisdom for scenario-based software search. In *Asia-Pacific Symposium on Internetware* (2016), ACM.

[13] LIU, J., BALTES, S., TREUDE, C., LO, D., ZHANG, Y., AND XIA, X. Characterizing search activities on Stack Overflow. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE* (2021), ACM.

[14] LIU, J., KIM, S., MURALI, V., CHAUDHURI, S., AND CHANDRA, S. Neural query expansion for code search. In *International Workshop on Machine Learning and Programming Languages* (2019), ACM, pp. 29–37.

[15] LIU, M., PENG, X., JIANG, Q., MARCUS, A., YANG, J., AND ZHAO, W. Searching Stack Overflow questions with multi-faceted categorization. In *Tenth Asia-Pacific Symposium* (2018), ACM.

[16] NASEHI, S. M., SILLITO, J., MAURER, F., AND BURNS, C. What Makes a Good Code Example? A Study of Programming Q&A in Stack Overflow.

[17] NIE, L., JIANG, H., REN, Z., SUN, Z., AND LI, X. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Transactions on Services Computing* (Sept. 2016).

[18] SENGUPTA, S., AND HAYTHORNTHWAITE, C. Learning with comments: An analysis of comments and community on Stack Overflow. In *Hawaii International Conference on System Sciences* (2020), HICSS.

[19] YANG, D., HUSSAIN, A., AND LOPES, C. V. From query to usable code: An analysis of Stack Overflow code snippets. In *Working Conference on Mining Software Repositories (MSR)* (2016), MSR.

[20] ZOU, J., XU, L., GUO, W., YAN, M., YANG, D., AND ZHANG, X. Which non-functional requirements do developers focus on? In *Working Conference on Mining Software Repositories* (2015), MSR.