UC Irvine
UC Irvine Electronic Theses and Dissertations

Title
Improving the Retrieval of Related Questions in StackOverflow

Permalink
https://escholarship.org/uc/item/2ng3k524

Author
Ghaderi, Rezvan

Publication Date
2015

License
CC BY-NC-SA 4.0

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Improving the Retrieval of Related Questions in StackOverflow

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Information and Computer Science

by

Rezvan Ghaderi

Dissertation Committee:
Professor Cristina V. Lopes, Chair
Professor Andre van der Hoek
Associate Professor James A. Jones

2015

© 2015 Rezvan Ghaderi

DEDICATION

To my mother for her great love and her wishes for my success

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vi**

ACKNOWLEDGMENTS **vii**

ABSTRACT OF THE THESIS **viii**

1 Introduction **1**
1.1 StackOverflow . . . 2
1.2 Motivation . . . 3
1.3 Organization . . . 5

2 Context **6**
2.1 Information retrieval . . . 6
2.2 Vector space model . . . 7
2.3 Document similarity algorithms . . . 8
2.3.1 String-based algorithms . . . 8
2.3.2 Corpus-based algorithms . . . 9
2.3.3 Knowledge-based algorithms . . . 10

3 Approach **12**
3.1 Dataset . . . 13
3.2 Preprocessing steps . . . 13
3.2.1 Dataset cleaning . . . 14
3.2.2 Stopword elimination . . . 15
3.2.3 Synonym replacement and abbreviation expansion . . . 17
3.2.4 Stemming . . . 18
3.3 Retrieval algorithm . . . 19
3.3.1 Similarity measure . . . 20
3.3.2 Code similarity . . . 21
3.3.3 Configuring algorithm . . . 22

4 Evaluation **25**
4.1 Study design . . . 25
4.2 Subjects . . . 26
4.3 Evaluation dataset . . . 28
4.4 Evaluation metric . . . 29
4.5 Evaluation results . . . 29
4.6 Statistical analysis . . . 31

5 Discussion **33**
5.1 Analyzing the results . . . 33
5.2 Threats to validity . . . 35

6 Related Work **36**
6.1 Studies on Q&A websites . . . 36
6.2 Studies on StackExchange . . . 38

7 Conclusion and Future Work **40**

Bibliography **42**

A User study instructions **46**

B Participants questionnaire **48**

C User study interface **50**

LIST OF FIGURES

Page

3.1 Vector space model 20
3.2 Results of manually inspecting performance 24

4.1 Familiarity with Software Engineering fields 28
4.2 Distribution of the precisions 31

LIST OF TABLES

Page

4.1 Level of education of participants 27
4.2 Level of proficiency at Java programming 27
4.3 Number and percentage of results 30
4.4 Average of precision 30
4.5 Results of overall performance 30

ACKNOWLEDGMENTS

I would like to thank my adviser, Professor Cristina V. Lopes. Her guidance through different phases of my thesis helped me to conduct this study.

I would like to thank Professor Andre van der Hoek and Professor James A. Jones for serving as my committee members.

Also, I would like to thank my friends and colleagues who participated in my user study. I really appreciate their help and the time that they spent for this research.

ABSTRACT OF THE THESIS

Improving the Retrieval of Related Questions in StackOverflow

By

Rezvan Ghaderi

MASTER OF SCIENCE in Information and Computer Science

University of California, Irvine, 2015

Professor Cristina V. Lopes, Chair

StackOverflow is a very popular Q&A website, known to all software developers. Developers can either post their coding questions on the website to be answered by other developers or explore the existing questions and their answers to find the solution they are looking for. Finding questions which are related to the desired topic and might include either the exact answer or some hints which can help to resolve the issue is very helpful for users of StackOverflow. This will decrease the time that users have to spend on search engines trying different keywords or waiting on Q&A websites for other users to reply to them.

In this work, I aim to improve related posts retrieved for each question in the StackOverflow website through information retrieval techniques customized for this website. The approach is based on text similarity algorithms applied on the content of posted questions including normal text and code. Firstly, I configured the algorithm based on the manual evaluation of the results of a small dataset. Then, I performed a user study with professional developers and graduate students in software engineering for evaluating the approach with the best configuration from the previous step in comparison to existing related posts in the Stack-Overflow. The results of this study revealed that my approach performs better than the algorithm used in StackOverflow website. Moreover, the statistical analysis of the results proved that this improvement is statistically significant.

Chapter 1

Introduction

The social web has been an invaluable infrastructure for new forms of peer communication. People can easily exchange their experience and knowledge on websites and forums being used by information seekers. Question and answer (Q&A) websites create a platform that makes sharing knowledge easier and provide a structure for asking questions and answering them.

Many people use Q&A websites for finding solutions to their problems and this is no exception for software developers. Programming is difficult, and despite being experts in their fields, programmers still might face some issues that force them to search for a solution. There are many books and manuals with thousands of pages published as programming references. However, because of the nature of coding, errors are inevitable. Developers might spend hours to investigate why their code is not working, while only a small spelling error has occurred.

Developers these days do not rely only on their knowledge of programming for doing their tasks. They always need to have access to multiple resources that can help them for fixing bugs or adding new features to their software. In some platforms such as the web, solving

certain technical issues requires knowledge of multiple languages or tools. Moreover, in the always growing area of IT, new technologies emerge which developers have to use and therefore, they will find new issues that they have never experienced. Also, lack of enough documentation for some API or library is another reason for seeking help from other people. In these cases, an online community of peers who are working with the same technology is very helpful.

1.1 StackOverflow

StackOverflow[1] is one the websites in StackExchange, which is a network consisting of multiple free Q&A websites. StackOverflow was created in 2008 and currently is the most popular free Q&A website in the IT area, widely used by developers to seek help for their programming questions. These questions are from a wide range of topics including algorithms, languages and tools [1].

Other developers that have experienced the same issue and already have found the solution, or experts who have broad knowledge about the topic and know how to resolve it, will help with sharing their knowledge through posting comments or answers. StackOverflow creates a collaborative environment which allows users to vote and rank the answers. Also, users who have earned sufficient privilege in the website are able to edit comments or posts, delete or close a question, and also flag duplicate questions or link similar questions.

As of September 2015, StackOverflow has more than 4.6 million registered users and over 10 million questions together with more than 16.5 million answers. Currently, an average of 7000 questions are posted on this website in a day [2]. Recent research shows that StackOverflow is replacing the use of search engines and forums, and is becoming the primary resource for finding solutions to software problems [3]. Even when developers search for some

[1] http://StackOverflow.com/

programming questions in search engines, those engines usually send them to most popular Q&A websites including StackOverflow. A study on search results for the JQuery API shows that in 84.4 percent of searches, StackOverflow is at least one of the top ten pages retrieved by Google search engine [4].

Each question asked by users of StackOverflow must contain a title and a brief textual description of the problem. Users can also add code snippets to the body of a question. Embedded code is distinguishable from the normal text by using a different style of formatting. Each question must be tagged with at least one to five tags that are expected to be representative of the content of the question.

1.2 Motivation

It is very common that a question is posted on StackOverflow by multiple people and with different wordings, making it likely that the answer which the user is looking for already exists in the community. Moreover, there are some questions that are not the exact duplicate of a question; however they are related to the context of it and their answers can help the user find the solution. Currently, there are more than 250,000 questions marked as duplicate and over 2 million questions flagged as linked in StackOverflow[2]. Therefore, exploring the existing posts and finding related questions is helpful for developers to find answers without the need to wait for others to reply to them.

Traditional search engines are a widely-used alternative to find solutions. However, these are not designed for finding solutions to long questions or questions which are embedded with a code snippet. Developers might have to try different keywords that enable search engines to bring them more relevant links. Findings of a recent case study at Google reveals that the average number of keywords that developers use for searching code is 1.85 with maximum

[2]http://data.stackexchange.com/ (As of August 2015)

of 11 [5]. Therefore, retrieval systems which are designed specifically to find solutions of programming questions and which allow developers to elaborate their questions in natural language are more advantageous than traditional search engines. These systems can utilize all of the information included in the question including title, text, code snippet and tags to obtain the best relevant solutions and to retrieve more accurate links.

There are several specialized search engines which developers can use for querying source code and reusable components such as Krugle[3], SourceForge[4] and Searchcode[5]. However, finding source code is only one of the reasons that developers search through resources or seek help of others. Reusing code is not as easy as cutting and pasting code.

StackOverflow provides its users with some features to help them to search and explore through the website. Search box, marking duplicate questions and displaying related and linked questions are some of these features. The "Related" section from StackOverflow has links to the top 10 most relevant questions from existing questions in the database. In this study, I aim to improve retrieving these related posts based on the information retrieval algorithms. Here, I apply string similarity metrics on text and code included in each question to find similar posts.

Presenting more relevant posts to users will help them to spend less time on searching for their answers. If users are sent to StackOverflow by a search engine, exploring related posts will increase the chance of finding a solution. Also, if relevant questions are presented to users after entering their question on the form and before final submission, it will help to reduce posting duplicate questions. Moreover, findings of this study can be employed in designing search engines specialized for coding questions.

[3]krugle.org (As of August 2015)
[4]http://sourceforge.net/ (As of August 2015)
[5]https://searchcode.com/ (As of August 2015)

1.3 Organization

The rest of this document is structured as follows: Chapter 2 is assigned to describe the context of information retrieval and similarity measures. Chapter 3 is designed to explain the approach that I proposed and implemented for retrieving related posts in StackOverflow. Chapter 4 will present the user study design and evaluation results. In Chapter 5, I will discuss the results of the study and also threats to validity of my findings. Chapter 6 includes related works and finally Chapter 7 will provide the conclusion of this thesis.

Chapter 2

Context

In this chapter, I review the background of my study including information retrieval and the vector space model. Since the focus of this study is on the usage of text similarity algorithms on StackOverflow posts, I also present a brief review of some of these algorithms.

2.1 Information retrieval

Information retrieval (IR) is the field of finding information which satisfies user needs within all available information. Although information retrieval is mostly known because of its usage in search engines, it was proposed decades before emergence of the World Wide Web. In 1940s, automated information retrieval was introduced for helping scientists to access and filter out required information among repositories of scientific articles and books [6].

Information retrieval systems can be related to any kind of data such as text, image, audio, etc. However, since this thesis is based on finding similar texts, this chapter will focus on text-based IR systems. I continue with describing the representation model used for documents in my approach.

2.2 Vector space model

Vector space model is the most commonly used model for text-based information retrieval systems because of its simplicity and effectiveness[7]. Proposed by Salton [8], this model is the underlying model in many information retrieval algorithms including document relevance, document clustering and classification [9]. In the vector space model, each document is represented as a vector of terms that are extracted from whole documents existing in the document collection. In this view, existence or number of times that a term occurs in the document matters, but relative order of terms is ignored when the document vector is constructed. In literature, this is called "bag of words" which is widely used in retrieval problems [9].

Associated weight with any term in the document vector indicates the relative importance of the term in the vector and can be calculated differently based on the purpose of the algorithm and also different configurations. The most common form of weight for terms is TF-IDF which consists of two factors: "how often the term j occurs in the document i (the term frequency $tf_{i,j}$ and how often it occurs in the whole document collection (the document frequency df_j)" [10]. In this schema, weight for each term is calculated through the following formula [10]:

$$W_{i,j} = tf_{i,j} \times logN/df_j$$

N indicates the number of whole documents in the document collection. The second factor in this formula gives higher weights to terms that have higher term frequency in a smaller subset of documents [10]. This factor in IR literature is called"Inverse Document Frequency" (idf_j). The above formula equals the following formula [10]:

$$W_{i,j} = tf_{i,j} \times idf_j$$

Higher *idf* indicates that word is more specific and exists in fewer documents than terms with lower *idf*. Therefore, it has more discriminative value, which is more useful in retrieving relevant documents.

After representing documents within the selected model, a similarity metric is applied to quantify similarity between documents and rank them based on the similarity score. The next section provides an overview of document similarity algorithms.

2.3 Document similarity algorithms

Document similarity has been under research for decades and many algorithms have been developed for this purpose. These algorithms are usually classified into three different groups: string-based, corpus-based and knowledge-based. String-based algorithms aim to find texts which are lexically similar, while corpus-based and knowledge-based algorithms try to find texts which are semantically similar.

2.3.1 String-based algorithms

String-based algorithms rely upon matching lexical units between two documents. A lexical similarity metric is selected and the similarity score is calculated based on the lexical units which exist in documents. Some of the more common metrics in this category are provided here:

- Cosine similarity: This metric quantifies the similarity between two documents represented in vector space model by measuring cosine of the angle between two document vectors [11].

- Euclidean distance: Euclidean distance in geography is the ordinal distance between two points. Given two documents which are represented as document vectors, the Euclidean distance measures the distance between points that vectors are pointing to them in the space [9]

- Jaccard coefficient: Jaccard coefficient measures the ratio of number of common words in two documents to total number of unique words in both documents [12].

- Dice coefficient: This metric is very similar to Jaccard coefficient, since this is as twice of the number of shared words in comparing documents over total number of words in both of documents [12].

Jaccard coefficient and Dice coefficient do not consider any weight assigned to words. Therefore, they are not applicable to the TF-IDF model which favors words with more discriminative value by considering their term frequency and document frequency in the document collection. Among mentioned metrics, Cosine similarity has been widely used [10] and findings of several studies shows that it has equal [13, 14] or better [15] performance among string similarity metrics.

It is worth mentioning that these string-based metrics are not necessarily used for single words. Based on the purpose of the approach, they can be applied on different lexical units such as n-grams or roots of words. An overview of string-based similarity metrics can be found in [16].

2.3.2 Corpus-based algorithms

In corpus-based algorithms, similarity between words is captured from a large set of documents. These algorithms usually use statistical approaches to find co-occurrence of the words [17]. A well-known algorithm in this category is Latent Semantic Analysis.

Latent semantic analysis (LSA)

Latent semantic analysis was proposed by Landauer in 1998 [18]. This algorithm assumes that words which occur together in similar texts in the corpus are related to the same concept. In this algorithm, at first a term-by-document matrix is constructed. Each row indicates the count of each term in each document. Dimensionality of this matrix is reduced by applying an algebraic algorithm which is called Singular Value Decomposition (SVD). Then, similarity between resulting vectors with reduced dimension is calculated by standard Cosine metric [17].

2.3.3 Knowledge-based algorithms

Knowledge-based algorithms are based on the structural information obtained from semantic networks [17]. Hierarchical information provided by semantic networks can be used in finding similar words by calculating the semantic distance between them. These hierarchies associate words with concepts [19]. Therefore, some metrics have been introduced to calculate semantic relatedness between a pair of concepts. Semantic relatedness is more general than semantic similarity, since it does not necessarily mean similarity. An overview of some of these metrics can be found in [17].

WordNet[1] is one of the most well-known semantic networks that is widely used in knowledge-based similarity algorithms.

WordNet

WordNet is a lexical database of English words, but it is beyond a thesaurus, since it groups together words which share a common meaning. These groups are called synsets. In addi-

[1] https://wordnet.princeton.edu/ (As of August 2015)

tion to the synonymy relation in each synset, some interlinked relations are defined between synsets such as super-subordinate relation (ISA relation is a kind of super-subordinate relation) [20].

Chapter 3

Approach

A brief look at information retrieval literature reveals that this area has been extensively studied for decades. Researchers constantly introduce new algorithms and provide results of comparing their performance to the older ones. And, many standard data sets are designed to provide a baseline for evaluation of results such as TREC[1] data.

However, we find fewer studies focusing on adapting information retrieval methods for particular domains. I believe that when it comes to a specific domain such as StackOverflow, performance of retrieval algorithms could be significantly improved by considering specific characteristics of this domain.

In this study, I aim to obtain the most relevant posts for each question posted on StackOverflow. Since vector space model associated with Cosine similarity metric is widely accepted as a baseline in information retrieval algorithms, I use this model for representing normal text and source code of the posts. In this chapter, I provide details of the approach, adjusted based on the data provided in the StackOverflow.

[1]Text REtrieval Conference. On-going series of evaluation workshops targeted at encouraging research on IR.

3.1 Dataset

StackExchange publishes an anonymized dump of their websites' archives under the Creative Commons license regularly[2]. The StackOverflow archive includes separate compressed XML files for posts, users, tags, votes, comments, badges, posts history, post links and etc. For this study, I used the dump published on March 16th, 2015 and downloaded the XML file of my interest which contains all posts existing in this website. The size of this file in compressed format is approximately 6.8 gigabytes.

For handling data in a more convenient way, I created a database in MySQL and imported the XML file into a table. Each record in the resulting table corresponds to a "<row>" element in the XML file and consists of 21 fields including title, body, tags, creation date, type of post, answer count, comment count, etc.

I queried posts with the type "question". I assume that there is more similarity between two related questions than a question and questions' answers. Answers might consist of very diverse solutions, and might contain technical words which are not mentioned in the question, or they might refer to external resources. When related questions are retrieved and displayed to users, they can readily access their respective answers.

I limited the dataset to posts which were under tag "Java". This subset included 810070 Java questions from more than 23 million posts existing in the StackOverflow dump.

3.2 Preprocessing steps

In the "bag of words" model, all words regardless of their characteristics such as number of characters or their part of speech have equal influence in the resulting document vector.

[2]https://archive.org/details/stackexchange (As of August 2015)

Therefore, it is very important to consider what we are injecting as input to this model. Based on the purpose of the algorithm, input needs to be filtered out appropriately. I applied four steps of preprocessing on my dataset including dataset cleaning, stopword elimination, synonym replacement and abbreviation expansion and stemming.

3.2.1 Dataset cleaning

Before performing any type of process on the dataset, it is necessary to clean the data and remove irrelevant characters or strings. I started with extracting contents of each question including title, body and associated tags from the dataset. The body of each question can contain normal text and source code which are separated from each other with special tags such as "<code>" and "<pre>". Users might include several code snippets in the body. I wrote a Python program to clean the data through the following steps:

- Splitting textual and code contents.
- Concatenating all code segments embedded in the body.
- Removing special tags of StackOverflow.
- Trimming and removing punctuation marks and extra spaces.
- Lowercasing all text in the title and body of posts.
- Splitting function names with periods.
- Removing some special characters. Since the content of posts, even after splitting text from the code snippets, might include special characters related to code (such as operators), I did not follow the general cleaning approach for removing all special characters. I only eliminated a few special characters such as parenthesis, curly braces, question marks at the end of titles and underscores.

3.2.2 Stopword elimination

One of the fundamental ideas in the context of information retrieval is removing common words that appear frequently in the documents, but do not provide helpful information for the users' needs. These words, which are called "stop" words, can decrease the retrieval rate. Therefore, they are removed from documents before starting to process them. Eliminating stopwords also reduces dimensionality in many information retrieval algorithms including the vector space model [21].

Removing stopwords is not a new idea; every Natural Language Processing (NLP) package has a default English stoplist (a list of stopwords). However, there is not a general stoplist that can be applicable to documents of all domains. Consider the word *learning* that is a stopword in the *education* domain while it is a keyword in the *computer science* domain. Eliminating domain specific stopwords will improve the performance of algorithms in retrieval tasks [22].

On the other hand, there are some words in common English stoplists that shall not be removed from texts of some particular domains. Consider words "if", "for", "do" and "while" which can be found in any English stoplist. However, these are keywords in some programming languages and users might use them in the title or description of a question. Therefore, it is required to create a domain specific stoplist for StackOverflow posts to avoid removing programming keywords from the normal text of questions.

Creating stoplist

One of the general strategies for creating a domain specific stoplist, as proposed by Manning et al. [9], is based on the collection frequency. Collection frequency of a term is the total number of occurrences of the term in the whole collection. If a term appears multiple times

in a document, it will be counted toward calculating collection frequency. In this method, at first the most frequent words are selected according to collection frequency. Then, domain experts manually evaluate these words for constituting a stoplist.

n-gram Stoplist

I followed the collection frequency approach to create a stoplist for StackOverflow questions. However, not only did I obtain the most frequent single terms (unigrams), but I also extracted the most frequent bigrams and trigrams. In this context, n-gram is a consecutive sequence of n words. Unigram, bigram and trigram are the special cases of n-gram, respectively consisting of one, two and three words.

Using n-gram stopwords has some advantages over individual stopwords in the Q&A websites. Since these websites are designed for specific purposes and are limited to questions about a certain area of knowledge, we can find sequences of words that are used frequently for expressing user needs. "How can I", "How do I", "How should I", "How would you", "What is the", "How to use", "Is it possible", "Is there a", "best way to", "not able to" and "the difference between" are some examples of the most frequent trigrams which were found in StackOverflow questions. Since these trigrams do not have discriminative value in the retrieval algorithm, they are removed from the dataset.

Since phrases are more meaningful than single words, it is easier for domain experts to recognize stopwords during the manually check step. A sequence of words is more informative about the context and this is important in our domain, where technical words can be embedded in normal text. Consider "How do I" in frequent trigrams. "do" is a keyword in most programming languages, but in this trigram it is obvious that "do" is not a keyword and can be eliminated with confidence.

Five hundred of the most frequent items of each trigrams, bigrams and unigrams were found. They were manually inspected and a stoplist was created to the best of my knowledge. For better utilizing the n-gram stoplist, eliminating stopwords was done in a way that trigrams were eliminated at first and unigrams were removed last after removing bigrams.

3.2.3 Synonym replacement and abbreviation expansion

Using synonyms, short forms and abbreviations is very usual in the programming related context. For example, "C#", "C #", "c sharp" and "csharp" reference the same term; "CSS" stands for "Cascading Style Sheets" and "regex" is the short form of "regular expression". Therefore, synonym replacement and abbreviation expansion is a fundamental step in preprocessing.

To resolve this, I created a dictionary to replace synonyms with a standard term, and abbreviations and short forms with their full forms. I found synonym tags from StackOverflow data as the best available resource for the purpose of this study. StackOverflow allows users who have enough privilege to fix incorrect tags or tags with alternate spelling and phrasing by substituting them with correct tags. The goal here is to avoid using multiple tags for the same thing. This data as synonym tags are available from StackExchange data explorer[3] and currently includes more than 3000 synonym pairs.

Synonym tag pairs are suggested by users based on different strategies such as similarity, tag synonymy, word synonymy, acronym, abbreviation, plural & singular form, variant spelling of the same word, etc [23]. The synonym tags list was reviewed manually to remove synonym tag pairs where substituting them is not aligned with the purpose of this step. For example, I did not intend to replace "if" with "if-statement". The resulting list thereafter consists of about 900 entries that were used for replacing synonyms and abbreviations.

[3]http://data.stackexchange.com/ [As of August 2015]

3.2.4 Stemming

Manual inspection of the questions in the StackOverflow revealed that users use different grammatical forms of words and verbs for mentioning the same issue. Text in the questions can be in any structure and because of the nature of the normal text, same or similar questions can be asked in different variations. For example, consider these two questions titles from StackOverflow:

"How can I rename a file in Java?"

"Renaming a file (Java)"

If we input these titles to the model without any preprocessing, "rename" and "renaming" will be treated as two completely different words, while these are two variations of the same verb. For overcoming this mismatch and generalizing words in normal text, I applied a stemming algorithm as a part of preprocessing steps.

Stemming was introduced more than five decades ago and has been under research for a long time [24]. Stemming refers to the procedure of transforming different variations of the same word to their stem, usually through stripping suffixes and prefixes. Although stemming is a very commonly used process in information retrieval, it might cause false matching of some words with different stems to the same root [25]. For example both "universal" and "university" would stem to "univers" which is not desirable. However, evaluation experiments have proved performance enhancement through stemming algorithms, due to outweighing the number of the meaningful matches over the number of incorrect matches [26]. Moreover, since groups of words which have the same root will be stemmed to only one word, dimensionality will be reduced.

For this study, out of many stemming methods, I selected Porter stemmer which is a rule-based stemmer and widely used for English stemming [27, 28]. I applied this stemmer for

extracting stemmed version of the terms in the normal text and titles of the questions in the dataset.

Since source code has a predefined structure and does not follow grammatical and structural variations existing in natural languages, there is no reason to perform stemming on the code snippets. For the purpose of this study, I rely on keywords included in the code snippets and try to keep them intact. Since stemming is based on some linguistic rules such as removing suffixes, it is expected that stemming has minimum effect on the technical words included in normal text and title.

3.3 Retrieval algorithm

In this section, I explain how the most relevant posts for a question are retrieved from the dataset. After the preprocessing steps, all questions in the dataset in a tokenized format are represented as a vector in the model. Then the pairwise similarity score of the question with all posts existing in the dataset is calculated. Questions with higher similarity score are expected to be more relevant to the question. This approach is based on this assumption that related posts share more words than non-related ones.

Since questions in StackOverflow are centered on coding questions, they have many technical words that do not exist in natural language dictionaries. Also, users can include code snippets or error logs in their post. Because computer programs do not have the diversity and variations of terms existing in natural language, as they have their own keywords with a strict syntax, it is expected that technical words (such as “process”, “thread”, “pointer”) and programming keywords have high discriminative value in retrieving relevant questions.

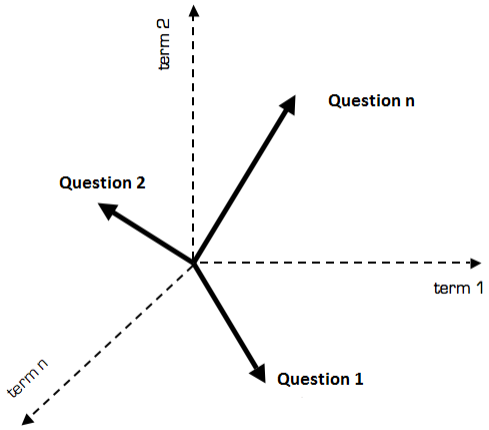


Figure 3.1: Vector space model[4]

3.3.1 Similarity measure

For scoring the similarity of two questions, I used Cosine similarity, which is a standard form of measuring document similarity in vector space model [29]. This measure is the cosine of the angle between two vectors and can be in the range of 0 (orthogonal vectors) to 1 (identical vectors). Cosine similarity between two vectors of $\vec{x}$ and $\vec{y}$ is calculated by the normalized inner product of $\vec{x}$ and $\vec{y}$ and is defined with the following formula [9]:

$$cos(\vec{x}, \vec{y}) = \frac{\vec{x}.\vec{y}}{||\vec{x}||.||\vec{y}||}$$

In this context, each vector is a question, and terms included in the question consist the features of the vector (Figure 3.1). In the Cosine metric, calculating the angle between vectors of two questions will indicate how related they are. Since Cosine metric is an efficient way for calculating similarity, especially in sparse matrixes, it is very commonly used for high dimensional vectors in text mining and information retrieval [30].

[4]Adapted from http://blog.christianperone.com/

3.3.2 Code similarity

In this approach, I did not rely only on the natural text included in the title and body of the questions. Since users of StackOverflow are able to post code snippets or error logs and this data can provide much information about the context of the questions, I decided to find similarities based not only on normal text in questions and titles, but also included code similarity in my algorithm.

Sometimes users post their codes and ask people to help them with the code, while they do not provide enough description about it. Also, there are some cases that descriptions are very similar, while they are related to different areas of programming. For example "could not find or load main class" is an error in Java that occurs in diverse programming contexts. It is expected that a code snippet associated with text will help discriminate them from an unrelated question with a similar description and reduce the number of false positives. Moreover, users of StackOverflow are not limited to English-speaking people. Developers from around the world with different knowledge of English can post their questions, while the nature of coding forces them to follow the standard syntax in code snippets.

Users usually post a small chunk of the code related to their question, not the whole of their source code. I assume that language keywords, libraries, methods or API names referred in this chunk of code will help to distinguish the context of the question. For example, "getConnection", "PrepareStatement" and "executeQuery" are Java keywords that are related to database. Since keywords are predefined, it is expected that lexical matching increases the retrieval performance of the algorithm.

There are similar classes or libraries that are developed for the same purposes and we can consider them as semantically similar. However, it is not reasonable to retrieve questions related to a different component when the user has included a code snippet related to a particular component.

I treated code as plain text and quantified code similarity based on the matching terms between two code snippets, the same term-based approach followed for normal text. Treating code as text has been followed in many code-based information retrieval algorithms such as Google Code Search (currently deprecated), Krugle and Koders [31].

The goal here is different from code search or source code retrieval algorithms that might be based on the whole source code available in the document collection. This approach does not aim to find similar code snippets. There are cases that code snippets are very similar but they are posted for asking different topics. I faced an example that someone had posted the same code for three different questions. The main focus is still on the similarity of normal text included in the title and the body. Therefore, I set a cutoff for the code similarity in the formula; the effect of high similarity score between two code snippets are reduced if it is more than 0.8 and the similarity score of text in the body and the title is less than 0.5.

3.3.3 Configuring algorithm

As stated by Swanson [32], information retrieval is an iterative and interactive process and trial and error is the essence of IR. For finding the best configuration of the algorithm, I tried it on real data and adjust it based on the evaluation results.

I started with evaluating the algorithm on the duplicate questions from StackOverflow. Users who are privileged can flag duplicate questions in StackOverflow, and this data are available in the StackOverflow dump. It is assumed that a user-specified duplicate of a question is the best related question to it. Therefore, it is expected that any improvement in retrieving duplicate questions results in enhancing performance of related questions retrieval. However, I found out that not all duplicate questions are marked in StackOverflow database. I saw examples of duplicate questions retrieved by my algorithm, while they were not considered as duplicate by StackOverflow users.

Therefore, I continued with manually evaluating related posts, although it was very time consuming. I randomly sampled 35 questions from the dataset to have a baseline for finding the best configuration of my algorithm. Based on the inspection of retrieved questions for this test dataset, the algorithm was adapted in multiples steps to improve performance. Here, I provide different steps that were tried on the test dataset. These steps were cumulative and were applied one after another.

- Step 1: calculating code similarity and normal text similarity separately between two bodies of questions and summing the two resulting scores
- Step 2: adding score of title similarity to the two similarity scores from previous step
- Step 3: stemming on normal text and setting cutoff for code similarity
- Step 4: using n-gram stopwords
- Step 5: adding tags to the normal text of the questions and replacing synonyms and abbreviation expansion
- Step 6: ignoring term frequency for the normal text

The first two steps were tested on the duplicate questions from StackOverflow. Adding title score in step 2 improved finding duplicates by 6 percent. Results of manual inspection in the next steps are summarized in Figure 3.2. As we can see, the best performance is obtained after step 5 that is the baseline for my user study explained in the next chapter.

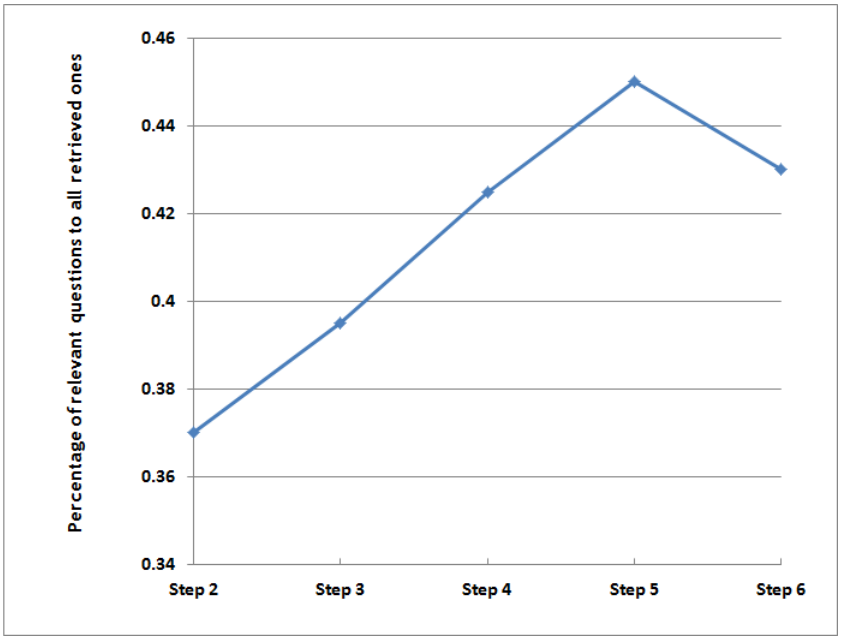


Figure 3.2: Results of manually inspecting performance

Chapter 4

Evaluation

Information retrieval (IR) is a user centric study and user satisfaction has an important role in evaluation of IR systems [33]. Since I aim to find the most relevant posts from the users' perspective, I conducted a user study to evaluate the performance of the proposed approach. This experiment was based on the best configuration of the algorithm from the previous chapter. In this chapter, details of this study and its findings are provided.

4.1 Study design

I compared evaluation metrics of related posts from my approach to related posts retrieved from StackOverflow website. My research question was "Has my algorithm improved retrieval performance of related questions for developers?".

For this purpose, I developed a web user interface. Each page in this interface displayed a question from StackOverflow at the top and two lists of related posts at the bottom, one list from my approach and one list obtained from StackOverflow website (see Appendix C). Participants were asked to rate these two different lists of related posts. To avoid having

influence on users' opinions, they were not informed about the source of these lists. There was no difference between appearance of the two lists, and their order at right or left of the page was randomly switched. Users were able to click on the titles in both lists to see the complete post in StackOverflow website.

To make this study easier for the participants to complete, I decided to use binary evaluation instead of multi-level grading, meaning that participants were asked to rate retrieved lists based on a binary relevance ("Yes" as relevant vs. "No" as non-relevant). Binary relevance is a straightforward measure for evaluating algorithms in information retrieval and also classification tasks [34]. In addition, users were able to choose the option "Maybe" (as possibly relevant) when they were not certain about relevancy. Adding this option ensured me that users were confident when they chose "Yes" or "No" answers.

Users were also asked to rate the overall performance of the lists of related posts. They had the option to propose one of the lists as the better one or to select the "No difference" option. Moreover, they were able to enter an optional comment when submitting their answers.

4.2 Subjects

A total of 13 people provided values for this study. Each received an instruction file (see Appendix A) and also links to 20 questions. Each link points to a webpage, displaying a question and two associated related lists. Participants were asked to complete a questionnaire designed as a Google form (see Appendix B) for collecting some general information about them.

The users included 3 professional software developers and 10 graduate students with major in Software Engineering. Since graduate students in Software Engineering do extensive pro-

gramming, they use StackOverflow frequently as a reference for finding their coding answers and are the most suitable participants for this study.

Subjects included 2 females and 11 males, and were between 25 to 50 years old, and had at least 6 years education in Software/Computer Science/Informatics or related fields. Their level of education is summarized in Table 4.1:

Level of Education	Number
PhD	2
PhD Student	7
Master	4

Table 4.1: Level of education of participants

As shown in Table 4.2, except for one of the participants, all subjects had at least intermediate proficiency at Java programming. Therefore, I believe that they had good insight on how Java programming questions are related to each other.

Proficiency Level	Number
Little	1
Intermediate	5
Proficient	5
Very proficient	2

Table 4.2: Level of proficiency at Java programming

Users had different degrees of expertise, and were comfortable with a broad range of areas in Software Engineering. I asked them about their knowledge of these areas, and these data are displayed in Figure 4.1. Each bar indicates number of participants who are familiar with its corresponding field.

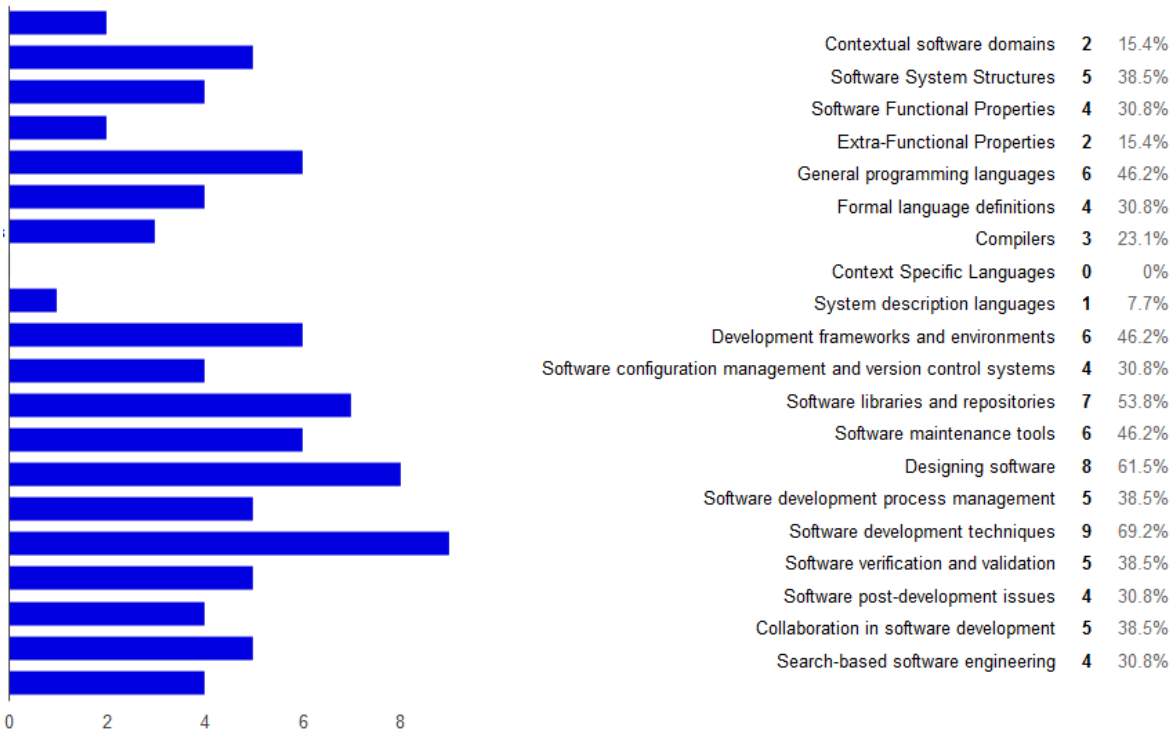


Figure 4.1: Familiarity with Software Engineering fields

4.3 Evaluation dataset

User study was designed to evaluate how my algorithm performs in comparison to related posts from StackOverflow. I did not intend to investigate how different users evaluate the algorithm based on the same data. Therefore, instead of assigning a smaller sample to all users, I used a larger sample that consisted of more questions and therefore ensured me that enough diverse programming questions were included.

20 Java questions, randomly selected from the dataset, were assigned to each potential participant. Every participant received a different set of questions, accessible through the links to the user interface in the server.

4.4 Evaluation metric

For this user study, the cutoff number for retrieved results was set to 10 to make it consistent to related posts retrieved from StackOverflow. As a measure for quantifying the results, I used precision at n (suggested by Kelly [35]) that is defined as "the number of relevant documents in the top n results divided by n". Since in my case n equals 10, the following formula was used for calculating this metric:

$$Precision\ at\ 10 = \frac{\text{The number of relevant questions in the top 10}}{10}$$

For the sake of conciseness, I use precision instead of precision at 10 for the rest of this document.

4.5 Evaluation results

Some of the participants completed the study for all 20 questions, while others skipped some of the questions or only helped with a subset of them. Overall, the approach was tested on 144 questions with 1440 related posts from each of my approach and StackOverflow.

Table 4.3 summarizes the overall number and percentage of related ("Yes" answers), non-related ("No" answers) and possibly related ("Maybe" answers) questions to total retrieved posts for all 144 questions, separated by each approach. From the raw data in this table, we can see that my approach obtained higher values for the number of related and possibly related questions.

Table 4.4 provides average of precision as the evaluation metric for each approach. First row is based on considering only related posts (only "Yes" answers), while in the second row

Relevancy	My Approach		StackOverflow	
	Number	Percentage	Number	Percentage
Yes	574	0.40	450	0.31
Maybe	149	0.10	91	0.06
No	717	0.50	899	0.63
Total	1440		1440	

Table 4.3: Number and percentage of results

precision is based on adding "Maybe" answers to "Yes" answers.

	My Approach	StackOverflow
Related questions	0.40	0.31
Possibly related questions	0.50	0.38

Table 4.4: Average of precision

As we can see, my approach achieved 40 percent as average of precision for related posts that indicates an improvement of 9 percent to StackOverflow performance. If possibly related answers are included in the evaluation formula, this improvement is increased to 12 percent. This means that for each question, my approach will display approximately one more related link than StackOverflow.

Participants were asked to evaluate performance of the approach in overall in comparison to StackOverflow. They opted more for my approach and selected it as the better retrieval approach in 49 percent of questions, while they only voted the StackOverflow at 27 percent. Answers of users for this question are summarized in Table 4.5.

Approach	Number	Percentage
My Approach	70	0.49
StackOverflow	39	0.27
No difference between algorithms	35	0.24
Total	144	

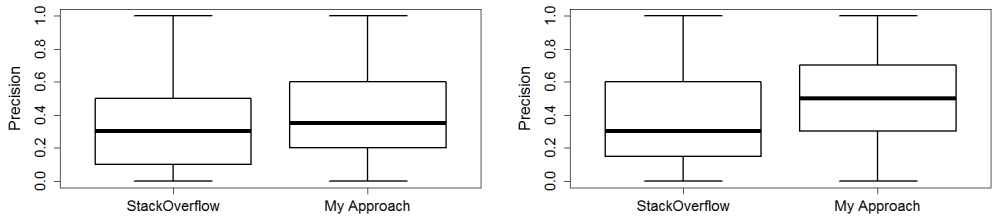
Table 4.5: Results of overall performance

From the results depicted in Table 4.5, we can see that from the users' perspective, my approach had better or equal performance in 73 percent of cases.

4.6 Statistical analysis

In this section, I provide the results of statistical analysis for assessing validity of my findings from the user study. My null hypothesis (H0) is "There is no difference between precision metric from two approaches" and the alternate hypothesis (H1) to test is "My approach surpasses StackOverflow in terms of precision".

Figure 4.2 represents the distribution of the precisions from my approach compared to StackOverflow. Figure 4.2a is based on considering only related questions (where the user answered "Yes"), while Figure 4.2b depicts precision when possibly related questions are also included in measuring precision (users answered "Yes" or "Maybe"). From both figures, we can see that precision resulting from my approach is relatively higher than precision from StackOverflow. This difference is more visible in Figure 4.2b. While 75 percent of the cases from my approach have precision higher than 0.3, only 50 percent of questions have higher precision than 0.3 from StackOverflow (considering possibly related questions).



(a) Related questions (b) Possibly related questions

Figure 4.2: Distribution of the precisions

To detect if this difference is significant and could not happen due to chance, a statistical test was required. As suggested by Hull [36], I conducted paired t-tests to compare precision

from two different approaches at both cases. Since p-values in both cases are smaller than 0.001, the null hypothesis is rejected in favor of alternate hypothesis. I conclude that there is significant evidence that my approach improves the precision of related posts retrieved for each question.

Chapter 5

Discussion

5.1 Analyzing the results

Experimental results from the previous chapter proved that my approach surpasses StackOverflow's approach in retrieving related questions. There is no information describing how StackOverflow manages to retrieve related posts. I only found this paragraph on their website[1]:

"One of our major performance optimizations for the 'related questions' query is removing the top 10,000 most common English dictionary words (as determined by Google search) before submitting the query to the SQL Server 2008 full text engine. It's shocking how little is left of most posts once you remove the top 10k English dictionary words. This helps limit and narrow the returned results, which makes the query dramatically faster".

They mention that they use a general stoplist from Google for removing common words. I am not certain if this information is still valid and they are using the same approach. As

[1]https://blog.stackexchange.com/2008/12/podcast-32/ (As of August 2015)

discussed in Chapter 3, stopwords must be domain-specific. From Figure 3.2, we saw that removing stopwords improved the performance of retrieval algorithm by about 3 percent.

Another improvement in my approach, which was visible in manual evaluation, is related to questions which include special characters. One example of a question found on StackOverflow was:

"What does this operator |= mean?"

For this question, my approach retrieved 9 related questions, while only one of the related posts from StackOverflow was related to this operator (others were related to different operators, such as "—>" and "!!"). As I explained in Chapter 3, I believe that customized preprocessing steps are necessary to keep discriminative terms, such as operators. My research seems to support this customization has a relevant impact on the retrieval of related questions.

One of the steps that improved performance of my algorithm significantly was calculating title similarity separately and adding it to normal text similarity and code similarity. This gives more emphasis to title similarity since it has a separate score and is not included in normal text similarity score. This noticeable improvement demonstrates that although titles are short, they describe the main context of the problem and have high value in finding similarities.

Although my dataset only contained Java questions, the approach is general enough to be language independent and could be applied to dataset of questions of any language or mixture of multiple languages.

5.2 Threats to validity

- Configuration dataset only included 35 questions. Manual inspection of related posts was tedious and time consuming. Therefore, I was not able to work on a larger dataset.

- For the same reason, I had to merge some of the modifications of the approach into one step. For example, I added tags to the normal text and replaced synonym and abbreviations in the same step. Therefore, I did not evaluate all individual improvements for any of these modifications.

- Because of the memory restriction, I had to limit the dataset to "Java" questions. Therefore, the test dataset and the evaluation dataset were limited to Java questions. This is partly covered by Java being a General Domain Language, and participants' knowledge covering a wide range of topics.

- Participants might have not been familiar with some of the questions assigned to them, therefore they could not rate them precisely. The option "Maybe" was added as an attempt to capture this situation.

- Participants were not the real information seekers. If I had the chance to evaluate the approach by developers who were actually looking for the information and come to StackOverflow and rate the related questions, I would have better insight to the behaviour of my approach.

Chapter 6

Related Work

With the availability of Q&A websites' archives, researchers have opportunities to conduct diverse research especially big data analysis on these data. Because of its visibility among software developers, StackOverflow has been under research extensively. However, there are fewer works focused on retrieval algorithms applied on question answering websites. In this chapter, I summarize related work in two categories: retrieval studies on general Q&A websites and retrieval studies on StackExchange specifically. My work is different from the first category, since my research is based on the special characteristics of questions from StackOverflow. The studies in the second category also differ from my work since they are targeted at different goals or users.

6.1 Studies on Q&A websites

Joen et al. [37] proposed a question retrieval method to find similar questions from a community based question and answer service called "Naver". Their work is based on the assumption that if two questions have similar retrieval results, questions are semantically similar. They

built a machine translation model based on the features derived from the answers and stated that this model is superior to finding similarity between questions directly.

Based on the same assumption, there is another work by Hao et al. [38] for finding similar questions from Yahoo! Answers archive. They propose a novel method called "bootstrapping-based pattern extension" which tries to find pattern of questions which share the same answer. These patterns are used as seed to match more questions and create a larger set of equivalent patterns. Their algorithm has improved the retrieval results comparing to the baseline models.

In another study by Bernhard et al. [39], they aimed to find the answer of a user's question through retrieving the best matching paraphrase question within the already existing answered questions in WikiAnswers repository. They compared performance of different string similarity metrics in retrieving questions paraphrases. Based on their experiments, combination of Lucene, Term Vector Similarity with stemming and n-gram overlap with spelling correction yielded the best accuracy across all applied methods.

A similar work by Liu et al. [40] is designed to find the best potential answer to a new question through topic identification in Yahoo! Answers. Their approach is a mixture of the Latent Dirichlet Allocation (LDA) model and language model for quantifying the relation between a question and its answer. Also, user activity and authority information is integrated to the model for finding users who are active and have more accurate answers. Although their results displayed effectiveness of their approach, it is not evaluated on a large scale dataset.

6.2 Studies on StackExchange

Muthmann et al. [41] provides some experiments to identify topical near-duplicate questions in StackExchange website. Base on their definition, "two questions are equal if it is possible to find the most specific answer satisfying the information needs of both users". They used several supervised classification algorithms over pairs of forum posts to classify them as semantically near duplicate questions or non near-duplicates. Based on their experiment on four different datasets consisting of 260 to 2210 posts, logistic regression had better or equal performance than other classification algorithms.

Philip et al. [42] in their research paper introduce Dr. Tux as an application developed in Python that uses information retrieval and natural language processing for finding answers for technical questions. This application aims to find an accurate answer through searching data from different forums in a reasonable time. For the first phase of their project, they focus on questions related to Ubuntu, and data comes from AskUbuntu.com, a part of the StackExchange network. Firstly, they find the top 20 questions with highest TF-IDF results for a specific question. Then, they apply a dependency parser to calculate semantically similarity scores. Their work is limited to find the exact answer for Ubuntu questions, not finding the related ones.

Wang et al. [43] studied a recommendation system that particularly relates to API issues. This system aims to provide API designers with a subset of posts from StackOverflow that are more likely concerned about API bugs and issues. In order to create this reduced set of posts, they use posts and some metadata from StackOverflow, filter them out and rank. At first, they find the expert users from StackOverflow automatically. Then, they apply Latent Dirichlet Allocation (LDA) to posts for reducing the dimensionality of posts and assigning a topic to each of them. Next, they identify questions that are answered late or are not answered, since questions that are answered quickly are related to known bugs. And, finally

they rank them based on the questions difficulties. This recommendation system is designed to help API designers not StackOverflow users.

Bacchelli et al. [44] introduced "Seahawk", a recommendation system as an Eclipse plugin which allows developers to search StackOverflow directly within the IDE. They aim to help developers to find answers for their questions without the need to interrupt their flow and refer to the website. This plugin is based on the vector space model and TF-IDF weighting. The plugin mines StackOverflow knowledge and can import the code snippet to the Eclipse IDE.

Chapter 7

Conclusion and Future Work

In this thesis, I worked on improving retrieval of related questions at StackOverflow. Because of its popularity among software developers, improving StackOverflow features will help them spend less time finding the solutions they are looking for.

My work was focused on lexical similarity metrics between questions, while considering special characteristics of questions existing in this website. I used customized preprocessing steps on the data from StackOverflow for improving retrieval performance. I implemented a special cleaning step and created a customized stoplist for this domain. In addition, synonym tags from StackOverflow were used to replace short forms and abbreviated forms.

Since questions posted in StackOverflow can include code snippets in addition to normal text, I utilized both similarity of normal text and similarity of code in my algorithm. My approach is based on the assumption that technical words embedded in natural text and also programming keywords included in code snippets have high discriminative value in finding similar questions due to sharing the same context.

I used manual configuration at different steps and enhanced the retrieval metrics based on inspecting retrieved questions at each step. For evaluating the approach, I conducted a user study with 13 StackOverflow users. Findings of this user study showed that my approach improved the precision of related posts by 9 to 12 percent and results of statistical analysis proved that this improvement is significant.

This work can be extended in many ways. Although I improved the retrieval performance of related posts, precision is still less than 50 percent. I used Cosine similarity that is a baseline for calculating string similarities. As a future work, other methods such as LDA and topic modelling from probabilistic models can be integrated into the approach.

StackOverflow is a collaborative environment and utilizes users' feedback for finding duplicate and linked questions. In the future work, this data can be included in the algorithm for finding more accurate relevant questions. Also, some of the manual steps, such as creating stoplists could be automated.

Bibliography

[1] Bosu, Amiangshu, Christopher S. Corley, Dustin Heaton, Debarshi Chatterji, Jeffrey C. Carver, and Nicholas A. Kraft. Building reputation in stackoverflow: an empirical investigation. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 89-92. IEEE Press, 2013.

[2] Joorabchi, Arash, Michael English, and Abdulhussain E. Mahdi. Automatic mapping of user tags to Wikipedia concepts: The case of a QA websiteStackOverflow. *Journal of Information Science* (2015): 0165551515586669.

[3] Mamykina, Lena, Bella Manoim, Manas Mittal, George Hripcsak, and Bjrn Hartmann. Design lessons from the fastest qa site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 2857-2866. ACM, 2011.

[4] Parnin, Chris, and Christoph Treude. Measuring api documentation on the web. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pp. 25-30. ACM, 2011.

[5] Sadowski, Caitlin, Kathryn T. Stolee, and Sebastian Elbaum. How Developers Search for Code: A Case Study.

[6] Frakes, W. B. Introduction to information storage and retrieval systems. *Space* 14 (1992): 10.

[7] Zhao, Yao-hong, and Xiao-feng Shi. The Application of Vector Space Model in the Information Retrieval System. In *Software Engineering and Knowledge Engineering: Theory and Practice*, pp. 43-49. Springer Berlin Heidelberg, 2012.

[8] Salton, Gerard, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM* 18, no. 11 (1975): 613-620.

[9] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schtze. *Introduction to information retrieval*. Vol. 1. Cambridge: Cambridge university press, 2008.

[10] Lee, Dik L., Huei Chuang, and Kent Seamons. Document ranking and the vector-space model. *Software*, IEEE 14, no. 2 (1997): 67-75.

[11] Berry, Michael W., Zlatko Drmac, and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM review* 41, no. 2 (1999): 335-362.

[12] Markines, Benjamin, Ciro Cattuto, Filippo Menczer, Dominik Benz, Andreas Hotho, and Gerd Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *Proceedings of the 18th international conference on World wide web*, pp. 641-650. ACM, 2009.

[13] Huang, Anna. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008)*, Christchurch, New Zealand, pp. 49-56. 2008.

[14] Strehl, Alexander, Joydeep Ghosh, and Raymond Mooney. Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pp. 58-64. 2000.

[15] Lewis, James, Stephan Ossowski, Justin Hicks, Mounir Errami, and Harold R. Garner. Text similarity: an alternative way to search MEDLINE. *Bioinformatics* 22, no. 18 (2006): 2298-2304.

[16] Malakasiotis, Prodromos, and Ion Androutsopoulos. Learning textual entailment using SVMs and string similarity measures. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 42-47. Association for Computational Linguistics, 2007.

[17] Mihalcea, Rada, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, vol. 6, pp. 775-780. 2006.

[18] Landauer, Thomas K., Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes* 25, no. 2-3 (1998): 259-284.

[19] Li, Yuhua, Zuhair Bandar, and David McLean. An approach for measuring semantic similarity between words using multiple information sources. *Knowledge and Data Engineering, IEEE Transactions on* 15, no. 4 (2003): 871-882.

[20] Miller, George A. WordNet: a lexical database for English. *Communications of the ACM* 38, no. 11 (1995): 39-41.

[21] Sinka, Mark P., and David Corne. Evolving Better Stoplists for Document Clustering and Web Intelligence. In *HIS*, pp. 1015-1023. 2003.

[22] Makrehchi, Masoud, and Mohamed S. Kamel. Automatic extraction of domain-specific stopwords from labeled documents. In *Advances in information retrieval*, pp. 222-233. Springer Berlin Heidelberg, 2008.

[23] Beyer, Stefanie, and Martin Pinzger. Synonym Suggestion for Tags on Stack Overflow

[24] Lovins, Julie B. *Development of a stemming algorithm.* MIT Information Processing Group, Electronic Systems Laboratory, 1968.

[25] Metzler, Donald, Susan Dumais, and Christopher Meek. *Similarity measures for short segments of text.* Springer Berlin Heidelberg, 2007.

[26] Kraaij, Wessel, and Rene Pohlmann. Viewing stemming as recall enhancement. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 40-48. ACM, 1996.

[27] Kantrowitz, Mark, Behrang Mohit, and Vibhu Mittal. Stemming and its effects on TFIDF ranking (poster session). In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 357-359. ACM, 2000.

[28] Porter, Martin F. An algorithm for suffix stripping *Program* 14, no. 3 (1980): 130-137.

[29] Singhal, Amit. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.* 24, no. 4 (2001): 35-43.

[30] Dhillon, Inderjit S., and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning* 42, no. 1-2 (2001): 143-175.

[31] Reiss, Steven P. Semantics-based code search. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pp. 243-253. IEEE, 2009.

[32] Swanson, Don R. Information retrieval as a trial-and-error process. *The Library Quarterly* (1977): 128-148.

[33] Al-Maskari, Azzah, and Mark Sanderson. A review of factors influencing user satisfaction in information retrieval. *Journal of the American Society for Information Science and Technology* 61, no. 5 (2010): 859-868.

[34] Luaces, Oscar, Jorge Dez, Jos Barranquero, Juan Jos del Coz, and Antonio Bahamonde. Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence* 1, no. 4 (2012): 303-313.

[35] Kelly, Diane. Methods for evaluating interactive information retrieval systems with users. *Foundations and Trends in Information Retrieval* 3, no. 12 (2009): 1-224.

[36] Hull, David. Using statistical testing in the evaluation of retrieval experiments. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 329-338. ACM, 1993.

[37] Jeon, Jiwoon, W. Bruce Croft, and Joon Ho Lee. Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pp. 84-90. ACM, 2005.

[38] Hao, Tianyong, and Eugene Agichtein. Finding similar questions in collaborative question answering archives: toward bootstrapping-based equivalent pattern learning. *Information retrieval* 15, no. 3-4 (2012): 332-353.

[39] Bernhard, Delphine, and Iryna Gurevych. Answering learners' questions by retrieving question paraphrases from social QA sites. In *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 44-52. Association for Computational Linguistics, 2008.

[40] Liu, Mingrong, Yicen Liu, and Qing Yang. Predicting best answerers for new questions in community question answering. In Web-Age Information Management, pp. 127-138. Springer Berlin Heidelberg, 2010.

[41] Muthmann, Klemens, and Alina Petrova. An Automatic Approach for Identifying Topical Near-Duplicate Relations between Questions from Social Media Q/A Sites. WSCBD'14 New York City, New York, USA, 2014.

[42] Philip, Bijil Abraham, Manas Jog, and Apurv Milind Upasani. Dr. Tux: A Question Answering System for Ubuntu users.

[43] Wang, Wei, Haroon Malik, and Michael W. Godfrey. Recommending Posts Concerning API Issues in Developer QA Sites. The 12th Working Conference on Mining Software Repositories (MSR 2015), Florence, Italy, May 1617, 2015.

[44] Bacchelli, Alberto, Luca Ponzanelli, and Michele Lanza. *Harnessing stack overflow for the ide*. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, pp. 26-30. IEEE Press, 2012.

Appendix A

User study instructions

This is a study to understand how we can improve and search related question in StackOverflow.

- Please start by filling the Google form: http://goo.gl/forms/fbPd7g4sxq
  - This information is merely for statistical purposes
  - We do not associate any information with any user. All the data we receive, from the forms and from the questions, is never associated with a specific user.
- You received a file including 20 links. For accessing these links you need to be on campus or connected to UCI VPN.
- Please only use Firefox, preferably latest version (other browsers might raise errors because of incompatibility with our html parser).
  - It might take a few seconds for every page to load. Please be patient.

- In each link you will see a question at the top of the page. Below you find two columns with ("Related Posts"). Please rate each ("Related Post") according to how it relates to the main question.
  - Please remember that "Yes" or "No" answers are more helpful than "Maybe" Answers.
  - It is possible to click on every title on each ("Related Post") to see the full question in the StackOverFlow website.
- At the end, please enter a quick comparison about which column overall you feel better relates to the original question.
- There is an optional field, if you want to mention difficulty/question/comment you have.
  - Even if it seems irrelevant, your feedback might help our research, so please feel free to write anything you want.
- After pushing submit button, you will see a page that says "Data is submitted".
  - Otherwise, data is not saved to the server; please let me know if you face this case.

Please let me know if you see any issue by replying to this email. Thank you very much for your help and for your valuable time.

Appendix B

Participants questionnaire

User Study Participant's Questionaire

* Required

Gender *

- ( ) Female
- ( ) Male
- ( ) Not interested to answer

Age (Year)

What is your current degree that you have or are studying? *

- ( ) PhD
- ( ) PhD Student
- ( ) Master
- ( ) Master Student
- ( ) Bachelor
- ( ) Bachelor Student

How long have you been studying Software/Computer science/Informatics or related fields? (Years) *

How proficient are you at Java programming? *

- ( ) very little
- ( ) little
- ( ) Intermediate
- ( ) proficient
- ( ) very proficient

What the field/fields you feel more comfortable in? *

Please select at least one

- [ ] Contextual software domains
- [ ] Software System Structures
- [ ] Software Functional Properties
- [ ] Extra-Functional Properties
- [ ] General programming languages
- [ ] Formal language definitions
- [ ] Compilers
- [ ] Context Specific Languages
- [ ] System description languages
- [ ] Development frameworks and environments
- [ ] Software configuration management and version control systems
- [ ] Software libraries and repositories
- [ ] Software maintenance tools
- [ ] Designing software
- [ ] Software development process management
- [ ] Software development techniques
- [ ] Software verification and validation
- [ ] Software post-development issues
- [ ] Collaboration in software development
- [ ] Search-based software engineering

Submit

Never submit passwords through Google Forms.

Powered by Google Forms

This form was created inside of University of California, Irvine.

Report Abuse - Terms of Service - Additional Terms

Appendix C

User study interface

Android OnClickListener Mechanism

I am beginner of Android and java .I cannot understand this portion of my code. May be it is to create an anonymous object. I cannot understand the mechanism of this code.Please help me.

```
btn.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub

    }
});
```

java android

Related Posts1

- N/A Difference between px, dp, dip and sp in Android?
- N/A Is there a unique Android device ID?
- N/A How to handle onClickListener in android?
- N/A Android SDK installation doesn't find JDK
- N/A Problems with OnClickListener for a tab in android
- N/A setOnClickListener(new OnClickListener(){})
- N/A Proper use cases for Android UserManager.isUserAGoat()?
- N/A Returning the position in list in Android with OnClickListener on Button
- N/A OnClickListener interface in android
- N/A Android onClickListener - Android Studio bug or my own mistake?

Related Posts2

- N/A Android: When I set an OnClickListener for a ListView
- N/A OnClicklistener not working
- N/A onClickListener onclicklistener Android Java programming
- N/A Android OnClickListener, intent and context
- N/A OnClickListener super
- N/A Android app with onClickListener not working
- N/A Android - Can we extends OnClickListener another Class?
- N/A View.OnClickListener usage as parameter in Android
- N/A Android - OnClickListener issue
- N/A Android - Scrollable TextView with OnClickListener

In overall which related list results seems better: Please select

Any Comment? (Optional)

Submit