Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science and Media

Task-Oriented Query Classification

Master's Thesis

Ehsan Fatehifar
Born Apr 30, 1988 in Ahvaz, Iran

Matriculation Number 115590

1. Referee: Prof. Dr. Benno Stein
2. Referee: Prof. Dr. Unknown Yet

Submission date: May 7, 2018

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, May 7, 2018

. . . . . .
Ehsan Fatehifar

Abstract

In this thesis, we focus on finding the proper task for the user's submitted query as fast as possible. We presume that a query log categorized by tasks is already available and focus on the problem of finding the right task for a new query. This will help the user accomplishing a search task or fulfilling an information need more satisfying. The main aspects of solving this problem are speed and accuracy.

Moreover, we created three different datasets which are publicly available. Two of them have approximately 100,000 queries and the third one contains more than 300,000 queries. These dataset have significantly more queries than available query logs.

To find the related task to a query, we experimented 4 methods: (1) Trie data structure to search through queries to find the right task, (2) a hash-based method called Minhash LSH, (3) Word2Vec models and (4) Elastic search. Also we investigated the idea of making pseudo-document from the queries in a task and representing the task with this pseudo-document. The obtained accuracy by Elastic search is higher than other methods. Also, it is a fast method and can define a task in $\sim$2 milliseconds in the given datasets, which makes it usable in real-time scenarios. We used Word Mover's Distance(WMD) algorithm to calculate the distance between two queries in Word2Vec models. Word2Vec models are accurate but slow for real-time operations. MinHash LSH, a hash-based method, is faster than Elastic search and Word2vec model, but its accuracy is lower.

The analysis of the queries in different datasets shows that there are no distinguishable pattern in these queries that can help us figure out if it is a hard query to find a task for or not.

Contents

Acknowledgements

A very special gratitude to Matthias Hagen and Michael Völske for their supervision and support.

To my wife, my parents and my sister: because I owe it all to you. Many Thanks!

Chapter 1

Introduction

There is lots of research on user behaviour when interacting with search engines that aims to improve the user experience. For users, the main goal of submitting search queries to search engines is to retrieve the information they want. A search engine should be able to fulfil the needs of the user quickly and precisely to keep the user satisfied.

Jansen et al. [2008] aimed to help the user by discovering the search intents in a search session. Gayo-Avello [2009] tried to improve the suggestions based on users' previous behaviour. Nogueira and Cho [2017] suggest reformulated queries to the user for getting better results.

In this thesis, we focus on finding the proper task for the user's submitted query as fast as possible, which has never been done before. We presume that a query log categorized by tasks is already available and focus on the problem of finding the right task for a new query. The important aspects are the speed and accuracy of the search. Considering the fact that single second delay in a website loading time can result in a 7 percent loss in conversion, and 40 percent of web users will abandon a website if it takes longer than 3 seconds to load[Forrester.com], it is vital for the retrieval method to be fast. However, just being fast is not enough.

In order to suggest meaningful and useful queries, the accuracy should be taken into account. Each query contains almost four words in average[Statista.com]. Therefore, most queries are short and ambiguous and that makes the classification harder. Moreover, given the fact that 15 percent of searches on Google every day are new[Google.com], we should have a solution for the queries, which are new and have no related task in the corpus.

Some users look for a web page containing specific information about a person or a subject. For this kind of queries, search engines solved the problem. The search engines work quite well in this area. Usually, the answer is on the first page of retrieved results. The search engine can even take care of the

spelling errors. But some of the queries are more complicated and need more than a simple answer.

Suppose that a user is looking for information to do a multi-step task. The answer to this query is not just one result, word or link. The question is too general to be answered like this. In such cases, users often submit queries subsequently to gather all of the information they need. Search engines can help the user by suggesting queries that not only have similar words but also are related to the same task. These queries are helpful hints to accomplishing the task.

A very good example of this problem is the query 'plan a wedding'. Planning a wedding consists of different subtasks. For instance, the user needs further information about the following tasks: buy a dress, reserve a hotel, choose a band, order flowers, wedding ring catalogue. By the time of writing this thesis, after submitting a query, search engines show a result page and suggest related queries.

The only search engine that provides the user something more than a traditional result page, is Google. For some queries, in addition to the search results and query suggestions, Google shows a step-by-step guide. This step-by-step guide is more related to the query and its corresponding task. But query suggestions are mostly appeared on the result page based on word similarity with the query. They are not related to one task in most scenarios.

The ability to guide a user to accomplish a search task faster and to inform a user about other possible steps of the task simultaneously takes the usability of search engines to the next level.

In our experiments, we used different methods(e.g. Locality-sensitive hashing, Word2Vec models, etc.) to find out which one results in the best performance and is suitable for finding the related task for a query.

Since we don't have access to search engines' task-categorized query logs, we must make datasets ourselves. The bigger the dataset, the closer to real-world scenarios it is. We use available AOL annotated query logs, WEBIS textReuse Corpus, TREC Task track and TREC Session Track logs. In addition, we consider websites like WikiHow, that show step-by-step instructions for doing a task as an extra source that has tasks and queries related to the tasks.

This thesis is divided into five chapters: Chapter 2 discusses in more detail the related work and previous literature in this field and the used concepts. Our literature survey shows that the problem hasn't been tackled on the same scale before. In Chapter 3 we will talk about approaches that we used. We use (1) Trie data structure to search through queries to find the right task, (2) a hash-based method called Minhash LSH, (3) Word2Vec models, (4) Elastic search and (5) the idea of making a pseudo-document from queries in a task. The experiments that carried out are investigated in Chapter 4. We reached an

accuracy of ∼0.78 on two of three datasets that we prepared by using Elastic search. Also, the time needed for finding a task is ∼2-3 milliseconds, which shows it can be used for this task in real-time. Moreover, MinHash LSH has a satisfying performance with a high speed but lower accuracy. Word2vec models have a higher accuracy compared to MinHash LSH but its speed issue makes it inefficient. These experiments are based on methods described in Chapter 3. Besides, we describe how we combine available datasets, to create more suitable datasets for our purpose. In the final chapter, future steps and conclusions will be discussed.

Chapter 2

Related Work

In this chapter, we will write about the Task-Oriented Search and the terms used in the field. Next, we are going to survey the previous work in Task-Oriented Query Classification and its contribution to this thesis. Then we talk about the available data and how they helped us to build a bigger dataset. In the end, a brief history of the methods that we decided to use for doing the experiments will be presented.

2.1 What is a Task?

Users submit queries to search engines to satisfy their information needs. Jones and Klinkner [2008] defined a Search Task as follow:

Definition 2.1.1 *Search Task. A search task is an atomic information need resulting in one or more queries.*

Awadallah et al. [2014] extended Definition 2.1.1 and defined a complex search task as:

Definition 2.1.2 *Complex Search Task. A complex search task is a multi-aspect or a multi-step information need consisting of a set of related subtasks, each of which might recursively be complex.*

According to other research, the task or the goal in a search session can be defined as the cluster of information on different aspects of a query that user groups want to obtain in order to retrieve the useful information[Lu et al., 2013].

Spink et al. [2006] defined a Session as:

Definition 2.1.3 *Session. A session composed of queries issued by users having in mind a particular task/goal.*

Example 2.1.1 *Let's say a user wants to 'plan a wedding'. So, the user will submit some queries like 'how to plan a wedding', 'plan a wedding check-list', 'booking a band', 'wedding flowers' consequently. In this example, 'plan a wedding' is a complex task and search session consists of all the mentioned queries.*

For a complex task, the user's goal at the time of searching is not completely clear. The goal of a Web search application is to be effective and efficient. So, if we could be able to help the user finding what he/she needs, we can improve the search results relevancy and user experience. As you can see in Example 2.1.1, when the user submits one of the queries, the search engine will return the results which are related to different aspects of the search task that help the user to accomplish his/her search goals.

Task definition is not absolute and is completely another problem that should be taken care of separately. For example, "reserving a hotel" can be a subtask for these two tasks: "plan a wedding" and "trip to London". Or it can be a task itself. In this thesis, we are not considering the task structure as hierarchical. We consider the tasks as flat and do the experiments. So, we are using Definition 2.1.1.

2.2 Task Extraction from Query Logs

For our work, we need a search engine log, in which the queries are categorized by the task they are related to. The available logs are not suitable for our purpose. In these logs, there is information about which URL a user clicked on and the time of the search. In case the search logs are categorized, the categorization is too general. For example, a query is related to the 'Health' topic. But we want something more specific; e .g. if the query is related to 'healthy diet' or 'physical problem'. Session and task extraction from query logs became an active field and researchers decided to do that automatically.

Automatic task and session detection is not an easy task, because assessing millions of queries manually is very difficult and time-consuming. When there is a high number of tasks, one cannot keep track of the query relations all over the query log. One of the reasons for that is multitasking during a search session. A study shows that 75% of the users searching for more than one goal during a search session[Lucchese et al., 2011]. Nowadays, multitasking is common in Web searching. A user may have a single task or multiple tasks in mind when he/she starts a search session. Even, there might be new queries related to a new task submitted during the search process. This is why it is

not enough just to detect session. That means even when someone can detect sessions 100% accurate, then he or she should be able to deal with multiple tasks in a session and recognize them. Considering the short length of queries, it makes it even more difficult to find the similarity and semantic relation between queries. Now we are going to mention the methods that tried to detect sessions and tasks automatically. The simplest method for session detection is time-based. Researchers tried different time intervals 30 minutes, 60 minutes and 120 minutes and consider all the queries in between as the same session. They also considered the queries in each session as if they are belonging to the same task. Because of the fact that even if they can separate the sessions, they still have to deal with task separation considering the multitasking during a search session, this method has a very low accuracy. Some researchers tried to use clicked URL as an extra information to be able to separate tasks in a search session, but this information is not always available. Broder [2002] manually classified a small set of queries into transactional, navigational, and informational tasks using a pop-up survey of AltaVista users, and manual inspection. Beitzel et al. [2007] took a random sample of 20,000 queries out of the entire AOL query stream for 1 week. These queries were manually classified into the same set of 18 categories by an AOL team of human assessors.

Jansen and Booth [2010] investigated a methodology to classify automatically Web queries by topic and user intent. Taking a 20,000 plus Web query dataset sectioned by topic[Beitzel et al., 2007], in the first level, they manually classified each query using a three-level hierarchy of user intent which were introduced by Broder [2002]. Then for the second level classification, they defined user intent as the expression of an effective, cognitive, or situational goal in an interaction with a Web search engine. Rather than the goal itself, user intent is concerned with how the goal is expressed because the expression determines what type of resource the user desires in order to address their underlying need. The topics they defined are too general. There are 20 topics, e.g. auto, sports. We want more specific topics for our purpose.

There are many examples that such mapping can lead to significant improvements in retrieval performance. For example, successful identification of topic can help alleviate synonym issues (e.g., cobra the snake versus cobra the anime)[Jansen and Booth, 2010].

Verma and Yilmaz [2014] explored entity based task extraction from search logs. They claimed that by considering the entities and their associations one can extract better semantics from user queries. They tried to find entity-oriented tasks for each category, by populating words that co-occur with entities from that category and represented tasks as a collection of diverse but conceptually related terms. Existing work extracts tasks from independent sessions, thus providing information only about a single user. Such tasks have

limited applications as they do not give a complete picture about tasks that exist globally. However, entity-oriented tasks can be extracted from search sessions across several users. Such a global set of tasks can benefit related search applications too. For instance, it can be used to find similar users by mining their task histories or for query suggestions[Verma and Yilmaz, 2014].

In their later work [Verma and Yilmaz, 2016], they said: While text-based features do not exploit entities directly, task dictionaries do not provide a concise or distinct representation of tasks. We overcome these shortcomings by extracting category oriented tasks by exploiting properties of an existing, publicly available category hierarchy(DBpedia). Based on an empirical evaluation they showed that category based task extraction results in more accurate and useful tasks.

One of the most relevant and helpful studies has been conducted by Hagen et al. [2013] which proposed 'Search Mission Detection' which aims to identify those queries a user submits for the same information need. They claimed their approach is applicable within the time-critical online scenario, where a search engine tries to support users by incorporating knowledge about their search history on the fly. A search mission is characterized by logical sessions, multitasking behaviour, and hierarchical goals. A logical search session is characterized by consecutive queries for the same information need within a physical session. A physical search session is characterized by the time gap between queries. To be able to evaluate their approach, they needed a corpus of manually labelled query logs. Therefore, they created a new corpus. The provided dataset is really helpful for us and is a base for making good dataset for our research. It will be discussed in section 2.3.

Yang and Nyberg [2015] attempted to bridge the gap between two evolving research areas: development of procedural knowledge bases (such as wikiHow) and task-oriented search. They tried to extract useful steps from wikiHow to improve the suggested suggestions to the user. We got the idea of using wikiHow from this article, but in another way. We do not use the content of wikiHow pages, because we don't want to extract the steps. For each query, we want to find a set of related queries. So, we can make a dataset based on these query pairs which are related to the same task.

Speed is also an important factor for us. In order to make the approach executable in real-time, we do not get help from Web pages or other contents from external sources like some query classification approaches which are applicable in post-classification, . As a result, we just rely on the query itself.

2.3 Available Data

We chose the following datasets to work on:

- Webis-SMC-12[1]: Webis Search Mission Corpus 2012
- Lucchese AOL annotation corpus[Lucchese et al., 2011]
- Webis-TRC-12[2] : Webis Text Reuse Corpus 2012
- TREC Tasks Track 2015[3]
- TREC Tasks Track 2016[4]
- TREC Session Track 2014[5]
- wikiHow website[6]

Hagen et al. [2013] choose the Gayo-Avello [2009] sample as their basis. Gayo-Avello's corpus consists of 11 484 queries from 215 users sampled from the 2006 AOL query log which is divided into 4040 sessions by a human annotator. Hagen et al. [2013] extracted all queries of the 215 users contained in the Gayo-Avello sample. They removed the few queries that are empty or just a URL. Also, the queries from the 88 users that submitted less than 4 queries in total. In the end, the Webis-SMC-12 corpus contains 8840 queries from 127 users. Two human annotators divided this sample into 2881 logical sessions and 1378 missions. The main drawback of Gayo-Avello's corpus is no relationships between different sessions on the same information need are annotated. Plus, they removed some queries with no reason.

Lucchese et al.'s [2011] corpus consists of 1424 queries from 13 users also sampled from the AOL log and clustered from 307 time-gap sessions. It is manually annotated. But, it does not contain all queries from the sampled users. Their sample dataset was based on the 500 user sessions with the highest number of queries. They just considered queries in a very small time gap(i.e., the first week of user activities).

Potthast et al. [2013] hired writers to write articles on each of the 150 topics used at the TREC Web Tracks 2009-2011. They logged the author's interaction with a search engine and stored their queries. Webis-SMC-12 interaction log contains 3826 unique queries for 150 topics.

[1] `http://www.webis.de/data/data.html`
[2] `http://www.webis.de/data/data.html`
[3] `http://trec.nist.gov/data/tasks2015.html`
[4] `http://trec.nist.gov/data/tasks2016.html`
[5] `http://trec.nist.gov/data/session2014.html`
[6] `http://www.wikihow.com`

TREC Tasks Track 2015 dataset contains 50 tasks with some suggestions, which are queries that may help to accomplish the user's task. In total it contains 547 queries. TREC Tasks Track 2016 dataset contains 50 tasks and 405 queries. Here is an example:

Example 2.3.1 *A sample task from TREC Tasks Track 2015 data.*
Task id: 1
Task: getting organized at work [I need to get organized at work]
Hints:

- *Checklist for getting organized at work*
- *How to organize office desk*
- *Tips for getting organized at work*
- *Organize schedule at office*
- *How to create a todo/task list*
- *How to keep a calendar of scheduled meetings and travel*
- *How to set deadlines and goals*
- *How to organize your work space*
- *How to log the time you spend*
- *Methods to track your progress towards goals*
- *How to set up a filing system with a binder or folders*

TREC Session Track 2014 dataset contains information on user sessions interacting with a search engine looking for information on a specific topic. The dataset contains 60 topics and 4666 queries.

Example 2.3.2 *A sample topic from TREC Session Track 2014 data.*
<topic num='5'>
<desc>
Suppose you are writing an article about face transplants. You want general information about face transplants and how face transplants affect the lives of patients. Find web pages about face transplants.
</desc>

</topic>

Queries in a session for topic '5':

- *face transplants*
- *face transplant wiki*
- *face transplant results*

In Table 2.1 you can see the statistics of available data.

Corpus	WEBIS AOL	Lucchese AOL	WEBIS TRC	TREC Track15	TREC Track16	TREC Session14
Tasks	1298	233	150	50	50	60
Queries	8840	1424	13881	547	405	4666
Uniq. Queries	3736	792	3826	547	405	3248
Max. Q. in T.	138	55	122	20	15	100
Min. Q. in T.	1	1	1	4	5	16
Avg. Q. in T.	2.88	3.4	25.21	10.94	8.1	54.13
Std. Q. in T.	5.75	4.69	18.25	3.64	2.17	18.21

Table 2.1: Statistics of available datasets.

Since these datasets do not have a large number of queries and have some tasks in common, we merge them. Also, we get suggestions for each query from different search engines to extend the dataset. The details are available in section 3.1

2.4 Methods

2.4.1 MinHash and LSH

MinHash (or the min-wise independent permutations locality sensitive hashing scheme) is a Hash-Based technique for measuring the similarity between two sets. This technique replaces large sets by much smaller representations called "signatures". Also, we can compare two sets by producing their signatures and measure the distance between signatures using Jaccard similarity. But keep in mind, the signatures do not exactly represent the sets. They are close estimations. The bigger the signature is, provides a more accurate estimation.

In our case, we consider each query as a set and make the following sample characteristic matrix:

The columns of the matrix correspond to the queries, and the rows correspond to the set of all words exist in the corpus. There is a 1 in row r

Element	Q1	Q2	Q3
W1	0	1	0
W2	1	0	1
W3	1	0	0

Figure 2.1: A matrix representing a query set.

and column c if the element for row r is a member of the set for column c. Otherwise, the value in position (r, c) is 0. To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows. The minhash value of any column is the number of the first row, in the permuted order, in which the column has a 1. [Leskovec et al., 2014]

There is a remarkable connection between minhashing and Jaccard similarity of the sets that are minhashed.The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets. [Leskovec et al., 2014]

The Jaccard similarity coefficient is a commonly used indicator of the similarity between two sets. For sets A and B it is defined to be the ratio of the number of elements of their intersection and the number of elements of their union:

$$J(A, B) = \frac{(A \cap B)}{(A \cup B)}$$

Even though we can use minhashing to compress large documents into small signatures and preserve the expected similarity of any pair of documents, it still may be impossible to find the pairs with the greatest similarity efficiently. The reason is that the number of pairs of documents may be too large, even if there are not too many documents. we need to focus our attention only on pairs that are likely to be similar, without investigating every pair. [Leskovec et al., 2014]

There is a general theory of how to provide such focus, called locality-sensitive hashing (LSH) or near-neighbor search. One general approach to LSH is to "hash" items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are. We then consider any pair that hashed to the same bucket for any of the hashings to be a candidate pair. We check only the candidate pairs for similarity. If we have minhash signatures for the items, an effective way to choose the hashings is to divide the signature matrix into b bands consisting of r rows each. [Leskovec et al., 2014]

MinHash LSH has a good performance compared to other methods. It is amazingly fast but not as accurate as Elastic search.

2.4.2 Elastic search

Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements.[elastic.co]

It is developed in Java and is released as open source under the terms of the Apache License. Elasticsearch can be used on all documents across all types. It provides scalable search and has near real-time search.[elastic.co]

Elasticsearch, the most popular enterprise search engine[db engines.com], is a Lucene-based search engine. It provides a distributed, full-text search engine with schema-free JSON documents.

Elasticsearch stores complex entities as structured JSON documents and indexes all fields by default, providing a higher performance. It is schema-free and stores a large quantity of semi-structured (JSON) data in a distributed fashion. It also attempts to detect the data structure, index the data present and makes it search-friendly.

Elasticsearch performs linguistic searches against documents and returns the documents that matches the search condition. Result relevancy for the given query is calculated using TF/IDF algorithm.

2.4.3 Trie Data Structure

De La Briandais [1959] described Trie for the first time. The name 'TRIE' is coined from the word 're**trie**ve'. Trie is the data structure similar to Binary Tree. It is based on the prefix of a string. The prefix of a string means any n letters from the start of a string with the length l, where $n < l$. In this tree, all the descendants of a node have a common prefix (associated with that node). The nodes do not contain key values. Key values will be defined by the position of the node in the tree.

We use this data structure to be able to search fast through queries. It is a simple method and can be used as a base performance measure compared to more complicated methods.

2.4.4 Bag of Words Model(BoW)

The Bag of Words model is used in natural language processing and information retrieval. In this model, extracted features from a text are represented as sets of words disregarding grammar and word order but with a measure of the presence of known words. A text (such as a sentence or a document)

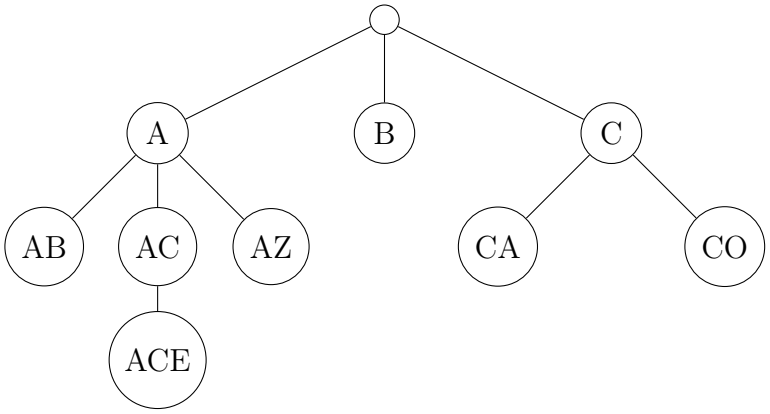

Figure 2.2: A trie for keys "A","AB", "AC", "ACE", "AZ", "B", "C", "CA", and "CO".

is represented as the bag of its words, disregarding grammar and even word order.

Any information about the order or structure of words in the document is discarded. That's why It is called a "bag" of words. The extracted features by BoW can be used in machine learning algorithms.

The Bag of Words model learns a vocabulary from all of the documents, then models each document by counting the number of times each word appears.

Here is a famous example for BoW:

Example 2.4.1 *BoW example.*

Consider the following two sentences:

Sentence 1: "The cat sat on the hat"

Sentence 2: "The dog ate the cat and the hat"

From these two sentences, our vocabulary is as follows:

{ the, cat, sat, on, hat, dog, ate, and }

To get our bags of words, we count the number of times each word occurs in each sentence. In Sentence 1, "the" appears twice, and "cat", "sat", "on", and "hat" each appear once, so the feature vector for Sentence 1 is:

{ the, cat, sat, on, hat, dog, ate, and }

Sentence 1: { 2, 1, 1, 1, 1, 0, 0, 0 }

Similarly, the features for Sentence 2 are: { 3, 1, 0, 0, 1, 1, 1, 1}

2.4.5 Word Embedding Model

Word Embedding is another approach for representing text data and differs from other feature extraction methods. In this model, the words will be represented based on their usage. As a result, despite the bag of words model, the words with same meaning, will have the same representation. In the bag of words model, if we want to represent different words similarly, we have to explicitly define the relationship of the related words, because in BoW the usage of the word is not taken into account. Obviously, it is impossible for big text collections.

A goal of statistical language modelling is to learn the joint probability function of sequences of words in a language. A fundamental problem that makes language modelling and other learning problems difficult is the curse of dimensionality. [Bengio et al., 2003]

Bengio et al. [2003] proposed "to fight the curse of dimensionality by learning a distributed representation for words which allows each training sentence to inform the model about an exponential number of semantically neighbouring sentences. Each training sentence informs the model about a combinatorial number of other sentences".

Associate with each word in the vocabulary a distributed word feature vector. The feature vector represents different aspects of the word: each word is associated with a point in a vector space(Figure 2.3). The number of features is much smaller than the size of the vocabulary.[Bengio et al., 2003]

Collobert and Weston [2008] proposed a general deep NN architecture for NLP that formed the foundation for many current approaches. But the landmark in the history of word embedding is Word2Vec model created by Mikolov et al. [2013a].

2.4.6 Word2Vec Model

Word2vec is the most popular word2vec model. Word2vec is a group of related models that are used to produce word embeddings. It learns a vector representation for each word using a (shallow) neural network language model. It takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.

Despite Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA), word2vec is highly scalable. It learns high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary. [Mikolov et al., 2013a]

To produce accurate vectors for the words, it is not enough for a model to

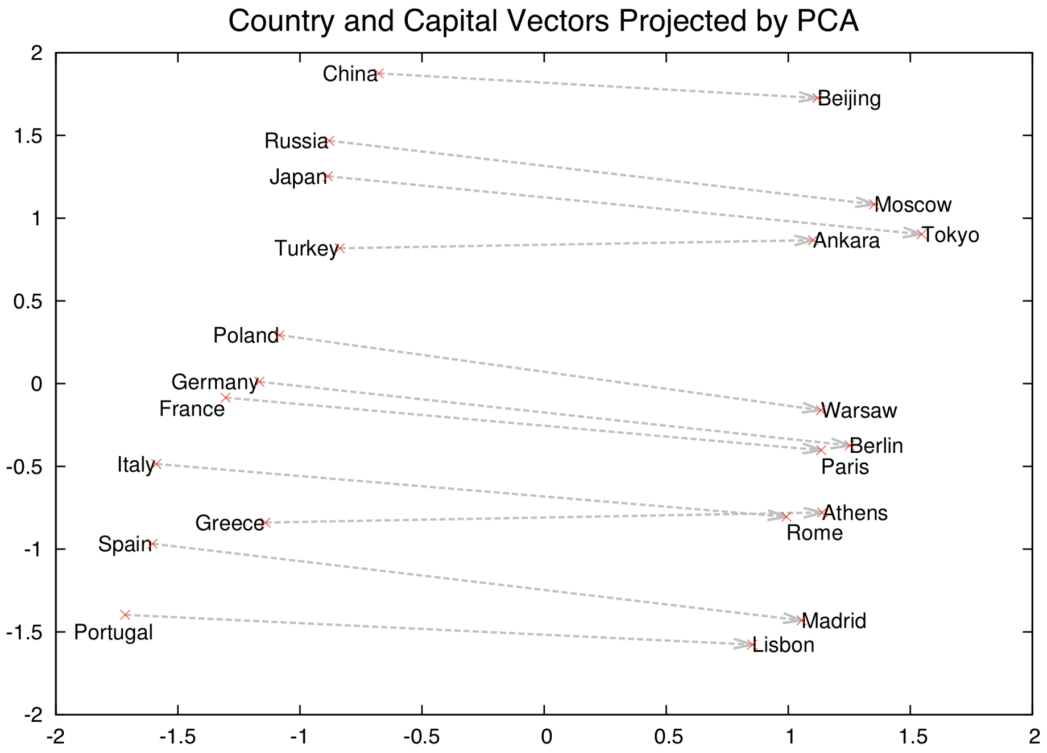

Figure 2.3: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.[Mikolov et al., 2013b]

consider only previous words for its prediction. Word2vec solves this problem by utilizing either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word w_t from a window of surrounding context words. Like the bag-of-words model, the order of the context words does not influence prediction. In the continuous skip-gram architecture, the model uses the current word w_t to predict the surrounding window of the context words. [Mikolov et al., 2013a] (See Figure 2.4)

The skip-gram architecture weighs nearby context words more heavily than more distant context words.

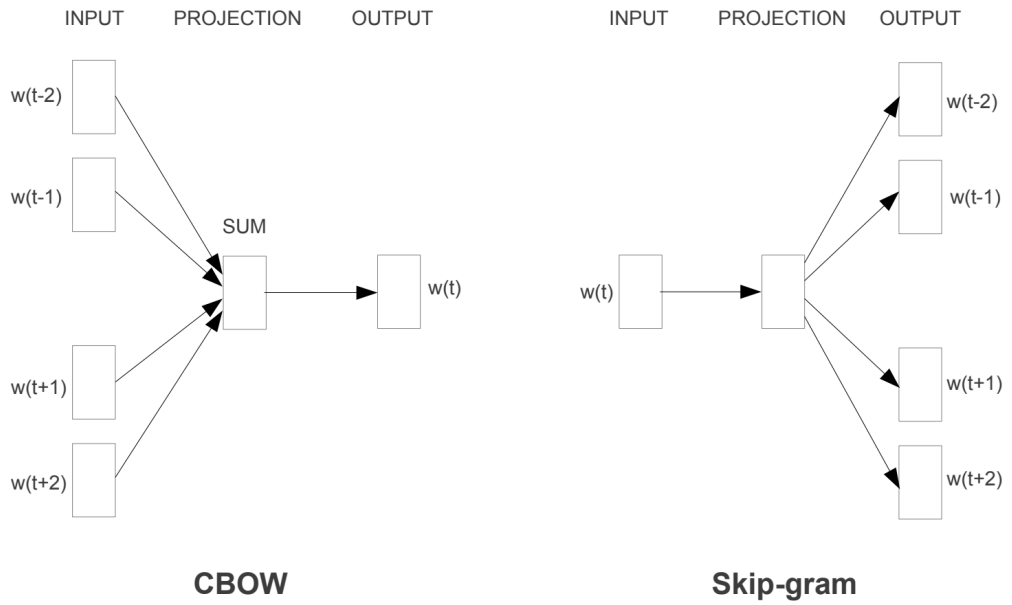

Figure 2.4: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.[Mikolov et al., 2013a]

2.4.7 Word Mover's Distance (WMD)

WMD is an instance of the Earth Mover's Distance(EMD), a well-studied transportation problem for which several highly efficient solvers have been developed. Kusner et al. [2015] presented the Word Mover's Distance (WMD), a novel distance function between text documents. Their work is based on recent results in word embeddings that learn semantically meaningful representations for words from local co-occurrences in sentences.

The WMD distance measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to "travel" to reach the embedded words of another document.[Kusner et al., 2015]

In Figure 2.5 you can see how to compute the distance between two sentences: "Obama speaks to the media in Illinois." and "The President greets the press in Chicago.".

In Chapter 4, we show that WMD is more efficient and accurate in calculating distance between queries than measuring simply distances between each word pairs in two queries and average them.

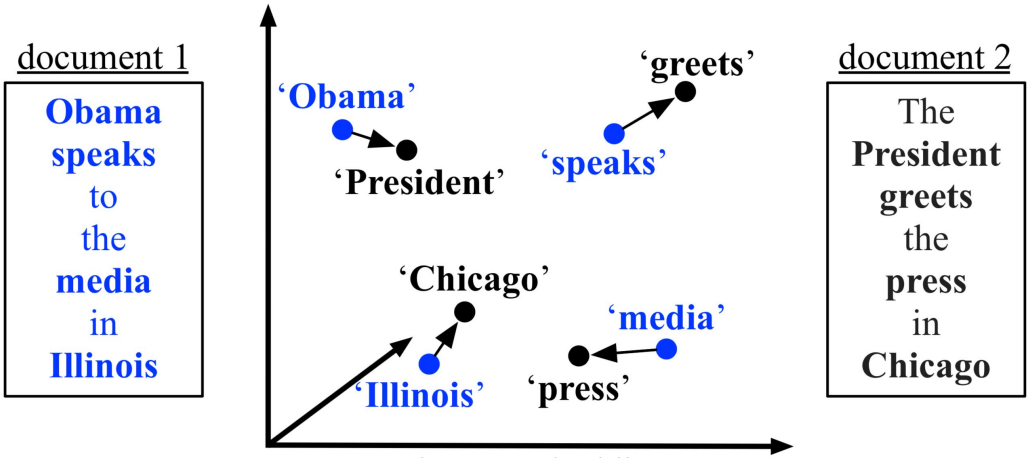

Figure 2.5: An illustration of the word mover's distance. All non-stop words (bold) of both documents are embedded into a word2vec space. The distance between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2.[Kusner et al., 2015]

Chapter 3

Approaches and Data

In this chapter we will describe the preparation process of the data we want to use. After that, the experiment criteria for each of the methods discussed in section 2.4 will be explained.

We created these datasets because the available data(section 2.3) are not suitable for our experiments. They are not big enough or the categories are too general. Some corpora needed more manual annotation.

Also, we are going to describe the methods that we used for doing the experiments.

3.1 Data

We used available datasets discussed in section 2.3 as a base to develop bigger and more useful datasets for our purpose. We created three datasets based on them:

1. D1: Combination and merge of WEBIS TextReuse Corpus, TREC Tasks Track 2015, TREC Tasks Track 2016 and TREC Session Track 2014 after manual verification(section 3.1.1)

2. D2: Merging Webis-SMC-12 and Lucchese AOL annotation corpus after manually re-annotation (section 3.1.3)

3. D3: WikiHow data based on bidirectional relationship of questions(section 3.1.3)

3.1.1 D1 Dataset

For making this dataset, we used following datasets:

- WEBIS TextReuse Corpus
- TREC Tasks Track 2015
- TREC Tasks Track 2016
- TREC Session Track 2014

The statistics of each dataset is available in table 2.1 in section 2.3.

In total, they have 310 topics. We manually checked the topics and found out that they have 34 topics in common. After merging them, there are 8,026 unique queries in 276 topics. The advantage of these corpora is that the topics are defined clearly and the queries related to them are completely related. So, this corpus has no possible noise and has a high quality. But 8,000 queries are not enough for testing different methods on a higher scale.

To extend this corpus we got suggestions for each of the queries from four search engines:

- Google
- Bing
- Ask
- AOL

After getting suggestions, corpus contained 98,676 unique queries in 276 topics. Compared to each of the four datasets, the difference in size of the result is significant. For example WEBIS-TRC-12 has just 3,826 unique queries.

3.1.2 D2 Dataset

In order to build this dataset, we used following datasets:

- Webis-SMC-12
- Lucchese AOL annotation corpus

The statistics of each dataset is available in table 2.1 in section 2.3.

Despite the datasets used to build D1, the topics in these datasets were not defined prior to the search. Tasks are manually annotated by looking at a part of AOL search logs. The queries were categorized based on a task that user wanted to accomplish in one or multiple session. In both corpora, the common tasks between users were not taken into account. Also, we re-annotate them manually again to solve this problem. Moreover, in order to merge them, we had to find the common tasks between two corpora, which was also done manually.

Corpus	Merged	+Suggestions
Topics	310	310
Uniq. Topics	276	276
Uniq. (Queries, Task)	19499	102171
Uniq. Queries	8026	98676
Max. Queries in Task	141	2430
Min. Queries in Task	1	23
Avg. Queries in Task	53.51	358.82
Std. Queries in Task	38.42	280.03
Tasks have 1 Query	2	0
Tasks have 2 Queries	0	0
Tasks have 3 Queries	0	0
Tasks have 4 Queries	2	0

Table 3.1: D1 dataset statistics.(Combination and merge of WEBIS TextReuse Corpus, TREC Tasks Track 2015, TREC Tasks Track 2016 and TREC Session Track 2014 after manual verification)

After the merge, in these corpora, there were 1531 tasks with 102 tasks in common. They don't have any user in common. Also, corpus contained 4,528 unique queries belong to 1,429 tasks.

Like the first dataset(D1) we got suggestions from Google, Bing, Ask and AOL for each query. After getting suggestions, corpus contains 90,282 unique queries in 1,429 tasks. Like D1, this dataset is also big enough for doing the experiments.

3.1.3 D3 Dataset

For making this dataset, we crawled an English wikiHow data dump, which contains 198,163 questions. WikiHow suggests for each question a list of related questions, but after verifying these suggestions we observed that they are not necessarily related. Especially for our purpose, they are not related to the same task.

As a result, we decided to consider questions related to the same task if they are both mentioned in each other's suggestions. After applying this criterion, 15,914 of the queries remained and 7,202 tasks were identified.

We got suggestions from Google, Bing, Ask and AOL for each query. After getting suggestions, corpus contains 331,389 unique queries in 7,202 tasks. This is the biggest dataset available.

The number of queries in a task is an important factor. While looking for a task to assign to a query, a task with more queries has a higher chance.

Corpus	Merged	+Suggestions
Tasks	1429	1429
Uniq. (Queries, Task)	4528	93391
Uniq. Queries	4528	90111
Max. Queries in Task	147	2810
Min. Queries in Task	1	7
Avg. Queries in Task	3.17	65.62
Std. Queries in Task	6.38	114.41
Tasks have 1 Query	653	0
Tasks have 2 Queries	325	0
Tasks have 3 Queries	144	0
Tasks have 4 Queries	104	0

Table 3.2: D2 dataset statistics.(Merging Webis-SMC-12 and Lucchese AOL annotation corpus after manually re-annotation)

we keep tasks which have less than 5 queries to do the experiments. Also, including them in experiments can help us understand how to deal with them. The only dataset that has tasks with less than 5 queries is D3.

Corpus	WikiHow	+Suggestions
Tasks	7202	7202
Uniq. (Queries, Task)	15914	331389
Uniq. Queries	15914	331389
Max. Queries in Task	22	448
Min. Queries in Task	1	2
Avg. Queries in Task	2.21	46.01
Std. Queries in Task	1.36	28.75
Tasks have 1 Queries	2436	0
Tasks have 2 Queries	2731	2
Tasks have 3 Queries	1025	6
Tasks have 4 Queries	531	5

Table 3.3: D3 dataset statistics.(WikiHow data based on bidirectional relationship of questions)

3.1.4 Data Overview

Now, we have three different datasets with different sizes. D1 contains predefined tasks and queries related to them. D2 contains queries from different

users which manually divided into tasks. D3 contains data extracted from WikiHow.com.

In table 3.4 you can see the overview of the collected data.

Corpus	D1	D2	D3
Tasks	310	1429	7202
Uniq. Tasks	276	1,429	7202
Uniq. (Queries, Task)	102,171	93,391	331,389
Uniq. Queries	98,676	90,111	331,389
Max. Queries in Task	2,430	2,810	448
Min. Queries in Task	23	7	2
Avg. Queries in Task	358.82	65.62	46.01
Std. Queries in Task	280.03	114.41	28.75
Tasks have 1 Query	0	0	0
Tasks have 2 Queries	0	0	2
Tasks have 3 Queries	0	0	6
Tasks have 4 Queries	0	0	5

Table 3.4: Overview of D1, D2 and D3 datasets statistics.

As you can see D3 is the biggest dataset with the least standard deviation regarding the number of queries in a task.

Queries contain approximately 3.5 to 5 words in each of the corpora. In corpus D1, the standard deviation of the number of words in a query is higher, because some queries in TREC Tasks Track contain a description or extra information about that query. For example, "look for signs of cyberbullying (e.g. anxiety about using the computer, closing or blocking the computer screen, changes in using mobile devices)". We kept the queries like this as they are because the search engines returned good suggestions for them.

In D2 and D3 there are queries with one character. In D2 they may be submitted accidentally(e.g. user press enter before typing the whole query). In D3, some of them are suggestions from search engines. We keep these queries because want to test on a noisy data. However, these are less than 1% of the queries.

In table 3.5 more details are available on the query statistics in the corpora.

3.2 Trie

Trie implementation is simple. We can modify the data structure in case any change happens(e.g. adding new queries to the corpus). Regarding this,

Corpus	D1	D2	D3
Words	445,382	332,752	1,647,644
Unique Words	37,816	42,340	62,601
Max. Stopwords in Query	0	0	0
Min. Stopwords in Query	20	10	12
Avg. Stopwords in Query	0.71	0.35	1.28
Std. Stopwords in Query	1.14	0.74	1.44
Min. Words in Query	2	1	1
Max. Words in Query	46	17	19
Avg. Words in Query	4.36	3.56	4.97
Std. Words in Query	2.63	1.56	2.03
Min. Length of Query(characters)	4	1	1
Max. Length of Query(characters)	309	91	116
Avg. Length of Query(characters)	27.90	22.57	27.42
Std. Length of Query(characters)	16.52	9.43	9.71

Table 3.5: Overview of D1, D2 and D3 datasets queries statistics.

making a Trie is fast and can be done simultaneously with the arrival of new queries.

We determine the task for each query by finding the longest prefix for that query. In the following example, we show how it works.

Example 3.2.1 *Trie structure and retrieval*

Suppose we have three queries: q1: how to organize q2: how to organize your desk q3: how to become a doctor

q1 and q2 are related to task 1 and q3 is related to task 2.

For each word in the query a node and a list values should be created. here is how tree is made:

```
trie['how'] = [1,2]
trie['how/to'] = [1,2]
trie['how/to/organize'] = [1]
trie['how/to/be'] = [1]
trie['how/to/be/organized'] = [1]
trie['how/to/organize'] = [1]
trie['how/to/organize/your'] = [1]
trie['how/to/organize/your/desk'] = [1]
trie['how/to/become/a/doctor'] = [2]
```

As you see, the word 'how' and 'how to' are common between task 1 and 2.

Now, if we query this tree and get the longest prefix, we get the following result for each of them:

```
'how to': ('how/to', [1, 2])
'how to be an organized person': ('how/to/be', [1])
'organize': (None, None)
'become a doctor': (None, None)
```

So, for 'how to' the longest prefix is 'how/to' which is related to task 1 and 2. 'organize' does not have any prefix because none of the queries started with the word 'organized'.

Making Trie takes around 30 seconds for dataset D1 and 90 seconds for dataset D3 which has three times more queries than dataset D1. It is significantly faster than indexing in the MinHash LSH method. Finding a task can be done in microseconds. However, the accuracy of this method is quite low. It was expected because the search method in a Trie is very basic. If the queries don't have the same prefix there is no chance that they match. We can't expect this method to take the semantics of the queries into account. Maximum achieved accuracy by this method is 0.14 for dataset D1. The detailed results are available in section 4.1 table 4.1.

We use this method as the simplest possible method to see how good we can solve the problem. Also, it gives us a clue about the difficulty of a dataset. If we achieve a high accuracy in finding related task using this method, a conclusion would be that the dataset is easy. So, this method can be used to measure the dataset's difficulty. In section 4.1 you can see the details of experiments on different datasets.

3.3 MinHash LSH

MinHash LSH is a fast method which has an acceptable accuracy. We carried out the experiments on the three datasets introduced in sections 3.1.1, 3.1.2 and 3.1.3. The goal of the experiment is to find a related task for a query as fast and as accurate as possible.

For each query in the corpus, we take the query out from the corpus and look into the rest of the corpus to find the related task. Since MinHash returns a list of similar queries for each query, it is possible to choose the most common task between them or just get top task returned.

Hashing the queries and build a corpus is fast. This method takes 214 seconds for dataset D3, 53.8 seconds for D2 and 61 seconds for D1. You can see the detailed results in section 4.2 table 4.2. As it is shown in table 4.2, the time for returning the results will change almost linearly with the growth of the size of the corpus.

Also, adding and removing elements from the hashes is fast and easy. A query can be hashed and added in less than 3 milliseconds, no matter how big a corpus is.

The accuracy of this method is 0.6570 for dataset D1, 0.6586 for dataset D2 and 0.3750. Compared to other corpora, it performed poorly on WikiHow data.

As we discussed in section 2.1, a query is not always related to just one task. It may relate to more than one task. Based on this assumption, we considered the second predicted task by MinHash LSH method and then recalculate the accuracy. Accuracy increased in each case around 0.1.

In another experiment, we removed tasks which have less than 50 queries. Also, we did the experiment on the tasks which have more than 100 queries. These experiments resulted in a better accuracy, especially in dataset D3. That means the bigger the numbers of queries in tasks are, would be a better help to find the related task for a query.

We also did another experiment by determining the task based on just the most similar query instead of getting the most common task. The results are available in table 4.6. The most common task strategy performs significantly better, which is an improvement with an almost no cost. Because the time we spend to find the most common task is trivial compared to search for similar queries.

3.4 Word2Vec Model

Word2Vec model is working well in finding similarity between two words. It also can define the semantic similarity. This model should be trained on a big corpus, to be able to cover an extended range of words. There are lots of pre-trained models available.

One problem with this model is dealing with the words which are not in the model. One way is to assign a random vector to them, but it is not a good solution and does not show the real similarity to the other words. It is impossible to have all the words in a corpus. For example, even though the corpus provided by Mikolov et al. contains 300-dimensional vectors for 3 million words and phrases, some words like 'Dog' are not in the corpus. Also, there are lots of new searches every day which contain new words. As a result,

the model should be updated regularly which is very difficult to do real-time because it takes time.

For each query in the corpus, we take it out from the corpus and look into the rest of the corpus to find the related task.

We compare two queries by comparing each word in a query to each word in another query and get the average similarity by using similarities returned by comparing word vectors using GoogleNews-vectors-negative300 corpus. All the similarity values will be summed up and divided by the number of combinations of word pairs. This value will be used later to find the most similar query to the query we have. The smaller the value, means more similar the queries to each other are.

Since using Word2Vec model returns a similarity value for each query comparison, we use different heuristics to choose the similar task.

It is possible to decide the task based on most similar queries returned. In different experiments, we chose this number n to be 1, 5 or 10. If the number n is greater than 1, we chose the most common task number between the returned values. After doing experiments we found that the best value is 10, and finding the common task has a significantly better performance compared to choosing the task based on just the most similar query.

Despite a good accuracy, comparisons in this method are too slow compared to MinHash LSH or Trie. This makes it impossible to use it in real-time. We chose a random sample set from each of the dataset to see how this method performs.

For testing this method we had a time limitation. For example, defining a task in D3 corpus takes 150 to 200 seconds and this corpus contains around 300,000 queries. That means we need approximately 10,000 hours for doing the experiments on just one of the datasets. Considering this limitation, we tested this method on TREC Tasks 2015 to see how accurate can it predict the task for each query. Although all the tasks were manually defined and have different subjects and vocabulary, the accuracy of this method is 0.31 which is not a good performance for such a dataset(table 4.9). Also, the time for defining a task for a single query was 1.4 seconds which is too long for a corpus with $\sim$500 queries.

As we mentioned before, there are lots of words in the datasets D1, D2 and D3 which are not in GoogleNews corpus, which causes lower accuracy in measuring the distance between two queries(table 4.8).

The detailed results are available in table 4.11.

3.5 Word Mover's Distance

This method is an optimization of the method we used in section 3.4 to calculate similarity of two queries. WMD is based on Word2Vec model. The distance between two queries will be calculated as the minimum amount of distance that the embedded words of one query need to "travel" to reach the embedded words of another query.

To compare WMD with the simple method for calculating similarity, we did the same experiment on TREC Tasks 2015 dataset. We used GoogleNews corpus for this experiment too. In the same amount of time, WMD performed significantly better with an accuracy of 0.61.

This method also has the speed issue. Each comparison takes a long time compared to MinHash LSH. Finding a task takes ∼60 seconds in D1. But WMD causes improvement in accuracy compared to the previous method which calculates the similarity of two queries by simply adding the similarity values and average them. As a result, we use WMD in the experiments and ignore the other method.

Using WMD, resulted in a accuracy more than 0.7 on datasets D1 and D2, and ∼0.56 on dataset D3. It performs better than MinHash LSH considering accuracy, but we can't use it regarding it is speed.

3.6 Elastic Search

Elastic search allows you to store, search, and analyze big volumes of data quickly and in near real time. It performs linguistic searches against documents and returns the documents that matches the search condition. Result relevancy for the given query is calculated using TF/IDF algorithm.

Like the experiments in MinHash LSH section, we take out one query and look into the rest of queries to find similar queries and define the task based on them.

There are no special parameters in this methods that we can tune. We used "match" query from "Full text queries' group. As in the Elastic search official documentation[1], it is the standard query for performing full text queries, including fuzzy matching and phrase or proximity queries.

The performance of this method is the highest between all the methods that we investigated. It has the highest accuracy and has a high indexing and search speed. It can find the task for a single query quickly and not as quickly as MinHash LSH, this time will increase slightly not exponentially with the growth of corpus.

[1] `https://www.elastic.co/guide/index.html`

It's performance beats MinHash LSH in all three datasets, specially in dataset D3. For datasets D1, D2 and D3 we achieved the accuracy of 0.7722, 0.7885 and 0.6181 consequently.

By assigning a second task to each query the accuracy improved to 0.85. Again, the dataset D3 had the highest improvement(table 4.12).

Also, experimenting on tasks with more than 50 queries or 100 queries resulted in an improvement. The accuracy improvement in dataset D3 was significant.

Moreover, choosing the most common task from returned results performed better than choosing the task based on the task of most similar query. But the difference is less significant compared to MinHash LSH.

We also analyzed the queries of each corpus for which the Elastic search could predict their related tasks correct or incorrect to see what are their differences. The detailed results of experiments are available in section 4.5.

3.7 The Idea of pseudo-documents

Each task contains one or more queries. A pseudo-document can be made by using the queries in each task and combine them to make a single document which represents the task.

In general, this will lead to faster indexing because the size of indexes is equal to the number of tasks, not the number of queries. For instance, in dataset D1 we have 276 tasks but ∼100,000 queries.

There are several questions about how to make these documents. For example, should the stop words be removed? Or if we combine the queries, should the word order be preserved? Should we extract keywords from queries and present a task as a collection of keywords?

We tested all the scenarios above and achieved a high accuracy just by using Elastic search. We used the queries as they were, without any pre-processing. The accuracy of this method is 0.78 in average on dataset D1. Indexing pseudo-documents is more efficient than indexing queries in a task separately and results in higher accuracy.

There are several problems using this approach for methods other than Elastic search. The main problem is that the length of queries and documents are extremely different. For example, a query contains in average 4-5 words in our datasets, but the average length of a document is ∼400 words in dataset D1 if we remove stop words and duplicate words from a document. Additionally, in methods like MinHash LSH and WMD, this length difference makes them inefficient.

Second, the queries are often short by default. When we remove stop words

from them, they will be shorter. This will result in less accuracy even in Elastic search. For example, using MinHash LSH has the accuracy of 0.3 when using pseudo-documents because of the length difference.

Therefore, the idea of using a pseudo-document can be helpful in this problem and can lead to very good results.

Chapter 4

Experiments

In this Chapter, we are going to write about the experiments and specifications of each method in detail. We examine 4 methods that were discussed in previous Chapters. We look for patterns in queries based on correct and false predictions to see how these methods perform on each of the datasets introduced in Chapter 3. Then we will compare their performance and describe their advantages and disadvantages and conclude which is the best to use for solving the problem of assigning a task to the new coming query.

All the experiments were done on a single computer with following specifications:

- Processor: 2.8GHz quad-core Intel Core i7, Turbo Boost up to 3.8GHz, with 6MB shared L3 cache
- Storage: 256GB PCIe-based onboard SSD
- Memory: 16GB of 2133MHz LPDDR3 onboard memory

4.1 Trie

In this method, we will make the Trie structure for each dataset. Then we check the longest prefix of the query in Trie. Since there may be more than 1 task assigned to that node, we choose the most common task between them. We can't decide on just one task(e.g. most similar task) in this structure, because the order of the values in the node is meaningless.

For the experiments Google Pygtrie[1] library is used. Building the Trie structure can be done quickly. For instance, for dataset D1 it took approximately 30 seconds to build the structure for the whole corpus. For dataset D2 it took 16.5 seconds and for dataset D3, 85 seconds.

[1] `https://github.com/google/pygtrie`

It is significantly faster than indexing in the MinHash LSH method. Moreover, searching through this data structure is fast and finding a task takes in average around 75 microseconds in D1, 35 microseconds in D2 and 58 microseconds in D3. Despite the promising speed, the accuracy of this method is quite low. It was expected because the search method in a Trie is very basic. It is like traversing a tree. If the queries don't have the same prefix there is no chance that they match. For example, if we have "how to become a doctor" and "doctor" in the tree and a new query "become a doctor" comes, the two queries in the Trie won't be taken into account, because the new query starts with the word "become". As a result, the longest prefix for this query would be a Null value. We can't expect this method to take the semantics of the queries into account. By using Trie, we achieved an accuracy of 0.14 for dataset D1, 0.0914 for D2 and 0.0332 for D3. The detailed results are available in section 4.1 table 4.1.

We use this method as the simplest possible method to see how good we can solve the problem. Also, it gives us a clue about the difficulty of a dataset. If we achieve a high accuracy in finding related task using this method, a conclusion would be that the dataset is easy. Later, in the experiments, we see that the highest accuracy obtained on dataset D1 almost in all cases and the lowest accuracy obtained in D3. So, this method can be used to measure the dataset's difficulty.

Corpus	D1	D2	D3
Accuracy	0.1403	0.0914	0.0332
Find task for a single query	75μs	44μs	69μs
Time for Making Trie	30	16.5	85
Time Adding a single query to the Trie	75μs	69μs	76μs
Number of Queries	102,171	93,391	331,389

Table 4.1: Trie results

4.2 MinHash LSH

We use datasketch[2] library for experimenting MinHash LSH method. As they described, It is important to note that the query does not give you the exact result, due to the use of MinHash and LSH. There will be false positives - sets that do not satisfy your threshold but returned, and false negatives - qualifying sets that are not returned. However, the property of LSH assures

[2] `https://ekzhu.github.io/datasketch/lsh.html`

that sets with higher Jaccard similarities always have higher probabilities to get returned than sets with lower similarities. Moreover, LSH can be optimized so that there can be a "jump" in probability right at the threshold, making the qualifying sets much more likely to get returned than the rest.

Algorithm 4.1: Assign a task to a query

inputs : MinHashLsh: A corpus contains hashed values of available queries,
query: A single query
output: A task which is the query q related to
$similar\_queries\_list = MinHashLsh.query(query)$
$taskID = find\_most\_common(similar\_queries\_list)$
return *taskID*

There are two parameters that can be set for this method: number of permutation and Jaccard similarity threshold. The Jaccard similarity threshold must be set at initialization, and cannot be changed. So does the number of permutation functions parameter. More permutation functions improve the accuracy, but also increases query cost, since more processing is required as the MinHash gets bigger.

We preprocess the queries before hashing them by doing following steps:

- remove stop words[3]
- remove duplicate words
- apply stemming on each of the words[4]
- remove queries with more than 10 words
- remove tasks with less than 50 queries

Based on multiple experiments we chose the number of permutation equal to 128 and Jaccard similarity threshold equal to 0.4. These values gave us the best performance considering a trade-off between accuracy and speed for this method.

The accuracy of this method is 0.6570 for dataset D1, 0.6586 for dataset D2 and 0.3750. The reason for low accuracy on dataset D3 is that compared to D1 and D2, lots of the queries start with "how to". Another reason is the lower average of queries in a task.

[3]Using stop words from nltk.corpus.stopwords
[4]Using Porter stemmer

Corpus	D1	D2	D3
Accuracy	0.657	0.6586	0.375
Accuracy with 2 tasks	0.7586	0.7538	0.4831
Time for making hash corpus	61s	53.8s	214s
Find task for a single query	1.3ms	0.7ms	3.7ms
Time hashing and adding a single query	∼3ms	∼3ms	∼3ms
Number of queries	102,171	93,391	331,389

Table 4.2: MinHash LSH results

Now, let's see what are the specifications of queries which their task is predicted right or wrong. As you can see in table 4.3, in dataset D1, length of queries for which MinHash LSH could predict their task correctly is bigger than the false predicted ones considering either query length or average words in a query. Also, false predicted queries have more stop words. They also have more unique words. ((Conclusion: more stop words, longer length, more new words make it harder to predict))

In dataset D2, like in dataset D1, for correct predicted queries, average query length is bigger than the false predicted ones considering either query length or average words in a query. Plus, false predicted queries have more stop words. But, despite D1, They have less unique words(table 4.4).

In dataset D3, the situation is exactly like dataset D1 (table 4.5).

If we choose the most similar query instead of getting the most common task, the accuracy decreases in all three datasets. So, choosing the most common task is a better decision. The results are available in table 4.6.

To see the impact of the number of queries in tasks, we decided to keep the tasks which have more than 50 queries and remove the rest. In another experiment, the tasks with more than 100 queries were investigated. The results in table 4.7 show that the accuracy increases when the tasks have more queries. In dataset D3 we see a significant increase in accuracy after deleting tasks which have not at least 50 queries. If we change the threshold to 100, the increase happens again. The results are available in table 4.7.

4.3 Word2Vec Model

For using GoogleNews Word2vec model, we use Gensim[Řehůřek and Sojka, 2010] library. Gensim[5] is a free python library which can be used for text analysis. The reason that we chose this library, is that its implementation is efficient and robust.

[5] `https://radimrehurek.com/gensim/`

Corpus	D1		
	All	Correct	False
Words	445,025	279,988	165,037
Unique Words	37,825	23,272	26,508
Min. Stop words in Query	0	0	0
Max. Stop words in Query	20	20	16
Avg. Stop words in Query	0.71	0.61	0.9
Std. Stop words in Query	1.14	1.03	1.31
Min. Words in Query	2	2	2
Max. Words in Query	46	46	46
Avg. Words in Query	4.36	4.17	4.71
Std. Words in Query	2.62	2.03	3.44
Min. Length of Query	4	4	5
Max. Length of Query	309	282	309
Avg. Length of Query	27.9	26.95	29.73
Std. Length of Query	16.52	12.76	21.87

Table 4.3: Dataset D1: Compare query specifications between queries with correct predicted task and queries with false predicted task, using MinHash LSH method

Corpus	D2		
	All	Correct	False
Words	332,750	224,217	108,533
Unique Words	42,325	30,155	26,784
Min. Stop words in Query	0	0	0
Max. Stop words in Query	10	9	10
Avg. Stop words in Query	0.35	0.33	0.39
Std. Stop words in Query	0.74	0.72	0.78
Min. Words in Query	1	1	1
Max. Words in Query	17	16	17
Avg. Words in Query	3.56	3.65	3.4
Std. Words in Query	1.56	1.49	1.67
Min. Length of Query	1	1	1
Max. Length of Query	91	86	91
Avg. Length of Query	22.57	23.21	21.34
Std. Length of Query	9.43	9.09	9.93

Table 4.4: Dataset D2: Compare query specifications between queries with correct predicted task and queries with false predicted task, using MinHash LSH method

Corpus	D3		
	All	Correct	False
Words	1,647,644	565,914	1,081,730
Unique Words	62,591	38,618	51,557
Min. Stop words in Query	0	0	0
Max. Stop words in Query	12	11	12
Avg. Stop words in Query	1.28	0.84	1.55
Std. Stop words in Query	1.44	1.21	1.5
Min. Words in Query	1	1	1
Max. Words in Query	19	19	19
Avg. Words in Query	4.97	4.55	5.22
Std. Words in Query	2.03	1.83	2.1
Min. Length of Query	1	1	1
Max. Length of Query	116	107	116
Avg. Length of Query	27.42	26.72	27.85
Std. Length of Query	9.71	9.24	9.95

Table 4.5: Dataset D3: Compare query specifications between queries with correct predicted task and queries with false predicted task, using MinHash LSH method

Corpus	D1	D2	D3
Accuracy	0.4875	0.4995	0.1725
Number of Queries	102,171	93,391	331,389

Table 4.6: Choose the top task returned by MinHash LSH instead of most common task.

Finding the proper task for a query using Word2vec model is highly dependant on the number of words which are in the Word2vec model. If a word is not in the model, we can't measure the similarity of the words. So, we checked how many of the words in different datasets are available in Google-News model. We got every word in the queries and checked if they are in the corpus or not. On average ∼15% of the words in the datasets are not in the in the Word2vec model. If we consider unique words, this value increases to ∼36%. Despite the high percentage of absent words, this method still obtains a high accuracy. The results are in table 4.8.

Since calculating query similarity using this model needs a longer time compared to other methods(e.g. MinHash LSH, Trie, Elastic search), we tested it on a small sample to see how it performs(table 4.9). After doing experiments, we found out that this method is not only slow but also obtains a low accuracy on a simple dataset. The results can be improved by using WMD algorithm,

Corpus	D1		
Queries in Task	All	>50	>100
Accuracy	0.6570	0.7589	0.7588
No. of Queries	102,171	102,115	101,266
Corpus	D2		
Queries in Task	All	>50	>100
Accuracy	0.6586	0.7797	0.8039
No. of Queries	93,391	66,004	45,224
Corpus	D3		
Queries in Task	All	>50	>100
Accuracy	0.3750	0.5760	0.7130
No. of Queries	331,389	184,436	47,293

Table 4.7: MinHash LSH prediction accuracy considering tasks which contains more than 50 and 100 queries.

Corpus	D1	D2	D3
Words	445,025	332,750	1,647,644
Unique Words	37,825	42,325	62,591
Total Words not in GoogleNews	79,105	45,048	256,437
Unique Words not in GoogleNews	13,491	18,125	20,458
Total Words not in GoogleNews %	17.7	13.5	15.5
Unique Words not in GoogleNews %	35.6	42.8	32.68

Table 4.8: Words in datasets, which are not in GoogleNews Corpus

which is described in next section.

4.4 Word Mover's Distance (WMD)

To compare the performance of WMD algorithm and see how it is more efficient than simply calculating the distance between queries, we did the experiment on TREC Tasks 2015 dataset. We used GoogleNews corpus for this experiment too. In the same amount of time, WMD performed significantly better with an accuracy of 0.61(table 4.9). But the speed issue is not solved, because of the same amount of comparisons. This algorithm used those similarity values wisely instead of simply adding them and average them.

This method can be used also to evaluate the quality of suggestions from

Method	Simple	WMD
Accuracy	0.36	0.61
Time defining task for a query	1.4s	1.4s

Table 4.9: Compare Word Mover's Distance(WMD) algorithm performance with simple distance calculation on TREC Tasks 2015 dataset

different search engines. We checked the suggestions for queries from different tasks from different search engines, to see how close are these suggestions to each other. We checked the Ask suggestions because of the lower accuracy of WMD method. Later, after doing experiments, we found out that Ask suggestions have a bigger share of false predictions than the suggestions from other search engines. This happens because in some of the Ask suggestions there are German words because of the location, although the query itself was in English(table 4.10).
We use WMD function in Gensim library. Here is sample code of how to calculate the distance between two sentences:

Example 4.4.1 *Calculate the distance between two sentences using WMD algorithm.*

```
model = gensim.models.KeyedVectors.
          load_word2vec_format(model_name)
#two sample sentences
s1 = 'the first sentence'
s2 = 'the second sentence'
#calculate distance between two sentences
#using WMD algorithm
distance = model.wmdistance(s1, s2)
```

the calculation method illustrated in Figure 2.5.

In average we achieved an accuracy of 0.7794 defining a task for approximately 14,000 queries. The time it takes to define a task in such dataset regarding the size, takes 15 seconds, which unacceptable not only in real-time scenarios but also in pre or post-processing.

Also, we checked the task similarity using WMD algorithm to see how similar are the queries in a task compared to themselves and to queries in other tasks. We summed up the distance between each query pairs of two tasks and

	Queries	Accuracy
TREC	547	0.8665
Bing	2131	0.8601
Google	3707	0.8581
AOL	1887	0.8542
Ask	5989	0.6712
avg. accuracy:		0.7794
total queries:		14261
avg. time assigning a task:		15s

Table 4.10: Compare suggestions for queries in TREC Tasks 2015 dataset from different search engines using WMD algorithm.

average them. As you can see in Figure 4.1 even the distance between the queries in one task is sometimes more than the distance to queries from other tasks. The only task which has near zero distance to itself is task number 6.

If we consider tasks as a collection of words in their queries and then measure their distance to each other using WMD, separating the tasks based on the distance factor in Word2Vec model is very difficult. In Figure 4.2 the distance between a task to itself is 0.

In table 4.11 the results of testing WMD algorithm on query sample from datasets D1, D2 and D3 are available. Despite its accuracy, the time it takes to look for word vectors and compute the distances between them makes this method inefficient.

Corpus	D1	D2	D3
Accuracy	0.7334	0.7216	0.5687
No. of samples	5500	2500	1000

Table 4.11: Accuracy of WMD on samples of dataset D1, D2 and D3.

4.5 Elastic Search

We used official python library of Elastic search[6] in order to do the experiments.

Like in the MinHash LSH section experiments, we take out one query and look into the rest of queries to find similar queries and define the task based on them. In Minhash LSH it is not fixed that how many similar queries will be

[6] `https://github.com/elastic/elasticsearch-py`

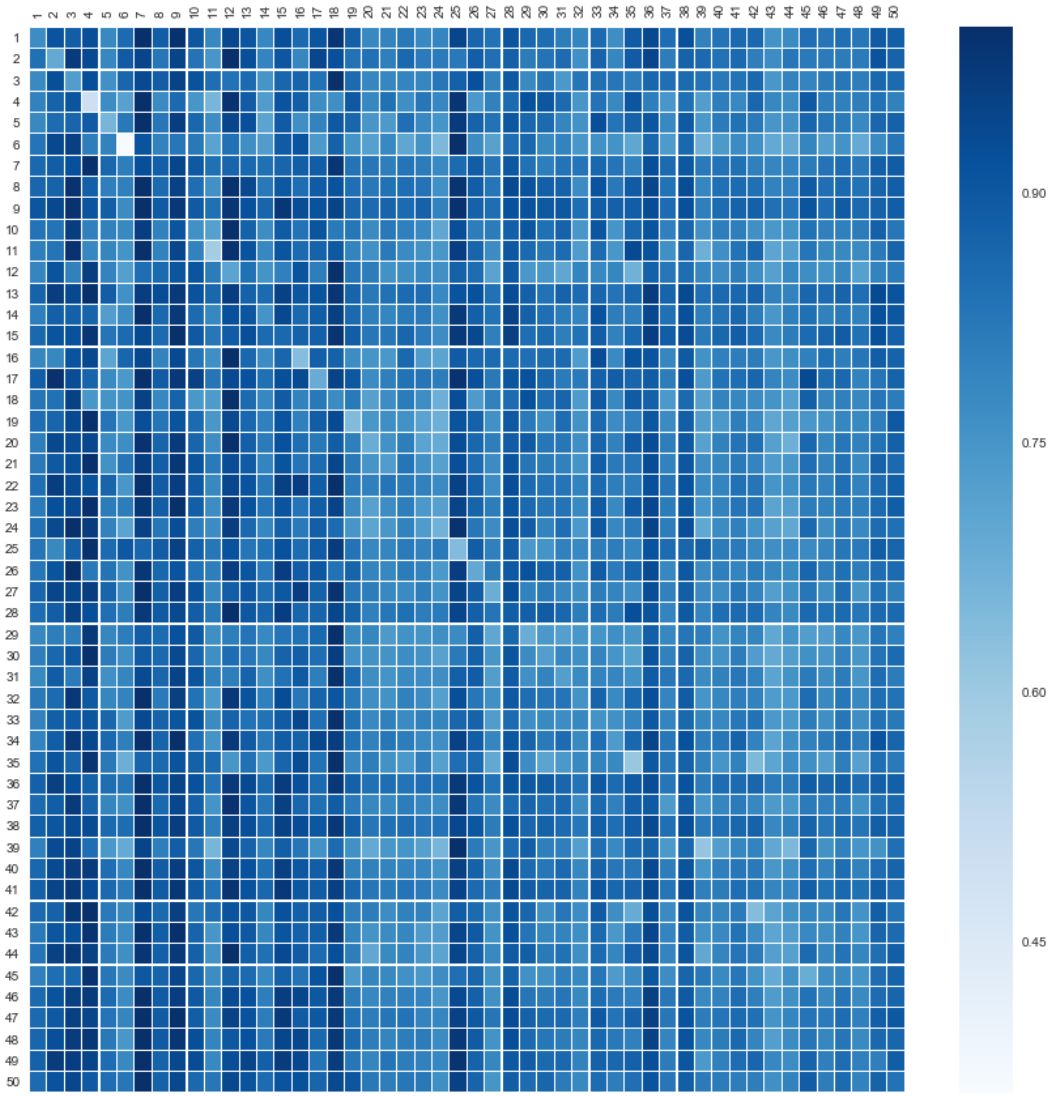

Figure 4.1: Task Similarity heat map for TREC Tasks 2015 dataset, distance between queries in a task

returned. In Elastic search submitting a query returns 10 results by default. Then, the task will be assigned based on these results.

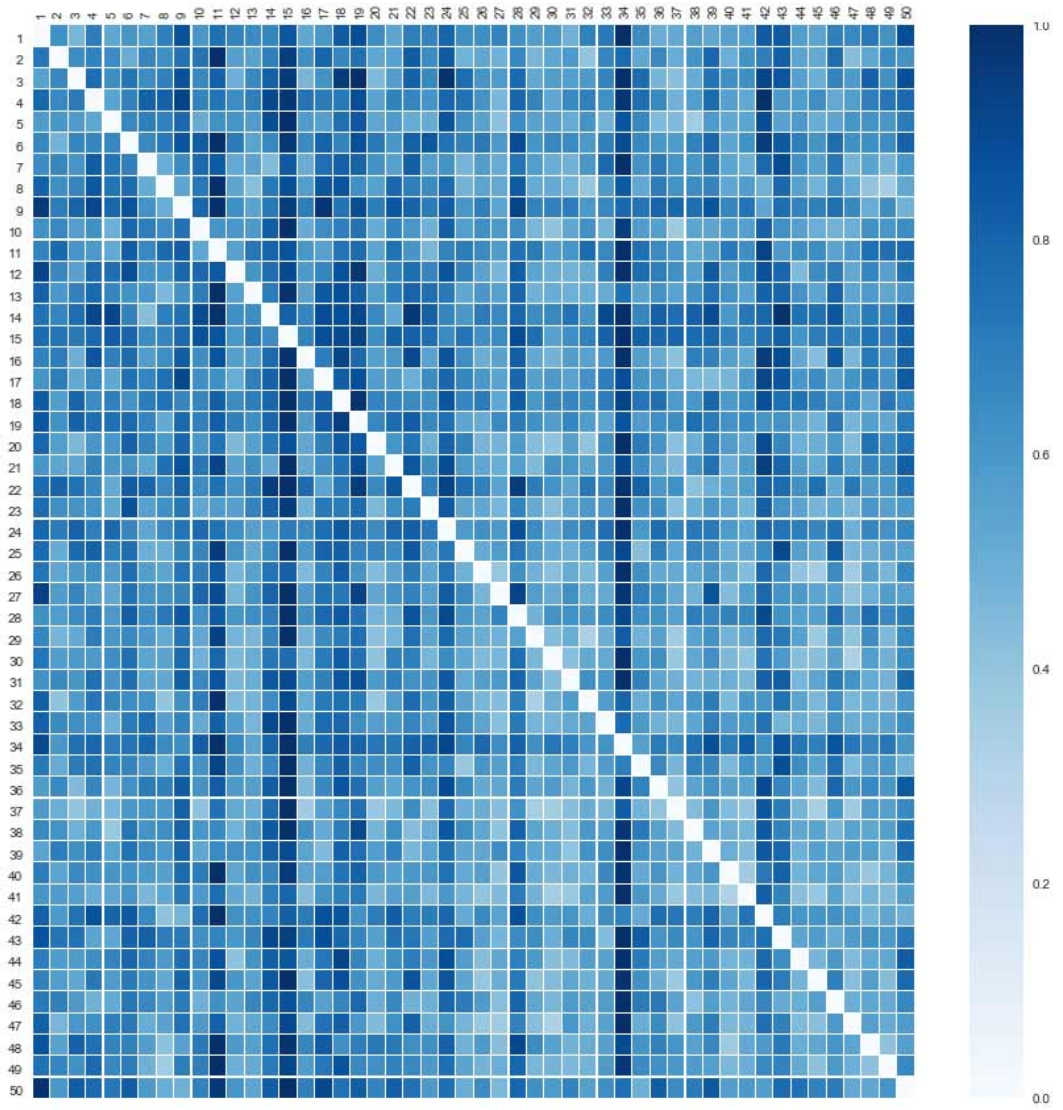

Figure 4.2: Task Similarity heat map for TREC Tasks 2015 dataset, distance between tasks as a set of words

Algorithm 4.2: Assign a task to a query, using Elastic search

inputs : q_list: list of (query, task) pairs, query: A single query
output: A task which is the query q related to
$elastic = index(q_list)$
$similar\_queries\_list = elastic.search(query)$
$taskID = find\_most\_common(similar\_queries\_list)$
return *taskID*

Because of the type our task, we used "match" query from "Full-text queries' group. It is the standard query for performing full-text queries, including fuzzy matching and phrase or proximity queries.

This method has the highest among all the methods that we investigated. The accuracy, indexing and search time of this method are very good. As you can see in table 4.12, the time for finding a task for a single query will increase slightly with the growth of corpus.

Its performance is better than MinHash LSH in all cases on different datasets. For datasets D1, D2 and D3 we achieved the accuracy of 0.7722, 0.7885 and 0.6181 consequently.

By assigning a second task to each query the accuracy improved to 0.85 in datasets D1 and D2. Again, the dataset D3 had the highest improvement(table 4.12).

Corpus	D1	D2	D3
Accuracy	0.7722	0.7885	0.6181
Accuracy with 2 Tasks	0.8576	0.8597	0.7311
Time for indexing all queries	169s	145s	543s
Find task for a single query	2.3ms	2ms	3.3ms
Time indexing and adding a single query	∼2ms	∼2ms	∼2ms
Number of Queries	102,171	93,391	331,389

Table 4.12: Elastic Search results

We also analyzed the queries of each corpus for which the Elastic search could predict their related tasks correct or incorrect to see what are their differences. In dataset D1, the queries which their task is correctly predicted, have lower average stop words, lower average words in a query and shorter length regarding the number of characters than the queries which their task is incorrectly predicted. In dataset D2, the situation is different. The difference between average stop words is trivial. But the average length of queries regarding either words or characters is higher in the queries which their task is incorrectly predicted. In dataset D3, the queries which their task is correctly predicted, have slightly lower average stop words like in D2, longer length regarding the number of characters and higher average number of words in the query than the queries which their task is incorrectly predicted. The only common factor between three datasets is lower average stop words in the queries which their task is correctly predicted compared to the queries which their task is incorrectly predicted. But the difference is not significant in each of datasets, which means that we can't find a pattern by looking at the length of

query and number of stop words in a query.

Corpus	D1		
	All	Correct	False
Words	445,025	336,498	108,527
Unique Words	37,825	28,448	20,087
Min. Stop words in Query	0	0	0
Max. Stop words in Query	20	20	15
Avg. Stop words in Query	0.71	0.67	0.82
Std. Stop words in Query	1.14	1.08	1.31
Min. Words in Query	2	2	2
Max. Words in Query	46	45	46
Avg. Words in Query	4.36	4.26	4.66
Std. Words in Query	2.62	2.27	3.54
Min. Length of Query	4	4	4
Max. Length of Query	309	290	309
Avg. Length of Query	27.9	27.36	29.76
Std. Length of Query	16.52	14.23	22.49

Table 4.13: Dataset D1: Compare query specifications between queries with correct predicted task and queries with false predicted task, using Elastic Search

If we define the task based on the task of the most similar query, the accuracy decreases (table 4.16). So, here is also choosing the most common task between returned results is a better heuristic.

Experimenting on tasks with more than 50 queries or 100 queries resulted in an improvement. The accuracy improvement in dataset D3 was significant.

Moreover, choosing the most common task from returned results performed better than choosing the task based on the task of the most similar query. But the difference is less significant compared to MinHash LSH.

4.6 Pseudo-Document

This idea works best with the Elastic search. WMD and MinHash LSH do not perform on pseudo-documents successfully. To show this, we experimented both methods on a dataset consisting of TREC Task 2015 queries and suggestion for them from Bing, Google and AOL. We didn't involve ASK suggestions because they had lots of words which were not in the GoogleNews corpus. In the first attempt, the obtained accuracies were very low. Then we decided to do preprocessing on the queries to see if the results improve. For making pseudo-documents, we did the following steps:

Corpus	D2		
	All	Correct	False
Words	332,750	270,269	62,481
Unique Words	42,325	35,454	17,918
Min. Stop words in Query	0	0	0
Max. Stop words in Query	10	10	8
Avg. Stop words in Query	0.34	0.35	0.37
Std. Stop words in Query	0.74	0.73	0.76
Min. Words in Query	1	1	1
Max. Words in Query	17	17	14
Avg. Words in Query	3.56	3.67	3.16
Std. Words in Query	1.56	1.52	1.66
Min. Length of Query	1	1	1
Max. Length of Query	91	91	86
Avg. Length of Query	22.57	23.33	19.74
Std. Length of Query	9.43	9.19	9.77

Table 4.14: Dataset D2: Compare query specifications between queries with correct predicted task and queries with false predicted task, using Elastic Search

Corpus	D3		
	All	Correct	False
Words	1,647,644	1,038,426	609,218
Unique Words	62,591	46,529	41,455
Min. Stop words in Query	0	0	0
Max. Stop words in Query	12	11	12
Avg. Stop words in Query	1.28	1.27	1.30
Std. Stop words in Query	1.44	1.43	1.46
Min. Words in Query	1	1	1
Max. Words in Query	19	19	19
Avg. Words in Query	4.97	5.07	4.81
Std. Words in Query	2.03	1.97	2.11
Min. Length of Query	1	1	1
Max. Length of Query	116	107	116
Avg. Length of Query	27.42	28.14	26.27
Std. Length of Query	9.71	9.38	10.12

Table 4.15: Dataset D3: Compare query specifications between queries with correct predicted task and queries with false predicted task, using Elastic Search

Corpus	D1	D2	D3
Accuracy	0.7518	0.7635	0.5812
Number of Queries	102,171	93,391	331,389

Table 4.16: Choose the top task returned by Elastic Search instead of most common task.

Corpus	D1		
Queries in Task	All	>50	>100
Accuracy	0.7722	0.7721	0.7721
No. of Queries	102,171	102,115	101,266
Corpus	D2		
Queries in Task	All	>50	>100
Accuracy	0.7885	0.7890	0.8147
No. of Queries	93,391	66,004	45,224
Corpus	D3		
Queries in Task	All	>50	>100
Accuracy	0.6181	0.6847	0.8082
No. of Queries	331,389	184,436	47,293

Table 4.17: Elastic Search prediction accuracy considering tasks which contains more than 50 and 100 queries.

- remove one word queries
- remove queries with character length than 3
- remove stop words from queries
- remove top 100 words which are common between multiple tasks
- then join remained words in queries in queries of each task to form a pseudo-document

WMD obtained an accuracy of 0.05 which is very low compared to its performance when not representing each task as a single document. Its accuracy on the same data is 0.85. But using pseudo-documents caused a significant improvement in the task assignment. The time decreased from 15 seconds to 7 milliseconds, however, the accuracy drops considerably. In MinHash LSH the accuracy drops from 0.63 to 0.20, with having the speed as same as before, near 1 milliseconds.

Corpus	D1	D2	D3
ES predicted correctly	78,898	73,647	204,862
ES accuracy	0.7722	0.7886	0.6182
MLSH predicted correctly	67,129	61,511	124,288
MLSH accuracy	0.6570	0.6586	0.3751
TT*	60,351	56,888	104,922
TF*	18,547	16,759	99,940
FT*	6,778	4,623	15,121
FF*	16,495	15,121	107,161
ES no related task	125	1,270	1,885
MLSH no related task	7,071	7,730	9,190

TT* : ES and MLSH both found the correct task.
TF * : ES found correct and MLSH found a wrong task.
FT* : MLSH found correct and ES found a wrong task.
FF* : ES and MLSH both found a wrong task.

Table 4.18: A comparison between Elastic Search and MinHash LSH

Despite Minhash LSH and WMD, Elastic search needs no preprocessing when dealing with pseudo-documents. Also, it's performance is significantly better than those methods. It obtained an accuracy of 0.81 by using pseudo-documents on this dataset, compared to 0.90 by not using pseudo-documents.

In table 4.19 you can see the results of experiments using Elastic search and pseudo-documents, by using train-test split on the three datasets: D1, D2 and D3. As the results show, the performance of this method is very good despite the low number of train samples(50/50). The samples were chosen randomly from each and cover all the tasks. On the same data, considering each query separately instead of making a pseudo-document for each task, the results are almost the same with the standard deviation of 0.02. But the indexing time for pseudo-document is faster.

Corpus	D1	D2	D3
Accuracy (train/test%: 80/20)	0.7856	0.7464	0.5966
Accuracy (train/test%: 50/50)	0.7585	0.7232	0.5772

Table 4.19: Define a task using Elastic search and pseudo-documents.

4.7 Overview

The results in table 4.20 shows that Elastic search has the best accuracy on three datasets compared MinHash Lsh and Trie. Trie has its lowest accuracy on dataset D3, which is the same for Elastic search and MinHash LSH.

Corpus	D1	D2	D3
Trie	0.1403	0.0914	0.0332
MinHash LSH	0.6570	0.6586	0.3750
Elastic search	0.7722	0.7885	0.6181

Table 4.20: Accuracy of Trie, MinHash LSH and Elastic search on dataset D1, D2 and D3.

To see the effects of very short and meaningless queries and the tasks which not contained enough queries, we filtered these queries and cleaned the data. We did the following steps to clean the datasets:

- remove queries that belong to more than one tasks
- remove queries with less than 3 characters
- remove single word queries
- remove queries with more than 10 words
- remove tasks with less than 50 queries

After cleaning datasets, accuracy decreased slightly, which shows that Elastic search works fine with noisy data.

Corpus	D1	D2	D3
Accuracy on noisy data	0.7722	0.7885	0.6181
Accuracy on clean data	0.7651	0.7817	0.5866
Queries: clean data	96,097	85,225	319,324
Queries: noisy data	102,171	93,391	331,389

Table 4.21: Accuracy of Elastic search on dataset D1, D2 and D3, after cleaning the data.

As we mentioned before, WMD method can't be tested on all queries of all dataset. To compare the performance of all different methods, we took a random sample from each dataset and use the rest of data as train dataset and run experiments on them. We took out 1% of queries as test and 99% as training. This results in ∼1000 test queries for D1 and D2 and ∼3000 for D3.

Elastic search outperforms all other methods and achieves a higher accuracy. Pseudo-document performs slightly better on D1 and D3.

In dataset D1 test samples, for 11% of the queries, none of the methods returned the true task. In D2 samples this number is 12.5% and in D3 sample, 15.5%. Here are some queries for which none of the methods found the correct task:

- "an Easy", "how do I": which are ambiguous queries
- "how to start a speech": it is a suggestion from AOL for question starting with "how to start". Although the assigned task is false based on the training set when we checked the dataset, WMD and Elastic search assigned tasks are more related to this query, and exactly about the speech.
- "when to replace tires": All the methods assign the same task to it. Its related task in the dataset is about replacing tires of a bike, the assigned tasks are related to replacing the tires of a car, which is also a true task.
- "RIPAIM": which is a misspelt for RIP AIM(AOL Instant Messenger).
- "lvi.ch", "OK.gov": URLs related to institutes' names
- "How to delete things from the internet": this query is related to a task with the subject of online affairs. The assigned task by methods is about using internet and different websites like Twitter and Facebook.

When all of the methods return the same task for a query but it is false, normally the assigned task to the query in the datasets is not absolutely correct. This can be used to improve the datasets' accuracies.

As shown in this Chapter, the Elastic search has the best performance among all of the methods. It is fast and accurate and works perfectly by using the idea of making a pseudo-document. Also, there is no need to pre-process the queries, because it will take care of everything out of the box. Adding and removing an element from the index is fast and easy and we can add as much as meta-data to the queries and index them. It also deals better than other methods with noisy data. The second best method is MinHash LSH which is faster than Elastic search, but considering accuracy it can't beat Elastic search. Word2Vec models are not suitable for this task because, despite their accuracy, they are extremely slow. In addition, it suffers from the problem of missing words from the corpus, and should the model be trained after a while to cover this limitation, which is also time-consuming and not possible in real-time.

Corpus	D1	D2	D3
MinHash LSH	0.6112	0.6257	0.3935
WMD	0.7220	0.6906	0.6614
Elastic search	0.7725	0.7884	0.7025
Pseudo-doc.: Elastic	0.7793	0.7804	0.7304

Table 4.22: Accuracy of WMD, MinHash LSH and Elastic search on sample of dataset D1, D2 and D3.

Chapter 5

Conclusion

In this thesis, we focus on finding the proper task for the user's submitted query as fast as possible, which has never been done before. We presume that a query log categorized by tasks is already available and focus on the problem of finding the right task for a new query. This will help the user accomplishing a search task or fulfilling an information need more satisfying. The main aspects of solving this problem are speed and accuracy.

Since there was no proper dataset available for our purpose, we created three different datasets which are publicly available. Two of them have approximately 100,000 queries and the third one contains more than 300,000 queries. These datasets have significantly more queries than available query logs.

To find the related task to a query, we experimented 4 methods: (1) Trie data structure to search through queries to find the right task, (2) a hash-based method called Minhash LSH, (3) Word2Vec models and (4) Elastic search. Also, we investigated the idea of making pseudo-document from the queries in a task and representing the task with this pseudo-document. The obtained accuracy of the Elastic search is higher than other methods. Also, it is a fast method and can define a task in $\sim$2 milliseconds in the given datasets, which makes it usable in real-time scenarios. Adding and removing an element from the index is fast and easy. It also deals better than other methods with noisy data. We used Word Mover's Distance(WMD) algorithm to calculate the distance between two queries in Word2Vec models. Word2Vec models are accurate but slow for real-time operations. MinHash LSH, a hash-based method, is faster than Elastic search and Word2vec model, but its accuracy is lower.

The analysis of the queries in different datasets shows that there is no distinguishable pattern in these queries that can help us figure out if it is a hard query to find a task for or not.

Future Work

A bigger verified query log is always better for doing the experiments. It helps the experiments to be done more similar to the real-world situation. Also, improving current datasets by manual annotation can be helpful. There are false suggestions that are not related to a query in a task which should be removed from datasets or re-assigned with a new task that they are related to. A procedure which can clean the data automatically and can recognize queries which are ambiguous or meaningless or belong to a wrong task would be a lifesaver in the process of preparing a dataset and keeping it clean. Moreover, different methods other than the ones we used can be examined. The queries are short and sometimes ambiguous. Finding a way to add extra information to a query enrich them and to be able to search better in query log helps to find the related task more accurately. Trying to improve the performance of hash-based methods regarding their speed will also be very profitable. Using supervised machine learning algorithms, Facebook's Fasttext algorithm and Convolutional Neural Networks may deliver faster and more accurate results.

This problem has not been tackled on the same scale before, and there are still many unknown areas around it.

Bibliography

Ahmed Hassan Awadallah, Ryen W. White, Patrick Pantel, Susan T. Dumais, and Yi-Min Wang. Supporting complex search tasks. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 829–838, 2014. doi: 10.1145/2661829.2661912. URL http://doi.acm.org/10.1145/2661829.2661912. 2.1

Steven M. Beitzel, Eric C. Jensen, David D. Lewis, Abdur Chowdhury, and Ophir Frieder. Automatic classification of web queries using very large unlabeled query logs. *ACM Trans. Inf. Syst.*, 25(2):9, 2007. doi: 10.1145/1229179.1229183. URL http://doi.acm.org/10.1145/1229179.1229183. 2.2

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003. URL http://www.jmlr.org/papers/v3/bengio03a.html. 2.4.5

Andrei Z. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002. doi: 10.1145/792550.792552. URL http://doi.acm.org/10.1145/792550.792552. 2.2

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 160–167, 2008. doi: 10.1145/1390156.1390177. URL http://doi.acm.org/10.1145/1390156.1390177. 2.4.5

db engines.com. Db-engines ranking of search engines. https://db-engines.com/en/ranking/search+engine. Accessed: 2018-02-18. 2.4.2

Rene De La Briandais. File searching using variable length keys. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Confer-*

ence, IRE-AIEE-ACM '59 (Western), pages 295–298, New York, NY, USA, 1959. ACM. doi: 10.1145/1457838.1457895. URL http://doi.acm.org/10.1145/1457838.1457895. 2.4.3

elastic.co. Elastic search official website. https://www.elastic.co. Accessed: 2018-02-18. 2.4.2

Forrester.com. The biggest prize in mobile commerce is influencing offline sales. https://www.forrester.com/report/The+Biggest+Prize+In+Mobile+Commerce+Is+Influencing+Offline+Sales/-/E-RES136483/. Accessed: 2018-02-18. 1

Daniel Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Inf. Sci.*, 179(12):1822–1843, 2009. doi: 10.1016/j.ins.2009.01.026. URL https://doi.org/10.1016/j.ins.2009.01.026. 1, 2.3

Google.com. Our latest quality improvements for search. https://blog.google/products/search/our-latest-quality-improvements-search/. Accessed: 2018-02-18. 1

Matthias Hagen, Jakob Gomoll, Anna Beyer, and Benno Stein. From search session detection to search mission detection. In *Open research Areas in Information Retrieval, OAIR '13, Lisbon, Portugal, May 15-17, 2013*, pages 85–92, 2013. URL http://dl.acm.org/citation.cfm?id=2491769. 2.2, 2.3

Bernard J. Jansen and Danielle L. Booth. Classifying web queries by topic and user intent. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Extended Abstracts Volume, Atlanta, Georgia, USA, April 10-15, 2010*, pages 4285–4290, 2010. doi: 10.1145/1753846.1754140. URL http://doi.acm.org/10.1145/1753846.1754140. 2.2

Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. Determining the informational, navigational, and transactional intent of web queries. *Inf. Process. Manage.*, 44(3):1251–1266, 2008. doi: 10.1016/j.ipm.2007.07.015. URL https://doi.org/10.1016/j.ipm.2007.07.015. 1

Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management,*

CIKM 2008, Napa Valley, California, USA, October 26-30, 2008, pages 699–708, 2008. doi: 10.1145/1458082.1458176. URL `http://doi.acm.org/10.1145/1458082.1458176`. 2.1

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 957–966, 2015. URL `http://jmlr.org/proceedings/papers/v37/kusnerb15.html`. 2.4.7, 2.4.7, 2.5

Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. ISBN 978-1107077232. 2.4.1

Zheng Lu, Hongyuan Zha, Xiaokang Yang, Weiyao Lin, and Zhaohui Zheng. A new algorithm for inferring user search goals with feedback sessions. *IEEE Trans. Knowl. Data Eng.*, 25(3):502–513, 2013. doi: 10.1109/TKDE.2011.248. URL `https://doi.org/10.1109/TKDE.2011.248`. 2.1

Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. Identifying task-based sessions in search engine query logs. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 277–286, 2011. doi: 10.1145/1935826.1935875. URL `http://doi.acm.org/10.1145/1935826.1935875`. 2.2, 2.3

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL `http://arxiv.org/abs/1301.3781`. 2.4.5, 2.4.6, 2.4, 3.4

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013b. URL `http://arxiv.org/abs/1310.4546`. 2.3

Rodrigo Nogueira and Kyunghyun Cho. Task-oriented query reformulation with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 574–583, 2017. URL `https://aclanthology.info/papers/D17-1061/d17-1061`. 1

Martin Potthast, Matthias Hagen, Michael Völske, and Benno Stein. Crowdsourcing interaction logs to understand text reuse from the web. In *Proceedings of the 51st Annual Meeting of the Association for Computational*

Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers, pages 1212–1221, 2013. URL http://aclweb.org/anthology/P/P13/P13-1119.pdf. 2.3

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en. 4.3

Amanda Spink, Minsoo Park, Bernard J. Jansen, and Jan O. Pedersen. Multitasking during web search sessions. *Inf. Process. Manage.*, 42(1):264–275, 2006. doi: 10.1016/j.ipm.2004.10.004. URL https://doi.org/10.1016/j.ipm.2004.10.004. 2.1

Statista.com. Global search engine market share as of august 2017, by search query size. https://www.statista.com/statistics/413229/search-query-size-search-engine-share/. Accessed: 2018-02-18. 1

Manisha Verma and Emine Yilmaz. Entity oriented task extraction from query logs. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1975–1978, 2014. doi: 10.1145/2661829.2662076. URL http://doi.acm.org/10.1145/2661829.2662076. 2.2

Manisha Verma and Emine Yilmaz. Category oriented task extraction. In *Proceedings of the 2016 ACM Conference on Human Information Interaction and Retrieval, CHIIR 2016, Carrboro, North Carolina, USA, March 13-17, 2016*, pages 333–336, 2016. doi: 10.1145/2854946.2854997. URL http://doi.acm.org/10.1145/2854946.2854997. 2.2

Zi Yang and Eric Nyberg. Leveraging procedural knowledge for task-oriented search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 513–522, 2015. doi: 10.1145/2766462.2767744. URL http://doi.acm.org/10.1145/2766462.2767744. 2.2