PUTTING NEWS ARTICLES IN CONTEXT

by

Rahul Gautam
under the guidance of
Dr. Mandar Mitra

Submitted in partial fulfillment of the requirements
for the degree of Master of Technology in Computer Science

at

Indian Statistical Institute
Kolkata, West Bengal
July 2020

Dedicated to my parents

Table of Contents

Abstract

The work of this dissertation has been done along the lines of TREC News Track *Background Linking* task. The task is, given a news article suggest other news articles that provide *context* and *background* to the current article. As we know, *context* and *background* are highly subjective terms. Here they are measured by comparing the system retrieved documents with a set of documents already marked relevant according to a panel of experts. The entire task is done on the Washington Post data set, A collection of 591537 news articles that appeared in Washington Post from 2012 to 2017.

In this dissertation we explore Six methods used to solve this task. These techniques are based on standard Information Retrieval methods and Natural Language Processing techniques. We compare them with each other and pit them against the best performing methods. We use JAVA as the main programming language for data parsing, indexing and searching. Python is also used for data exploration in some limited cases.

Keywords: verbose queries, query expansion, word embedding.

Acknowledgements

"If I have seen further, it is by standing upon the shoulders of giants"

Sir Issac Newton

First of all I would like to thank my supervisor Dr.Mandar Mitra. Without his guidance none of this would have been possible. I thank him for taking me under his wings. He provided me with absolute freedom to to explore my ideas with the right kind of guidance and insight whenever required. Every conversation and discussion with him always enriched me in some shape or form and filled me with enthusiasm to carry out work.

Another person without whom it would not be possible is Dr. Dwaipayan Roy(Doida). He constantly motivated me through thick and thins. In time of dire need, his Java code proved to be robust and beautifully documented for my purpose. A lot of diagrams and concepts used in this dissertation is borrowed from his materials. Thanks to my classmate Saurav Saha for helping me around IR Lab. Also a big thanks to Suchana Datta for helping me with various inputs throughout the task. Last but not the least I would like to thank the creators of Java programming language and Apache Lucene toolkit. Standing on the shoulders of such giants whatever work I did was possible.

Chapter 1

Introduction

In this era of a nearly ubiquitous Internet, online websites serve as the main source of news for many people. Online news encompasses a wide range of formats, from bite-sized stories for quick consumption, to big multimedia enriched articles. Taking advantage of the enormous flexibility provided by the Web as a platform for news dissemination, we can explore use cases that are unavailable to traditional print media. As an example, online news articles are typically accompanied by a list of suggested or 'related' articles that may be of interest to the reader. In this dissertation, we explore methods for identifying particular types of articles related to any given news article, viz., articles that provide additional *context* and/or *background* for the given article. This task is termed the *Background Linking* (BL) task.

Consider an article dated May 23, 2016, from the Washington Post, titled *Love in the time of climate change: Grizzlies and polar bears are now mating.*[1]. This article describes and analyzes a phenomenon where grizzlies and polar bears are mating to create a new species known as pizzlies or grolars. It explains why this is happening, and points out that it happens (or has happened) to other species as well. The following articles might be regarded as strongly related, in the background linking sense.

- *Coywolves, coyote-wolf hybrids, are prowling Rock Creek Park and D.C. suburbs* (Washington Post, July 1, 2014)
- *Humans and Neanderthals may have interbred 50,000 years earlier than previously thought* (Washington Post, February 17, 2016)

In contrast, the article *Why do seals keep trying to have sex with penguins?* (Washington Post, November 18, 2014) has only a passing mention of interbreeding, and might be regarded as substantially less important (possibly even irrelevant) as a background

[1] `https://www.washingtonpost.com/news/animalia/wp/2016/05/23/love-in-the-time-of-climate-change-grizzlies-and-polar-bears-are-now-mating/`

article.

1.1 Challenges

At first glance, the Background Linking task may appear to be just an instance of the conventional Information Retrieval (IR) task. Recall that the conventional IR task may be described as follows: given a collection of documents (such as webpages or scholarly articles), and a query provided by a user, find documents containing information that is relevant to the user's query. This task may be mapped to BL as follows.

- A given set of news articles constitutes the document collection. If the scope of BL is extended to include linking to appropriate articles from an encyclopaedia, then the document collection could include Wikipedia (or a similar encyclopaedia) as well.
- The given news article is regarded as the query.

However, some differences between BL and conventional IR make BL an interesting and challenging problem in its own right.

- **Keyword Selection.** First, in traditional IR, user-queries (e.g., *kolkata containment zones*) are usually short and contain only a few keywords, all of which are typically related to the information needed by the user. In contrast, a full news article contains many keywords. A non-trivial number of these keywords are likely to be unrelated to the core subject matter of the article. Using the set of all keywords as a search query is thus likely to reduce effectiveness / accuracy. Further, processing a long query (i.e., one containing many keywords) places an unnecessary strain on computational resources.

  For effective Background Linking therefore, we first need to address the problem of selecting a relatively small number of keywords from a given news article, so that, when this set of keywords is used as a query, the most promising articles (from the BL perspective) are likely to be retrieved.

- **High Precision Retrieval.** In response to a search query, conventional Web search engines generally provide at least a list of ten links to webpages. In recent times, this list of "ten blue links" is sometimes supplemented by information

nuggets, images, and/or video clips.

Comparatively much less space is allocated to displaying links to background / context articles alongside a given news article. As both space and user attention are limited, the requirement of being precise and accurate is more stringent for BL systems than for conventional search engines.

1.2 Outline of Dissertation

As discussed above, the main objective of BL is (i) to process a newspaper article to produce a keyword query, and (ii) to use the query to accurately retrieve a small set of related articles. In this dissertation, we study several standard term-weighting methods that assign numeric importance scores to the words in a document. The words with the highest scores are selected for inclusion in the query. We also explore techniques that have been proposed for effectively processing verbose queries. Finally, we investigate the use of *entities* to improve search accuracy.

The rest of this report is organised as follows. In Chapter 2, we briefly review some basic terminology and concepts used in Information Retrieval (including term-weighting, document scoring, and evaluation metrics). Readers familiar with basic IR may skip this chapter. Chapter 3 provides a quick overview of previous work done in this area within the framework provided by the TREC News track [16]. We highlight some of the interesting approaches taken by participants in the TREC News track to solve this problem, as well as their performances. The dataset and evaluation protocol used in our experiments are introduced in Chapter 4. In Chapter 5, we motivate and describe the various approaches for BL that we tried in the course of this dissertation. We also describe the parameter tuning process for these methods. Experimental results and performance comparisons are presented in Chapter 6. Finally, Chapter 7 summarises our conclusions, and outlines some possibilities for further investigation.

Chapter 2

Prerequisites

The work done in the course of this dissertation leverages Information Retrieval and NLP methods. So a brief discussions of basic Ideas would be necessary to understand the work done for this dissertation.

2.1 Information Retrieval

Human beings can be counted as the best performing species in the known universe in terms of acquiring knowledge and gaining skills. A modern human being in his lifetime will encounter millions of profound information about its surroundings and master thousands of skills. Passing valuable knowledge through generations is not a very easy task. Human beings developed language so that they can encode their learning in a set of symbols. The set of symbols ranges from being very complex (Hieroglyph) to very simple(binary). With the ability to encode information and a suitable medium(paper) to store it, human beings started recording information. Soon we had so much information in recorded form that Finding relevant information for need became a challenge. The task became even more daunting with the advent of digital methods of information recording and internet.

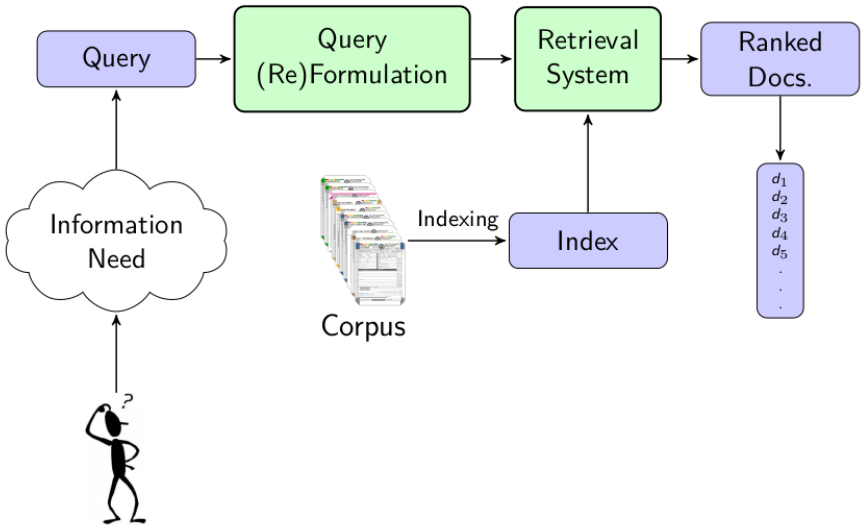
Now a days, virtually every information in the world is available to us if our computer is connected to internet. When a person is in need of information, we must have some method with which the user can query and retrieve accurate information as quickly as possible. This is where *Information Retrieval* comes in. *Information Retrieval* tries to develop systems that satisfies the information need of of an user by returning a set of relevant documents for his query. The information need may come in many forms, by ways of spoken words or a search string typed in google. The retrieved document can also be any thing from a picture or video to a chapter of a book.

So we see the meaning of information retrieval is broad. From finding the phone

number of the dean of students, Indian Statistical Institute, to reading about flamingos in the internet, Information Retrieval encompasses it all. But for our purpose we define information retrieval in the following manner.

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).[9]

Figure 2.1: Information Retrieval: A Graphical Representation



2.2 Collection, Document, and Query

When a user is having an *information need*, s/he express it in terms of a sequence of words, we call the set of words a *query*. *queries* are formal statements of information needs.

Say, we are trying to find out the year Sukumar Ray was born.To do that we go to google and type words like: "birth year of Sukumar Ray" or "DOB of Sukumar Ray" or simply "Sukumar Ray". All of these strings can be called a query for the information need. As we can see, there can be many forms of queries for a single information need. Extracting meaningful words from the query is an important part of Information Retrieval.

For every information need, we wish to retrieve a set of objects that satisfy that need. In Information Retrieval terminology, these objects are called *documents*. The

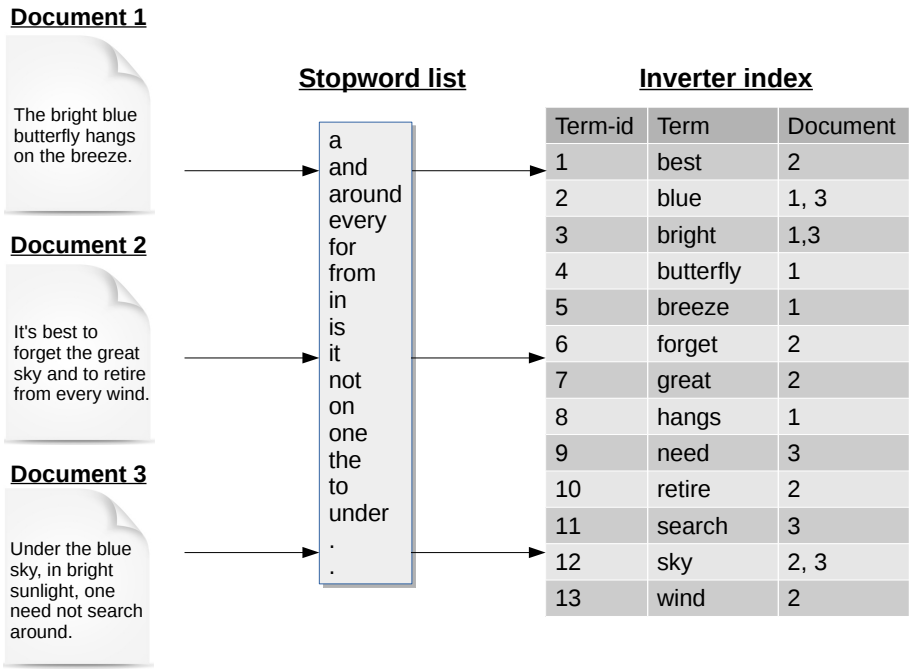


Figure 2.2: The indexing process in an IR system.

document can be anything ranging from being an actual document to a multimedia file. Corresponding to a *query*, we wish to get a list of documents where the most relevant document appears at the top and the least relevant at the end. We try to measure the relevance of a document by attaching a *score* with individual documents retrieved for a *query* The *score* is the measure of usefulness of the document corresponding to the query with respect to the IR system. The entire set from which we retrieve the documents is called *collection*.

In the context of the current project we have to construct a *query* from a given news article. The *documents* we retrieve are news articles. And the *collection* is the entire Washington post dataset.

2.3 Indexing

Given an information need, IR systems need to deliver a set of documents to the user. Storing the document as is would not be time or space efficient for searching. Indexing is the set of methods used to store the collection in such way that the following key purposes are served, They are

- Minimizing the time to perform a search.

- Making sure that the storage space is used efficiently.

While indexing we store the collection in adequate data structure. We generally remove terms that are less informative, like "the", "to","is" etc. We call such terms *stopwords*. *Stemming* is also performed to decrease the complexity of retrieval.

Stemming is the process of reducing a set of words to their root, one token that represents the entire concept for the words. For example: "waiting", "waiter","waited" words like these are reduced to their root word "wait".

After performing these measures, we store the words in *inverted list* or *posting list*. In an Inverted list, each term of the vocabulary is stored along with a list of documents containing that term. The list containing all the terms of the vocabulary is called a dictionary. For fast retrieval, the dictionary is kept in the primary memory with pointers to each inverted list. The inverted lists themselves are stored into secondary memory. This process is done for quick and efficient retrieval. Additionally the dictionary stores for each term, many important statistics that are relevant for scoring a document.

Indexing itself is a vast topic, we refer the user to information retrieval book[9] for further info.

2.4 Scoring

Given an Information need expressed in terms of a sequence of strings that we call *query*, We wish to return the user a set of ranked documents that satisfies the information need.

Thus, given a *query* we have to return a ranked list of *documents* sorted in terms of usefulness. This usefulness of a document is captured by the *score* of the *document*. Score is a function that given a query Q and a document d, returns a number indicating the usefulness of the document. We have many scoring functions in information retrieval. We started with very simple functions but they gradually became less intuitive to address the shortcomings of their ancestors.

2.4.1 Bag of Words

In Information Retrieval, we view query or document as a multiset of strings, we dont consider the order of the words. This is called the "Bag of Words" view of the

document.

2.4.2 Vector Space Model

Lets say there is a corpus C, Collection of c documents $\{d_1, d_2, d_3, ...d_c\}$. Let V be the vocabulary set(set of all possible words of the corpus)

$$V = \{v_1, v_2..v_m\}$$

We represent each document d_j as an m dimensional vector over indexed vocabulary space. Where w_i denotes the weight of ith word in the document indicating the importance of that term in the document. This is a *Bag of Words* model, implying we don't care about the location of individual terms in the document. Only the number of occurrences matter. Weight for a term is 0 if the term is absent in the document. We assign higher weights to important terms in the documents. How we determine weights are discussed later. A query Q is also defined in the same way as vector over vocabulary space.

$$\vec{d_i} = (w_1, w_2, w_3...w_m)$$
$$\vec{Q} = (w'_1, w'_2, w'_3...w'_m)$$

Where w_j is the weight of the j th word of the vocabulary in document. Lets take an example. Say, we have two documents d_1= "god made men" and d_2 = "god is good" Now according to VSM the vocabulary is

$$V = \{god, made, men, is, good\}$$

The documents here now would be represented as a 5 dimensional vector. Each component corresponding to the weight of the ith word in vocabulary. Lets assume if a word is present, the weight of the word is equal to 1. We also know if a word is absent, the weight of it would be 0. Then the documents d_1 and d_2 would be

represented as follows.

$$\vec{d_1} = (1, 1, 1, 0, 0)$$
$$\vec{d_2} = (1, 0, 0, 1, 1)$$

Having represented both the query and document as a vector in the vocabulary space, assigning score for a query to a given document is easy. We just have to take the cosine similarity between the document d and the query Q.

$$\text{Score}(\vec{Q}, \vec{d}) = \frac{\vec{Q}.\vec{d}}{|\vec{Q}||\vec{d}|}$$

Term Frequency(TF)

To assign weights to each term in the vocabulary for a document. We see how many times a term occurs in a document. If a term occurs more in a document, that means the document probably talks more about the term. This is an important metric for deciding weights for terms. Term frequency *tf* of term t in document d is the number of times a term t appears in a document d. It is written as $tf(t, d)$ or simply tf. There are many other forms of tf that are used throughout information retrieval. Some of them are,

$$1 + log(tf), 1 + log(1 + log(tf)), \frac{tf}{k + tf}$$

Inverse Document Frequency(IDF)

Term frequency is important to decide weight for a term. But there are many terms like *a, an, to, the* that occur many times in a document but is of very little importance for the document. We want "special" words that are used less frequently and generally convey more meaning. We suspect these words appear in less number of documents than common words.

How to decide if a word is "special"? We measure it with *Inverse Document Frequency or IDF*.

$$idf(t) = \log_{10} \frac{N}{df(t)}$$

Where N is the total number of documents in the corpus and $df(t)$ is the number of documents where a term t occurs. Or another function like

$$log\frac{N - df + 0.5}{df + 0.5}$$

From the formula we can see that terms, that occur in lesser number of documents get a higher IDF score.

Weight of a term

Having learned about tf and idf, we can assign weight for a term t in document d as

$$weight(t, d) = tf(t, d)idf(t)$$

There are many other methods of assigning weights to a term. There are many other forms of tf and idf but they derive from the same basic principles.

2.4.3 BM25 Scoring

BM25 is a traditional probabilistic retrieval model [13, 14] that performs very well in various IR tasks. It is also the default scoring method used in *lucene*, the library we use to retrieve documents. The following Equation mathematically presents the BM25 model to score a document d for a given query Q.

$$\text{Score}(Q, d) = \sum_{q \in Q} \log \frac{N - df(q) + 0.5}{df(q) + 0.5} \frac{tf(q, d)(k_1 + 1)}{tf(q, d) + k_1 \dot{(}1 - b + b\frac{|d|}{avgdl})} \tag{2.1}$$

In the equation, N is the number of documents in the collection, $df(q)$ is the number of documents in the collection containing the term q, $tf(q, d)$ corresponds to the count of term q in document d, and $avgdl$ is the average document length of the collection. b is a length normalization parameter, and k_1 is a positive tuning parameter that calibrates the document term frequency scaling [14].

2.5 Word Embedding

All the words are different in some way, but some have similarity with others. We say two words are similar when they roughly appear in the same context. We can say that the word "Mango" is more similar to "Apple" than say a word like "Dog". How do

we quantify the similarity between words? We have to define words in mathematical terms to do such operations.

One such idea is to represent words as vectors. We call these vectors *Word Embeddings*. Having represented words as vectors, we could explore the nice properties that vectors display. We could find words similar to a given word by exploring nearest neighbours of the vector representation of the word. We could add or subtract vectors to get interesting results. We could do something like this famous example stated below.

$$\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$$

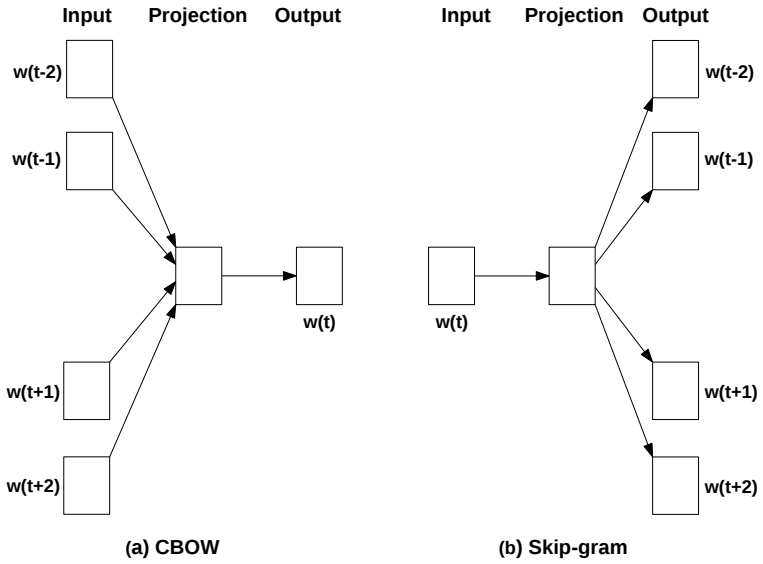
Where $\vec{king}$, $\vec{man}$, $\vec{woman}$, $\vec{queen}$ represents the vector representation of the corresponding words.

Although the idea of using vector representations for words has existed for a long time [2], the interest in word embedding has expanded a lot after the introduction of word2vec algorithm by Mikolov et al[10]. Later on ELMO[11] and BERT[4] pushed the limit for word embeddings.

In our case we use word2vec model for generating word embeddings. There are two methods to train the word embeddings, *i)* skip-gram model and *ii)* continuous bag-of-words (CBOW) model. Both of these models use a neural network with a single hidden layer. The skip-gram model *predicts the context, given a word* whereas the CBOW model *predicts the word, given its context.*

In the skip gram model, We make the training process unsupervised by feeding the algorithm a sequence of words from the corpus and fixing a window size. Words within the window are assumed to be related. We slide the window over the entire corpus. We fix a *target* word inside the window. All other words within the window are called *context* words. Given a tuple $(t, c) = target, context$ we train the network to predict the probability that c is actually the context word. for each *context* word in the window we add some other random words from the corpus. They serve as negative samples. Positive and negative samples both are used in training for a *target* word. This method is called *negative sampling.* After training is done, we can extract the word embeddings from the weights of the hidden layer.

Figure 2.3: Graphical representation of *skip-gram* and *continuous bag-of-words* models of `word2vec`



2.6 Evaluation Metric

The system we are trying to build is, given a news article suggest others that provide *context* and *background* to the report. We call such documents *useful*. Our use case is to provide some suggestions to an user who is reading the article form an app or webpage. The space is limited so we have to provide a limited set of suggestions. In this problem the number of suggestions is limited to 5. Also the suggestions should be sorted. The best suggestion should always appear at the top of the list, followed by the second best and so on.

We have to evaluate Information Retrieval systems that given a news article, suggest 5 articles. . We also have for every document, a *relevance judgement*, a number marking its degree of relevance. The *relevance judgement* is provided by a panel of experts. Scores range from 0 to 5 .Score divisions are as follows.

- 0 : The linked document provides little or no useful background information.
- 1 : The linked document provides some useful background or contextual information that would help the user understand the broader story context of the query article.
- 2 : The document provides significantly useful background

- 3 : The document provides essential useful background
- 4 :The document MUST appear in the sidebar otherwise critical context is missing.

So while judging every list of documents we have to count the following factors

- Highly relevant documents are more valuable than marginally relevant documents.
- Documents ranked low are less valuable.

Keeping all these factors in mind the ranked lists are judged with their NDCG@5 score.

2.6.1 NDCG@K Score

NDCG stands for *Normalized Discounted Cumulative Gain* let us try to explain the concept by breaking it into smaller parts. By learning *Cumulative Gain(CG)* then *Discounted Cumulative Gain(CG)* and then NDCG.

Cumulative Gain(CG):

Lets say we have a set of gain values(relevance judgements). These are the set of permitted score a document can get. For each document, we have a gain score denoting its usefulness. Lets say we also have a ranked list of documents D.

$$D = \{d_1, d_2..d_K\}$$

We construct another list G from D replacing each document with its gain score.

Let $G[i]$ denote the *Gain* Score of the ith document in the list. Then $CG@i$ is defined as

$$CG[i] = \sum_{j=1}^{i} G[i]$$

For a list of documents $CG@K$ basically denotes the sum of the scores of first K documents.

Discounted Cumulative Gain(DCG)

Discounted cumulative gain basically penalizes a document for appearing further away in the list. Discounting some part of its score from the contribution. $DCG@i$ is defined as.

$$DCG[i] = CG[i] \qquad if, i < b \tag{2.2}$$

$$DCG[i] = DCG[i-1] + \frac{G[i]}{\log_b i} \qquad if, i >= b \tag{2.3}$$

Here b is a threshold after which we penalize the score of our doument. b is usually set to 2.

NDCG@K

For a list G, its gain values can come in any order. The numbers in G might not be sorted in a descending order. Indicating this is not a perfect list. A perfect list would have the most *useful* document at the beginning followed by the $2nd$ most useful and so on. We define $NDCG@i$ as follows.

$$NDCG[i] = \frac{DCG[i]}{IdealDCG[i]} \tag{2.4}$$

An Ideal list where every document is sorted by its relevance score, would have an NDCG score of 1. A list where no document is relevant would get an NDCG score of 0. So NDCG score always ranges from 0 to 1.

Chapter 3

Previous Work

We can find the details of previous submissions in News Track overview papers of 2018[16] and 2019. We briefly discuss some of them here.

The best performing method of TREC 2018 is a method called *htwsaar4* [1]. It achieves a mean NDCG@5 score of 0.4609. This method simply constructs a verbose query by concatenating the title and content of the the article and issues a query for the index.

Ding et. al. [5] use an approach of combining BM25[14] and rocchio [15] relevance feedback method. They run an initial retrieval with BM25 method and then judge top K document as positive feedback and bottom K documents as negative feedback. They turn the document into vector with tf-Idf feature vectors. Now, according to Rocchio method, they modify the query document vector in such a way that the vector moves towards the centroid of the positive feedback documents and moves away from the negative feedback documents. This method achieves a NDCG@5 score of 0.4438 in the 2019 track.

Qu et. al. [12] Used the 2018 data for a supervised learning approach. We know every document is given a score of 0 to 4. They transform the ranking problem to a 5 class classification problem. They treat relevance grades as class labels without order. Given a query and a document, the trained classifier for each class, predicts the probability of belonging to that class. The score for each query, document pair (q,d) is calculated as:

$$score(q,d) = \sum_{r \in \{0,1,2,3,4\}} w_r . p_\theta(y = r|q,d)$$

where y is the predicted relevance grade using a multiclass classifier. w_r is a for a relevance level and get higher score for higher levels. The values of w_r is assigned like $w_r = r$. To train the classifier, 2018 queries and relevance judgement is used. Classifier for each class is trained using one vs all method.

Five features are generated to train the classifier. They are *similarity of title, similarity of content, similarity of first 100 words, similarity of mentions, similarity of category*. The first 3 similarities are measured by cosine distance of the tf-idf vectors generated by words. *similarity of mentions* is measured by Jaccard similarity of the two extracted entity sets. Each of these similarity values are within 0-1 range. The submitted run *sils_news_run2* for News track 2019 fetches an NDCG@5 score of 0.5502.

The top performing method of 2019 *UDInfolab_all*[8] uses a simple method, it extracts all the entities from the article using DBpedia Spotlight [3]. All entities are resolved to their canonical names. It issues the query containing all the words and all the extracted entities. This achiveas a mean NDCG@5 score of 0.606 on 2019 track.

Chapter 4

TREC News Track Dataset

The dataset provided by the TREC News Track consists of a single, 6.7 GB text file that contains a collection of 595,037 news articles from the Washington Post, covering the period from 2012 to 2017[1] Each article corresponds to a single JSON object[2] that is stored in one line of a text file.

- The JSON objects contain name/value pairs corresponding to the article's *title, author, date, url, source etc.*
- Every article has a unique ID.
- There were some duplicates in the dataset. Duplicate articles have the same title, content and author, although they have different IDs.
- The textual content of each article is given as an array of paragraphs from the original news.
- In many articles, the text is interspersed with other media such as images and videos, referenced by URIs. These URIs point back to the original online version of the article available from the Washington Post website.
- Each article is assigned to a *category*. Common categories include *style* (12,133 articles), *politics* (12,581 articles), and *sports* (13,478 articles). The largest category was *local*, with 39885 articles.
- The collection contains 5,557,541 distinct index terms, or keywords (please see Section 5.1 for details).

A list of 50 'query' articles was provided in 2018. For each query article, as mentioned in Chapter 1, systems are required to return a ranked list of *five* other articles as

[1]This is officially termed the *TREC Washington Post collection, Verion 2*. For 2020, Version 3 of the dataset was used. Version 3 (V3) contains 671947 articles, and includes additional articles covering the period 2017–2019. We used V3 for our official submissions to TREC 2020 as participants. The ground-truth and performance figures for this dataset are expected to be announced later this year.

[2]`https://www.json.org/json-en.html`

possible background / context articles. For 2019, a different list of 60 query articles was provided. We use the 2018 queries and ground-truth as 'training data' to tune various parameters of the proposed methods presented in the next chapter. These methods are then evaluated and compared using the 2019 queries and ground-truth as test data.

Chapter 5

Proposed Background Linking Approaches

5.1 Data Processing and Indexing

We used Java as the main programming language for this project. The Apache Lucene library[1] was used to index and query the collection.

Since the news articles correspond to their online versions, the text contains many HTML tags. We stripped the HTML tags with the help of the Jsoup library[2]. We indexed every article as a document with the following fields:

- `id`: a unique id for the article,
- `url`: link to its version,
- `date`: date of publication of the article,
- `articleType`: which category the article was published under,
- `author`
- `title`: the title of the news story,
- `content`: its textual content,
- `title-ner`: named entities (names of persons, places, organizations) contained in the title, and
- `content-ner`: named entities contained in the boody.

Not all articles contained all the fields. We did not index articles that had an empty `id` or `title` or `content`. We used Lucene's `EnglishAnalyzer` to lowercase all words during indexing. Stopwords were removed and words were stemmed using Porter's stemmer. We used the Stanford CoreNLP[3] library to identify named entities of type PERSON, PLACE or ORGANIZATION.

[1] `https://lucene.apache.org/`
[2] `https://jsoup.org/`
[3] `https://stanfordnlp.github.io/CoreNLP/`

5.2 Lucene Queries and Scoring

Before describing the approaches that we tried, we briefly review Lucene's querying and scoring mechanism. As discussed in Section 2.2, a query is simply a set of terms. By default, for a given query, Lucene computes the score for a document using the BM25 formula (Section 2.4.3). Additionally, each query term can be assigned a numerical *boost value*. The BM25 score for each matching query term is multiplied by the boost value; these scores are added up to get the total similarity score for the document, as shown in the equation below.

$$\text{Score}(Q,d) = \sum_{q \in Q} boostvalue(q) \log \frac{N - df(q) + 0.5}{df(q) + 0.5} \frac{tf(q,d)(k_1+1)}{tf(q,d) + k_1(1 - b + b\frac{|d|}{avgdl})} \tag{5.1}$$

A ranked list of the top scoring documents is retrieved by the system. For each query article, we return the top-scoring 5 articles that were published on the same date as the query article or earlier. This filtering on the basis of time reportedly results in a significant gain in performance [8].

5.3 Our Experiments

In the rest of this chapter, we discuss the different approaches we tried in order to solve the BL task. For each approach, we discuss its motivation, the method, and the results obtained for the 2018 queries. As mentioned in Chapter 4, the query set and ground-truth for 2018 is treated as training data that is used for parameter tuning. These parameter values are used when reporting results on the 2019 dataset.

5.3.1 Approach 1: Boosted Query

Motivation

We first try to establish a baseline result. We create a query from a given article, and issue it against the index. Initially, our main focus was on choosing a subset of the article's keywords as the query. We also consider the following questions.

- Is the title more important, or the content?
- Should the date of publishing be considered when creating the result list?

We start with a very basic model, and explore various tweaks to get better results.

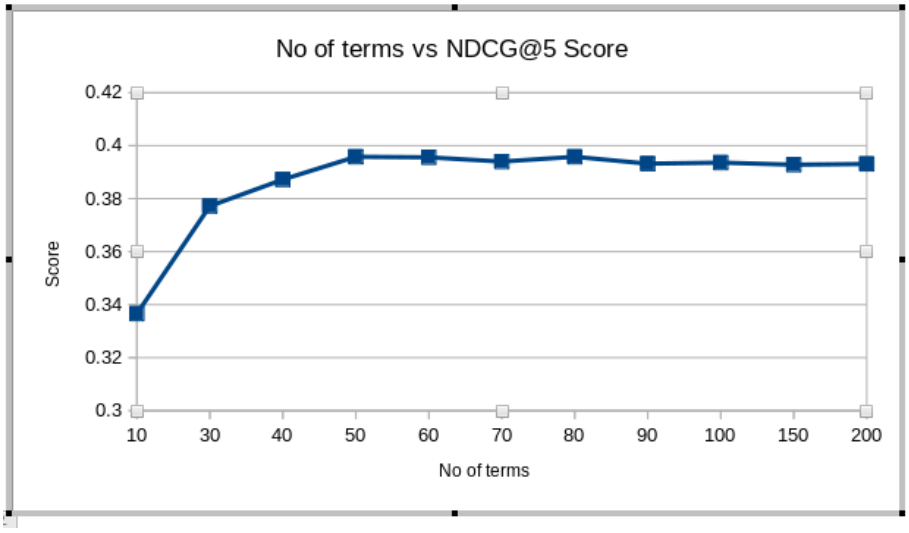
Method

Since the title of a news story summarizes its content and captures the essence of the story in a few words, we initially construct a query containing only words from the title of the query article. The next issue to consider is: while matching the query with documents, do we consider only their titles, only their contents, or both the titles and the contents? Based on preliminary experiments, we found that it is generally safest to consider both the title and the content of articles during retrieval. In subsequent experiments, therefore, queries are always matched against both the title and content of other articles, unless otherwise stated. Apart from using the title to construct the query, we also construct the query using terms from the body. We found out that the queries constructed with terms from the body offers significant performance gain. So, in this approach, we use terms from the body only for the query.

For the experiments above, every word in the query was given a boostvalue of 1. We next assigned to each query term a boostvalue equal to its *tf-idf* score. For a term t in a document field d, this score is given by $tf(t,d) \times idf(t)$, where $tf(t,d)$ is the number of occurrences of t in d, and $idf(t)$ is given by $1 + \log((1+N)/(1+df))$.

We select the top K terms on the basis of their *tf-idf* weights, and use this set as the query. As mentioned above, these weights are also used as the boostvalues in Lucene's scoring formula (Equation 5.1). The figure below shows how NDCG@5 scores vary as the value of K is changed. It is clear from above, that increasing K

Figure 5.1: Query length (K) vs. NDCG@5 score



beyond 50 offers little improvement in performance.

Results and Discussion

Table 5.3.1 presents a summary of the results obtained for the experiments described above. As discussed earlier, results for the 2019 dataset are reported using the settings for which best results are obtained for the 2018 dataset.

Year	Boostvalues used	K	NDCG@5
2018	no	10	0.3354
	yes	10	0.3367
	no	100	0.3479
	yes	100	0.3940
	yes	80	**0.3964**
	yes	70	0.3940
	yes	60	0.3956
	yes	50	0.3958
	yes	40	0.3872
2019	yes	80	**0.5263**

Table 5.1: Results for Approach 1

5.3.2 Approach 2: Using BM25 Scores as boostvalues

Motivation and Method

Since the well-known BM25 [14] term-weighting method yields much better results than the plain *tf-idf* scheme for the conventional IR task, we next try using BM25 scores in place of *tf-idf* weights as boostvalues within the setup described above. No other change is made. As above, K terms are selected from the content field.

Result and Discussion

We expected to see better performance when using BM25 weights as boostvalues instead of tf-idf weights. We were surprised to see that this method did not outperform the very simple tf-idf based weighing scheme.

year	Params	K	NDCG@5
2018	k1=1.2, b=0.5	50	0.3191
	k1=1.2, b=0.5	80	**0.3256**
	k1=1.2, b=0.5	100	0.3237
2019	k1=1.2, b=0.5	80	**0.4860**

Table 5.2: Different parameters tried for Approach 2

5.3.3 Approach 3: Document Vectorization

Motivation

We try to look at different methods for solving this problem. Like we had word embeddings, we wish to find a way to represent the news articles as vectors, not just any vectors. We want to turn documents into vectors in such a way, that two documents having similar themes are closer in the N-Dimensional vector space than two documents that talk about two different things. We call such vectors document vectors henceforth. Having represented documents as such vectors, we try to solve our background linking task in the following way:

given an article we report a ranked list of most similar document vectors to the query as our answer. We measure similarity by *cosine similarity*. The score for a document is the cosine similarity between vector representations of the document and query article.

Method

How do we turn documents into such vectors? We know about word embedding [10]. Say, from each article we choose top K words that captures the essence of the story. We suspect that if we take the sum of the word embeddings , in the sum, we would capture the essence of the story. We use the previous method to choose the top words from a document. We choose the top K words and their weights using the previous method(Approach 1).

To generate the word embedding we train a Word2Vec model on the data. Google's implementation of Word2Vec[10] has been used[4]. The CBOW model has been used for training the model and the generated vectors are chosen to have 500 dimensions.

[4] `https://code.google.com/p/word2vec/`

The window size for training was fixed as 10.

After training we get the word embeddings for every word in the vocabulary. From here we take two different approaches to generate a vector representation of the document. They are as follows

1. We sum the top K vectors to get the vector representation of a document.

2. We also tried the weighted sum of the top K vectors to generate the document vector. Weight for a word vector is the *tf-idf* weight calculated in the previous method. By experimenting on the TREC 2018 queries we find out that weighted sum of the vectors perform significantly better than their non-weighted counterpart. We tweak the no of words parameter K, We found out this method performs well for K=25 or we choose top 25 words to sum up

Results and Discussion

The results are recorded in the following table

year	weighted	K	NDCG@5
2018	no	20	0.2614
	no	25	0.2736
	yes	20	0.3178
	yes	25	**0.3270**
	yes	30	0.3183
	yes	35	0.3175
2019	yes	25	**0.4446**

Table 5.3: Different parameters tried for Approach 3

5.3.4 Approach 4 : Named Entity Recognition

Motivation

All of our approaches have been around extracting important key terms from document. Named entities, like person, place or organization tend to be more important than other words in the document. If we can identify named entities from the passages, we would be able to use the information to increase performance by incorporating them in the query.

Method

We use Stamford Stanford Named Entity Recognizer (NER)[5] to extract entities (person, place or organization) belonging to the title and content of the document. We index the entities in a two separate fields, One for entities extracted from title field and another for entities from the content field during indexing. We form a query consisting only of named entities We fixed the maximum number of entities as 20. The query consisting of named entity only could not perform very well. So we try to mix it with other methods in the following manner.

We generate two queries, one generated by method 1 with best performing parameters. Let us call this query q_1. We generate another query by weighing the named entities by their tf-idf score as in method 1. We call this query q_2.We merge two queries in the following way. let $q[i].weight$ is the weight of the ith term of query q. we change weight of the terms as

$$q_1[i].weight = \lambda * q_1[i].weight$$
$$q_2[i].weight = (1 - \lambda) * q_2[i].weight$$
$$where(0 \leq \lambda \leq 1)$$

We join the two queries to form the new query. We tweak the value of lambda to get best performance in 2018 data. We extracted named Entities from title and content field to compare their performances.

Results and Discussion

We see entity extracted from content doesn't contribute to better scores. We suspect this is due to including irrelevant terms. Whereas Entities extracted from title does increase score from method 1.

[5] `https://nlp.stanford.edu/software/CRF-NER.html`

year	Entity Extracted From	λ	NDCG@5
2018	content	0.2	0.3002
	content	0.8	0.3873
	content	0.9	0.3898
	content	1	0.3958
	title	0.5	0.3941
	title	0.6	**0.3969**
	title	0.8	0.3961
2019	title	0.6	**0.5254**

Table 5.4: Different parameters tried for Approach 4

5.3.5 Approach 5: Relevance Model

Motivation

Relevance model[6, 7] is one of the best performing models for query expansion. We attempted to put it to use in this track.

A lot of the times direct query term matching is difficult, many relevant documents may be there for not containing some exact term in the query, it might not appear in the final result set. RM3 tries to address this problem by first issuing a normal query. Then it extends the query by adding terms from the set of retrieved documents to the original query and assigning weights to the terms.

Method

In Relevance Model we have to query the documents two times, the 1st time we do it to get a set of pseudo relevant documents, 2nd time we generate a modified query by adding terms from the retrieved pseudo relevant documents. We issue the modified query to the index to produce final results.

We run into the same problem as method 1. Which field to extract query from and which field to issue query against? We use initial results on the 2018 dataset to decide that.

year	initial query words extracted from	initial query issued against	Expansion terms added from	NDCG@5
2018	title	title	title	0.1300
	title	title	content	0.1335
	title	content	content	0.1419*

Judging by the results, we choose to construct query from the title, issue the query on content of the indexed articles, and add expansion terms from content of the pseudo relevant results. For all the cases we choose top 20 documents as pseudo relevant. We try to add expansion terms such that a total of 70 query terms are present in each case. We briefly try to tune λ parameter that controls the weight of the query terms compared to terms added as part of pseudo relevant feedback. The result is discussed in the section below.

Results and Discussion

year	λ	NDCG@5
2018	0.6	0.1442
	0.8	0.1608
	0.9	**0.1641**
2019	0.6	**0.2975**

We see relatively poor performance of relevance model in comparison with other methods. We could not understand the reason for this behaviour. We suspect that when a query is short and can be expanded with pseudo relevance judgements, relevance method works well. In this case we already have a huge pool of candidate from title and content. The title and content of the query article is a better source to add expansion words from than pseudo relevant judgements.

5.3.6 Approach 6: Merging two Ranked lists

Motivation

We tried turning documents to vectors and using cosine similarity to pick top documents for BL task. Although the document vector approach did not get best results,

we suspect that it has included some documents that could not be ranked by query based ranking methods otherwise. Here we try to get better results by merging two ranked lists. We pick Method 1 and Method 3 to merge the ranked lists of them to get final rank list.

Methods

We retrieve 100 documents with the best performing parameters of both methods. Let the list of documents retrieved with method 1 called t_1 and by method 3 called t_2. We sum normalize the scores of both the tables. let the ith entry in a table t called $t[i]$ and the score for it is denoted as $t[i].score$, Next we do the following.

$$t_1[i].score = \lambda \times t_1[i].score$$
$$t_2[i].score = (1 - \lambda) \times t_2[i].score$$
$$where(0 \leq \lambda \leq 1)$$

After that we merge two tables by doing a union of them. If a document appears in both the tables, the score for the document is the sum of the score from each individual table. We vary the value of λ to get the best performance on 2018 dataset. We apply the best learned value of λ in the 2019 dataset.

Results and Discussion

We can see for parameter $\lambda = 0.6$, we get the best NDCG@5 score among all our methods for query set in 2018. We see a gain in score compared to both the methods by merging two tables.

year	λ	NDCG@5
2018	0.4	0.4022
	0.6	**0.4041**
	0.8	0.4007
2019	0.6	**0.5131**

Table 5.5: Different parameters tried for Approach 6

Chapter 6

Experimental Results

In this chapter, we compare the results of the methods explored by us. We also compare these results with the best results of all official submissions for BL task.

Here, the main metric of comparison is NDCG@5. However, Systems can retrieve up to 100 results for other levels of comparisons. NDCG metric is used because it is sensitive to the ordering of the results. News Track Authorities choose to calculate the NDCG of only the top 5 documents because, as the use case suggest, we want to provide these articles as a suggestion to the news reader. A normal reader would hardly pay attention to more than 5 suggestions.

To prepare the judgement set, corresponding to each query, a set of possible results are retrieved and they are graded by experts on a scale of 0 to 4. 0 meaning the document is not relevant at all and 4 meaning this document must appear as a recommendation. One limitation of this relevance judgement is for a query only a small subset of answers are relevance judged. There can be a case where the system retrieved document is relevant but there is no relevance judgement for it. Ideally, for every query, each document in the corpus should be graded. This is ideal but impossible to achieve, because for each query, we have to provide manual relevance judgment to almost 600000 documents. We accept such limitation because this is as close as we can get to an ideal judgement; However, While evaluating any system by its score, we have to keep this limitation in mind.

6.1 Comparison of Results

In this section we take the results from all our methods and compare them with each other.We also compare our results with the best submissions of TREC News Track 2018 and 2019.

The best submission of TREC News 2018 track was by A. Bimantara et al. code-named *htwsaar4* [1]. We can also find the best submission for 2019 track in the run

codenamed *UDInfolab_all*[8] from University of Delaware.

In table 6.1 we can see that for any given method, score for 2019 track is generally higher than score for 2018 track. We attribute this to queries being less complex. In trials, our best NDCG@5 score for 2018 track was **0.4041** and best score for 2019 track was **0.5263**. Overall best runs outperformed our runs, with NDCG@5 score of **0.4619** for 2018 and **0.606** in 2019.

Submission Name	Description	2018 Score	2019 Score
Method 1	boosted query	0.3964	**0.5263***
Method 2	BM25 boosted query	0.3256	0.4860
Method 3	Weighted sum of document vector	0.3270	0.4446
Method 4	entity extracted from title added in the boosted query	0.3969	0.5254
Method 5	RM3	0.1641	0.2975
Method 6	Merged Ranked list of method 1 and 3	**0.4041***	0.5131
htwsaar4	verbose query comprising of title and content	**0.4619****	-
UDInfolab_all	All words and Entities used as query	0.438	**0.606****

Table 6.1: Comparison between all methods
* Best performance among our methods
** Best performance among all TREC submissions

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The TREC News Track is where we explore a very modern task of suggesting newspaper articles. The way we measure the goodness of this task is by comparing the list of documents retrieved by our system with a set of documents relevance graded by experts.

In attempting this task we tried to explore a varied set of methods. We had to balance between exploring new methods and tuning parameters to extract performance gains. We found out that our simple weighted query (*Approach 1*) was working better than some more intricate methods submitted as part of the track. In *Approach 2*, we tried to add weight to the terms with BM252.1, a more advanced weighing formula than Method 1. To our surprise, it did not work as well as expected with drop in performance.

We shift from query based approaches in *Approach 3*, here we try to represent a document as a sum of the word embeddings of its most meaningful words. Although *Approach 3* did not achieve the best results, it was used later as a part of another method, producing good results. In *Approach 4*, We use the power of named entity recognition at our case. We also try to incorporate pseudo relevance feedback in BL task. In *Approach 5*, we use RM3[6] for pseudo relevance feedback. In our case, *Approach 5* did not produce good results. At the very end we try to mix of two different kind of approaches to get the best of both worlds, in *Approach 6* , we merge two ranked lists from *Approach 1* and *Approach 3*. The first one is a query based approach and the second one involves turning document into vectors and calculating their cosine similarity to get results.

In TREC News Track we discover the recurring theme of simple methods working better than their complex counterparts. Top performing submissions for this task,

namely, *htwsaar4*[1] *UDInfolab_all*[8] employ very simple method of query construction. In our case also, we noticed, the simple method of tf-idf weighted query achieves very good performance as compared with its BM25 counterpart.

None of our method could not beat the top performing methods in terms of score but some of them are simpler and a lot more effiecient than the top performing methods. For exmple, the best performing method, *UDInfolab_all*, generates a query using all the words and all the extracted entities. Such queries are computationally expensive and may not be possible in a real world scenario.

There are many possible ways to extend the work and improve performance. We leave The source code of the project available for reproducing the results and for further experimentation[1].

7.2 Future Work

Any scientific pursuit is never complete, for every method we explored, we left many other that could have been tried. Here we list such cases where further investigation can be done

- While indexing, the articles had many stray HTML tags. We could extract some information from the tags and use that intelligently to identify entities or important words. For example any text that is inside bold tag or something that is hyperlinked is bound to be an important term for the query.
- The *content* part of every article was given in separate paragraphs. While indexing, we concatenated all paragraphs into a monolithic chunk of text. We could keep the division of contents and intelligently explore some properties of them for increasing query quality.
- Parameter tuning has been one of the main issues we faced. Whenever any experiment was performed, there was always many parameters that remained unexplored. By optimally tuning all the parameters we could improve the performance of all of our methods.
- While exploring all the methods we found out that RM3 could not perform up to the mark. We can investigate the reason behind such case.

[1] `https://github.com/arrgee23/WapoDataMining`

- As no modern day problem can do without, We can employ sate of the art Neural networks based language Modelling approaches to this task.
- We have only explored single term queries. This leaves a void that can be filled by methods like Phrase query, term dependencies etc.
- We have relevance judgement results for 2018 and 2019, we can use this result for Learning to Rank approaches.
- Every news article consists of multiple fields. like *title, content, article type, date, author, etc.* We can explore the properties of the fields to get a better solution.
- We saw huge performance improvement when we recommended a news article dated on or before the query article. The performance improvement is so drastic that, All of our methods, along with submissions from other research institutes use this. We still don't know the reason for such improvement. Further research is needed.

This marks the end of my dissertation. Thank you for your precious time.

Bibliography

[1] Agra Bimantara, Michelle Blau, Kevin Engelhardt, Johannes Gerwert, Tobias Gottschalk, Philipp Lukosz, Shenna Piri, Nima Saken Shaft, and Klaus Berberich. htw saar@ trec 2018 news track. In *TREC*, 2018.

[2] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011.

[3] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 121–124, 2013.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[5] Yuyang Ding, Xiaoying Lian, Houquan Zhou, Zhaoge Liu, Hanxing Ding, and Zhongni Hou. Ictnet at trec 2019 news track. In *TREC*, 2019.

[6] Nasreen Abdul Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah S. Larkey, Xiaoyan Li, Mark D. Smucker, and Courtney Wade. Umass at TREC 2004: Novelty and HARD. In *Proc. TREC '04*, 2004.

[7] Victor Lavrenko and W. Bruce Croft. Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 120–127, New York, NY, USA, 2001. ACM.

[8] Kuang Lu and Hui Fang. Leveraging entities in background document retrieval for news articles. In *TREC*, 2019.

[9] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[11] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[12] Jiaming Qu and Yue Wang. Unc sils at trec 2019 news track. In *TREC*, 2019.

[13] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[14] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.

[15] J. J. Rocchio. Relevance feedback in information retrieval. In *The SMART retrieval system – Experiments in automatic document processing*. Prentice Hall, 1971.

[16] Ian Soboroff, Shudong Huang, and Donna Harman. Trec 2018 news track overview. In *TREC*, 2018.