COMP8130 Course Project Report: **Retrieval Engine for The University of Memphis**

Laqin Fan

December 11, 2019

1 Introduction

Information retrieval is the science of searching for documents or information in documents, which is also defined as a software program dealing with organization, storage, retrieval and evaluation of textual information from document repositories. Users can use the retrieval system to find the information they need but it doesn't return the answers directly, it informs the information existence and location of the documents that might consist of the required information. We call the documents which satisfy users' requirements relevant documents.

The main purpose of information retrieval is to develop a model for retrieving information from a database of documents. A retrieval model specifies the details of document representation, query representation and retrieval function. There are several classical retrieval models: boolean model, vector space model, and probabilistic model. In boolean model, a document is represented as a set of keywords, queries are boolean expressions of keywords, connected by AND, OR, and NOT including the use of brackets to indicate scope. The vector space model assumes t distinct terms remain after preprocessing, called index terms. These "orthogonal" terms form a vector space: $Dimension = t = |vocabulary|$. This model uses $tf - idf$ weighting as the term importance indicator. In probabilistic model, a document is typically represented by a bag of words, the retrieval is based on similarity between query and documents [1].

In this project, we implement a retrieval engine for the University of Memphis based on the vector space model, accomplish the ad hoc retrieval task with

fixed document corpus and varied queries. Meanwhile, we rank the searching results using cosine similarity, evaluate the model by computing the recall, precision and F-measure for 10 different queries.

2 Approach

In our project, we choose to implement vector space model for our search engine rather than boolean model or probabilistic model , since boolean model has some disadvantages, such as the query language is expensive as well as complicated, there is no ranking for retrieval documents, and the model similarity function is Boolean, there would be no partial matches. Whereas, the probabilistic model has no idea how to determine the importance of a term in a document, and how to determine the degree of similarity between a document and the query. However, in vector space model, the index representation of documents and the queries are considered as vectors embedded in a high dimensional Euclidean space. It uses cosine similarity to measure the importance between the document and the query.

2.1 Vector Space Model

In vector space model, each term i in a document or query j is given a real-valued weight, w_{ij}. Both documents and queries are expressed as t-dimensional vectors: $d_j = (w_{1j}, w_{2j}, ..., w_{tj})$. In Figure 1, the query and documents are represented by a two-dimensional vector space, the terms are *car*, *insurrance*. There is one query and three documents in the vector space. The top ranked document in response to the terms would be document $d2$, since the angle between q and $d2$ is the smallest. This means both *car* and *insurrance* have the highest weights in document $d2$.

2.2 Term Weighting

Term weighting is the weight on the term in vector space, also called term frequency. More frequent terms in a document are more important, i.e. more indicative of the topic, f_{ij} = frequency of term i in document j. Terms that appear in many different documents are less indicative of overall topic. df_i = document frequency of term i, the number of documents containing term i.

Inverse document frequency(idf) weighting is another way of term weighting, the important point is the term's scarcity across the collection is a measure

of its importance and importance is inversely proportional to frequency of occurrence. idf_i = inverse document frequency of term i,

$$idf_i = \log_2(N/df_i) \tag{1}$$

N is the total number of documents.

The typical combined term importance indicator is $tf - idf$ weighting,

$$w_ij = tf_{ij} * idf_i = tf_{ij} * \log_2(N/df_i) \tag{2}$$

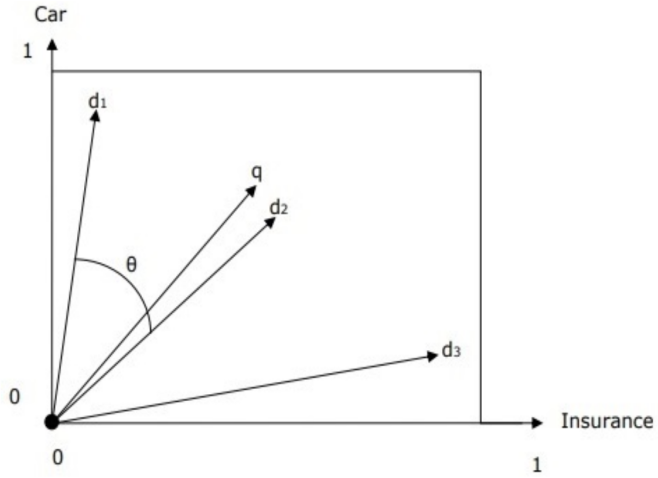

Figure 1: Graphic Representation

2.3 Cosine Similarity Measure

In this project, we rank the retrieved documents in order of presumed relevance based on cosine similarity. Cosine similarity measure is a function which computes the degree of similarity between two vectors. It is also known normalized inner product, which can be computed with the formula below:

$$cossim(d_j, d_k) = \frac{\vec{d_j} * \vec{d_k}}{|d_j| * |d_k|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}} \tag{3}$$

Which computes the cosine score between document d_j and d_k.

3 Design and Implementation

In our design, we keep using the IR system architecture in our lecture [2], see Figure 2.

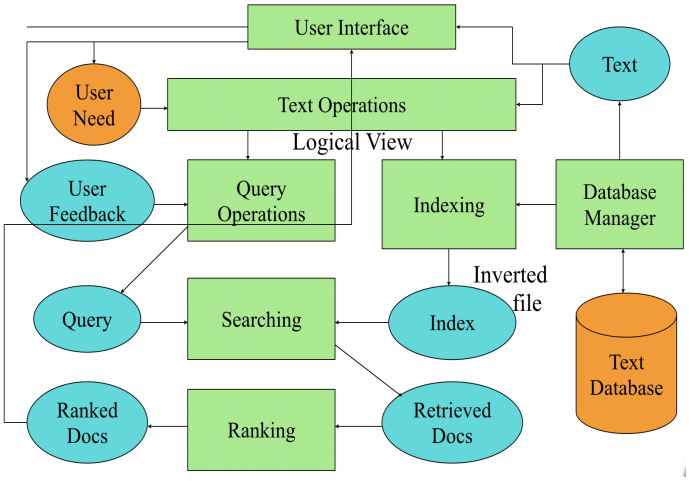

Figure 2: IR system Architecture

To accomplish this IR system and deploy it to be retrieval engine for the University of Memphis, we divide the work into several small tasks: web crawler, data preprocessing, inverted-index generation, retrieval with keyword query, and web interface. The programming language we use is Python(version 3.7.4).

3.1 Web Crawler

We implement the automatic web crawler to download 10,000 pages from the web, this is to collect database for the IR system. The pages can be used by the search engine to create the inverted index. The basic idea of how web crawler works as follows:

1. The web crawler starts with known "seed" pages, in our case, the seed is $https://www.memphis.edu/$.
2. Add them into a queue, Q
3. Extract URL from Q: fetch URL, parse it and extract the URLs on the queue, add non-visited URLs on the queue. We apply breadth-first search in the web crawler, it starts from the seed page and explore other pages following the breadth-first style.

4. Repeat until the queue Q is empty.

In our implementation, we use Beautiful Soup to get data from web pages, Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree [3]. Meanwhile, we convert the downloaded html files and pdf files to text files for later processing.

3.2 Data Preprocessing

After the file conversion, we get the text files. To construct the inverted index as the input for search engine, we have some text operations to form index words (tokens). During the preprocessing, the work we do as follows:

- Remove digits and punctuation
- Remove stop words, like "it", "they", "about", etc.
- Remove urls and other html-like strings
- Convert uppercases to lowercase
- Morphological variations

In the implementation, we utilize *nltk*, a leading platform for building Python programs to work with human language data, to process the texts. The package $word_tokenize$ helps to get the tokens and $PorterStemmer$ works to get the stemming of words, *stopwords* contains a list of stop words, which can be used to remove stop words.

3.3 Inverted-index Generation

This step is to build an inverted index, with an entry for each word in the vocabulary. The output (tokens) from the data preprocessing will be the input for indexing generation. An inverted index is an index data structure storing a mapping from content(words) to a set of documents. One entry is a word in the vocabulary, Figure 3 is an example of inverted index.

- For each entry, keep a list of documents alongside with the corresponding frequency, tf.
- For each entry, keep total number of occurrences in all documents, df.

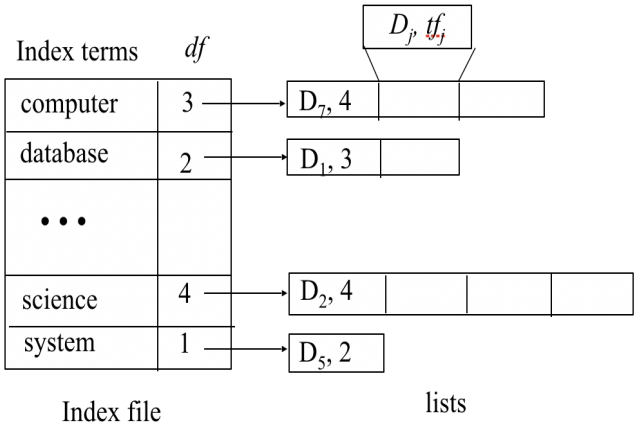

Figure 3: Inverted index example

In our implementation, we choose hashtable as the data structure for fast access, in python, we use nested dictionary to address it. After having the inverted index, we could compute the idf and weight of a token $tf * idf$.

3.4 Retrieval System with Query

In this step, the input are query Q and inverted index. The goals are to use inverted index to find the limited set of documents that contain at least one of the query words, to incrementally compute cosine similarity of each indexed document as query words are processed one by one, and to accumulate a total score for each retrieval document, having a hashtable to map the document id and cosine score.

```
Initialize an empty list R of retrieved documents
For each token, T, in query Q:
        Let I be the IDF of T, and K be the count of T in Q;
        Set the weight of T in Q: W = K * I;
        Let L be the list of document frequencies for T;
        For each entry, O, in L:
                Let D be the document of O, and C be the count of O (tf of
                    T in D);
                If D is not already in R (D was not previously retrieved)
                        Then add D to R and initialize score to 0.0;
                Increment D score by W * I * C; (product of T−weight in
                    Q and D)
```

Compute the length, L, of the vector Q (square−root of the **sum** of the squares of its weights)
For each retrieved document D in R:
    Let S be the current accumulated score of D;
    (S **is** the dot−product of D and Q)
    Let Y be the length of D;
    Normalize D final score to S/(L ∗ Y);
Sort retrieved documents in R by final score and **return** results

Listing 1: Inverted-Index retrieval algorithm

In our project, we follow the retrieval algorithm(See Listing 1) to implement the search engine. The output is a ranked list of documents in reversed order of their relevance based on cosine similarity.

3.5 Web Interface Development

Finally, we develop a web interface for users to search the information they require. The framework for the user interface is built by using Dash Plotly, which is productive Python framework for building web applications. This interface allows users to type in the keywords they need, the results are the ranked relevant document hyperlinks alongside with the corresponding cosine scores. Figure 4 shows the results while searching "computer science" through this web interface.

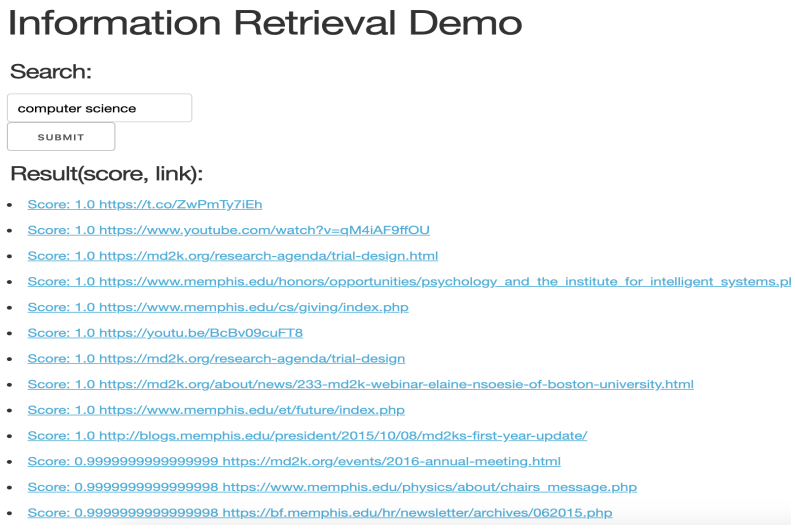

Figure 4: The search results of "computer science" in the web interface

4 Result and Evaluation

In this section, we evaluate our search engine using precision, recall and F-measure for 10 queries. In our evaluation, we rank a set of 20 documents to compute the recall, precision for our retrieval system.

$$recall = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents}} \tag{4}$$

Recall has the ability of the search to find all of the relevant items in the corpus.

$$precision = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved}} \tag{5}$$

Precision has the ability to retrieve top-ranked documents that are mostly relevant.

$$F - measure = 2 * \frac{\text{precisionr*recall}}{\text{precision + recall}} \tag{6}$$

4.1 Results

We test our search engine using 10 queries: "Computer Science", "President of the university", "Information Retrieval" , "Learner Data Institute", "International student office", "Graduate school admissions", "What is the mascto of the university", "Who is Dr. Lan Wang?" , "Dunn Hall" and "to be or not to be".

For a given query, we produce a ranked list of retrievals. The retrieval results are listed in the tables below, where we mark the relevant documents using "y". We compute the precision using the formula 5, and compute recall using formula 4, then get the average recall for each query, finally use formula 6 to get F-measure value. Table 1 shows the evaluation results for the queries. The query "to be or not to be" has no retrieval result from our search engine, because all the documents in our database are preprocessed to remove all stop words.

Table 1: Evaluation Results for all queries

Query	**Precision**	**Recall**	**F-Measure**
Computer Science	0.35	0.57	0.43
President of the university	0.5	0.55	0.57
Information Retrieval	0.4	0.56	0.46
Learner Data Institute	0.25	0.6	0.35
International student office	0.1	0.75	0.18
Graduate school admissions	0.1	0.75	0.18
What is the mascto of the university	0.1	0.75	0.18
Who is Dr. Lan Wang?	0.57	0.62	0.58
Dunn Hall	0.1	0.75	0.18
to be or not to be	0	0	0

Query1: Computer Science

Num	**relevant**
1	
2	
3	
4	y
5	
6	
7	y
8	
9	y
10	
11	
12	
13	y
14	
15	
16	
17	y
18	y
19	
20	y

Q2: President of the university

Num	**relevant**
1	y
2	
3	y
4	
5	
6	
7	y
8	
9	y
10	y
11	
12	
13	y
14	
15	y
16	
17	y
18	y
19	
20	y

Q3: Information Retrieval

Num	**relevant**
1	
2	y
3	y
4	y
5	y
6	y
7	y
8	
9	
10	
11	
12	
13	
14	
15	
16	y
17	
18	
19	y
20	

Q4: Learner Data Institute

Num	relevant
1	
2	
3	
4	
5	
6	
7	y
8	y
9	y
10	
11	
12	
13	
14	y
15	y
16	
17	
18	
19	
20	

Q5: International student office

Num	relevant
1	
2	
3	
4	
5	
6	
7	y
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	y
19	
20	

Q6: Graduate school admissions

Num	relevant
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	y
11	
12	
13	
14	
15	
16	
17	y
18	
19	
20	

Q7: What is the mascto of the university

Num	relevant
1	
2	
3	
4	y
5	
6	
7	
8	
9	
10	
11	
12	y
13	
14	
15	
16	
17	
18	
19	
20	

Q8: Who is Dr. Lan Wang?

Num	relevant
1	y
2	
3	y
4	
5	y
6	y
7	

Q9: dunn hall

Num	relevant
1	
2	
3	
4	
5	y
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	y
20	

5 Conclusion and Future Work

In this project, we develop the search engine for the University of Memphis using vector space model, rank the retrieval lists of documents based on cosine similarity, and evaluate the query results by computing precision, recall and F-measure. However, our search engine depends on lexical matching, it misses relevant information, and retrieve irrelevant information.

For the future work, first, we will enlarge the dataset, currently we only have 10000 documents stored; Second, we will improve the search algorithm to support phrase search, (e.g. "Information Retrieval"), by using positional inverted index. Furthermore, we could deploy LSA(Latent Semantic Analysis) model in the search engine, which finds the latent semantic space underlies the documents. This will be helpful and efficient to find the relevant documents.

References

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Modern Information Retrieval. Second Edition 1999. (Vol. 463): ACM press New York.

[2] COMP8130 Information Retrieval slides. *indexing and searching.*

[3] Beautiful Soup 4.4.0. *https://www.crummy.com/software/BeautifulSoup/bs4/doc/*