Information Retrieval and Web Search

Assignment 1

Rajat Singh (csz208507)

Objective: In this assignment we tried build an end-to-end retrieval system for English and perform a toy-scale performance evaluation of Vector-space retrieval model.

The program has the following components:

1. **Inverted indexing formation of the collection**
2. **Extended Vector-space Retrieval**
3. **Prefix search**
4. **Restriction to named entities**

Steps to run the program

Step 1: **Inverted indexing formation of the collection**

Step 1 of the assignment focus on construction of inverted index from the collection of documents. I tried to achieve this functionality by using list of lists of dictionaries and dictionary of dictionary data structure.

a. First, we parse each document and find the special tag words and their frequency and store it with combination of special tag and word (P:<word> O:<word> or L:<word>).
b. Now the document is free from special words so we will parse the document, remove the stop words, and then calculate the term frequency of the remaining word in the document and store in a dictionary. Term Frequency also known as TF measures the number of times a term (word) occurs in a document.
c. All terms are considered equally important in the above steps but certain terms that occur too often have little power in deciding the relevance. So, we need an algorithm to **weigh down** the effects of too often occurring terms. Also, the terms that occur less in the document can be more relevant. We need a way to **weigh up** the effects of less often occurring terms. We will achieve this by computing Inverse Document Frequency (IDF). I have used a dictionary to store IDF for each term.

$$IDF_i = 1 + \log_e(\text{Total Number of Documents} / \text{Number of Documents with term i in it})$$

d. After calculating IDF, I have calculated weight of the term in the document (TF-IDF) and then normalized the value and store in form of dictionary.
e. After calculating the weight of each term, we will build inverted index and store in the **<file name>.idx** file.

Command to build inverted index:

$ python3 invidx_cons.py <Location of documents> <index file name>

Step 2: **Implement a helper tool for printing the dictionary**

Step 2 of the assignment focus on the printing of the stored details of the vocabulary and the inverted index in the given format to check whether the files are properly stored or not:

<indexterm>:<df>:<offset-to-its-postingslist-in-idx-file>

a. First read the data from the document (vocabulary and the inverted index).
b. For each term count the number of entries in the posting list with respect to that term which will be the document frequency for that term.
c. Print term and its document frequency and the offset to its postingslist in idx file.

Command to print the dictionary:

$ python3 printdict.py <dict file name>

Step 3: **Search and Rank the documents according to the query**

Step 3 of the assignment primarily focus on listing the relevant document from the corpus with respect to the query provided to the function.

a. First step is to read and load the stored pre-computed dictionary file and the inverted index file from the memory to RAM. From this data only we are going to rank the documents.
b. After loading the pre-computed file, we will load the query file and parse the query file to get the relevant query-id and query and store in the list.
c. (**Handling the special words**) If a query has special words then we are handling it separately.

    O:term

    O:term*

    Term*

    Or combination of any of the above given terms

    If any of the format of the term appears in the query, then first we process those words and store in the list of the query and rank the documents accordingly.

d. If query have a term many times, then we have increased the weight of that term logarithmetically and then normalize the terms.
e. After normalizing we will calculate the score of the document by multiplying the normalized weight of the query term to the normalized weight of the he documents and then store the details in the list and sort them in descending order.
f. Document whose score is greater is the most relevant document for the given query.

Formula used:

Weight of word = IDF of that word * (1 + $\log_e$ (term frequency in the query))

Score of the document = sum of (weight of the word * normalized weight of the document for that term)

Command to perform query:

$ python3 vecsearch.py --query <query file address> --cutoff <cutoff value> --output <output file name> --index <index file name> --dict <dictionary file name>