Documentation

Abhinav, 2018TT10867

Implementation of Vector Space Model of Information Retrieval is in the version of Python 3.8.5

Libraries used :

- External : NLTK
- In-built : os, sys, math, bisect(for performing binary search)

All the reading and writing to files is done line by line, using inbuilt open() and readline() functions.

Inverted Index :

The construction of Inverted Index is done by invidx.py python file.

Inverted Index is implemented by using a dictionary data structure whose key is an index word(word in vocabulary) and value is corresponding postings list. In the postings list, first element of value array of index word key is document frequency of that word and all the other elements are lists in which first element is document id and second element is frequency of index word in that particular document.

Dictionary structure : {index_word : [doc_freq, [docid_1,tf_1], [docid_2,tf_2]…]}

For text in every document of the given document collection, words except stop-words and numbers are extracted as an array of strings.

All the **untagged** words have been stored in lower case.

All the **tagged** words have been stored in vocabulary in two ways:

- Simply stored in lower case
- The word has been transformed into a new word by taking account of the tag associated with it and lower casing it afterwards.
  For eg :
  - If the tag is <PERSON> it is stored as p:word
  - If the tag is <LOCATION> it is stored as l:word
  - If the tag is <ORGANIZATION> it is stored as o:word

Then all the keys in the dictionary structure have been sorted in lexicographical order. Then the vocabulary was written into a file named indexfile.dict and postings list was written into a file named indexfile.idx .

Query Search:

Format of query used is same as the topics file that was given to us.
After reading all the queries from the queryfile, all these queries were stored in form of a dictionary data structure in which every key was query-id and for each key the value was a list of words in that query (all words have been converted to lower case).
Query dictionary : {query-id : [words in query]}

The inverted index structure was retrieved from the generated file (indexfile.idx) in the form of dictionary data structure and another dictionary was created to store the magnitude of all the document vectors.
Magnitude of vectors dictionary : {doc-id : magnitude of the vector}

Now for every word in a particular query, the possible situations were:

- If the word was a **simple** query word(ie. no prefix and no named entity restriction) then searched for that query word in the inverted index structure and for all the documents in that posting list store them in a dictionary named document_vectors which essentially stored the doc-id and corresponding vector(ie. List of tf*idf score for the word) for that doc-id.
  Document_vectors : {doc-id : [(query_word, tf*idf score)…]}

- If the word had a **named entity restriction**:
  - If word started with "P:" or "L:" or "O:" then simply searched it as a simple word because I had already stored that in the same format in my vocabulary.
  - If word started with "N:" then transformed that word into three new words which started with "P:", "L:" and "O:".
    For e.g. word=**"N:abhi"** then the new words are **"p:abhi"**, **"l:abhi"** and **"o:abhi"**

- If the word required **prefix search**:
  I had already maintained a sorted list of words in the vocabulary and on performing binary search on that list, I found all the index words which had prefix as the query word and then made the document vector of all the doc-ids in the respective postings lists of those index words.

Now after doing this for all the query words for a particular query I had some document vectors and a query vector. Then I found the value of cosine between all those document vectors with the query vector and sorted them in the decreasing order of their cosine score. By doing same for rest of the queries I found top k documents for every query and wrote them in **qrels** format into the **resultfile** and ran **trec_eval** tool on it.

Scores:

F1 @100 : 0.1453

NDCG : 0.2154