Project Phase 4

Professor: Dr. Claudia Pearce

Name : Chelsea Morais

I. Abstract

The previous phases involved calculating the tf*idf scores, preprocessing the data, and creating a posting file. I have now used all the previous information to calculate the cosine similarity scores for the query and displayed a list of high-scoring documents.

II. Input

With weights -> python3 phase4.py input output W

retrieve edit 0.3 dog 0.0

Without weights -> python3 phase4.py input output

retrieve edit dog

III. Queries:

1. Preprocessing the query terms: The code has two options either we give the query terms with their weights or we input only the query terms. If there are only query terms then we pre-process the query by removing the stopwords that are present in the stopwords.txt file as well as all numeric and special characters otherwise we extract we create a dictionary containing the key: value pair as follows {1:{edit:0.3,dog:0.0}}
2. We calculate the tf-idf scores for the query terms as we did in earlier phases.

The formula for tf calculations is :

Formula : tf(w,d) = count(w,d)/|D|

Here count(w,d) is the frequency of a word in a document and D is the total number of terms in that document.

The formula for idf calculations is :

Formula : idf(w) = log(|N|/wf(w))

Here N is the total number of documents which is 503 and wf(w) in the total number of times a word w occurs in the document. The log component is used to diminish the size for large values of C which in this case is 503.

Cosine similarity scores :

This code calculates the cosine similarity scores between queries and documents based on an inverted index and the calculated tf-idf scores for the query. It uses the formula for cosine similarity to calculate the relevance of each document to each query term, and then aggregates the scores across terms to produce an overall score for each document. Finally, it sorts the documents by their overall scores and stores the results in a dictionary for displaying the most relevant documents. Once all terms in the query have been processed, the doc_sim dictionary for the current query is sorted in descending order by the cosine similarity scores.

- The code begins by iterating over a dictionary containing the preprocessed query terms. A dictionary named doc_s is created to store the cosine similarity scores between the query and each document that contains at least one term from the query.
- For each term in the query, the code checks if the term exists in an inverted index named inverted_index. If the term exists in the index, the code

retrieves the list of documents that contain the term and iterates over each document.

- For each document, the code retrieves a tf*idf score that represents the relevance of the document to the current term. It then retrieves the length of the document vector named d_len and the score of the term in the query named query_s and the length of the query vector named q_len. Using these values, the code calculates the cosine similarity between the query and the document as (doc_s * query_s) / (d_len * q_len).
- The resulting cosine similarity score is added to the corresponding entry in the dictionary for the current document. If the document is not already in the dictionary, a new entry is created with the current cosine similarity score, and finally, the scores are sorted in descending order.

III. Sample query outputs:

1. Input query: international affairs

```
(base) chelseamorais@Laptop-2 ir % python3 Phase3.py input output
retrieve international affairs
3
Top scoring documents
(219, 0.13041134443006652)
(247, 0.07795793275864067)
(161, 0.07350143419974488)
(133, 0.0628916427946884)
(243, 0.05981886549485449)
(138, 0.05474742173298901)
(125, 0.04979904038327494)
(331, 0.04527847997368063)
(232, 0.04481325206823683)
(197, 0.04385560705796113)
(base) chelseamorais@Laptop-2 ir %
```

2. Input query: computer network

```
(base) chelseamorais@Laptop-2 ir % python3 Phase3.py input output
retrieve computer network
Top scoring documents
(156, 0.18752374848549994)
(60, 0.15492522018232815)
(181, 0.10311800577767953)
(380, 0.09671578341541799)
(501, 0.08674555674636483)
(223, 0.08389517076821741)
(502, 0.0723629283920459)
(135, 0.07013592266258017)
(64, 0.05512215357975224)
(315, 0.05507808552420902)
(base) chelseamorais@Laptop-2 ir %
```

3. Input query : identity theft

```
(base) chelseamorais@Laptop-2 ir % python3 Phase3.py input output
retrieve identity theft
Top scoring documents
(380, 0.0695638463747808)
(379, 0.06744503732078612)
(301, 0.040002514661254894)
(245, 0.036248585142802985)
(298, 0.02093792517527538)
(328, 0.020146410877433872)
(292, 0.01441242072669754)
(235, 0.008962258833797596)
(304, 0.00788529705284148)
(307, 0.007555173833524936)
```

4. Input query : diet

```
(base) chelseamorais@Laptop-2 ir % python3 Phase3.py input output
retrieve diet
Top scoring documents
(18, 0.26106144519109337)
(252, 0.050572440683768934)
(263, 0.04668059177174051)
(9, 0.04225430423753215)
(50, 0.016314114857775824)
(152, 0.01600093259445659)
(353, 0.014048856111200611)
```