Assignment 1 Part B

This is an individual assignment. If you get help from others you must write their names down on your submission and explain how they helped you. If you use external resources you must mention them explicitly. You may use third party libraries but you need to cite them, too.

Date posted: March 7th, 2016

Date Due: March 22th, 2016

Goal: Indexing, Retrieval Models and Text Transformation

Task1: Building your own Index

A. Read in Data collection

Read in the document collection provided in the file ***tccorpus.txt*** inside *tccorpus.zip*, which is an early standard collection of abstracts from the *Communications of the ACM*.

The provided tccorpus.txt file is in the format:

- A # followed by a document ID.
- Lines below the document ID line contain **stemmed words** from the document.
- For example:
  - `# 1`
  - `this is a tokenized line for document 1`
  - `this is also a line of document 1`
  - `# 2`
  - `from here lines for document 2 begin`
  - `...`
  - `...`
  - `# 3`
  - `...`

- For tokenization, simply break the character sequence at any run of whitespace (space, newline, etc.).

- Also, because this each document ends with a list of numerical cross-references, you can, for this assignment, ignore any tokens that contain only the digits 0-9.

B. Build an Inverted Index

The following data structures are required when building the inverted Index.

- Term frequency (*tf*) are stored in the inverted lists for each index term: word -> (docid1, tf), (docid2, tf), …, (docidN, tf). Please make sure that words are ranked in alphabetically ascending order, and docids are ranked in numerically ascending order.
- For this particular assignment, you do not need to consider the term position information.
- Store the vocabulary of the entire data collection as well as the number of tokens per document in a separate data structure.
- You may employ any concrete data structures convenient for the programming language you are using, as long as you can write them to disk and read them back in when you want to run some queries.

Build the inverted index as described above. You should invoke your Indexer as:

indexer tccorpus.txt indexTC.out

Task 2: Building Retrieval Models

A. Vector Space Model; TF-IDF

This is the first vector space model using TF-IDF. The scoring function can be represented as follows:

$$tfidf(d,q) = \sum_{w \in q} okapi\_tf(w,d) \cdot \log \frac{D}{df_w}$$

$$okapi\_tf(w,d) = \frac{tf_{w,d}}{tf_{w,d} + 0.5 + 1.5 \cdot (len(d) / avg(len(d)))}$$

where:

- D is the total number of documents in the corpus.
- df_w is the number of documents containing term w.
- $tf_{w,d}$ is the term frequency of term w in document d.
- len(d) is the length of document d.
- avg(len(d)) is the average document length for the entire corpus.

Implement the above ranking algorithm, and write a program to provide a ranked list of for queries. Please invoke your ranker as:

vsm1 indexTC.out queries.txt 100 > resultsVSM1.eval

"100" means that you want to output the top 100 docs. Please indicate the docid as well as the computed ranking scores for each returned document.

B. Vector Space Model: TF-IDF and Cosine Similarity

In this second vector space model, please represent each document and each query as a *tf.idf* vector. For computational efficiency concern, you do not need to record the weight of the terms not appearing in the document. You can compute *tf* by following the *okapi_tf* as mentioned above, and *idf* by using $\log(\frac{D}{df_w})$. The cosine similarity between document and query vector can be calculated as:

$$\text{cosine}(d,q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2 \sum_{j=1}^{t} q_j^2}}$$

where, t is the vocabulary size in the corpus.

Implement the above ranking algorithm, and write a program to provide a ranked list of for queries. Please invoke your ranker as:

vsm2 indexTC.out queries.txt 100 > resultsVSM2.eval

C. Okapi_BM25

BM25 is a probability model based on binary independence model. Its scoring function can be computed as:

$$bm25(d,q) = \sum_{w \in q} [\log(\frac{D+0.5}{df_w+0.5}) \cdot \frac{tf_{w,d} + k_1 \cdot tf_{w,d}}{tf_{w,d} + k_1((1-b) + b \cdot \frac{len(d)}{avg(len(d))})} \cdot \frac{tf_{w,q} + k_2 \cdot tf_{w,q}}{tf_{w,q} + k_2}]$$

where:

- $tf_{w,q}$ is the term-frequency of term w in query q.
- k1, k2, and b are constants. You can set them as k1=1.2, b=0.75, and k2=100.

Implement the above ranking algorithm, and write a program to provide a ranked list of for queries. Please invoke your ranker as:

bm25 indexTC.out queries.txt 100 > resultsBM25.eval

D. Test Queries.

Use the following stemmed test queries, also provided in *queries.txt*

Query ID	Query Text
1	portabl oper system
2	code optim for space effici
3	parallel algorithm
4	distribut comput structur and algorithm

Task 3. Comparison and Evaluation

Since we have no knowledge on the relevant documents, we still cannot conduct effective comparison between different models with regards to their ranking performance. Therefore, please re-apply the Kendall Tau coefficient metric over three ranking lists: resultsVSM1.eval, resultsVSM2.eval and resultsBM25.eval, and report three Kendall tau values.

Extra Credits [+20]: Text transformation

Working on the Wiki page full content you have crawled and downloaded, you can implement your own tokenizing, removing stop-words and stemming.

EC.Task1: Tokenizing

The first step of indexing is tokenizing documents from the collection. That is, given a raw document you need to produce a sequence of *tokens*. For the purposes of this assignment, a token is a contiguous sequence of characters which matches the regular expression \w+(\.?\w+)* – that is, any number of letters and numbers, possibly separated by single periods in the middle. For instance, bob and 376 and 98.6 and 192.160.0.1 are all tokens. 123,456 and aunt's are not tokens (each of these examples is two tokens). All alphabetic characters should be converted to lowercase during tokenization, so bob and Bob and BOB are all tokenized into bob.

EC.Task2: Stopping and Stemming

Please use the stoplist.txt of stop words (obtained from NLTK (Natural Language Toolkit) http://www.nltk.org) to remove stop words.

You may use any standard stemming library to conduct stemming. For instance, the python stemming package and/or the Java Weka package containing stemmer implementations.

EC.Task3: Indexing

Once you have finished tokenization, removing stop words and stemming, build an Index for these Wiki pages in the similar way as you built the Index for tccorpus.txt. However, to facilitate checking your results (since you might have different mapping schemes), please build you index as follows:

word -> (doc1Name, tf1), (doc2Name, tf2), ……, (docNname, tfN)

Please sort both the word and docName in alphabetically ascending order. The docName for web page with URL https://en.wikipedia.org/wiki/Renewable_energy is Renewable_energy.

What to hand in:

1) The Index you built for tccorpus.txt.
2) Your codes for generating your index.
3) Your codes for implementing the three retrieval models (you can submit three different files or one file integrating three of them).
4) For each query, the three results files: resultsVSM1.eval, resultsVSM2.eval, and resultsBM25.eval.
5) Three Kendall Tau values for the above three ranking lists for each query.
6) The Index you built for crawled Wiki pages (20' extra credits).

*Note: please make your codes modular, so that many methods/functions can be re-used in future assignments/project.