Information Retrieval and Text Mining

Programming Assignment 2

Assigned: 13th Oct 2018 Due: 25th Oct 2018

Overview

In this assignment, you will use **the index you created in Assignment 1** to rank documents and create a search engine. You will implement several different scoring functions and compare their results against a baseline ranking produced by expert analysts.

Running Queries

For this assignment, you will need the following two files:

- topics.xml (\\sandata\xeon\Maryam Bashir\Information Retrieval\topics.xml) contains the queries you will be testing. You should run the queries using the text stored in the `<query>` elements. The `<description>` elements are only there to clarify the information need which the query is trying to express.
- corpus.qrel (\\sandata\xeon\Maryam Bashir\Information Retrieval\corpus.qrel) contains the relevance grades from expert assessors. While these grades are not necessarily entirely correct (and defining correctness unambiguously is quite difficult), they are fairly reliable and we will treat them as being correct here.

  The format here is:

  <topic> 0 <docid> <grade>

  - <topic> is the ID of the query for which the document was assessed.
  - 0 is part of the format and can be ignored.
  - <docid> is the name of one of the documents which you have indexed.
  - <grade> is a value in the set {-2, 0, 1, 2, 3, 4}, where a higher value means that the document is more relevant to the query. The value -2 indicates a spam document, and 0 indicates a non-spam document which is completely non-relevant. Most queries do not have any document with a grade of 4, and many queries do not have any document with a grade of 3. This is a consequence of the specific meaning assigned to these grades here and the manner in which the documents were collected.

  This QREL does not have assessments for every (query, document) pair. If an assessment is missing, we assume the correct grade for the pair is 0 (non-relevant).

You will write a program which takes the name of a scoring function as a command line argument and which prints a ranked list of documents for all queries found in topics.xml using that scoring function. For example:

```
$ ./query.py --score TF-IDF
202 0 clueweb12-0000tw-13-04988 1 0.73 run1
202 0 clueweb12-0000tw-13-04901 2 0.33 run1
202 0 clueweb12-0000tw-13-04932 3 0.32 run1
...
214 0 clueweb12-0000tw-13-05088 1 0.73 run1
214 0 clueweb12-0000tw-13-05001 2 0.33 run1
214 0 clueweb12-0000tw-13-05032 3 0.32 run1
...
250 0 clueweb12-0000tw-13-05032 500 0.002 run1
```

The output should have one row for each document which your program ranks for each query it runs. These lines should have the format:

<topic> 0 <docid> <rank> <score> <run>

- <topic> is the ID of the query for which the document was ranked.
- 0 is part of the format and can be ignored.
- <docid> is the document identifier.
- <rank> is the order in which to present the document to the user. The document with the highest score will be assigned a rank of 1, the second highest a rank of 2, and so on.
- <score> is the actual score the document obtained for that query.
- <run> is the name of the run. You can use any value here. It is meant to allow research teams to submit multiple runs for evaluation in competitions such as TREC.

Query Processing

Before running any scoring function, you should process the text of the query in exactly the same way that you processed the text of a document. That is:

1. Split the query into tokens (it is most correct to use the regular expression, but for these queries it suffices to split on whitespace)
2. Convert all tokens to lowercase
3. Apply stop-wording to the query using the same list you used in assignment 1
4. Apply the same stemming algorithm to the query which you used in your indexer

Evaluation

To evaluate your results, we will use a modified form of average precision called Graded Average Precision, or GAP. (Average Precision is used for binary relevance, GAP is modified version of AP for multiple grade relevance. The paper introducing the metric is here, if you're curious.) Script for GAP is available here(\\sandata\xeon\Maryam Bashir\Information

Retrieval\gap.py), you should use it for evaluation. In order to evaluate a run, redirect the output of your program to a file (which I'll call run.txt) and run gap.py on it:

```
$ ./query.py --score TF-IDF > run.txt
$ ./gap.py corpus.qrel run.txt -v
```

The program will output the individual GAP score for each query in run.txt, and also the mean GAP score for all queries.

Scoring Functions

The `--score` parameter should take one of the following values, indicating how to assign a score to a document for a query.

Scoring Function 1: Okapi TF

The parameter `--score TF` directs your program to use a vector space model with term frequency scores. We will use a slightly modified form of the basic term frequency scores known as Okapi TF, or Robertson's TF. In a vector space model, a query or a document is (conceptually, not literally) represented as a vector with a term score for each term in the vocabulary. Using term frequency scores, the component of the document vector for term i is:

$$d_i = oktf(d,i)$$

$$oktf(d,i) = \frac{tf(d,i)}{tf(d,i) + 0.5 + 1.5 \cdot (len(d)/avg(len(d)))}$$

where $tf(d,i)$ is the number of occurrences of term i in document (or query) d, $len(d)$ is the number of terms in document d, and $avg(len(d))$ is the average document length taken over all documents. You should assign a ranking score to documents by calculating their cosine similarity to the query's vector representation. That is,

$$score(d) = \frac{\vec{d} \cdot \vec{q}}{\|\vec{d}\| \cdot \|\vec{q}\|}$$

where $\|\vec{x}\| = \sqrt{\sum_i x_i^2}$ is the norm of vector $\vec{x}$.

Scoring Function 2: TF-IDF

The parameter `--score TF-IDF` directs your program to use a vector space model with TF-IDF scores. This should be very similar to the TF score, but use the following scoring function:

$$d_i = oktf(d,i) \cdot \log \frac{D}{df(i)}$$

where D is the total number of documents, and $df(i)$ is the number of documents which contain term i.

Scoring Function 3: Okapi BM25

The parameter `--score BM25` directs your program to use BM25 scores. This should use the following scoring function for document d and query q:

$$score(d,q) = \sum_{i \in q} \left[\log \left(\frac{D + 0.5}{df(i) + 0.5} \right) \cdot \frac{(1+k_1) \cdot tf(d,i)}{K + tf(d,i)} \cdot \frac{(1+k_2) \cdot tf(q,i)}{k_2 + tf(q,i)} \right]$$

$$K = k_1 \cdot \left((1-b) + b \cdot \frac{len(d)}{avg(len(d))} \right)$$

Where k_1,k_2, and b are constants. For start, you can use the values suggested in the lecture on BM25 (k_1 = 1.2, k_2 varies from 0 to 1000, b = 0.75). Feel free to experiment with different values for these constants to learn their effect and try to improve performance.

Scoring Function 4: Language model with Jelinek-Mercer Smoothing

The parameter `--score JM` directs your program to use a language model with Jelinek-Mercer smoothing. This uses the same scoring function and the same language model, but a slightly different smoothing function. In this case we use

$$p_d(i) = \lambda \frac{tf(d,i)}{len(d)} + (1-\lambda) \frac{\sum_d tf(d,i)}{\sum_d len(d)}$$

The parameter $\lambda \in [0,1]$ allows us to mix the probability from a model trained on document *d* with the probability from a model trained on all documents. You can use λ=0.6. You are encouraged, but not required, to play with the value of λ to see its effect. (What do you expect would happen with λ=0 or λ=1?)

Note that you can calculate the second term using "the total number of occurrences of the term in the entire corpus" (the cumulative term frequency, or ctf) which you have stored in term_info.txt.

Report

Write a short report providing the following information. Describe how each method performed on the queries. Was there a method which was significantly better than the others? Was there a query which was significantly harder than the others? Explain why you got the results you did, and how that compares to what you expected to happen before implementing the rankers. Include a table of the GAP score for each scoring function on each query, and the average GAP score of each scoring function across all queries.

Submission Checklist

Submit your files in a zipped folder named your roll number on **Slate**.

- Your report
- Your source code
- The output ranking for each scoring function (zipped or gzipped)
- **DO NOT SUBMIT INDEX**

Rubric

- Scoring function implementation: 4 Points

  1 point: You correctly implemented Okapi TF
  1 point: You correctly implemented TF-IDF
  1 point: You correctly implemented Okapi BM25
  1 point: You correctly implemented a language model with Jelinek-Mercer smoothing

- Report: 1 Point