Information Retrieval

Programming Assignment 2

Assigned : 8th Oct 2019 **Due: 21st Oct 2019**

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 two 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 clueweb12-0000tw-13-04988 1 0.73 run1
202 clueweb12-0000tw-13-04901 2 0.33 run1
202 clueweb12-0000tw-13-04932 3 0.32 run1
...
214 clueweb12-0000tw-13-05088 1 0.73 run1
214 clueweb12-0000tw-13-05001 2 0.33 run1
214 clueweb12-0000tw-13-05032 3 0.32 run1
...
250 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> <docid> <rank> <score> <run>

- <topic> is the ID of the query for which the document was ranked.
- <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

Scoring Function 1: Okapi BM25

Implement 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 2: Language model with Dirichlet Smoothing

Implement a language model with Dirichlet smoothing. The parameter mu should be set equal to average document length in collection.

Evaluation

To evaluate your results, we will write a program that computes mean average precision of the rank list of documents for different queries. The input to program will be the qrel file (relevance judgments) and scoring file that has rank list of documents. The output should be following measures

P@5

P@10

P@20

P@30

MAP

These measures should be computed for each query. Average for all queries should also be computed.

Report

Make a table for results of each query.

Submission Checklist

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

- Your source code
- The output ranking for each scoring function (zipped or gzipped)
- Report with table of results with Mean average precision for each scoring function
- **DO NOT SUBMIT INDEX**