COMPSCI546_132815_FA20-Applied Information Retrieval Fall 2020

Programming Assignment: Retrieval Models

The purpose of this project is to explore retrieval models on a small collection of documents. You will use the collection, your index, and your retrieval APIs from the previous project.

Scoring

Vector Space Model

Implement cosine similarity as a scoring function. Use log tf $(1 + log(f_{ted})$, log idf $(log(N/n_t)$ for the weights.

BM25

Implement the BM25 as a scoring function. Use k1 =1.5, k2=500, b=0.75

Language-Modeling or Query-Likelihood (QL)

Implement the query likelihood model using:

- Jelinik-Mercer smoothing Use lambda = 0.2
- Dirichlet smoothing Use mu = 1200

Evaluation

We provide a number of "queries" specified in plain English. Please run these queries under each of the models. Be sure to specify all parameters.

Please output the results of these queries in ***trecrun*** format.

TREC Run format

Output Format for the tasks.

For each tasks, a submission consists of a single text file in the format used for most TREC submissions, which we repeat here for convenience. White space is used to separate columns. The width of the columns in the format is not important, but it is important to have exactly six columns per line with at least one space between the columns.

```
Q1 skip alls_well_that_ends_well:0.0            1 1.000 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:0.1            2 0.500 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:0.2            3 0.333 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:1.0            4 0.250 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:1.1            5 0.200 jjfoley-alphabetic
```

```
Q1 skip alls_well_that_ends_well:1.2            6 0.167 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:1.3            7 0.143 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:1.4            8 0.125 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:2.1            9 0.111 jjfoley-alphabetic
Q1 skip alls_well_that_ends_well:2.4            10 0.100 jjfoley-alphabetic
...
Q1 skip winters_tale:4.0                        399 0.003 jjfoley-alphabetic
Q1 skip winters_tale:4.1                        400 0.003 jjfoley-alphabetic
Q1 skip winters_tale:4.2                        401 0.002 jjfoley-alphabetic
...
Q2 skip alls_well_that_ends_well:0.0            1 1.000 jjfoley-alphabetic
Q2 skip alls_well_that_ends_well:0.2            2 0.500 jjfoley-alphabetic
Q2 skip alls_well_that_ends_well:1.0            3 0.333 jjfoley-alphabetic
Q2 skip alls_well_that_ends_well:1.2            4 0.250 jjfoley-alphabetic
...
Q5 skip winters_tale:1.0                        210 0.005 jjfoley-alphabetic
Q5 skip winters_tale:2.1                        211 0.005 jjfoley-alphabetic
Q5 skip winters_tale:3.3                        212 0.005 jjfoley-alphabetic
```

etc. where:

- the first column is the topic number.
- the second column is currently unused and should always be "skip".
- the third column is the scene identifier of the retrieved document.
- the fourth column is the rank the document is retrieved. You will list things sorted by rank within query.
- the fifth column shows the score (integer or floating point) that generated the ranking. This score must be in descending (non-increasing) order.
- the sixth column is called the "run tag" and is traditionally a unique identifier for your group AND for the method used. That is, each run should have a different tag that identifies the group and the method that produced the run.
  - *Use your OIT identifier, and either bm25, ql-jm, or ql-dir. Include the relevant parameters in the run tag*
  - e.g. ***jjfoley-bm25-<k1>-<k2>, jjfoley-ql-jm-<lambda>, jjfoley-ql-dir-<mu>,*** *where you have substituted the actual parameter value for the <parameter> expression.*

Queries

- ***Q1***: the king queen royalty
- ***Q2***: servant guard soldier
- ***Q3***: hope dream sleep
- ***Q4***: ghost spirit
- ***Q5***: fool jester player
- ***Q6***: to be or not to be
- **Q7**: alas
- **Q8**: alas poor
- **Q9**: alas poor yorick
- **Q10**: antony strumpet

Grading Rubric

(5%) Submission is in the correct format.

A ***single*** archive file (zip, tar.gz) was submitted to Moodle and it contains at least the following contents:

- ***report.pdf*** - your report (see below)
- ***src/**** - your source code
- ***README***
  - It has instructions for downloading dependencies the code.
  - It has instructions for building the code.

- It has instructions for running the code.
- ***ql-jm.trecrun, ql-dir.trecrun, bm25.trecrun, vs.trecrun***
  - Query Output Files: these should be in the trecrun format above.
  - Please generate a separate file for each QL, vector space, and BM25 runs.
  - Each file should contain all 10 queries.
- ***judgments.txt***
  - Each line should be a scene id and a number. Please ***only*** include a scene and a number: e.g. `printf("%s %d\n", scene, judgment);` See the the report questions.

(50%) Source code that implements the retrieval models.

Please be aware that you may lose points for problems in your code even if it appears to run correctly. We expect to see code used for all parts of the assignment in your submission.

(45%) The report discusses your work.

- (5%) Description of the system, design tradeoffs, questions you had and how you resolved them, etc. List the software libraries you used, and for what purpose.
- (5%) Do you expect the results for ***Q6*** to be good or bad? Why?
- (5%) Do you expect the results for a new query `setting the scene` to be good or bad? Why?
- (10%) Look at the top ten results for ***Q3*** for each QL and BM25 (maximum of 30 results in total, union the sets of scene identifiers).
  - For each result, record a number from 0 to 3, detailing whether you believe it to be:
    - 0: Not relevant
    - 1: Not really relevant
    - 2: Somewhat relevant
    - 3: Relevant. Note that something can be relevant even if it doesn't contain exactly those query words as a phrase. Ultimately you are the judge of relevance.
  - Include this data in your submission. Each line should be a scene id and a number. Please ***only*** include a scene and a number: e.g. `printf("%s %d\n", scene, judgment);`
- (10%) What will have to change in your implementation to support phrase queries or other structured query operators?
- (10%) How does your system do? Which method appears to be better? On which queries? Justify your answer.

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Friday, September 25, 2020, 11:59 PM
Time remaining	3 days 6 hours
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet

You have not made a submission yet

◀ Java Indexer solution files

Jump to...

Quiz: Retrieval Models ▶