COMPSCI546_132815_FA20-Applied Information Retrieval Fall 2020

Programming Assignment: Indexer

We encourage you to implement the project using Java because we believe it provides solid support for the work in this class. If you wish to use Python or C++ instead, that is fine. If you want to use a different language, please send an email message to the professor to request permission. You should be aware that the TAs and professor are less likely to be able to help you with technical programming problems if you move outside of our comfort zones.

The purpose of this project is to implement the infrastructure necessary for the remainder of the course projects.

Implement an Inverted Index with Positional Information.

Dataset

The dataset is the scenes from the complete works of Shakespeare. This data has been tokenized and is made available as ***shakespeare-scenes.json.gz*** on Moodle.

This dataset has been preprocessed by stripping out punctuation and stemmed using the Krovetz Stemmer. No stopwords have been removed.

An example scene is below:

```
{
  "playId" : "antony_and_cleopatra",
  "sceneId" : "antony_and_cleopatra:2.8",
  "sceneNum" : 549 ,
  "text" : "scene ix another part of the plain enter mark antony and domitiu enobarbu mark antony set we our squadron on
yond side o the hill in eye of caesar s battle from which place we may the number of the ship behold and so proceed
accordingly exeunt "
}
```

This is from the play "Anthony and Cleopatra", Act III, Scene ix (that is not a mistake: the "sceneId" and the actual act/scene are not the same). You will need to use both the "play_id" key and "scene_id" key as metadata in subsequent projects.

Your indexer should perform the following

- Read in the tokenized and stemmed document collection provided in the assignment.
  - Term vectors should be constructed by splitting the given text by spaces (the regex `\\s+`), and ignoring blank strings.
- Build a simple inverted index with positional information. See figure 5.5 (p.135) in the textbook.
  - Use a command line parameter to choose between compressed and uncompressed inverted lists. Compressed lists should use both delta encoding and vbyte compression.
  - Inverted lists and all ancillary index structures must be written to disk, enabling reopening a written index. DO NOT simply write serialized java objects to disk. Bear in mind, although this particular index is small enough to fit into memory, real indexes aren't, so don't implement a memory only solution.
  - Add an appropriate API to your index to enable accessing the vocabulary, term counts, document counts and other statistics that you will require to perform the evaluation activities

will require to perform the evaluation activities.
- Create a Retrieval API that enables performing query retrieval on your index. For this exercise, a raw count model is sufficient.

Evaluation

Using your indexer, you will construct two indexes, one using compression, the second without compression.

1. Compare the vocabulary of the two indexes (terms and counts) to ensure they are identical.
2. Randomly select 7 terms from the vocabulary. Record the selected terms, their term frequency and document frequency. Do this 100 times.
3. Using Dice's coefficient (see section 6.2.1 and page 201), identify the highest scoring two word phrase for each of the 7 terms in your set of 100.
4. Using your Retrieval API and the 100 sets of 7 terms as your queries, perform a timing experiment to examine the compression hypothesis. Repeat the experiment using the 100 sets of ~~7 two~~ 14 ~~word phrases~~ terms.

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.
- Query files: these should contain the 100 sets of 7 terms, one set per line, in a plain text file and the 100 sets of ~~7 two~~ 14 terms ~~phrases~~, one set per line.

(50%) Implementation

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%) Report

- (10%) Description of the system, design tradeoffs, questions you had and how you resolved them, etc. Document your index and retrieval APIs.
- (5%) Why might counts be misleading features for comparing different scenes? How could you fix this?
- (5%) What is the average length of a scene? What is the shortest scene? What is the longest play? The shortest play?
- (25%) Present your experimental results. Does the compression hypothesis hold? What conditions might change the results of the experiment, and how would you expect them to change?

shakespeare-scenes.json.gz

Submission status

Submission status	No attempt
Grading status	Not graded

Due date	Friday, September 11, 2020, 11:59 PM
Time remaining	2 days 8 hours
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet

◀ Architecture

Jump to...

Quiz: Indexing ▶