Department of Computer Science and Software Engineering, Concordia University

COMP 479/6791 Final Project

Due Date: November 29th, midnight

Demos: December 2nd and 3rd

This is a group project to be performed in groups of 3 students.
A penalty of 10% per day will applied to late submissions.

Description: Build an Information Retrieval system that crawls the entire concordia.ca domain (or a domain that you feel would provide the information needed for writing a good report, such as other university websites, etc), treats each web page as a document which are then tokenized and indexed, clusters the documents using K-means clustering and allows the user to perform queries for which the results are ranked using BM25F.

BM25F is slightly different from BM25, but not by much. The difference is that term frequencies are multiplied by weights which are based on where the terms are extracted from in the document. For example, take a document like:

```
<title>These terms have more weight</title>
<h1>These terms have a bit less weight</h1>
<p>These terms have a weight equal to their raw term frequency</p>
```

You could say that terms found between <title> tags should get a weight of TF x 5, between the <h1> tag should get a weight of TF x 2, and terms found between the <p> tag get a weight of TF x 1. Note that the weights provided here are bogus and we expect you to play around with your system in order to find the weighting scheme that seems the most effective to you. There is no "god almighty said this is the one" type answer to this, you will need to test and argue for your preferred weighting scheme.

You may build everything from scratch, or use libraries that are available online (so long as you mention which ones you have used and for what purpose). You may also re-use implementations from Project 1 and 2. Note that when using libraries or other people's code, you should still be capable of explaining to us what is happening in the background, not just tell us "I used this API call and was given documents that were ranked!" You should know *how* these documents are ranked and be capable of arguing about why this is a good solution to the problem we are asking you to solve.

In order to do complete this assignment, your system will need several components. We provide here a number of libraries that can be used for these tasks:

- A crawler (Python: http://scrapy.org/, Java: http://www.cs.cmu.edu/~rcm/websphinx/, C#: https://code.google.com/p/abot/)
- An html parser (Python: http://www.crummy.com/software/BeautifulSoup/, Java: http://jsoup.org/)
- An indexer (Python: https://pypi.python.org/pypi/Whoosh/, Java: http://lucene.apache.org/core/)
- A ranked retrieval system (Whoosh and Lucene also provide this)
- A clustering system using K-means clustering methods
- A user interface allowing to perform queries and retrieve documents

Deliverables: As per the last projects, you will be graded on three things.

- The implementation **(25%)**
    1. Crawling (2.5%)
    2. Parsing (2.5%)
    3. Indexing (2.5%)
    4. Clustering (10%)
    5. BM25F document retrieval (5%)
    6. User Interface(2.5%)

- A report **(50%)**
    1. Description of the crawling, parsing and indexing (methods or libraries used, data structures, description of the overall flow of the system) (10%)
    2. Description of the clustering system (10%)
    3. Description of the user interface and ranked retrieval querying system (10%)
    4. An analysis of the clustering system (how many clusters have been created, what can you tell us about the pages that are clustered together, how does your system use said clusters during retrieval of documents, how would this work on a much larger document collection such as the entire world wide web) (20%)

- Demonstration **(25%)**
    1. Explain the crawling and parsing steps (5%)
    2. Explain the clustering methodology and describe to us the results (10%)
    3. Demonstrate the system in action (10%)

A few notes:

- When you are crawling, as you parse the documents, two things should happen: you should find links to other documents, and you should index the current document. Both these steps happen at the same time.
- When crawling: respect robots.txt, limit your crawler to one page every 10 seconds, limit the depth of your crawl to 4 pages and limit the final crawl to 1000 documents.
- While developing: you don’t need to crawl 1000 documents every time, limit your crawler to a very small number of pages if you are only testing different components of your system.
- You crawler should allow you to specify the user-agent string that is used when fetching a page. It would be nice to identify yourself there as a student from this course, that way when the IT department gets mad about the fact that you did not respect robots.txt, they’ll know who to yell at.
- For the report: we do not care so much about the specifics of your implementation, but instead expect you to provide insightful analysis of the results of using such an implementation. In other words, we do not care about how some for loop is used to weight the terms while using BM25F, but we do care about how you decided to weight the terms and how that influences the results you see.