Assignment 4: MetaData
CS430 - Information Retrieval

Michael Metral (mdm257)

December 12, 2007

1   Absract

Given a corpus of web pages, this project is intended to proceed through the various listings collecting metadata for each page, organizing the metadata and finally organizing it. The procedure is as follows:

- Choose a web page

- Navigate through the source code of the web page extracting all embedded hyperlinks, or better known as anchor's specifying another web page with a certain text. i.e. a link pointing to `http://www.google.com` might have the description 'Google Search'

- After all anchor's have been extracted, calculate the PageRank of all the web pages in the corpus. PageRank is used as a means of organizing websites through the idea that web pages that are linked to the most, tend to have a higher result when performing a search.

  Thus, the more popular web pages relevant to a search query will be the top hits as opposed to the lower ranked hits in the search results.

- Finally, after the metadata has been organized and PageRank calculated, a search on the metadata is available on the corpus, returning relevant results on a query on a scale similar to that of Google's current search algorithm.

2   How To Run

1. Unpack the zipped file a4.zip

Command Line:
2.   Direct the command line to the a4/Code directory
3. Type: *python metadata.py*

Directory Icon:
2. Open the directory a4/Code
3. Double-Click metadata.py

*Note: Only 1 program is written to test all 5 parts.

Running the program:
To test the search, enter in one search term. Return: The results ranked by PageRank displaying the following information:

- The URL ID of the web page

- The URL of the web page

- The title of the web page

- A snippet of the web page which could be either actual content from the page (based on paragraph tags) or the first 250 bytes of the web page's source code if had empty paragraph tags

i.e. Enter in a search term such as "research" to see the information displayed above for the query. Note: As previously stated the snippet of website varies. If a web page has text within paragraph tags this will be the context printed. However, if the web page does not contain any text between paragraph tags, the first 250 bytes of the web page's source code is used to fill the snippet.

Note:
-The metadata file is located in a4/Code/metadata.txt
-The test and results file is located in a4/Code/test_and_results.txt
-The program is written in Python 2.5
-The program uses NumPy, a matrix package for Python. The download is at `http://numpy.scipy.org/` under the side option, "Download NumPy"

3 Data Structures

3.1 Word List

To organize the index to allow both random look-up and sequential processing a tree structure and dictionary were implemented. Using Python's list structure to implement the tree, an item is distinguished amongst the rest by assigning each item in the list a tuple as such:

((word, parent, left child, right child), term). What this means is that as each word is indexed into the tree all words have a parent and children words based on the actual indexing itself.

3.2 Postings File

After the word has been indexed into the tree, the postings file is updated to keep track of the word's various posts. Using Python's dictionary structure each term is inserted into the dictionary using a key:value. The key is the word and the value is itself a dictionary. In this value there is also a key:value pair as such:
(URL : short index record).
   Thus, an entry in the postings dictionary looks like such:
(term : (URL : short index record). So if a word appears in multiple web page's, we would only have one entry in the postings dictionary for the word, and various entries for the dictionary used as its value to distinguish all websites holding the word.

4 Functions and Algorithms

4.1 Similarity between Query and Index

With the search option matrix extracted from the postings file, we can use each cell of this particular matrix to locate the web pages with the highest term frequency for the query. This allows a set of pages to then be ranked by PageRank and return results for the query.
   However, we cannot directly compute the dot-products with the query as a list of terms so it must be constructed into a vector. Once a vector is formed we can use the matrices returned

from the postings file to compute the similarity comparison.

4.2 Query Vector Creation

When a query is given, it is taken in by the program as a list. This list must be converted to a vector of size *Terms x 1* in order to perform the necessary functions through out the similarity process.

4.3 PageRank

PageRank is used to estimate the popularity of web pages. The higher the count that other web pages point to a particular web page, the higher its rank will be. Links from highly ranked pages are given a greater weight than links from lower ranked pages.

PageRank thus ranks the pages according to the relative frequency with which they are visited. With a damping factor and number of iterations, PageRank works to converge the frequency to a stable value and operates as such :

- Start at a random page on the web

- With probability 1-d, selects any random page and goes to it

- With probability 1-d, selects a random hyperlink from the current page and jumps to that page

- Repeat steps 2 and 3 a large number of times

When a similarity comparison has been computer between a query and the index of the terms from all web pages in the corpus, the results are ranked according to their corresponding PageRank value.

4.4 HTML Parser

To extract all the anchor text from the hyperlinks within a web page, the Python library HTMLParser was used. With this library, a parsing class was constructed to navigate the source code of the web page being examined collecting 'a href' (hyperlink) tags.

The parsing class was designed specifically to collect the anchor and the anchor text for both correct and malformed HTML source code to capture all possible hyperlinks without leaving any behind.

Note: A timeout limit of 2 seconds was implemented and used to dictate web pages who hang or do not respond to a HTML GET command. This can vary the results in a miniscule amount if web pages that pertain to the corpus used do not respond to the GET command in time, but not enough to skew the results immenseley.

5 Data Types

To simplify the storing of various information through out the collection of metadata from the corpus of web pages, a set of data types were constructed to organize the information efficiently.

5.1 URL Data

This is a class used to hold information pertinent to each URL of the corpus. It includes:

- URL ID - The id associated with the web page in the test data

- URL - The web page's URL

- Title - The title of the web page

- Extracted Anchors (an Anchor Data DataType)

5.2   Anchor Data

This class encapsulates all the information relevant to an anchor that has been extracted from a web page, it consists of:

- Anchor - The URL of the web page

- Anchor Text - The text describing the hyperlink

- URL ID - Which URL hold's this anchor text in their web page

5.3   Short Index Record

This class is used to hold all pertinent metadata corresponding to a URL. It consists of:

- URL ID - The id associated with the web page in the test data

- PageRank - the PageRank for the specific URL

- Cited Anchor Text - The anchor text used to describe a certain web page from other web pages pointing to it.

5.4   Tests and Results

5.4.1   Test #1 - Hits That Include/Don't Include Anchor Text Terms

Test: Will websites who are referred to using a specific anchor text actually contain the anchor text? i.e. if a website cites http://www.google.com as "Google Search", does the google.com website actually have the words "Google" and "Search"

Referring to Figure 1, we notice of the following:

Results: Although the majority of the hits for a search query contain the words pertinent to

Query: "talk"		
URL	Contains Anchor Text	
106	No	
107	Yes	
108	No	
109	Yes	

Figure 1: Search Hits That Include/Don't Include Anchor Text Terms

the anchor text used for citing these pages, some webpages on occasion do NOT actually contain the word itself.

5.4.2   Test #2 - Iteration Convergence in PageRank

Iterations	Query	Ranked Results by URL ID
50	home	3,4,2,12, 82, 34
20	home	3,4,2,12, 82, 34
50	research	6,7,21,42,122,111
20	research	6,7,21,42,122,111
50	spring	14,51,49,47,45
20	spring	14,51,49,47,45

Figure 2: Iteration Convergence in PageRank

Test: Will changing the number of iterations used in the PageRank algorithm affect the ranking of the hits for a specific query?

Referring to Figure 2, we notice of the following:

Results: After using a significantly smaller amount of iterations, the use of different iterations in the PageRank algorithm does NOT affect the ranking of the results on a search query.

5.4.3 Test #3 - High Citing Links vs. Hit Ranking

Query: "about"	
URL ID	# of Citing Links
3	17
135	6
91	13
Query: "home"	
URL ID	# of Citing Links
3	17
4	25
2	18
12	29
82	5
34	12

Figure 3: High Citing Links vs. Hit Ranking

Test: Will the rank of hits for a search query be affected by the count of citing anchors a particular website has? i.e. is a popular website ranked higher than a less popular website

Referring to Figure 3, we notice of the following:

Results: The number of websites citing a website through the use of anchor text's do NOT correlate with the ranking of hits for a search query. Therefore, the popularity of a page based on anchor text's does not translate into the ranking of websites.

5.4.4 Test #4 - Polysemy

Test: Will all the definitions of a term be fairly returned as hits when performing a search query?

Referring to Figure 4, we notice of the following:

Results: Multiple definitions occur very rarely

Query: "time"		
URL ID	Definition	
98	Systematic structure	
84	A specific moment	

Figure 4: Polysemy

throughout the metadata, but in this instance of the term "time," both interpretations are captured. Though polysemy isn't correlated to PageRank or web searching, the matching of term's in a search query to the corpus seem to present a varied amount of results as opposed to just one specific set for a certain definition.

5.4.5 Test #5 - Average Search Time

Query	Search Time (seconds)	
home	0.000865	
arts	0.000247	
about	0.000569	
forms	0.000633	
mobile	0.000446	
cornell	0.001532	
for	0.001789	
Avg. Time	0.000869	seconds

Figure 5: Average Search Time

Test: What is the average search time a query takes to execute and return results?

Referring to Figure 5, we notice of the following:

5

Results: The average search time is pretty satisfactory. The only reasoning for any delay is the fact that when URL's don't have correctly formed paragraph tags, I use the a bit of the source code to represent the web page's snippet. Pulling this source code from the webpage is what delay's the process a bit, but not by much.