Assignment 1

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

September 19, 2016

Contents

Listings

List of Tables

1 Question 1.1

1.1 Question

Think up and write down a small number of queries for a web search engine. Make sure that the queries vary in length (i.e., they are not all one word). Try to specify exactly what information you are looking for in some of the queries. Run these queries on two commercial web search engines and compare the top 10 results for each query by doing relevance judgments. Write a report that answers at least the following questions: What is the precision of the results? What is the overlap between the results for the two search engines? Is one search engine clearly better than the other? If so, by how much? How do short queries perform compared to long queries?

1.2 Resources

The search engines Google [1] and DuckDuckGo [2] were used to obtain the results.

1.3 Methodology

The following search queries were issued to each of the two mentioned search engines:

1. professional skateboarder song
2. daewoo song
3. rollerblade song korea
4. daewon song

The expected relevant pages will contain information regarding the professional skateboarder named Daewon Song. A score is assigned to each search engine for each query using the following equation:

$$\frac{R}{10}$$

Where R the number of relevant documents returned for the given query. Overlap is calculated by using the following equation:

$$\frac{\|G_i \cap D_i\|}{10}$$

Where G_i is the set of pages returned by Google for query i and D_i is the set of pages returned by DuckDuckGo for query i. This result is divided by the size of the overall result set for each query per search engine, ten.

A brief description of the results is provided, with any items that stand out to the reviewer given extra measure. After the detailed listing of results, Table 1 lists the relevancy scores and overlap for each search engine per query.

First Query: “professional skateboarder song”

The relevancy scores from the first query were evenly matched for both search engines, five out of ten relevant documents each. Interestingly, Google provided more social media related results, whereas DuckDuckGo provided a couple celebrity net worth results.

Second Query: "daewoo song"

Trying to throw a curveball at the engine with this one, it is intentionally misspelled. Results for both engines show active error correction and query suggestion functionality, which helps to increase relevance when a user perhaps does not know the correct spelling of an item they wish to find. Google wins this round with eight out of ten versus DuckDuckGo's result of 6 out of ten.

Third Query: "rollerblade song korea"

Somewhat of another curve ball, although this is *close* to a "correct" query, since rollerblading and skateboarding are similar "extreme" sports. The results, however, show that these types of judgements are not available to a search engine, as both websites returned zero relevant documents pertaining to Daewon Song. Perhaps the error simply lies between the chair and the keyboard.

Fourth Query: "daewon song"

This is a control, as it is exactly the name of the target of the search query for this question. Both search engines provided a perfect relevancy score, which was not surprising considering that the information need was embodied in a person with a unique name, making narrowing down the relevant results quite easy based on the name alone.

1.4 Results

Table 1 shows the relevancy scores and overlap for each search engine per query.

Query	Google Relevance	DuckDuckGo Relevance	Overlap
professional skateboarder song	0.5	0.5	0.1
daewoo song	0.8	0.6	0.2
rollerblade song korea	0.0	0.0	0.0
daewon song	1.0	1.0	0.7

Table 1: Relevance Scores

Both search engines performed relatively similarly to each other for the given queries. Neither one provided a significantly higher number of relevant results for any of the queries. When given the ideal query, overlap was high, although not at 1.0, while also providing a "perfect" relevancy score of 1.0. The length of the query did not seem to be a strong controlling factor, term accuracy seemed to decide the relevancy of the top ten results for a particular topic.

2 Question 1.2

2.1 Question

Site search is another common application of search engines. In this case, search is restricted to the web pages at a given website. Compare site search to web search, vertical search, and enterprise search.

2.2 Resources

The textbook *Search Engines: Information Retrieval in Practice* [3] was used to answer this question.

2.3 Answer

Site search, as stated in the question, refers to a general search query, with results limited to pages within a particular host domain, or website. This is similar to a vertical search when the site used as the site-constraint for a site-specific search is a website that focuses on a single topic, but

Many websites provides a site-specific search feature, such as Internet Movie Database, an informational website in the domain of film and television. This search functionality is provided to allow a user to quickly find a movie, TV show or actor they'd like to learn about. A vertical search engine may be about movies, but it is not necessarily going to have it's own extensive repository of information within the domain of interest.

The key difference between a vertical search and a site-specific search is the search agent, or who is conducting the search, and who controls the document collection over which the search is made. With a vertical search, the agent is retrieving results from many different websites, which are outside the control of the search agent. Thus, to build the search index on these documents, the vertical search provider must crawl the websites pertaining to their search topic, a task which requires care if the information is to be kept on-topic. This differs from a site-specific search in that site-specific searches are conducted by the site administrators; the search itself is a service offered by a website, using the document collection owned by the site administrators. This gives the proprietors of such a service an advantage in that web crawling is not required, eliminating many problems public, general search engines face. These include network bandwidth, not a problem if the index-builders run on the local machines.

3 Question 3.7

3.1 Question

Write a program that can create a valid sitemap based on the contents of a directory on your computer's hard disk. Assume that the files are accessible from a website at the URL http://www.example.com. For instance, if there is a file in your directory called homework.pdf, this would be available at http://www.example.com/homework.pdf. Use the real modification date on the file as the last modified time in the sitemap, and to help estimate the change frequency.

3.2 Resources

With the Python programming language [4], the script `sitemapgen.py`, in Listing 3, was created to perform the necessary tasks to complete Question 3.7. The output matches the format found at Sitemaps.org [5].

The script uses the last modified time of each file to estimate the change frequency and also escapes special characters to ensure URLs are valid. The DOM is built programatically while the script traverses the file system and when this process is complete the document is printed to standard out.

3.3 Answer

Running the script on the test directory matching the structure in Listing 1 and the output is shown in Listing 2.

```
smgen
+---sitemapgen.py
+---testdir
    +---dir1
    |   +---testfile
    +---dir2
        +---dir3
        |   +---testfile
        +---testfile
```

Listing 1: test directory structure

```
<?xml version="1.0" ?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
    <url>
        <loc>http://www.example.com/testdir/dir1/testfile</loc>
        <lastmod>2016-09-18</lastmod>
        <changefreq>daily</changefreq>
        <priority>0.5</priority>
    </url>
    <url>
        <loc>http://www.example.com/testdir/dir2/testfile</loc>
        <lastmod>2016-09-18</lastmod>
        <changefreq>daily</changefreq>
        <priority>0.5</priority>
    </url>
    <url>
        <loc>http://www.example.com/testdir/dir2/dir3/testfile</loc>
        <lastmod>2016-09-18</lastmod>
        <changefreq>daily</changefreq>
        <priority>0.5</priority>
    </url>
    <url>
        <loc>http://www.example.com/sitemapgen.py</loc>
        <lastmod>2016-09-19</lastmod>
        <changefreq>always</changefreq>
        <priority>0.5</priority>
    </url>
</urlset>
```

Listing 2: site map generator output

4 Question 3.8

4.1 Question

Suppose that, in an effort to crawl web pages faster, you set up two crawling machines with different starting seed URLs. Is this an effective strategy for distributed crawling? Why or why not?

4.2 Resources

The textbook *Search Engines: Information Retrieval in Practice* [3], *Parallel Crawlers* [6], and *Efficient URL Caching for World Wide Web Crawling* [7] were used to answer this question.

4.3 Answer

5 Appendix A

```
import sys
import os
import argparse
import datetime
import time
import urllib

from os.path import getmtime, isdir, isfile

import xml.dom.minidom as md


def append(doc, urlset, loc, lastmod, changefreq, priority='0.5'):
    url = doc.createElement('url')
    urlset.appendChild(url)

    loc_element = doc.createElement('loc')
    loc_element.appendChild(doc.createTextNode(loc))
    url.appendChild(loc_element)

    lastmod_element = doc.createElement('lastmod')
    lastmod_element.appendChild(doc.createTextNode(lastmod))
    url.appendChild(lastmod_element)

    changefreq_element = doc.createElement('changefreq')
    changefreq_element.appendChild(doc.createTextNode(changefreq))
    url.appendChild(changefreq_element)

    priority_element = doc.createElement('priority')
    priority_element.appendChild(doc.createTextNode(priority))
    url.appendChild(priority_element)


YEAR = 3.154e+7
MONTH = 2.592e+6
WEEK = 604800.0
DAY = 86400
HOUR = 3600
MINUTE = 60

def estimate_changefreq(posixtime):
    timenow = time.time()
    delta = timenow - posixtime
    if delta > YEAR:
        return 'never'
    elif delta > MONTH:
        return 'yearly'
    elif delta > WEEK:
        return 'monthly'
    elif delta > DAY:
        return 'weekly'
    elif delta > HOUR:
        return 'daily'
    elif delta > MINUTE:
        return 'hourly'
    else:
        return 'always'


def convertdate(posixtime):
    return datetime.datetime.utcfromtimestamp(posixtime).strftime('%Y-%m-%d')


def delve(root, folder, doc, urlset):
    items = os.listdir(root + folder)
    for item in items:
        filepath = root + folder + item
        if isfile(filepath):
            loc = args.host + urllib.quote(folder + item)
            lastmodsecs = getmtime(filepath)
            lastmod = convertdate(lastmodsecs)
            changefreq = estimate_changefreq(lastmodsecs)
            append(doc, urlset, loc, lastmod, changefreq)
        elif isdir(filepath):
```

```python
            delve(root, folder + item + os.sep, doc, urlset)


def test(doc, urlset):
    append(doc, urlset, 'www.google.com', '2016-09-17', 'always', '0.8')
    append(doc, urlset, 'www.duckduckgo.com', '2016-09-17', 'daily')
    print doc.toprettyxml()


if __name__ == '__main__':
    # parse arguments
    parser = argparse.ArgumentParser('site map generator')
    parser.add_argument(
        '-test',
        '-t',
        action='store_true')
    parser.add_argument(
        '--path',
        '-p',
        default='.')
    parser.add_argument(
        '--host',
        default='http://www.example.com/')
    args = parser.parse_args()

    # create a document
    doc = md.Document()
    urlset = doc.createElement('urlset')
    urlset.setAttribute('xmlns', 'http://www.sitemaps.org/schemas/sitemap/0.9')
    doc.appendChild(urlset)

    # if desired, perform simple test and return
    if args.test:
        test(doc, urlset)
        sys.exit(0)

    # parse path from args
    path = args.path
    if path[len(path)-1] != os.sep:
        path = path + os.sep

    # iterate over all items in doc
    delve(path, '', doc, urlset)

    print doc.toprettyxml()
```

Listing 3: sitemapgen.py

6 References

[1] Google. Available at: http://www.google.com. Accessed: 2016/09/17.

[2] DuckDuckGo. Available at: http://www.duckduckgo.com. Accessed: 2016/09/17.

[3] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Pearson, first edition, February 2009.

[4] The Python Programming Language. Available at: https://www.python.org/. Accessed: 2016/09/17.

[5] Sitemaps XML format. Available at: http://www.sitemaps.org/protocol.html. Accessed: 2016/09/17.

[6] Junghoo Cho and Hector Garcia-Molina. Parallel Crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 124–135, New York, NY, USA, 2002. ACM.

[7] Andrei Z. Broder, Marc Najork, and Janet L. Wiener. Efficient URL Caching for World Wide Web Crawling. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 679–689, New York, NY, USA, 2003. ACM.