Assignment 3

Dr. Nwala

Nathaniel Everett

February 11, 2018

Contents:

Problem 1. Download the 1000 URIs from assignment #2. "curl", "wget", or "lynx" are all good candidate programs to use. We want just the raw HTML, not the images, stylesheets, etc.

from the command line:

% curl http://www.cnn.com/ > www.cnn.com

% wget -O www.cnn.com http://www.cnn.com/

% lynx -source http://www.cnn.com/ > www.cnn.com

"www.cnn.com" is just an example output file name, keep in mind that the shell will not like some of the characters that can occur in URIs (e.g., "?", "&"). You might want to hash the URIs to associate them with their respective filename, like:

% echo -n "http://www.cs.odu.edu/show_features.shtml?72" | md5
41d5f125d13b4bb554e6e31b6b591eeb

("md5sum" on some machines; note the "-n" in echo -- this removes the trailing newline.)

Now use a tool to remove (most) of the HTML markup for all 1000 HTML documents. "python-boilerpipe" will do a fair job see (http://ws-dl.blogspot.com/2017/03/2017-03-20-survey-of-5-boilerplate.html):

```
from boilerpipe.extract import Extractor
extractor = Extractor(extractor='ArticleExtractor', html=html)
extractor.getText()
```

Keep both files for each URI (i.e., raw HTML and processed).
Upload both sets of files to your github account.

The first part of this was done via a shell script, named getHTML.sh was constructed for this one purpose. It is run with a parameter, which would be the linklists.txt output from Assignment 2. The program uses the curl method. The output is put into a directory named raw_html, which is itself in another directory named data. The script, if run more than once, **will** overwrite the files currently in raw_html. I managed to get 996 different html files, all in the local directory, and all can be opened properly, although not all of them were what I originally expected. Some of the sites did not come out with the raw_html, but others were fine. Some I found out were redirects and some only had 0 kb meaning they were blank pages.

The second part was attempted as a python script. Unfortunately, my machine was unable to run any of the required or recommended python libraries. The boilerpipe3 library would not work even after a pip install on my machine, and I even attempted to install it on my home computer with no success. Justext was suggested to me, but it too would not install. I ended up using a function known as clean_html. The python script is called gethtmlscript.py. I had some major problems with the urllib.request library, so I decided on codecs along with BeautifulSoup library to try and extract all of

the unneeded HTML parts and get the plain text. The output was supposed to be in a text file named url0.proc.txt, showcasing the processed text. Not only did this not get formatted properly, but the utf-8 encoding stopped at a certain file. I can imagine this working better on a directory with fixed URL files, however.

Problem 2. Choose a query term (e.g., "shadow") that is not a stop word (see week 5 slides) and not HTML markup from step 1 (e.g., "http") that matches at least 10 documents (hint: use "grep" on the processed files). If the term is present in more than 10 documents, choose any 10 from your list. (If you do not end up with a list of 10 URIs, you've done something wrong).

As per the example in the week 5 slides, compute TFIDF values for the term in each of the 10 documents and create a table with the TF, IDF, and TFIDF values, as well as the corresponding URIs. The URIs will be ranked in decreasing order by TFIDF values. For example:

Table 1. 10 Hits for the term "shadow", ranked by TFIDF.

TFIDF	TF	IDF	URI
0.150	0.014	10.680	http://foo.com/
0.044	0.008	10.680	http://bar.com/

You can use Google or Bing for the DF estimation. To count the number of words in the processed document (i.e., the deonminator for TF), you can use "wc":

```
% wc -w www.cnn.com.processed
   2370 www.cnn.com.processed
```

It won't be completely accurate, but it will be probably be consistently inaccurate across all files. You can use more accurate methods if you'd like, just explain how you did it.

Don't forget the log base 2 for IDF, and mind your significant digits!

https://en.wikipedia.org/wiki/Significant_figures#Rounding_and_decimal_places

---

The query term used was “coach” as it pertains to the links retrieved. An attempt to use another python script, named tfidf.py. I had looked up a very interesting site for both TF and IDF and how to integrate them into a python script: https://stevenloria.com/tf-idf/. The code in tfidf.py was written much like the one on that website, and I found it useful. The program however was made to manually read documents, and still did not function as I would expect, but changes to the document variables could make it run better.

Unfortunately due to not being able to correctly figure out how to do the second part of Problem 1, I decided to manually look up random links in the raw_html directory for the ones that had the term "coach" in them. I ended up manually computing both term frequency and inverted document frequency for each of the 10 URIs I picked out. They are all listed in a .xls spreadsheet document. The spreadsheet was converted into PDF format, called TableTFIDF.pdf. Look for the second page, 10 Hits for the term "coach", ranked by TFIDF to find the proper part of Problem 2. The computations were rounded to three significant digits where applicable.

Problem 3. Now rank the same 10 URIs from question #2, but this time by their PageRank. Use any of the free PR estimaters on the web, such as:

http://pr.eyedomain.com/
http://www.prchecker.info/check_page_rank.php
http://www.seocentro.com/tools/search-engines/pagerank.html
http://www.checkpagerank.net/

If you use these tools, you'll have to do so by hand (they have anti-bot captchas), but there are only 10 to do. Normalize the values they give you to be from 0 to 1.0. Use the same tool on all 10 (again, consistency is more important than accuracy). Also note that these tools typically report on the domain rather than the page, so it's not entirely accurate.

Create a table similar to Table 1:

Table 2. 10 hits for the term "shadow", ranked by PageRank.

PageRank	URI
0.9	http://bar.com/
0.5	http://foo.com/

Briefly compare and contrast the rankings produced in questions 2 and 3.
This was much easier than the previous two parts. http://pr.eyedomain.com/ was used to get the page ranks. These are shown in TableTFIDF.pdf at the third page, and also is provided as another pdf file named PageRank.pdf. The PageRank rankings correspond mostly to how popular a site is. A page rank of 8 is a very popular site, like www.espn.com. The values are normalized between 0 and 1.0, meaning that a page rank of 8 is returned as 0.8 in the pdf file. Notice that the table is not sorted, unlike the TF-IDF values, which were sorted in decreasing order. There is not much order between either PageRank or TF-IDF rankings.

Problem 5. Compute a ranking for the 10 URIs from Q2 using Alexa information (see week 4 slides). Compute the correlation (as per Q4) for all pairs of combinations for TFIDF, PR, and Alexa.

This one involved going to https://www.alexa.com/siteinfo and inputting each of the URIs. The ranks provided are on the first page of TableTFIDF.pdf along with the TFIDF and PageRank values. The

ranks are all global ranks. Alexa's rankings are inversely proportional to PageRank's rankings, as the higher numbers correlate to the lower page ranks. So www.espn.com has a high PageRank, but a low Alexa rank. The Alexa ranks are also on a different pdf file, Alexa.pdf.

Problem 6. Give an in-depth analysis, complete with examples,
graphs, and all other pertinent argumentation for
Kristen Stewart's (of "Twilight" fame) Erdos-Bacon number.

Https://en.wikipedia.org/wiki/Talk:Erdös-Bacon_number has an interesting discussion point involving the coauthor of the paper who wrote with Kristen. Her Erdös-Bacon number has been set to 5:

1. Kristen co-authored *Bringing Impressionism to Life with Neural Style Transfer in Come Swim* with **Bhautik Joshi**
2. Bhautik co-authored *Application-Driven Quantitative Assessment of Approaches to Mesh Generation* with **Simon K. Warfield**
3. Simon co-authored *An EM algorithm for shape classification based on level sets.* with **Alan S. Willsky**
4. Alan co-authored *Energy-Latency Tradeoff for In-Network Function Computation in Random Networks* with **Béla Bollobás**
5. Béla co-authored *Random induced graphs* with **Paul Erdös**

There was an interesting argument in where Bhautik, the co-author, had been told that there needed to be a reliable source. The name Kristen Stewart could actually belong to someone else who is not the actress, along with a few claims that the actress herself never attended college nor publish something in a scientific journal.

Problems answered: 1, 2, 3, 5, 6
Files included:
Assignment3report.pdf (this file)
listlinks.txt (output from Assignment2)
data/raw_html directory (contains the raw html files from the links in listlinks.txt as well as url0.proc.txt and getcleanhtml.py, which are the attempts at getting the clean html texts in a text file).
tfidf.py (attempt at computing TF-IDF files by python script)
TableTFIDF.pdf (shows the full comparison between TF-IDF, PageRank, and Alexa, along with several pages of the other tables, TF-IDF comparison on page 2)
PageRank.pdf (separate pdf for PageRank rankings)
Alexa.pdf (separate pdf for Alexa rankings)