Introduction to Web Science: Assignment #3

Dr. Nelson

Alexander Nwala

Thursday, October 2, 2014

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, 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. “lynx” will do a fair job:

```
% lynx -dump -force_html www.cnn.com > www.cnn.com.processed
```

Keep both files for each URI (i.e., raw HTML and processed).
If you’re feeling ambitious, “boilerpipe” typically does a good job for removing templates:
https://code.google.com/p/boilerpipe/

SOLUTION
The solution for this problem is outlined by the following steps:

1. **Hash URIs to derive filenames:** This was achieved through the python hashlib.md5 method as outlined in Listing 1:

Listing 1: Hash function

```
#Hash URIs function
def getHashFromURI(uri):
     md5hash = ''
     if( len(uri) > 0 ):
          # Assumes the default UTF-8; http://www.pythoncentral.io/hashing-strings
              -with-python/
          hash_object = hashlib.md5(uri.encode())
          md5hash = hash_object.hexdigest()

     return md5hash
```

2. **Extract raw HTML:** This was achieved through the use of the “curl” command as outlined in Listing 2:

Listing 2: Extract HTML function

```
#Extract HTML function
def extractHTMLAndSave():

     if( len(uriHashDictionary) > 0 ):
```

Problem 1 continued on next page. . .

```
        count = 0
        for uri in uriHashDictionary:

            filename =  str(count) + '-' + uriHashDictionary[uri] + '.html'
            co = 'curl -s -L ' + uri + ' > ./RawHtml/' + filename

            commands.getoutput(co)
            count = count + 1
```

3. **Process HTML files:** To remove the HTML tags, the "lynx" command was used as outlined in Listing 3:

Listing 3: Extract HTML function

```
#Process HTML functions; remove HTML tags
def extractHTMLAndSave():
    if( len(uriHashDictionary) > 0 ):
        count = 0
        for uri in uriHashDictionary:
            filename =  str(count) + '-' + uriHashDictionary[uri] + '.html'
            co = 'curl -s -L ' + uri + ' > ./RawHtml/' + filename

            commands.getoutput(co)
            count = count + 1
```

```
The folder "RawHtml" contains all the downloaded html files, and the folder
"ProcessedHtml" contains the processed versions
```

Problem 2

Choose a query term (e.g., "shadow") that is not a stop word (see week 4 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 4 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/
0.085   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
```

Problem 2 continued on next page. . .

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

SOLUTION
Using the query term "plan", 49 documents were retrieved and the TF-IDF calculated for all 49; the top 10 (TF-IDF criteria) documents were collected: Table 1 summarizes the outcome. The total number of documents in the corpus was set to 20 billion, and with the use of Google, the count of document with the term was set to 206 million.

Table 1: Rank Table Based On TF-IDF

ITEM	TF-IDF	TF	IDF	URI
1	0.001	0.000	6.601	http://market-ticker.org/...
2	0.002	0.000	6.601	http://www.seattlepi.com/
3	0.002	0.000	6.601	http://www.philly.com/
4	0.002	0.000	6.601	http://www.huffingtonpost.com/...
5	0.002	0.000	6.601	http://www.entretiempo.com.co/
6	0.004	0.001	6.601	http://www.booking.com/
7	0.004	0.001	6.601	http://www.ebay.de/...
8	0.004	0.001	6.601	http://news.google.com/
9	0.004	0.001	6.601	http://thecpb.com/
10	0.005	0.001	6.601	http://www.solidverbal.com/...

```
The file TFIDF-TF-IDF-URI.txt contains the complete table
```

The results in Table 1 was facilitated by the function in Listing 4.

Listing 4: Calculate TF-IDF

```
#TF-IDF function
def calculateTFIDFforUris(filenamesOfHits, term):

    uri_TFIDF_TF_IDF_tuples = []
    if( len(filenamesOfHits) > 0 ):



        #TF-IDF = TF * IDF = occurrence in doc / words in doc * log2(total docs in
            corpus / docs with term)
        for f in filenamesOfHits:

            f = './ProcessedHtml/'+f
            occurrenceInDoc = countTheNumberOfOccurencesInFile(f, term)
            wordsInDoc = getWordCountFromInFile(f)

            hashFilename = f.split('-')
            hashFilename = hashFilename[1].split('.')[0]
            uri = [key for key, value in uriHashDictionary.iteritems() if value ==
                hashFilename][0]

            TF = ( float(occurrenceInDoc) / float(wordsInDoc) )
```

```
                IDF = logBase2( float(totalNumberOfDocumentInCorpus) / float(
                    documentWithTerm) )
                TFIDF = TF * IDF

                #print uri, ": ", occurrenceInDoc, wordsInDoc, TFIDF

                uriTuple = (uri, TFIDF, TF, IDF)
                uri_TFIDF_TF_IDF_tuples.append(uriTuple)

    return uri_TFIDF_TF_IDF_tuples
```

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://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 is only 10. 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).
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.

SOLUTION
With the use of this page rank: http://www.seocentro.com/tools/search-engines/pagerank.html applied upon the URIs in Table 1, the following Table 2 was obtained

```
The file PAGERANK-URI.txt contains the complete table
```

Since TF-IDF captures how important a term is in a document, the TF-IDF ranking in Problem 2. is a mechanical procedure which tries to reward documents according to their relatedness with respect to the term. The page rank comparison takes into account the backlink structure; so it seeks to answer the question: How is this page relevant/important with respect to other pages? In summary, the page rank comparison quantifies the relevance of the document with respect to other documents in the web, but the TF-IDF metric seeks to rank documents based on the level of their similarity to the term in question. Also the TF-IDF metric considers the term while the second does not. This means the first metric could rank an unpopular site very high if the the term has a high TF-IDF score, however, the second metric could rank the unpopular site low because of the poor quality of its backlinks.

Table 2: Rank Rank

ITEM	PAGE RANK	URI
1	0.8	http://news.google.com/
2	0.7	http://www.seattlepi.com/
3	0.7	http://www.philly.com/
4	0.7	http://www.booking.com/
5	0.0	http://www.ebay.de/...
6	0.0	http://market-ticker.org/...
7	0.0	http://thecpb.com/
8	0.0	http://www.solidverbal.com/
9	0.0	http://www.entretiempo.com.co/
10	0.0	http://www.huffingtonpost.com/...

Problem 4

Compute the Kendall Tau_b score for both lists (use "b" because there will likely be tie values in the rankings). Report both the Tau value and the "p" value.
See:

`http://stackoverflow.com/questions/2557863/measures-of-association-in-r-kendalls-tau-b-and-tau-c`
`http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient#Tau-b`
`http://en.wikipedia.org/wiki/Correlation_and_dependence`

SOLUTION[1], [2]
Kendall's tau b (accounts for ties) was used to measure the association between the ranks produced by TF-IDF and the Page Rank. The solution for this problem is outlined by the following steps:

1. **Convert the rank tables to there respetive rank vectors:**

   Page Rank Vector = {1, 2, 2, 2, 2, 3, 3, 3, 3, 3}, TF-IDF Rank Vector = {1, 2, 2, 2, 3, 3, 3, 3, 3, 4}

2. **Obtain the rank coefficient with R:** The following code in Listing 5. was used to calculate the tau coefficient and significance (z-value) as well as the p-value

   Listing 5: Calculate TF-IDF

```
#!/usr/bin/env Rscript

pageRankVector <- c(1, 2, 2, 2, 2, 3, 3, 3, 3, 3)
tdIdfRankVector <- c(1, 2, 2, 2, 3, 3, 3, 3, 3, 4)

cor.test(pageRankVector, tdIdfRankVector, method="kendall")
```

3. **Output:**tau = 0.8207, z = 2.7146, p-value = 0.0066.

   Due to the fact that half (5) of the URIs do not have rank data (rank 0), thus result in tied page ranks, Kendall's tau coefficient shows a high correlation; but this is mainly due to the absence of rank data and very small variations in TF-IDF values.

References

[1] Calculating Kendalls Tau Rank Correlation Coefficient. http://stamash.org/calculating-kendalls-tau-rank-correlation-coefficient/. Accessed: 2014-10-01.

[2] R tutorial. http://www.r-tutor.com/gpu-computing/correlation/kendall-tau-b/. Accessed: 2014-10-01.