Assignment 10
CS532, Web Science, Spring 2017
Old Dominion University, Computer Science Dept

Hussam Hallak
CS Master's Student
Prof: Dr. Nelson

Question 1:

Using the data from A8:
- Consider each row in the blog-term matrix as a 1000 dimension vector, corresponding to a blog.
- From chapter 8, replace numpredict.euclidean() with cosine as the distance metric. In other words, you'll be computing the cosine between vectors of 1000 dimensions.
- Use knnestimate() to compute the nearest neighbors for both:
http://f-measure.blogspot.com/ http://ws-dl.blogspot.com/
for k=1,2,5,10,20.

Answer:

In order to compute the nearest neighbors to both blogs, F-Measure and WSDL, using knnestimate() using the cosine between the vectors to compute the distance instead of the euclidean function, I created a python script, "knn.py". The script takes a blog term matrix as command line argument for its input, saved in a file "blogdata.txt" copied from Assignment 8. The functions used in the script were taken from "PCI" book with small modifications. I utilized "Scipy" module to calculate the cosine distance. I saved the output to a file and named it "knn.txt".

Listing 1: The content of knn.py

```
import sys
import math
from scipy import spatial

def readfile(filename):
   lines = [line for line in open(filename)]
   colnames = lines[0].strip().split('\t')[1:]
   rownames = []
   data = []
   for line in lines[1:]:
      p = line.strip().split('\t')
      rownames.append(p[0])
      data.append([float(x) for x in p[1:]])
   return (rownames, colnames, data)

def cosine(v1, v2):
   return spatial.distance.cosine(v1, v2)

def getdistances(data,vec1):
   distancelist=[]
   for i in range(len(data)):
      vec2=data[i]
      distancelist.append((cosine(vec1,vec2),i))
   distancelist.sort()
   return distancelist
```

```
def knnestimate(blogs, data,vec1,k):
   dlist=getdistances(data,vec1)
   for i in range(k):
      print i+1, '. ', blogs[dlist[i][1]], ': ', dlist[i][0]
   return dlist

def main():
   if len(sys.argv) < 2:
      print "Usage: python getpages.py <input_file>"
      print "e.g: python getpages.py blogdata.txt"
      exit()

   (blogs,words,data) = readfile(sys.argv[1])
   ks = [1,2,5,10,20]

   counter = 0
   for blog in blogs:
      if (blog == "F-Measure"):
         fm = blog
         blogdata = data[counter]
         blogs.pop(counter)
         data.pop(counter)
         counter = counter + 1

   print fm
   print "=============="

   for k in ks:
      print 'For K = ', k, ' :'
      print '******************'
      knnestimate(blogs, data, blogdata, k)

   blogs.append(fm)
   data.append(blogdata)

   print "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"

   counter = 0
   for blog in blogs:
      if (blog == "Web Science and Digital Libraries Research Group"):
         wsdl = blog
         blogdata = data[counter]
         blogs.pop(counter)
         data.pop(counter)
         counter = counter + 1

   print wsdl
   print "=============="
        for k in ks:
                print 'For K = ', k, ' :'
                print '******************'
                knnestimate(blogs, data, blogdata, k)


if __name__ == "__main__":
    main()
```

Listing 2: Running knn.py

```
root@ima-app:/var/www/Hussam/A10# python knn.py blogdata.txt > knn.txt
```

Listing 3: The Output (the content of knn.txt)

```
F-Measure
==============
For K = 1  :
******************
1 . music of the moment : 0.77468595644
For K = 2  :
******************
1 . music of the moment : 0.77468595644
2 . Floorshime Zipper Boots : 0.781375062422
For K = 5  :
******************
1 . music of the moment : 0.77468595644
2 . Floorshime Zipper Boots : 0.781375062422
3 . She May Be Naked : 0.791026707141
4 . Pithy Title Here : 0.813625170897
5 . Cuz Music Rocks : 0.836821512034
For K = 10  :
******************
1 . music of the moment : 0.77468595644
2 . Floorshime Zipper Boots : 0.781375062422
3 . She May Be Naked : 0.791026707141
4 . Pithy Title Here : 0.813625170897
5 . Cuz Music Rocks : 0.836821512034
6 . Web Science and Digital Libraries Research Group : 0.851859829515
7 . Playing Favorites : 0.855704178927
8 . Stephanie Veto Photography : 0.855861237123
9 . Steel City Rust : 0.858130158188
10 . Bonjour Girl : 0.861324950944
For K = 20  :
******************
1 . music of the moment : 0.77468595644
2 . Floorshime Zipper Boots : 0.781375062422
3 . She May Be Naked : 0.791026707141
4 . Pithy Title Here : 0.813625170897
5 . Cuz Music Rocks : 0.836821512034
6 . Web Science and Digital Libraries Research Group : 0.851859829515
7 . Playing Favorites : 0.855704178927
8 . Stephanie Veto Photography : 0.855861237123
9 . Steel City Rust : 0.858130158188
10 . Bonjour Girl : 0.861324950944
11 . Pirate's Log : 0.862401864906
12 . jaaackie. : 0.870612757625
13 . a duchess nonethelesss : 0.874878174209
14 . Spinitron Charts : 0.893449051029
15 . DaveCromwell Writes : 0.897499681747
16 . Stories From the City, Stories From the Sea : 0.900109513716
17 . Did Not Chart : 0.904191943265
18 . IoTube    :) : 0.906238553812
19 . Music-Drop Magazine : 0.906384561863
```

```
20 . MarkEOrtega's Journalism Portfolio : 0.906840082631
+++++++++++++++++++++++++++++++++++++++++++++++++++++++
Web Science and Digital Libraries Research Group
==============
For K = 1  :
******************
1 . She May Be Naked : 0.773789658939
For K = 2  :
******************
1 . She May Be Naked : 0.773789658939
2 . a duchess nonethelesss : 0.781415962198
For K = 5  :
******************
1 . She May Be Naked : 0.773789658939
2 . a duchess nonethelesss : 0.781415962198
3 . jaaackie. : 0.788127039725
4 . Stephanie Veto Photography : 0.791060931843
5 . Pirate's Log : 0.797067144815
For K = 10  :
******************
1 . She May Be Naked : 0.773789658939
2 . a duchess nonethelesss : 0.781415962198
3 . jaaackie. : 0.788127039725
4 . Stephanie Veto Photography : 0.791060931843
5 . Pirate's Log : 0.797067144815
6 . Steel City Rust : 0.809397458336
7 . Pithy Title Here : 0.813947898116
8 . Floorshime Zipper Boots : 0.826084927115
9 . Stonehill Sketchbook : 0.831709541798
10 . Did Not Chart : 0.839871846195
For K = 20  :
******************
1 . She May Be Naked : 0.773789658939
2 . a duchess nonethelesss : 0.781415962198
3 . jaaackie. : 0.788127039725
4 . Stephanie Veto Photography : 0.791060931843
5 . Pirate's Log : 0.797067144815
6 . Steel City Rust : 0.809397458336
7 . Pithy Title Here : 0.813947898116
8 . Floorshime Zipper Boots : 0.826084927115
9 . Stonehill Sketchbook : 0.831709541798
10 . Did Not Chart : 0.839871846195
11 . The Great Adventure 2016 : 0.845811331102
12 . Eli Jace | The Mind Is A Terrible Thing To Paste : 0.850558351687
13 . nonsense a la mode : 0.857965189424
14 . DaveCromwell Writes : 0.868706671357
15 . MarkEOrtega's Journalism Portfolio : 0.871719034509
16 . music of the moment : 0.877015300416
17 . @65 Sounding Booth : 0.879863520273
18 . MTJR RANTS & RAVES ON MUSIC : 0.887927965465
19 . MAGGOT CAVIAR : 0.888088609794
20 . holaOLA : 0.899028245979
```

Included Files:

knn.py, blogdata.txt, knn.txt

Question 2:

Rerun A9, Q2 but this time using LIBSVM. If you have n categories, you'll have to run it n times. For example, if you're classifying music and have the categories:

metal, electronic, ambient, folk, hip-hop, pop

you'll have to classify things as:

metal / not-metal electronic / not-electronic ambient / not-ambient

etc.

Use the 1000 term vectors describing each blog as the features, and your mannally assigned classifications as the true values. Use 10-fold cross-validation (as per slide 46, which shows 4-fold cross-validation) and report the percentage correct for each of your categories.

Answer:

I spent so much time trying to install LIBSVM in so many different ways on multiple computers with different operating systems, but was not successful. This part of the assignment is rather simple, but the technical difficulties are making it hard. I moved on to the next question.

Question 3:

Re-download the 1000 TimeMaps from A2, Q2. Create a graph where the x-axis represents the 1000 TimeMaps. If a TimeMap has "shrunk", it will have a negative value below the x-axis corresponding to the size difference between the two TimeMaps. If it has stayed the same, it will have a "0" value. If it has grown, the value will be positive and correspond to the increase in size between the two TimeMaps.

As always, upload all the TimeMap data. If the A2 github has the original TimeMaps, then you can just point to where they are in the report.

Answer:

Similar to my solution for Assignment 2. In fact, copying from there, I performed the following steps:

1. Downloading the TimeMaps for each of the target URIs using the ODU Memento Aggregator. I used the same python script from Assignment 2 to do that "download-timemapsjson.py". The script takes three command line arguments, an input file that has the links for which we want to download the TimeMaps "uniquelinks.txt", an output file to store the downloaded TimeMaps, "timemap.jsonx" where "x" is a number that identifies the timemap (e.g., 1, 2, 3, ...), and another output file, "uniquelinkswithmemes.txt", to store the unique links with mementos in.

Note: The number of output files named , "timemap.jsonx" may not be the same as the number of links in the input file "uniquelinks.txt". The reason is because not all links have mementos.

Listing 4: The content of downloadtimemapsjson.py

```
import sys
import urllib2
import json
if len(sys.argv) != 4:
    print "Usage: Python downloadtimemapsjson.py <input_file> <output_file1> <
        output_file2>"
    print "e.g: Python downloadtimemapsjson.py uniquelinks.txt timemap.json
        uniquelinkswithmemes.txt"
else:
    i = 1
    fh_input = open(sys.argv[1], 'r')
    fh_output = open(sys.argv[3], 'w')
    for line in fh_input:
        try:
            link = "http://memgator.cs.odu.edu/timemap/json/" + line
            response = urllib2.urlopen(link)
            content = json.load(response)
            output_file_name = sys.argv[2] + str(i)
            fh_json_output = open(output_file_name, "w")
            json.dump(content, fh_json_output)
            fh_output.write(line)
       i = i + 1
            fh_json_output.close()
        except:
            print "This link came with an error code:"
            print "http://memgator.cs.odu.edu/timemap/json/" + line
    fh_input.close()
    fh_output.close()
```

Listing 5: Running downloadtimemapsjson.py

```
root@ima-app:/var/www/Hussam/A10# python downloadtimemapsjson.py uniquelinks.txt
    timemap.json uniquelinkswithmemes.txt
```

Now I have 463 links with mementos out of the original 1107 unique links in the input file.

Listing 6: Links with mementos:

```
root@ima-app:/var/www/Hussam/A10# cat uniquelinks.txt | wc -l
1107
root@ima-app:/var/www/Hussam/A10# cat uniquelinkswithmemes.txt | wc -l
463
root@ima-app:/var/www/Hussam/A10# ls timemap.json* | wc -l
463
```

2. Parse the downloaded TimeMaps, JSON files, to find the number of mementos for each URI that has a TimeMap. I used the same Python script I wrote for Assignment 2, "parsejsontimemap.py", with some modification to do that. The script loops over all TimeMaps' files "timemap.jsonx" and parses them to find the number of mementos for each link and saves the link with its number of mementos in an output file named "linksnmemesnumA10.txt". The program also saves the number of mementos only in a file as output, "timemapreport.txt". This is the file that I will use to generate a histogram

in R in the next step. The file “linksnmemesnumA10.txt” will later be used to generate the desired graph in this assignment that compares the results from this assignment with the results from Assignment 2. I had to run the script “parsejsontimemap.py” on the timemaps from Assignment 2 to generate “linksnmemesnumA2.txt” as well.

Listing 7: The content of parsejsontimemap.py

```
import sys
import json
import os
if len(sys.argv) != 3:
   print "Usage: Python parsejsontimemap.py <input_file> <output_file>"
   print "e.g: Python parsejsontimemap.py timemap.json timemapreport.txt"
else:
   fh_output = open(sys.argv[2] , 'w')
   fh_output10 = open('linksnmemesnumA10.txt', 'w')
   for i in range(1,464):
      input_file_name = sys.argv[1] + str(i)
      data_file = open(input_file_name, 'r')
      data = json.load(data_file)
      N = len(data['mementos']['list'])
      fh_output10.write(data['original_uri'] + '\t' + str(N))
      N = str(N)
      fh_output.write(N)
      fh_output.write("\n")
      fh_output10.write('\n')
      data_file.close()
   fh_output.close()
   fh_output10.close()
```

Listing 8: Running parsejsontimemap.py

```
root@ima-app:/var/www/Hussam/A10# python parsejsontimemap.py timemap.json
    timemapreport.txt
root@ima-app:/var/www/Hussam/A10# cat timemapreport.txt | wc -l
463
```

3. Now we are ready to generate the histogram for URIs vs. the number of Mementos in R.

This line of code reads the data in the file and stores it in the vector timemaps_data.

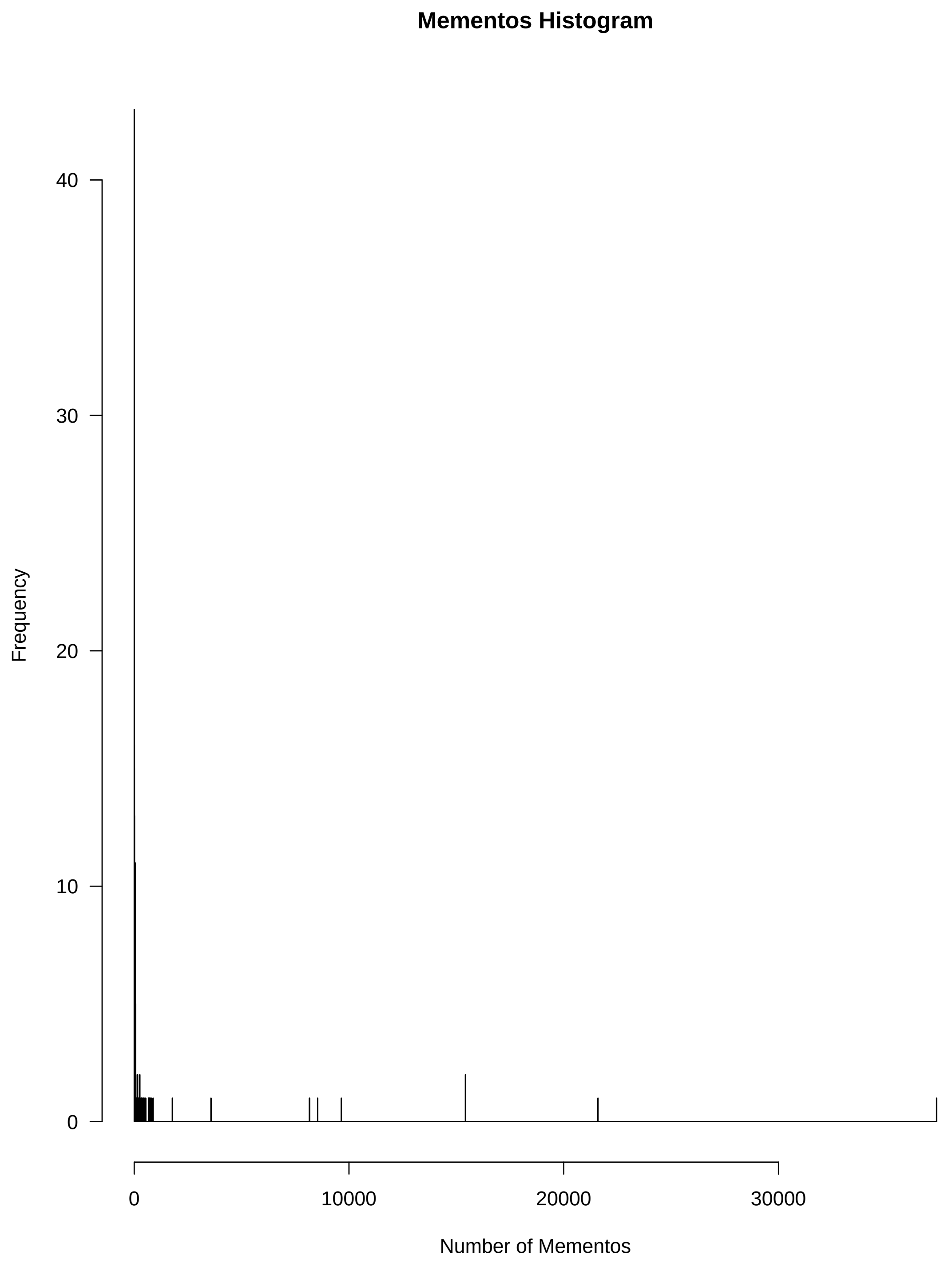
```
> timemaps_data <- read.table("C:/R/timemapreport.txt", header=F, sep="\t")
```

This line of code defines a variable max_num and assigns the maximum value in the vector timemaps_data to it.

```
> max_num <- max(timemaps_data)
```

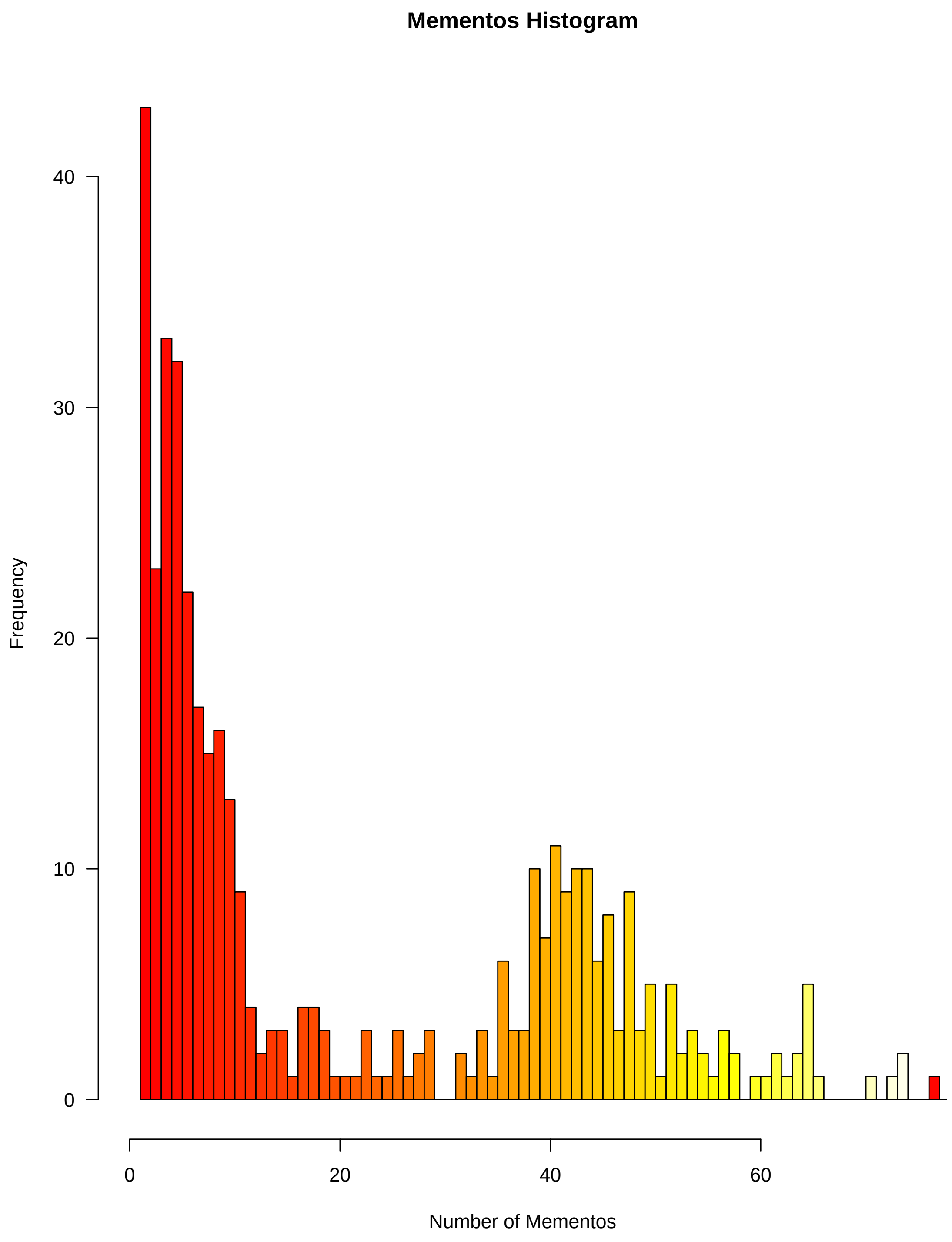
This line of code creates a histogram for timemaps_data with fire colors, sets bin = 1 so each number is in its own group, makes x axis range from 0 to max_num, disables right-closing of cell intervals, sets heading, changes the label of the x-axis, and makes y-axis labels horizontal

```
> hist(timemaps_data[,1], col=heat.colors(max_num), breaks=max_num, xlab="Number
    of Mementos", xlim=c(0,max_num), right=F, main="Mementos Histogram", las=1)
```


Mementos Histogram
Frequency
0
10
20
30
40
0
10000
20000
30000
Number of Mementos

We notice that our histogram did not provide a good representation of the results. I am going to zoom into the part where most of the data is and exclude the outliars in the data. Modifying the last line of code to this one will do just that.

```
> hist(timemaps_data[,1], col=heat.colors(max_num/500), breaks=max_num,
    xlab="Number of Mementos", xlim=c(0,max_num/500), right=F, main="Mementos
    Histogram", las=1)
```



4. In order to create a graph where the x-axis represents the 1000 TimeMaps comparing the results from Assignment 2 to this assignment, I wrote a python script, “comparetm.py”, to do that. The script takes two command line arguments, which contain the data, in this case “linksnmemesnumA2.txt” and “linksnmemesnumA10.txt”. It subtracts the number of mementos for each URI and saves the results to an output file named “graphtm.txt”. This file will be used to generate the desired graph in this question.

Listing 9: The content of comparetm.py

```
import sys

if len(sys.argv) != 3:
   print "Usage: python comparetm.py <input_file_1> <input_file_2>"
   print "e.g: python comparetm.py linksnmemesnumA2.txt linksnmemesnumA10.txt"
   exit()
A2 = open(sys.argv[1], 'r')
A10 = open(sys.argv[2], 'r')
output = open('graphtm.txt', 'w')

list2 = []
list10 = []
outlist = []

for line in A2:
   temp = []
   line = (line.strip()).split('\t')
   temp.append(line[0])
   temp.append(line[1])
   list2.append(temp)

for line in A10:
   temp = []
   line = (line.strip()).split('\t')
        temp.append(line[0])
        temp.append(line[1])
        list10.append(temp)

for key2,value2 in list2:
   for key10,value10 in list10:
      if (key2 == key10):
        diff = int(value2) - int(value10)
        outlist.append(diff)

for item in outlist:
   output.write(str(item))
   output.write('\n')

A2.close()
A10.close()
output.close()
```

Listing 10: Running comparetm.py

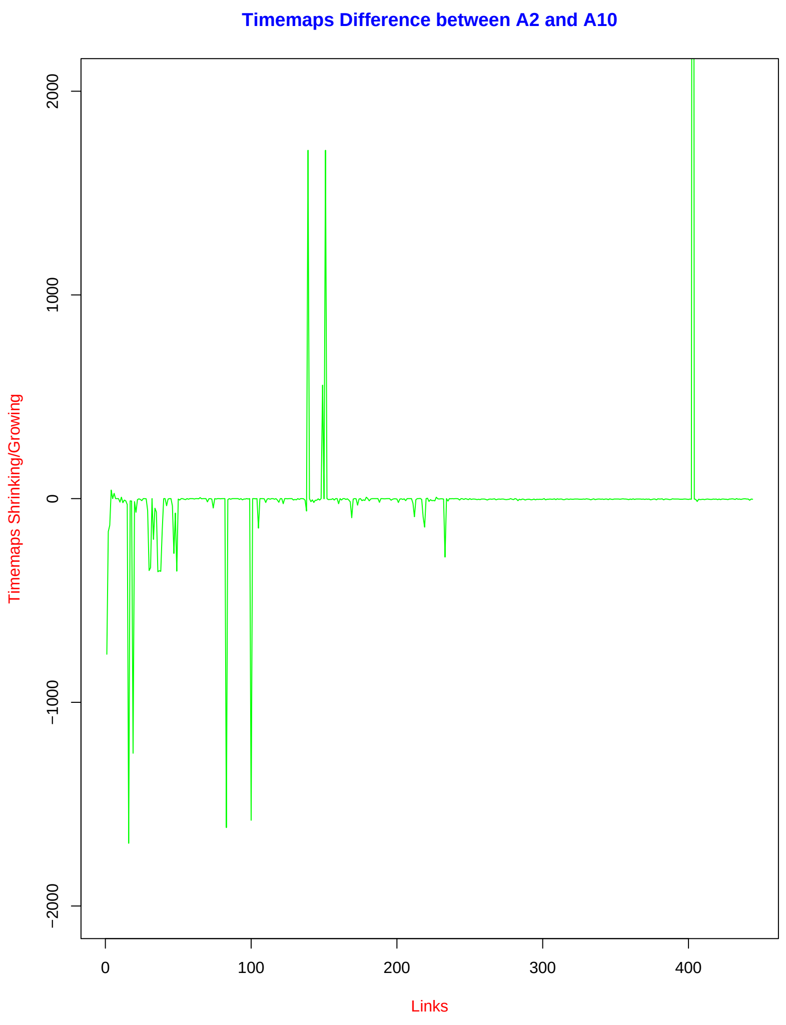
```
root@ima-app:/var/www/Hussam/A10# python comparetm.py linksnmemesnumA2.txt
    linksnmemesnumA10.txt
```

We are finally ready to create the desired graph in R. This R code does exactly that.

Listing 11: R code to create the graph

```
> data <- read.table("C:/R/graphtm.txt")
> plot(data$V1, type="l", main="Timemaps Difference between A2 and A10", col.main
    ="blue", col="green", xlab ="Links",ylab="Timemaps Shrinking/Growing", col.
    lab="red", ylim=c(-2000,2000))
```

Here is the desired graph that shows the difference in the number of mementos between Assignment 2 and this assignment.



Included Files:

downloadtimemapsjson.py, uniquelinks.txt, uniquelinkswithmemes.txt, parsejsontimemap.py, timemapreport.txt, linksnmemesnumA10.txt, linksnmemesnumA2.txt, comparetm.py, graphtm.txt, hist1.pdf, hist2.pdf, graphtm.pdf
Folder: timemaps that has all timemap.json* files,

Question 4:

Repeat A3, Q1. Compare the resulting text from February to the text you have now. Do all 1000 URIs still return a ”200 OK” as their final response (i.e., at the end of possible redirects)?

Create two graphs similar to that described in Q3, except this time the y-axis corresponds to difference in bytes (and not difference in TimeMap magnitudes). For the first graph, use the difference in the raw (unprocessed) results. For the second graph, use the difference in the processed (as per A3, Q1) results.

Of the URIs that still terminate in a ”200 OK” response, pick the top 3 most changed (processed) pairs of pages and use the Unix ”diff” command to explore the differences in the version pairs.

Answer:

Similar to what I did in Assignment 3, the approach is divided into two steps:

1. Take the unique links collected from Assignment 2, saved in “uniquelinks.txt” and download their content, then save it to text files, one file for each link. For that purpose I used md5 hashing to name the output files.

2. Remove (most) of the HTML markup from the content of the downloaded HTML pages.

I wrote a python script, “parselinks.py”, that will combine both steps. First it will download the raw HTML for each link and save it to a file, then it will strip HTML markup and save it to another file with the same name postfixed with “.processed”. The program will also create a text file named “map.txt” to map each link to its new file name.

Listing 12: The content of parselinks.py

```
import sys
import subprocess
from bs4 import *
import urllib2
import re

if len(sys.argv) != 2:
   print "Usage: Python parselinks.py <file_name>"
   print "e.g: Python parselinks.py uniquelinks.txt"
   exit()
def visible(element):
   if element.parent.name in ['style', 'script', '[document]', 'head', 'title']:
      return False
   elif re.match(r"[\s\r\n]+",unicode(element)):
      return False
   return True

fh_input = open(sys.argv[1], 'r')
fh_map = open("map.txt", 'w')
for link in fh_input:
   try:
      cmd = 'echo -n "' + link + '" | md5sum'
      output = subprocess.check_output(cmd, shell=True)
      output_file_name = output[0:32]
      html_page = urllib2.urlopen(link)
```

```
        soup = BeautifulSoup(html_page, "html.parser")
        texts = soup.findAll(text=True)
        output_texts = str(texts)
        fh_output = open(output_file_name, 'w')
        fh_output.write(output_texts)
        fh_output.close()
        output_file_name_processed = output_file_name + '.processed'
        fh_output_processed = open(output_file_name_processed, 'w')
        visible_texts = filter(visible, texts)
        output_texts = str (visible_texts)
        fh_output_processed.write(output_texts)
        fh_output_processed.close()
        fh_map.write(link)
        fh_map.write('\t')
        fh_map.write(output_file_name_processed)
        fh_map.write('\n')
    except:
        print "This link generated an error code:"
        print link
fh_input.close()
fh_map.close()
```

Listing 13: Running parselinks.py to download raw HTML and visible text from URIs

```
root@ima-app:/var/www/Hussam/A10/Q4# python parselinks.py uniquelinks.txt
root@ima-app:/var/www/Hussam/A10/Q4# ls | wc -w
799
```

The program created 796 output files, two files for each link. This means that the content of 398 URIs was downloaded and processed successfully. The rest of the links generated errors. That could be due to different reasons including bad HTML markup, the page no longer exists on the web, or other reasons. The interesting part is that the number of URIs that were downloaded and processed successfully at the time I did Assignment 3 was 397 which is one URI less than this time, Assignment 10. The only explanation I can think of is that one or more URIs were not available at the time I did Assignment 3 and now they are, or bad HTML in one of the URIs that has been fixed since I did Assignment 3. Most of the links I collected belong to famous news channels which explains why most, if not all, of the ones that returned "200 OK" in Assignment 3 returned "200 OK" now.

The next step is creating the graphs. I wrote a python script, “comparefiles.py”, that will compare the sizes of files that belong to each URI. The program does that for both the raw and the processed files. The script accepts one command line argument, that is “map.txt” in our case to get the file names, then it finds the sizes for both old and new files in both cases raw and processed. The program then subtracts the sizes and saves the difference between the raw files to “rawdiff.txt” and the difference between the processed files to “processeddiff.txt”. These two files will be used to generate our graphs in R.

I copied the raw and processed files from Assignment 3 to the folders “rawold” and “processedold” respectively. I also copied the newly created raw and processed files, from this assignment, to the folders “rawnew” and “processednew” respectively. This will make things more organized for the script “comparefiles.py”.

Listing 14: The content of comparefiles.py

```
import sys
import os
import re

if len(sys.argv) != 2:
   print "Usage: python comparefiles.py <map_file>"
   print "e.g: python comparefiles.py map.txt"
   exit()

map_file = open(sys.argv[1] ,"r")
raw_output = open("rawdiff.txt" ,"w")
processed_output = open("processeddiff.txt", "w")

raw_file_names = []

counter = 0
for line in map_file:
   if (counter % 2 == 1):
      line = line.strip()
      line = re.sub('\.processed$', '', line)
      raw_file_names.append(line)
   counter = counter + 1

for name in raw_file_names:
   try:
      rawold = os.path.getsize("rawold/" + name)
      rawnew = os.path.getsize("rawnew/" + name)
      diff = rawold - rawnew
      raw_output.write(str(diff) + "\n")
      processedold = os.path.getsize("processedold/" + name + ".processed")
      processednew = os.path.getsize("processednew/" + name + ".processed")
      diff = processedold - processednew
      processed_output.write(str(diff) + "\n")
   except:
      print 'File does not exist because the URI did not return "200 OK"'

map_file.close()
raw_output.close()
processed_output.close()
```

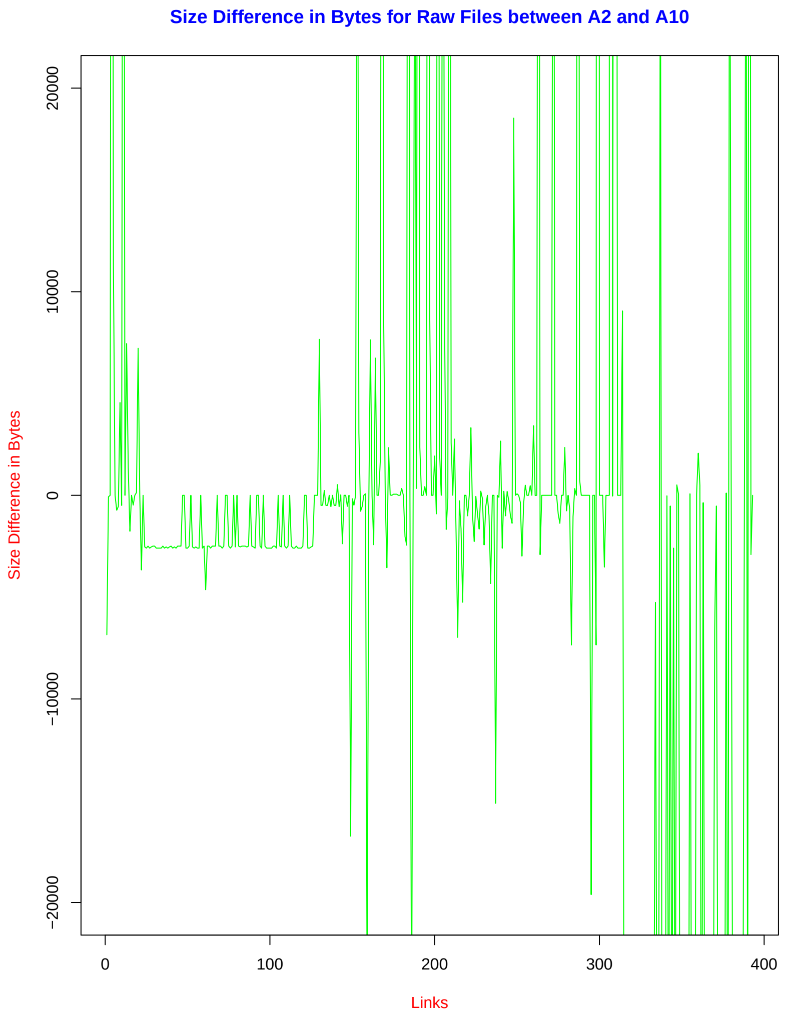
Listing 15: Running comparefiles.py to create the files rawdiff.txt and processeddiff.txt to generate the graphs in R.

```
root@ima-app:/var/www/Hussam/A10/Q4# python comparefiles map.txt
```

We are now ready to create the desired graph for raw files in R. This R code does exactly that.

Listing 16: R code to create the graph

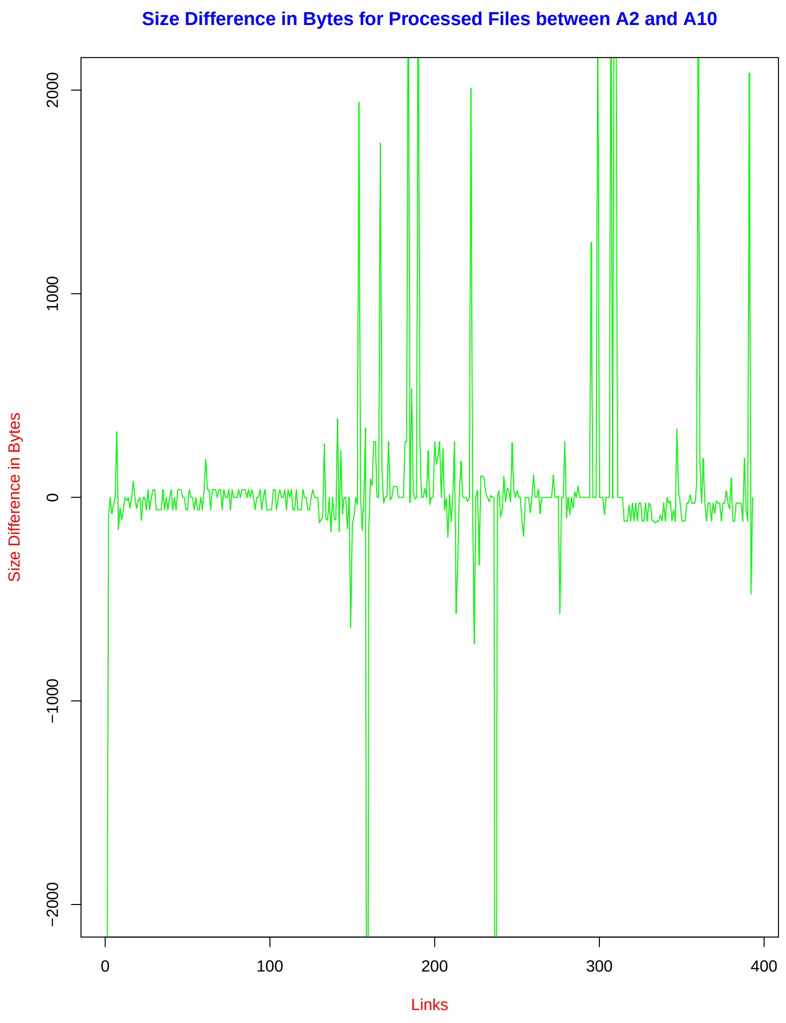
```
> data <- read.table("C:/R/rawdiff.txt")
> plot(data$V1, type="l", main="Size Difference in Bytes for Raw Files between A2
     and A10", col.main="blue", col="green", xlab ="Links",ylab="Size Difference
    in Bytes", col.lab="red", ylim=c(-20000,20000))
```



In a similar fashion, I created the desired graph for processed files in R.

Listing 17: R code to create the graph

```
> data <- read.table("C:/R/processeddiff.txt")
> plot(data$V1, type="l", main="Size Difference in Bytes for Processed Files
    between A2 and A10", col.main="blue", col="green", xlab ="Links",ylab="Size
    Difference in Bytes", col.lab="red", ylim=c(-2000,2000))
```



Now let's pick the top 3 most changed (processed) pairs of pages per the requirements in the question. I created a small php script, "top3.php", to help me do that. The script takes a command line argument as input, it is the file "map.txt", which maps each link to the name of its processed file. It extracts all the processed file names and computes the size difference in bytes between each pair of processed files from Assignment 3 and the files created in this assignment. It saves these values in an associative array with the file name as the key, and the difference in bytes as the value. Then it sorts the array by the value maintaining index association, keeping the (key,value) together. Finally, the sorted array that has the file names along with the difference in bytes is sent as an output to a file named "sizedifference.txt".

Listing 18: The content of top3.php

```
<?php
if (count($argv) != 2) {
   print "Usage: php top3.php <map_file>\n";
        print "e.g: php top3.php map.txt\n";
        exit();
}

$diff = array();
```

```
$map_file = fopen($argv[1], "r") or die("Unable to open map file!");

$counter = 0;
while(!feof($map_file)) {
   $raw_file_name = trim(fgets($map_file));
   if ($counter % 2 == 1) {
      $diff[$raw_file_name] = 0;
   }
   $counter++;
}

fclose($map_file);

foreach ($diff as $key => $value) {
   try {
      $old_file_path = "processedold/".$key;
      $old_file_size = filesize($old_file_path);
      $new_file_path = "processednew/".$key;
      $new_file_size = filesize($new_file_path);
      $difference = abs($old_file_size - $new_file_size);
      $diff[$key] = $difference;
   } catch (Exception $e) {
      print $e->getMessage();
   }
}

asort($diff);

$output_file = fopen("sizedifference.txt", "w") or die("Unable to create output
    file");
foreach ($diff as $key => $value) {
   $output = $key." --> ".$value."\n";
   fwrite($output_file, $output);
}
fclose($output_file);
?>
```

Listing 19: Running top3.php to sort the processed files based on the difference in size

```
root@ima-app:/var/www/Hussam/A10/Q4# php top3.php map.txt
```

In order to pick top 3 most changed (processed) pairs of pages, I executed the following command to show me the last 3 lines in the file "sizedifference.txt".

Listing 20: Using tail on sizedifference.txt to pick top 3 most changed (processed) pairs of pages.

```
root@ima-app:/var/www/Hussam/A10/Q4# tail sizedifference.txt -n 3
79c8e70bf1211bc717dca09b66a27aef.processed --> 11558
29c1253c338efaad05365a28c1b25495.processed --> 17381
68046d7c24bb2dacd6dc28333caa0625.processed --> 25594
root@ima-app:/var/www/Hussam/A10/Q4#
```

Included Files:

parselinks.py, uniquelinks.txt, map.txt, comparefiles.py, rawdiff.txt, processeddiff.txt. top3.php, sizedifference.txt, rawdiff.pdf, processeddiff.pdf
Folder “rawold” contains all raw HTML files from Assignment 3
Folder “processedold” contains all visible text in HTML files from Assignment 3
Folder “rawnew” contains all raw HTML files from this assignment
Folder “processednew” contains all visible text in HTML files from this assignment