

Web Science: Homework 7

Clustering

Dr. Michele Weigle

Harveen Kaur

Thursday, April 9, 2020

Contents

Problem 1	3
Problem 2	5
Problem 3	6
Problem 4	8
Problem 5	8

Problem 1

Generate a list of 100 popular accounts on Twitter. The accounts must be verified, have $> 10,000$ followers, and have > 5000 tweets. See GET users/lookup and User object for details on obtaining this information for a set of accounts. You may also generate this information manually by visiting individual account pages. For example:

- <https://twitter.com/weiglemc> - not verified, 414 followers, 2189 tweets - don't include
- <https://twitter.com/WNBA> - verified (blue checkmark), 615,000+ followers, 69,000+ tweets - could include

Save the list of accounts (screen_names) in a text file (one per line) and upload to your GitHub repo.

Download 200 tweets from the 100 accounts. See GET statuses/user_timeline for details. Note that you may receive fewer than 200 tweets in a single API call due to deleted or protected tweets. It's OK as long as you get somewhere close to 200 tweets for each account.

Save the responses received from the 100 accounts to your GitHub repo. It can all be in a single file or a separate file for each account. Since this is an intermediate file, the format is up to you. Find 3 users who are closest to you in terms of age, gender, and occupation.

This user is the substitute you.

SOLUTION :

I collected the information by manually visiting individual account pages. I followed the criteria of the account being verified, having $> 10,000$ followers and have > 5000 tweets. The list of twitter handles can be found in the file **twitterHandler**. I used 112 handles to collect data.

Listing 1: Assignment7_1.py

```
1.YouTube
2.jimmyfallon
3.Google
4.amazon
5 5.amazonmusic
6.Uber
7.lyft
8.tim_cook
9.BillGates
10 10.elonmusk
11.richardbranson
12.rupertmurdoch
13.levie
```

Above given are a few twitter handles that I used. The file **twitterHandler** is available in the GitHub repo.

```
import twitter
import preprocessor as p
import collections
5 import re
```

```

# For removing URLs from the tweets
url_regex = 'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\\(\), ]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'

# for removing extra emojis
remove_list = [" ", ",", " ", " ", " ", " ", " "]

res_dict = collections.defaultdict(lambda: collections.defaultdict(int))

#for removing URLs, Emojis, Smileys and Solo Numbers.
p.set_options(p.OPT.URL, p.OPT.EMOJI, p.OPT.SMILEY, p.OPT.NUMBER)

#user_list = ["sundarpichai", "Android", "elonmusk"]
user_list = open('twitterHandler.txt', 'r').read().split('\n')

'''
Create Twitter Instance. All the fields can be collected from the developer site of
Twitter
I also used the tweet mode as extended to get all the tweet text data in a response
header.
'''
api = twitter.Api(consumer_key='#####',
                  consumer_secret='#####',
                  access_token_key='#####',
                  access_token_secret='#####',
                  sleep_on_rate_limit=True, tweet_mode="extended"
                  )

for user in user_list:
    statuses = api.GetUserTimeline(screen_name=user, count=200)
    texts = [ x.full_text for x in statuses ]
    text = " ".join(texts)
    ntext = re.sub(url_regex, "", text)
    n2text = p.clean(ntext)
    n3text = n2text
    for x in remove_list:
        n3text = n3text.replace(x, "")
    # n4list = re.split(r'[;,\s()." ]', n3text)
    # Split words by all non-alpha characters and remove unecessary spaces.
    words = re.compile(r'[^A-Z^a-z]+').split(n3text)
    # Convert to lowercase
    n5list = [word.lower() for word in words if word != '']
    res_dict[user].update(dict(collections.Counter(n5list)))

key_list = set()
for user in list(res_dict.keys()):
    key_list.update(set(res_dict[user].keys()))

key_list = list(key_list)

with open("tweetdata.txt", "w") as f:
    f.write("Username")

```

```

60     for k in key_list:
        f.write(f"\t{ k }")
    f.write("\n")
    for user in user_list:
        f.write(f"{ user }")
65     for k in key_list:
        f.write(f"\t{ res_dict[user][k] }")
    f.write("\n")

```

Above given code generates the account term matrix.

Problem 2

Generate an account-term matrix from the accounts' tweets.

Using the responses from Q1, extract the text from each tweet to generate terms. Remove any URIs in the tweets, but keep regular text and hashtags as terms. Limit the number of terms to the most "popular" (i.e., frequent) 1000 terms, this is after the criteria on p. 32 (chapter 3 PCI book) (slide 11 - Week 10) has been satisfied.

Save the terms for each account in a file and upload to your GitHub repo. It can all be in a single file or a separate file for each account. Since this is an intermediate file, the format is up to you.

In the account-term matrix, the account screen_name is the account identifier and should be start each row of the matrix. Use the (max 1000) terms for the columns of the matrix. The values are the frequency of occurrence. Essentially you are replicating the format of the "blogdata.txt" file included with the PCI book code.

Save the matrix in a text file (either tab-separated like blogdata.txt or comma-separated) and upload to your GitHub repo.

SOLUTION

Account term matrix was created using the code given in listing7_1. The account term matrix was generated using the tweets from 112 accounts(verified).

Listing 2: Assignment7_2.py

```

with open("tweetdata.txt", "w") as f:
    f.write("Username")
    for k in key_list:
        f.write(f"\t{ k }")
5    f.write("\n")
    for user in user_list:
        f.write(f"{ user }")
        for k in key_list:
            f.write(f"\t{ res_dict[user][k] }")
10    f.write("\n")

```

The result is stored in the file tweetdata.txt and is available on GitHub.

Problem 3

Create an ASCII dendrogram and a JPEG dendrogram that uses hierarchical clustering to cluster the most similar accounts (see slides 21 - 23 - Week 10). Include the JPEG in your report and upload the ASCII file to GitHub (it will be too unwieldy for inclusion in the report).

SOLUTION

The python script denodo.py calls the methods from clusters.

Using the file `tweetdata.txt` we get the ASCII dendrogram and the JPEG dendrogram.

Reference: Slide 21 and 23.

Listing 3: Assignment7_3.py

```
import clusters
import sys

5 def createDendrogram():
    Username, colnames, data = clusters.readfile('tweetdata.txt')
    cluster = clusters.hcluster(data)
    clusters.drawdendrogram(cluster, Username, jpeg='Dendrogram.jpg')
    f = open("ASCII.txt", 'w')
10 sys.stdout = f
    clusters.printclust(cluster, labels=Username)
    f.close()
    sys.stderr.close()

15 if __name__ == "__main__":
    createDendrogram()
```

The diagram given below displays the generated JPEG dendrogram. The ASCII dendrogram is available on GitHub. I have not uploaded it here as it is too unwieldy to be included in the report (as mentioned in the problem).

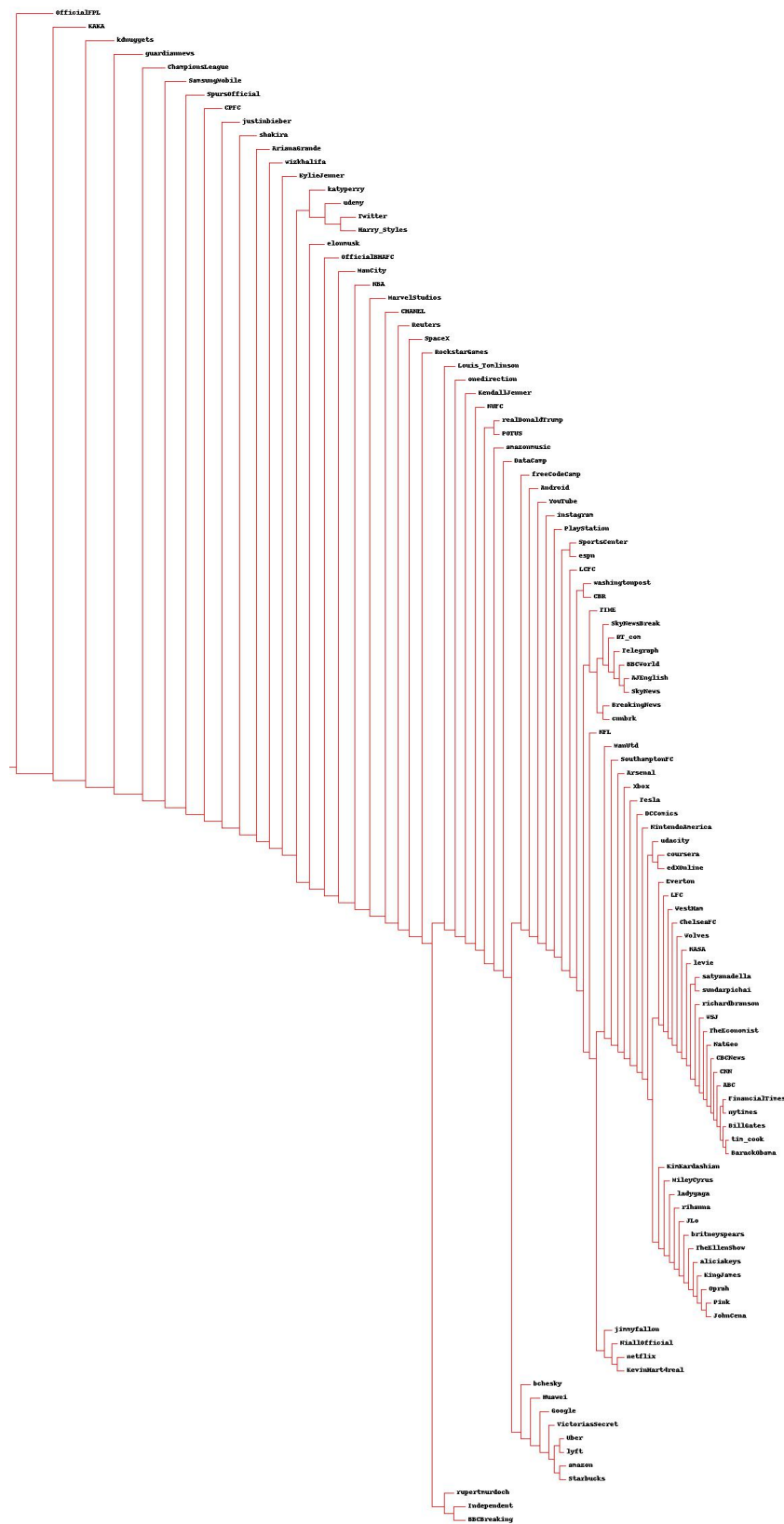


Figure 1: JPEG dendrogram

Problem 4

Cluster the accounts using k-Means, using k=5,10,20 (see slide 37 - Week 10). Print the accounts in each cluster, for each value of k. How many iterations were required for each value of k?

SOLUTION

With the help of the code given in the slide 37 I coded the script kmeanclustering.py.

Listing 4: Assignment7_4.py

```
import clusters
def kMean():
    kMeanValues = [5, 10, 20]
    Username, colnames, data = clusters.readfile('tweetdata.txt')
5     for i in kMeanValues:

        kclust, itercount = clusters.kcluster(data, k=i)
        print(kclust)
        f = open("kclust_%d.txt" % i, 'w')
10     f.write("Total Number Of Iterations: %d \n" % itercount)
        print(len(kclust))
        clusterCount = 1
        for cluster in kclust:
            i=1
15         f.write("---\n")
            f.write("Cluster %d \n" % clusterCount)
            for tweetID in cluster:
                f.write(str(i)+"\t"+Username[tweetID] + "\n")
                i+=1
20         f.write("\n")
            clusterCount+=1
if __name__ == "__main__":
    kMean()
```

The results have been uploaded as text files **kclust_5**, **kclust_10** and **kclust_20** respectively. For the clusters it took 6 iterations, 5 iterations and 5 iterations respectively.

Problem 5

Use MDS to create a JPEG of the accounts (see slide 50 - Week 10). Include the JPEG in your report. How many iterations were required?

SOLUTION

Created the following python script.

Listing 5: Assignment7_5.py

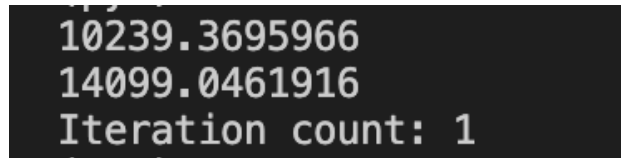
```
import clusters

Username, words, data = clusters.readfile('tweetdata.txt')
```



```
5 | coords, itercount = clusters.scaledown(data)
   | clusters.draw2d(coords, labels=Username, jpeg='mds.jpg')
   | print ('Iteration count: %d' % itercount)
```

To generate the result 1 iteration was required.

A terminal window with a black background and white text. The text shows the results of an MDS iteration: '10239.3695966' on the first line, '14099.0461916' on the second line, and 'Iteration count: 1' on the third line.

```
10239.3695966
14099.0461916
Iteration count: 1
```

Figure 2: MDS iteration

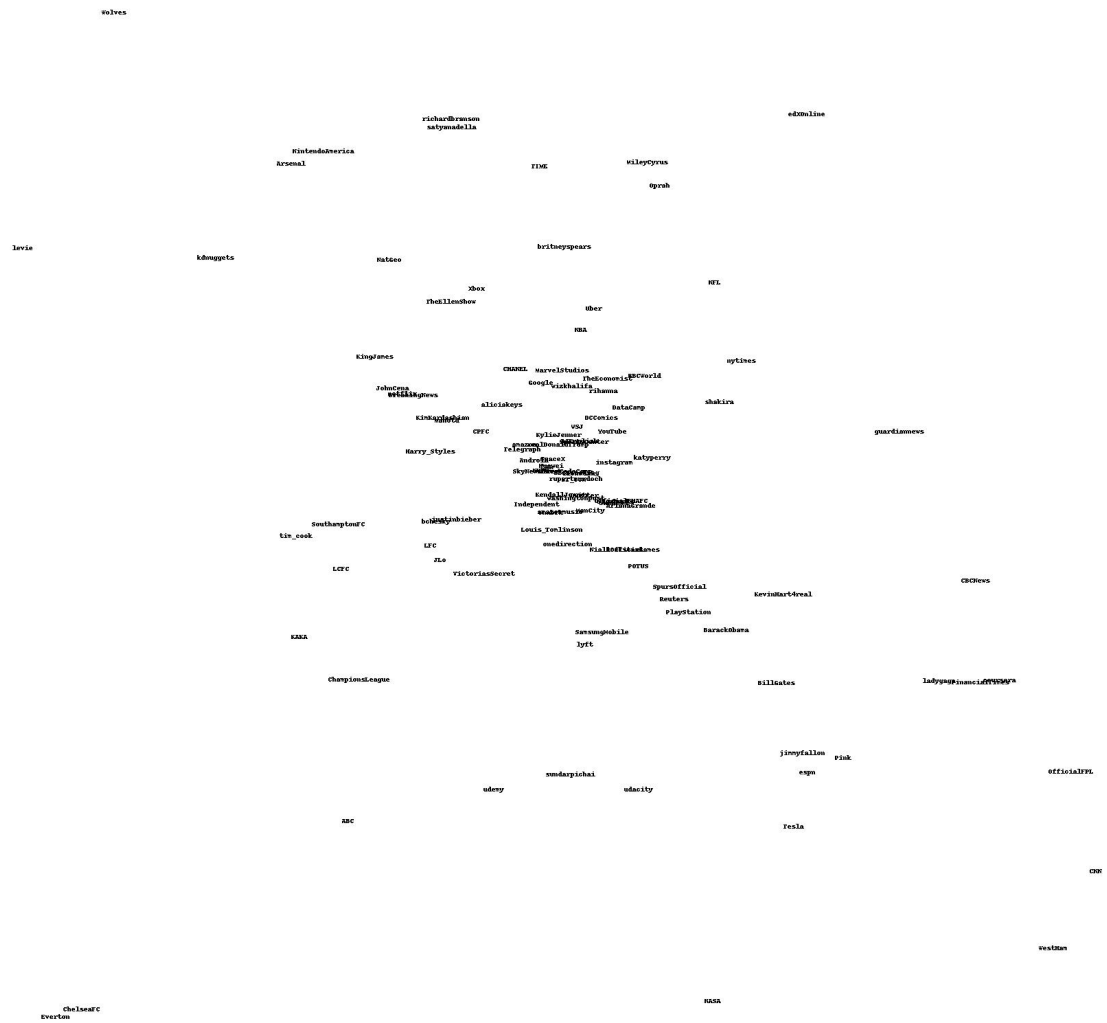


Figure 3: MDS