HW8 - Clustering

Adeniran Adeniyi
Sunday, April 18, 2021 by 11:59pm

Q1

Generate a list of 100 popular accounts on Twitter. The accounts must be verified, have < 10,000 followers, and have > 5000 tweets.

For example: https://twitter.com/weiglemc - not verified, 457 followers, 2554 tweets - don't include https://twitter.com/WNBA - verified (blue checkmark), 672,800+ followers, 77,900+ tweets - could include
See GET users/lookup, Tweepy's API.lookup_users(), 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 Because we're trying to cluster the accounts based on the text in their tweets, you should choose several sets of accounts that are similar (political, tech, sports, etc.) to see if they'll get clustered together later. Save the list of accounts (screen_names), one per line, in a text file named accounts.txt and upload to your GitHub repo.

How did you choose to collect the accounts?

What topics/categories do the accounts belong to?

You don't need to specify a grouping for each account, but what general topics/categories will you expect to be revealed by the clustering?

Answer

```
# tweetparser.py
# MCW - 4/1/2021
import json
import tweepy
import pandas as pd
import numpy as np

def confirm_usermetsrequiremetn(twAuth,ids):
    #Find followers that are in this category and them to the list
    status= False
    try:
        a = twAuth.get_user(id = ids)
        if a.statuses_count >= 5000 and a.followers_count >= 10000 and a.verified == True:
            status = True
        else:
            status = False
    except:
        print("There was an error")
    return status
def setup_api(filename):
    ’’’
    filename: file where Twitter API keys are stored
    returns Twitter API object to pass into parse()
    ’’’

    # load Twitter API keys from a file so they’re not hard-coded
    with open(filename, "r") as secretsFile:
        secrets = json.load(secretsFile)

    # set up Twitter API with OAuth2 procedure
    consumer_key = secrets[’consumer_key’]
    consumer_secret = secrets[’consumer_secret’]
    try:
        auth = tweepy.AppAuthHandler(consumer_key, consumer_secret)
        api = tweepy.API(auth,wait_on_rate_limit=True, wait_on_rate_limit_notify=True)
        return api
    except tweepy.TweepError as e:
        print ("Tweepy Error: %s" % str(e))

def parse(api, screen_name, num_tweets=200):
    ’’’
    api: Twitter API object, use setup_api() to create
    screen_name: Twitter screen_name
    num_tweets: Number of tweets to request (default: 200)
    returns dict with {’screen_name’: screen_name, ’tweets’: [tweet1, tweet2, ...]}
    ’’’

    tweet_data = []
    try:
        for tweet in tweepy.Cursor(api.user_timeline, screen_name=screen_name, count=num_tweets, lang=’en’,
                                tweet_mode=’extended’, exclude_replies=True, include_rts=False).items():
            tweet_data.append(tweet.full_text)
    except tweepy.TweepError as e:
        print ("Tweepy Error: %s" % str(e))

    account_data = {'screen_name': screen_name, 'tweets': tweet_data}
    return account_data
```

Listing 1: tweetparser.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 17 13:02:33 2021

@author: aadeniran
"""
import tweepy
from tweetparser import setup_api, confirm_usermetsrequiremetn
import pandas as pd
import numpy as np
def  per_account(types,secret =setup_api("secrets.json") ):

    #Get twitter friends screen_name of the parse account, as one arrayList
    public_tweets = tweepy.Cursor(secret.friends_ids, id = types)
    count= 1
    resultList =[] # store the final result of screen names
    for user in public_tweets.pages():
        #print(user)
        #print(user.dtypes())
        #Transverse the list of followers ids
        for i in user:
            #check if the friends meet the requirement
            #print(confirm_usermetsrequiremetn(secret,i))
            if(confirm_usermetsrequiremetn(secret,i) ==True):
                if( i not in resultList):

                    #instead of the user id get the user name
                    user_sc = secret.get_user(i)
                    print("{}:{}".format(count,user_sc.screen_name))
                    resultList.append(user_sc.screen_name)
                    count += 1
            #Get maximum 30 file accounts screen_names
            if(count >= 30):
                break
    print(resultList)

    #build text file and save file in Q1 directory
    filename =  'Q1/' + types + '.txt'
    with open(filename, 'w') as filehandle:
        for listitem in resultList:
            filehandle.write('%s\n' % listitem)
    return resultList
"""
Tech account = @WIRED
Sport account = @WNBA
politics = @POTUS45
music = @future_of_music
"""
#Get all screen_names with that fulfils 10,000 followers and have 5000 tweets and verified
per_account("WIRED")
per_account("WNBA")
per_account("POTUS45")
per_account("future_of_music")

"""
Bring all result from the files in to One text file
accounts.txt
"""
#get the list of the created files
column_name= ["User_screen_names"]
final = pd.DataFrame(columns= column_name)
fileList = ["Q1/WIRED.txt","Q1/WNBA.txt","Q1/POTUS45.txt","Q1/future_of_music.txt"]
df = pd.DataFrame()


for t in fileList:
    frame = pd.read_csv(t,header=None)
    frame.columns = column_name
    for ind in frame.index:
        final.loc[len(final)] =[frame['User_screen_names'][ind]]

#drop duplicated values keep just one of them
# dropping duplicate values
final.User_screen_names.drop_duplicates(inplace=True)

#confirm that there are all unique values there are 113
print(final.User_screen_names.nunique())

# Number of rows to drop
n = 14

# Dropping last n rows using drop
final.drop(final.tail(n).index,
        inplace = True)
#print(final.User_screen_names.nunique())
```

```
numpy_array = final.to_numpy()
#print as a text file to in the same directory for future use
np.savetxt(r'accounts.txt', numpy_array,fmt="%s")
```

Listing 2: gatherId.py

Discussion

In gathering users screen names I did took into consideration of the following:

- I collected the screen names going through 4 major popular twitter accounts WIRED, WNBA, POTUS45, and future_of_music

```
#Get all screen_names with that fulfils 10,000 followers and have 5000 tweets and verified
per_account("WIRED")
per_account("WNBA")
per_account("POTUS45")
per_account("future_of_music")
```

Listing 3: A snap shot of gatherIds.py show these accounts doing screen_name account extraction

following list, I extracted accounts the met the requirement of being verified, have > 10,000 followers, and have > 5000 tweets

- The function confirm_usermetsrequiremetn() in tweetparser.py on line 8 to 19 check to make sure the accounts fulfilled these requirement

```
 8 def confirm_usermetsrequiremetn(twAuth,ids):
 9     #Find followers that are in this category and them to the list
10     status= False
11     try:
12         a = twAuth.get_user(id = ids)
13         if a.statuses_count >= 5000 and a.followers_count >= 10000 and a.verified == True:
14             status = True
15         else:
16             status = False
17     except:
18         print("There was an error")
19     return status
```

Listing 4: A snap shot of tweetparser.py that check screen_name account meets requirements

Imports of the file was made on line 8 of gatherIds.py

```
7 import tweepy
8 from tweetparser import setup_api, confirm_usermetsrequiremetn
9 import pandas as pd
```

```
import numpy as np
```

Listing 5: A snap shot of gatherIds.py that shows the needed imports

- Function per_account() produce a text files (names based on the arguement supplied) that gets stored in Q1 folder
- I then merge all the produced files from each of the accounts to form the main accounts.txt file. Seen in line 55 - 85

```
55 """
56 Bring all result from the files in to One text file
57 accounts.txt
58 """
59 #get the list of the created files
60 column_name= ["User_screen_names"]
61 final = pd.DataFrame(columns= column_name)
62 fileList = ["Q1/WIRED.txt","Q1/WNBA.txt","Q1/POTUS45.txt","Q1/
       future_of_music.txt"]
63 df = pd.DataFrame()
64 
65 
66 for t in fileList:
67     frame = pd.read_csv(t,header=None)
68     frame.columns = column_name
69     for ind in frame.index:
70         final.loc[len(final)] =[frame['User_screen_names'][ind]]
71 
72 #drop duplicated values keep just one of them
73 # dropping duplicate values
74 final.User_screen_names.drop_duplicates(inplace=True)
75 
76 #confirm that there are all unique values there are 113
77 print(final.User_screen_names.nunique())
78 
79 # Number of rows to drop
80 n = 14
81 
82 # Dropping last n rows using drop
83 final.drop(final.tail(n).index,
84         inplace = True)
85 #print(final.User_screen_names.nunique())
```

Listing 6: A snap shot of gatherIds.py shows the merging process of the files generated by each account

Q2

Create Account-Term Matrix

Answer

```python
# generatetweetvector.py
# Based on generatefeedvector.py from
# https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter3/generatefeedvector.py

from tweetparser import setup_api, parse
import re

def getwordcounts(api, screen_name):
    """
    api: Twitter API object
    screen_name: Twitter screen_name
    returns screen_name and dictionary of word counts for a Twitter account
    """

    # Parse the Twitter feed
    d = parse(api, screen_name)
    wc = {}

    # Loop over all the entries
    for tweet in d['tweets']:

        # Extract a list of words
        words = getwords(tweet)
        for word in words:
            wc.setdefault(word, 0)
            wc[word] += 1

    return (d['screen_name'], wc)

def getwords(tweet):
    """
    returns lowercase list of words after filtering
    """

    # Remove URLs
    text = re.compile(r'(http://|https://|www\.)([^ \'\"]*)').sub('', tweet)
    "
    # Remove other screen names (start with @)
    text = re.compile(r'(@\w+)').sub('', text)

    # Split words by all non-alpha characters
    words = re.compile(r'[^A-Z^a-z]+').split(text)

    # Filter for words between 3-15 characters, convert to lowercase, and return as a list
    return [word.lower() for word in words if (len(word) >= 3 and len(word) <= 15)]

# MAIN CODE STARTS HERE

# set up Twitter API object
api = setup_api("secrets.json")

apcount = {}      # number of accounts each word appears in
wordcounts = {}   # words and frequency in each account
sumcounts = {}    # words and frequency over all accounts (to determine top 1000)

# list of screen names should be in 'accounts.txt', one per line
accountlist = [line.strip() for line in open('accounts.txt')]
#print(accountlist)
#print(len(accountlist))

for screen_name in accountlist:
    try:
        # get tweets, filter and count words
        (user, wc) = getwordcounts(api, screen_name)
        wordcounts[user] = wc

        # count number of accounts each term appears in
        for (word, count) in wc.items():
            apcount.setdefault(word, 0)
            sumcounts.setdefault(word, 0)
            if count > 1:
                apcount[word] += 1        # counting accounts with the word
                sumcounts[word] += count  # summing total counts for the word
    except:
        print ('Failed to parse account %s' % screen_name)


"""
```

```
Did some testing to figure out the datatype
and sample output for one acccount
Spyder ide  gives flexibility of running each variable without running the whole code after your ran the whole code at least once
"""
#print("Done Counting words")
#print(type(sumcounts))
#print(sumcounts.keys())
#for elem in listofTuples :
#   print(elem)


# remove stopwords ("fake" way)
wordlist = []
for (w, ac) in apcount.items():
    # w is the word, ac is the account count (was bc ’blog count’ in textbook)
    frac = float(ac) / len(accountlist)
    if frac > 0.1 and frac < 0.5:
        wordlist.append(w)

# Save only the 1000 most frequent terms over all accounts (see sumcounts) in popularlist
popularlist = []
#
# INSERT YOUR CODE HERE
#
# Create a list of tuples sorted by index 1 i.e. value field

listofTuples = sorted(sumcounts.items() ,  key=lambda x: x[1],reverse=True)
#extract only 1000 rows
listofTuples = listofTuples[:1000]
#store in the dictionary
popularlist = [i[0] for i in listofTuples]

print("Writing it to text file")
# write out popular word list
with open(’popularlist.txt’, ’w’) as outf:
    for word in popularlist:
        outf.write(word + ’\n’)

# write out account-term matrix
with open(’tweetdata.txt’, ’w’) as outf:
    # write header row ("Account", list of words)
    outf.write(’Account’)
    for word in popularlist:
```

```
122             outf.write('\t%s' % word)
123         outf.write('\n')
124 
125         # write each row (screen_name, count for each word)
126         for (screen_name, wc) in wordcounts.items():
127             outf.write(screen_name)
128             for word in popularlist:
129                 if word in wc:
130                     outf.write('\t%d' % wc[word])
131                 else:
132                     outf.write('\t0')
133             outf.write('\n')
```

Listing 7: generatetweetvector.py

Discussion

The code drive started from line 47 to line 133

- Getting the twitter api authentication at line 50, I passeding secrets.json which contained my twitter consumer_key and consummer_secret
- Read the account files containing the list of scree_names while removeing any spaces each account name
- From lines 61 to 75, uses a major function called getwordcounts(api,screen_name). In this function the parse(api,screen_name) is called. The parse function returns users tweets full text that are not retweets nor replies. When that full tweet text is gotten for a particular user, getwords(tweet) function removes unwanted text from the tweets gotten such as URLS and Mentions. It then extracts word that has at least 3 to 15 length size in each sentence in the tweets and converts them to lower cases.

  The result is stored for each words in a dictionary where by if the word repeats itself again the word(which is the key of the dictionary) increments the values by one.

  getwordcounts returns the screen name with a dictionary or words with its frequency count as well.
- Moving we count the number of accounts each term appear. I noticed that each works and users account are store in a variable outside the for loop, so basically result gotten gets added on each screen_names.
- It appears to keep track of the overall frequency of the word too for every user combined.
- I had done some testing to understand the result for one user in lines 78 to 87

```
"""
Did some testing to figure out the datatype
and sample output for one acccount
Spyder ide  gives flexibility of running each variable without running the whole code after your ran the whole code at least once
"""
#print("Done Counting words")
#print(type(sumcounts))
#print(sumcounts.keys())
#for elem in listofTuples :
#   print(elem)
```

Listing 8: A snapshot for generatetweetvector.py trying to test the output of sumcounts variable listing out the keys values

- Removes stop words (this is calculated, not gotten from a list of unwanted words) basically getting the word count divided by the total number of account names gotten from the accounts.txt, once is satisfy a particular fraction between 0.1 and 0.5 the word is added to the group of wordlist

- For popularlist, From my deduced test above, I was able to sort the item using a lambda function an placed the result in a list of tuples.It goes from highest to lowest. Next I retried just 1000 row items from the tuples(by slicing) and stored it in the dictionary variable popularlist.

```
listofTuples = sorted(sumcounts.items() ,  key=lambda x: x[1], reverse=True)
#extract only 1000 rows
listofTuples = listofTuples[:1000]
#store in the dictionary
popularlist = [i[0] for i in listofTuples]
```

Listing 9: A snapshot for generatetweetvector.py Sort tuples Get 1000 rows Save in dictionary variable popularlist

- Finally, results of popularlist variable words are saved in a text file while tweetdata.txt as a header of the popularlist or words and the screen_name with count of each word on a single row.

- The word i viewed in popularlist.txt makes a lot of sense because it is as a form of connection to politics,sports,music or tech

Q3

Create Dendrogram

Create an ASCII dendrogram and a JPEG dendrogram that uses hierarchical clustering to cluster the most similar accounts (see Module 12, slides 21, 23). Include the JPEG in your report and upload the ASCII file to GitHub (it will be too unwieldy for inclusion in the report).

How well did the hierarchical clustering do in grouping similar accounts together? Were there any particularly odd groupings?

Answer

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 17 21:07:52 2021

@author: aadeniran
"""

from PIL import Image, ImageDraw
from math import sqrt
import random
import csv
import pandas as pd
def readfile(filename):
  # This is a function that I wrote from scratch   -MCW
  data = []
  rownames = []
  colnames = []
  num_rows = 0
  with open(filename) as tsvfile:
    reader = csv.reader(tsvfile, delimiter='\t')
    for row in reader:
      if num_rows > 0:
        rownames.append(row[0])     # save the row names
        data.append([float(x) for x in row[1:]])   # save the values as floats
      else:
        for col in row[1:]:
          colnames.append(col)     # save the column names
      num_rows = num_rows + 1
  return (rownames, colnames, data)

"""
names, word, data= readfiles('tweetdata.txt')
"""

def pearson(v1, v2):
  # Simple sums
    sum1 = sum(v1)
    sum2 = sum(v2)

  # Sums of the squares
    sum1Sq = sum([pow(v, 2) for v in v1])
    sum2Sq = sum([pow(v, 2) for v in v2])

  # Sum of the products
    pSum = sum([v1[i] * v2[i] for i in range(len(v1))])

  # Calculate r (Pearson score)
    num = pSum - sum1 * sum2 / len(v1)
    den = sqrt((sum1Sq - pow(sum1, 2) / len(v1)) * (sum2Sq - pow(sum2,
   2)
               / len(v1)))
    if den == 0:
        return 0

    return 1.0 - num / den
"""
MD5 Scaling
"""
def scaledown(data, distance=pearson, rate=0.01):
    n = len(data)

  # The real distances between every pair of items
    realdist = [[distance(data[i], data[j]) for j in range(n)] for i in
                range(0, n)]

  # Randomly initialize the starting points of the locations in 2D
    loc = [[random.random(), random.random()] for i in range(n)]
    fakedist = [[0.0 for j in range(n)] for i in range(n)]

    lasterror = None
    for m in range(0, 1000):
    # Find projected distances
        for i in range(n):
            for j in range(n):
                fakedist[i][j] = sqrt(sum([pow(loc[i][x] - loc[j][x],
   2)
                                      for x in range(len(loc[i]))]))

    # Move points
        grad = [[0.0, 0.0] for i in range(n)]

        totalerror = 0
        for k in range(n):
            for j in range(n):
                if j == k:
                    continue
        # The error is percent difference between the distances
                errorterm = (fakedist[j][k] - realdist[j][k]) / realdist[j][k]

        # Each point needs to be moved away from or towards the other
        # point in proportion to how much error it has
                grad[k][0] += (loc[k][0] - loc[j][0]) / fakedist[j][k] \
                    * errorterm
                grad[k][1] += (loc[k][1] - loc[j][1]) / fakedist[j][k] \
                    * errorterm

        # Keep track of the total error
                totalerror += abs(errorterm)
        print (totalerror)

    # If the answer got worse by moving the points, we are done
        if lasterror and lasterror < totalerror:
            break
        lasterror = totalerror

    # Move each of the points by the learning rate times the gradient
        for k in range(n):
            loc[k][0] -= rate * grad[k][0]
            loc[k][1] -= rate * grad[k][1]

    return loc

def draw2d(data, labels, jpeg):
    img = Image.new(’RGB’, (2000, 2000), (255, 255, 255))
    draw = ImageDraw.Draw(img)
    for i in range(len(data)):
        x = (data[i][0] + 0.5) * 1000
        y = (data[i][1] + 0.5) * 1000
        draw.text((x, y), labels[i], (0, 0, 0))
    img.save(jpeg, ’JPEG’)

def rotatematrix(data):
    newdata = []
    for i in range(len(data[0])):
        newrow = [data[j][i] for j in range(len(data))]
        newdata.append(newrow)
    return newdata

"""
Hierarchical Clustering
class bicluster - data structure to hold the clustering information
hcluster(rows, distance=pearson) - does the hierarchical clustering, default distance function is pearson()
printclust(clust, labels=None, n=0) - traverses the cluster and prints an ASCII text representation
"""
class bicluster:

    def __init__(self, vec, left=None, right=None, distance=0.0, id=None,):
        self.left = left
        self.right = right
        self.vec = vec
        self.id = id
        self.distance = distance

def hcluster(rows, distance=pearson):
    distances = {}
    currentclustid = -1

  # Clusters are initially just the rows
    clust = [bicluster(rows[i], id=i) for i in range(len(rows))]

    while len(clust) > 1:
        lowestpair = (0, 1)
        closest = distance(clust[0].vec, clust[1].vec)

    # loop through every pair looking for the smallest distance
        for i in range(len(clust)):
            for j in range(i + 1, len(clust)):
        # distances is the cache of distance calculations
                if (clust[i].id, clust[j].id) not in distances:
                    distances[(clust[i].id, clust[j].id)] = \
                        distance(clust[i].vec, clust[j].vec)

                d = distances[(clust[i].id, clust[j].id)]

                if d < closest:
                    closest = d
                    lowestpair = (i, j)

    # calculate the average of the two clusters
```

```python
        mergevec = [(clust[lowestpair[0]].vec[i] + clust[lowestpair[1]].vec[i])
                    / 2.0 for i in range(len(clust[0].vec))]

    # create the new cluster
        newcluster = bicluster(mergevec, left=clust[lowestpair[0]],
                               right=clust[lowestpair[1]], distance=closest,
                               id=currentclustid)

    # cluster ids that weren't in the original set are negative
        currentclustid -= 1
        del clust[lowestpair[1]]
        del clust[lowestpair[0]]
        clust.append(newcluster)

    return clust[0]


def printclust(clust, labels=None, n=0):
  # indent to make a hierarchy layout
    for i in range(n):
        print (' ', end =" ")
    if clust.id < 0:
    # negative id means that this is branch
        print ('-')
    else:
    # positive id means that this is an endpoint
        if labels == None:
            print (clust.id)
        else:
            print (labels[clust.id])

  # now print the right and left branches
    if clust.left != None:
        printclust(clust.left, labels=labels, n=n + 1)
    if clust.right != None:
        printclust(clust.right, labels=labels, n=n + 1)

"""
Dendrogram
"""
def getheight(clust):
  # Is this an endpoint? Then the height is just 1
    if clust.left == None and clust.right == None:
        return 1

  # Otherwise the height is the same of the heights of
  # each branch
    return getheight(clust.left) + getheight(clust.right)


def getdepth(clust):
  # The distance of an endpoint is 0.0
    if clust.left == None and clust.right == None:
        return 0

  # The distance of a branch is the greater of its two sides
  # plus its own distance
    return max(getdepth(clust.left), getdepth(clust.right)) + clust.distance

def drawdendrogram(clust, labels, jpeg='clusters.jpg'):
  # height and width
    h = getheight(clust) * 20
    w = 1200
    depth = getdepth(clust)

  # width is fixed, so scale distances accordingly
    scaling = float(w - 150) / depth

  # Create a new image with a white background
    img = Image.new('RGB', (w, h), (255, 255, 255))
    draw = ImageDraw.Draw(img)

    draw.line((0, h / 2, 10, h / 2), fill=(255, 0, 0))

  # Draw the first node
    drawnode(
        draw,
        clust,
        10,
        h / 2,
        scaling,
        labels,
        )
    img.save(jpeg, 'JPEG')

def drawnode(
    draw,
    clust,
    x,
    y,
    scaling,
    labels,
    ):
    if clust.id < 0:
        h1 = getheight(clust.left) * 20
        h2 = getheight(clust.right) * 20
        top = y - (h1 + h2) / 2
        bottom = y + (h1 + h2) / 2
    # Line length
        ll = clust.distance * scaling
    # Vertical line from this cluster to children
        draw.line((x, top + h1 / 2, x, bottom - h2 / 2), fill=(255, 0,
   0))

    # Horizontal line to left item
        draw.line((x, top + h1 / 2, x + ll, top + h1 / 2), fill=(255,
   0, 0))

    # Horizontal line to right item
        draw.line((x, bottom - h2 / 2, x + ll, bottom - h2 / 2), fill
   =(255, 0,
                  0))

    # Call the function to draw the left and right nodes
        drawnode(
            draw,
            clust.left,
            x + ll,
            top + h1 / 2,
            scaling,
            labels,
            )
        drawnode(
            draw,
            clust.right,
            x + ll,
            bottom - h2 / 2,
            scaling,
            labels,
            )
    else:
    # If this is an endpoint, draw the item label
        draw.text((x + 5, y - 7), labels[clust.id], (0, 0, 0))
"""
K-Means Clustering
"""
def kcluster(rows,k,distance=pearson ):
  # Determine the minimum and maximum values for each point
```

```
    ranges = [(min([row[i] for row in rows]), max([row[i] for row in rows]))
              for i in range(len(rows[0]))]

  # Create k randomly placed centroids
    clusters = [[random.random() * (ranges[i][1] - ranges[i][0]) + ranges[i][0]
                for i in range(len(rows[0]))] for j in range(k)]

    lastmatches = None
    for t in range(100):
        print (’Iteration %d’ % t)
        bestmatches = [[] for i in range(k)]

    # Find which centroid is the closest for each row
        for j in range(len(rows)):
            row = rows[j]
            bestmatch = 0
            for i in range(k):
                d = distance(clusters[i], row)
                if d < distance(clusters[bestmatch], row):
                    bestmatch = i
            bestmatches[bestmatch].append(j)

    # If the results are the same as last time, this is complete
        if bestmatches == lastmatches:
            break
        lastmatches = bestmatches

    # Move the centroids to the average of their members
        for i in range(k):
            avgs = [0.0] * len(rows[0])
            if len(bestmatches[i]) > 0:
                for rowid in bestmatches[i]:
                    for m in range(len(rows[rowid])):
                        avgs[m] += rows[rowid][m]
                for j in range(len(avgs)):
                    avgs[j] /= len(bestmatches[i])
                clusters[i] = avgs

    return bestmatches

"""
Question 3
"""
#running the code
tweetdata, word,data =readfile("tweetdata.txt")
```

```
#clust = hcluster(data)
#print(clust.vec)

"""
Question 3  ASCIII
To view  cluster

printclust(clust,labels=tweetdata)
"""
"""
Question 3 Dendrogram


drawdendrogram(clust,tweetdata,jpeg="Q3/tweetdata.jpeg")

"""
"""
Question 4
"""
"""
For 5  kcluster
number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
    Iteration 6
    Iteration 7
    Iteration 8
    Iteration 9
    Iteration 10
    Iteration 11
    Iteration 12
Cluster summary:
    cluster  1 :  0
    cluster  2 :  42
    cluster  3 :  0
    cluster  4 :  0
    cluster  5 :  56


"""

kclust5 = kcluster(data,5)
#print(len(kclust5))
```

```
for i in range(len(kclust5)):
  print ("cluster ", i+1, ": ", len(kclust5[i]))
  for r in kclust5[i]:
      filen= 'Q4/kclust5cluster'+ str(i+1)+'.csv'
      with open(filen, 'a+') as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")

"""
For 10  kcluster

number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
Cluster summary:
    cluster  1 :  0
    cluster  2 :  0
    cluster  3 :  0
    cluster  4 :  5
    cluster  5 :  3
    cluster  6 :  0
    cluster  7 :  1
    cluster  8 :  87
    cluster  9 :  0
    cluster  10 :  2
"""

kclust10 = kcluster(data,10)
for i in range(len(kclust10)):
  print ("cluster ", i+1, ": ", len(kclust10[i]))
  for r in kclust10[i]:
      filen= 'Q4/kclust10cluster'+ str(i+1)+'.csv'
      with open(filen, 'a+') as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")

"""
For 20  kcluster

number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
clusters summary:
    cluster  1 :  0
    cluster  2 :  54
    cluster  3 :  0
    cluster  4 :  0
    cluster  5 :  7
    cluster  6 :  0
    cluster  7 :  0
    cluster  8 :  0
    cluster  9 :  0
    cluster  10 :  0
    cluster  11 :  0
    cluster  12 :  0
    cluster  13 :  0
    cluster  14 :  1
    cluster  15 :  0
    cluster  16 :  0
    cluster  17 :  6
    cluster  18 :  26
    cluster  19 :  0
    cluster  20 :  4
"""


kclust20 = kcluster(data,20)
for i in range(len(kclust20)):
  print ("cluster ", i+1, ": ", len(kclust20[i]))
  filen= ’Q4/kclust20cluster’+ str(i+1)+’.csv’
  for r in kclust20[i]:
      with open(filen, ’a+’) as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")

"""
Question 5 =’mds2d.jpg’

23380.029615352396
140722.48343299245

98 lenght of coord variable
"""
coords = scaledown(data)
print(len(coords))
draw2d(coords, tweetdata, jpeg=’Q5/tweets2d.jpg’)
```

Listing 10: question3.py for questions 3 4 and 5

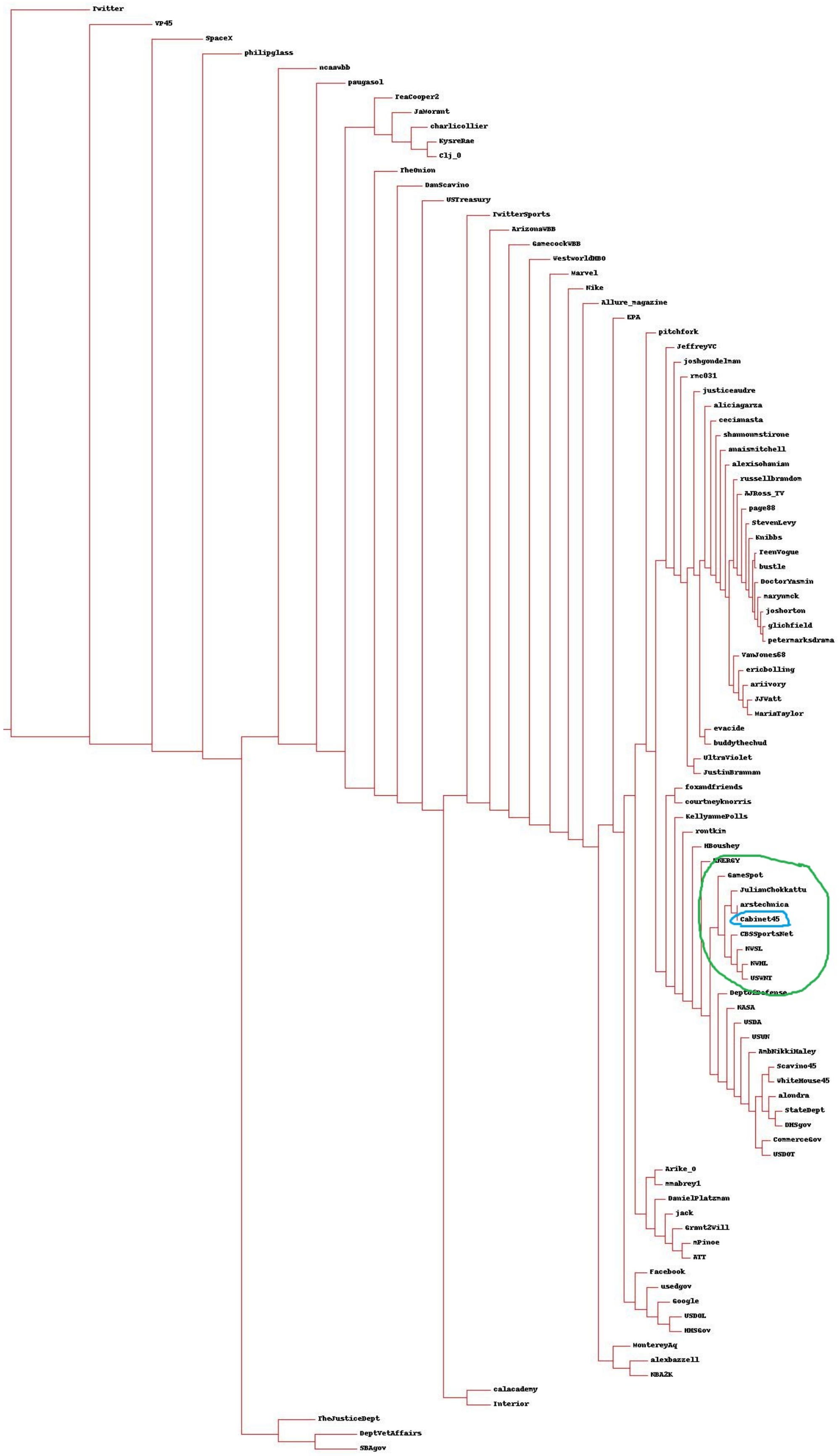

Figure 1: The resulting output for Question 3 twitter account Cabinet45 included

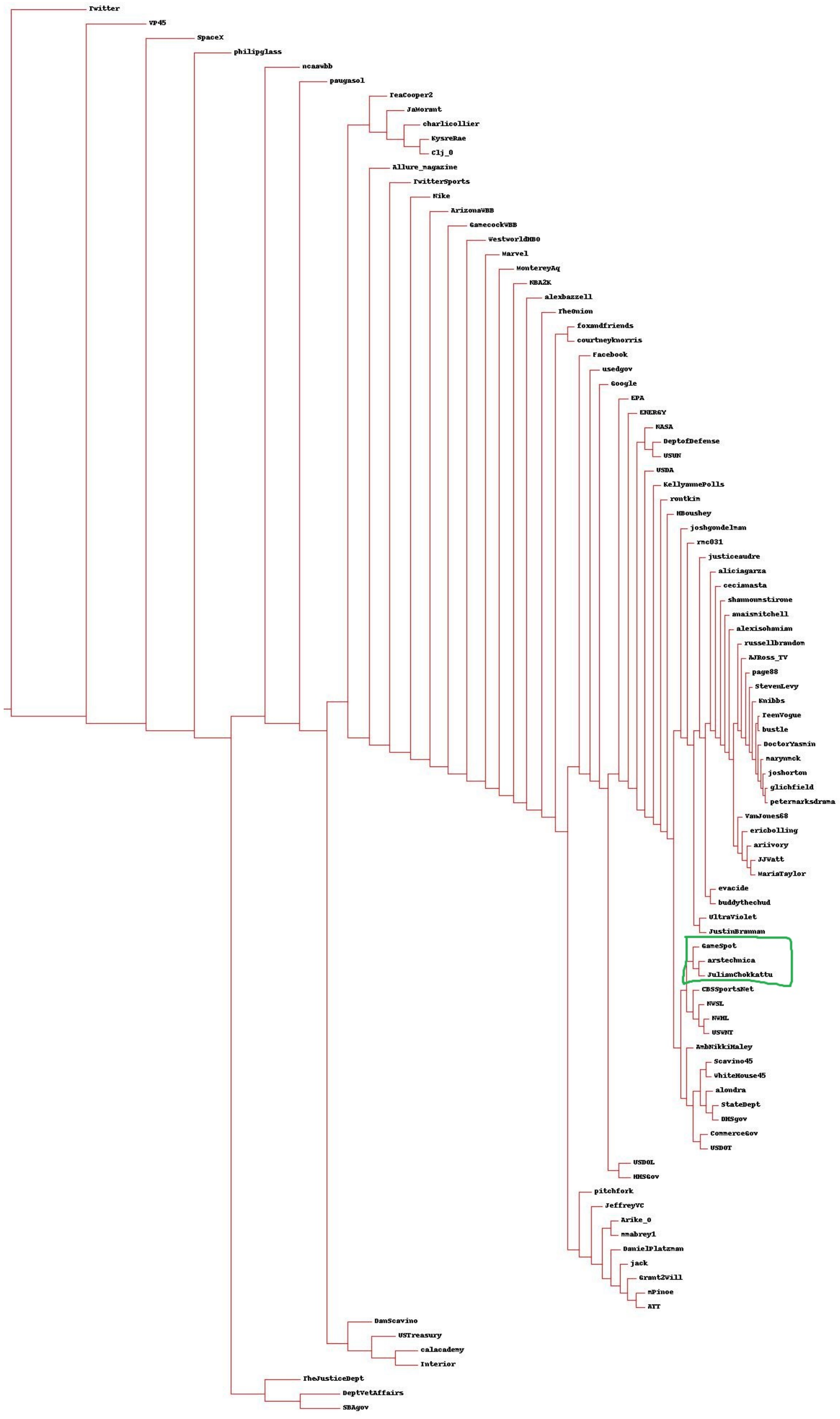

Figure 2: The resulting output for Question 3 after removing tweet account Cabinet45

Discussion

In getting the ASCII and Dendrogram the following had to be put in place:

- Line 349 to 358 is the drive for the code.

```
340
341     return bestmatches
342
343 """
344 Question 3
345 """
346 #running the code
347 tweetdata, word,data =readfile("tweetdata.txt")
348 #clust = hcluster(data)
349 #print(clust.vec)
350
351 """
352 Question 3  ASCIII
353 To view  cluster
354
355 printclust(clust,labels=tweetdata)
356 """
357 """
358 Question 3 Dendrogram
```

Listing 11: A snapshot question3.py driver for Q3

- Used the readfile() function to read in the textdata.txt, it gets all data and returns the usernames as rownames, the words as colnames, and the value as data(data stored as a 2D array of float values).
- Next I parsed in data in hcluster(data) aka hierarchical clustering, this function used the pearson() function when called.

  The pearson function takes two vector arrays as arguments and then returns the pearson correlation between these values.

  hcluster() function also uses a bi-cluster class which is an helper for comparing two clusters.

  hcluster does the hierarchical clustering by agglomerative clustering which involves merging the best matches cluster with a new cluster. It repeats this cycle untill there is just one cluster left.
- printcluster() function prints recursively the final end product of the hclusters returned function. The printcluster has an argument called labels, the label when removed prints the cluster

based on the position number of accounts names but when the label is supplied it prints the actual name of the labels. Print produces the ASCII text of the output. I basically copied and pasted in a text file.

- drawdendrogram() does the similar objective as the printcluster but it produces a jpeg format of the recursive output of the cluster.
- Before i removed the Cabinet45.

  I noticed the cluster algorithm was not doing a great job grouping the scree name properly mainly because the account Cabinet45 came out with all zero result during account-term matrix was generation.

  This also caused odd grouping with GameSpot (tech)
  arstechnica (tech)
  ulianChokkattu(he is a senior associate editor for a tech company WIRED - I chose this company at the beginning) GameSpot(tech) NWSL (sport) NWHL (sport) USWNT(sport)

  After removing the account the clustering was much logical. It broke account even further as seen in figure 1 and figure 2.

  For figure 2 the circled part we see that it only constitute tech screen names as mentioned above.The hierarchical clustering did a good job to me at the end of the day especially just a single variable could make it more accurate.

Q4

Cluster using k-Means

Cluster the accounts using k-Means, using k=5,10,20 (see Module 12, slide 37). For each value of k, create a file that lists the accounts in each cluster and upload to your GitHub repo.

Give a brief explanation of how the k-Means algorithm operates on this data. What features is the algorithm considering?

How many iterations were required for each value of k?

Which k value created the most reasonable clusters? For that grouping, characterize the accounts that were clustered into each group.

Answer

```
"""
Question 3  ASCIII
To view  cluster

printclust(clust,labels=tweetdata)
"""
"""
Question 3 Dendrogram


drawdendrogram(clust,tweetdata,jpeg="Q3/tweetdata.jpeg")

"""
"""
Question 4
"""
"""
For 5  kcluster
number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
    Iteration 6
    Iteration 7
    Iteration 8
    Iteration 9
    Iteration 10
    Iteration 11
    Iteration 12
Cluster summary:
    cluster  1 :  0
    cluster  2 :  42
    cluster  3 :  0
    cluster  4 :  0
    cluster  5 :  56


"""

kclust5 = kcluster(data,5)
#print(len(kclust5))
```

```
for i in range(len(kclust5)):
  print ("cluster ", i+1, ": ", len(kclust5[i]))
  for r in kclust5[i]:
      filen= ’Q4/kclust5cluster’+ str(i+1)+’.csv’
      with open(filen, ’a+’) as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")

"""
For 10  kcluster

number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
Cluster summary:
    cluster  1 :  0
    cluster  2 :  0
    cluster  3 :  0
    cluster  4 :  5
    cluster  5 :  3
    cluster  6 :  0
    cluster  7 :  1
    cluster  8 :  87
    cluster  9 :  0
    cluster  10 :  2
"""

kclust10 = kcluster(data,10)
for i in range(len(kclust10)):
  print ("cluster ", i+1, ": ", len(kclust10[i]))
  for r in kclust10[i]:
      filen= ’Q4/kclust10cluster’+ str(i+1)+’.csv’
      with open(filen, ’a+’) as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")

"""
For 20  kcluster

number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
clusters summary:
    cluster  1 :  0
    cluster  2 :  54
    cluster  3 :  0
    cluster  4 :  0
    cluster  5 :  7
    cluster  6 :  0
    cluster  7 :  0
    cluster  8 :  0
    cluster  9 :  0
    cluster  10 :  0
    cluster  11 :  0
    cluster  12 :  0
    cluster  13 :  0
    cluster  14 :  1
    cluster  15 :  0
    cluster  16 :  0
    cluster  17 :  6
    cluster  18 :  26
    cluster  19 :  0
    cluster  20 :  4
"""


kclust20 = kcluster(data,20)
for i in range(len(kclust20)):
  print ("cluster ", i+1, ": ", len(kclust20[i]))
  filen= ’Q4/kclust20cluster’+ str(i+1)+’.csv’
  for r in kclust20[i]:
      with open(filen, ’a+’) as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")

"""
Question 5 =’mds2d.jpg’

23380.029615352396
140722.48343299245

98 lenght of coord variable
"""
coords = scaledown(data)
print(len(coords))
draw2d(coords, tweetdata, jpeg=’Q5/tweets2d.jpg’)
```

Listing 12: A snapshot question3.py solution driver for Q4

Discussion

K-mean algorithm work in the manner :

- in the kcluster function, it first try to organize the data in ranges of minimum and maximum values for each row so that the clusters can be represents as a coordinate(in 2D form).
- This coordinates then is used to determine best clusters closest to each row of data. The close the value is to zero the closer it is to the cluster or centroid.
- The last part is update the clusters average(mean) to their new members, this changes the location of the clusters and groups them together and closer based on the average of all the group members.
- Final check its to make sure that the best is the matches is the list of rows in each clusters. This ensure the group members are more close together on average.
- kcluster = 5 had 12 iteration starting from zero and produced a total cluster of 5 starting from 1

```
For 5  kcluster
number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
    Iteration 6
    Iteration 7
    Iteration 8
    Iteration 9
    Iteration 10
    Iteration 11
    Iteration 12
Cluster summary:
    cluster  1 :  0
    cluster  2 :  42
    cluster  3 :  0
    cluster  4 :  0
    cluster  5 :  56
```

Listing 13: A snapshot question3.py cluster 5 showing interations and cluster summary

- kcluster = 10 hand 4 iteration starting from zero and produced a total cluster of 5 starting from 1

```
"""
For 10  kcluster

number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
Cluster summary:
    cluster  1 :  0
    cluster  2 :  0
    cluster  3 :  0
    cluster  4 :  5
    cluster  5 :  3
    cluster  6 :  0
    cluster  7 :  1
    cluster  8 :  87
    cluster  9 :  0
    cluster  10 :  2
```

Listing 14: A snapshot question3.py cluster 10 showing interations and cluster summary

- kcluster = 20 had 5 iteration starting from zero and produced a total cluster of 20 starting from 1

```
"""
For 20  kcluster

number of iteration:
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
clusters summary:
    cluster  1 :  0
    cluster  2 :  54
    cluster  3 :  0
    cluster  4 :  0
    cluster  5 :  7
    cluster  6 :  0
    cluster  7 :  0
    cluster  8 :  0
    cluster  9 :  0
    cluster  10 :  0
    cluster  11 :  0
    cluster  12 :  0
    cluster  13 :  0
    cluster  14 :  1
    cluster  15 :  0
    cluster  16 :  0
    cluster  17 :  6
    cluster  18 :  26
    cluster  19 :  0
    cluster  20 :  4
```

Listing 15: A snapshot question3.py cluster 20 showing interations and cluster summary

- From all the list of clusters made I would say none of them sastisfied a uniform grouping but if I would have to choose k equals 10.

  In its cluster 4, it had a odd grouping of Twitter which does not fit any of my previous group(tech, politics, sport and music), the rest of the members where all politics.

  For cluster 5, it had all political screen_name groupings in it which is good.

  In cluster 7 it has just only one sport grouping. In cluster 8,had majorly tech scree name grouping, followed my political grouping then sports, this is why classification I said the clustering did do a good job. The finally cluster 10 was just about tech.

- The code for each k values of (5,10,20) writes the twitter screen names row by row for each cluster based on the cluster summary. Each cluster gets a particular file name.

```
for i in range(len(kclust5)):
  print ("cluster ", i+1, ": ", len(kclust5[i]))
  for r in kclust5[i]:
      filen= 'Q4/kclust5cluster'+ str(i+1)+'.csv'
      with open(filen, 'a+') as myfile:
          myfile.write(tweetdata[r])
          myfile.write("\n")
```

Listing 16: A snapshot question3.py sample of how the code writes to each file based on k value

Q5

Create MDS Image

Use MDS to create a JPEG of the accounts (see Module 12, slide 50). Include the JPEG in your report.

How many iterations were required?

How well did the MDS do in grouping similar accounts together?

Were there any particularly odd groupings?

Answer

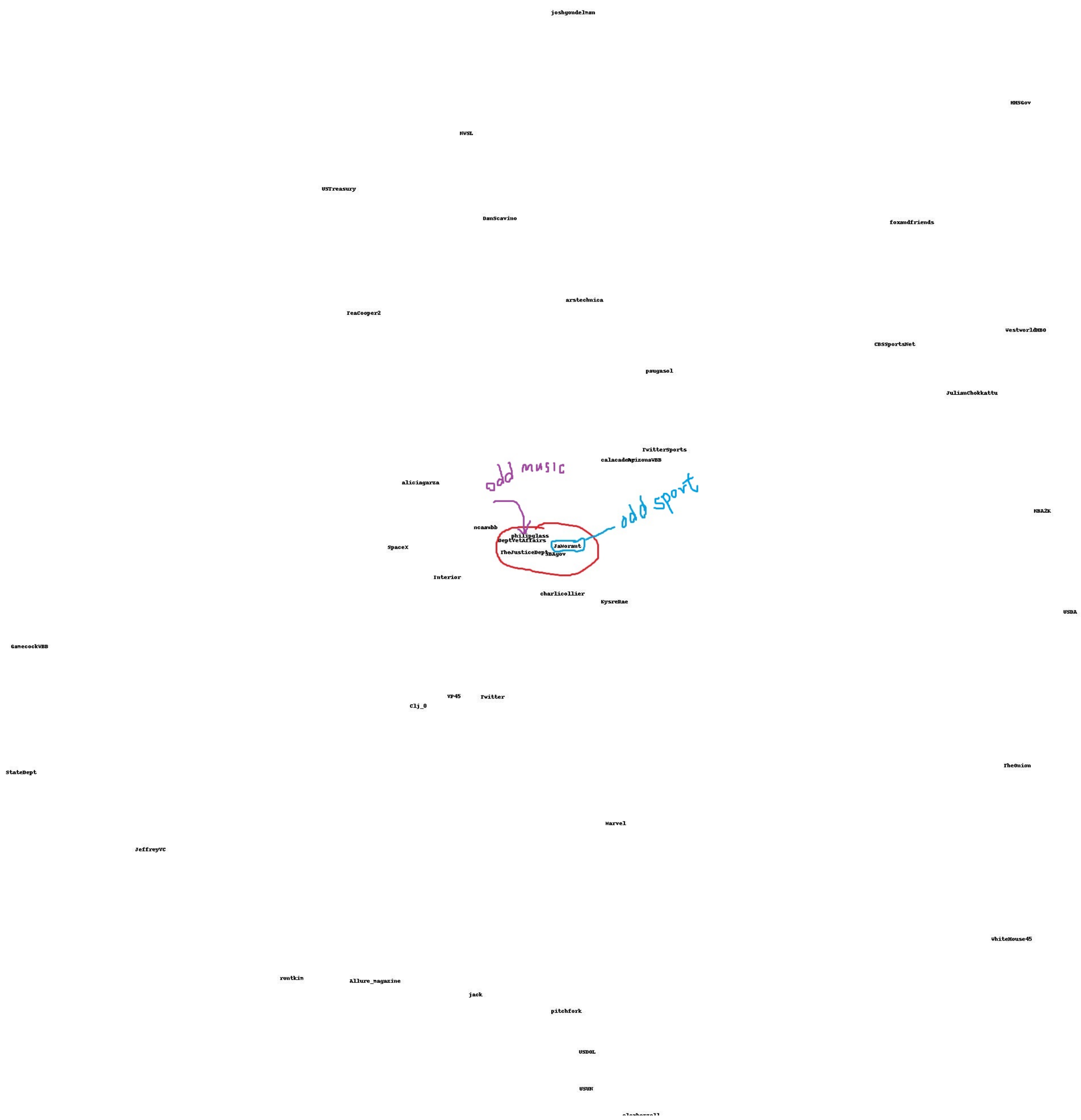

Figure 3: MDS Image Question 5

Discussion

I had the following result:

- There were 98 iteration in total by checking the length of coord in line 486

```
477 """
478 Question 5 ='mds2d.jpg'
479 
480 23380.029615352396
481 140722.48343299245
482 
483 98 lenght of coord variable
484 """
485 coords = scaledown(data)
486 print(len(coords))
487 draw2d(coords, tweetdata, jpeg='Q5/tweets2d.jpg')
```

Listing 17: A snapshot question3.py cluster 20 showing iterations and cluster summary

- MDS did fairly good but there was a group containing two particular odd group members Jamorant (Sport category) and philipglass(Music category), the two were odd because the group was filled with mostly political screen names.
- For the code to work, we would have to read in the tweetdata.txt and parse in the data variable into scaledown(data) function. The functions are from the lecture notes. The function scaledown then creates the file tweets2d.jpg Figure 3.
- I remember running into and error when I parse in data in the scaledown() function.

```
        errorterm = (fakedist[j][k] - realdist[j][k]) / realdist[j][k]
        ZeroDivisionError: float division by zero

```

  Apparently when the code was calculating teh percentage difference between the distance of two pairs, result in a zero value which should not be the case. This meant that one of my screen_names() the Cabinet45 in account-term matric had an output of all zeros values. (View tweetdataWithCabinet45.txt in github line 80)

References

- https://stackoverflow.com/questions/48230230/typeerror-mismatch-between-array-dtype-object-and-format-specifier-18e/48231106
- https://www.kite.com/python/answers/how-to-write-contents-of-a-dataframe-into-a-text-file-in-python

- https://www.geeksforgeeks.org/remove-last-n-rows-of-a-pandas-dataframe/
- https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates/
- https://www.geeksforgeeks.org/how-to-add-one-row-in-an-existing-pandas-dataframe/
- https://stackoverflow.com/questions/10897339/python-fetch-first-10-results-from-a-list
- https://stackoverflow.com/questions/22412258/get-the-first-element-of-each-tuple-in-a-list-in-python