Implementing K-Means Clustering

**Gaurav Jurani**

**16BCE1120**

**CODE -**

from \_\_future\_\_ import division import requests import re

from bs4 import BeautifulSoup import numpy as np

def get\_web\_text(i):

try:

pg= requests.get(i) except:

return '0'

body=pg.content

soup = BeautifulSoup(body, 'html.parser') [tag.decompose() for tag in soup("style")] [tag.decompose() for tag in soup("style")] res = soup.get\_text().strip().lower() res= re.sub(r'[\s,\W,\d]'," ",res) cont=res.split()

return cont

def create\_vect(keywords, texts):

for i in keywords:

for z in i: for j in texts.keys(): if z in (k for k in (texts[j])):

vectors[j][keywords.index(i)] = texts[j].count(z)

#vectors[j][keywords.index(i)]+1

todel=[] for i in vectors.keys(): if sum(vectors[i])==0: print("Failed to fetch: "+i) todel.append(i) for i in todel: del vectors[i]

return vectors

def mandist(a,b): temp=0 for i in range(len(a)): temp+= abs(a[i]-b[i])

return temp

def calc\_centroid(K\_clusts,vectors): for i in K\_clusts.keys():

#print(i)

temp=[0 for j in range(len(keywords))]

#print(temp) for j in K\_clusts[i]: #print(vectors[j]) temp=np.add(temp,vectors[j]) temp2=[] for j in range(len(temp)): x=round((temp[j])/(len(K\_clusts[i])),2)

temp2.append(x) centroids[i]=temp2

return centroids

def iterate(new\_clusts,K\_clusts,centroids):

for i in range(int(k)): new\_clusts["C"+str(i+1)]=[] #print(new\_clusts) dmat={} for i in vectors.keys(): temp=[] for j in centroids.keys():

temp.append(mandist(vectors[i], centroids[j])) dmat[i]=temp

#print(dmat)

for i in dmat.keys(): mind=min(dmat[i])

new\_clusts["C"+str(dmat[i].index(mind)+1)].append(i)

#print(new\_clusts)

flag=0

for i in K\_clusts.keys():

if sorted(K\_clusts[i])==sorted(new\_clusts[i]): flag+=1 else: flag=0

print('\n')

if flag==len(list(K\_clusts.keys())): print("The final cluster is:") print(new\_clusts) else:

K\_clusts=new\_clusts print("The new cluster is:") print(K\_clusts)

centroids= calc\_centroid(K\_clusts, vectors) new\_clusts={}

iterate(new\_clusts,K\_clusts,centroids)

texts={} vectors={} use= input("If need to read docs, input 0, else if you want to read webpages enter

1: ")

if(use=='0'):

docs=["Doc1.txt", "Doc2.txt", "Doc3.txt", "Doc4.txt", "Doc5.txt", "Doc6.txt",

"Doc7.txt", "Doc8.txt","Doc9.txt"]

keywords= [["automotive"],["car","cars"],["motorcycles"], ["self-drive"],["iot"],

["hire"], ["dhoni"]] for i in docs:

texts[i]= ((((open("part1/"+i,"r+")).read()).strip()).lower()).split() else:

docs=["Doc1.txt", "Doc2.txt", "Doc3.txt", "Doc4.txt", "Doc5.txt", "Doc6.txt", "Doc7.txt", "Doc8.txt","Doc9.txt", "Doc10.txt", "Doc11.txt", "Doc12.txt"] keywords= [["tesla"],["electric"],["car","vehicle","automobile"], ["pollution"],["demonetisation"],["gst"], ["blackmoney"]] links=input("Enter the name of the document to be read:") linklist= open(links,"r+").readlines()

for i in linklist:

texts[str(docs[(linklist.index(i))])]= get\_web\_text(i)

#print(texts[str(docs[(linklist.index(i))])])

for i in docs:

vectors[i]=[0 for j in range(len(keywords))]

create\_vect(keywords,texts)

print('\n')

print("The vector representation of documents are:") print(vectors)

K\_clusts={}

#arbitrary initialization of clusters

k=input("Enter the number of arbitrary clusters you want: ") print('\n')

for i in range(int(k)): K\_clusts["C"+str(i+1)]=[]

for i in range(len(list(vectors.keys()))):

K\_clusts["C"+str(int((i%int(k))+1))].append(list(vectors.keys())[i])

centroids={}

centroids=calc\_centroid(K\_clusts, vectors)

#print(centroids)

new\_clusts={} print("The arbitrary cluster is:") print(K\_clusts)

iterate(new\_clusts, K\_clusts,centroids)

**OUTPUT:**

Word listing of each document

1 [19, 0, 20, 0, 0, 0, 0]

2 [0, 0, 30, 0, 0, 4, 0]

3 [0, 0, 112, 0, 0, 0, 0]

4 [0, 0, 17, 0, 0, 0, 0]

5 [0, 0, 8, 0, 0, 0, 0]

Distance Matrix

[0, 33, 111, 22, 31]

[33, 0, 86, 17, 26]

[111, 86, 0, 95, 104]

[22, 17, 95, 0, 9]

[31, 26, 104, 9, 0]

Final Clusters

1:-->[1]

2:-->[2, 4, 5]

3:-->[3]

2ND

Word listing of each document

1 [16, 0, 24, 0, 0, 0, 0]

2 [0, 0, 14, 0, 0, 0, 0]

3 [0, 0, 94, 0, 0, 0, 0]

4 [1, 0, 13, 0, 0, 0, 0]

5 [0, 0, 9, 0, 0, 0, 0]

6 [0, 0, 1, 0, 0, 0, 0]

7 [0, 0, 22, 0, 0, 32, 0]

8 [0, 0, 21, 0, 0, 6, 0]

9 [0, 0, 0, 0, 0, 0, 0]

10 [0, 0, 0, 0, 0, 0, 0]

11 [0, 0, 0, 0, 0, 2, 0]

12 [0, 0, 0, 0, 0, 19, 0]

Distance Matrix

[0, 26, 86, 26, 31, 39, 50, 25, 40, 40, 42, 59]

[26, 0, 80, 2, 5, 13, 40, 13, 14, 14, 16, 33]

[86, 80, 0, 82, 85, 93, 104, 79, 94, 94, 96, 113]

[26, 2, 82, 0, 5, 13, 42, 15, 14, 14, 16, 33]

[31, 5, 85, 5, 0, 8, 45, 18, 9, 9, 11, 28]

[39, 13, 93, 13, 8, 0, 53, 26, 1, 1, 3, 20]

[50, 40, 104, 42, 45, 53, 0, 27, 54, 54, 52, 35]

[25, 13, 79, 15, 18, 26, 27, 0, 27, 27, 25, 34]

[40, 14, 94, 14, 9, 1, 54, 27, 0, 0, 2, 19]

[40, 14, 94, 14, 9, 1, 54, 27, 0, 0, 2, 19]

[42, 16, 96, 16, 11, 3, 52, 25, 2, 2, 0, 17]

[59, 33, 113, 33, 28, 20, 35, 34, 19, 19, 17, 0]

Final Clusters

1:-->[1]

2:-->[2, 5, 6, 7, 8, 9, 10, 11, 12]

3:-->[3]

4:-->[4]