# -\*- coding: utf-8 -\*-

"""

Created on Mon Apr 25 08:59:59 2022

@author: marrd

"""

# -\*- coding: utf-8 -\*-

"""

Created on Sat Apr 23 15:38:07 2022

@author: davem

"""

"""

This script will read a raw 2d text file from my gitHub repo (here)

https://github.com/davemarr621/test2/blob/main/test.txt

and format into a 2d numpy array

Note: This is the original dbscan algorithm - no maxrs - compare to 25C

"""

import urllib.request # the lib that handles the url stuff

import numpy as np

import matplotlib.pyplot as plt

from intervaltree import Interval, IntervalTree

from sklearn.cluster import DBSCAN

from sklearn.cluster import KMeans

from scipy import spatial

import math

import csv

import os

import base64

import pandas as pd

# Get current working directory

cwd = os.getcwd()

print(" Current working directory ", cwd)

# Change to ths directory

os.chdir(r"H:\GMU")

cwd = os.getcwd()

print(" Current working directory ", cwd)

# List file

for file in os.listdir(cwd):

if file.endswith('.csv'):

print(file)

file\_csv = cwd + "\\" + file

print(file\_csv)

url = 'https://raw.githubusercontent.com/davemarr621/test2/main/test.txt'

url\_file = [] # init list

# read url and create a file from 2d text file

for line in urllib.request.urlopen(url):

line = line.decode('utf8') # decode to remove the b

#print(type(line))

newStr = line.replace("set([u'","")

newStr = newStr.split()

#print(newStr)

url\_file.append(newStr)

# Raw file

print(type(url\_file))

file\_csv = "output.csv"

# Convert raw input file to output csv

with open(file\_csv, 'w', newline='') as f\_output:

csv\_output = csv.writer(f\_output)

#csv\_output.writerow(['lat', 'long'])

csv\_output.writerows(url\_file)

# MaxRS starts here:

# create array from file

np\_arr = np.array(url\_file)

print("Array shape from read (row, col) ", np\_arr.shape)

#convert 1d to 2d array

out\_arr = np.reshape(np\_arr, (-1, 2))

print("Array shape from 1d to 2d array ", out\_arr.shape)

# remove characters from array

# our\_arr = np.char.decode(out\_arr)

#convert 2d array to float

new\_our\_arr = (out\_arr.astype(float))

#print(new\_our\_arr)

# Set the width and height offsets of the rectangles

num\_width = 0.5

num\_height = 0.5

stack\_xy\_sorted = new\_our\_arr[new\_our\_arr[:,0].argsort()]

# Number of rows

n = len(stack\_xy\_sorted)

print("Number of rows ",n)

# Set the width and height offsets of the rectangles

# \*\*\* change num\_width to adjust pararmeters

num\_width = 0.5

num\_height = 0.5

pt\_coord = [] # the point

ll\_x\_lr\_x = [] # list of lower left x and lower right x

# 0 at end means left edge (add to interval tree)

# 1 at end means right edge (remove from interval tree)

for zi, zi\_val in enumerate(stack\_xy\_sorted):

#print(zi,zi\_val)

x = zi\_val[0]

y = zi\_val[1]

pt\_coord.append([x,y])

ll\_x = x - num\_width

ll\_y = y - num\_height

ul\_x = x - num\_width

ul\_y = y + num\_height

lr\_x = x + num\_width

lr\_y = y - num\_height

ur\_x = x + num\_width

ur\_y = y + num\_height

# three columns

ll\_x\_lr\_x.append([ll\_x,ll\_y,ul\_y,"0"]) # add lower left x

ll\_x\_lr\_x.append([lr\_x,lr\_y, ur\_y,"1"]) # add lower right x

# print("X dimension => ", x\_dimension)

# print("Y\_dimension => ", y\_dimension)

# Set the height and width of each rectangle

width = np.full((2,n),num\_width) # width of rectangle = 1

height = np.full((2,n),num\_height) # height of rectangle

all\_x\_sorted = []

all\_x\_sorted = sorted(ll\_x\_lr\_x)

# the interval tree code and init variables

tree = IntervalTree()

test\_overlap =[]

overlap\_count = []

list\_overlap\_count = []

max\_count =[]

# Loop through EVENT (X coords) list

# index, ll\_x, ll\_y, ul\_y, 0 Add y-interval or

# index, lr\_x, lr\_y, ur\_y, 1 remove y-interval

for i, val in enumerate(all\_x\_sorted):

# print (i, ",",val)

if '0' in val: #add y-interval

# print("Adding interval")

tree.addi(val[1],val[2])

test\_overlap = tree[val[1]:val[2]] # test for overlap

overlap\_count = (len(test\_overlap))# overlap count

# new three column array with max count in last column

list\_overlap\_count.append([overlap\_count,val[0],val[1]])

#print("Overlap count => ", list\_overlap\_count)

if '1' in val: # remove y-interval

#print(tree)

# tree.removei(val[1],val[2]) throws error - see note below

tree.discardi(val[1],val[2]) # this should remove the interval and be silent if not there - so no error thrown (per Interval Tree author)

print("Tree should be empty => ", tree) # tree should be null

# Convert list\_overlap\_count to a three column numpy array

array\_list\_overlap\_count = np.array(list\_overlap\_count)

#print("All MaxRS points ", array\_list\_overlap\_count)

print("array\_list\_overlap\_count dimensions => ",array\_list\_overlap\_count.shape)

# Create empty array of m x n dimensions

# offset maxrs points by x + 0.5 and y +0.5

add\_array = np.zeros((len(array\_list\_overlap\_count),len(array\_list\_overlap\_count[0]-2)), dtype=float)

# add two columns of 0.5 for offsets

add\_array = np.insert(add\_array, 1, num\_width, axis=1)

add\_array = np.insert(add\_array, 1, num\_width, axis=1)

add\_array=np.delete(add\_array,3,1) # del last column

add\_array=np.delete(add\_array,3,1) # del last column

offset\_array = np.add(array\_list\_overlap\_count, add\_array)

# Slice array\_list\_overlap\_count

# Set number of minimum overlap here

min\_pts\_overlap = 20

slice\_array\_list\_overlap\_count = offset\_array[np.where(offset\_array[:,0] >= min\_pts\_overlap)]

print(slice\_array\_list\_overlap\_count)

print("MaxRS seed points threshold = ", min\_pts\_overlap)

# and list follows: " )

print()

point\_radius\_index\_list = []

# Iterate through x,y coord list and locate all points in the highest density list, within a given radius

for j,val\_j in enumerate(slice\_array\_list\_overlap\_count):

x\_val = (val\_j[1]) # x coordinate

y\_val = (val\_j[2]) # y coordinate

# print(x\_val,y\_val)

# Append index of closest points to this list

point\_radius\_index\_list.append([x\_val, y\_val])

# This list represents the maxrs seed points that meet the threshold

#print(point\_radius\_index\_list)

# This code was written by Dr. Zufle from scratch for DBSCAN

# Date: 20 APR 2020

# Import required libraries

import math # For distance calculations

import queue # For point queries

# This class adds a new data type to Python

# This is the initializing method for the class

# We will cal this method to construct an instance p of the point class

# Instances

class Point:

def \_\_init\_\_(self, idx, x, y):

self.id = idx

self.name = "No name"

self.x=x

self.y=y

self.label = "Noise" #"Noise", "Border", "Core"

self.cluster\_id = -1

def print\_point(self):

print('({0},{1})'.format(self.x, self.y))

# This class will implement functions for : 1) database; 2) distance

# range\_query, and expand cluster

class database:

def \_\_init\_\_(self,points):

self.points = points

def distance(p,q):

return(math.sqrt((p.x-q.x)\*\*2+(p.y-q.y)\*\*2))

def range\_query(p,DB,eps):

count = 0

neighbors = []

for q in DB.points:

if distance(p,q)<=eps:

neighbors.append(q)

return(neighbors)

def expand\_cluster(DB, seeds, eps, minPts, cluster\_id):

point\_queue = queue.Queue()

for i in seeds:

point\_queue.put(i)

while(True):

p = point\_queue.get()

new\_seeds = range\_query(p,DB,eps)

if len(new\_seeds)<minPts: #Not Core

p.label="Border"

p.cluster\_id=cluster\_id

if len(new\_seeds)>=minPts: #Core

if (p.label!="Core"):

p.label="Core"

p.cluster\_id=cluster\_id

for neighbor in new\_seeds:

if(neighbor.label!="Core"):

point\_queue.put(neighbor)

if (point\_queue.empty()):

return

return

def DBSCAN\_with\_maxrs\_seeds(DB, eps, minPts,max\_rs):

global collector # use this variable to collect all resuts collecte

collector = []

next\_clusterID = 1

for p in max\_rs:

print("I'm now processing Point:", p.name)

# all results collect here

collector.append([p.id, p.x,p.y,p.label,p.cluster\_id])

#print(p.x,p.y,p.label,p.cluster\_id)

if (p.label=="Noise"):

seeds = range\_query(p,DB, eps)

if len(seeds)<minPts: #p is not a core-point

p.label = "Noise"

if len(seeds)>=minPts: #p is a core-point

p.label = "Core"

p.cluster\_id = next\_clusterID

expand\_cluster(DB,seeds,eps,minPts,next\_clusterID)

next\_clusterID += 1

return

# Read csv output file

var\_list = []

i = 0

with open(file\_csv, newline='') as csv\_file:

reader = csv.reader(csv\_file, delimiter = ' ',quotechar='|')

for row in reader:

row\_strip = str(row).replace('[','').replace(']','').replace("'",'')

#print(row\_strip)

#print(type(row\_strip))

list\_of\_items\_in\_line = row\_strip.split(",")

i = i +1

x\_coord = (list\_of\_items\_in\_line[0])

y\_coord = (list\_of\_items\_in\_line[1])

p\_num = "p" + str(i)

#print(p\_num,x\_coord,y\_coord)

var\_list.append([p\_num, x\_coord,y\_coord]) # add lower right x

# This list represents the maxrs seed points that meet the threshold

print("List of MaxRS points ", point\_radius\_index\_list)

print()

# P plus point ID, X, Y

all\_points = []

#print(type(var\_list))

#print(type(point\_radius\_index\_list))

#print(var\_list)

max\_rs = []

# Find out the id based on the coordinate of the two lists

for kk, kx in enumerate(point\_radius\_index\_list):

print(kx[0],kx[1])

for mm, mx in enumerate(var\_list):

if math.isclose(float(kx[0]), float(mx[1])) and math.isclose(float(kx[1]), float(mx[2])):

print("Found ",mx[0], mx[1],mx[2])

P = Point(mx[0],float(mx[1]),float(mx[2])) # convert to float

max\_rs.append(P)

print(type(max\_rs))

print(max\_rs)

for idx, x in enumerate(var\_list):

#print(idx + 1, x[0],x[1],x[2])

#print(x[0]) # the point id

# THis will create a instance for each point using the Point class

P = Point(x[0],float(x[1]),float(x[2])) # convert to float

print(P)

all\_points.append(P)

# Add all object instances of points to a database

DB = database(all\_points)

#print(type(DB))

#max\_rs = [p62,p63,p64]

# Run the DBSCAN algorithm using eps = and minPts =

DBSCAN\_with\_maxrs\_seeds(DB,1,3,max\_rs)

"""

# Print all results

x=[]

y=[]

c = []

p\_type = []

print(type(collector))

for k in collector:

#print(k[0],k[1])

x.append(k[1])

y.append(k[2])

c.append(k[4]) # cluster id

p\_type.append(k[3])

plt.plot(x,y,'o')

df = pd.DataFrame(dict(x=x,y=y,label=c))

groups = df.groupby('label')

fig, ax = plt.subplots()

ax.margins(0.05)

for name, group in groups:

ax.plot(group.x,group.y, marker='o', linestyle = '', ms=4, label=name)

ax.legend()

for i, txt in enumerate(p\_type):

ax.annotate(txt, (x[i],y[i]),fontsize = 8)

plt.show()

"""