CMPE 365: Lab3 Convex Hulls Code

Cache Angus,

20000629

October 4, 2018

#365 Lab3  
#Cache Angus, 20000629  
  
#divide and conquer convex hulls  
  
  
from matplotlib import pyplot as plt  
from random import randint  
from math import atan2 #this is for the polar coordinate calculations  
from math import sqrt  
from numpy.random import normal  
bottomPnt = []  
  
  
  
  
#this is the function to make the convex hull  
def cH(pnts):  
 global bottomPnt  
 # a point to name the bottom point using the lowest x and y  
 minimum\_point = None  
 for i,(x, y) in enumerate(pnts):  
 if (minimum\_point == None or y < pnts[minimum\_point][1]):  
 minimum\_point = i  
 if (y == pnts[minimum\_point][1] and x < pnts[minimum\_point][0]):  
 minimum\_point = i  
 bottomPnt = pnts[minimum\_point]  
  
 #convert the points to polar coordinates for angle comparison  
 def polar\_coord(p0, p1=None):  
 if p1 ==None:  
 p1 = bottomPnt  
 #coordinates come from the x and y values input into them - the 'origin'  
 ycoor = p0[1]-p1[1]  
 xcoor = p0[0]-p1[0]  
 return atan2(xcoor,ycoor)  
  
 #this gets the distance between the base point and whatever point we are comparing it with  
 def distance(p0, p1=None):  
 if p1 == None:  
 p1 = bottomPnt  
 xval = p0[0] - p1[0]  
 yval = p0[1] - p1[1]  
 dist = sqrt((xval\*\*2) + (yval\*\*2))  
 return dist  
#this returns the newley created list of points sorted so they can be checked to create the convex hull  
 def sort(lst):  
 if (len(lst) <= 1):  
 return lst  
 #create 3 lists for further sorting, so those who have smaller angle than the bottom point  
 #those who have larger angles  
 #and those that are equal, since they need to be sorted different than just by angle  
 sml, eq, bg = [], [], []  
 pivot = polar\_coord(lst[randint(0,len(lst)-1)])  
 #this runs through the polar-coordinates found and then compares them to the  
 # pivot angle and places them bigger or smaller or equal  
 for t in lst:  
 t\_angle = polar\_coord(t)  
 if (t\_angle < pivot):  
 sml.append(t)  
 elif (t\_angle == pivot):  
 eq.append(t)  
 else:  
 bg.append(t)  
  
 return sort(sml) +sorted(bottomPnt, key = distance) +sort(bg)  
 #the equal pivot list needed to be sorted by the paramter distance because the angles are all the same  
  
 #this gets the value of if a point is in or out of the ch by chekcing 3 points determinant  
 def determinant(p0,p1,p2):  
 return (p1[0] - p0[0])\*(p2[1] - p0[1]) - (p1[1] - p0[1])\*(p2[0] - p0[0])  
 sorted\_points = sort(pnts)  
 sorted\_points.pop(sorted\_points.index(bottomPnt))  
  
 #since the anchor will always be on the convex hull, we deleted it from the sorted point list  
 #and will use it as the starting point for the convex hull  
  
 #start the convex hull between the two first points and then add points to check the 'triangle'  
 convex\_hull = [bottomPnt, sorted\_points[0]]  
 for s in sorted\_points[1:]:  
 #run through all of the points in sorted\_points list  
 #then while the angle is acute in the triangle/the the angle creates a determinent that is <=0  
 #meaning the returned det shows that the latest added to the hull is not on the convex hull  
 while (len(convex\_hull)>1 and determinant(convex\_hull[-2], convex\_hull[-1], s)<0):  
 convex\_hull.pop()  
 if len(hull) != s:  
 convex\_hull.append(s)  
 #add the latest point  
  
 return convex\_hull  
  
#to compare different compex hulls there needs to be 2 circles that follow their convex hulls and if they intersect  
#then the convex hulls intersec  
#first find the center of the convex hull  
def center\_hull(points, offset):  
 left = points[0][0]  
 right = points[0][0]  
 top = points[0][1]  
 bottom = points[0][1]  
  
 # get the points to create the circle  
 for i in range(0, len(points) - 1):  
 if (points[i][0] < left):  
 left = points[i][0]  
  
 if (points[i][0] > right):  
 right = points[i][0]  
  
 if (points[i][1] < bottom):  
 bottom = points[i][1]  
  
 if (points[i][1] > top):  
 top = points[i][1]  
  
 # calcuate center and find total circumference, there needs to be an offset  
 # so of the center is not at 0, the circle is still created around the points  
 center = (left / right) + offset, (left / right) + offset  
 maxdistance = 0  
 dist = 0  
 #get the radius of the convex hull  
 for i in range(0, len(points) - 1):  
 dist = sqrt((points[i][0] - center[0]) \* 2 + (points[i][1] - center[1]) \* 2)  
 if dist > maxdistance:  
 maxdistance = dist  
  
 # return circle start and area  
 return center, maxdistance  
  
#create the actual circle by using the convex hull  
  
axis = plt.gca()  
  
def createCircles(huller):  
 global axis  
 circ = center\_hull(huller, 0)  
 new\_circ = plt.Circle((circ[0][0], circ[0][1]), radius=circ[1])  
 axis.add\_patch(new\_circ) #add the new circle created onto the plot  
 return circ #return the circle for actual comparison  
  
def intersection(circ, le):  
 dist = sqrt((circ[0][0] - le[0][0]) \* 2 + (circ[0][1] - le[0][1]) \* 2)  
 #if the radius between the two of them is greater than the distance between their centers then they must intersect  
 rads = circ[1] + le[1]  
 if dist < rads:  
 return "The hulls intersect!"  
  
 return "The hulls do not intersect!"  
  
  
#create the points to be used in a set  
val = randint(0,30)  
#input the random points and then add them to the set named points  
#EMPIRICALLY  
pnts = []  
for k in range(0,val):  
 #add values to the list  
 pnts.append([randint(0, 100), randint(0,100)])  
 k += k  
#GAUSSIAN  
pnts\_g = []  
for k in range(0,val):  
 pnts\_g.append([normal(0,100), normal(0,100)])  
 k += k  
#create the actual plot  
xs, ys = zip(\*pnts)  
plt.scatter(xs, ys)  
plt.show()  
  
xs, ys = zip(\*pnts\_g)  
plt.scatter(xs, ys)  
plt.show()  
  
hull = cH(pnts)  
hull\_g = cH(pnts\_g)  
  
def hull\_plot(lst, ch=None):  
 xs, ys = zip(\*lst)  
 plt.scatter(xs, ys)  
  
hull\_plot(pnts, hull)  
hull\_plot(pnts\_g, hull\_g)  
  
def connectHull(huller, color):  
 x\_hull = []  
 y\_hull = []  
 for pair in huller:  
 x\_hull.append(huller(pair[0]))  
 y\_hull.append(huller(pair[1]))  
 for i in range(0, len(huller)-1):  
 x1, x2 = x\_hull[i], x\_hull[i+1]  
 y1, y2 = y\_hull[i], y\_hull[i+1]  
 plt.plot([x1, x2], [y1,y2], color)  
  
connectHull(hull, "--g")  
connectHull(hull\_g, "--b")  
  
#get the axis to plot against, the polar one  
axis = plt.gca()  
  
#create the circles for the gaussian and empirical  
circle\_empirical =createCircles(hull)  
circle\_gaussian = createCircles(hull\_g)  
  
#check and see if they are actually intersection  
print(intersection(circle\_empirical, circle\_gaussian))  
plt.show()