import numpy as np

# preparing input

x\_coords = [-1, -1, 0, 0, 1, 1, 2, 2, 3, 3]

y\_coords = [2, 1, 3, 2, 3, -1, 0, -1, 1, 0]

points = [list(x) for x in zip(x\_coords, y\_coords)]

classification\_input = ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']

def euclidean\_dist(x1, y1, x2, y2):

# returns euclidean distance between (x1, y1) and (x2, y2)

return np.sqrt((x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2));

def nearest\_neighbors(x\_coords, y\_coords, classification, x, y):

# returns a list of points sorted in ascending order by distance from (x,y)

distances = []

for i in range(0, len(x\_coords)):

distances.append((euclidean\_dist(x, y, x\_coords[i], y\_coords[i]), x\_coords[i], y\_coords[i], classification[i]))

# append a tuple to the list of form (euclidean\_dist(x, y, xi, yi), xi, yi, class of (xi, yi))

distances.sort()

# sort the list by euclidean\_dist i.e. first element of the tuple (default)

return distances

def get\_knn(x\_coords, y\_coords, classification, x, y, k):

# returns 'k' nearest neighbors of (x,y)

return nearest\_neighbors(x\_coords, y\_coords, classification, x, y)[0: k];

def knn\_prediction(x\_coords, y\_coords, classification, x, y, k):

# checks class of k nearest neighbours and returns the majority class as prediction for that point

knn = get\_knn(x\_coords, y\_coords, classification, x, y, k)

# print(x, y, knn)

PLUSes = 0

Os = 0

for j in range(0, k):

if(knn[j][3]=='o'):

Os += 1;

else:

PLUSes += 1

if(Os>PLUSes):

return 'o'

elif(PLUSes>Os):

return '+'

else:

return '='

def knn(x\_coords, y\_coords, classification, k):

# returns array of predicted values for all the points in the input set

prediction = []

for i in range(0, len(x\_coords)):

prediction.append(knn\_prediction(x\_coords, y\_coords, classification, x\_coords[i], y\_coords[i], k))

return prediction

def getLOOCVError(x\_coords, y\_coords, classification, k):

wrong\_prediction = 0

for i in range(0, len(x\_coords)):

new\_x\_coords = x\_coords[:i]+x\_coords[i+1:]

new\_y\_coords = y\_coords[:i]+y\_coords[i+1:]

new\_classification = classification[:i]+classification[i+1:]

element\_prediction = knn\_prediction(new\_x\_coords, new\_y\_coords, new\_classification, x\_coords[i], y\_coords[i], k)

# print(x\_coords[i], y\_coords[i], element\_prediction, classification[i])

if(element\_prediction != classification[i]):

wrong\_prediction += 1

return wrong\_prediction/len(x\_coords)

# Running classification on the training set for an example output]

print('Actual Classes:', classification\_input)

predictionk1 = knn(x\_coords, y\_coords, classification\_input, 1)

print('Predicted Classes:')

print('k=1 ', predictionk1)

predictionk3 = knn(x\_coords, y\_coords, classification\_input, 3)

print('k=3 ', predictionk3)

predictionk5 = knn(x\_coords, y\_coords, classification\_input, 5)

print('k=5 ', predictionk5)

predictionk7 = knn(x\_coords, y\_coords, classification\_input, 7)

print('k=7 ', predictionk7)

predictionk9 = knn(x\_coords, y\_coords, classification\_input, 3)

print('k=9 ', predictionk9)

print('\nLOOCV Errors: ')

print('k=1 ', getLOOCVError(x\_coords, y\_coords, classification\_input, 1))

print('k=3 ', getLOOCVError(x\_coords, y\_coords, classification\_input, 3))

print('k=5 ', getLOOCVError(x\_coords, y\_coords, classification\_input, 5))

print('k=7 ', getLOOCVError(x\_coords, y\_coords, classification\_input, 7))

print('k=9 ', getLOOCVError(x\_coords, y\_coords, classification\_input, 9))