John Soderstrom
CS 4342
Project 4

The starting seed is 95. Other tested seeds are: 50, 24, 127, 421, 379, 879, 801, 699, 555, and 333.

In both k = 2 and k = 4 presented below, distances for cluster 2 increased across multiple iterations. In k = 2, both types of distances for cluster 2 increase across all iterations. In k = 4, they increase until iteration 8 and will sometimes continue to increase. The TSSE values never increased, while the TSE value increased in exactly one iteration: iteration 10. It appears a cluster may increase its errors as long as the overall error (TSSE, at least) is decreasing.

Across every other seed, the trend continued. The only seeds that saw no increase in any cluster's error were those that needed only one iteration to complete. There were two cases where this occurred in k = 2 examples. Otherwise, every seed had at least one cluster that usually increased in error.

TSSE decreased in every case. TSE decreased in all but one case; during seed 333 on k = 2, TSE increased on iteration 4. The value changed from 1518.35 to 1519.23. However, TSSE still decreased on that iteration.

Output

Iteration #	Clust#1		Clust#2		Clust#3		Clust#4		TSSE (sq-dist)	TSE (dist)
	Intra-sq-dist	Intra-dist	Intra-sq-dist	Intra-dist	Intra-sq-dist	Intra-dist	Intra-sq-dist	Intra-dist		
1	11929.74	471.45	101.3	14.23	292.02	32.55	20059.25	598.84	32382.3	1117.08
2	7916.24	362.14	999.17	72.19	1198.2	61.89	16134.06	506.52	26247.67	1002.75
3	5438.15	277.01	2026.69	128.14	1684.64	87.4	14966.32	471.31	24115.8	963.86
4	5438.15	277.01	2147.5	133.29	2054.69	105.3	13842.04	436.45	23482.38	952.06
5	5438.15	277.01	2979.76	175.31	2054.69	105.3	12138.31	375.93	22610.91	933.56
6	5438.15	277.01	2666.29	157.39	2270.49	121.99	12138.31	375.93	22513.23	932.32
7	5438.15	277.01	4497.79	201.5	2270.49	121.99	9936.72	330.67	22143.14	931.17
8	5438.15	277.01	4059.94	181.41	2526.49	139.9	9936.72	330.67	21961.29	928.99
9	5438.15	277.01	3423.17	155.63	2956.92	160.2	9936.72	330.67	21754.95	923.52
10	5438.15	277.01	5778.83	210.21	2956.92	160.2	7048.39	280.85	21222.3	928.27
11	5438.15	277.01	4363.85	180.42	3838.62	203.83	6111.6	253.36	19752.21	914.62
12	5438.15	277.01	4109.79	181.64	4332.04	224.74	4677.52	219.34	18557.49	902.73
13	5438.15	277.01	4559.49	203.38	4332.04	224.74	4075.21	197.47	18404.88	902.6

Iteration #	Clust#1		Clust#2		TSSE (sq-dist)	TSE (dist)
	Intra-sq-dist	Intra-dist	Intra-sq-dist	Intra-dist		
1	56742.68	1474.83	509.13	45.63	57251.81	1520.46
2	43985.38	1207.75	3510.49	183.44	47495.86	1391.2
3	33933.49	969.7	8534.63	355.83	42468.12	1325.53
4	29753.94	858.48	11694.72	452.18	41448.66	1310.66
5	28881.76	828.57	12438.28	479.13	41320.04	1307.7

Source Code

```
######################################
# Author: John Soderstrom
# Due: 5/13/2020
#
# Randomly generate a given number of 2D points (1.0 <= x, y <= 100.0)
# and divide them into 2 or 4 clusters. Scales to any number of clusters,
# as long as they do not outnumber the points.
#
# Uses euclidean distance to judge nearby clusters.
#
# Centroids are randomly selected from the random points before
# being adjusted with every iteration. The program ends when
# centroids do not move from the previous iteration.
#
# With a little more work, this program could scale not just to the
# number of clusters, but also the number of dimensions. For now, this
# program merely works with 2D points.
#######################################

import random
import copy
from pprint import pprint as pp

###
# Stores randomly generated points given bounds, and creates/tracks
# centroids. Tracks the number of iterations, and prints results
# with formatting to screen as they happen.
###
class Clusters:
    def __init__(self, kVal, numPoints, minX, maxX, minY, maxY):
        self.kVal = kVal
        self.numPoints = numPoints
        self.iterations = 0
        self.points = []
        self.centroids = []
        self._generatePoints(minX, maxX, minY, maxY)
        self._generateCentroids()
        self._printHeader()
        self._iterate()

    ###
    # Given bounds and number of points to generate, fills a
    # list with 2D point data. An extra element is added to
    # track which cluster they will be part of.
    ###
    def _generatePoints(self, minX, maxX, minY, maxY):
        for i in range(self.numPoints):
            tempX = random.uniform(minX, maxX)
            tempY = random.uniform(minY, maxY)
            self.points.append([tempX, tempY, -1])

    ###
    # After points are generated, randomly select a given number
    # equal to the number of clusters as centroids from the points.
    ###
    def _generateCentroids(self):
        indices = random.sample(range(self.numPoints), self.kVal)
        for i in indices:
            self.centroids.append(self.points[i][:-1].copy())

    ###
    # Get the euclidean distance between two points. This doubles
    # as the error from various points to the mean, or their centroid.
    ###
    def _euclid(self, p1, p2):
        dist = 0
        for i in range(len(p1) - 1):
            dI = p1[i] - p2[i]
            dist += dI * dI
        return pow(dist, .5)

##    def _dist(self, p1, p2):
##        dist = 0
##        for i in range(len(p1) - 1):
##            dI = p1[i] - p2[i]
##            dist += abs(dI)
##        print(dist)
##        return dist

    ###
    # Gets the squared distance between two points. Used to output
    # information to the screen from print statements.
    ###
    def _distSq(self, p1, p2):
        dist = 0
        for i in range(len(p1) - 1):
            dI = p1[i] - p2[i]
            dist += dI * dI
        return dist
```

```
###
# Assign points to clusters and update centroids. Repeat
# until centroids do not change from previous iteration.
# Prints for each iteration except the last, the repeat.
###
def _iterate(self):
    keepIter = True
    while keepIter:
        self._setClusters()
        testUpdate = self._updateCentroids()
        keepIter = self._checkUpdatedCentroids(testUpdate)
        self.iterations += 1
        if keepIter:
            self._printLine()

###
# Assign each point to a centroid. Finds the euclidean distance
# from a point to all centroids and selects the one that is closest.
# The centroid number takes the last element's place in each point.
###
def _setClusters(self):
    for count, i in enumerate(self.points):
        nearDist = -1
        for countCent, j in enumerate(self.centroids):
            tempDist = self._euclid(i, j)
            if nearDist > tempDist or nearDist is -1:
                nearDist = tempDist
                self.points[count][-1] = countCent

###
# Returns a list containing the updated centroids after an
# iteration. Quickly cycles through all points adding their
# values to the updated centroid list, tracking the number
# of points for each centroid. Divides the final result by
# that number to get the mean.
###
def _updateCentroids(self):
    numPoints = [0] * self.kVal
    nextCentroid = []
    for i in range(self.kVal):
        nextCentroid.append([0, 0])

    for countPoint, i in enumerate(self.points):
        numPoints[i[-1]] += 1
        for j in range(len(i) - 1):
            nextCentroid[i[-1]][j] += i[j]
    for count, i in enumerate(numPoints):
        for j in range(len(self.centroids[0])):
            nextCentroid[count][j] /= i

    return nextCentroid

###
# Given a list of new centroids, checks them against the
# previous list. If any differences are found, assign the updated
# values to the actual centroids and return early. Returning False
# means there are no changes and iterations can stop.
###
def _checkUpdatedCentroids(self,testUpdate):
    for countPoint, i in enumerate(testUpdate):
        for countPart, j in enumerate(i):
            if abs(j - self.centroids[countPoint][countPart]) > .0001:
                self.centroids = testUpdate
                return True
    return False

###
# Given the number associated with a centroid/cluster, sum all
# the squared distances between the centroid and all points
# within the cluster. Return the sum.
###
def _clustDistSq(self, clustNum):
    dist = 0
    for i in self.points:
        if i[-1] is clustNum:
            dist += self._distSq(i, self.centroids[clustNum])
    return dist

###
# Given the number associated with a centroid/cluster, sum all
# the distances between the centroid and all points within
# the cluster. Return the sum.
###
def _clustDist(self, clustNum):
    dist = 0
    for i in self.points:
        if i[-1] is clustNum:
            dist += self._euclid(i, self.centroids[clustNum])
    return dist

###
```

```
# Called once at the start. Prints two formatted lines with header
# information for the final table. Cluster specific information is
# scaled to the number of clusters.
###
def _printHeader(self):
    # Line 1, most important information here
    headFormat = ['Iteration #']
    headString = '{:<13s}'
    for i in range(self.kVal):
        headFormat.append('Clust#{}'.format(i + 1))
        headString += '{:<25s}'
    headFormat.append('TSSE (sq-dist)')
    headFormat.append('TSE (dist)')
    headString += '{:<15s}{:<11s}'
    print(headString.format(*headFormat))

    # Line 2, divides up cluster specific columns
    headFormat = ['']
    headString = '{:<13s}'
    for i in range(self.kVal):
        headFormat.append('Intra-sq-dist')
        headFormat.append('Intra-dist')
        headString += '{:<14s}'
        headString += '{:<11s}'
    print(headString.format(*headFormat))

###
# Called once for each iteration. Prints a formatted string
# containing iteration number, error values across clusters,
# and error values specific to each cluster.
###
def _printLine(self):
    lineString = '{:<13s}'
    lineFormat = [str(self.iterations)]
    tsse = 0
    tse = 0
    for i in range(self.kVal):
        lineString += '{:<14s}'
        lineString += '{:<11s}'
        tempSq = self._clustDistSq(i)
        tempDist = self._clustDist(i)
        tsse += tempSq
        tse += tempDist
        lineFormat.append(str(round(tempSq, 2)))
        lineFormat.append(str(round(tempDist, 2)))
    lineString += '{:<15s}{:<11s}'
```

```
        lineFormat.append(str(round(tsse, 2)))
        lineFormat.append(str(round(tse, 2)))
        print(lineString.format(*lineFormat))

# Checking for "bad" seeds that required more iterations than most (on 4 clusters), I got these:
#  (seeds, iterations): (0, 13), (10, 14), (53, 16), (189, 19), (209, 21), (592, 27)
# Then I realized I did that for 100 points when we only needed 50.
# Doing it again for 50 points got the following:
#  (18, 11), (25, 12), (64, 13), (95, 14), (284, 15), (580, 16)
random.seed(95)
numPoints = 50
numClust = 4
minX = minY = 1.0
maxX = maxY = 100.0

test = Clusters(numClust, numPoints, minX, maxX, minY, maxY)
#test._iterate()
```