CSE 469: Assignment 3
k-Means and Hierarchical Clustering

Peter M. VanNostrand

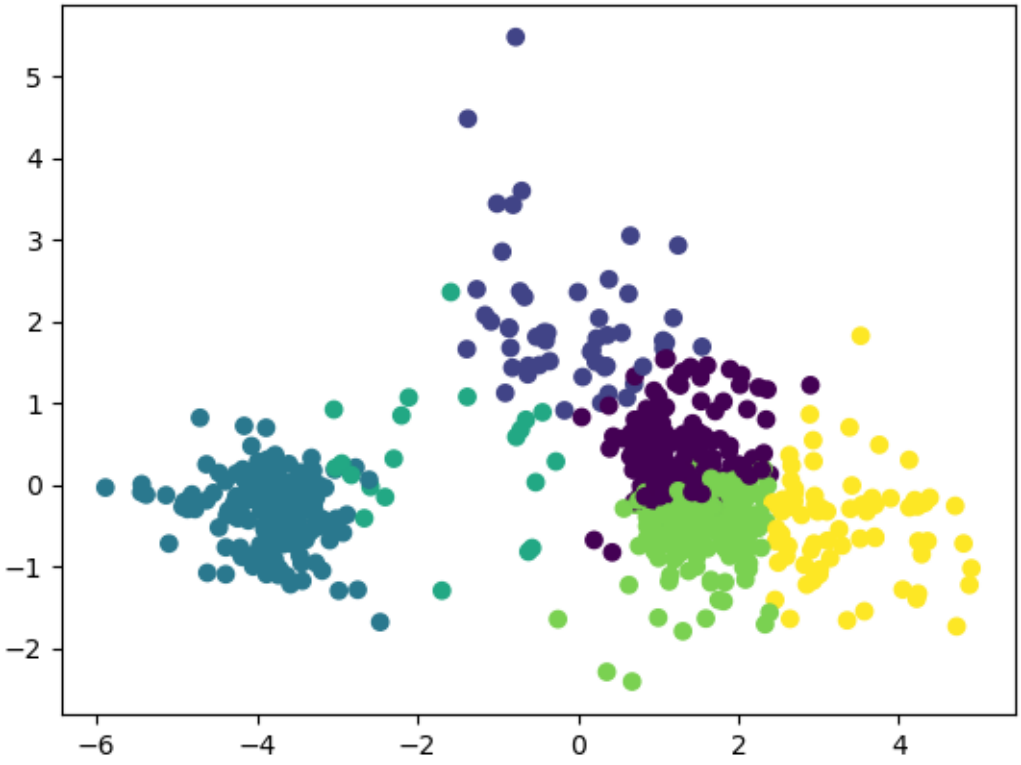
11/04/2019

ASSIGNMENT 3: CLUSTERING

k-Means Cluster Centroids: YeastGene Dataset

-0.2413	-0.1288	0.0623	0.1734	0.2179	1.6517	1.9053
-0.9535	-1.4716	0.0775	-0.1795	-1.0048	1.1512	0.9688
0.1651	0.0917	-0.1039	-0.5526	-0.6301	-1.7232	-1.7548
0.0233	0.2508	-0.2770	-0.3640	-0.7354	-0.8946	0.7001
-0.0016	0.1565	0.3563	0.7016	1.0097	1.8423	1.6434
-0.0393	0.1539	0.4361	1.1058	1.4487	3.0163	2.8294

PCA of Clusters: YeastGene Dataset



Hierarchical Clustering: Utilities

Merging	Cluster1	Cluster2	New Cluster
0	12	21	23
1	10	13	24
2	4	24	25
3	7	23	26
4	20	25	27
5	14	19	28
6	1	18	29
7	15	26	30
8	28	29	31
9	2	27	32
10	8	16	33
11	30	32	34
12	22	34	35
13	9	31	36
14	35	36	37
15	6	37	38
16	3	38	39
17	33	39	40
18	17	40	41
19	11	41	42
20	5	42	43

Code: k-Means Clustering

Assign Cluster

```
def assignCluster(dataSet, k, centroids):
    clusterAssment = [0] * dataSet.shape[0]

    for i in range(0, dataSet.shape[0]):
        minDist = float("inf")
        for j in range(0, centroids.shape[0]):
            # Euclidean sqrt((y1-x1)^2 + ... + ((yn-xn)^2)
            euclDist = math.sqrt(np.sum(np.square(dataSet[i] - centroids[j])))
            if(euclDist < minDist):
                minDist = euclDist
                clusterAssment[i] = j
    return clusterAssment
```

Get Centroid

```
def getCentroid(dataSet, k, clusterAssment):
    # centroids.reshape((k, dataSet.shape[1]))
    centroids = np.mat(np.zeros((k, dataSet.shape[1]))) # array of new cluster centroids
    dpInCluster = np.zeros((k, 1)) # Number of datapoints in a given cluser

    # Compute the new centroids as average of all points within the corresponding cluster
    for i in range(0, dataSet.shape[0]): # Take the sum of all points within the cluster
        centroids[clusterAssment[i]] += dataSet[i]
        dpInCluster[clusterAssment[i]] += 1
    centroids /= dpInCluster # Divide by the number of points in the cluster to get average
    return centroids
```

Code: Hierarchical Clustering

Merge Cluster

```
def merge_cluster(distance_matrix, cluster_candidate, T):
    merge_list = []
    minDist = np.min(distance_matrix, axis=None) # find the minimum distance in the array
    minIndex = np.where(distance_matrix == minDist)[0] # find the fist occurrence of min value

    # Indices of minimum distance
    i = minIndex[0] # i-th row
    j = minIndex[1] # j-th column

    # The cluster IDs corresponding to the i,j row/cols
    clustID1 = rowToClust[i]
    clustID2 = rowToClust[j]

    # Get the points from each cluster
    points1 = cluster_candidate[clustID1]
    points2 = cluster_candidate[clustID2]
    newPoints = points1 + points2

    # Remove the old clusters and add a new merged cluser
    del cluster_candidate[clustID1]
    del cluster_candidate[clustID2]
    cluster_candidate[T] = newPoints

    # Record which clusters were merged
    merge_list = [(clustID1, points1), (clustID2, points2)]

    return cluster_candidate, merge_list
```

Update Distance

```
def update_distance(distance_matrix, cluster_candidate, merge_list, T):
    # Get which clusters were merged
    clustID1 = merge_list[0][0]
    clustID2 = merge_list[1][0]

    # Get the corresponding row/col values
    global rowToClust
    i = min(rowToClust.index(clustID1), rowToClust.index(clustID2))
    j = max(rowToClust.index(clustID1), rowToClust.index(clustID2))

    # Calculate the new distance between each cluster and the merged cluster
    newDists = {}
    for row in range(0, distance_matrix.shape[0]):
        if (row==i or row==j): continue
        newDists[rowToClust[row]] =
         min(distance_matrix[row][i], distance_matrix[row][j])

    # Remove j-th row and update rowToClust mapping
    newDistMat = np.delete(distance_matrix, j, axis=0)
    for idx in range(j, len(rowToClust)-1):
        rowToClust[idx] = rowToClust[idx+1]

    # Remove i-th row and update rowToClust mapping
    newDistMat = np.delete(newDistMat, i, axis=0)
    for idx in range(i, len(rowToClust)-1):
        rowToClust[idx] = rowToClust[idx+1]

    # Remove i-th and j-th cols, mapping already updated
    newDistMat = np.delete(newDistMat, j, axis=1)
    newDistMat = np.delete(newDistMat, i, axis=1)

    # Add a new row to the bottom and update rowToClust mapping
    newRow = [0] * newDistMat.shape[1]
    newDistMat = np.vstack((newDistMat, newRow))
    lastRow = newDistMat.shape[0] - 1

    # Fill the row with the new distances
    rowToClust[lastRow] = T
    for col in range(0, newDistMat.shape[1]):
        newDistMat[lastRow][col] = newDists[rowToClust[col]]

    # Add new col to right, as matrix is symmetric use the transpose of new row
    # Adding the self-self dist in bottom right corner
    newCol = np.append(newDistMat[lastRow], 100000)
    newDistMat = np.vstack((newDistMat.T, newCol)).T

    distance_matrix = newDistMat
    return distance_matrix
```