File: homework4.pdf
Author: P.J. Leyden
Date: November 13th, 2019

Homework 4

1)

Input: k = number of clusters | datapoints = set of data points
Output: data points will be labeled with their cluster id

```
k_means(k, datapoints):
    centroids[k] //array of size k for storing centroids
    for i = 0, i < k, ++i:
        centroids[i] = random point from datapoints
    bool significant_change = true
    while(significant_change)
        significant_change = false
        for each x in datapoints
            smallest_dist = dist(x, centroids[0])
            smallest_dist_id = 0
            cluster_id = 0
            for each c in centroids
                if(dist(x, c) < smallest_dist)
                    smallest_dist = dist(x, c)
                    smallest_dist_id = cluster_id
                cluster_id += 1
            if(x.label != smallest_id)
                significant_change = true
            x.label = smallest_id
end
```

2)

Note: Will be using a KDTree for quick nearest neighbour lookup
Input: k = number of clusters | datapoints = set of data points
Output: data points will be labeled with their cluster id

```
agglomerative(k, datapoints):
    kdtree kdt = kdtree(datapoints)
    while kdt.root.num_children > k:
        node1 = datapoints[0]
        node2 = datapoints[1]
        shortest_dist = dist(node1, node2)
        for each x in datapoints
            y = kdt.nearestNeighbour(x)
            if(dist(x, y) < shortest_dist)
                node1 = x; node2 = y
                shortest_dist = dist(x, y)
        z = merge(node1, node2)
        kdtree.remove(node1)
```

```
                kdtree.remove(node2)
                kdtree.add(z)
end
```

3)

```
Input: k = number of clusters | datapoints = set of data points
Output: data points wili be labeled with their cluster id

divisive(k, datapoints):
        num_clusters = 2
        k_means(2, datapoints) //initial split of the system
        while num_clusters < k
                x = random cluster id
                particular_cluster = datapoints.cluster(x)
                k_means(2, particular_cluster)
```

4)

Algorithm 18.1: Density-based Clustering Algorithm

```
dbscan (D, ε, minpts)    :
foreach x ∈ D do
    Compute N_ε(x)
    Classify x as core, border, or noise
id = 0
foreach x ∈ D, such that x is core and unmarked do
    id = id + 1
    DensityConnected(x, id)
return Clustering {D_i}_{i=1}^{id}, where D_i = {x ∈ D : x has label i}

DensityConnected (x, id):
Mark x with current cluster id
foreach y ∈ N_ε(x) do
    Mark y with current cluster id
    if y is core then
        DensityConnected(y, id)
```

Above is the pseudo-code for the DBSCAN Algorithm given in the slides. I will explain what is happening in this algorithm, step by step.

First – D is the set of all datapoints, epsilon(eps) is the radius for finding neighbors, minpts is the minimum number of close neighbors required to be a core.
Second – The first foreach loop iterates through all the points, calculates it's neighbors by finding all other points within the radius epsilon and labels them core, border or noise.

Core is for points that have minpts amount of points within eps.
Border is for points that have fewer than minpts but fall within the range of a core.
Noise is for points that have fewer than minpts and don't fall within range of a core.

Third – We then perform another foreach loop going through all points in D that are core and not labeled. Each one is given an id and sent to the DensityConnected function. This functions purpose is to label all points within core and border distance of the initial point with the same label. This process will label all core and border points that are close, giving us our density clusters.
Finally – We return the clusters.

Bonus:

C1 – {(2, 10), (2, 5), (8, 4)} : Centroid – (4, 6.33)
C2 – {(5, 8), (7, 5), (6, 4)} : Centroid – (6, 5.67)
C3 – {(1, 2), (4, 9)} : Centroid – (2.5, 5.5)