# CSCI 6333 Data Mining & Warehousing

**Module 4: Cluster Analysis**

**Homework Assignment Three**

**All problems are equal-weighted with 20 points each.**

**Name: **Ulvi Bajarani**, SID: **20539914****

1. Estimate the time complexity of the k-means clustering algorithm in terms of the dimension , the number of data points and the number of iterations ?

**Answer 1.** For data points equal to **n**, number of iterations equal to **I**, number of attributes equal to **d**, and number of clusters equal to **k**, the time complexity of k-means clustering equals to

**O(n\*k\*I\*d)**

1. Suppose that a set of -dimensional points has pairwise disjoint clusters . Show how to find the centroids , to minimize the sum of squared errors SSE below

where is the Euclidean distance and .

**Answer 2.** For each data point **x** in cluster Ci with mi objects, we can calculate the new centroid of Ci by the formula:



1. It seems that the k-means clustering algorithm is simple and runs linearly in the size of the data set at each iteration. Explain why the k-mean clustering problem, when regarded as optimization problem to minimize the sum of squared errors, is computationally hard?

**Answer 3.** Since we initiate the algorithm with the given number of clusters for the given number of attributes, the results lead to the local optimum (*the optimal solution for the provided sets of values.*). The calculating global optimum (*the optimal solution for all sets of values. In our case, the number of clusters, the given number of attributes, and the number of data points*) is the NP-Hard problem, since there is no known polynomial-time algorithm. The problem is solved in polynomial time for n-data points when *k* (the number of clusters) and *d* (the number of attributes/dimensions) are fixed. But in this case, the time complexity is O(ndk+1) [see **Applications of weighted Voronoi diagrams and randomization to variance-based *k*-clustering**].

1. In hierarchical clustering, a proximity matrix is maintained for all clusters. Initially, each single point is viewed as a cluster. In addition to the proximity matrix, we also maintain a sorted list of all entries in the matrix. Explain in detail at each step of merging two closest clusters, how the proximity matrix can be updated in time? How the sorted list of the matrix entries can be updated in time? Here, is the dimension and is the number of points. Assume we use the MAX distance as the distance between two clusters.

**Answer 4.** After the first calculation, we will have NxN proximity matrix, which could be stored in 1/2\*N2 space. Then, we sort all proximity values in the sorted list, which will take O(N\*logN) time. After that, we can start to merge clusters. It is done in these steps:

1) Find the maximum distance in the sorted list.

2) Merge two clusters having such distance. The new cluster will be C = C1 ∪ C2;

3) Delete all proximity records containing either C1 or C2 (or both)

4) Add C to the proximity matrix.

5) Calculate the maximum distance between all clusters and the new cluster. For each cluster Ci, it is done by d(C, Ci)=MAX(d(C1, Ci), d(C2, Ci))

6) Sort the list with proximity values.

Since for the each dimension *d* we can have own matrix, updating the proximity matrix takes O(*d*N). Removing the values takes O(n) time [due to the linear search], adding new values to the list takes O(1) time, and sorting takes O(N\*logN) time. So, updating the list with distances takes O(N\*logN) time.

1. Given two parameters Eps and MinPts, give a simple algorithm to find all core points, border points and noise points. Estimate the time complexity of your algorithm in terms of dimension and the number of points . Can you find a more efficient algorithm?

**Answer 5.** Here is the simple DBSCAN algorithm to find the all core, border, and noise points:

set **cluster\_label** = 0;

for each point P in the set of points PS:

***if*** point P has the label ***then*** continue;

set N equal to the neighbors of P;

***if*** |N| < MinPts:

set label of P as “Noise”;

continue;

set **cluster\_label** += 1;

set label of P as **cluster\_label;**

set **corePoints** = N \ {P};

for each point Q in the set of neighbors **corePoints**:

***if*** the label of Q is “Noise” ***then*** set the label of Q as **cluster\_label;**

***if*** point Q has the label ***then*** continue;

set label of P as **cluster\_label;**

set N equal to the neighbors of Q;

***if*** |N| ≥ MinPts:

set **corePoints** = **corePoints** ∪ N

The complexity of the algorithm above in the worst case is O(*d*\*N2), where *d* is the number of dimensions. It might be reduced to O(*d*\*N\*logN), if the proper data structure is used. Also, DSDBSCAN algorithm is also possible, which uses disjoint-set data structure:

for each point P in the set of points PS:

Create the tree T(P) = P;

for each point P in the set of points PS:

set N equal to the neighbors of P;

***if*** |N| ≥ MinPts:

set P as a core point;

for each point P’ in the set of neighbors N:

if P’ is a core point:

Union (P, P’);

else if P’ is not labeled:

set P’ as the “Border Point”;

Union (P, P’);