CS4035 - Cyber Data Analytics | Assignment 3: Data Stream Mining

By : Hendra Hadhil Choiri (4468457)
Prahesa Kusuma Setia (4501543)

k-Means

For k-Means implementation, `cluster_assignment` and `get_centroid` functions have been created. To test this algorithm, we created test_kmeans script. For initial centroids, instead of using pre-defined fixed centroids, we pick them randomly from the points in dataset (by using `gendat` function from prtools). For every test in this assignment, we use k=6 and max_iter=15. For each iteration, the object points are assigned to nearest centroid (by using `cluster_assignment`), and then new centroids are calculated (by using `get_centroid`). From 10 executions, the average time is 3.30 s and the standard deviation is 0.20. Some plots of various result examples are shown in Figure 1a-1d in Appendix.
From the figures, we can see that the resulted clusters may vary based on selected initial centroids. Visually, we may think that the clustering in Figure 1a is more preferred. However if there are some initial centroids that lie on the same "high density" area, this area may be separated into different clusters. For example, yellow cluster in Figure 1a becomes 2 clusters in Figure 1b (red and green). However, we cannot say that clustering in 1b-1d are bad because actually there are 6 dimensions to be considered, and the figures only show the scatterplot in 2 dimensions.
For MapReduce and MPI k-Means, the resulted clustering is not much differs from these Figures, so they will not be plotted again.

Distributed k-Means

MapReduce k-Means

In MapReduce version of k-Means, we implement the Mapper in `kmeans_map` and Reducer in `kmeans_reduce`. Basically what they do is each Mapper calculate its local centroid of the data and initial centroid given to them and they create key-value mapping of each centroid and its member point assignment, each Reducer then has responsibility to calculate the average centroid of each centroid information given by each Mapper (by the shuffler). After executing 10 times by using the same configuration, we get μ_{time} = 1.22 s and σ_{time} = 0.20. By using parallel computing toolbox, we get μ_{time} = 5.81 s and σ_{time} = 2.05.

Message Passing Interface (MPI) k-Means

To implement MPI k-Means, `kmeans_mpi_slave` function is created to apply 1 iteration of k-Means in a slave group. Then the local centroids from each slave are combined (weighted averaged) by using `kmeans_mpi_master` function. After executing 10 times by using the same configuration, we get μ_{time} = 1.25 s and σ_{time} = 0.11. By using parallel computing toolbox, we get μ_{time} = 3.22 s and σ_{time} = 0.30.

P2P k-Means

In P2P version of the k-Means we already been given a template to work on, we only need to implement the node function in the `kmeans_p2p_node` file that basically compute new centroids based on given data and initial centroids value. After executing 10 times by using the same configuration, we get μ_{time} = 1.65 s and σ_{time} = 0.66. However, in some iterations, there is possibility that a centroid does not have any member. So when combined, this cluster is undefined. Finally there are only 2 clusters in a node as shown in Figure 2 of Appendix.

The maximum deviations per iteration are in order: 67.9840 (iter 1), 99.7922, 64.9591, 65.2208, 72.0503 (iter 5), 16.1357, 14.7388, 9.0444, 4.2198, 4.0828 (iter 10), 4.0828, 4.0828, 4.0828, 4.0828, 4.0828 (iter 15). We can observe that initially there are high deviations because the initial centroids in each node are different and may scattered in very different positions (e.g. centroid 1 in node a is at top left, but centroid 1 in node b is at bottom right). Then for each iteration, the centroids in a node are influenced by the centroids in neighboring nodes, the max deviation is decreasing. After 10 iterations, each node gets influence from all of the nodes in the network (as there are 10 nodes in connected graph). Then the maximum deviation is converged.

Brief Discussion

Parallel vs Centralized

From the experiments, we can observe that by using distributed approach, the clustering can be done faster (from >3 seconds for centralized k-Means, then becomes < 2 seconds after using distributed). By using distributed approach, the computing load is divided into several locations parallelly. Moreover, each mapper/slave/node has less data so the k-Means clustering can be done faster locally.

However, when simulating distributed process by using Parallel Computing Toolbox, the execution time become longer. This may happens because in our experiment we only used local parallel cluster environment as opposed to use truly clustered computing environment. When we use local parallel cluster environment we can only split the task into a number of parallel task that is equal to the number of our processor's physical core, the performance may also downgraded because we use our local machine to do other processing tasks (for example: opening web browser and such). If we had a chance to test it using truly clustered environment, like using HPC or Amazon EC2 for example, then the performance might be better because they are specially built for that purpose.

MapReduce vs MPI

MapReduce and MPI are quite similar in some parts. The whole data are distributed into several locations, called mappers in MapReduce and slaves in MPI. In MapReduce, for each iteration the mappers will read the same data again and again, so it has more burden to read the dataset. Conversely in MPI, the data portions are assigned to each slave so it only requires to read the data once.

MPI vs P2P

P2P needs more iterations to create converged clusters because each node can only influence neighboring nodes in one iteration. But, this approach is very scalable and can be applied for very large number of nodes. For MPI, the master can combine directly the local centroids from all slaves so the clustering can be converged in less iterations. However, if the number of slaves is too many, the master may be too burdened to handle all of them and may cause bottleneck.

Cost Analysis

To make uniform initial centroids in all variants, we use (Km+1)'th points as initial centroids (default as given in the code skeleton). The cost of centralized, MapReduce, and MPI k-Means are 8.278×10^7, 8.709×10^7, and 8.738×10^7 respectively. We can see that by using distributed k-Means, the cost becomes higher because each server only gives information of local centroids (that assumed can represent each of local data). However, in centralized k-Means the information of the whole data are known so the centroids can be computed more accurately.

For P2P k-Means, the centroids are initialized differently for each node, and during the iterations, some of the centroids do not have any member so the number of resulted clusters is less than 6. In this condition, the cost value becomes unavailable.

Appendix

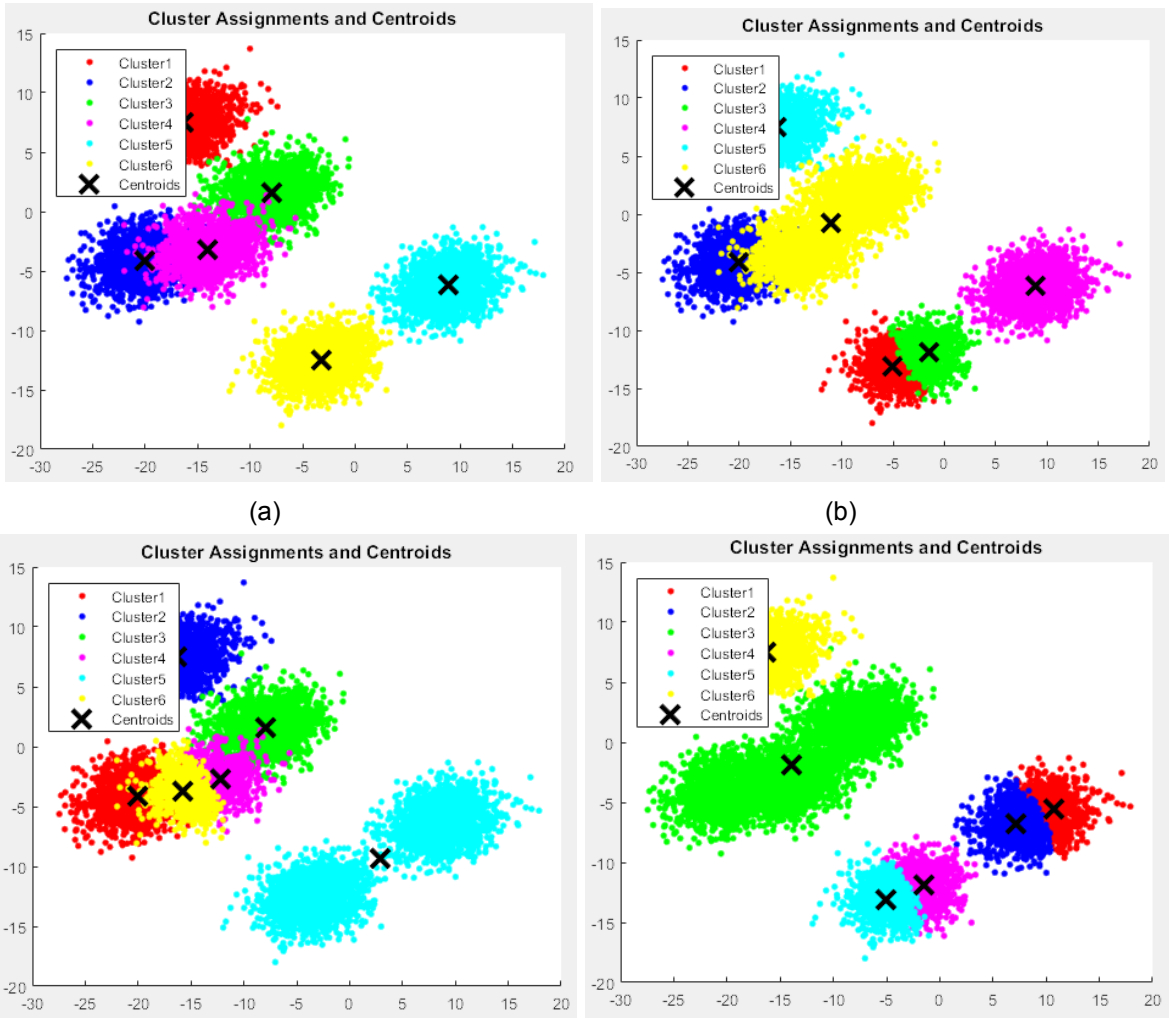

Figure 1. Various clustering results by using normal k-Means

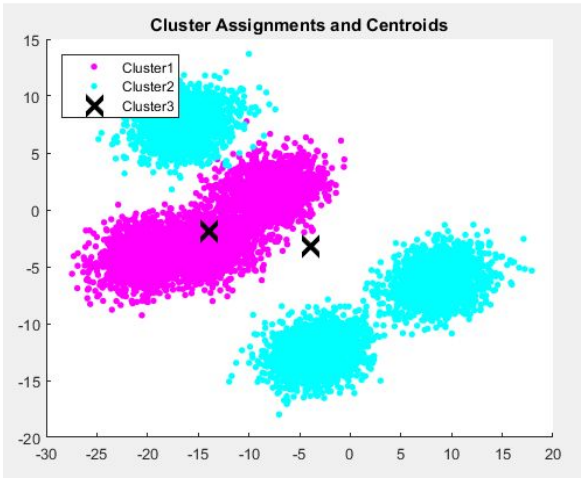

Figure 2. Clustering results by using P2P k-Means