**Fuzzy Clustering**

1. **Problem Description**

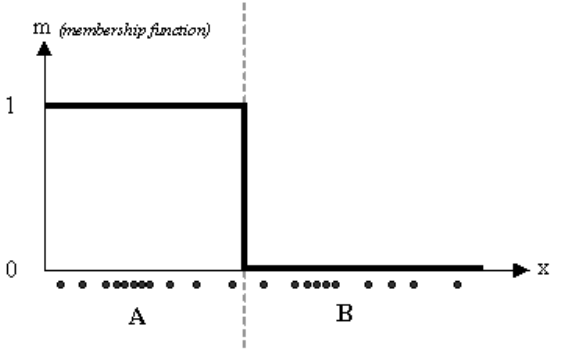
Cluster analysis or clustering is the task of grouping a set of data objects in such a way that data objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters).

Data are bound to each cluster by means of a Membership Function, which represents the fuzzy behaviour of this algorithm.

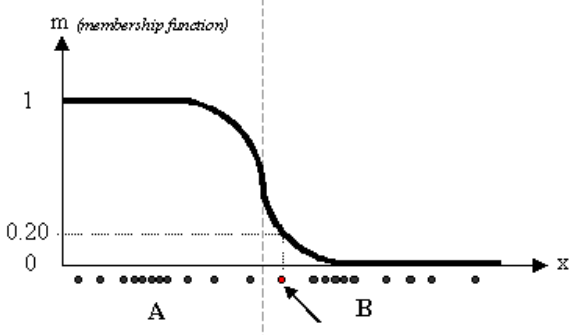
For a better understanding, we may consider this simple mono-dimensional example. Given a certain data set, suppose to represent it as distributed on an axis. The figure below shows this:



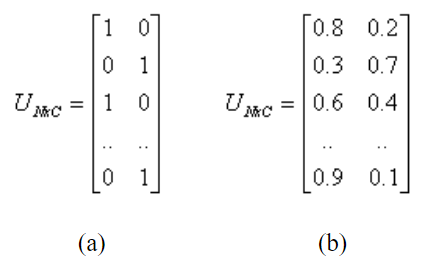
Looking at the picture, we may identify two clusters in proximity of the two data concentrations. We will refer to them using ‘A’ and ‘B’. We associated each datum to a specific centroid; therefore, this membership function looked like this:



In the FCM approach, instead, the same given datum does not belong exclusively to a well defined cluster, but it can be placed in a middle way. In this case, the membership function follows a smoother line to indicate that every datum may belong to several clusters with different values of the membership coefficient.

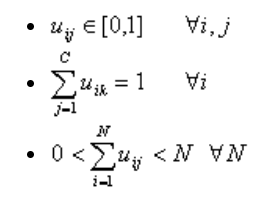


In the figure above, the datum shown as a red marked spot belongs more to the B cluster rather than the A cluster. The value 0.2 of ‘m’ indicates the degree of membership to A for such datum. Now, instead of using a graphical representation, we introduce a matrix U whose factors are the ones taken from the membership functions:



The number of rows and columns depends on how many data and clusters we are considering. More exactly we have C = 2 columns (C = 2 clusters) and N rows, where C is the total number of clusters and N is the total number of data. The generic element is so indicated: *uij*.

In the examples above we have considered the k-means (a) and FCM (b) cases. We can notice that in the first case (a) the coefficients are always unitary. It is so to indicate the fact that each datum can belong only to one cluster. Other properties are shown below:

****

1. **Background**

Fuzzy c-Means Algorithm. The fuzzy c-means (FCM) algorithm is a clustering algorithm .It is useful when the required number of clusters are pre-determined; thus, the algorithm tries to put each of the data points to one of the clusters. What makes FCM different is that it does not decide the absolute membership of a data point to a given cluster; instead, it calculates the likelihood (the degree of membership) that a data point will belong to that cluster..

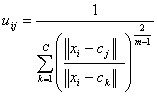
1. **Description of the own solution**

Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters. This method (developed by [Dunn in 1973](http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html#dunn) and improved by [Bezdek in 1981](http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html#bezdek)) is frequently used in pattern recognition. It is based on minimization of the following objective function:

http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image019.gif     ,     http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image021.gif

where *m* is any real number greater than 1, *uij* is the degree of membership of *xi* in the cluster *j*, *xi* is the *i*th of d-dimensional measured data, *cj* is the d-dimension center of the cluster, and ||\*|| is any norm expressing the similarity between any measured data and the center.

Fuzzy partitioning is carried out through an iterative optimization of the objective function shown above, with the update of membership *uij* and the cluster centers *cj* by:

     ,     

This iteration will stop when http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image027.gif, where http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image002.gif is a termination criterion between 0 and 1, whereas *k* are the iteration steps. This procedure converges to a local minimum or a saddle point of *Jm*.  
The algorithm is composed of the following steps:

|  |
| --- |
| 1. *Initialize U=[uij] matrix, U(0)* 2. *At k-step: calculate the centers vectors C(k)=[cj] with U(k) http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image025.gif* 3. *Update U(k) , U(k+1) http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image023.gif* 4. *If || U(k+1) - U(k)||<http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image002.gif then STOP; otherwise return to step 2.* |

**Step 01:**

for (i = 0; i < num\_data\_points; i++) {

sum = 0.0;

p = 100;

for (j = 1; j < num\_clusters; j++) {

randomVal = rand() % (p + 1);

p -= randomVal;

degreeOfMemb[i][j] = randomVal / 100.0;

sum += degreeOfMemb[i][j];

}

degreeOfMemb[i][0] = 1.0 - sum;

}

**Step 02:**

// Calculate array of power degree of member of fuzziness

for (i = 0; i < num\_data\_points; i++) {

for (j = 0; j < num\_clusters; j++) {

temp[i][j] = pow(degreeOfMemb[i][j], fuzziness);

}

}

// Calculate cluster centre

for (j = 0; j < num\_clusters; j++) {

for (k = 0; k < num\_dimensions; k++) {

numerator = 0.0;

denominator = 0.0;

for (i = 0; i < num\_data\_points; i++) {

numerator += temp[i][j] \* dataPoint[i][k];

denominator += temp[i][j];

}

clusterCentre[j][k] = numerator / denominator;

}

}

**Step 03:**

// Loop for each cluster and data points then get maximum difference in degree of membership

for (j = 0; j < num\_clusters; j++) {

for (i = 0; i < num\_data\_points; i++) {

newDoM = getNewDoM(i, j);

diff = newDoM - degreeOfMemb[i][j];

if (diff > max\_difference)

max\_difference = diff;

degreeOfMemb[i][j] = newDoM;

}

}

return max\_difference;

**step 04:**

max\_difference > epsilon

1. **Algorithmic complexity and correctness analysis**
2. **Complexity**

Firstly we will analyze the  complexity of each step separately

Keeping the number of data points constant we may assume that n = 300, d = 3, i = 20 and varying number of clusters where n = number of data points, c = number of cluster, d = number of dimension and i = number of iterations.

The number of iterations in this case is defined by condtion max\_difference less than epsilon. Which epsilon = 0.005

**Step 1**

It is very clear that the complexity of this step is O(nc).

**Step 2**

In order to calculate center of each clusters this  algorithm have complexity is  O(ncd)

**Step 3**

Two loops interiate though number of clusters and number of data points. So complexity of this step is O(nc2di)

**Step 4**

Two loop interiate though number of clusters and number of data points. So complexity of this step is O(n)

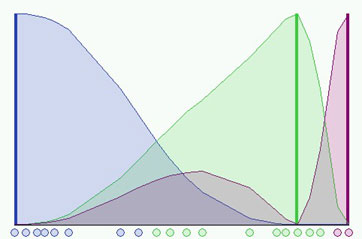
Conclusion: Total complexity of this algorithm is O(nc2di)

1. **Quality Analysis**

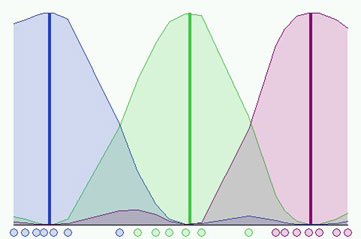
Explaining how the quality depends on the values of input parameters.

**Example:**

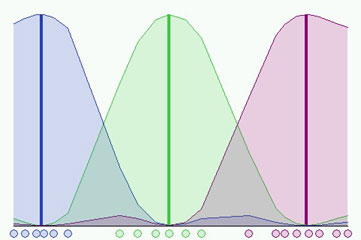
Here, we consider the simple case of a mono-dimensional application of the FCM. Twenty data and three clusters are used to initialize the algorithm and to compute the U matrix. Figures below show the membership value for each datum and for each cluster. The color of the data is that of the nearest cluster according to the membership function.



In the simulation shown in the figure above we have used a fuzzyness coefficient m = 2 and we have also imposed to terminate the algorithm when http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image048.gif. The picture shows the initial condition where the fuzzy distribution depends on the particular position of the clusters. No step is performed yet so that clusters are not identified very well. Now we can run the algorithm until the stop condition is verified. The figure below shows the final condition reached at the 8th step with m=2 and http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image002.gif=0.3:



Is it possible to do better? Certainly, we could use an higher accuracy but we would have also to pay for a bigger computational effort. In the next figure we can see a better result having used the same initial conditions and http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image002.gif=0.01, but we needed 37 steps!



It is also important to notice that different initializations cause different evolutions of the algorithm. In fact it could converge to the same result but probably with a different number of iteration steps.

The quality of the algorithms depend on the epsilon value, the more small epsilon the higher accuracy of the algorithm. But there will be a trade of for more computational effort with more steps need to be run.

1. **User Manual**

**Step 01:**

Run DataMining\_FuzzyClustering.exe in folder below:

C:\Users\Daniel\Documents\Visual Studio 2015\Projects\DataMining\_FuzzyClustering\Debug

**Or** open Visual Studio 2015 solution and reconfigure the numbers of data points, clusters, dimension, fuzziness and epsilon. Then run the debug.

*These are maximum dataset, which application can run*

*#define MAX\_DATA\_POINTS 1000*

*#define MAX\_CLUSTERS 100*

*#define MAX\_DATA\_DIMENSION 100*

In source code, you can change these parameter for your test.

*int num\_data\_points = 1000; // Number of data points*

*int num\_clusters = 20; // Number of clusters*

*int num\_dimensions = 100; // Number of dimension*

*int num\_sparity = 10000; // Level of sparity of data points - sparity is the the range of specific cordinate on 1 axis of a certain point on a dimension. For example: in this case, datapoint p with cordinates p[i0][i1]…[in] with 0< i0, i1,…,in <10000 .*

*int fuzziness = 2; // Fuzzyness coefficient 1 < fuzziness (m)*

*double epsilon = 0.000005; // Termination criterion: 0.0 =< epsilon <= 1.0*

Notes: These tests will create some file

Initial\_membership.txt, input.txt, membership.txt, Cluster\_Num[x].txt, report.txt

**Step 02:**

**Input:**

This application will automatically generate **input.txt** file. Which contains cordinates of all points according to clusters.

It’s also create **Initial\_membership.txt** which contain the degree of membership of point i to cluster j in the first time.

**Output:**

After running the FCM algorithm, it will create:

A **membership.txt** file, which describes the latest update of membership of point i to cluster j. This file will be create at the end of all the test running.

Numbers of cluster file with name: **Cluster\_Num[x].txt** (x is the index of cluster file) denote points belong to which cluster file. Note, this creation will consume a lot of time for running. So I comment out for better testing.

//// Create cluster files

//for (int i = 0; i < num\_clusters; i++) {

// createClusterFile(clusterFilePath(i));

//}

//// Write data to cluster files

//writeNewLineDataToClusterFiles();

Report information will be write to **report.txt file**. Which will record the fuzziness, epsilon, runningTime and steps for each line.

1. **Technical documentation**

Main procedures:

// Main function of fuzzy C-means

int fuzzyCMeans(int fuzziness, double epsilon) {

double max\_difference;

// Initialize degree of membership of a given data point i to cluster j.

initializeDoM(fuzziness, epsilon);

// Create Initial\_Membership of points i to cluster j

createMembershipFile(fileMembership("Initial\_Mebership.txt"));

do {

// Recalculate the centre vector

calculateCentreVectors(fuzziness);

// Update new degree of membership

max\_difference = updateDoM(fuzziness);

} while (max\_difference > epsilon); // Termination contidion

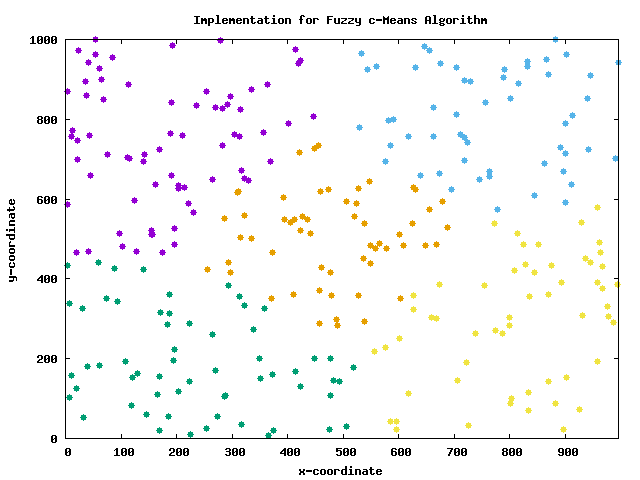
return 0;

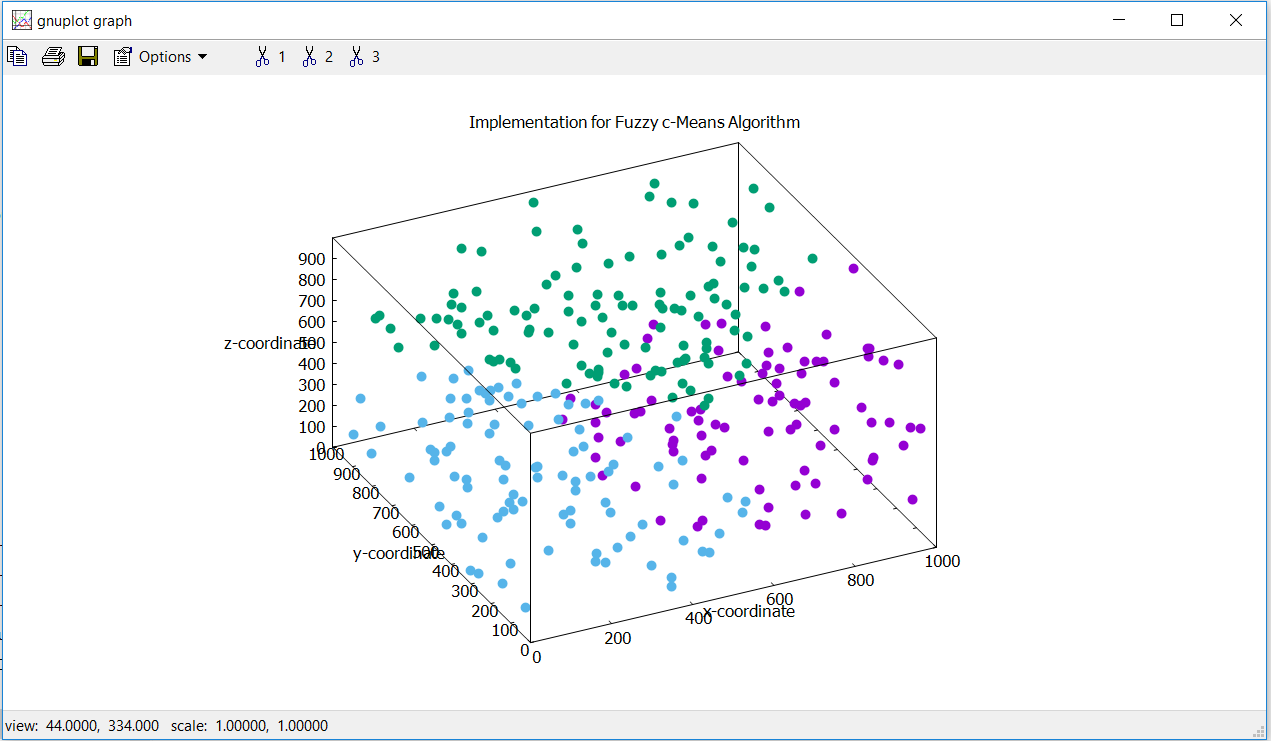
}

1. **Description of tests**
   1. **Normal points dataset**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| numOfDataPoints | numOfCluster | m | dimension | epsilon |
| 300 | 3 | 2 | 3 | 0.005 |
| 300 | 5 | 2 | 2 | 0.005 |

Please check result at the plot below for 2D and 3D





* 1. **Large dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| numOfDataPoints | numOfCluster | m | dimension | Epsilon | Time (ms) |
| 1000 | 5 | [2,6] | 10 | [0.000005 , 0.005] | 47-425 |

After running this largedataset, running time have a range from 47 to 425 milliseconds.

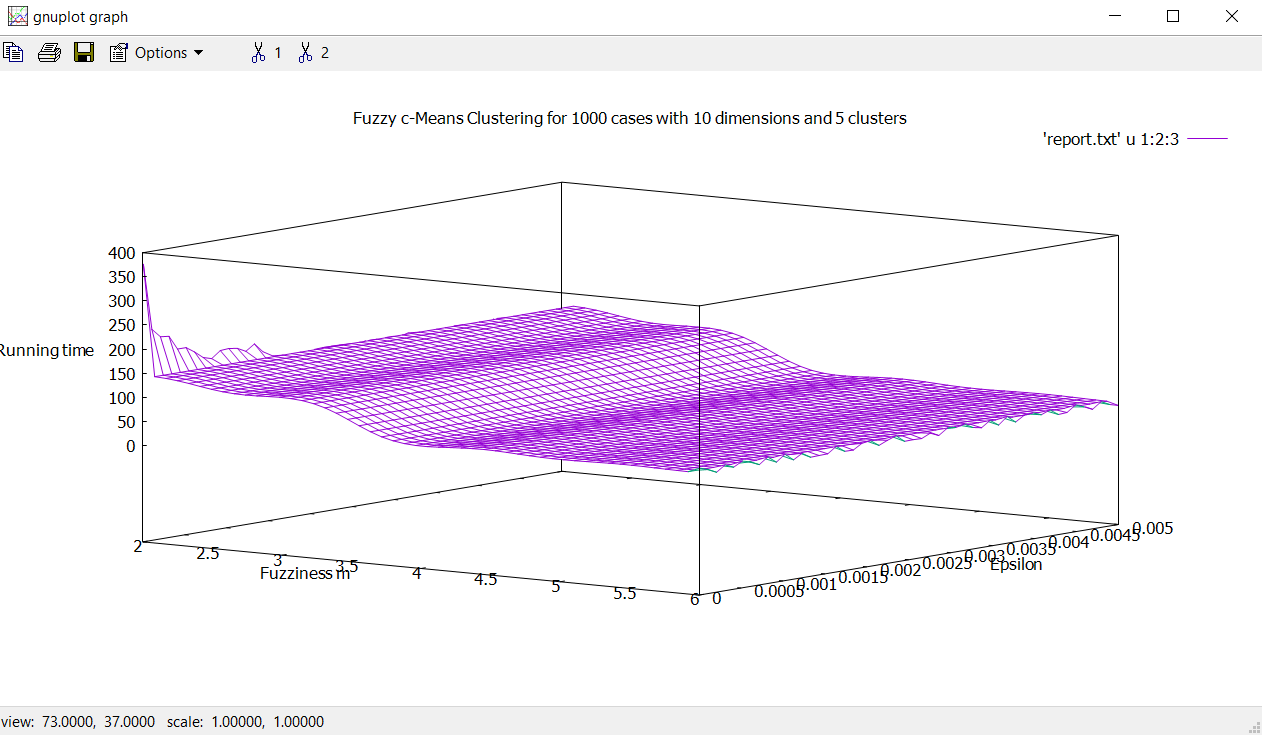
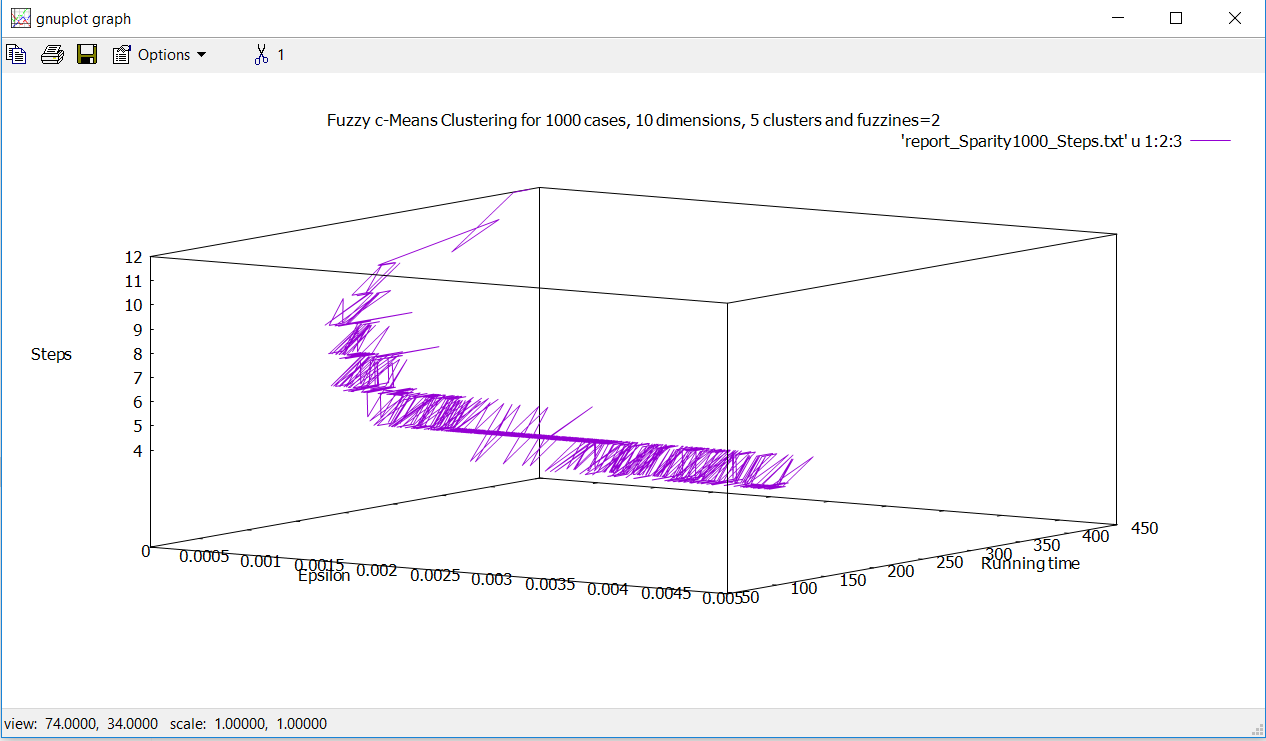


Figure 01

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| numOfDataPoints | numOfCluster | m | dimension | Epsilon | Steps |
| 1000 | 5 | 2 | 10 | [0.000005 , 0.005] | 4-12 |

****

* 1. **Huge dataset**

These tests use fixed dimension and epsilon with changing in number of cluster. The more clusters the more time needed to run as table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| numOfDataPoints | numOfCluster | m | dimension | Epsilon | Time |
| 1000 | 12 | 2 | 100 | 0.00005 | 23 minutes |
| 1000 | 20 | 2 | 100 | 0.00005 | 36 minutes |
| 1000 | 100 | 2 | 100 | 0.00005 | 54 minutes |

These tests use only one dynamic parameters is m – fuzziness have the value from 2 to 6. Others parameters are not changed as the table. As you can see, increase m will increase computational time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| numOfDataPoints | numOfCluster | m | dimension | Epsilon | Time (ms) |
| 1000 | 5 | 2 | 100 | 0.00005 | 1045 |
| 1000 | 5 | 3 | 100 | 0.00005 | 1019 |
| 1000 | 5 | 4 | 100 | 0.00005 | 526 |
| 1000 | 5 | 5 | 100 | 0.00005 | 505 |
| 1000 | 5 | 6 | 100 | 0.00005 | 536 |

These tests use only one dynamic parameters is epsilon have the value from 0.0000005 to 0.000256. Others parameters are not changed as the table below. My conclusion is that, the smallest epsilon is, the cost of running time and steps increase accordingly.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| numOfDataPoints | numOfCluster | m | dimension | Epsilon | Time (ms) | Steps |
| 1000 | 5 | 2 | 100 | 0.0000005 | 1437 | 6 |
| 1000 | 5 | 2 | 100 | 0.000001 | 1473 | 6 |
| 1000 | 5 | 2 | 100 | 0.000002 | 1203 | 5 |
| 1000 | 5 | 2 | 100 | 0.000004 | 1281 | 5 |
| 1000 | 5 | 2 | 100 | 0.000008 | 1282 | 5 |
| 1000 | 5 | 2 | 100 | 0.000016 | 1234 | 5 |
| 1000 | 5 | 2 | 100 | 0.000032 | 1204 | 5 |
| 1000 | 5 | 2 | 100 | 0.000064 | 1016 | 4 |
| 1000 | 5 | 2 | 100 | 0.000128 | 969 | 4 |
| 1000 | 5 | 2 | 100 | 0.000256 | 969 | 4 |

1. **Conclusion and comments**

As the largedata set as below:

Unchanged (fixed) parameters:

Number of points = 1000 points

Number of dimension = 10 dimensions

Number of cluster = 5

Changed (Dynamic) parameters:

Fuzziness m: from 2 to 6

Epsilon from 0.00005 to 0.005

As you can see in the Figure 01, the changed in epsilon does not change much running time with the same fuzziness. But the running time will decrease steadily from 150ms to 50ms according increasing in fuzziness. The more fuzziness, the faster algorithm run.

In order to identify the effect of changing epsilon, I try to count the steps of running for each cases. And the result can be review that, with the same fuzziness = 2, incresing epsilon using the algorithm will reduce the steps need to be run. (please check file “report\_Sparity1000\_Steps.txt” for more information). Or you can say that with higher accuracy (minimize epsilon), we would have also to pay for a bigger computational effort (more steps).

1. **List of references to literature, web pages**

https://www.researchgate.net/file.PostFileLoader.html?id=57a26d35217e209ac366eec1&assetKey=AS%3A391123769020417%401470262581196

http://ieeexplore.ieee.org/document/7796188/

http://www.sciencedirect.com/science/article/pii/089571779390202A

http://home.deib.polimi.it/matteucc/Clustering/tutorial\_html/cmeans.html

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1516160

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.7600&rep=rep1&type=pdf>

1. **Members of this project**

Only Duy Nguyen Ngoc for this project