A Comparative Study of Data Clustering Techniques

Khaled Hammouda
Prof. Fakhreddine Karray
University of Waterloo, Ontario, Canada

Abstract – **Data clustering is a process of putting similar data into groups. A clustering algorithm partitions a data set into several groups such that the similarity within a group is larger than among groups. This paper reviews four of the most representative off-line clustering techniques: K-means clustering, Fuzzy C-means clustering, Mountain clustering, and Subtractive clustering. The techniques are implemented and tested against a medical problem of heart disease diagnosis. Performance and accuracy of the four techniques are presented and compared.**

Index Terms—data clustering, k-means, fuzzy c-means, mountain, subtractive.

I. INTRODUCTION

DATA CLUSTERING is considered an interesting approach for finding similarities in data and putting similar data into groups. Clustering partitions a data set into several groups such that the similarity within a group is larger than that among groups [1]. The idea of data grouping, or clustering, is simple in its nature and is close to the human way of thinking; whenever we are presented with a large amount of data, we usually tend to summarize this huge number of data into a small number of groups or categories in order to further facilitate its analysis. Moreover, most of the data collected in many problems seem to have some inherent properties that lend themselves to natural groupings. Nevertheless, finding these groupings or trying to categorize the data is not a simple task for humans unless the data is of low dimensionality (two or three dimensions at maximum.) This is why some methods in soft computing have been proposed to solve this kind of problem. Those methods are called "Data Clustering Methods" and they are the subject of this paper.

Clustering algorithms are used extensively not only to organize and categorize data, but are also useful for data compression and model construction. By finding similarities in data, one can represent similar data with fewer symbols for example. Also if we can find groups of data, we can build a model of the problem based on those groupings.

Another reason for clustering is to discover relevance knowledge in data. Francisco Azuaje *et al.* [2] implemented a Case Based Reasoning (CBR) system based on a Growing Cell Structure (GCS) model. Data can be stored in a knowledge base that is indexed or categorized by cases; this is what is called a Case Base. Each group of cases is assigned to a certain category. Using a Growing Cell Structure (GCS) data can be added or removed based on the learning scheme used. Later when a query is presented to the model, the system retrieves the most relevant cases from the case base depending on how *close* those cases are to the query.

In this paper, four of the most representative off-line clustering techniques are reviewed:

- K-means (or Hard C-means) Clustering,
- Fuzzy C-means Clustering,
- Mountain Clustering, and
- Subtractive Clustering.

These techniques are usually used in conjunction with radial basis function networks (RBFNs) and Fuzzy Modeling. Those four techniques are implemented and tested against a medical diagnosis problem for heart disease. The results

K. M. Hammouda,
Department of Systems Design Engineering,
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

are presented with a comprehensive comparison of the different techniques and the effect of different parameters in the process.

The remainder of the paper is organized as follows. Section II presents an overview of data clustering and the underlying concepts. Section III presents each of the four clustering techniques in detail along with the underlying mathematical foundations. Section IV introduces the implementation of the techniques and goes over the results of each technique, followed by a comparison of the results. A brief conclusion is presented in Section V. The MATLAB code listing of the four clustering techniques can be found in the appendix.

II. DATA CLUSTERING OVERVIEW

As mentioned earlier, data clustering is concerned with the partitioning of a data set into several groups such that the similarity within a group is larger than that among groups. This implies that the data set to be partitioned has to have an inherent grouping to some extent; otherwise if the data is uniformly distributed, trying to find clusters of data will fail, or will lead to artificially introduced partitions. Another problem that may arise is the overlapping of data groups. Overlapping groupings sometimes reduce the efficiency of the clustering method, and this reduction is proportional to the amount of overlap between groupings.

Usually the techniques presented in this paper are used in conjunction with other sophisticated neural or fuzzy models. In particular, most of these techniques can be used as preprocessors for determining the initial locations for radial basis functions or fuzzy if-then rules.

The common approach of all the clustering techniques presented here is to find *cluster centers* that will represent each cluster. A cluster center is a way to tell where the heart of each cluster is located, so that later when presented with an input vector, the system can tell which cluster this vector belongs to by measuring a similarity metric between the input vector and al the cluster centers, and determining which cluster is the *nearest* or most similar one.

Some of the clustering techniques rely on knowing the number of clusters apriori. In that case the algorithm tries to partition the data into the given number of clusters. K-means and Fuzzy C-means clustering are of that type. In other cases it is not necessary to have the number of clusters known from the beginning; instead the algorithm starts by finding the first large cluster, and then goes to find the second, and so on. Mountain and Subtractive clustering are of that type. In both cases a problem of known cluster numbers can be applied; however if the number of clusters is not known, K-means and Fuzzy C-means clustering cannot be used.

Another aspect of clustering algorithms is their ability to be implemented in on-line or off-line mode. On-line clustering is a process in which each input vector is used to update the cluster centers according to this vector position. The system in this case *learns* where the cluster centers are by introducing new input every time. In off-line mode, the system is presented with a training data set, which is used to find the cluster centers by analyzing all the input vectors in the training set. Once the cluster centers are found they are fixed, and they are used later to classify new input vectors. The techniques presented here are of the off-line type. A brief overview of the four techniques is presented here. Full detailed discussion will follow in the next section.

The first technique is *K-means* clustering [6] (or *Hard C-means* clustering, as compared to *Fuzzy C-means* clustering.) This technique has been applied to a variety of areas, including image and speech data compression, [3, 4] data preprocessing for system modeling using radial basis function networks, and task decomposition in heterogeneous neural network architectures [5]. This algorithm relies on finding cluster centers by trying to minimize a cost function of dissimilarity (or distance) measure.

The second technique is *Fuzzy C-means* clustering, which was proposed by Bezdek in 1973 [1] as an improvement over earlier Hard C-means clustering. In this technique each data point belongs to a cluster to a degree specified by a membership grade. As in K-means clustering, Fuzzy C-means clustering relies on minimizing a cost function of dissimilarity measure.

The third technique is *Mountain* clustering, proposed by Yager and Filev [1]. This technique builds calculates a mountain function (density function) at every possible position in the data space, and chooses the position with the greatest density value as the center of the first cluster. It then destructs the effect of the first cluster mountain function and finds the second cluster center. This process is repeated until the desired number of clusters have been found.

The fourth technique is *Subtractive* clustering, proposed by Chiu [1]. This technique is similar to mountain clustering, except that instead of calculating the density function at every possible position in the data space, it uses the positions of the data points to calculate the density function, thus reducing the number of calculations significantly.

III. DATA CLUSTERING TECHNIQUES

In this section a detailed discussion of each technique is presented. Implementation and results are presented in the following sections.

A. K-means Clustering

The K-means clustering, or Hard C-means clustering, is an algorithm based on finding data clusters in a data set such that a cost function (or an objection function) of dissimilarity (or distance) measure is minimized [1]. In most cases this dissimilarity measure is chosen as the Euclidean distance.

A set of n vectors $\mathbf{x}_j$, $j=1,\ldots,n$, are to be partitioned into c groups G_i, $i=1,\ldots,c$. The cost function, based on the Euclidean distance between a vector $\mathbf{x}_k$ in group j and the corresponding cluster center $\mathbf{c}_i$, can be defined by:

$$J = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \left(\sum_{k,\ \mathbf{x}_k \in G_i} \left\| \mathbf{x}_k - \mathbf{c}_i \right\|^2 \right), \qquad (1)$$

where $J_i = \sum_{k,\ \mathbf{x}_k \in G_i} \left\| \mathbf{x}_k - \mathbf{c}_i \right\|^2$ is the cost function within group i.

The partitioned groups are defined by a $c \times n$ binary membership matrix $\mathbf{U}$, where the element u_{ij} is 1 if the jth data point $\mathbf{x}_j$ belongs to group i, and 0 otherwise. Once the cluster centers $\mathbf{c}_i$ are fixed, the minimizing u_{ij} for Equation (1) can be derived as follows:

$$u_{ij} = \begin{cases} 1 \text{ if } \left\| \mathbf{x}_j - \mathbf{c}_i \right\|^2 \le \left\| \mathbf{x}_j - \mathbf{c}_k \right\|^2, \text{ for each } k \ne i, \\ 0 \text{ otherwise.} \end{cases} \qquad (2)$$

Which means that $\mathbf{x}_j$ belongs to group i if $\mathbf{c}_i$ is the closest center among all centers.

On the other hand, if the membership matrix is fixed, i.e. if u_{ij} is fixed, then the optimal center $\mathbf{c}_i$ that minimize Equation (1) is the mean of all vectors in group i:

$$\mathbf{c}_i = \frac{1}{\left| G_i \right|} \sum_{k,\ \mathbf{x}_k \in G_i} \mathbf{x}_k , \qquad (3)$$

where $\left| G_i \right|$ is the size of G_i, or $\left| G_i \right| = \sum_{j=1}^{n} u_{ij}$.

The algorithm is presented with a data set $\mathbf{x}_i, i=1,\ldots,n$; it then determines the cluster centers $\mathbf{c}_i$ and the membership matrix $\mathbf{U}$ iteratively using the following steps:

Step 1: Initialize the cluster center $\mathbf{c}_i, i=1,\ldots,c$. This is typically done by randomly selecting c points from among all of the data points.

Step 2: Determine the membership matrix $\mathbf{U}$ by Equation (2).

Step 3: Compute the cost function according to Equation (1). Stop if either it is below a certain tolerance value or its improvement over previous iteration is below a certain threshold.

Step 4: Update the cluster centers according to Equation (3). Go to step 2.

The performance of the K-means algorithm depends on the initial positions of the cluster centers, thus it is advisable to run the algorithm several times, each with a different set of initial cluster centers. A discussion of the

implementation issues is presented later in this paper.

B. Fuzzy C-means Clustering

Fuzzy C-means clustering (FCM), relies on the basic idea of Hard C-means clustering (HCM), with the difference that in FCM each data point belongs to a cluster to a degree of membership grade, while in HCM every data point either belongs to a certain cluster or not. So FCM employs fuzzy partitioning such that a given data point can belong to several groups with the degree of belongingness specified by membership grades between 0 and 1. However, FCM still uses a cost function that is to be minimized while trying to partition the data set.

The membership matrix $\mathbf{U}$ is allowed to have elements with values between 0 and 1. However, the summation of degrees of belongingness of a data point to all clusters is always equal to unity:

$$\sum_{i=1}^{c} u_{ij} = 1, \quad \forall j = 1, \ldots, n. \tag{4}$$

The cost function for FCM is a generalization of Equation (1):

$$J(U, \mathbf{c}_1, \ldots, \mathbf{c}_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m d_{ij}^2, \tag{5}$$

where u_{ij} is between 0 and 1; $\mathbf{c}_i$ is the cluster center of fuzzy group i; $d_{ij} = \|\mathbf{c}_i - \mathbf{x}_j\|$ is the Euclidean distance between the ith cluster center and the jth data point; and $m \in [1, \infty)$ is a weighting exponent.

The necessary conditions for Equation (5) to reach its minimum are

$$\mathbf{c}_i = \frac{\sum_{j=1}^{n} u_{ij}^m \mathbf{x}_j}{\sum_{j=1}^{n} u_{ij}^m}, \tag{6}$$

and

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{kj}} \right)^{2/(m-1)}}. \tag{7}$$

The algorithm works iteratively through the preceding two conditions until the no more improvement is noticed. In a batch mode operation, FCM determines the cluster centers $\mathbf{c}_i$ and the membership matrix $\mathbf{U}$ using the following steps:

Step 1: Initialize the membership matrix $\mathbf{U}$ with random values between 0 and 1 such that the constraints in Equation (4) are satisfied.

Step 2: Calculate c fuzzy cluster centers $\mathbf{c}_i, i = 1, \ldots, c$, using Equation (6).

Step 3: Compute the cost function according to Equation (5). Stop if either it is below a certain tolerance value or its improvement over previous iteration is below a certain threshold.

Step 4: Compute a new $\mathbf{U}$ using Equation (7). Go to step 2.

As in K-means clustering, the performance of FCM depends on the initial membership matrix values; thereby it is advisable to run the algorithm for several times, each starting with different values of membership grades of data points.

C. Mountain Clustering

The mountain clustering approach is a simple way to find cluster centers based on a density measure called the *mountain function*. This method is a simple way to find approximate cluster centers, and can be used as a preprocessor for other sophisticated clustering methods.

The first step in mountain clustering involves forming a grid on the data space, where the intersections of the grid lines constitute the potential cluster centers, denoted as a set V.

The second step entails constructing a mountain function representing a data density measure. The height of the mountain function at a point $\mathbf{v} \in V$ is equal to

$$m(\mathbf{v}) = \sum_{i=1}^{N} \exp\left(-\frac{\|\mathbf{v} - \mathbf{x}_i\|^2}{2\sigma^2} \right), \tag{8}$$

where $\mathbf{x}_i$ is the ith data point and σ is an application specific constant. This equation states

that the *data density* measure at a point $\mathbf{v}$ is affected by all the points $\mathbf{x}_i$ in the data set, and this density measure is inversely proportional to the distance between the data points $\mathbf{x}_i$ and the point under consideration $\mathbf{v}$. The constant σ determines the height as well as the smoothness of the resultant mountain function.

The third step involves selecting the cluster centers by sequentially destructing the mountain function. The first cluster center $\mathbf{c}_1$ is determined by selecting the point with the greatest density measure. Obtaining the next cluster center requires eliminating the effect of the first cluster. This is done by revising the mountain function: a new mountain function is formed by subtracting a scaled Gaussian function centered at $\mathbf{c}_1$:

$$m_{\text{new}}(\mathbf{v}) = m(\mathbf{v}) - m(\mathbf{c}_1)\exp\left(-\frac{\left\|\mathbf{v}-\mathbf{c}_1\right\|^2}{2\beta^2}\right) \quad (9)$$

The subtracted amount eliminates the effect of the first cluster. Note that after subtraction, the new mountain function $m_{\text{new}}(\mathbf{v})$ reduces to zero at $\mathbf{v}=\mathbf{c}_1$.

After subtraction, the second cluster center is selected as the point having the greatest value for the new mountain function. This process continues until a sufficient number of cluster centers is attained.

D. Subtractive Clustering

The problem with the previous clustering method, mountain clustering, is that its computation grows exponentially with the dimension of the problem; that is because the mountain function has to be evaluated at each grid point. Subtractive clustering solves this problem by using data points as the candidates for cluster centers, instead of grid points as in mountain clustering. This means that the computation is now proportional to the problem size instead of the problem dimension. However, the actual cluster centers are not necessarily located at one of the data points, but in most cases it is a good approximation, especially with the reduced computation this approach introduces.

Since each data point is a candidate for cluster centers, a *density measure* at data point $\mathbf{x}_i$ is defined as

$$D_i = \sum_{j=1}^{n}\exp\left(-\frac{\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2}{(r_a/2)^2}\right), \quad (10)$$

where r_a is a positive constant representing a neighborhood radius. Hence, a data point will have a high density value if it has many neighboring data points.

The first cluster center $\mathbf{x}_{c_1}$ is chosen as the point having the largest density value D_{c_1}. Next, the density measure of each data point $\mathbf{x}_i$ is revised as follows:

$$D_i = D_i - D_{c_1}\exp\left(-\frac{\left\|\mathbf{x}_i-\mathbf{x}_{c_1}\right\|^2}{(r_b/2)^2}\right) \quad (11)$$

where r_b is a positive constant which defines a neighborhood that has measurable reductions in density measure. Therefore, the data points near the first cluster center $\mathbf{x}_{c_1}$ will have significantly reduced density measure.

After revising the density function, the next cluster center is selected as the point having the greatest density value. This process continues until a sufficient number of clusters is attainted.

IV. IMPLEMENTATION AND RESULTS

Having introduced the different clustering techniques and their basic mathematical foundations, we now turn to the discussion of these techniques on the basis of a practical study. This study involves the implementation of each of the four techniques introduced previously, and testing each one of them on a set of medical data related to heart disease diagnosis problem.

The medical data used consists of 13 input attributes related to clinical diagnosis of a heart disease, and one output attribute which indicates whether the patient is diagnosed with the heart disease or not. The whole data set consists of 300 cases. The data set is partitioned into two data sets: two-thirds of the data for training, and one-

Performance measure	Test Runs									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
No. of iterations	7	6	9	6	5	5	3	7	11	7
RMSE	0.469	0.469	0.447	0.469	0.632	0.692	0.692	0.447	0.447	0.469
Accuracy	78.0%	78.0%	80.0%	78.0%	60.0%	52.0%	52.0%	80.0%	80.0%	78.0%
Regression Line Slope	0.564	0.564	0.6	0.564	0.387	0.066	0.057	0.6	0.6	0.564

Table 1. K-means Clustering Performance Results

third for evaluation. The number of clusters into which the data set is to be partitioned is two clusters; i.e. patients diagnosed with the heart disease, and patients not diagnosed with the heart disease. Because of the high number of dimensions in the problem (13-dimensions), no visual representation of the clusters can be presented; only 2-D or 3-D clustering problems can be visually inspected. We will rely heavily on performance measures to evaluate the clustering techniques rather than on visual approaches.

As mentioned earlier, the similarity metric used to calculate the similarity between an input vector and a cluster center is the Euclidean distance. Since most similarity metrics are sensitive to the large ranges of elements in the input vectors, each of the input variables must be normalized to within the unit interval $[0,1]$; i.e. the data set has to be normalized to be within the unit hypercube.

Each clustering algorithm is presented with the training data set, and as a result two clusters are produced. The data in the evaluation set is then tested against the found clusters and an analysis of the results is conducted. The following sections present the results of each clustering technique, followed by a comparison of the four techniques.

MATLAB code for each of the four techniques can be found in the appendix.

A. K-means Clustering

As mentioned in the previous section, K-means clustering works on finding the cluster centers by trying to minimize a cost function J. It alternates between updating the membership matrix and updating the cluster centers using Equations (2) and (3), respectively, until no further improvement in the cost function is noticed. Since the algorithm initializes the cluster centers randomly, its performance is affected by those initial cluster centers. So several runs of the algorithm is advised to have better results.

Evaluating the algorithm is realized by testing the accuracy of the evaluation set. After the cluster centers are determined, the evaluation data vectors are assigned to their respective clusters according to the distance between each vector and each of the cluster centers. An error measure is then calculated; the root mean square error (RMSE) is used for this purpose. Also an accuracy measure is calculated as the percentage of correctly classified vectors. The algorithm was tested for 10 times to determine the best performance. Table 1 lists the results of those runs. Figure 1 shows a plot of the cost function over time for the best test case.

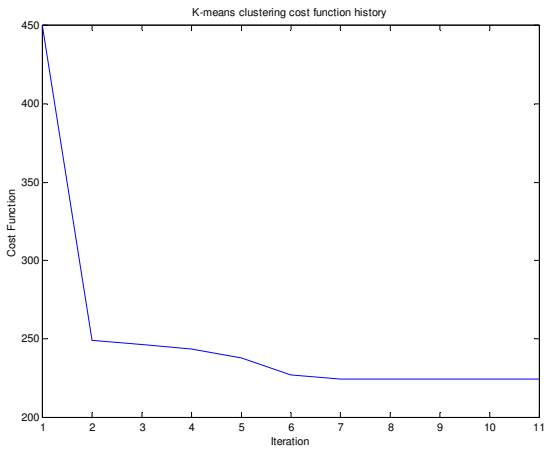

Figure 1. K-means clustering cost function plot

Performance measure	Weighting exponent *m*							
	1.1	**1.2**	**1.5**	**2**	**3**	**5**	**8**	**12**
No. of iterations	18	19	17	19	26	29	33	36
RMSE	0.469	0.469	0.479	0.469	0.458	0.479	0.479	0.479
Accuracy	78.0%	78.0%	77.0%	78.0%	79.0%	77.0%	77%	77%
Regression Line Slope	0.559	0.559	0.539	0.559	0.579	0.539	0.539	0.539

Table 2. Fuzzy C-means Clustering Performance Results

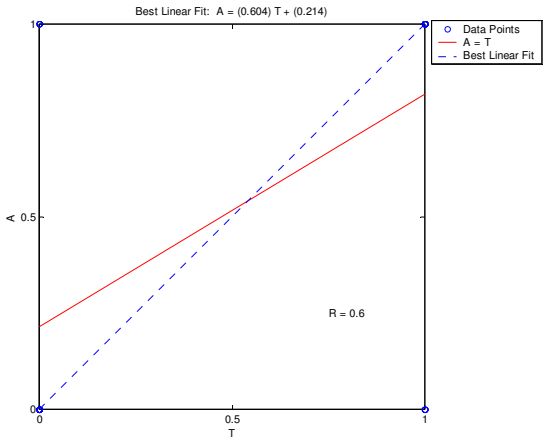

Figure 2. Regression Analysis of K-means Clustering

To further measure how accurately the identified clusters represent the actual classification of data, a regression analysis is performed of the resultant clustering against the original classification. Performance is considered better if the regression line slope is close to 1. Figure 2 shows the regression analysis of the best test case.

As seen from the results, the best case achieved 80% accuracy and an RMSE of 0.447. This relatively moderate performance is related to the high dimensionality of the problem; having too much dimensions tend to disrupt the coupling of data and introduces overlapping in some of these dimensions that reduces the accuracy of clustering. It is noticed also that the cost function converges rapidly to a minimum value as seen from the number of iterations in each test run. However, this has no effect on the accuracy measure.

B. Fuzzy C-means Clustering

FCM allows for data points to have different degrees of membership to each of the clusters; thus eliminating the effect of hard membership introduced by K-means clustering. This approach employs fuzzy measures as the basis for membership matrix calculation and for cluster centers identification.

As it is the case in K-means clustering, FCM starts by assigning random values to the membership matrix $\mathbf{U}$, thus several runs have to be conducted to have higher probability of getting good performance. However, the results showed no (or insignificant) variation in performance or accuracy when the algorithm was run for several times.

For testing the results, every vector in the evaluation data set is assigned to one of the clusters with a certain degree of belongingness (as done in the training set). However, because the output values we have are crisp values (either 1 or 0), the evaluation set degrees of membership are defuzzified to be tested against the actual outputs.

The same performance measures applied in K-means clustering will be used here; however only the effect of the weighting exponent m is analyzed, since the effect of random initial membership grades has insignificant effect on the final cluster centers. Table 2 lists the results of the tests with the effect of varying the weighting exponent m. It is noticed that very low or very high values for m reduces the accuracy; moreover high values tend to increase the time taken by the algorithm to find the clusters. A value of 2 seems adequate for this problem since

Performance measure	Test Runs									
	1	2	3	4	5	6	7	8	9	10
RMSE	0.566	0.469	0.566	0.49	0.548	0.566	0.566	0.529	0.693	0.469
Accuracy	68.0%	78.0%	68.0%	76.0%	70.0%	68.0%	68.0%	72.0%	52.0%	78.0%
Regression Line Slope	0.351	0.556	0.343	0.515	0.428	0.345	0.345	0.492	0.026	0.551

Table 3. Mountain Clustering Performance Results

it has good accuracy and requires less number of iterations. Figure 3 shows the accuracy and number of iterations against the weighting factor.

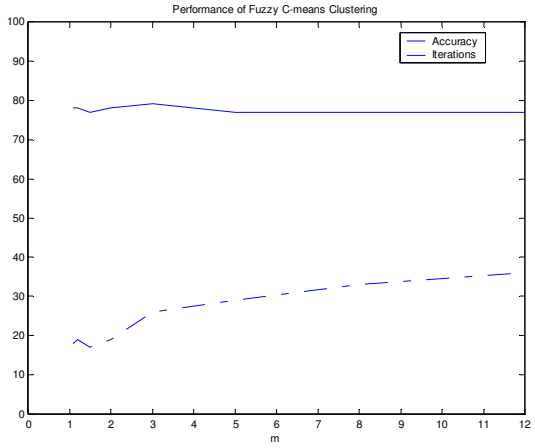

Figure 3. Fuzzy C-means Clustering Performance

In general, the FCM technique showed no improvement over the K-means clustering for this problem. Both showed close accuracy; moreover

FCM was found to be slower than K-means because of fuzzy calculations.

C. Mountain Clustering

Mountain clustering relies on dividing the data space into grid points and calculating a mountain function at every grid point. This mountain function is a representation of the density of data at this point.

The performance of mountain clustering is severely affected by the dimension of the problem; the computation needed rises exponentially with the dimension of input data because the mountain function has to be evaluated at each grid point in the data space. For a problem with c clusters, n dimensions, m data points, and a grid size of g per dimension, the required number of calculations is:

$$N = \underbrace{m \times g^n}_{\text{1st cluster}} + \underbrace{(c-1)g^n}_{\text{remainder clusters}} \quad (12)$$

So for the problem at hand, with input data of 13-dimensions, 200 training inputs, and a grid size of 10 per dimension, the required number of mountain function calculation is approximately 2.01×10^{15} calculations. In addition the value of the mountain function needs to be stored for every grid point for later calculations in finding subsequent clusters; which requires g^n storage locations, for our problem this would be 10^{13} storage locations. Obviously this seems impractical for a problem of this dimension.

In order to be able to test this algorithm, the dimension of the problem have to be reduced to a reasonable number; e.g. 4-dimensions. This is achieved by randomly selecting 4 variables from the input data out of the original 13 and performing the test on those variables. Several tests involving differently selected random variables are conducted in order to have a better understanding of the results. Table 3 lists the results of 10 test runs of randomly selected variables. The accuracy achieved ranged between 52% and 78% with an average of 70%, and average RMSE of 0.546. Those results are quite discouraging compared to the results achieved in K-means and FCM clustering. This is due to the fact that not all of the variables of the input data contribute to the clustering process; only 4 are chosen at random to make it possible to conduct the tests. However, with only 4 attributes chosen to do the tests, mountain clustering required far much more time than any other technique during the tests; this is because of the fact that the

Performance measure	Neighborhood radius r_a								
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
RMSE	0.67	0.648	0.648	0.5	0.5	0.5	0.5	0.5	0.648
Accuracy	55.0%	58.0%	58.0%	75.0%	75.0%	75.0%	75.0%	75.0%	58.0%
Regression Line Slope	0.0993	0.1922	0.1922	0.5074	0.5074	0.5074	0.5074	0.5074	0.1922

Table 4. Subtractive Clustering Performance Results

number of computation required is exponentially proportional to the number of dimensions in the

problem, as stated in Equation (12). So apparently mountain clustering is not suitable for problems of dimensions higher than two or three.

D. Subtractive Clustering

This method is similar to mountain clustering, with the difference that a density function is calculated only at every data point, instead of at every grid point. So the data points themselves are the candidates for cluster centers. This has the effect of reducing the number of computations significantly, making it linearly proportional to the number of input data instead of being exponentially proportional to its dimension. For a problem of c clusters and m data points, the required number of calculations is:

$$N = \underbrace{m^2}_{\text{1st cluster}} + \underbrace{(c-1)m}_{\text{remainder clusters}} \qquad (13)$$

As seen from the equation, the number of calculations does not depend on the dimension of the problem. For the problem at hand, the number of computations required is in the range of few ten thousands only.

Since the algorithm is fixed and does not rely on any randomness, the results are fixed. However, we can test the effect of the two variables r_a and r_b on the accuracy of the algorithm. Those variables represent a radius of neighborhood after which the effect (or contribution) of other data points to the density function is diminished. Usually the r_b variable is taken to be as $1.5r_a$. Table 4 shows the results of varying r_a. Figure 4 shows a plot of accuracy and RMSE against r_a.

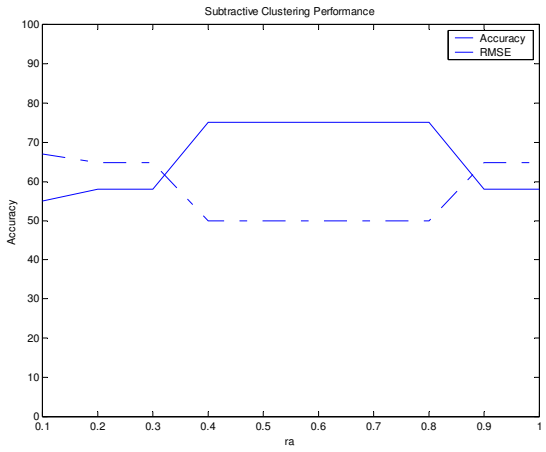

Figure 4. Subtractive Clustering Performance

It is clear from the results that choosing r_a very small or very large will result in a poor accuracy because if r_a is chosen very small the density function will not take into account the effect of neighboring data points; while if taken very large, the density function will be affected account all the data points in the data space. So a value between 0.4 and 0.7 should be adequate for the radius of neighborhood.

As seen from table 4, the maximum achieved accuracy was 75% with an RMSE of 0.5. Compared to K-means and FCM, this result is a little bit behind the accuracy achieved in those other techniques.

E. Results Summary and Comparison

According to the previous discussion of the implementation of the four data clustering techniques and their results, it is useful to

summarize the results and present some comparison of performances.
A summary of the best achieved results for each of the four techniques is presented in Table 5.

Algorithm	Comparison Aspect			
	RMSE	Accuracy	Regression Line Slope	Time (sec)
K-means	0.447	80.0%	0.6	0.9
FCM	0.469	78.0%	0.559	2.2
Mountain	0.469	78.0%	0.556	118.0
Subtractive	0.5	75.0%	0.5074	3.6

Table 5. Performance Results Comparison

From this comparison we can conclude some remarks:

- K-means clustering produces fairly higher accuracy and lower RMSE than the other techniques, and requires less computation time.
- Mountain clustering has a very poor performance regarding its requirement for huge number of computation and low accuracy. However, we have to notice that tests conducted on mountain clustering were done using part of the input variables in order to make it feasible to run the tests.
- Mountain clustering is suitable only for problems with two or three dimensions.
- FCM produces close results to K-means clustering, yet it requires more computation time than K-means because of the fuzzy measures calculations involved in the algorithm.
- In subtractive clustering, care has to be taken when choosing the value of the neighborhood radius r_a, since too small radii will result in neglecting the effect of neighboring data points, while large radii will result in a neighborhood of all the data points thus canceling the effect of the cluster.
- Since non of the algorithms achieved enough high accuracy rates, it is assumed that the problem data itself contains some overlapping in some of the dimensions; because of the high number of dimensions tend to disrupt the coupling of data and reduce the accuracy of clustering.

As stated earlier in this paper, clustering algorithms are usually used in conjunction with radial basis function networks and fuzzy models. The techniques described here can be used as preprocessors for RBF networks for determining the centers of the radial basis functions. In such cases, more accuracy can be gained by using gradient descent or other advanced derivative-based optimization schemes for further refinement. In fuzzy modeling, the clustering cluster centers produced by the clustering techniques can be modeled as if-then rules in ANFIS for example; where a training set (including inputs and outputs) to find cluster centers $(\mathbf{x}_i, y_i)$ via clustering first and then forming a zero-order Sugeno fuzzy modeling in which the ith rule is expressed as

If $\mathbf{X}$ is close to $\mathbf{x}_i$ then $\mathbf{Y}$ is close to y_i

Which means that the ith rule is based on the ith cluster center identified by the clustering method. Again, after the structure is determined, backpropagation-type gradient descent and other optimization schemes can be applied to proceed with parameter identification.

V. CONCLUSION

Four clustering techniques have been reviewed in this paper, namely: K-means clustering, Fuzzy C-means clustering, Mountain clustering, and Subtractive clustering. These approaches solve the problem of categorizing data by partitioning a data set into a number of clusters based on some similarity measure so that the similarity in each cluster is larger than among clusters. The four methods have been implemented and tested against a data set for medical diagnosis of heart disease. The comparative study done here is concerned with the accuracy of each algorithm, with care being taken toward the efficiency in calculation and other performance measures. The medical problem presented has a high number of dimensions, which might involve some complicated relationships between the variables in the input data. It was obvious that mountain

clustering is not one of the good techniques for problems with this high number of dimensions due to its exponential proportionality to the dimension of the problem. K-means clustering seemed to over perform the other techniques for this type of problem. However in other problems where the number of clusters is not known, K-means and FCM cannot be used to solve this type of problem, leaving the choice only to mountain or subtractive clustering. Subtractive clustering seems to be a better alternative to mountain clustering since it is based on the same idea, and uses the data points as cluster centers candidates instead of grid points; however, mountain clustering can lead to better results if the grid granularity is small enough to capture the potential cluster centers, but with the side effect of increasing computation needed for the larger number of grid points.

Finally, the clustering techniques discussed here do not have to be used as stand-alone approaches; they can be used in conjunction with other neural or fuzzy systems for further refinement of the overall system performance.

VI. REFERENCES

[1] Jang, J.-S. R., Sun, C.-T., Mizutani, E., "Neuro-Fuzzy and Soft Computing – A Computational Approach to Learning and Machine Intelligence," *Prentice Hall.*

[2] Azuaje, F., Dubitzky, W., Black, N., Adamson, K., "Discovering Relevance Knowledge in Data: A Growing Cell Structures Approach," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 30, No. 3, June 2000 (pp. 448)*

[3] Lin, C., Lee, C., "Neural Fuzzy Systems," *Prentice Hall, NJ, 1996.*

[4] Tsoukalas, L., Uhrig, R., "Fuzzy and Neural Approaches in Engineering," *John Wiley & Sons, Inc., NY, 1997.*

[5] Nauck, D., Kruse, R., Klawonn, F., "Foundations of Neuro-Fuzzy Systems," *John Wiley & Sons Ltd., NY, 1997.*

[6] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," Applied Statistics, 28:100--108, 1979.

[7] The MathWorks, Inc., "Fuzzy Logic Toolbox – For Use With MATLAB," *The MathWorks, Inc., 1999.*

Appendix

K-means Clustering (MATLAB script)

```
% K-means clustering

% ------------------- CLUSTERING PHASE -------------------
% Load the Training Set
TrSet = load('TrainingSet.txt');
[m,n] = size(TrSet);       % (m samples) x (n dimensions)
for i = 1:m                % the output (last column) values (0,1,2,3) are mapped to (0,1)
   if TrSet(i,end)>=1
      TrSet(i,end)=1;
   end
end

% find the range of each attribute (for normalization later)
for i = 1:n
   range(1,i) = min(TrSet(:,i));
   range(2,i) = max(TrSet(:,i));
end

x = Normalize(TrSet, range);      % normalize the data set to a hypercube
x(:,end) = [];                        % get rid of the output column
[m,n] = size(x);

nc = 2; % number of clusters = 2

% Initialize cluster centers to random points
c = zeros(nc,n);
for i = 1:nc
   rnd = int16(rand*m + 1);   % select a random vector from the input set
   c(i,:) = x(rnd,:);         % assign this vector value to cluster (i)
end

% Clustering Loop

delta = 1e-5;
n = 1000;
iter = 1;
while (iter < n)

% Determine the membership matrix U
% u(i,j) = 1 if euc_dist(x(j),c(i)) <= euc_dist(x(j),c(k)) for each k ~= i
% u(i,j) = 0 otherwise

for i = 1:nc
   for j = 1:m
      d = euc_dist(x(j,:),c(i,:));
      u(i,j) = 1;
      for k = 1:nc
         if k~=i
            if euc_dist(x(j,:),c(k,:)) < d
               u(i,j) = 0;
            end
         end
      end
   end
end

% Compute the cost function J

J(iter) = 0;
for i = 1:nc
   JJ(i) = 0;
   for k = 1:m
      if u(i,k)==1
         JJ(i) = JJ(i) + euc_dist(x(k,:),c(i,:));
      end
   end
   J(iter) = J(iter) + JJ(i);
end
```

```
% Stop if either J is below a certain tolerance value,
% or its improvement over previous iteration is below a certain threshold
str = sprintf('iteration: %.0d, J=%d', iter, J(iter));
disp(str);
if (iter~=1) & (abs(J(iter-1) - J(iter)) < delta)
   break;
end

% Update the cluster centers
% c(i) = mean of all vectors belonging to cluster (i)

for i = 1:nc
   sum_x = 0;
   G(i) = sum(u(i,:));
   for k = 1:m
      if u(i,k)==1
         sum_x = sum_x + x(k,:);
      end
   end
   c(i,:) = sum_x ./ G(i);
end

iter = iter + 1;
end % while
disp('Clustering Done.');

% ----------------- TESTING PHASE -------------------------

% Load the evaluation data set
EvalSet = load('EvaluationSet.txt');
[m,n] = size(EvalSet);
for i = 1:m
   if EvalSet(i,end)>=1
      EvalSet(i,end)=1;
   end
end

x = Normalize(EvalSet, range);
x(:,end) = [];
[m,n] = size(x);

% Assign evaluation vectors to their respective clusters according
% to their distance from the cluster centers

for i = 1:nc
   for j = 1:m
      d = euc_dist(x(j,:),c(i,:));
      evu(i,j) = 1;
      for k = 1:nc
         if k~=i
            if euc_dist(x(j,:),c(k,:)) < d
               evu(i,j) = 0;
            end
         end
      end
   end
end

% Analyze results
ev = EvalSet(:,end)';
rmse(1) = norm(evu(1,:)-ev)/sqrt(length(evu(1,:)));
rmse(2) = norm(evu(2,:)-ev)/sqrt(length(evu(2,:)));

subplot(2,1,1);
if rmse(1) < rmse(2)
   r = 1;
else
   r = 2;
end

str = sprintf('Testing Set RMSE: %f', rmse(r));
disp(str);
ctr = 0;
for i = 1:m
   if evu(r,i)==ev(i)
```

```
      ctr = ctr + 1;
   end
end
str = sprintf('Testing Set accuracy: %.2f%%', ctr*100/m);
disp(str);
[m,b,r] = postreg(evu(r,:),ev);       % Regression Analysis
disp(sprintf('r = %.3f', r));
```

Fuzzy C-means Clustering (MATLAB script)

```
% Fuzzy C-means clustering

% ------------------- CLUSTERING PHASE -------------------
% Load the Training Set
TrSet = load('TrainingSet.txt');
[m,n] = size(TrSet);       % (m samples) x (n dimensions)
for i = 1:m                % the output (last column) values (0,1,2,3) are mapped to (0,1)
   if TrSet(i,end)>=1
      TrSet(i,end)=1;
   end
end

% find the range of each attribute (for normalization later)
for i = 1:n
   range(1,i) = min(TrSet(:,i));
   range(2,i) = max(TrSet(:,i));
end

x = Normalize(TrSet, range);      % normalize the data set to a hypercube
x(:,end) = [];                        % get rid of the output column
[m,n] = size(x);

nc = 2; % number of clusters = 2

% Initialize the membership matrix with random values between 0 and 1
% such that the summation of membership degrees for each vector equals unity
u = zeros(nc,m);
for i = 1:m
   u(1,i) = rand;
   u(2,i) = 1 - u(1,i);
end

% Clustering Loop

m_exp = 12;
prevJ = 0;
J = 0;
delta = 1e-5;
n = 1000;
iter = 1;
while (iter < n)

% Calculate the fuzzy cluster centers

for i = 1:nc
   sum_ux = 0;
   sum_u = 0;
   for j = 1:m
      sum_ux = sum_ux + (u(i,j)^m_exp)*x(j,:);
      sum_u = sum_u + (u(i,j)^m_exp);
   end
   c(i,:) = sum_ux ./ sum_u;
end

% Compute the cost function J

J(iter) = 0;
for i = 1:nc
   JJ(i) = 0;
   for j = 1:m
      JJ(i) = JJ(i) + (u(i,j)^m_exp)*euc_dist(x(j,:),c(i,:));
```

```
   end
   J(iter) = J(iter) + JJ(i);
end

% Stop if either J is below a certain tolerance value,
% or its improvement over previous iteration is below a certain threshold
str = sprintf('iteration: %.0d, J=%d', iter, J);
disp(str);
if (iter~=1) & (abs(J(iter-1) - J(iter)) < delta)
   break;
end

% Update the membership matrix U

for i = 1:nc
   for j = 1:m
      sum_d = 0;
      for k = 1:nc
         sum_d = sum_d + (euc_dist(c(i,:),x(j,:))/euc_dist(c(k,:),x(j,:)))^(2/(m_exp-1));
      end
      u(i,j) = 1/sum_d;
   end
end

iter = iter + 1;
end % while
disp('Clustering Done.');

% ----------------- TESTING PHASE --------------------------

% Load the evaluation data set
EvalSet = load('EvaluationSet.txt');
[m,n] = size(EvalSet);
for i = 1:m
   if EvalSet(i,end)>=1
      EvalSet(i,end)=1;
   end
end

x = Normalize(EvalSet, range);
x(:,end) = [];
[m,n] = size(x);

% Assign evaluation vectors to their respective clusters according
% to their distance from the cluster centers

for i = 1:nc
   for j = 1:m
      sum_d = 0;
      for k = 1:nc
         sum_d = sum_d + (euc_dist(c(i,:),x(j,:))/euc_dist(c(k,:),x(j,:)))^(2/(m_exp-1));
      end
      evu(i,j) = 1/sum_d;
   end
end

% defuzzify the membership matrix
for j = 1:m
   if evu(1,j) >= evu(2,j)
      evu(1,j) = 1; evu(2,j) = 0;
   else
      evu(1,j) = 0; evu(2,j) = 1;
   end
end

% Analyze results
ev = EvalSet(:,end)';
rmse(1) = norm(evu(1,:)-ev)/sqrt(length(evu(1,:)));
rmse(2) = norm(evu(2,:)-ev)/sqrt(length(evu(2,:)));

subplot(2,1,1);
if rmse(1) < rmse(2)
   r = 1;
else
   r = 2;
end
```

```
str = sprintf('Testing Set RMSE: %f', rmse(r));
disp(str);
ctr = 0;
for i = 1:m
   if evu(r,i)==ev(i)
      ctr = ctr + 1;
   end
end
str = sprintf('Testing Set accuracy: %.2f%%', ctr*100/m);
disp(str);
[m,b,r] = postreg(evu(r,:),ev);       % Regression Analysis
disp(sprintf('r = %.3f', r));
```

Mountain Clustering (MATLAB script)

```
% Mountain Clustering

%---------------------------------------------------------------------
% Setup the training data
%---------------------------------------------------------------------

% Load the Training Set
TrSet = load('TrainingSet.txt');
[m,n] = size(TrSet);       % (m samples) x (n dimensions)
for i = 1:m                % the output (last column) values (0,1,2,3) are mapped to (0,1)
   if TrSet(i,end)>=1
      TrSet(i,end)=1;
   end
end

% find the range of each attribute (for normalization later)
for i = 1:n
   range(1,i) = min(TrSet(:,i));
   range(2,i) = max(TrSet(:,i));
end

x = Normalize(TrSet, range);      % normalize the data set to a hypercube
x(:,end) = [];                        % get rid of the output column
[m,n] = size(x);

% Due to memory and speed limitations, the number of attributes
% will be set to a maximum of 4 attributes. Extra attributes will
% be dropped at random.
n_dropped = 0;
if n>4
    for i = 1:(n-4)
    attr = ceil(rand*(n-i+1));
      x(:,attr) = [];
      dropped(i) = attr;  % save dropped attributes positions
      n_dropped = n_dropped+1;
   end
end
[m,n] = size(x);

%---------------------------------------------------------------------
% First: setup a grid matrix of n-dimensions (V)
% (n = the dimension of input data vectors)
% The gridding granularity is 'gr' = # of grid points per dimension
%---------------------------------------------------------------------

gr = 10;
% setup the dimension vector [d1 d2 d3 .... dn]
v_dim = gr * ones([1 n]);
% setup the mountain matrix
M = zeros(v_dim);
sigma = 0.1;

%---------------------------------------------------------------------
% Second: calculate the mountain function at every grid point
%---------------------------------------------------------------------
```

```
% setup some aiding variables
cur = ones([1 n]);
for i = 1:n
   for j = 1:i
      cur(i) = cur(i)*v_dim(j);
   end
end
max_m = 0; % greatest density value
max_v = 0; % cluster center position

disp('Finding Cluster 1...');
% loop over each grid point
for i = 1:cur(1,end)

   % calculate the vector indexes
   idx = i;
   for j = n:-1:2
      dim(j) = ceil(idx/cur(j-1));
      idx = idx - cur(j-1)*(dim(j)-1);
   end
   dim(1) = idx;
   % dim is holding the current point index vector
   % but needs to be normalized to the range [0,1]
   v = dim./gr;

   % calculate the mountain function for the current point
   M(i) = mnt(v,x,sigma);
   if M(i) > max_m
      max_m = M(i);
      max_v = v;
      max_i = i;
   end

   % report progress
   if mod(i,5000)==0
      str = sprintf('vector %.0d/%.0d; M(v)=%.2f', i, cur(1,end), M(i));
      disp(str);
   end

end

%----------------------------------------------------------------------
% Third: select the first cluster center by choosing the point
%        with the greatest density value
%----------------------------------------------------------------------

c(1,:) = max_v;
c1 = max_i;
str = sprintf('Cluster 1:');
disp(str);
str = sprintf('%4.1f', c(1,:));
disp(str);
str = sprintf('M=%.3f', max_m);
disp(str);

%----------------------------------------------------------------------
% CLUSTER 2
%----------------------------------------------------------------------

Mnew = zeros(v_dim);
max_m = 0;
max_v = 0;
beta = 0.1;

disp('Finding Cluster 2...');
for i = 1:cur(1,end)

   % calculate the vector indexes
   idx = i;
   for j = n:-1:2
      dim(j) = ceil(idx/cur(j-1));
      idx = idx - cur(j-1)*(dim(j)-1);
   end
   dim(1) = idx;
   % dim is holding the current point index vector
```

```
   % but needs to be normalized to the range [0,1]
   v = dim./gr;

   % calculate the REVISED mountain function for the current point
   Mnew(i) = M(i) - M(c1)*exp((-euc_dist(v,c(1,:))./(2*beta^2));

   if Mnew(i) > max_m
      max_m = Mnew(i);
      max_v = v;
      max_i = i;
   end

   % report progress
   if mod(i,5000)==0
      str = sprintf('vector %.0d/%.0d; Mnew(v)=%.2f', i, cur(1,end), Mnew(i));
      disp(str);
   end

end

c(2,:) = max_v;
str = sprintf('Cluster 2:');
disp(str);
str = sprintf('%4.1f', c(2,:));
disp(str);
str = sprintf('M=%.3f', max_m);
disp(str);

%----------------------------------------------------------------------
% Evaluation
%----------------------------------------------------------------------

% Load the evaluation data set
EvalSet = load('EvaluationSet.txt');
[m,n] = size(EvalSet);
for i = 1:m
   if EvalSet(i,end)>=1
      EvalSet(i,end)=1;
   end
end

x = Normalize(EvalSet, range);
x(:,end) = [];
[m,n] = size(x);

% drop the attributes corresponding to the ones dropped in the training set
for i = 1:n_dropped
   x(:,dropped(i)) = [];
end
[m,n] = size(x);

% Assign every test vector to its nearest cluster
for i = 1:2
   for j = 1:m
      d = euc_dist(x(j,:),c(i,:));
      evu(i,j) = 1;
      for k = 1:2
         if k~=i
            if euc_dist(x(j,:),c(k,:)) < d
               evu(i,j) = 0;
            end
         end
      end
   end
end

% Analyze results
ev = EvalSet(:,end)';
rmse(1) = norm(evu(1,:)-ev)/sqrt(length(evu(1,:)));
rmse(2) = norm(evu(2,:)-ev)/sqrt(length(evu(2,:)));

if rmse(1) < rmse(2)
   r = 1;
else
   r = 2;
end
```

```
str = sprintf('Testing Set RMSE: %f', rmse(r));
disp(str);
ctr = 0;
for i = 1:m
   if evu(r,i)==ev(i)
      ctr = ctr + 1;
   end
end
str = sprintf('Testing Set accuracy: %.2f%%', ctr*100/m);
disp(str);
[m,b,r] = postreg(evu(r,:),ev);
disp(sprintf('r = %.3f', r));
```

Subtractive Clustering (MATLAB script)

```
% Subtractive Clustering

%----------------------------------------------------------------------
% Setup the training data
%----------------------------------------------------------------------

% Load the Training Set
TrSet = load('TrainingSet.txt');
[m,n] = size(TrSet);       % (m samples) x (n dimensions)
for i = 1:m                % the output (last column) values (0,1,2,3) are mapped to (0,1)
   if TrSet(i,end)>=1
      TrSet(i,end)=1;
   end
end

% find the range of each attribute (for normalization later)
for i = 1:n
   range(1,i) = min(TrSet(:,i));
   range(2,i) = max(TrSet(:,i));
end

x = Normalize(TrSet, range);      % normalize the data set to a hypercube
x(:,end) = [];                        % get rid of the output column
[m,n] = size(x);

%----------------------------------------------------------------------
% First: Initialize the density matrix and some variables
%----------------------------------------------------------------------

D = zeros([m 1]);
ra = 1.0;

%----------------------------------------------------------------------
% Second: calculate the density function at every data point
%----------------------------------------------------------------------

% setup some aiding variables
max_d = 0; % greatest density value
max_x = 0; % cluster center position

disp('Finding Cluster 1...');
% loop over each data point
for i = 1:m

   % calculate the density function for the current point
   D(i) = density(x(i,:),x,ra);
   if D(i) > max_d
      max_d = D(i);
      max_x = x(i,:);
      max_i = i;
   end

   % report progress
   if mod(i,50)==0
      str = sprintf('vector %.0d/%.0d; D(v)=%.2f', i, m, D(i));
```

```
      % disp(str);
   end

end

%-----------------------------------------------------------------
% Third: select the first cluster center by choosing the point
%        with the greatest density value
%-----------------------------------------------------------------

c(1,:) = max_x;
c1 = max_i;
str = sprintf('Cluster 1:');
disp(str);
str = sprintf('%4.1f', c(1,:));
disp(str);
str = sprintf('D=%.3f', max_d);
disp(str);

%-----------------------------------------------------------------
% CLUSTER 2
%-----------------------------------------------------------------

Dnew = zeros([m 1]);
max_d = 0;
max_x = 0;
rb = 1.5*ra;

disp('Finding Cluster 2...');
for i = 1:m

   % calculate the REVISED density function for the current point
   Dnew(i) = D(i) - D(c1)*exp((-euc_dist(x(i,:),c(1,:)))./((rb/2)^2));

   if Dnew(i) > max_d
      max_d = Dnew(i);
      max_x = x(i,:);
      max_i = i;
   end

   % report progress
   if mod(i,50)==0
      str = sprintf('vector %.0d/%.0d; Dnew(v)=%.2f', i, m, Dnew(i));
      disp(str);
   end

end

c(2,:) = max_x;
str = sprintf('Cluster 2:');
disp(str);
str = sprintf('%4.1f', c(2,:));
disp(str);
str = sprintf('D=%.3f', max_d);
disp(str);

%-----------------------------------------------------------------
% Evaluation
%-----------------------------------------------------------------

% Load the evaluation data set
EvalSet = load('EvaluationSet.txt');
[m,n] = size(EvalSet);
for i = 1:m
   if EvalSet(i,end)>=1
      EvalSet(i,end)=1;
   end
end

x = Normalize(EvalSet, range);
x(:,end) = [];
[m,n] = size(x);

% Assign every test vector to its nearest cluster
for i = 1:2
   for j = 1:m
```

```
      dd = euc_dist(x(j,:),c(i,:));
      evu(i,j) = 1;
      for k = 1:2
         if k~=i
            if euc_dist(x(j,:),c(k,:)) < dd
               evu(i,j) = 0;
            end
         end
      end
   end
end

% Analyze results
ev = EvalSet(:,end)';
rmse(1) = norm(evu(1,:)-ev)/sqrt(length(evu(1,:)));
rmse(2) = norm(evu(2,:)-ev)/sqrt(length(evu(2,:)));

if rmse(1) < rmse(2)
   r = 1;
else
   r = 2;
end

str = sprintf('Testing Set RMSE: %f', rmse(r));
disp(str);
ctr = 0;
for i = 1:m
   if evu(r,i)==ev(i)
      ctr = ctr + 1;
   end
end
str = sprintf('Testing Set accuracy: %.2f%%', ctr*100/m);
disp(str);
[m,b,r] = postreg(evu(r,:),ev);
disp(sprintf('r = %.3f', r));
```