A Comparative Study of Clustering Algorithms

Satchidanandan Dehuri, Chinmay Mohapatra, Ashish Ghosh and Rajib Mall
Department of Information and Communication Technology, Fakir Mohan University, Vyasa Vihar, Balasore, India
School of Mathematics, Statistics and Computer Science, Utkal University, Vani Vihar, India
Machine Intelligence Unit, Indian Statistical Institute, 203, B.T. Road, Kolkata-3700108, India
Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur-721302, India

Abstract: Data clustering is an unsupervised task that can generate different shapes of clusters for a particular type of dataset. Hence choosing an algorithm for a particular type of dataset is a difficult problem. This study presents the choice of an appropriate clustering algorithm by a comparative study of three representative techniques like K-means, Kohonen's Self Organizing Map (SOM) and Density Based Spatial Clustering of Applications with Noise (DBSCAN) based on the extensive simulation studies. Comparison is performed on the basis of cluster quality index 'B', percentage of samples correctly classified and CPU time. The experimental results show that if the clusters are of arbitrary shape, a density based clustering algorithm like DBSCAN is preferable, where as if the clusters are of hyper spherical or convex shape and well-seperated then the SOM or K-means is preferable.

Key words: Data clustering, DBSCAN, SOM, K-means

INTRODUCTION

Recently there has been an enormous growth in the amount of commercial and scientific data, such as protein sequence, retail transaction and web logs (Fayyad et al., 1996). The questions arise, why so much of data? People store data because they think some valuable assets are implicitly coded within it. But raw data is rarely of direct benefit. Discovering groups of similar data items in the data set, known as cluster analysis, is an interesting and challenging issue. There are many different data clustering algorithms emerged from several disciplines such as statistics, pattern recognition and machine learning ( Jain et al., 1999). They give different types of clusters for a particular type of dataset. Some of them require some additional prior knowledge about the nature of the data. Each clustering algorithm works well for certain types of data and with certain applications. The choice of an appropriate clustering algorithm can be made from a comparative study of the clustering algorithms.

Furthermore, the comparison is necessary as it is widely used in many areas like information retrieval and text mining (Cutting et al., 1992; Steinbach et al., 2000; Dhillon et al., 2001), spatial database applications, e.g., Geographical Information System (GIS) or astronomical data (Xu et al., 1998; Sandar et al., 1998; Ester et al., 2000), sequence and heterogeneous data analysis (Cadez et al., 2001), web applications (Heer et al., 2001; Foss et al., 2001), DNA analysis in computational biology (Ben-Dor et al., 1999) and many others.

The clustering problem is the problem of dividing a given set \( \{x_1, ..., x_n\} \) of \( N \) data points into several non-overlapping homogenous groups. Each such group or cluster should contain similar data items and data items from different groups should not be similar. An \( k \) clustering of a data set \( X \) is the partition of \( X \) into \( k \) clusters, \( C_1, ..., C_k \) such that the following three conditions are met:

i) \( C_i \neq \emptyset, i = 1, ..., k \)
ii) \( \bigcup_{i=1}^{k} C_i \) (except outliers)
iii) \( C_i \cap C_j = \emptyset, i \neq j, i, j = 1, ..., k \) (sometimes violated)

The various data clustering algorithms can be classified into partitional, density based and clustering using artificial neural networks. In this work, the three representative algorithms like K-means (Hartigan et al., 1979; Hartigan, 1975), DBSCAN (Ester et al., 1996) and Kohonen Self-organizing Map.
(SOM) (Kohonen, 1990) corresponds to partitional, density based and artificial neural networks are taken for comparative study.

The comparison is performed on the basis of cluster quality index '\( \beta \)', percentage of samples correctly clustered and CPU time. Mathematically, it is defined as

\[
\beta = \frac{\sum_{i=1}^{k} \sum_{j=1}^{n_i} (x_{ij} - \bar{x})^T (x_{ij} - \bar{x})}{\sum_{i=1}^{k} \sum_{j=1}^{n_i} (x_{ij} - \bar{x}_i)^T (x_{ij} - \bar{x}_i)}
\]

where, \( n_i \) is the number of points in the \( i \)th (\( i = 1, \ldots, k \)) cluster, \( x_i \) is the feature vector of the \( j \)th pattern (\( j = 1, \ldots, n_i \)) in cluster \( i \), \( \bar{x}_i \) is the mean of \( n_i \) patterns of the \( i \)th cluster, \( n \) is the total number of patterns and \( \bar{x} \) is the mean value of the entire set of patterns. '\( \beta \)' is the ratio of the total variation and within-cluster variation. For a given data and number of clusters \( k \), the higher the homogeneity within the clustered regions, the higher would be the \( \beta \) value.

**CLUSTERING ALGORITHMS**

This technique produces clusters by optimising a criterion function defined either locally or globally. The user should have prior knowledge of the number of clusters. The algorithm runs multiple number of times with different starting points and the best configuration obtained from all of the runs is used as the output clustering. Let us discuss each of the clustering algorithms.

**K-means:** The k-means algorithm (Hartigan *et al.*, 1979; Hartigan, 1975) is the most popular clustering tool used in scientific and industrial applications. The name comes from representing each of \( k \) clusters \( c_i \) by the mean (or weighted average) \( \bar{x}_i \) of its points, the so-called centroid. While this obviously does not work well with categorical attributes, it has the good geometric and statistical sense for numerical attributes. The sum of discrepancies between a point and its centroid expressed through appropriate distance measure is used as the criterion function. For example, the \( L_2 \)-norm based objective function, the sum of the squares of errors between the points and the corresponding centroids, is equal to the total intra-cluster variance

\[
E(c_i) = \sum_{i=1}^{n_i} ||x_i - \mu_i||^2,
\]

where \( k \) is the number of points in cluster \( j \).

The sum of the squares of errors can be rationalized as a negative of log-likelihood for normally distributed mixture model and is widely used in statistics (SSE). Therefore k-means algorithm can be derived from general probabilistic framework. The inter-cluster variance is measured as

\[
E(c_i) = \sum_{j=1}^{k} \sum_{x_j \in c_i} ||x_j - \mu_j||^2
\]

where, \( k \) is the number of clusters. Two version of k-means iterative optimisation are known. The first version is known as Forgy's algorithm (Forgy, 1965) and consists of the following steps:

1. Choose the number of clusters, \( k \).
2. Set initial centres of clusters, \( c_1, c_2, \ldots, c_k \) to the arbitrarily selected \( k \) vectors from the dataset.
3. Classify each vector \( x_i = [x_{i1}, x_{i2}, \ldots, x_{id}] \) (\( d \) is the dimension of the input vectors) into the closest centre \( c_i \) by Euclidean distance measure:

\[
||x_i - c_i|| = \min ||x_i - c_j||
\]

4. Recompute the estimates for the cluster centres \( c_i \). Let \( c_i = [c_{i1}, c_{i2}, \ldots, c_{id}] \), \( c_{in} \) is computed by

\[
c_{in} = \frac{\sum_{x_n \in cluster c_i} x_{in}}{n_i}
\]

where \( n_i \) is the number of vectors in the \( i \)th cluster.

5. If none of the cluster centres \( c_i, i = 1, 2, \ldots, k \) changes in step 4, stop; otherwise go to step 3.

The second version (classic in iterative optimisation) of k-means iterative optimisation reassigns points based on more detailed analysis of effects on the objective function caused by moving a point from its current cluster to a potentially new one. If a move has a positive effect, the point is relocated and the two centroids are recomputed. It is not clear that this version is computationally feasible, because the outlined analysis requires an inner loop over all member points of involved clusters affected by centroids shifts. However, in \( L_2 \) case it is known (Duda *et al.*, 1973) that all computations can be algebraically reduced to simply computing a single distance! Therefore, in this case both versions have the same computational complexity.

The wide popularity of k-means algorithm is well deserved. It is simple, straightforward and is based on the firm foundation of analysis of variances. The K-means algorithm also suffers from all the usual suspects: i) the result strongly depends on the initial guess of centroids.
Kohonen's Self-Organizing Map (SOM): Kohonen self-organizing map (Kohonen, 1990) is an unsupervised, competitive learning, clustering network, in which only one neuron (or only one neuron in a group) is “on” at a time. Self-organizing maps (SOMs) can be used as a data visualization tool as they can reduce the dimensions of data through the use of self-organizing neural networks while preserving their topological structure. The problem that data visualization attempts to solve is that humans simply cannot visualize high dimensional data as is so techniques are created to help us understand this high dimensional data. The way SOMs go about reducing dimensions is by producing a map of usually one or two dimensions, which plot the similarities of the data by grouping similar data items together. The plot is drawn by giving each output neuron a gray scale colour that is proportionate to its weight vectors average distance from it immediate neighbours weight vectors. Thus SOMs accomplish two things, they reduce dimensions and display similarities.

Basic architecture: Figure 1 shows the architecture of Kohonen SOM. It contains a single layer of neurons in addition to an input layer of branching nodes. There are ‘k’ neurons in the neural layer and each has a parametric weight vector \( w_k \) of dimension ‘n’, which is same as the dimension of the input feature vectors \( \vec{x} = (x_1, x_2, ..., x_n) \). The weight vectors \( \vec{w}_1, \vec{w}_2, ..., \vec{w}_k \) are randomly initialized in the feature space at the beginning. One exemplar input vector \( \vec{x} \) is selected from the sample and put into the network and squared distances between \( \vec{x} \) and each \( \vec{w}_i \) are computed by Euclidean distance. The minimum distance is then determined to obtain the neuron that is the winner over the other neurons. In this network the weight vectors are multiplied by three different strategies.

- In the first strategy called winner-take-all strategy, the winning neuron updates its parametric weight vector. All other neurons keep their old values.
- In second strategy, update positively (reinforce, or reward) all neurons that are close to the winning neuron and to update negatively all of those neurons that are farther away from the winner.
- In the third strategy, when a vector \( \vec{x} \) is presented to a Kohonen network, the dot product \( y_k = \vec{x} \cdot \vec{w}_k \) is computed as output from each ‘k’ neurons.

Neighbourhoods of a unit designated by \( \# \) of radii \( R = 2, 1 \) and 0 in a one-dimensional topology with 5 cluster units is shown in Fig. 2. A neighborhood for a 2D planar rectangular grid topology is shown in Fig. 3.

Algorithm SOM

1. Fix the number of output neurons \( k \), each of which represents a cluster.
2. Initialise the weight vector for each output neuron with i) random values, or ii) the first \( k \) input vectors.
3. Set topological neighbourhood parameters and learning rate parameter \( \alpha \).
4. While stopping condition is false
5. For each input \( x \)
6. Update weight vector \( w_k \) of the closest output neuron and its neighbors as follows:
   \[ w_i(\text{new}) = w_i(\text{old}) + \alpha [x - w_i(\text{old})], \ i = 1, ..., n \]
7. End for
8. Reduce learning rate
9. Reduce radius of neighborhood at specified intervals.
10. End while

Fig. 1: Kohonen self-organizing map

<table>
<thead>
<tr>
<th>R = 2</th>
<th>R = 1</th>
<th>#R = 0</th>
<th>R = 1</th>
<th>R = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R = 2</td>
<td>R = 2</td>
<td>R = 2</td>
<td>R = 2</td>
<td>R = 2</td>
</tr>
<tr>
<td>R = 2</td>
<td>R = 1</td>
<td>R = 1</td>
<td>R = 1</td>
<td>R = 2</td>
</tr>
<tr>
<td>R = 2</td>
<td>R = 1</td>
<td>#R = 0</td>
<td>R = 1</td>
<td>R = 2</td>
</tr>
<tr>
<td>R = 2</td>
<td>R = 1</td>
<td>R = 1</td>
<td>R = 1</td>
<td>R = 2</td>
</tr>
<tr>
<td>R = 2</td>
<td>R = 2</td>
<td>R = 2</td>
<td>R = 2</td>
<td>R = 2</td>
</tr>
</tbody>
</table>

Fig. 2: Linear topology

Fig. 3: 2-D rectangular grid topology
The learning process has two separate but related phases, that is, the ordering phase and the convergence phase. During the ordering phase, the learning rate should be set close to unity and then gradually decreased, but not allowed to go below a certain threshold value (usually 0.1). It is during this part of the learning process that the topological ordering of the weight vectors is carried out. The convergence phase is the second phase of the learning that is generally the longest part of the network learning (typically 80% of the epochs). The remaining iterations are necessary during this phase for carrying out fine adjustments of the map i.e., the weight vectors. In this phase the learning parameter should have very small values for a long time. Typically, it is slowly reduced from 0.1 to zero. The neighbourhood radius is kept relatively large at the beginning of the training and then shrunk monotonically with the epochs to zero. The learning rule drags the weight vector associated with the winning unit towards the input vector and it also drags the weights of the closest unit (i.e., those within its neighbourhood radius) along with it. We can imagine the working of the SOM as an elastic net in the input space that wants to come as close as possible to the inputs of the network. The elastic net has the topology of the output array and the points of the net can be thought of as having the weights as coordinates.

**DBSCAN:** In the Euclidean space a set can be divided into a subset of connected components. The implementation for partitioning a set of points requires concepts of density, connectivity and boundary. A cluster, defined as a connected dense component, grows in any direction that density leads. Therefore density-based algorithms are capable of discovering clusters of arbitrary shapes. Also this provides a natural protection against outliers. Figure 4 shows some cluster shapes that present a problem for partitioning relocation clustering (e.g., k-means), but are handled properly by density-based algorithms. They also have good scalability.

The algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) (Ester et al., 1996) targeting low dimensional spatial data is the major representative in density-based connectivity. The other algorithm of this category includes GDBSCAN (Sandar et al., 1998) and DBCLASD (Ester et al., 2000). The basic ideas of density-based clustering involve a number of new definitions. We intuitively present these definitions and then follow up with a high-level algorithm.

- The $\varepsilon$-neighbourhood of a point $p$, denoted by $\varepsilon(p)$ and $N_\varepsilon(p)$ is defined as $N_\varepsilon(p) = \{ q \in D | d(p,q) \leq \varepsilon \}$.
- There are two kinds of points in a cluster, points inside of the cluster (core points) and points on the border of the cluster (border points). In general, $\varepsilon$-neighbourhood of a border point contains significantly less points than $\varepsilon$-neighbourhoods of a core point.
- A point $p$ is directly density-reachable from a point $q$ with respect to $\varepsilon$ and MinPts if
  i. $p \in N_\varepsilon(q)$ and
  ii. $|N_\varepsilon(q)| \geq$ MinPts (core point condition).
- A point $p$ is density reachable from a point $q$ with respect to $\varepsilon$ and MinPts if there is a chain of points $p_0, ..., p_n, p_n = q, p_0 = p$ such that $p_i$ is directly density-reachable from $p_{i-1}$.
- A point $p$ is density connected to a point $q$ with respect to $\varepsilon$ and MinPts if there is a point $O$ such that both, $p$ and $q$ are density-reachable from $O$ with respect to $\varepsilon$ and MinPts.

To find a cluster, DBSCAN starts with an arbitrary point $p$ and retrieves all points density-reachable from $p$ with respect to $\varepsilon$ and MinPts. If $p$ is a core point, this procedure yields a cluster. If $p$ is a border point, no points are density-reachable from $p$ and DBSCAN visits the next point of the dataset.

**Algorithm DBSCAN**

**Input:** i) A dataset denoted as SetOfPoints of points that are UNCLASSIFIED.
   ii) The global density parameters $\varepsilon$ and MinPts.

**Output:** Dense clusters and noise points.

1. ClusterId = 1;
2. For $i = 1$ to SetOfPoints.size do
3.   Point = SetOfPoints.get(i);
4.   If Point.ClId = UNCLASSIFIED then
5.     If ExpandCluster(SetOfPoints, Point, ClusterId, $\varepsilon$, MinPts) then
6.       ClusterId = ClusterId + 1;

Fig. 4: Irregular shapes of clusters
Experimeantal Studies

Description of the dataset: Experimental studies are performed on two artificially created data sets named as Art_data_1 and Art_data_2 and one real life dataset named as IRIS (Blake et al., 1998).

Art_data_1: This data set contains two non-convex clusters in 2D. Cluster 1 contains 7534 samples and cluster 2 contains 7346 samples. Each sample has two features. The underlying dataset contains two non-overlapping clusters. Figure 5 shows the distribution of the dataset.

Art_data_2: This dataset contains two linearly separable clusters of 500 samples each. The samples are a vector of three numeric feature values i.e. all the samples are in 3D. The classes are cubic shaped, with one having its center at (4,4,4) and sides of length 1 and the other having its center at (6.5,6.5,6.5) and sides of length 1.5. The two clusters are fairly close to each other but not overlap.
They are also of much different densities. The Art_data_2 is shown in Fig. 6.

**Iris data set:** The Iris data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. There are 4 attributes—sepal length, sepal width, petal length and petal width. One class is linearly separable from the other 2; the latter are not linearly separable from each other. To visualize the IRIS data set, the 4 attributes have been plotted in 2 triplets, as shown in Fig. 7.

**RESULTS**

The percentage of samples correctly clustered is calculated by using the true class labels of the samples as given in the data set. Each cluster discovered is assigned with the class label of the class to which the majority of the samples in it belong. Then the percentage of samples in clusters with the same class label as the actual class of the sample is calculated. The CPU time is obtained on a Pentium III 500MHz workstation with 256MB RAM.

**K-means:** Table 1 gives the results of the K-means algorithm obtained from all three data sets by taking 1000 iterations.

<table>
<thead>
<tr>
<th>Data set</th>
<th>No. of clusters (k)</th>
<th>Initial weight vectors</th>
<th>Cluster quality ((\mu))</th>
<th>Percent correct</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art_data_1</td>
<td>2</td>
<td>(302,100), (301,100)</td>
<td>1.5714675687911916</td>
<td>65.66</td>
<td>59</td>
</tr>
<tr>
<td>Art_data_2</td>
<td>2</td>
<td>1st two samples:</td>
<td>3.8910738021701077</td>
<td>99.9</td>
<td>6</td>
</tr>
<tr>
<td>Iris</td>
<td>3</td>
<td>1st three samples:</td>
<td>8.6240273263474284</td>
<td>88.67</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>No. of clusters (k)</th>
<th>Initial cluster centers</th>
<th>Cluster quality ((\mu))</th>
<th>Percent correct</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art_data_1</td>
<td>2</td>
<td>1st two samples:</td>
<td>1.7481265202402723</td>
<td>57.25</td>
<td>21</td>
</tr>
<tr>
<td>Art_data_2</td>
<td>2</td>
<td>1st two samples:</td>
<td>3.8910738021701077</td>
<td>99.9</td>
<td>4</td>
</tr>
<tr>
<td>Iris</td>
<td>3</td>
<td>1st three samples:</td>
<td>8.624888765260416</td>
<td>99.9</td>
<td>4</td>
</tr>
</tbody>
</table>

**Self organizing feature map:** Figure 8-10 shows the topographical map of 30 by 30-rectangular grid of output neurons obtained from three datasets. In this method, initially all the weight vectors are initialised to the first sample of the dataset and in the successive iterations the weights are adjusted. Table 2 gives the results of the SOM algorithm using all three data sets by taking 1000 iterations.

**DBSCAN:** The sorted k-dist graphs for the three data sets are shown in Fig. 11-13.

Based on the estimate made using the sorted k-dist graphs and some trial and error search around that value, fairly good density parameter values were chosen for each data set. The results of clustering by DBSCAN using those parameter values are shown in Table 3.

**Comparative analysis:** Table 4 shows the comparative performance of the three clustering algorithms, K-Means, SOM and DBSCAN, for each of the three data sets. DBSCAN could discover the clusters in the Art_data_1 data set with good accuracy while K-means and SOM were unable to cluster it properly. This shows that DBSCAN can detect non-convex clusters while K-means and SOM cannot. However, the DBSCAN algorithm takes considerably more time than K-means and SOM. The
Fig. 8: Topological similarity map for Art_data_1
(Number of epochs: 1000)

Fig. 9: Topological similarity map for Art_data_2
(Number of epochs: 1000)

Fig. 10: Topological similarity map for IRIS (Number of epochs: 1000)

Fig. 11: Sorted 4-dist graph for Art_data_1

Fig. 12: Sorted 4-dist graph for Art_data_2

Fig. 13: Sorted 4-dist graph for IRIS
Table 4: Comparative performance of clustering algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No. of clusters found</th>
<th>Cluster quality ($\beta$)</th>
<th>Percent correct</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art_data_1 artificial data set</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td>2</td>
<td>1.7481265202402723</td>
<td>57.25</td>
<td>21</td>
</tr>
<tr>
<td>SOM</td>
<td>2</td>
<td>1.37146790879791916</td>
<td>63.66</td>
<td>50</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>2</td>
<td>1.17645096580120072</td>
<td>90.50</td>
<td>168</td>
</tr>
<tr>
<td>Art_data_2 artificial data set</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td>2</td>
<td>3.891073821701077</td>
<td>99.9</td>
<td>4</td>
</tr>
<tr>
<td>SOM</td>
<td>2</td>
<td>3.891073821701077</td>
<td>99.9</td>
<td>6</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>2</td>
<td>3.8885508293751793</td>
<td>100.0</td>
<td>8</td>
</tr>
<tr>
<td>IRIS data set</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td>3</td>
<td>8.624488765269416</td>
<td>99.9</td>
<td>4</td>
</tr>
<tr>
<td>SOM</td>
<td>3</td>
<td>8.624027263474284</td>
<td>88.67</td>
<td>3</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>3</td>
<td>12.053916412888822</td>
<td>78.0</td>
<td>4</td>
</tr>
</tbody>
</table>

Cluster quality index $\beta$ for DBSCAN is lower because the actual clusters were not very homogeneous. The well separated convex clusters in Art_data_2 data set were discovered by all three algorithms equally well with DBSCAN taking the most CPU time followed by SOM and then K-means. Again DBSCAN has a slightly lower $\beta$ value but that resulted in a higher percentage of samples getting correctly clustered.

For the Iris data set, where two of the clusters are not linearly separable, DBSCAN has the least percentage of samples correctly classified as it misclassified some of the sample as noise when there is no noise in the data set. However, its $\beta$ value is much higher than the others because it removed some of the input data that were increasing the non-homogeneity as noise. So, DBSCAN is able to cluster non-linearly separable clusters better than or as well as K-means and SOM, particularly when the clusters are not convex or linearly separable.

It is difficult to determine the actual number of clusters for K-means and SOM particularly when the clusters are non-convex or non-linearly separable. The topological maps produced by the SOM can be used to determine the number of clusters when the clusters are well separated and linearly separable, but they are unable to show clusters that are not linearly separable. The number of clusters in Art_data_1 is not apparent from its topological map, which shows 3 separate regions of high similarity. Similarly, for Iris data set, only one region of high similarity is visible in its topological map, corresponding to the single linearly separable cluster in the data set. The other regions do not show a high similarity within it. The topological similarity map for Art_data_2 data set correctly shows the two clusters in the data set.

CONCLUSIONS AND FUTURE WORK

This study presents a comparative study of the clustering technique like K-means, SOM and DBSCAN based on the extensive simulation using the data set Art_data_1, Art_data_2 and IRIS. Our study indicate that DBSCAN is better than K-means and SOM in discovering non-convex clusters and it is as good or better than K-means and SOM in extracting convex clusters. It can also detect noise. However, DBSCAN algorithm also takes much more CPU time for large data sets. When the clusters are of arbitrary shape (e.g., Art_data_1 data set), a density based clustering algorithm like DBSCAN is preferable. On the other hand, when the clusters are of hyperspherical or convex shape and well separated and the data set is large, then SOM or K-means may be preferable as they are faster.

Future work includes comparing the performance of K-means, SOM, DBSCAN and hierarchical clustering algorithms using incomplete data sets.

REFERENCES


