Online Clustering Algorithms

Wesam Barbakh and Colin Fyfe,
The University of Paisley,
Scotland.
email:wesam.barbakh,colin.fyfe@paisley.ac.uk

Abstract

We introduce a set of clustering algorithms whose performance function is such that the algorithms overcome one of the weaknesses of K-means, its sensitivity to initial conditions which leads it to converge to a local optimum rather than the global optimum. We derive online learning algorithms and illustrate their convergence to optimal solutions which K-means fails to find. We then extend the algorithm by underpinning it with a latent space which enables a topology preserving mapping to be found. We show visualisation results on some standard data sets.

1 Introduction

The K-means algorithm is one of the most frequently used investigatory algorithms in data analysis. The algorithm attempts to locate K prototypes or means throughout a data set in such a way that the K prototypes in some way best represents the data. The algorithm is one of the first which a data analyst will use to investigate a new data set because it is algorithmically simple, relatively robust and gives 'good enough' answers over a wide variety of data sets: it will often not be the single best algorithm on any individual data set but it may be close to the optimal over a wide range of data sets. However the algorithm is known to suffer from the defect that the means or prototypes found depend on the initial values given to them at the start of the simulation: a typical program will converge to a local optimum. There are a number of heuristics in the literature which attempt to address this issue but, at heart, the fault lies in the performance function on which K-means is based. In this paper, we investigate alternative performance functions and show the effect the different functions have on the effectiveness of the resulting algorithms. We are specifically interested in developing algorithms which are effective in a worst case scenario: when the prototypes are initialised at the same position which is very far from the data points. If an algorithm can cope with this scenario, it should be able to cope with a more benevolent initialisation.

2 A family of new algorithms

In this paper we introduce a new family of algorithms that solve the problem of sensitivity to initial conditions in K-means. The central idea in the new algorithms is that each prototype before moving to any new location takes account of all the other prototypes' positions and, in particular, to their relative locations with respect to the data points, and hence it is possible for it to identify the free clusters that are not recognized by the other prototypes. For example if we have two data points (each data point represents a cluster) and two prototypes, one of them being closer to the first data point, the other prototype will, before responding, recognize that there is one prototype closer to the first data point and hence will prefer to move toward the second point.

2.1 Weighted K-means Algorithm (WK)

We have previously investigated this effect in detail in [3, 2]. Given an input data samples $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$, where $\mathbf{x}_i \in \Re^d$, and prototypes $(\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_K)$, where $\mathbf{m}_l \in \Re^d$, the performance function for K-means may be written as

$$J_K = \sum_{i=1}^{N} \min_{j=1}^{K} \parallel \mathbf{x}_i - \mathbf{m}_j \parallel^2 \tag{1}$$

which we wish to minimise by moving the prototypes to the appropriate positions. Alternatively, we may write

$$J_K = \sum_{i=1}^{N} \sum_{j=1}^{K} r_{ji} \parallel \mathbf{x}_i - \mathbf{m}_j \parallel^2 \tag{2}$$

where $r_{ji} = 1$ for $j = \arg\min_k \parallel \mathbf{x}_i - \mathbf{m}_k \parallel$ and 0 otherwise. Note that (1) detects only the prototypes closest to data points and then distributes them to give the minimum performance which determines the clustering. Any prototype which is still far from data is not utilised and does not enter any calculation that gives minimum performance. This may result in dead prototypes, which are never appropriate for any cluster. Thus initializing prototypes appropriately can have a big effect in K-means.
We might consider the following performance function:

$$J_A = \sum_{i=1}^{N} \sum_{j=1}^{K} \parallel \mathbf{x}_i - \mathbf{m}_j \parallel^2 \tag{3}$$

which provides a relationship between all the data points and prototypes, but it doesn't provide useful clustering at minimum performance since

$$\frac{\partial J_A}{\partial \mathbf{m}_k} = 0 \Longrightarrow \mathbf{m}_k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i, \forall k \tag{4}$$

Minimizing this performance function groups all the prototypes to the centre of data set regardless of the intitial position of the prototypes which is useless for identification of clusters.

We wish to form a performance function with the following properties:

- Minimum performance gives an intuitively 'good' clustering.
- It creates a relationship between all data points and all prototypes.

(3) provides an attempt to reduce the sensitivity to prototypes' initialization by making a relationship between all data points and all prototypes while (1) provides an attempt to cluster data points at the minimum of the performance function. (1) is too local while (3) is too global. Therefore it may seem that what we want is to combine features of (1) and (3) to make a performance function such as:

$$J_1 = \sum_{i=1}^{N} \left[\sum_{j=1}^{K} \| \mathbf{x}_i - \mathbf{m}_j \| \right] \min_{k=1}^{K} \| \mathbf{x}_i - \mathbf{m}_k \|^2 . \tag{5}$$

The rationale behind this performance function is that we wish to utilise the minimum distance in our learning algorithm but retain the global interaction which is necessary to ensure all prototypes play a part in the clustering. We derive the clustering algorithm associated with this performance function by calculating the partial derivatives of (5) with respect to the prototypes. We call the resulting algorithm Weighted K-means (though recognising that other weighted versions of K-means have been developed in the literature). Let

$$k* = \arg \min_{k=1}^{K} (\| \mathbf{x}_i - \mathbf{m}_k \|).$$

Then

$$\frac{\partial J_{1,i}}{\partial \mathbf{m}_{k*}} = -(\mathbf{x}_i - \mathbf{m}_{k*}) \{ \| \mathbf{x}_i - \mathbf{m}_{k*} \| + 2 \sum_{j=1}^{K} \| \mathbf{x}_i - \mathbf{m}_j \| \} = -(\mathbf{x}_i - \mathbf{m}_{k*}) a_{ik*} \tag{6}$$

when $\mathbf{m}_{k*}$ is the closest prototype to $\mathbf{x}_i$ and

$$\frac{\partial J_{1,i}}{\partial \mathbf{m}_j} = -(\mathbf{x}_i - \mathbf{m}_j) \frac{\| \mathbf{x}_i - \mathbf{m}_{k*} \|^2}{\| \mathbf{x}_i - \mathbf{m}_j \|} = -(\mathbf{x}_i - \mathbf{m}_j) b_{ij} \tag{7}$$

otherwise.
Note that i determines $k*$, the closest prototype to $\mathbf{x}_i$ and hence the value of b_{ij}. We then solve this by summing over the whole data set and finding the fixed point solution of

$$\frac{\partial J_1}{\partial \mathbf{m}_r} = \sum_{i=1}^{N} \frac{\partial J_{1,i}}{\partial \mathbf{m}_r} = 0 \tag{8}$$

which gives a solution of

$$\mathbf{m}_r(t+1) = \frac{\sum_{i \in V_r} \mathbf{x}_i a_{ir} + \sum_{i \in V_j, j \neq r} \mathbf{x}_i b_{ir}}{\sum_{i \in V_r} a_{ir} + \sum_{i \in V_j, j \neq r} b_{ir}} \tag{9}$$

where V_r contains the indices of data points that are closest to $\mathbf{m}_r$, V_j contains the indices of all the other points and

$$a_{ir} = \| \mathbf{x}_i - \mathbf{m}_r(t) \| + 2 \sum_{j=1}^{K} \| \mathbf{x}_i - \mathbf{m}_j \| \tag{10}$$

$$b_{ir} = \frac{\| \mathbf{x}_i - \mathbf{m}_{k*} \|^2}{\| \mathbf{x}_i - \mathbf{m}_r(t) \|} \tag{11}$$

where again $k* = \arg\min_k \| \mathbf{x}_i - \mathbf{m}_k \|$. We have given extensive analysis and simulations in [3] showing that this algorithm will cluster the data with the prototypes which are closest to the data points being positioned in such a way that the clusters can be identified. However there can be some potential prototypes which are not sufficiently responsive to the data and so never move to identify a cluster. In fact, these points move to (a weighted) centre of the data set. This may be an advantage in some cases in that we can easily identify redundancy in the prototypes however it does waste computational resources unnecessarily.

2.2 Inverse Weighted K-means Algorithm (IWK)

Consider the performance function

$$J_2 = \sum_{i=1}^{N} \left[\sum_{j=1}^{K} \frac{1}{\| \mathbf{x}_i - \mathbf{m}_j \|^p} \right] \min_{k=1}^{K} \| \mathbf{x}_i - \mathbf{m}_k \|^n . \tag{12}$$

The rationale for this performance function is that we do not wish the situation to arise in which one prototype is optimally responsive to a data point but all the other prototypes are ignoring this data point: we wish all prototypes to take into account all data points. Let $\mathbf{m}_{k*}$ be the closest prototype to $\mathbf{x}_i$. Then

$$\begin{aligned} J_2(\mathbf{x}_i) &= \left[\sum_{j=1}^{K} \frac{1}{\| \mathbf{x}_i - \mathbf{m}_j \|^p} \right] \| \mathbf{x}_i - \mathbf{m}_{k*} \|^n \\ &= \| \mathbf{x}_i - \mathbf{m}_{k*} \|^{n-p} + \sum_{j \neq k*} \frac{\| \mathbf{x}_i - \mathbf{m}_{k*} \|^n}{\| \mathbf{x}_i - \mathbf{m}_j \|^p}. \end{aligned} \tag{13}$$

Therefore the intuition behind this performance function is that it is minimised when the distance between the data and the closest prototype is minimal and the other prototypes are scattered widely throughout the data to give the largest

possible value in the denominator of the second part. Also writing the performance function in this manner highlights the constraints which will be necessary on the values of p and n (see below). To create a learning rule, we calculate

$$\begin{aligned}\frac{\partial J_2(\mathbf{x}_i)}{\partial \mathbf{m}_{k*}} &= -(n-p)(\mathbf{x}_i - \mathbf{m}_{k*}) \parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^{n-p-2} \\ &\quad -n(\mathbf{x}_i - \mathbf{m}_{k*}) \parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^{n-2} \sum_{j \neq k*} \frac{1}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel^p} \\ &= (\mathbf{x}_i - \mathbf{m}_{k*}) a_{ik*} \qquad (14) \\ \frac{\partial J_2(\mathbf{x}_i)}{\partial \mathbf{m}_j} &= p(\mathbf{x}_i - \mathbf{m}_j) \frac{\parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^n}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel^{p+2}} = (\mathbf{x}_i - \mathbf{m}_j) b_{ij}. \qquad (15)\end{aligned}$$

At convergence, $E(\frac{\partial J_2}{\partial \mathbf{m}_r}) = 0$ where the expectation is taken over the data set. With some abuse of our notation, we denote by V_r the set of points, $\mathbf{x}$ for which $\mathbf{m}_r$ is the closest, we have

$$\begin{aligned}\frac{\partial J_2}{\partial \mathbf{m}_r} = 0 \iff & \int_{\mathbf{x} \in V_r} \{(n-p)(\mathbf{x} - \mathbf{m}_r) \parallel \mathbf{x} - \mathbf{m}_r \parallel^{n-p-2} \\ & +n(\mathbf{x} - \mathbf{m}_r) \parallel \mathbf{x} - \mathbf{m}_r \parallel^{n-2} \sum_{j \neq r} \frac{1}{\parallel \mathbf{x} - \mathbf{m}_j \parallel^p} P(\mathbf{x})\} d\mathbf{x} \\ + & \sum_{j \neq r} \int_{\mathbf{x} \in V_j} p(\mathbf{x} - \mathbf{m}_j) \frac{\parallel \mathbf{x} - \mathbf{m}_r \parallel^n}{\parallel \mathbf{x} - \mathbf{m}_j \parallel^{p+2}} P(\mathbf{x}) d\mathbf{x} = 0 \qquad (16)\end{aligned}$$

where $P(\mathbf{x})$ is the probability measure associated with the data set. This is, in general, a very difficult set of equations to solve. However it is readily seen that, for example, in the special case that there are the same number of prototypes as there are data points, that one solution is to locate each prototype at each data point (at which time $\frac{\partial J_2}{\partial \mathbf{m}_r} = 0$). Again solving this over all the data set results in

$$\mathbf{m}_r(t+1) = \frac{\sum_{i \in V_r} \mathbf{x}_i a_{ir} + \sum_{i \in V_j, j \neq r} \mathbf{x}_i b_{ir}}{\sum_{i \in V_r} a_{ir} + \sum_{i \in V_j, j \neq r} b_{ir}} \qquad (17)$$

where V_r contains the indices of data points that are closest to $\mathbf{m}_r$, V_j contains the indices of all the other points and

$$\begin{aligned}a_{ir} &= -(n-p) \parallel \mathbf{x}_i - \mathbf{m}_r(t) \parallel^{n-p-2} \\ &\quad -n \parallel \mathbf{x}_i - \mathbf{m}_r(t) \parallel^{n-2} \sum_{j \neq k*} \frac{1}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel^p} \qquad (18) \\ b_{ir} &= p \frac{\parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^n}{\parallel \mathbf{x}_i - \mathbf{m}_r(t) \parallel^{p+2}}. \qquad (19)\end{aligned}$$

From (16), we see that $n \geq p$ if the direction of the first term is to be correct and $n \leq p+2$ to ensure stability in all parts of that equation. In practice, we have found that a viable algorithm may be found by using (19) for all prototypes (and thus never using (18) for the closest prototype). We will call this the Inverse Weighted K-means Algorithm (IWK).

2.2.1 Simulations

In Figure 1, the prototypes have all been initialized within a single cluster. As shown in the Figure, while the K-means algorithm failed to identify the clusters, the new algorithm IWK identified them successfully.

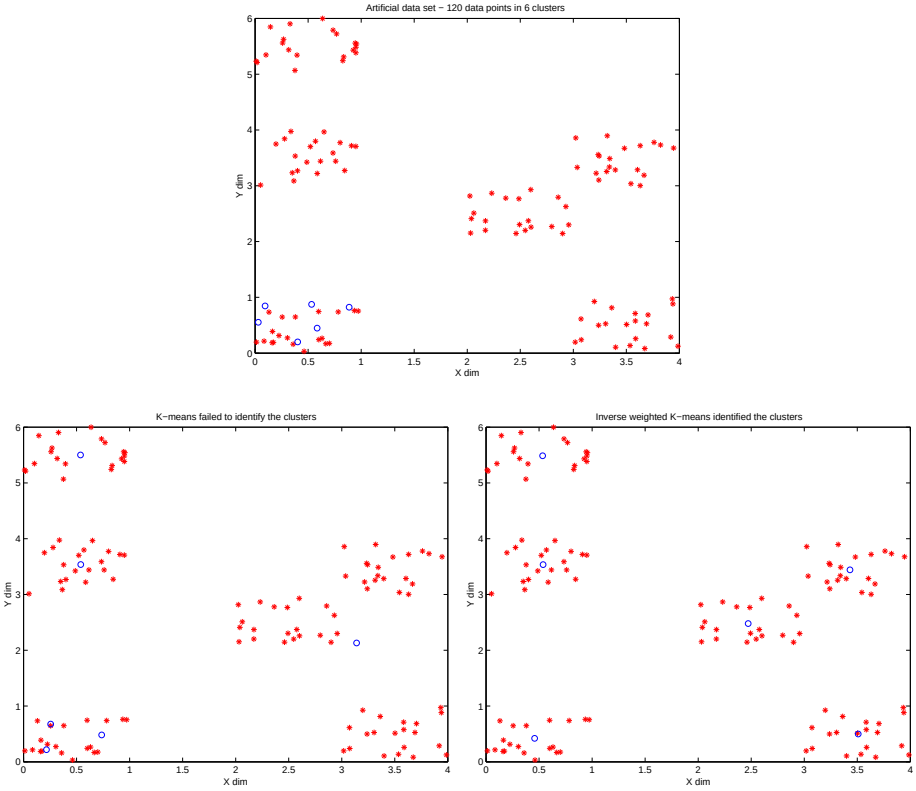

Figure 1: Top: Artificial data set: data set is shown as 6 clusters of red '*'s, prototypes are initialized to lie within one cluster and shown as blue 'o's. Bottom left: K-means result. Bottom right: IWK result.

Figure 2 shows the result of applying K-means and inverse weighted K-means algorithms to the same artificial data set but with bad initialization of the prototypes. The K-means failed to identify the clusters and there are 3 dead prototypes due to the bad initialization. Inverse weighted K-means succeeded in identifying the clusters under this bad initialization. In Figure 3 and Figure 4, the prototypes are initialized in the same location and very far from the data. In general this is an unlikely situation to happen, but it may be that all the prototypes are in fact initialized very far from a particular cluster. We initialized the prototypes in the same location to show that IWK has the ability to separate the joint prototypes because it uses two set of updates, see (17). Also we initialized the prototypes far from the data as this situation could happen

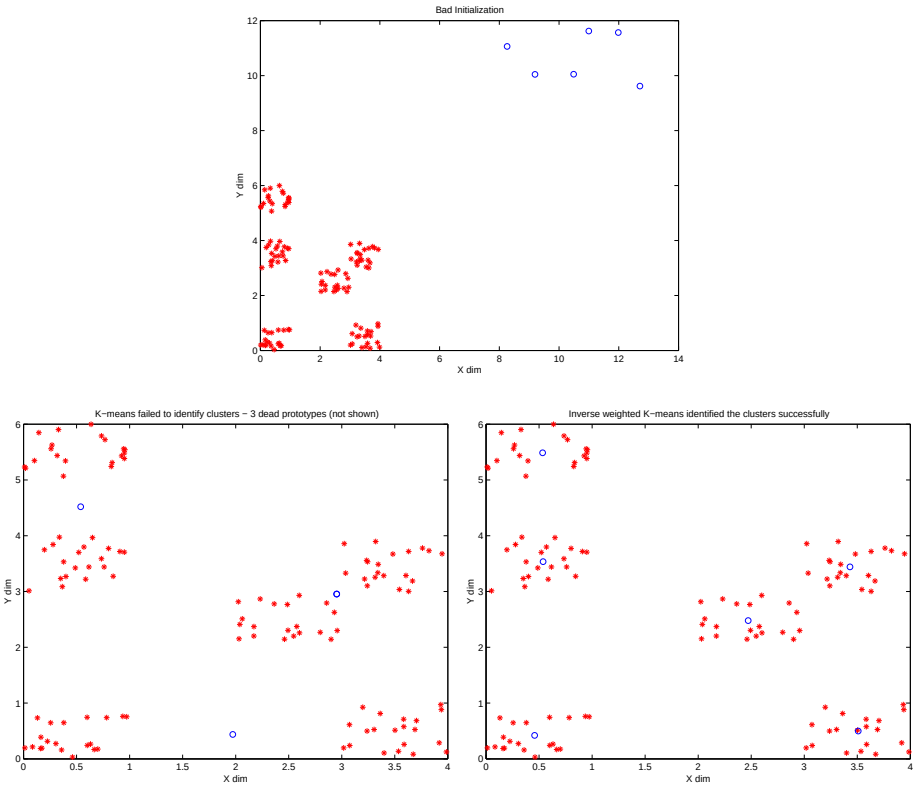

Figure 2: Top: Same artificial data set with bad prototypes' initialization. Bottom left: K-means result. Bottom right: IWK result.

when initializing the prototypes randomly in a high dimensional data set. As shown in Figure 3 and Figure 4, the K-means algorithm failed to identify the clusters. The inverse weighted K-means algorithm has the ability to separate the joint prototypes and then distribute them in such a way as to identify the clusters.

3 Online Clustering Algorithms

In this section, we show how we can extend the previous algorithms and allow them to learn in online mode.

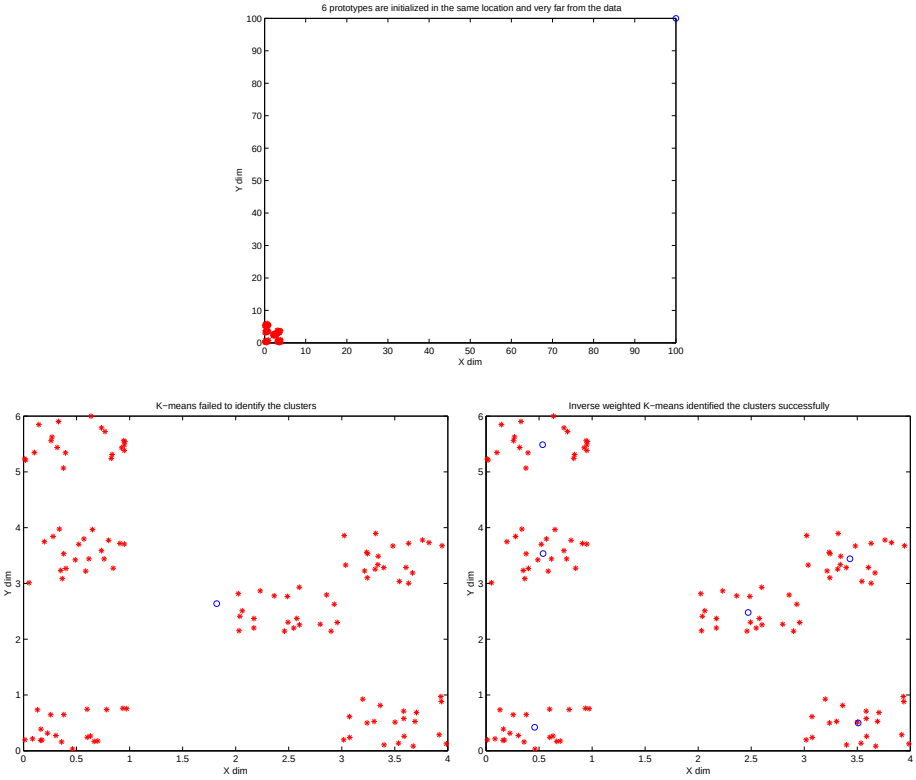

Figure 3: Top: Prototypes are initialized in same location and far from the data points at (100,100). Bottom left: K-means result. Bottom right: IWK result.

3.1 Online K-means Algorithm

The performance function for K-Means may be written as

$$J_1 = \sum_{i=1}^{N} \min_{j=1}^{K} \| \mathbf{x}_i - \mathbf{m}_j \|^2 \tag{20}$$

The implementation of the online K-means algorithm is as follows:

1. Initialization: initialize the cluster prototype vectors $\mathbf{m}_1, ..., \mathbf{m}_k$
2. Loop for M iterations:
   (a) for each data vector $\mathbf{x}_i$, set

$$k^* = \arg \min_{k=1}^{K} (\| \mathbf{x}_i - \mathbf{m}_k \|)$$

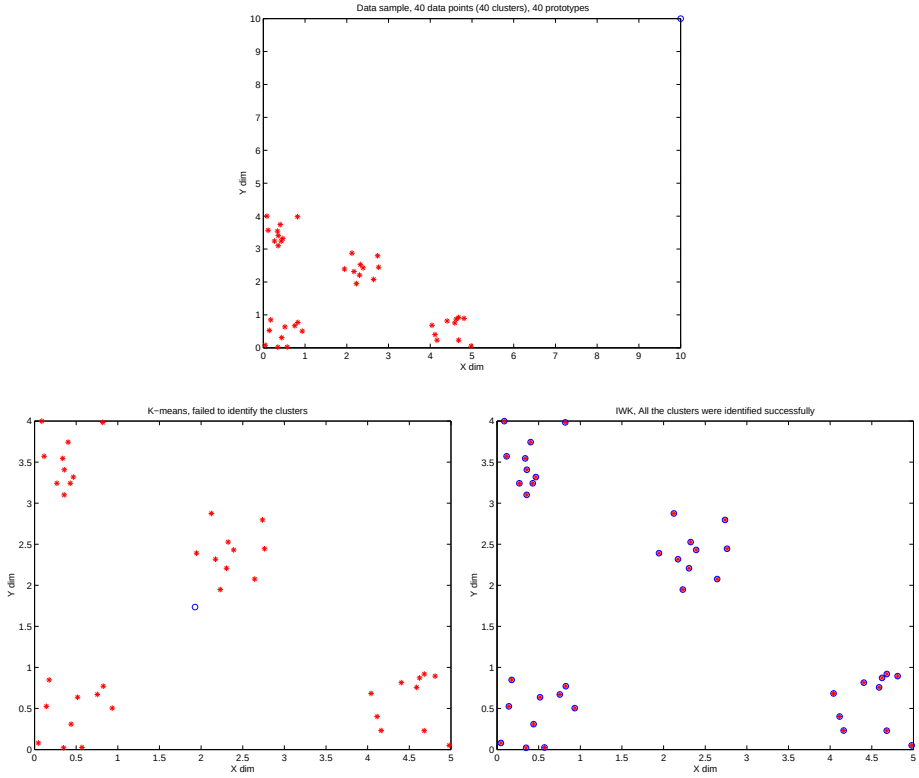

Figure 4: Top: Prototypes are initialized in same location and far from the data points at (10,10). Bottom left: K-means result. Bottom right: IWK result.

(b) update the prototype $\mathbf{m}_{k^*}$ as

$$\mathbf{m}_{k^*}^{(new)} = \mathbf{m}_{k^*} - \zeta \frac{\partial J_1}{\partial \mathbf{m}_{k^*}} = \mathbf{m}_{k^*} + \zeta(\mathbf{x}_i - \mathbf{m}_{k^*})$$

where ζ is a learning rate usually set to be a small positive number (e.g.,0.05).

The learning rate can also gradually decrease during the learning process.

3.2 IWK Online Algorithm v1 (IWKO1)

As shown in (17), in batch mode we have:

$$\mathbf{m}_k = \frac{a_1\mathbf{x}_1 + ... + a_N\mathbf{x}_N}{a_1 + ... + a_N} \tag{21}$$

where,

$$a_i = \begin{cases} \left(-(n-p) \parallel \mathbf{x}_i - \mathbf{m}_k \parallel^{n-p-2} -n \parallel \mathbf{x}_i - \mathbf{m}_k \parallel^{n-2} \sum_{j \neq k*} \frac{1}{\|\mathbf{x}_i - \mathbf{m}_j\|^p}\right) & \text{if } \mathbf{m}_k \text{ is closest to } \mathbf{x}_i \\ \left(\frac{\|\mathbf{x}_i - \mathbf{m}_{k*}\|^n}{\|\mathbf{x}_i - \mathbf{m}_k\|^{p+2}}\right) & \text{otherwise} \end{cases}$$

$\mathbf{m}_{k^*}$ is the closest prototype to $\mathbf{x}_i$

3.2.1 Implementation (Online Mode)

In online mode, for IWK we can do something similar to (21) (taking into account that we receive one input sample at a time) as follows:
1. Initialization:
- initialize the cluster prototype vectors $\mathbf{m}_1, ..., \mathbf{m}_k$
- initialize one dimensional array with k elements to one, $v_1 = 1, ..., v_k = 1$
note: v_k will represent the value that should be in denominator of (21) after feeding the input sample (it is used also for normalization).

2. Loop for M iterations:
- for each data vector $\mathbf{x}_i$, set

$$k^* = \arg \min_{k=1}^{K} (\parallel \mathbf{x}_i - \mathbf{m}_k \parallel)$$

and update all the prototypes $\mathbf{m}_k$ as

$$\begin{aligned} \mathbf{m}_k^{(new)} &= \mathbf{m}_k v_k \\ \mathbf{m}_k^{(new)} &= \mathbf{m}_k + a_i \mathbf{x}_i \\ v_k &= v_k + a_i \\ \mathbf{m}_k^{(new)} &= \frac{\mathbf{m}_k}{v_k} \end{aligned}$$

To clarify how this implementation is going to build (21), we will feed two data points as an example. After feeding the first data point $\mathbf{x}_1$, we will have the following:

$$\begin{aligned} \mathbf{m}_k^{(new)} &= \mathbf{m}_k^{(old)} * 1 \\ \mathbf{m}_k^{(new)} &= (\mathbf{m}_k^{(old)} * 1) + a_1 \mathbf{x}_1 \\ v_k &= 1 + a_1 \\ \mathbf{m}_k^{(new)} &= \frac{(\mathbf{m}_k^{(old)} * 1) + a_1 \mathbf{x}_1}{1 + a_1} \end{aligned}$$

After feeding the second data point $\mathbf{x}_2$, we will have the following:

$$\begin{aligned}
\mathbf{m}_k^{(new)} &= \left(\frac{(\mathbf{m}_k^{(old)} * 1) + a_1 \mathbf{x}_1}{1 + a_1} \right) (1 + a_1) \\
\mathbf{m}_k^{(new)} &= \left(\frac{(\mathbf{m}_k^{(old)} * 1) + a_1 \mathbf{x}_1}{1 + a_1} \right) (1 + a_1) + a_2 \mathbf{x}_2 \\
v_k &= (1 + a_1) + a_2 \\
\mathbf{m}_k^{(new)} &= \frac{\left(\frac{(\mathbf{m}_k^{(old)} * 1) + a_1 \mathbf{x}_1}{1 + a_1} \right) (1 + a_1) + a_2 \mathbf{x}_2}{(1 + a_1) + a_2} \\
&= \frac{\mathbf{m}_k^{(old)} + a_1 \mathbf{x}_1 + a_2 \mathbf{x}_2}{1 + a_1 + a_2}
\end{aligned}$$

And so on in the same way for all data points.

3.2.2 Simulation

Figure 5, right, shows the results after applying IWKO1 to the artificial data set in Figure 5, left. The disadvantage of this algorithm that it needs a large

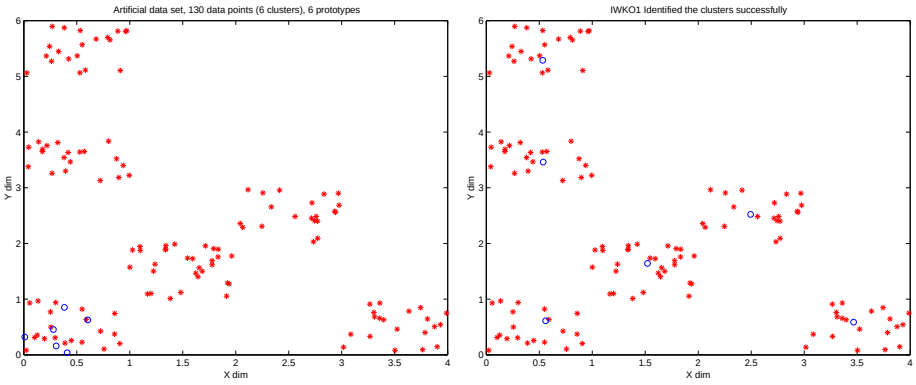

Figure 5: Left: Artificial data set: data set is shown as 6 clusters of red '*'s, prototypes are initialized to lie within one cluster and shown as blue 'o's. Right: IWKO1 succeeds in identifying all the clusters

number of iterations. This problem doesn't exist in the next version.

3.3 IWK Online Algorithm v2 (IWKO2)

In this section we show how it is possible to allow all the units (prototypes) to learn, not only the winner as in K-means or the winner with its neighbors as in SOM. In this algorithm, it is not necessary to specify any functions for the neighbors as all units learn with every input sample.

3.3.1 Implementation (Online Mode)

1. Initialization:
- initialize the cluster prototype vectors $\mathbf{m}_1, ..., \mathbf{m}_k$

2. Loop for M iterations:
- from (14) and (15) and ($p = -1$) we have:

$$\Delta \mathbf{m}_{k^*} = -\zeta(\mathbf{x}_i - \mathbf{m}_{k^*})a_{ik^*} \quad (22)$$

$$\Delta \mathbf{m}_k = -\zeta(\mathbf{x}_i - \mathbf{m}_k)b_{ik} \quad (23)$$

- for each data vector $\mathbf{x}_i$, set

$$k^* = \arg\min_{k=1}^{K}(\parallel \mathbf{x}_i - \mathbf{m}_k \parallel)$$

and update all the prototypes $\mathbf{m}_k$ as

$$\mathbf{m}_{k^*}^{(new)} = \mathbf{m}_{k^*} - \zeta\left(-(n+1) \parallel \mathbf{x}_i - \mathbf{m}_{k^*} \parallel^{n-1} -n \parallel \mathbf{x}_i - \mathbf{m}_{k^*} \parallel^{n-2} \sum_{j \neq k^*} \frac{1}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel^{-1}}\right)(\mathbf{x}_i - \mathbf{m}_{k^*})$$

$$\mathbf{m}_k^{(new)} = \mathbf{m}_k - \zeta\left(\frac{- \parallel \mathbf{x}_i - \mathbf{m}_{k^*} \parallel^n}{\parallel \mathbf{x}_i - \mathbf{m}_k \parallel}\right)(\mathbf{x}_i - \mathbf{m}_k)$$

where ζ is a learning rate usually set to be a small positive number (e.g.,0.05). The number can also gradually decrease during the learning process.
Note: if we want to choose a bigger value for n, we will need to choose smaller and smaller value for ζ (thus it is better to choose $n = 1$).

3.3.2 Simulation

We applied IWKO2 and K-means to some artificial data sets. We found K-means failed to identify all the clusters due to the bad initialization while IWKO2 algorithm succeeded. Figure 7 shows the results after applying K-means, left, and IWKO2, right, to the artificial data set shown in Figure 6.

Note: K-means can succeed with this artificial data set by choosing perfect prototypes' initialization, but we want to show that the new algorithm IWKO2 still works well even if we have a bad initialization in which K-means failed.

Note: for this algorithm, it is also possible to work in the same manner as the SOM and update the winner with its neighbors instead of updating all the units. In this way, we can create an algorithm that has the advantages of both SOM and IWKO2.

3.4 K-Harmonic Means - Online mode algorithm (KHMO)

In K-Harmonic Means we have the following performance function:

$$J_{HA} = \sum_{i=1}^{N} \frac{K}{\sum_{k=1}^{K} \frac{1}{\|\mathbf{x}_i - \mathbf{m}_k\|^2}}. \quad (24)$$

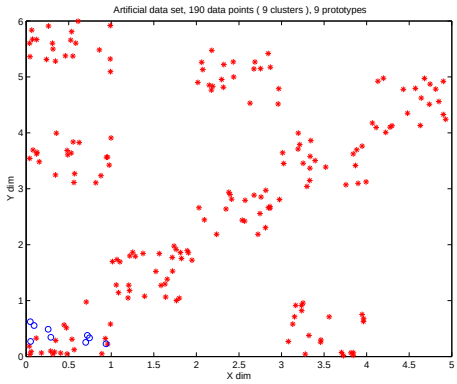

Figure 6: Artificial data set: data set is shown as 6 clusters of red '*'s, prototypes are initialized to lie within one cluster and shown as blue 'o's.

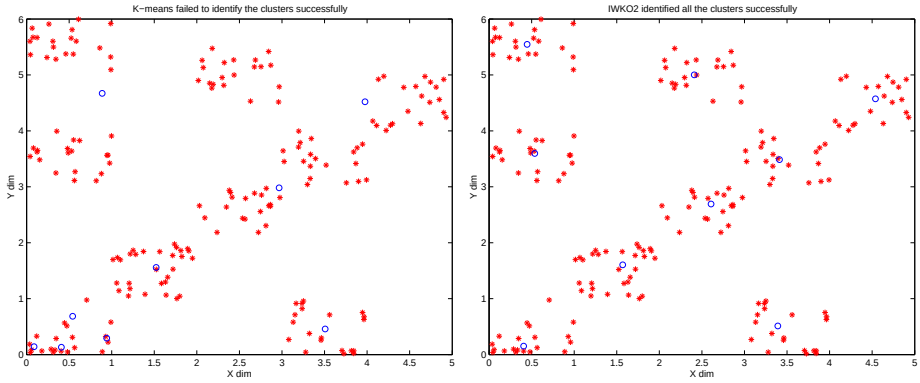

Figure 7: Left: K-means results. Right: IWKO2 succeeds in identifying all the clusters.

Then we wish to move the prototypes in such a way to minimize the performance function and hence identify the clusters

$$\frac{\partial J_{HA}}{\partial \mathbf{m}_k} = -K \sum_{i=1}^{N} \frac{2(\mathbf{x}_i - \mathbf{m}_k)}{\| \mathbf{x}_i - \mathbf{m}_k \|^4 \{\sum_{l=1}^{K} \frac{1}{\|\mathbf{x}_i - \mathbf{m}_l\|^2}\}^2} \tag{25}$$

Setting this equal to 0 and "solving" for the $\mathbf{m}_k$'s, we get a recursive formula (Batch Mode)

$$\mathbf{m}_k^{(new)} = \frac{\sum_{i=1}^{N} \frac{1}{d_{i,k}^4 (\sum_{l=1}^{K} \frac{1}{d_{i,l}^2})^2} \mathbf{x}_i}{\sum_{i=1}^{N} \frac{1}{d_{i,k}^4 (\sum_{l=1}^{K} \frac{1}{d_{i,l}^2})^2}} \tag{26}$$

where we have used $d_{i,k}$ for $\| \mathbf{x}_i - \mathbf{m}_k \|$ to simplify the notation. There are some practical issues to deal with in the implementation details of which are given in [11, 10].

[11] have extensive simulations showing that this algorithm converges to a better solution (less prone to finding a local minimum because of poor initialization) than both standard K-means or a mixture of experts trained using the EM algorithm.

3.4.1 Implementation (Online Mode)

From (25) we have:

$$\frac{\partial J_{HA}(\mathbf{x}_i)}{\partial \mathbf{m}_k} = -K \frac{2(\mathbf{x}_i - \mathbf{m}_k)}{\| \mathbf{x}_i - \mathbf{m}_k \|^4 \{\sum_{l=1}^{K} \frac{1}{\|\mathbf{x}_i - \mathbf{m}_l\|^2}\}^2} \tag{27}$$

$$\begin{aligned} \mathbf{m}_k^{(new)} &= \mathbf{m}_k + \zeta \left(\frac{2K}{\| \mathbf{x}_i - \mathbf{m}_k \|^4 \{\sum_{l=1}^{K} \frac{1}{\|\mathbf{x}_i - \mathbf{m}_l\|^2}\}^2} \right) (\mathbf{x}_i - \mathbf{m}_k) \\ &= \mathbf{m}_k + \zeta \left(\frac{1}{\| \mathbf{x}_i - \mathbf{m}_k \|^4 \{\sum_{l=1}^{K} \frac{1}{\|\mathbf{x}_i - \mathbf{m}_l\|^2}\}^2} \right) (\mathbf{x}_i - \mathbf{m}_k) \end{aligned}$$

where ζ is a learning rate usually set to be a small positive number (e.g.,0.05).

3.4.2 Simulation

Figure 8, right, shows the results after applying KHMO to the artificial data set in Figure 8, left.

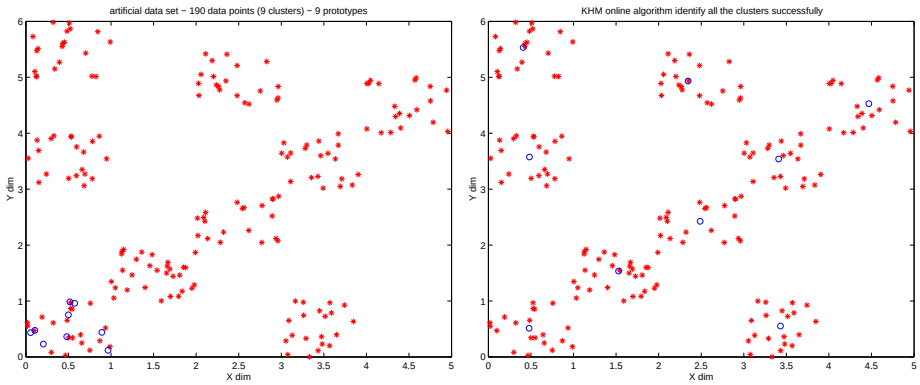

Figure 8: Left: Artificial data set: data set is shown as 9 clusters of red '*'s, prototypes are initialized to lie within one cluster and shown as blue 'o's. Right: KHMO succeeds in identifying all the clusters.

4 A Topology Preserving Mapping

In this section we show how it is possible to extend one of the previous clustering algorithms to provide a new algorithm for visualization and topology-preserving mappings.

4.1 Inverse-weighted K-means (online) Topology-preserving Mapping (IKoToM)

A topographic mapping (or topology preserving mapping) is a transformation which captures some structure in the data so that points which are mapped close to one another share some common feature while points which are mapped far from one another do not share this feature. The Self-organizing Map (SOM) was introduced as a data quantisation method but has found at least as much use as a visualisation tool.

Topology-preserving mappings such as the Self-organizing Map (SOM) [8] and the Generative Topographic Mapping(GTM) [4] have been very popular for data visualization: we project the data onto the map which is usually two dimensional and look for structure in the projected map by eye. We have recently investigated a family of topology preserving mappings [5] which are based on the same underlying structure as the GTM.

The basis of our model is K latent points, $t_1, t_2, \cdots, t_K$, which are going to generate the K prototypes, $\mathbf{m}_k$. To allow local and non-linear modeling, we map those latent points through a set of M basis functions, $f_1(), f_2(), \cdots, f_M()$. This gives us a matrix Φ where $\phi_{kj} = f_j(t_k)$. Thus each row of Φ is the response of the basis functions to one latent point, or alternatively we may state that each column of Φ is the response of one of the basis functions to the set of latent points. One of the functions, $f_j()$, acts as a bias term and is set to one for every input. Typically the others are gaussians centered in the latent space. The output of these functions are then mapped by a set of weights, W, into data space. W is $M \times D$, where D is the dimensionality of the data space, and is the sole parameter which we change during training. We will use $\mathbf{w}_i$ to represent the i^{th} column of W and Φ_j to represent the row vector of the mapping of the j^{th} latent point. Thus each basis point is mapped to a point in data space, $\mathbf{m}_j = (\Phi_j W)^T$.

We may update W either in batch mode or with online learning: with the Topographic Product of Experts [5], we used a weighted mean squared error; with the Harmonic Topographic Mapping [9], we used Harmonic K-Means. We now apply the Inverse Weighted K-means Online algorithm (IWKO) to the same underlying structure to create a new topology preserving algorithm.

Each data point is visualized as residing at the prototype on the map which would win the competition for that data point. However we can do rather better by defining the responsibility that the j^{th} prototype has for the i^{th} data point as

$$r_{ji} = \frac{\exp(-\gamma \parallel \mathbf{x}_i - \mathbf{w}_j \parallel^2)}{\sum_k \exp(-\gamma \parallel \mathbf{x}_i - \mathbf{w}_k \parallel^2)} \tag{28}$$

We then project points taking into account these responsiblities: let y_{ij} be the projection of the i^{th} data point onto the j^{th} dimension of the latent space; then

$$y_{ij} = \sum_k t_{kj} r_{ki} \tag{29}$$

where t_{kj} is the j^{th} coordinate of the k^{th} latent point.

4.2 Inverse-weighted K-means (batch mode) Topology-preserving Mapping (IKToM)

The IKToM algorithm, like teh IKoToM algorithm, has the same underlying structure as the GTM, with a number of latent points that are mapped to a feature space by M Gaussian functions, and then into the data space by a matrix W. Each latent point t indexed by k is mapped, through a set of M fixed basis functions $\phi_1()$, $\phi_2()$,...,$\phi_M()$ to a prototype in data space $m_k = W\phi(t_k)$. But the similarity ends there because the objective function is not a probabilistic function like the GTM neither it is optimised with the Expectation-Maximization (EM) algorithm. Instead, the IKToM uses the well proved clustering abilities of the K-means algorithm, improved by using Inverse Weighted K-means (IWK), batch mode, to make it insensitive to initialisation.

4.3 Simulation

4.3.1 Artificial data set

We create a simulation with 10 latent points deemed to be equally spaced in a one dimensional latent space, passed through 5 Gaussian basis functions and then mapped to the data space by the linear mapping W which is the only parameter we adjust. We generated 500 two dimensional data points, (x_1, x_2), from the function $x_2 = x_1 + 1.25\sin(x_1) + \mu$ where μ is noise from a uniform distribution in [0,1]. Final result from the IKoToM is shown in Figure 9 in which the projections of consecutive latent points are joined. We see that nearby latent points take responsibility for nearby data points.

4.3.2 Real data set

1- IRIS data set:
In this data set we have 150 samples with 4 dimensions and 3 types.
2- Algae data set:
In this data set we have 72 samples with 18 dimensions and 9 types.
3- Genes data set:
In this data set we have 40 samples with 3036 dimensions and 3 types.
4- Glass data set:
In this data set we have 214 samples with 10 dimensions and 6 types.

We show in Figures 10, 11, 12 and 13 the projections of the real data set

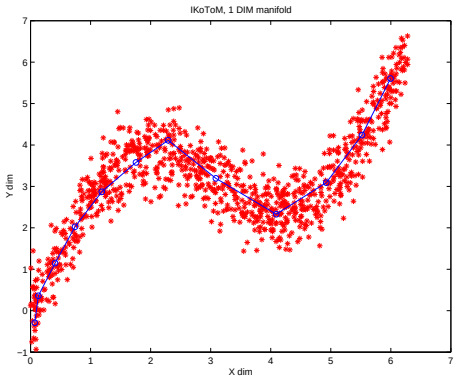

Figure 9: The resulting prototypes' positions after applying IKoToM. Prototypes are shown as blue 'o's.

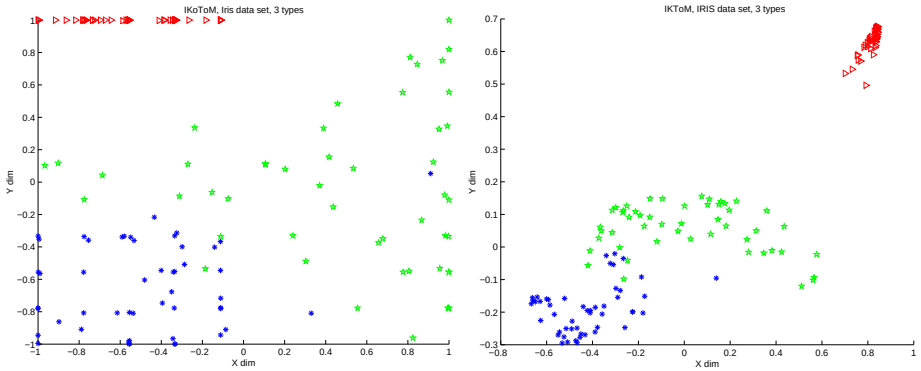

Figure 10: The result of applying IKoToM, left, and IKToM, right to the Iris real data set.

shown above onto a two dimensional grid of latent points using IKoToM, left, and IKToM, right. In each case, we see a projection of the classes into the latent space which gives a separation between the classes: most researchers will recognise the projection of the iris data set as familiar in that one class is readily separated while the other two are rather more difficult to separate; on the other data sets, we achieve results which are comparable with the best projections found by other methods we have used [6, 7, 1].

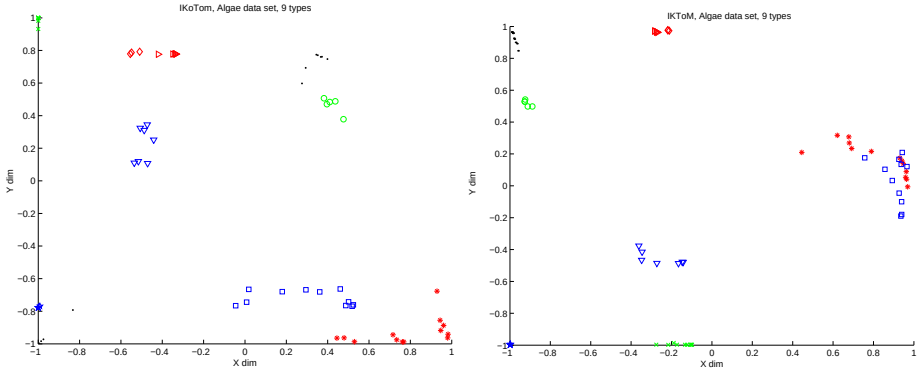

Figure 11: The result of applying IKoToM, left, and IKToM, right to the Algae real data set.

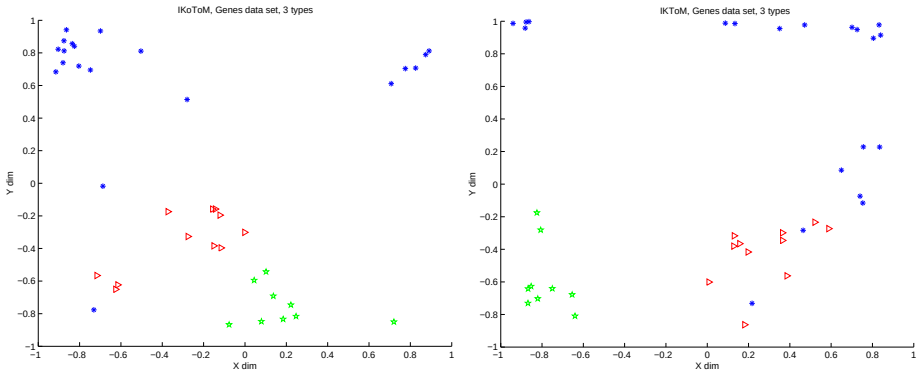

Figure 12: The result of applying IKoToM, left, and IKToM, right to the Genes real data set.

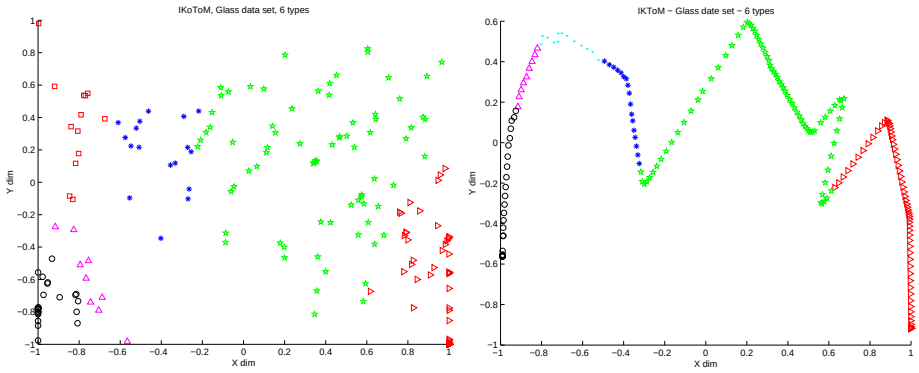

Figure 13: The result of applying IKoToM, left, and IKToM, right to the Glass real data set.

5 Conclusion

We have discussed one shortcoming of the K-means algorithm: its sensitivity to poor initialization which leads it to converge to a local rather than global optimum. We have shown how different performance functions lead to algorithms which incorporate the correct mixture of local and global knowledge to allow prototypes to optimally cluster a data set. We have derived both a batch algorithm and online algorithms from these performance functions.

We have extended these algorithms by using them with an underlying latent space which enables topology preserving mappings to be developed. We have illustrated these mappings (both batch and online versions) on a variety of data sets and shown how they may be used to visualise these data sets.

References

[1] W. Barbakh. The family of inverse exponential k-means algorithms. *Computing and Information Systems*, 11(1):1–10, February 2007. ISSN 1352-9404.

[2] W. Barbakh, M. Crowe, and C. Fyfe. A family of novel clustering algorithms. In *7th international conference on intelligent data engineering and automated learning, IDEAL2006*, pages 283–290, September 2006. ISSN 0302-9743 ISBN-13 978-3-540-45485-4.

[3] W. Barbakh and C. Fyfe. Performance functions and clustering algorithms. *Computing and Information Systems*, 10(2):2–8, May 2006. ISSN 1352-9404.

[4] C. M. Bishop, M. Svensen, and C. K. I. Williams. Gtm: The generative topographic mapping. *Neural Computation*, 1997.

[5] C. Fyfe. Two topographic maps for data visualization. *Data Mining and Knowledge Discovery*, 14:207–224, 2007. ISSN 1384-5810.

[6] C. Garcia-Osorio and C. Fyfe. The combined use of self-organising maps and andrews' curves. *International Journal of Neural Systems*, 2005.

[7] C. Garcia-Osorio and C. Fyfe. Visualisation of high dimensional data via orthogonal curves. *Journal of Universal Computer Science*, 11, 2005.

[8] Tuevo Kohonen. *Self-Organising Maps*. Springer, 1995.

[9] M. Peña and C. Fyfe. Model- and data-driven harmonic topographic maps. *WSEAS Transactions on Computers*, 4(9):1033–1044, 2005.

[10] B. Zhang. Generalized k-harmonic means – boosting in unsupervised learning. Technical report, HP Laboratories, Palo Alto, October 2000.

[11] B. Zhang, M. Hsu, and U. Dayal. K-harmonic means - a data clustering algorithm. Technical report, HP Laboratories, Palo Alto, October 1999.