The Inverse Exponential K-means Algorithms

Wesam Barbakh,
The University of Paisley,
Scotland.
email:wesam.barbakh@paisley.ac.uk

Abstract

We discuss one of the shortcomings of the standard K-means algorithm - its tendency to converge to a local rather than a global optimum. This is often accommodated by means of different random restarts of the algorithm, however in this paper, we attack the problem by amending the performance function of the algorithm in such a way as to incorporate global information into the performance function. We show on artificial data sets that the resulting algorithms are less initialisation-dependent than the standard K-means algorithm. We also show how to create a family of topology-preserving manifolds using these algorithms and an underlying constraint on the positioning of the prototypes.

1 Introduction

The K-means algorithm is a standard exploratory algorithm for identifying structure in a data set in an unsupervised manner. The algorithm allocates K prototypes throughout the data in order to minimise the mean square error between all data points and the nearest prototype to each data point. One of the major problems with K-means is that it is initialisation-dependent i.e. depending on the initial points selected for the prototypes, it may find a local optimum and not the global optimum. A variation on K-means is the so-called soft K-means [6] in which prototypes are allocated according to

$$\mathbf{m}_k = \frac{\sum_n r_{kn}\mathbf{x}_n}{\sum_{j,n} r_{jn}} \tag{1}$$

$$\text{where e.g. } r_{kn} = \frac{\exp(-\beta d(\mathbf{x}_n, \mathbf{m}_k))}{\sum_j \exp(-\beta d(\mathbf{x}_n, \mathbf{m}_j))} \tag{2}$$

and $d(a,b)$ is the Euclidean distance between a and b. Note that the standard K-means algorithm is a special case of the soft K-means algorithm in which the responsibilities, $r_{kn} = 1$ when $\mathbf{m}_k$ is the closest prototype to $\mathbf{x}_n$ and 0 otherwise. However the soft K-Means does increase the non-localness of the interaction since the responsibilities are typically never exactly equal to 0 for any data point-prototype combination. However there are still problems with Soft

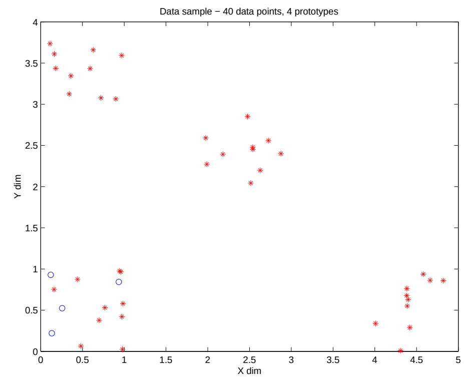

Figure 1: Data set is shown as 4 clusters of red '*'s, Initial positions of the prototypes are shown as blue 'o's.

K-Means: we illustrate by applying the soft K-means algorithm to the artificial data sample shown in Figure 1 in which we deliberately choose a poor initialisation. We find that with soft K-means it is important to choose a good value for β; if we choose a poor value we may have poor results in finding the clusters as shown in Figure 2. Even if we choose a good value, we will still find that soft K-means has the problem of sensitivity to the prototypes' initialization. As shown in Figure 3, while soft K-means succeeds in identifying the clusters, it also failed for the same data sample when we used a different initialization of the prototypes.

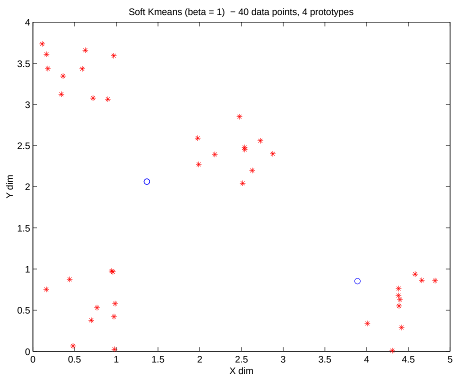

Figure 2: Due to poor initialisation, the soft K-means algorithm has failed to identify the four clusters.

2 New Algorithms

In this paper we introduce a new family of algorithms that solve the problem of sensitivity to initial conditions in soft K-means. The central idea in the new algorithms is that each prototype before moving to any new locations responds to 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.

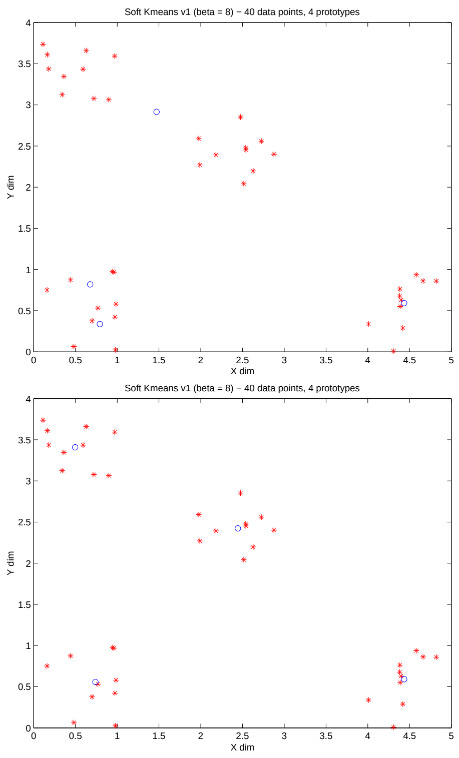

Figure 3: The vagaries of simulations: top, the soft k-means failed to identify the four clusters but the bottom figure is successful with the same data set.

2.1 Inverse Exponential K-means Algorithm 1 (IEK1)

The performance function for K-Means may be written as

$$J_K = \sum_{i=1}^{N} \min_{k=1}^{K} \parallel \mathbf{x}_i - \mathbf{m}_k \parallel^2 \qquad (3)$$

which we wish to minimise by moving the prototypes to the appropriate positions. Note that (3) 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 give minimum performance, which may result in dead prototypes, prototypes which are never appropriate for any cluster. Thus initializing prototypes appropriately can have a big effect in K-Means.

To solve this problem we provide a new performance function:

$$\begin{aligned} k* &= \arg\min_{k=1}^{K}(\parallel \mathbf{x}_i - \mathbf{m}_k \parallel) \\ J_J &= \sum_{i=1}^{N}\left[\sum_{j\neq k*}^{K}\frac{1}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel}\right] \\ &\quad * \left(1-\exp(-\parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^3)\right) \qquad (4) \end{aligned}$$

This performance function deals with the prototypes that are not detected by the minimum function and hence it solves the problem of sensitivity in K-means.

One of the advantages of this new algorithm is that we have two sets of update rules, as we will see in optimization and implementation section. This is beneficial when all prototypes are far from the data set: such a situation may very well happen when we have a high dimensional data set since in that situation most of the volume of the space lies in a thin shell far from the centre of the data so that even initialising prototypes to data points will not guarantee that the prototypes are close to many samples. We use (3) when $\mathbf{m}_j$ is the closest to $\mathbf{x}_i$ and use (4) for the other prototypes that are not the closest to $\mathbf{x}_i$.

The performance function (4) deals with all prototypes that not recognized by minimum function in (3). This function at minimum values tries to distribute the prototypes to fit the data and find the clusters. We need now to generate the new algorithm by optimizing the performance functions to give the minimum value and hence identifying the clusters.

2.1.1 Optimization and Implementation

To derive the new clustering algorithm, we need to find the partial derivative of (3) with respect to $\mathbf{m}_k$ and the partial derivative of (4) with respect to $\mathbf{m}_j$ where $\mathbf{m}_k$ represents the closest prototype to $\mathbf{x}_i$, and $\mathbf{m}_j$ represents the other prototypes.

$$\frac{\partial J_K}{\partial \mathbf{m}_k} = \sum_{i\in V_k} -2(\mathbf{x}_i - \mathbf{m}_k) \qquad (5)$$

where V_k contains the indices of data points that are closest to $\mathbf{m}_k$

$$\begin{aligned} \frac{\partial J_K}{\partial \mathbf{m}_k} &= 0 \\ \sum_{i\in V_k} -2(\mathbf{x}_i - \mathbf{m}_k) &= 0 \\ N_r\mathbf{m}_k &= \sum_{i\in V_k}\mathbf{x}_i \\ \mathbf{m}_k &= \frac{\sum_{i\in V_k}\mathbf{x}_i}{N_r} \\ \mathbf{m}_k &= \frac{\sum_{i\in V_k}\mathbf{x}_i a_{ik}}{\sum_{i\in V_k} a_{ik}} \end{aligned}$$

where N_r is the number of data points that are closest to $\mathbf{m}_k$, $a_{ik} = 1$

Notice: this is a part of how to calculate $\mathbf{m}_k$ from only the closest data points, but there is another calculation for $\mathbf{m}_k$ (by using the rest of data points) is provided from the second performance function as $\mathbf{m}_k$ might not be the closest to some data points $\mathbf{x}_i$, $i \in V_j$ where V_j is the index of data points that are not closest to $\mathbf{m}_k$, see (12), $\mathbf{m}_k$ should be calculated from all data points, not only the closest points as what happen in K-means algorithm.

The second performance function provides new calculations for the prototypes that are not closest to data points, and distributes them in a good way to identify the clusters.

$$\begin{aligned} \frac{\partial J_J}{\partial \mathbf{m}_j} &= \sum_{i\in V_d}\frac{(\mathbf{x}_i - \mathbf{m}_j)(1-\exp(-\parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^3)}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel^3} \\ &= \sum_{i\in V_d}\frac{(\mathbf{x}_i - \mathbf{m}_j)c_i}{\parallel \mathbf{x}_i - \mathbf{m}_j \parallel^3} \qquad (6) \end{aligned}$$

where V_d contains the indices of data points that are not closest to $\mathbf{m}_j$

$$c_i = 1 - \exp(-\parallel \mathbf{x}_i - \mathbf{m}_{k*} \parallel^3)$$

By assigning the partial derivative to zero and solving for $\mathbf{m}_j$ we have:

$$\begin{aligned}
\frac{\partial J_J}{\partial \mathbf{m}_j} &= 0 \\
\sum_{i \in V_d} \frac{(\mathbf{x}_i - \mathbf{m}_j) c_i}{\| \mathbf{x}_i - \mathbf{m}_j \|^3} &= 0 \\
\sum_{i \in V_d} \frac{\mathbf{x}_i c_i}{\| \mathbf{x}_i - \mathbf{m}_j \|^3} &= \sum_{i \in V_d} \frac{\mathbf{m}_j c_i}{\| \mathbf{x}_i - \mathbf{m}_j \|^3}
\end{aligned}$$

$$\begin{aligned}
\mathbf{m}_j(t+1) &= \frac{\sum_{i \in V_d} \frac{\mathbf{x}_i c_i}{\|\mathbf{x}_i - \mathbf{m}_j(t)\|^3}}{\sum_{i \in V_d} \frac{c_i}{\|\mathbf{x}_i - \mathbf{m}_j(t)\|^3}} \\
\mathbf{m}_j(t+1) &= \frac{\sum_{i \in V_d} \mathbf{x}_i b_{ij}}{\sum_{i \in V_d} b_{ij}} \qquad (7)
\end{aligned}$$

where,

$$\begin{aligned}
b_{ij} &= \frac{c_i}{\| \mathbf{x}_i - \mathbf{m}_j(t) \|^3} \\
&= \frac{1 - \exp(- \| \mathbf{x}_i - \mathbf{m}_{k*} \|^3)}{\| \mathbf{x}_i - \mathbf{m}_j(t) \|^3} \qquad (8)
\end{aligned}$$

The new locations for all prototypes can be calculated by:

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

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

$$\begin{aligned}
a_{ir} &= 1 \qquad (10) \\
b_{ir} &= \frac{1 - \exp(- \| \mathbf{x}_i - \mathbf{m}_{k*} \|^3}{\| \mathbf{x}_i - \mathbf{m}_r(t) \|^3} \qquad (11)
\end{aligned}$$

In execution we have found that the main computational cost is based on (11). Assume we have K prototypes and N data points, and then as every data point is closest to one and only one prototype, for one iteration we have N * K loops, and in each loop either (10) or (11) will be executed.
(10) will be executed L times, where L = N * 1
(11) will be executed H times, where H = N * (K-1)

Thus it is possible now to find a viable algorithm by using (11) for all prototypes (and thus never using (10) for the closest prototype)
However, (10) provides a strong advantage by introducing two sets of updates rather than a single update for all prototypes. Therefore, this allows the algorithm to separate the joint prototypes and hence distribute them to give good clustering. This is a symmetry-breaking factor which enables local minima to be avoided and allow the algorithm to be superior under some difficult conditions.
We assume every data point has only one minimum distance to the prototypes, in other words it is closest to one prototype only. We treat the other prototypes as distant prototypes even if they have the same minimum value. This step is optional, but it is very important to allow the algorithm to work very well in the case that all prototypes are initialized in the same location. This allows processing joint prototypes by two different equations and hence separates them.

2.2 Inverse Exponential K-means Algorithm 2 (IEK2)

This algorithm has the same idea like the previous one, but with some enhancement in implementation. The new locations for the prototypes can be calculated using the following equation:

$$\mathbf{m}_r(t+1) = \frac{\sum_{i \in V_r} \mathbf{x}_i a_{ir} + \sum_{i \in V_j, j \neq r} \mathbf{x}_i b_{ir}}{\sum_{i \in V_r} a_{ir} + \sum_{i \in V_j, j \neq r} b_{ir}} \qquad (12)$$

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

$$\begin{aligned}
a_{ir} &= \sum_k \frac{1}{\exp(-d(\mathbf{x}_i, \mathbf{m}_k)^\zeta)} \\
b_{ir} &= \frac{(1 - \exp(-(\min_{k=1}^{K}(d(\mathbf{x}_i, \mathbf{m}_k)))^3))^3}{(1 + \epsilon - \exp(-d(\mathbf{x}_i, \mathbf{m}_r(t))^3))^3}
\end{aligned}$$

where typically $\zeta = 0.01$ and $\epsilon = 0.001$. This algorithm provides a good way to distribute the prototypes to fit the manifold a little bit better than IEK1, see Figure (8).

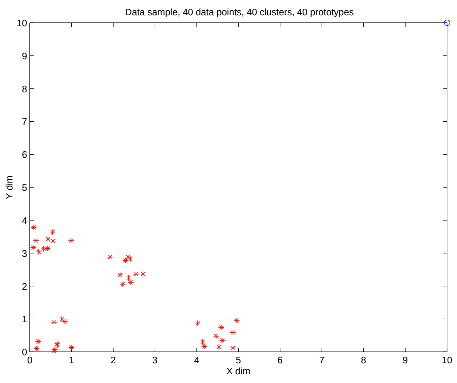

Figure 4: A poor initialisation of the prototypes' locations.

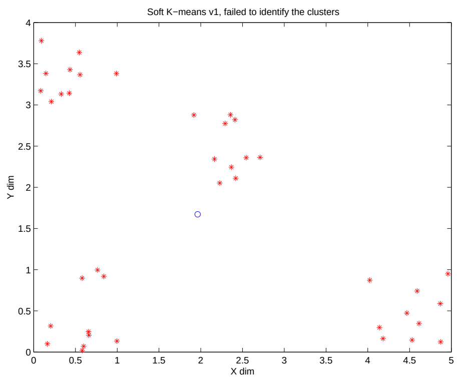

Figure 5: Results from the soft K-means algorithm; all the prototypes are in the centre of the data.

3 Simulations

We illustrate these algorithms with a few simulations on artificial two dimensional data sets, since the results are easily verified visually. Consider now the situation in which the prototypes are initialised very far from the data; this is an unlikely situation to happen in practice however a more likely situation is that all the prototypes are in fact initialised very far from a particular cluster. The question arises as to whether this cluster would be found by any algorithm. In Figure 4: we have 40 data points, each of which represents one cluster. All the prototypes are initialized in the same location and far from the clusters. Figure 5 shows the result after applying soft K-means algorithm. Figure 6 shows the result after applying IEK1 and algorithm IEK2: it is clear that under a very bad initialization the algorithms still work well and succeeds in identifying all the clusters.
Figure 8 shows a typical result after applying one of soft K-means, IEK1 and IEK2 to the noisy one dimensional manifold shown in Figure 7. From Figure 8 we can see the new algorithms distribute the prototypes in a good way to fit the manifold.

4 A Topology Preserving Mapping

In this part 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 Exponential K-means Topology-preserving Maping 1 (IEKToM1)

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) [5] and the Generative Topographic Mapping(GTM) [3] 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

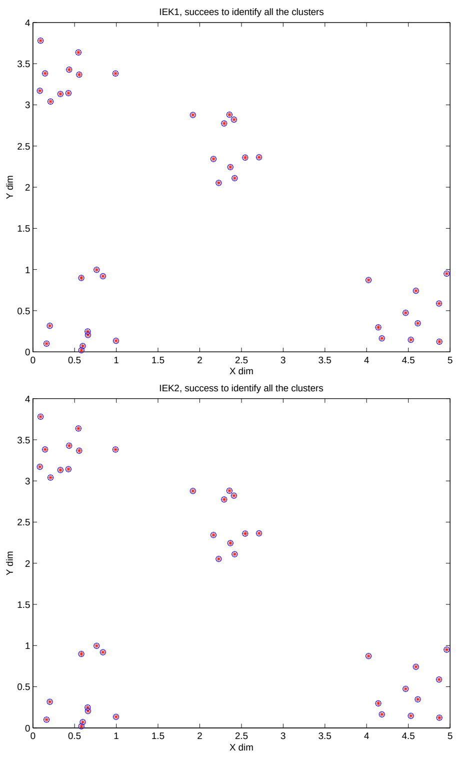

Figure 6: The resulting prototypes' positions after applying IEK1 and IEK2, respectively.

mappings [4] 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

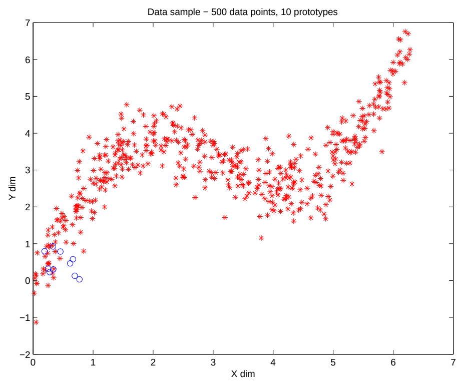

Figure 7: The two dimensional data lie on, or close to, a one dimensional manifold.

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 [4], we used a weighted mean squared error; with the Harmonic Topographic Mapping [7], we used Harmonic K-Means, with the Inverse-weighted K-means Topology-preserving Mapping (IKToM) [1, 2], we used Inverse Weighted K-means (IWK). We now apply the Inverse Exponential K-Means 1 (IEK1) algorithm 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

$$r_{ji} = \frac{\exp(-\gamma \parallel \mathbf{x}_i - \mathbf{w}_j \parallel^2)}{\sum_k \exp(-\gamma \parallel \mathbf{x}_i - \mathbf{w}_k \parallel^2)} \tag{13}$$

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

$$y_{ij} = \sum_k t_{kj} r_{ki} \tag{14}$$

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

4.2 Inverse Exponential K-means Topology-preserving Maping 2 (IEKToM2)

IEKToM2 algorithm like IEKToM1 has the same 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 IEKToM2 uses the well proved clustering abilities of the K-means algorithm, improved by using Inverse Exponential K-means 2 (IEK2) to make it insensitive to initialisation.

4.3 Simulation

4.3.1 Artificial data set

We create a simulation with 20 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 results from the IEKToM1 and IEKToM2 are shown in Figure 9.

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 Figure 10 the projections of the real data set shown above onto a two dimensional grid of latent points using IEKToM1. Figure 11 shows the same results with IEKToM2. In each case, we see a projection of the classes into the latent space which gives a separation between the classes which is comparable with the best projections found by other methods.

5 Conclusion

We have illustrated the problem of convergence to local optima in the original K-Means algorithm. We have shown that this may be overcome by incorporating some measure of global information in the algorithm and have illustrated the results on artificial data sets.

We have further used this algorithm with an underlying latent space and used the resulting mapping to visualise the underlying structure of the data set. We have illustrated the results of such simulations with some standard data sets which are well-known in the literature.

References

[1] 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*, 2006.

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

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

[4] C. Fyfe. Two topographic maps for data visualization. *Data Mining and Knowledge Discovery*, 2006.

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

[6] D. J. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press., 2003.

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

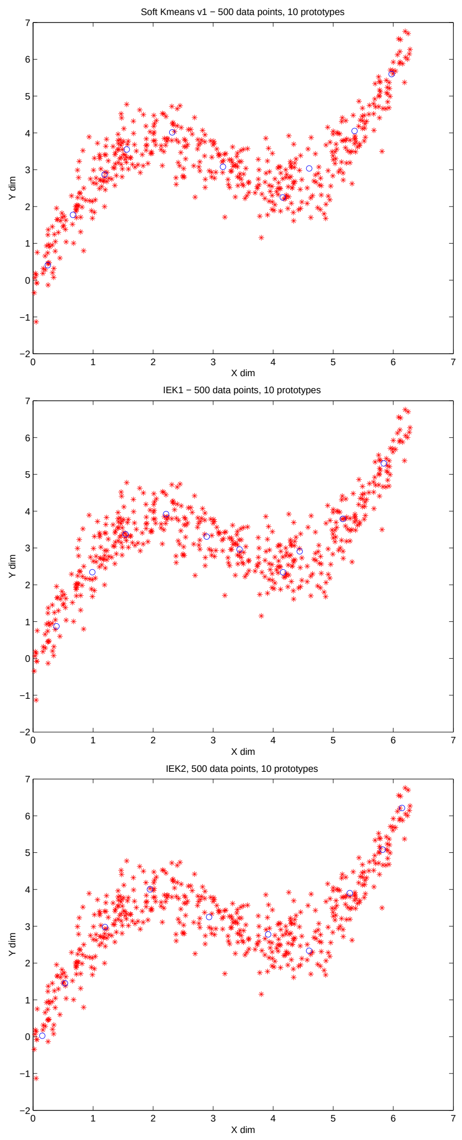

Figure 8: Top: the soft K-means results. Middle: IEK1. Bottom: IEK2.

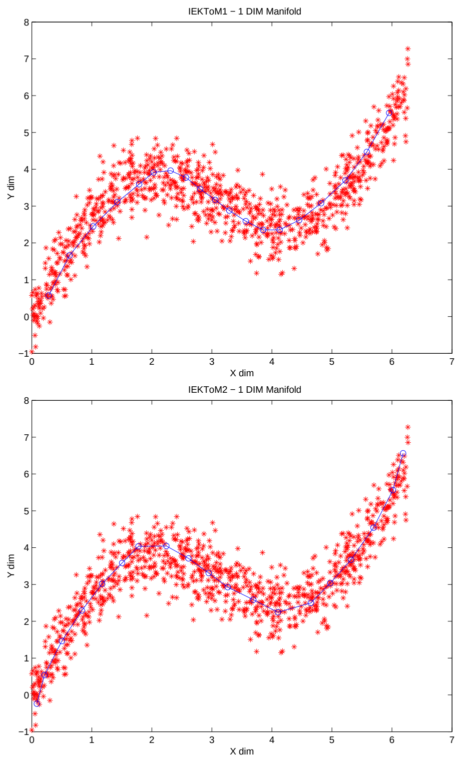

Figure 9: The resulting prototypes' positions after applying IEKToM1, top, and IEKToM2, bottom. prototypes are shown as blue 'o's

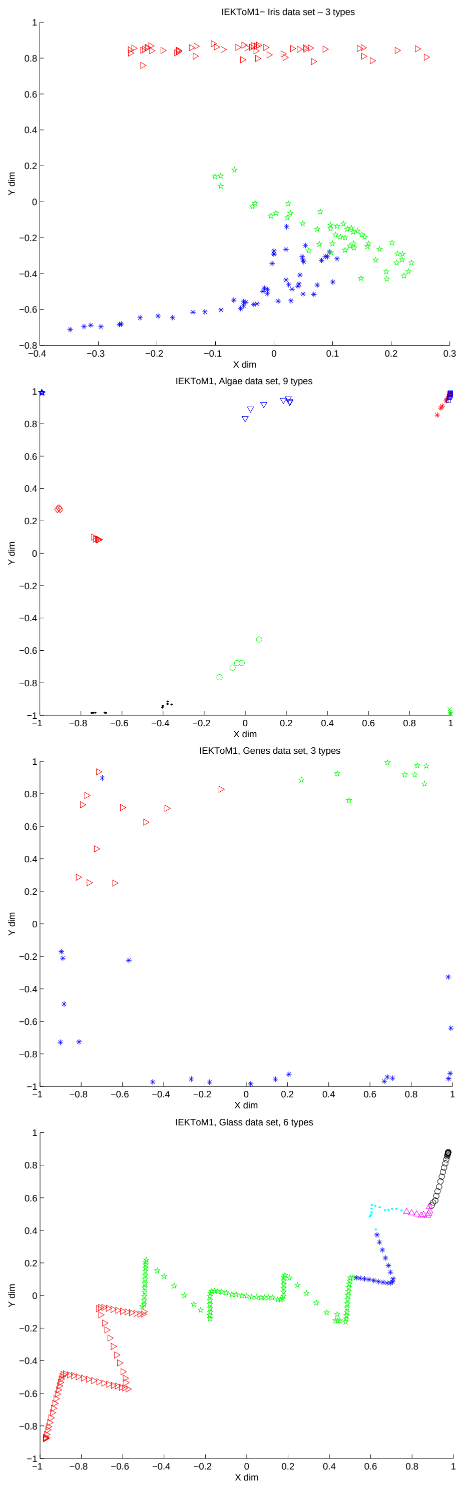

Figure 10: The results of using IEKToM1. The top line is the projection of the iris data set; the second line shows the algae data set; the third line shows the genes data set; the bottom line shows the glass data set.

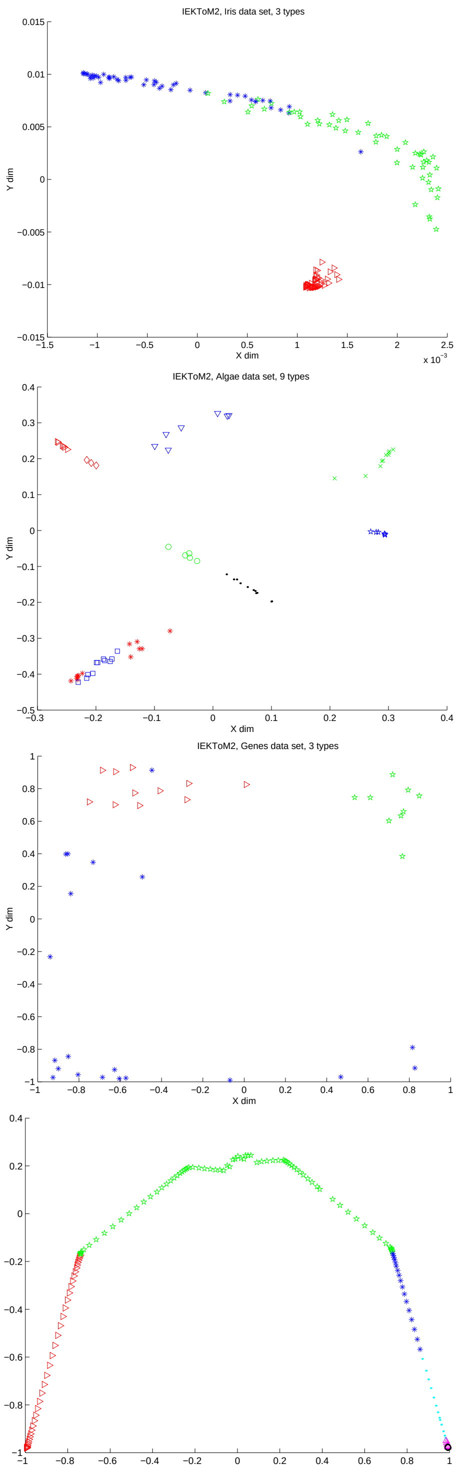

Figure 11: The results of using IEKToM2. The top line is the projection of the iris data set; the second line shows the algae data set; the third line shows the genes data set; the bottom line shows the glass data set.