6.2.3 Kernelised K-means

We can extend K-means using the kernel substitution trick that we introduced in Chapter 5. At an abstract level, the idea is the same: rather than making the algorithm more complex, we will transform the data into a space in which our simple algorithm works. We shall highlight this approach using the data shown in Figure 6.4(a).

We have seen that rather than actually performing the transformation of the data, kernel methods use kernel functions to directly compute inner (dot) products in the transformed space. As such, any algorithm where the data objects, $\mathbf{x}_1, \ldots, \mathbf{x}_N$, only appear as inner products ($\mathbf{x}_i^\mathsf{T}\mathbf{x}_j$ etc) can be given the kernel treatment, making it more powerful without any significant additional cost. Key to the operation of K-means is the computation of the distance between the nth object and the kth mean:

$$d_{nk} = (\mathbf{x}_n - \boldsymbol{\mu}_k)^\mathsf{T}(\mathbf{x}_n - \boldsymbol{\mu}_k),$$

where the mean, $\boldsymbol{\mu}_k$ is calculated according to Equation 6.1. Substituting this into the expression for d_{nk} gives:

$$d_{nk} = \left(\mathbf{x}_n - \frac{1}{N_k}\sum_{m=1}^{N} z_{mk}\mathbf{x}_m\right)^\mathsf{T}\left(\mathbf{x}_n - \frac{1}{N_k}\sum_{r=1}^{N} z_{rk}\mathbf{x}_r\right),$$

where $N_k = \sum_{n=1}^{N} z_{nk}$, the number of objects assigned to cluster k.

Multiplying out this expression results in the data ($\mathbf{x}_n$) only appearing in product terms:

$$d_{nk} \quad = \quad \mathbf{x}_n^\mathsf{T}\mathbf{x}_n - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk}\mathbf{x}_n^\mathsf{T}\mathbf{x}_m + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}\mathbf{x}_m^\mathsf{T}\mathbf{x}_r.$$

All that remains is to replace the inner products with kernel functions to give a kernelised distance:

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk}K(\mathbf{x}_n, \mathbf{x}_m) + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}K(\mathbf{x}_m, \mathbf{x}_r). \tag{6.3}$$

This distance is purely a function of the data and the current assignments, the cluster means do not appear. In fact, it is not, in general, possible to actually compute the cluster means in the transformed space. The original expression for the mean of cluster k is:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} z_{nk}\mathbf{x}_n}{\sum_{n=1}^{N} z_{nk}},$$

and the kernelised version is:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} z_{nk}\phi(\mathbf{x}_n)}{\sum_{n=1}^{N} z_{nk}}.$$

Within this expression, data objects appear on their own and not as inner products. In Chapter 5 we discussed how, for most kernel functions, we cannot compute

the transformation ($\mathbf{x}_n \rightarrow \phi(\mathbf{x}_n)$), only compute inner products in the transformed space ($\phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}_m)$). If we are unable to compute the transformation, we cannot compute $\boldsymbol{\mu}_k$.

Equation 6.3 suggests the following procedure for kernelised K-means:

1. Randomly initialise z_{nk} for each n (see below).
2. Compute $d_{n1}, \ldots, d_{nK}$ for each object using Equation 6.3.
3. Assign each object to the cluster with the lowest d_{nk}.
4. If assignments have changed, return to step 2, otherwise stop.

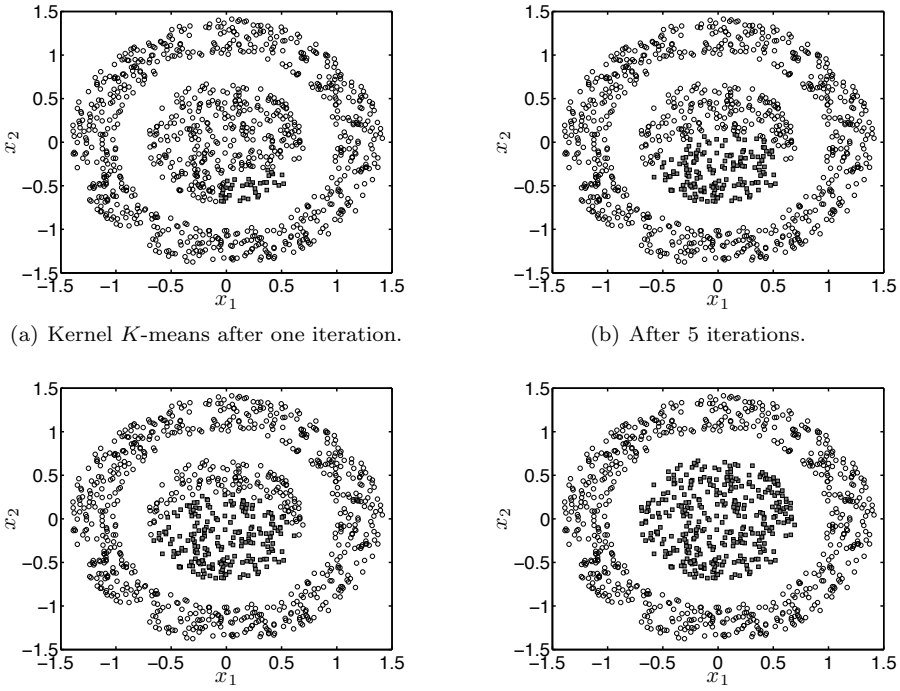
In standard K-means, we initialised the algorithm by randomly setting the means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$. In kernel K-means we do not have access to the means and we therefore initialise the algorithm via the object-cluster assignments, z_{nk}. We could do this completely randomly – for each n set one z_{nk} to 1 and all of the others ($z_{nl},\ l \neq k$) to zero but, given that we know K-means to be sensitive to initial conditions, it might be better to be more careful. Alternatively, we could run standard K-means and use the values of z_{nk} at convergence. This has the advantage that we can be sure that objects within the same cluster will be reasonably close to one another (something that we cannot guarantee if we set them randomly). A second alternative would be to assign $N - K + 1$ objects to cluster 1 and the remaining $K - 1$ objects to their own individual clusters. The performance of each iteration scheme will depend on the particular characteristics of the data being clustered.

Figure 6.5 shows the result of applying the kernel K-means algorithm to the data shown in Figure 6.4(a) (Matlab script: `kernelkmeans.m`). In this case, we have initialised by assigning all but one object to the 'circle' cluster and the remaining object to the 'square' cluster. A Gaussian kernel was used with $\gamma = 1$ (see Equation 5.19). Figure 6.5(a) shows the assignments one iteration after initialisation. As the algorithm progresses through 5, 10 and 30 iterations (Figures 6.5(b),6.5(c) and 6.5(d) respectively) the smaller cluster grows to take up the central circle. At convergence (Figure 6.5(d)), we can see that the algorithm has captured the interesting structure in the data.

Not only does kernel K-means allow us to find clusters that do not conform to our original idea of similarity, it also opens the door to performing analysis on other data types. We can cluster any type of data for which a kernel function exists and it is hard to find a data type for which there does not. Obvious examples are kernels for text (each object is a document) and kernels for graphs or networks. The latter is used widely in computational biology.

6.2.4 Summary

In the previous sections we introduced the K-means algorithm and showed how it could be kernelised. One of the great advantages of K-means is its simplicity – it is very easy to use and poses no great computational challenge. However, its simplicity is also a drawback: assuming that a cluster can be represented by a single point will often be too crude. In addition, there is no objective way to determine the



(a) Kernel K-means after one iteration.

(b) After 5 iterations.

(c) After 10 iterations.

(d) At convergence (30 iterations).

FIGURE 6.5 Result of applying kernelised K-means to the data shown in Figure 6.4(a).

number of clusters if our aim is just to cluster (remember that we mentioned how the number of clusters could be chosen as the one that gave best performance in some later task like classification). To overcome some of these drawbacks, we will now describe clustering with statistical mixture models. These models share some similarities with K-means but offer far richer representations of the data.

6.3 MIXTURE MODELS

In Figure 6.4(b) we showed a dataset for which the original K-means failed. The two clusters were stretched in such a way that some objects that should have belonged to one were in fact closer to the centre of the other. The problem our K-means algorithm had here was that its definition of a cluster was too crude. The characteristics of these stretched clusters cannot be represented by a single point and the squared distance. We need to be able to incorporate a notion of shape. Statistical mixture represent each cluster as a probability density. This generalisation leads to a powerful approach as we can model clusters with a wide variety of shapes in almost any type of data.