8

Ranking, clustering and data visualisation

In this chapter we conclude our presentation of kernel-based pattern analysis algorithms by discussing three further common tasks in data analysis: ranking, clustering and data visualisation.

Ranking is the problem of learning a ranking function from a training set of ranked data. The number of ranks need not be specified though typically the training data comes with a relative ordering specified by assignment to one of an ordered sequence of labels.

Clustering is perhaps the most important and widely used method of unsupervised learning: it is the problem of identifying groupings of similar points that are relatively 'isolated' from each other, or in other words to partition the data into dissimilar groups of similar items. The number of such clusters may not be specified a priori. As exact solutions are often computationally hard to find, effective approximations via relaxation procedures need to be sought.

Data visualisation is often overlooked in pattern analysis and machine learning textbooks, despite being very popular in the data mining literature. It is a crucial step in the process of data analysis, enabling an understanding of the relations that exist within the data by displaying them in such a way that the discovered patterns are emphasised. These methods will allow us to visualise the data in the kernel-defined feature space, something very valuable for the kernel selection process. Technically it reduces to finding low-dimensional embeddings of the data that approximately retain the relevant information.

8.1 Discovering rank relations

Ranking a set of objects is an important task in pattern analysis, where the relation sought between the datapoints is their relative rank. An example of an application would be the ranking of documents returned to the user in an information retrieval task, where it is hard to define a precise absolute relevance measure, but it is possible to sort by the user's preferences. Based on the query, and possibly some partial feedback from the user, the set of documents must be ordered according to their suitability as answers to the query.

Another example of ranking that uses different information is the task known as collaborative filtering. Collaborative filtering aims to rank items for a new user based only on rankings previously obtained from other users. The system must make recommendations to the new user based on information gleaned from earlier users. This problem can be cast in the framework of learning from examples if we treat each new user as a new learning task. We view each item as an example and the previous users' preferences as its features.

Example 8.1 If we take the example of a movie recommender system, a film is an example whose features are the gradings given by previous users. For users who have not rated a particular film the corresponding feature value can be set to zero, while positive ratings are indicated by a positive feature value and negative ratings by a negative value. Each new user corresponds to a new learning task. Based on a small set of supplied ratings we must learn to predict the ranking the user would give to films he or she has not yet seen. ■

In general we consider the following ranking task. Given a set of ranked examples, that is objects $\mathbf{x} \in X$ assigned to a label from an ordered set Y, we are required to predict the rank of new instances. Ranking could be tackled as a regression or classification problem by treating the ranks as real-values or the assignment to a particular rank value as a classification. The price of making these reductions is not to make full use of the available information in the reduction to classification or the flexibility inherent in the ordering requirement in the reduction to regression. It is therefore preferable to treat it as a problem in its own right and design specific algorithms able to take advantage of the specific nature of that problem.

Definition 8.2 [Ranking] A *ranking problem* is specified by a set

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$$

of instance/rank pairs. We assume an implicit kernel-defined feature space with corresponding feature mapping ϕ so that $\phi(\mathbf{x}_i)$ is in $\mathbb{R}^n$ for some n, $1 \leq n \leq \infty$. Furthermore, we assume its rank y_i is an element of a finite set Y with a total order relation. We say that $\mathbf{x}_i$ is preferred over $\mathbf{x}_j$ (or vice versa) if $y_i \succ y_j$ (or $y_i \prec y_j$). The objects $\mathbf{x}_i$ and $\mathbf{x}_j$ are not comparable if $y_i = y_j$. The induced relation on X is a partial ordering that partitions the input space into equivalence classes. A ranking rule is a mapping from instances to ranks $r : X \to Y$. ■

Remark 8.3 [An alternative reduction] One could also transform it into the problem of predicting the relative ordering of all possible pairs of examples, hence obtaining a 2-class classification problem. The problem in this approach would be the extra computational cost since the sample size for the algorithm would grow quadratically with the number of examples. If on the other hand the training data is given in the form of all relative orderings, we can generate a set of ranks as the equivalence classes of the equality relation with the induced ordering. ■

Definition 8.4 [Linear ranking rules] A *linear ranking rule* first embeds the input data into the real axis $\mathbb{R}$ by means of a linear function in the kernel-defined feature space $f(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$. The real-value is subsequently converted to a rank by means of $|Y|$ thresholds b_y, $y \in Y$ that respect the ordering of Y, meaning that $y \prec y'$ implies $b_y \leq b_{y'}$. We will denote by $\mathbf{b}$ the k-dimensional vector of thresholds. The ranking of an instance $\mathbf{x}$ is then given by

$$r_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \min \{ y \in Y : f(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_y \},$$

where we assume that the largest label has been assigned a sufficiently large value to ensure the minimum always exists. If $\mathbf{w}$ is given in a dual representation

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i),$$

the ranking function is

$$r_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \min \left\{ y \in Y : f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) < b_y \right\}.$$

■

A linear ranking rule partitions the input space into $|Y| + 1$ equivalence

classes corresponding to parallel bands defined by the direction $\mathbf{w}$ and the thresholds b_i as shown in the two upper diagrams of Figure 8.1. The lower diagrams give examples of nonlinear rankings arising from the use of appropriate kernel functions.

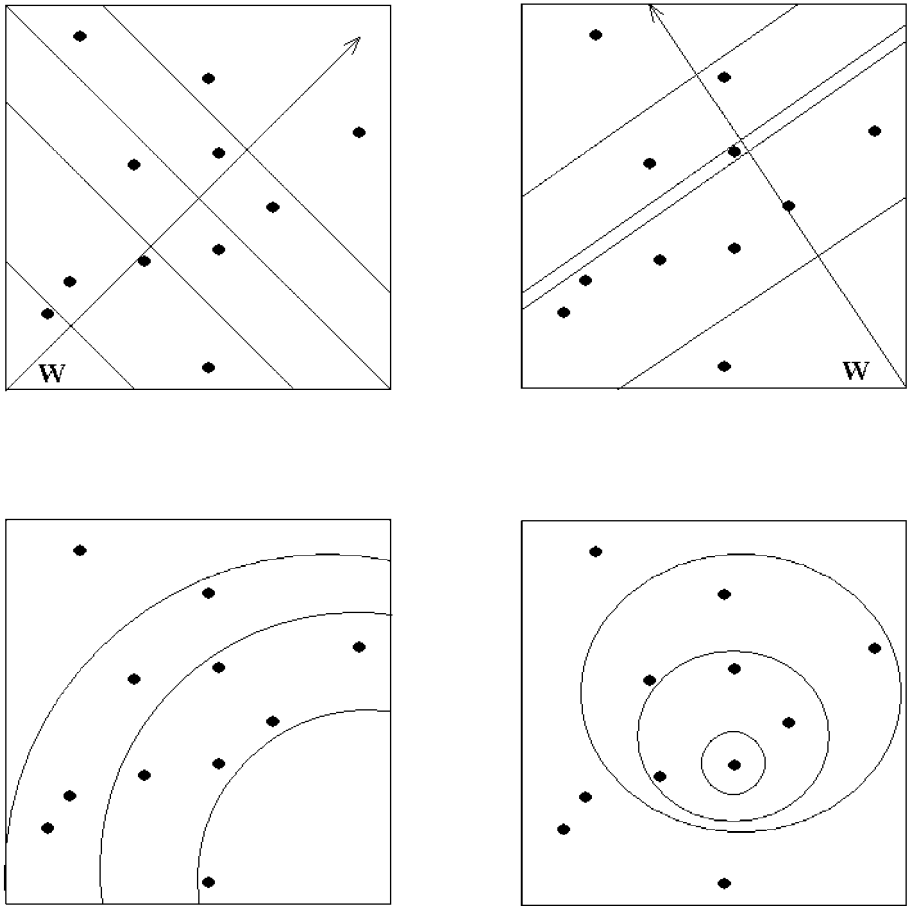

Fig. 8.1. Examples of the partitioning resulting from linear and nonlinear ranking functions.

Remark 8.5 [Degrees of freedom] The example of Figure 8.1 shows an important freedom available to ranking algorithms namely that the classes need not be equally spaced, we just need the ordering right. This is the key difference between the ranking functions we are considering and using regression on, for example, integer-ranking values. ■

Remark 8.6 [Ordering within ranks] The ranking functions described above have an additional feature in that the elements within each equivalence class can also be ordered by the value of the function $g(\mathbf{x})$, though we will ignore this information. This is a consequence of the fact that we represent the ranking by means of an embedding from X to the real line. ■

The algorithms we will discuss differ in the way $\mathbf{w}$ and $\mathbf{b}$ are chosen and as a result also differ in their statistical and computational properties. On the one hand, statistical considerations suggest that we seek stable functions whose testing performance matches the accuracy achieved on the training set. This will point for example to notions such as the margin while controlling the norm of $\mathbf{w}$. On the other hand, the computational cost of the algorithm should be kept as low as possible and so the size of the optimization problem should also be considered. Finally, we will want to use the algorithm in a kernel-defined feature space, so a dual formulation should always be possible.

8.1.1 Batch ranking

The starting point for deriving a batch-ranking algorithm will be consideration of statistical stability. Our strategy for deriving a stability bound will be to create an appropriate loss function that measures the performance of a choice of ranking function given by $\mathbf{w}$ and $\mathbf{b}$. For simplicity the measure of error will just be a count of the number of examples that are assigned the wrong rank, while the generalisation error will be the probability that a randomly drawn test example receives the wrong rank. For a further discussion of the loss function, see Remark 8.12.

Taking this view we must define a loss function that upper bounds the ranking loss, but that can be analysed using Theorem 4.9. We do this by defining two classification problems in augmented feature spaces such that getting both classifications right is equivalent to getting the rank right. We can think of one classification guaranteeing the rank is big enough, and the other that it is not too big.

Recoding the problem The key idea is to add one extra feature to the input vectors for each rank, setting their values to zero for all but the rank corresponding to the correct rank. The feature corresponding to the correct rank if available is set to 1. We use $\boldsymbol{\phi}$ to denote this augmented vector

$$\boldsymbol{\phi}(\mathbf{x}, y) = [\boldsymbol{\phi}(\mathbf{x}), 0, \ldots, 0, 1, 0, \ldots, 0] = [\boldsymbol{\phi}(\mathbf{x}), \mathbf{e}_y],$$

where we use $\mathbf{e}_y$ to denote the unit vector with yth coordinate equal to 1. We now augment the weight vector by a coordinate of $-b_y$ in the position of the feature corresponding to rank $y \in Y$

$$\hat{\mathbf{w}}_{\mathbf{b}} = \left[\mathbf{w}, -b_0, -b_1, -b_2, \ldots, -b_{|Y|}\right],$$

where for simplicity of notation we have assumed that $Y = \{1, \ldots, |Y|\}$ and have chosen b_0 to be some value smaller than $\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$ for all $\mathbf{w}$ and $\mathbf{x}$. Using this augmented representation we how have

$$\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y) \rangle = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle - b_y,$$

where y is the rank of $\mathbf{x}$. Now if $(\mathbf{w}, \mathbf{b})$ correctly ranks an example $(\mathbf{x}, y)$ then

$$\begin{aligned} y &= r_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \min\left\{y' \in Y : \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_{y'}\right\} \\ &= \min\left\{y' \in Y : \langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y) \rangle < b_{y'} - b_y\right\}, \end{aligned}$$

implying that

$$\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y) \rangle < 0. \tag{8.1}$$

Furthermore, we have

$$\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y-1) \rangle = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle - b_{y-1},$$

and so if $(\mathbf{w}, \mathbf{b})$ correctly ranks $(\mathbf{x}, y)$ then

$$\begin{aligned} y &= r_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \min\left\{y' \in Y : \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_{y'}\right\} \\ &= \min\left\{y' \in Y : \langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y-1) \rangle < b_{y'} - b_{y-1}\right\}, \end{aligned}$$

implying that

$$\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y-1) \rangle \geq 0. \tag{8.2}$$

Suppose that inequalities (8.1) and (8.2) hold for $(\mathbf{w}, \mathbf{b})$ on an example $(\mathbf{x}, y)$. Then since $\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y) \rangle < 0$, it follows that $\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_y$ and so

$$y \in \left\{y' \in Y : \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_{y'}\right\},$$

while $\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y-1) \rangle \geq 0$ implies $\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle \geq b_{y-1}$ hence

$$y - 1 \notin \left\{y' \in Y : \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_{y'}\right\},$$

giving

$$r_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \min\left\{y' \in Y : \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle < b_{y'}\right\} = y,$$

the correct rank. Hence we have shown the following proposition

Proposition 8.7 *The ranker* $r_{\mathbf{w},\mathbf{b}}(\cdot)$ *correctly ranks* $(\mathbf{x}, y)$ *if and only if* $\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y)\rangle < 0$ *and* $\langle \hat{\mathbf{w}}_{\mathbf{b}}, \boldsymbol{\phi}(\mathbf{x}, y-1)\rangle \geq 0$.

Hence the error rate of $r_{\mathbf{w},\mathbf{b}}(\mathbf{x})$ is bounded by the classifier rate on the extended set. The proposition therefore reduces the analysis of the ranker $r_{\mathbf{w},\mathbf{b}}(\mathbf{x})$ to that of a classifier in an augmented space.

Stability of ranking In order to analyse the statistical stability of ranking, we need to extend the data distribution $\mathcal{D}$ on $X \times Y$ to the augmented space. We simply divide the probability of example $(\mathbf{x}, y)$ equally between the two examples $(\boldsymbol{\phi}(\mathbf{x}, y), -1)$ and $(\boldsymbol{\phi}(\mathbf{x}, y-1), 1)$. We then apply Theorem 4.17 to upper bound the classifier error rate with probability $1-\delta$ by

$$\frac{1}{\ell\gamma}\sum_{i=1}^{\ell}\left(\xi_i^l + \xi_i^u\right) + \frac{4}{\ell\gamma}\sqrt{\mathrm{tr}(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)}{2\ell}},$$

where ξ_i^u, ξ_i^l are the slack variables measuring the amount by which the example $(\mathbf{x}_i, y_i)$ fails to meet the margin γ for the lower and upper thresholds. Hence, we can bound the error of the ranker by

$$P_{\mathcal{D}}\left(r_{\mathbf{w},\mathbf{b}}(\mathbf{x}) \neq y\right) \leq \frac{2}{\ell\gamma}\sum_{i=1}^{\ell}\left(\xi_i^l + \xi_i^u\right) + \frac{8}{\ell\gamma}\sqrt{\mathrm{tr}(\mathbf{K})} + 6\sqrt{\frac{\ln(2/\delta)}{2\ell}}, \quad (8.3)$$

where the factor 2 arises from the fact that either derived example being misclassified will result in a ranking error.

Ranking algorithms If we ignore the effects of the vector $\mathbf{b}$ on the norm of $\hat{\mathbf{w}}_{\mathbf{b}}$ we can optimise the bound by performing the following computation.

Computation 8.8 [Soft ranking] The soft ranking bound is optimised as follows

$$\begin{array}{ll} \min_{\mathbf{w},\mathbf{b},\gamma,\boldsymbol{\xi}^u,\boldsymbol{\xi}^l} & -\gamma + C\sum_{i=1}^{\ell}\left(\xi_i^u + \xi_i^l\right) \\ \text{subject to} & \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle \leq b_{y_i} - \gamma + \xi_i^l,\ y_i \neq |Y|,\ \xi_i^l \geq 0, \\ & \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle \geq b_{y_i-1} + \gamma - \xi_i^u,\ y_i \neq 1,\ \xi_i^u \geq 0, \\ & i = 1, \ldots, \ell, \text{ and } \|\mathbf{w}\|^2 = 1. \end{array} \quad (8.4)$$

■

Applying the usual technique of creating the Lagrangian and setting derivatives equal to zero gives the relationships

$$1 = \sum_{i=1}^{\ell}\left(\alpha_i^u + \alpha_i^l\right),$$

$$\begin{aligned}\mathbf{w} &= \frac{1}{2\lambda}\sum_{i=1}^{\ell}\left(\alpha_i^u - \alpha_i^l\right)\boldsymbol{\phi}\left(\mathbf{x}_i\right),\\ \sum_{i:y_i=y}\alpha_i^l &= \sum_{i:y_i=y-1}\alpha_i^u,\ y=2,\ldots,|Y|,\\ 0 &\leq \alpha_i^u,\alpha_i^l \leq C.\end{aligned}$$

Resubstituting into the Lagrangian results in the dual Lagrangian

$$L(\boldsymbol{\alpha}^u,\boldsymbol{\alpha}^l,\lambda) = -\frac{1}{4\lambda}\sum_{i,j=1}^{\ell}\left(\alpha_i^u-\alpha_i^l\right)\left(\alpha_j^u-\alpha_j^l\right)\kappa\left(\mathbf{x}_i,\mathbf{x}_j\right)-\lambda.$$

As in previous cases optimising for λ gives an objective that is the square root of the objective of the equivalent dual optimisation problem contained in the following algorithm.

Algorithm 8.9 [ν-ranking] The ν-ranking algorithm is implemented in Code Fragment 8.1. ■

Input	$S=\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_\ell,y_\ell)\}$, $\nu\in(0,1]$
$\max_{\boldsymbol{\alpha}^u,\boldsymbol{\alpha}^l}$	$W(\boldsymbol{\alpha}^u,\boldsymbol{\alpha}^l) = -\sum_{i,j=1}^{\ell}\left(\alpha_i^u-\alpha_i^l\right)\left(\alpha_j^u-\alpha_j^l\right)\kappa(\mathbf{x}_i,\mathbf{x}_j)$,
subject to	$\sum_{i:y_i=y}\alpha_i^l = \sum_{i:y_i=y-1}\alpha_i^u$, $y=2,\ldots,\lvert Y\rvert$,
	$0\leq\alpha_i^u,\alpha_i^l\leq 1/\left(\nu\ell\right)$, $i=1,\ldots,\ell$, $\sum_{i=1}^{\ell}\left(\alpha_i^u+\alpha_i^l\right)=1$
compute	$\alpha_i=\alpha_i^{u*}-\alpha_i^{l*}$
	$f(\mathbf{x})=\sum_{i=1}^{\ell}\alpha_i\kappa(\mathbf{x}_i,\mathbf{x})$
	$\mathbf{b}=\left(b_1,\ldots,b_{\lvert Y\rvert-1},\infty\right)$
where	$b_y=0.5\left(f\left(\mathbf{x}_i\right)+f\left(\mathbf{x}_j\right)\right)$
	$\gamma=0.5\left(f\left(\mathbf{x}_j\right)-f\left(\mathbf{x}_i\right)\right)$
where	$(\mathbf{x}_i,y)$, $(\mathbf{x}_j,y+1)$ satisfy $0<\alpha_i^{l*}<1/\left(\nu\ell\right)$
	and $0<\alpha_j^{u*}<1/\left(\nu\ell\right)$,
output	$r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$, γ

Code Fragment 8.1. Pseudocode for the soft ranking algorithm.

The next theorem characterises the output of Algorithm 8.9.

Theorem 8.10 *Fix $\nu\in(0,1]$. Suppose that a training sample*

$$S=\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_\ell,y_\ell)\}$$

drawn according to a distribution $\mathcal{D}$ over $X\times Y$, where $Y=\{1,\ldots,|Y|\}$ is a finite set of ranks and suppose $r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$, γ is the output of Algorithm

8.9, then $r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$ *optimises the bound of (8.3). Furthermore, there are at most* $\nu\ell$ *training points that fail to achieve a margin* γ *from both adjacent thresholds and hence have non-zero slack variables, while at least* $\nu\ell$ *of the training points have margin at least* γ*.*

Proof By the derivation given above setting $C = 1/(\nu\ell)$, the solution vector is a rescaled version of the solution of the optimisation problem (8.4). The setting of the values b_y follows from the Karush–Kuhn–Tucker conditions that ensure $\xi_i^{l*} = 0$ and the appropriate rescaling of $f(\mathbf{x}_i)$ is γ^* from the upper boundary if $0 < \alpha_i^{l*} < C$, with the corresponding result when $0 < \alpha_j^{u*} < C$. The bounds on the number of training points achieving the margin follow from the bounds on α_i^u and α_i^l. □

Remark 8.11 [Measuring stability] We have omitted an explicit generalisation bound from the proposition to avoid the message getting lost in technical details. The bound could be computed by ignoring $b_{|Y|}$ and b_0 and removing one of the derived examples for points with rank 1 or $|Y|$ and hence computing the margin and slack variables for the normalised weight vector. These could then be plugged into (8.3). ■

Remark 8.12 [On the loss function] We have measured loss by counting the number of wrong ranks, but the actual slack variables get larger the further the distance to the correct rank. Intuitively, it does seem reasonable to count a bigger loss if the rank is out by a greater amount. Defining a loss that takes the degree of mismatch into account and deriving a corresponding convex relaxation is beyond the scope of this book. ■

This example again shows the power and flexibility of the overall approach we are advocating. The loss function that characterises the performance of the task under consideration is upper bounded by a loss function to which the Rademacher techniques can be applied. This in turn leads to a uniform bound on the performance of the possible functions on randomly generated test data. By designing an algorithm to optimise the bound we therefore directly control the stability of the resulting pattern function. A careful choice of the loss function ensures that the optimisation problem is convex and hence has a unique optimum that can be found efficiently using standard optimisation algorithms.

8.1.2 On-line ranking

With the exception of Section 7.4 all of the algorithms that we have so far considered for classification, regression, novelty-detection and, in the current subsection, for ranking all assume that we are given a set of training examples that can be used to drive the learning algorithm towards a good solution. Unfortunately, training sets are not always available before we start to learn.

Example 8.13 A case in point is Example 8.1 given above describing the use of collaborative filtering to recommend a film. Here we start with no information about the new user. As we obtain his or her views of a few films we must already begin to learn and hence direct our recommendations towards films that are likely to be of interest. ■

The learning paradigm that considers examples being presented one at a time with the system being allowed to update the inferred pattern function after each presentation is known as *on-line learning*.

Perhaps the best known on-line learning algorithm is the *perceptron algorithm* given in Algorithm 7.52. We now describe an on-line ranking algorithm that follows the spirit of the perceptron algorithm. Hence, it considers one example at a time ranking it using its current estimate of the weight vector $\mathbf{w}$ and ranking thresholds $\mathbf{b}$. We again assume that the weight vector is expressed in the dual representation

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i),$$

where now the value of α_i can be positive or negative. The α_i are initialised to 0. The vector $\mathbf{b}$ must be initialised to an ordered set of integer values, which can, for example, be taken to be all 0, except for $b_{|Y|}$, which is set to ∞ and remains fixed throughout.

If an example is correctly ranked then no change is made to the current ranking function $r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$. If on the other hand the estimated rank is wrong for an example $(\mathbf{x}_i, y_i)$, an update is made to the dual variable α_i as well as to one or more of the rank thresholds in the vector $\mathbf{b}$.

Suppose that the estimated rank $y < y_i$. In this case we decrement thresholds $b_{y'}$ for $y' = y, \ldots, y_i - 1$ by 1 and increment α_i by $y_i - y$. When $y > y_i$ we do the reverse by incrementing the thresholds $b_{y'}$ for $y' = y_i, \ldots, y - 1$ by 1 and decrementing α_i by $y - y_i$. This is given in the following algorithm.

Algorithm 8.14 [On-line ranking] The on-line ranking algorithm is implemented in Code Fragment 8.2. ■

Input	training sequence $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \ldots$
Process	$\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{b} = \mathbf{0}$, $b_{\|Y\|} = \infty$, $i = 0$
2	repeat
3	$i = i + 1$
4	$y = r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x}_i)$
3	if $y < y_i$
4	$\alpha_i = \alpha_i + y_i - y$
5	$y' = y' - 1$ for $y' = y, \ldots, y_i - 1$
6	else if $y > y_i$
7	$\alpha_i = \alpha_i + y_i - y$
8	$y' = y' + 1$ for $y' = y_i, \ldots, y - 1$
9	end
10	until finished
Output	$r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$

Code Fragment 8.2. Pseudocode for on-line ranking.

In order to verify the correctness of Algorithm 8.14 we must check that the update rule preserves a valid ranking function or in other words that the vector of thresholds remains correctly ordered

$$y < y' \Longrightarrow b_y \leq b_{y'}.$$

In view of the initialisation of $\mathbf{b}$ to integer values and the integral updates, the property could only become violated in one of two cases. The first is if $b_y = b_{y+1}$ and we increment b_y by 1, while leaving b_{y+1} fixed. It is clear from the update rule above that this could only occur if the estimated rank was $y+1$, a rank that cannot be returned when $b_y = b_{y+1}$. A similar contradiction shows that the other possible violation of decrementing b_{y+1} when $b_y = b_{y+1}$ is also ruled out. Hence, the update rule does indeed preserve the ordering of the vector of thresholds.

Stability analysis of on-line ranking We will give an analysis of the stability of Algorithm 8.14 based on the bound given in Theorem 7.54 for the perceptron algorithm. Here, the bound is in terms of the number of updates made to the hypothesis. Since the proof is identical to that of Theorem 7.54, we do not repeat it here.

Theorem 8.15 *Fix $\delta > 0$. If the ranking perceptron algorithm makes $1 \leq k \leq \ell/2$ updates before converging to a hypothesis $r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$ that correctly*

ranks a training set

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$$

drawn independently at random according to a distribution $\mathcal{D}$*, then with probability at least* $1 - \delta$ *over the draw of the set* S*, the generalisation error of* $r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$ *is bounded by*

$$P_{\mathcal{D}}\left(r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x}) \neq y\right) \leq \frac{1}{\ell - k}\left(k \ln \ell + \ln \frac{\ell}{2\delta}\right). \tag{8.5}$$

Thus, a bound on the number of updates of the perceptron-ranking algorithm can be translated into a generalisation bound of the resulting classifier if it has been run until correct ranking of the (batch) training set has been achieved. For practical purposes this gives a good indication of how well the resulting ranker will perform since we can observe the number of updates made and plug the number into the bound (7.25). From a theoretical point of view one would like to have some understanding of when the number of updates can be expected to be small for the chosen algorithm.

We now give an a priori bound on the number of updates of the perceptron-ranking algorithm by showing that it can be viewed as the application of the perceptron algorithm for a derived classification problem and then applying Novikoff's Theorem 7.53. The weight vector $\mathbf{w}^*$ will be the vector solving the maximal margin problem for the derived training set

$$\hat{S} = \{(\boldsymbol{\phi}(\mathbf{x}, y), -1), (\boldsymbol{\phi}(\mathbf{x}, y-1), 1) : (\mathbf{x}, y) \in S\}$$

for a ranking training set S. The updates of the perceptron-ranking algorithm correspond to slightly more complex examples

$$\boldsymbol{\phi}\left(\mathbf{x}, y : y'\right) = \sum_{u=y}^{y'-1} \boldsymbol{\phi}(\mathbf{x}, u).$$

When the estimated rank $y < y_i$ the example $(\boldsymbol{\phi}(\mathbf{x}, y : y_i), 1)$ is misclassified and updating on this example is equivalent to the perceptron-ranking algorithm update. Similarly, when the estimated rank $y > y_i$ the example $(\boldsymbol{\phi}(\mathbf{x}, y_i : y), -1)$ is misclassified and the updates again correspond. Hence, since

$$\left\|\boldsymbol{\phi}\left(\mathbf{x}, y : y'\right)\right\|^2 \leq (|Y| - 1)\left(\|\boldsymbol{\phi}(\mathbf{x})\|^2 + 1\right),$$

we can apply Theorem 7.53 to bound the number of updates by

$$\frac{(|Y| - 1)\left(R^2 + 1\right)}{\gamma^2},$$

where R is a bound on the norm of the feature vectors $\boldsymbol{\phi}(\mathbf{x})$ and γ is the margin obtained by the corresponding hard margin batch algorithm. This gives the following corollary.

Corollary 8.16 *Fix $\delta > 0$. Suppose the batch ranking algorithm with $\nu = 1/\ell$ has margin γ on the training set*

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$$

drawn independently at random according to a distribution $\mathcal{D}$ and contained in a ball of radius R about the origin. Then with probability at least $1 - \delta$ over the draw of the set S, the generalisation error of the ranking function $r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x})$ obtained by running the on-line ranking algorithm on S in batch mode is bounded by

$$P_{\mathcal{D}}\left(r_{\boldsymbol{\alpha},\mathbf{b}}(\mathbf{x}) \neq y\right) \leq \frac{2}{\ell}\left(\frac{(|Y|-1)\left(R^2+1\right)}{\gamma^2}\ln\ell + \ln\frac{\ell}{2\delta}\right),$$

provided

$$\frac{(|Y|-1)\left(R^2+1\right)}{\gamma^2} \leq \frac{\ell}{2}.$$

8.2 Discovering cluster structure in a feature space

Cluster analysis aims to discover the internal organisation of a dataset by finding structure within the data in the form of 'clusters'. This generic word indicates separated groups of similar data items. Intuitively, the division into clusters should be characterised by within-cluster similarity and between-cluster (external) dissimilarity. Hence, the data is broken down into a number of groups composed of similar objects with different groups containing distinctive elements. This methodology is widely used both in multivariate statistical analysis and in machine learning.

Clustering data is useful for a number of different reasons. Firstly, it can aid our understanding of the data by breaking it into subsets that are significantly more uniform than the overall dataset. This could assist for example in understanding consumers by identifying different 'types' of behaviour that can be regarded as prototypes, perhaps forming the basis for targeted marketing exercises. It might also form the initial phase of a more complex data analysis. For example, rather than apply a classification algorithm to the full dataset, we could use a separate application for each cluster with the intention of rendering the local problem within a single cluster easier to solve accurately. In general we can view the clustering as making the data

simpler to describe, since a new data item can be specified by indicating its cluster and then its relation to the cluster centre.

Each application might suggest its own criterion for assessing the quality of the clustering obtained. Typically we would expect the quality to involve some measure of fit between a data item and the cluster to which it is assigned. This can be viewed as the pattern function of the cluster analysis. Hence, a stable clustering algorithm will give assurances about the expected value of this fit for a new randomly drawn example. As with other pattern analysis algorithms this will imply that the pattern of clusters identified in the training set is not a chance occurrence, but characterises some underlying property of the distribution generating the data.

Perhaps the most common choice for the measure assumes that each cluster has a centre and assesses the fit of a point by its squared distance from the centre of the cluster to which it is assigned. Clearly, this will be minimised if new points are assigned to the cluster whose centre is nearest. Such a division of the space creates what is known as a *Voronoi diagram* of regions each containing one of the cluster centres. The boundaries between the regions are composed of intersecting hyperplanes each defined as the set of points equidistant from some pair of cluster centres.

Throughout this section we will adopt the squared distance criterion for assessing the quality of clustering, initially based on distances in the input space, but subsequently generalised to distances in a kernel-defined feature space. In many ways the use of kernel methods for clustering is very natural, since the kernel-defines pairwise similarities between data items, hence providing all the information needed to assess the quality of a clustering. Furthermore, using kernels ensures that the algorithms can be developed in full generality without specifying the particular similarity measure being used.

Ideally, all possible arrangements of the data into clusters should be tested and the best one selected. This procedure is computationally infeasible in all but very simple examples since the number of all possible partitions of a dataset grows exponentially with the number of data items. Hence, efficient algorithms need to be sought. We will present a series of algorithms that make use of the distance in a kernel-defined space as a measure of dissimilarity and use simple criteria of performance that can be used to drive practical, efficient algorithms that approximate the optimal solution.

We will start with a series of general definitions that are common to all approaches, before specifying the problem as a (non-convex) optimisation problem. We will then present a greedy algorithm to find sub-optimal solu-

tions (local minima) and a spectral algorithm that can be solved globally at the expense of relaxing the optimisation criterion.

8.2.1 Measuring cluster quality

Given an unlabelled set of data

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\},$$

we wish to find an assignment of each point to one of a finite – but not necessarily prespecified – number N of classes. In other words, we seek a map

$$f : S \to \{1, 2, \ldots, N\}.$$

This partition of the data should be chosen among all possible assignments in such a way as to solve the measure of clustering quality given in the following computation.

Computation 8.17 [Cluster quality] The clustering function should be chosen to optimise

$$f = \operatorname*{argmin}_f \sum_{i,j:f_i=f(\mathbf{x}_i)=f(\mathbf{x}_j)=f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2, \tag{8.6}$$

where we have as usual assumed a projection function $\boldsymbol{\phi}$ into a feature space F, in which the kernel κ computes the inner product

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle.$$

■

We will use the short notation $f_i = f(\mathbf{x}_i)$ throughout this section. Figure 8.2 shows an example of a clustering of a set of data into two clusters with an indication of the contributions to (8.6). As indicated above this is not the most general clustering criterion that could be considered, but we begin by showing that it does have a number of useful properties and does subsume some apparently more general criteria. A first criticism of the criterion is that it does not seem to take into account the between-cluster separation, but only the within-cluster similarity. We might want to consider a criterion that balanced both of these two factors

$$\min_f \left\{ \sum_{i,j:f_i=f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 - \lambda \sum_{i,j:f_i\neq f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 \right\}. \tag{8.7}$$

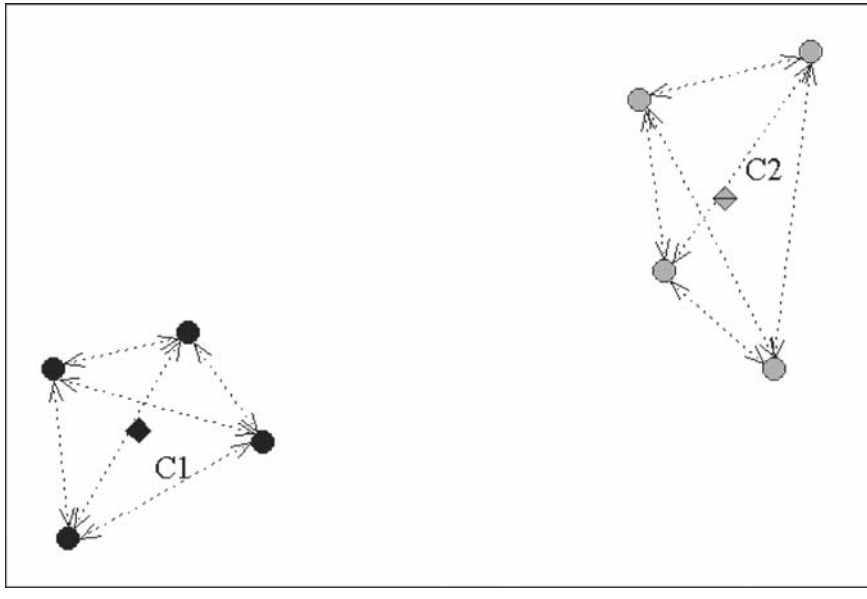

Fig. 8.2. An example of a clustering of a set of data.

However, observe that we can write

$$\begin{aligned}\sum_{i,j:f_i \neq f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 &= \sum_{i,j=1}^{\ell} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 \\ &\quad - \sum_{i,j:f_i = f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 \\ &= A - \sum_{i,j:f_i = f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2,\end{aligned}$$

where A is constant for a given dataset. Hence, equation (8.7) can be expressed as

$$\begin{aligned}&\min_f \left\{ \sum_{i,j:f_i = f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 - \lambda \sum_{i,j:f_i \neq f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 \right\} \\ = \quad &\min_f \left\{ (1+\lambda) \sum_{i,j:f_i = f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 - \lambda A \right\},\end{aligned}$$

showing that the same clustering function f solves the two optimisations (8.6) and (8.7). These derivations show that minimising the within-cluster

distances for a fixed number of clusters automatically maximises the between-cluster distances.

There is another nice property of the solution of the optimisation criterion (8.6). If we simply expand the expression, we obtain

$$\begin{aligned}
\text{opt} &= \sum_{i,j:f_i=f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 \\
&= \sum_{k=1}^{N} \sum_{i:f_i=k} \sum_{j:f_j=k} \langle \boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j), \boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j) \rangle \\
&= \sum_{k=1}^{N} 2 \left(\left| f^{-1}(k) \right| \sum_{i:f_i=k} \kappa(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i:f_i=k} \sum_{j:f_j=k} \kappa(\mathbf{x}_i, \mathbf{x}_j) \right) \\
&= \sum_{k=1}^{N} 2 \left| f^{-1}(k) \right| \sum_{i:f_i=k} \|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_k\|^2,
\end{aligned}$$

where the last line follows from (5.4) of Chapter 5 expressing the average-squared distance of a set of points from their centre of mass, and

$$\mu_k = \frac{1}{\left| f^{-1}(k) \right|} \sum_{i \in f^{-1}(k)} \boldsymbol{\phi}(\mathbf{x}_i) \tag{8.8}$$

is the centre of mass of those examples assigned to cluster k, a point often referred to as the *centroid* of the cluster. This implies that the optimisation criterion (8.6) is therefore also equivalent to the criterion

$$f = \operatorname*{argmin}_f \sum_{k=1}^{N} \left(\sum_{i:f_i=k} \|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_k\|^2 \right) = \operatorname*{argmin}_f \sum_{i=1}^{\ell} \left\| \boldsymbol{\phi}(\mathbf{x}_i) - \mu_{f(\mathbf{x}_i)} \right\|^2, \tag{8.9}$$

that seeks a clustering of points minimising the sum-squared distances to the centres of mass of the clusters. One might be tempted to assume that this implies the points are assigned to the cluster whose centroid is nearest. The following theorem shows that indeed this is the case.

Theorem 8.18 *The solution of the clustering optimisation criterion*

$$f = \operatorname*{argmin}_f \sum_{i,j:f_i=f_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2$$

of Computation 8.17 can be found in the form

$$f(\mathbf{x}_i) = \operatorname*{argmin}_{1 \le k \le N} \|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_k\|,$$

where μ_j is the centroid of the points assigned to cluster j.

Proof Let μ_k be as in equation (8.8). If we consider a clustering function g defined on S that assigns points to the nearest centroid

$$g(\mathbf{x}_i) = \operatorname*{argmin}_{1\leq k\leq N} \|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_k\|,$$

we have, by the definition of g

$$\sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_{g(\mathbf{x}_i)}\right\|^2 \leq \sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_{f(\mathbf{x}_i)}\right\|^2. \tag{8.10}$$

Furthermore, if we let

$$\hat{\mu}_k = \frac{1}{|g^{-1}(k)|} \sum_{i\in g^{-1}(k)} \boldsymbol{\phi}(\mathbf{x}_i)$$

it follows that

$$\sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - \hat{\mu}_{g(\mathbf{x}_i)}\right\|^2 \leq \sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_{g(\mathbf{x}_i)}\right\|^2 \tag{8.11}$$

by Proposition 5.2. But the left-hand side is the value of the optimisation criterion (8.9) for the function g. Since f was assumed to be optimal we must have

$$\sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - \hat{\mu}_{g(\mathbf{x}_i)}\right\|^2 \geq \sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - \mu_{f(\mathbf{x}_i)}\right\|^2,$$

implying with (8.10) and (8.11) that the two are in fact equal. The result follows. □

The characterisation given in Proposition 8.18 also indicates how new data should be assigned to the clusters. We simply use the natural generalisation of the assignment as

$$f(\mathbf{x}) = \operatorname*{argmin}_{1\leq k\leq N} \|\boldsymbol{\phi}(\mathbf{x}) - \mu_k\|.$$

Once we have chosen the cost function of Computation 8.17 and observed that its test performance is bound solely in terms of the number of centres and the value of equation (8.6) on the training examples, it is clear that any clustering algorithm must attempt to minimise the cost function. Typically we might expect to do this for different numbers of centres, finally selecting the number for which the bound on $\mathbb{E}_{\mathcal{D}} \min_{1\leq k\leq N} \|\boldsymbol{\phi}(\mathbf{x}) - \mu_k\|^2$ is minimal.

Hence, the core task is given a fixed number of centres N find the partition into clusters which minimises equation (8.6). In view of Proposition 8.18, we therefore arrive at the following clustering optimisation strategy.

Computation 8.19 [Clustering optimisation strategy] The clustering optimisation strategy is given by

input	$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$, integer N
process	$\boldsymbol{\mu} = \text{argmin}_{\boldsymbol{\mu}} \sum_{i=1}^{\ell} \min_{1 \leq k \leq N} \left\| \boldsymbol{\phi}(\mathbf{x}_i) - \mu_k \right\|^2$
output	$f(\cdot) = \text{argmin}_{1 \leq k \leq N} \left\| \boldsymbol{\phi}(\cdot) - \mu_k \right\|$

■

Figure 8.3 illustrates this strategy by showing the distances (dotted arrows) involved in computed the sum-squared criterion. The minimisation of this sum automatically maximises the indicated distance (dot-dashed arrow) between the cluster centres.

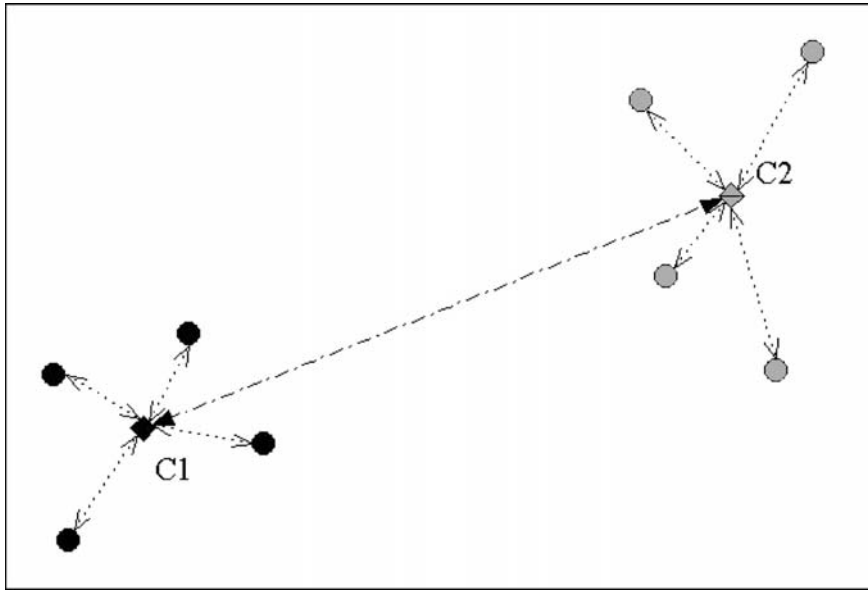

Fig. 8.3. The clustering criterion reduces to finding cluster centres to minimise sum-squared distances.

Remark 8.20 [Stability analysis] Furthermore we can see that this strategy suggests an appropriate pattern function for our stability analysis to

estimate

$$\mathbb{E}_{\mathcal{D}} \min_{1 \leq k \leq N} \|\boldsymbol{\phi}(\mathbf{x}) - \mu_k\|^2;$$

the smaller the bound obtained the better the quality of the clustering achieved. ■

Unfortunately, unlike the optimisation problems that we have described previously, this problem is not convex. Indeed the task of checking if there exists a solution with value better than some threshold turns out to be NP-complete. This class of problems is generally believed not to be solvable in polynomial time, and we are therefore forced to consider heuristic or approximate algorithms that seek solutions that are close to optimal.

We will describe two such approaches in the next section. The first will use a greedy iterative method to seek a local optimum of the cost function, hence failing to be optimal precisely because of its non-convexity. The second method will consider a relaxation of the cost function to give an approximation that can be globally optimised. This is reminiscent of the approach taken to minimise the number of misclassification when applying a support vector machine to non-separable data. By introducing slack variables and using their 1-norm to upper bound the number of misclassifications we can approximately minimise this number through solving a convex optimisation problem.

The greedy method will lead to the well-known k-means algorithm, while the relaxation method gives spectral clustering algorithms. In both cases the approaches can be applied in kernel-defined feature spaces.

Remark 8.21 [Kernel matrices for well-clustered data] We have seen how we can compute distances in a kernel-defined feature space. This will provide the technology required to apply the methods in these spaces. It is often more natural to consider spaces in which unrelated objects have zero inner product. For example using a Gaussian kernel ensures that all distances between points are less than $\sqrt{2}$ with the distances becoming larger as the inputs become more orthogonal. Hence, a good clustering is achieved when the data in the same cluster are concentrated close to the prototype and the prototypes are nearly orthogonal. This means that the kernel matrix for clustered data – assuming without loss of generality that the data are sorted by cluster – will be a perturbation of a block-diagonal matrix with

one block for each cluster

$$\begin{pmatrix} B_1 & 0 & 0 & 0 \\ 0 & B_2 & 0 & 0 \\ 0 & 0 & B_3 & 0 \\ 0 & 0 & 0 & B_4 \end{pmatrix}$$

Note that this would not be the case for other distance functions where, for example, negative inner products were possible. ■

On between cluster distances There is one further property of this minimisation that relates to the means of the clusters. If we consider the covariance matrix of the data, we can perform the following derivation

$$\begin{aligned}
\ell\mathbf{C} &= \sum_{i=1}^{\ell} \left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \boldsymbol{\phi}_S\right)\left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \boldsymbol{\phi}_S\right)' \\
&= \sum_{i=1}^{\ell} \left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i} + \mu_{f_i} - \boldsymbol{\phi}_S\right)\left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i} + \mu_{f_i} - \boldsymbol{\phi}_S\right)' \\
&= \sum_{i=1}^{\ell} \left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i}\right)\left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i}\right)' \\
&\quad + \sum_{k=1}^{N} \left(\sum_{i:f_i=k} \left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_k\right)\right)\left(\mu_k - \boldsymbol{\phi}_S\right)' \\
&\quad + \sum_{k=1}^{N} \left(\mu_k - \boldsymbol{\phi}_S\right) \sum_{i:f_i=k} \left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_k\right)' \\
&\quad + \sum_{i=1}^{\ell} \left(\mu_{f_i} - \boldsymbol{\phi}_S\right)\left(\mu_{f_i} - \boldsymbol{\phi}_S\right)' \\
&= \sum_{i=1}^{\ell} \left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i}\right)\left(\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i}\right)' \\
&\quad + \sum_{k=1}^{N} \left|f^{-1}\left(k\right)\right| \left(\mu_k - \boldsymbol{\phi}_S\right)\left(\mu_k - \boldsymbol{\phi}_S\right)'.
\end{aligned}$$

Taking traces of both sides of the equation we obtain

$$\operatorname{tr}\left(\ell\mathbf{C}\right) = \sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}\left(\mathbf{x}_i\right) - \mu_{f_i}\right\|^2 + \sum_{k=1}^{N} \left|f^{-1}\left(k\right)\right| \left\|\mu_k - \boldsymbol{\phi}_S\right\|^2.$$

The first term on the right-hand side is just the value of the Computation 8.19 that is minimised by the clustering function, while the value of the left-hand side is independent of the clustering. Hence, the clustering criterion automatically maximises the trace of the second term on the right-hand side. This corresponds to maximising

$$\sum_{k=1}^{N} \left|f^{-1}(k)\right| \left\|\mu_k - \boldsymbol{\phi}_S\right\|^2;$$

in other words the sum of the squares of the distances from the overall mean of the cluster means weighted by their size. We again see that optimising the tightness of the clusters automatically forces their centres to be far apart.

8.2.2 Greedy solution: k-means

Proposition 8.18 confirms that we can solve Computation 8.17 by identifying centres of mass of the members of each cluster. The first algorithm we will describe attempts to do just this and is therefore referred to as the *k-means algorithm*. It keeps a set of cluster centroids $C_1, C_2, \ldots, C_N$ that are initialised randomly and then seeks to minimise the expression

$$\sum_{i=1}^{\ell} \left\|\boldsymbol{\phi}(\mathbf{x}_i) - C_{f(\mathbf{x}_i)}\right\|^2, \tag{8.12}$$

by adapting both f as well as the centres. It will converge to a solution in which C_k is the centre of mass of the points assigned to cluster k and hence will satisfy the criterion of Proposition 8.18.

The algorithm alternates between updating f to adjust the assignment of points to clusters and updating the C_k giving the positions of the centres in a two-stage iterative procedure. The first stage simply moves points to the cluster whose cluster centre is closest. Clearly this will reduce the value of the expression in (8.12). The second stage repositions the centre of each cluster at the centre of mass of the points assigned to that cluster. We have already analysed this second stage in Proposition 5.2 showing that moving the cluster centre to the centre of mass of the points does indeed reduce the criterion of (8.12).

Hence, each stage can only reduce the expression (8.12). Since the number of possible clusterings is finite, it follows that after a finite number of iterations the algorithm will converge to a stable clustering assignment provided ties are broken in a deterministic way. If we are to implement in a dual form we must represent the clusters by an indicator matrix $\mathbf{A}$ of dimension $\ell \times N$

containing a 1 to indicate the containment of an example in a cluster

$$\mathbf{A}_{ik} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ is in cluster } k; \\ 0 & \text{otherwise.} \end{cases}$$

We will say that the *clustering is given* by matrix $\mathbf{A}$. Note that each row of $\mathbf{A}$ contains exactly one 1, while the column sums give the number of points assigned to the different clusters. Matrices that have this form will be known as *cluster matrices*. We can therefore compute the coordinates of the centroids C_k as the N columns of the matrix

$$\mathbf{X}'\mathbf{A}\mathbf{D},$$

where $\mathbf{X}$ contains the training example feature vectors as rows and $\mathbf{D}$ is a diagonal $N \times N$ matrix with diagonal entries the inverse of the column sums of $\mathbf{A}$, indicating the number of points ascribed to that cluster. The distances of a new test vector $\phi(\mathbf{x})$ from the centroids is now given by

$$\begin{aligned} \|\boldsymbol{\phi}(\mathbf{x}) - C_k\|^2 &= \|\boldsymbol{\phi}(\mathbf{x})\|^2 - 2\langle \boldsymbol{\phi}(\mathbf{x}), C_k \rangle + \|C_k\|^2 \\ &= \kappa(\mathbf{x}, \mathbf{x}) - 2\left(\mathbf{k}'\mathbf{A}\mathbf{D}\right)_k + \left(\mathbf{D}\mathbf{A}'\mathbf{X}\mathbf{X}'\mathbf{A}\mathbf{D}\right)_{kk}, \end{aligned}$$

where $\mathbf{k}$ is the vector of inner products between $\phi(\mathbf{x})$ and the training examples. Hence, the cluster to which $\phi(\mathbf{x})$ should be assigned is given by

$$\operatorname*{argmin}_{1 \le k \le N} \|\boldsymbol{\phi}(\mathbf{x}) - C_k\|^2 = \operatorname*{argmin}_{1 \le k \le N} \left(\mathbf{D}\mathbf{A}'\mathbf{K}\mathbf{A}\mathbf{D}\right)_{kk} - 2\left(\mathbf{k}'\mathbf{A}\mathbf{D}\right)_k,$$

where $\mathbf{K}$ is the kernel matrix of the training set. This provides the rule for classifying new data. The update rule consists in reassigning the entries in the matrix $\mathbf{A}$ according to the same rule in order to redefine the clusters.

Algorithm 8.22 [Kernel k-means] Matlab code for the kernel k-means algorithm is given in Code Fragment 8.3. ■

Despite its popularity, this algorithm is prone to local minima since the optimisation is not convex. Considerable effort has been devoted to finding good initial guesses or inserting additional constraints in order to limit the effect of this fact on the quality of the solution obtained. In the next section we see two relaxations of the original problem for which we can find the global solution.

8.2.3 Relaxed solution: spectral methods

In this subsection, rather than relying on gradient descent methods to tackle a non-convex problem, we make a convex relaxation of the problem in order

```
% original kernel matrix stored in variable K
% clustering given by a ell x N binary matrix A
% and cluster allocation function f
% d gives the distances to cluster centroids
A = zeros(ell,N);
f = ceil(rand(ell,1)* N);
for i=1,ell
  A(i,f(i)) = 1;
end
change = 1;
while change = 1
  change = 0;
  E = A * diag(1./sum(A));
  Z = ones(ell,1)* diag(E'*K*E)'- 2*K*E;
  [d, ff] = min(Z, [], 2);
  for i=1,ell
   if f(i) ~= ff(i)
    A(i,ff(i)) = 1;
    A(i, f(i)) = 0;
    change = 1;
   end
  end
  f = ff;
end
```

Code Fragment 8.3. Matlab code to perform k-means clustering.

to obtain a closed form approximation. We can then study the approximation and statistical properties of its solutions.

Clustering into two classes We first consider the simpler case when there are just two clusters. In this relaxation we represent the cluster assignment by a vector $\mathbf{y} \in \{-1,+1\}^{\ell}$, that associates to each point a $\{-1,+1\}$ label. For the two classes case the clustering quality criterion described above is minimised by maximising

$$\sum_{y_i \neq y_j} \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2 .$$

Assuming that the data is normalised and the sizes of the clusters are equal this will correspond to minimising the so-called cut cost

$$\begin{aligned} 2\sum_{y_i \neq y_j} \kappa(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{i,j=1}^{\ell} \kappa(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i,j=1}^{\ell} y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to } \mathbf{y} &\in \{-1,+1\}^{\ell}, \end{aligned}$$

since it measures the kernel 'weight' between vertices in different clusters. Hence, we must solve

$$\begin{array}{ll} \max & \mathbf{y}'\mathbf{K}\mathbf{y} \\ \text{subject to} & \mathbf{y} \in \{-1, +1\}^{\ell}. \end{array}$$

We can relax this optimisation by removing the restriction that $\mathbf{y}$ be a binary vector while controlling its norm. This is achieved by maximising the Raleigh quotient (3.2)

$$\max \frac{\mathbf{y}'\mathbf{K}\mathbf{y}}{\mathbf{y}'\mathbf{y}}.$$

As observed in Chapter 3 this is solved by the eigenvector of the matrix $\mathbf{K}$ corresponding to the largest eigenvalue with the value of the quotient equal to the eigenvalue λ_1. Hence, we obtain a lower bound on the cut cost of

$$0.5 \left(\sum_{i,j=1}^{\ell} \kappa(\mathbf{x}_i, \mathbf{x}_j) - \lambda_1 \right),$$

giving a corresponding lower bound on the value of the sum-squared criterion. Though such a lower bound is useful, the question remains as to whether the approach can suggest useful clusterings of the data. A very natural way to do so in this two-cluster case is simply to threshold the vector $\mathbf{y}$ hence converting it to a binary clustering vector. This naive approach can deliver surprisingly good results though there is no a priori guarantee attached to the quality of the solution.

Remark 8.23 [Alternative criterion] It is also possible to consider minimising a ratio between the cut size and a measure of the size of the clusters. This leads through similar relaxations to different eigenvalue problems. For example if we let $\mathbf{D}$ be the diagonal matrix with entries

$$\mathbf{D}_{ii} = \sum_{j=1}^{\ell} \mathbf{K}_{ij},$$

then useful partitions can be derived from the eigenvectors of

$$\mathbf{D}^{-1}\mathbf{K}, \quad \mathbf{D}^{-1/2}\mathbf{K}\mathbf{D}^{-1/2} \quad \text{and} \quad \mathbf{K} - \mathbf{D}$$

with varying justifications. In all cases thresholding the resulting vectors delivers the corresponding partitions. Generally the approach is motivated using the Gaussian kernel with its useful properties discussed above. ■

Multiclass clustering We now consider the more general problem of multiclass clustering. We start with an equivalent formulation of the sum-of-squares minimization problem as a trace maximization under special constraints. By successively relaxing these constraints we are led to an approximate algorithm with nice global properties.

Consider the derivation and notation introduced to obtain the program for k-means clustering. We can compute the coordinates of the centroids C_k as the N columns of the matrix

$$\mathbf{X}'\mathbf{A}\mathbf{D},$$

where $\mathbf{X}$ is the data matrix, $\mathbf{A}$ a matrix assigning points to clusters and $\mathbf{D}$ a diagonal matrix with inverses of the cluster sizes on the diagonal. Consider now the matrix

$$\mathbf{X}'\mathbf{A}\mathbf{D}\mathbf{A}'.$$

It has ℓ columns with the ith column a copy of the cluster centroid corresponding to the ith example. Hence, we can compute the sum-squares of the distances from the examples to their corresponding cluster centroid as

$$\begin{aligned}\left\|\mathbf{X}'\mathbf{A}\mathbf{D}\mathbf{A}' - \mathbf{X}'\right\|_F^2 &= \left\|\left(\mathbf{I}_\ell - \mathbf{A}\mathbf{D}\mathbf{A}'\right)\mathbf{X}\right\|_F^2 \\ &= \operatorname{tr}\left(\mathbf{X}'\left(\mathbf{I}_\ell - \mathbf{A}\mathbf{D}\mathbf{A}'\right)\mathbf{X}\right) \\ &= \operatorname{tr}\left(\mathbf{X}\mathbf{X}'\right) - \operatorname{tr}\left(\sqrt{\mathbf{D}}\mathbf{A}'\mathbf{X}\mathbf{X}'\mathbf{A}\sqrt{\mathbf{D}}\right),\end{aligned}$$

since

$$\left(\mathbf{I}_\ell - \mathbf{A}\mathbf{D}\mathbf{A}'\right)^2 = \left(\mathbf{I}_\ell - \mathbf{A}\mathbf{D}\mathbf{A}'\right)$$

as $\mathbf{A}'\mathbf{A}\mathbf{D} = \sqrt{\mathbf{D}}\mathbf{A}'\mathbf{A}\sqrt{\mathbf{D}} = \mathbf{I}_N$, indicating that $(\mathbf{I}_\ell - \mathbf{A}\mathbf{D}\mathbf{A}')$ is a projection matrix. We have therefore shown the following proposition.

Proposition 8.24 *The sum-squares cost function* $\operatorname{ss}(\mathbf{A})$ *of a clustering, given by matrix* $\mathbf{A}$ *can be expressed as*

$$\operatorname{ss}(\mathbf{A}) = \operatorname{tr}(\mathbf{K}) - \operatorname{tr}\left(\sqrt{\mathbf{D}}\mathbf{A}'\mathbf{K}\mathbf{A}\sqrt{\mathbf{D}}\right),$$

where $\mathbf{K}$ *is the kernel matrix and* $\mathbf{D} = \mathbf{D}(\mathbf{A})$ *is the diagonal matrix with the inverses of the column sums of* $\mathbf{A}$*.*

Since the first term is not affected by the choice of clustering, the proposition leads to the Computation.

Computation 8.25 [Multiclass clustering] We can minimise the cost $\mathrm{ss}(\mathbf{A})$ by solving

$$\begin{array}{ll} \max_{\mathbf{A}} & \mathrm{tr}\left(\sqrt{\mathbf{D}}\mathbf{A}'\mathbf{K}\mathbf{A}\sqrt{\mathbf{D}}\right) \\ \text{subject to} & \mathbf{A} \text{ is a cluster matrix and } \mathbf{D} = \mathbf{D}(\mathbf{A}). \end{array}$$

■

Remark 8.26 [Distances of the cluster centres from the origin] We can see that $\mathrm{tr}\left(\sqrt{\mathbf{D}}\mathbf{A}'\mathbf{K}\mathbf{A}\sqrt{\mathbf{D}}\right)$ is the sum of the squares of the cluster centroids, relating back to our previous observations about the sum-square criterion corresponding to maximising the sum-squared distances of the centroids from the overall centre of mass of the data. This shows that this holds for the origin as well and since an optimal clustering is invariant to translations of the coordinate axes, this will be true of the sum-squared distances to any fixed point. ■

We have now arrived at a constrained optimisation problem whose solution solves to the min-squared clustering criterion. Our aim now is to relax the constraints to obtain a problem that can be optimised exactly, but whose solution will not correspond precisely to a clustering. Note that the matrices $\mathbf{A}$ and $\mathbf{D} = \mathbf{D}(\mathbf{A})$ satisfy

$$\sqrt{\mathbf{D}}\mathbf{A}'\mathbf{A}\sqrt{\mathbf{D}} = \mathbf{I}_N = \mathbf{H}'\mathbf{H}, \tag{8.13}$$

where $\mathbf{H} = \mathbf{A}\sqrt{\mathbf{D}}$. Hence, only requiring that this $\ell \times N$ matrix satisfies equation (8.13), we obtain the relaxed maximization problem given in the computation.

Computation 8.27 [Relaxed multiclass clustering] The relaxed maximisation for multiclass clustering is obtained by solving

$$\begin{array}{ll} \max & \mathrm{tr}(\mathbf{H}'\mathbf{K}\mathbf{H}) \\ \text{subject to} & \mathbf{H}'\mathbf{H} = \mathbf{I}_N, \end{array}$$

■

The solutions of Computation 8.27 will not correspond to clusterings, but may lead to good clusterings. The important property of the relaxation is that it can be solved in closed-form.

Proposition 8.28 *The maximum of the trace* $\mathrm{tr}(\mathbf{H}'\mathbf{K}\mathbf{H})$ *over all* $\ell \times N$ *matrices satisfying* $\mathbf{H}'\mathbf{H} = \mathbf{I}_N$ *is equal to the sum of the first* N *eigenvalues*

of $\mathbf{K}$

$$\max_{\mathbf{H}'\mathbf{H}=\mathbf{I}_N} \operatorname{tr}\left(\mathbf{H}'\mathbf{K}\mathbf{H}\right) = \sum_{k=1}^{N} \lambda_k$$

while the $\mathbf{H}^*$ *realising the optimum is given by*

$$\mathbf{V}_N\mathbf{Q},$$

where $\mathbf{Q}$ *is an arbitrary* $N \times N$ *orthonormal matrix and* $\mathbf{V}_N$ *is the* $\ell \times N$ *matrix composed of the first* N *eigenvectors of* $\mathbf{K}$*. Furthermore, we can lower bound the sum-squared error of the best clustering by*

$$\begin{aligned}\min_{\mathbf{A}\ \textit{clustering matrix}} \operatorname{ss}(\mathbf{A}) &= \operatorname{tr}(\mathbf{K}) - \max_{\mathbf{A}\ \textit{clustering matrix}} \operatorname{tr}\left(\sqrt{\mathbf{D}}\mathbf{A}'\mathbf{K}\mathbf{A}\sqrt{\mathbf{D}}\right) \\ &\geq \operatorname{tr}(\mathbf{K}) - \max_{\mathbf{H}'\mathbf{H}=\mathbf{I}_N} \operatorname{tr}\left(\mathbf{H}'\mathbf{K}\mathbf{H}\right) = \sum_{k=N+1}^{\ell} \lambda_k.\end{aligned}$$

Proof Since $\mathbf{H}'\mathbf{H} = \mathbf{I}_N$ the operator $\mathbf{P} = \mathbf{H}\mathbf{H}'$ is a rank N projection. This follows from the fact that $(\mathbf{H}\mathbf{H}')^2 = \mathbf{H}\mathbf{H}'\mathbf{H}\mathbf{H}' = \mathbf{H}\mathbf{I}_\mathbf{N}\mathbf{H}' = \mathbf{H}\mathbf{H}'$ and $\operatorname{rank}\mathbf{H} = \operatorname{rank}\mathbf{H}'\mathbf{H} = \operatorname{rank}\mathbf{I}_N = N$. Therefore

$$\mathbf{I}_N\mathbf{H}'\mathbf{K}\mathbf{H}\mathbf{I}_N = \mathbf{H}'\mathbf{H}\mathbf{H}'\mathbf{X}\mathbf{X}'\mathbf{H}\mathbf{H}'\mathbf{H} = \left\|\mathbf{H}'\mathbf{P}\mathbf{X}\right\|_F^2 = \left\|\mathbf{P}\mathbf{X}\right\|_F^2.$$

Hence, we seek the N-dimensional projection of the columns of $\mathbf{X}$ that maximises the resulting sum of the squared norms. Treating these columns as the training vectors and viewing maximising the projection as minimising the residual we can apply Proposition 6.12. It follows that the maximum will be realised by the eigensubspace spanned by the N eigenvectors corresponding to the largest eigenvalues for the matrix $\mathbf{X}\mathbf{X}' = \mathbf{K}$. Clearly, the projection is only specified up to an arbitrary $N \times N$ orthonormal transformation $\mathbf{Q}$. □

At first sight we have not gained a great deal by moving to the relaxed version of the problem. It is true that we have obtained a strict lower bound on the quality of clustering that can be achieved, but the matrix $\mathbf{H}$ that realises that lower bound does not in itself immediately suggest a method of performing a clustering that comes close to the bound. Furthermore, in the case of multi-class clustering we do not have an obvious simple thresholding algorithm for converting the result of the eigenvector analysis into a clustering as we found in the two cluster examples. We mention three possible approaches.

Re-clustering One approach is to apply a different clustering algorithm in the reduced N-dimensional representation of the data, when we map each example to the corresponding row of the matrix $\mathbf{V}_N$, possibly after performing a renormalisation.

Eigenvector approach We will describe a different method that is related to the proof of Proposition 8.28. Consider the choice $\mathbf{H}^* = \mathbf{V}_N$ that realises the optimum bound of the proposition. Let $\mathbf{W} = \mathbf{V}_N\sqrt{\mathbf{\Lambda}_N}$ be obtained from $\mathbf{V}_N$ by multiplying column i by $\sqrt{\lambda_i}$, $i = 1, \ldots, N$. We now form the cluster matrix $\mathbf{A}$ by setting the largest entry in each row of $\mathbf{W}$ to 1 and the remaining entries to 0.

QR approach An alternative approach is inspired by a desire to construct an approximate cluster matrix which is related to $\mathbf{V}_N$ by an orthonormal transformation

$$\mathbf{A} \approx \mathbf{V}_N\mathbf{Q},$$

implying that

$$\mathbf{V}'_N \approx \mathbf{Q}\mathbf{A}'.$$

If we perform a QR decomposition of $\mathbf{V}'_N$ we obtain

$$\mathbf{V}'_N = \hat{\mathbf{Q}}\mathbf{R}$$

with $\hat{\mathbf{Q}}$ an $N \times N$ orthogonal matrix and $\mathbf{R}$ an $N \times \ell$ upper triangular. By assigning vector i to the cluster index by the row with largest entry in the column i of matrix $\mathbf{R}$, we obtain a cluster matrix $\mathbf{A}' \approx \mathbf{R}$, hence giving a value of $\mathrm{ss}(\mathbf{A})$ close to that given by the bound.

8.3 Data visualisation

Visualisation refers to techniques that can present a dataset

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$$

in a manner that reveals some underlying structure of the data in a way that is easily understood or appreciated by a user. A clustering is one type of structure that can be visualised. If for example there is a natural clustering of the data into say four clusters, each one grouped tightly around a separate centroid, our understanding of the data is greatly enhanced by displaying the four centroids and indicating the tightness of the clustering around them. The centroids can be thought of as prototypical data items.

Hence, for example in a market analysis, customers might be clustered into a set of typical types with individual customers assigned to the type to which they are most similar.

In this section we will consider a different type of visualisation. Our aim is to provide a two- or three-dimensional 'mapping' of the data, hence displaying the data points as dots on a page or as points in a three-dimensional image. This type of visualisation is of great importance in many data mining tasks, but it assumes a special role in kernel methods, where we typically embed the data into a high-dimensional vector space. We have already considered measures for assessing the quality of an embedding such as the classifier margin, correlation with output values and so on. We will also in later chapters be looking into procedures for transforming prior models into embeddings. However once we arrive at the particular embedding, it is also important to have ways of visually displaying the data in the chosen feature space. Looking at the data helps us get a 'feel' for the structure of the data, hence suggesting why certain points are outliers, or what type of relations can be found. This can in turn help us pick the best algorithm out of the toolbox of methods we have been assembling since Chapter 5. In other words being able to 'see' the relative positions of the data in the feature space plays an important role in guiding the intuition of the data analyst.

Using the first few principal components, as computed by the PCA algorithm of Chapter 6, is a well-known method of visualisation forming the core of the classical multidimensional scaling algorithm. As already demonstrated in Proposition 6.12 PCA minimises the sum-squared norms of the residuals between the subspace representation and the actual data.

Naturally if one wants to look at the data from an angle that emphasises a certain property, such as a given dichotomy, other projections can be better suited, for example using the first two partial least squares features. In this section we will assume that the feature space has already been adapted to best capture the view of the data we are concerned with but typically using a high-dimensional kernel representation. Our main concern will therefore be to develop algorithms that can find low-dimensional representations of high-dimensional data.

Multidimensional scaling This problem has received considerable attention in multivariate statistics under the heading of multidimensional scaling (MDS). This is a series of techniques directly aimed at finding optimal low-dimensional embeddings of data mostly for visualisation purposes. The starting point for MDS is traditionally a matrix of distances or similarities

rather than a Gram matrix of inner products or even a Euclidean embedding. Indeed the first stages of the MDS process aim to convert the matrix of similarities into a matrix of inner products. For metric MDS it is assumed that the distances correspond to embeddings in a Euclidean space, while for non-metric MDS these similarities can be measured in any way. Once an approximate inner product matrix has been formed, classical MDS then uses the first two or three eigenvectors of the eigen-decomposition of the resulting Gram matrix to define two- or three-dimensional projections of the points for visualisation. Hence, if we make use of a kernel-defined feature space the first stages are no longer required and MDS reduces to computing the first two or three kernel PCA projections.

Algorithm 8.29 [MDS for kernel-embedded data] The MDS algorithm for data in a kernel-defined feature space is as follows:

input	Data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$, dimension $k = 2, 3$.
process	$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, \ldots, \ell$ $\mathbf{K} - \frac{1}{\ell}\mathbf{j}\mathbf{j}'\mathbf{K} - \frac{1}{\ell}\mathbf{K}\mathbf{j}\mathbf{j}' + \frac{1}{\ell^2}(\mathbf{j}'\mathbf{K}\mathbf{j})\mathbf{j}\mathbf{j}'$, $[\mathbf{V}, \mathbf{\Lambda}] = \text{eig}(\mathbf{K})$ $\boldsymbol{\alpha}^j = \frac{1}{\sqrt{\lambda_j}}\mathbf{v}_j$, $j = 1, \ldots, k$. $\tilde{\mathbf{x}}_i = \left(\sum_{i=1}^{\ell} \boldsymbol{\alpha}_i^j \kappa(\mathbf{x}_i, \mathbf{x})\right)_{j=1}^{k}$
output	Display transformed data $\tilde{S} = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_\ell\}$.

■

Visualisation quality We will consider a further method of visualisation strongly related to MDS, but which is motivated by different criteria for assessing the quality of the representation of the data.

We can define the problem of visualisation as follows. Given a set of points

$$S = \{\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_\ell)\}$$

in a kernel-defined feature space F with

$$\boldsymbol{\phi} : X \longrightarrow F,$$

find a projection $\boldsymbol{\tau}$ from X into $\mathbb{R}^k$, for small k such that

$$\|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j)\| \approx \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|, \text{ for } i, j = 1, \ldots, \ell.$$

We will use $\boldsymbol{\tau}_s$ to denote the projection onto the sth component of $\boldsymbol{\tau}$ and with a slight abuse of notation, as a vector of these projection values indexed by the training examples. As mentioned above it follows from Proposition 6.12

that the embedding determined by kernel PCA minimises the sum-squared residuals

$$\sum_{i=1}^{\ell} \|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_i)\|^2,$$

where we make $\boldsymbol{\tau}$ an embedding into a k-dimensional subspace of the feature space F. Our next method aims to control more directly the relationship between the original and projection distances by solving the following computation.

Computation 8.30 [Visualisation quality] The quality of a visualisation can be optimsed as follows

$$\begin{aligned} \min_{\boldsymbol{\tau}} E(\boldsymbol{\tau}) &= \sum_{i,j=1}^{\ell} \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle \|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j)\|^2 \\ &= \sum_{i,j=1}^{\ell} \kappa(\mathbf{x}_i, \mathbf{x}_j) \|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j)\|^2, \\ \text{subject to } \|\boldsymbol{\tau}_s\| &= 1, \quad \boldsymbol{\tau}_s \perp \mathbf{j}, \quad s = 1, \ldots, k, \\ \text{and } \boldsymbol{\tau}_s &\perp \boldsymbol{\tau}_t, \quad s, t = 1, \ldots, k. \end{aligned} \tag{8.14}$$

■

Observe that it follows from the constraints that

$$\begin{aligned} \sum_{i,j=1}^{\ell} \|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j)\|^2 &= \sum_{i,j=1}^{\ell} \sum_{s=1}^{k} (\boldsymbol{\tau}_s(\mathbf{x}_i) - \boldsymbol{\tau}_s(\mathbf{x}_j))^2 \\ &= \sum_{s=1}^{k} \sum_{i,j=1}^{\ell} (\boldsymbol{\tau}_s(\mathbf{x}_i) - \boldsymbol{\tau}_s(\mathbf{x}_j))^2 \\ &= 2 \sum_{s=1}^{k} \left(\ell \sum_{i=1}^{\ell} \boldsymbol{\tau}_s(\mathbf{x}_i)^2 - \sum_{i,j=1}^{\ell} \boldsymbol{\tau}_s(\mathbf{x}_i) \boldsymbol{\tau}_s(\mathbf{x}_j) \right) \\ &= 2\ell k - 2 \sum_{s=1}^{k} \sum_{i=1}^{\ell} \boldsymbol{\tau}_s(\mathbf{x}_i) \sum_{j=1}^{\ell} \boldsymbol{\tau}_s(\mathbf{x}_j) = 2\ell k. \end{aligned}$$

It therefore follows that, if the data is normalised, solving Computation 8.30 corresponds to minimising

$$E(\boldsymbol{\tau}) = \sum_{i,j=1}^{\ell} \left(1 - 0.5 \|\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}(\mathbf{x}_j)\|^2\right) \|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j)\|^2$$

$$= \quad 2\ell k - \sum_{i,j=1}^{\ell} 0.5 \left\| \boldsymbol{\phi}\left(\mathbf{x}_i\right) - \boldsymbol{\phi}\left(\mathbf{x}_j\right) \right\|^2 \left\| \boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j) \right\|^2,$$

hence optimising the correlation between the original and projected squared distances. More generally we can see minimisation as aiming to put large distances between points with small inner products and small distances between points having large inner products. The constraints ensure equal scaling in all dimensions centred around the origin, while the different dimensions are required to be mutually orthogonal to ensure no structure is unnecessarily reproduced.

Our next theorem will characterise the solution of the above optimisation using the eigenvectors of the so-called Laplacian matrix. This matrix can also be used in clustering as it frequently possesses more balanced properties than the kernel matrix. It is defined as follows.

Definition 8.31 [Laplacian matrix] The *Laplacian matrix* $\mathbf{L}(\mathbf{K})$ of a kernel matrix $\mathbf{K}$ is defined by

$$\mathbf{L}(\mathbf{K}) = \mathbf{D} - \mathbf{K},$$

where $\mathbf{D}$ is the diagonal matrix with entries

$$\mathbf{D}_{ii} = \sum_{j=1}^{\ell} \mathbf{K}_{ij}.$$

■

Observe the following simple property of the Laplacian matrix. Given any real vector $\mathbf{v} = (v_1, \ldots, v_\ell) \in \mathbb{R}^\ell$

$$\begin{aligned} \sum_{i,j=1}^{\ell} \mathbf{K}_{ij} \left(v_i - v_j\right)^2 &= 2 \sum_{i,j=1}^{\ell} \mathbf{K}_{ij} v_i^2 - 2 \sum_{i,j=1}^{\ell} \mathbf{K}_{ij} v_i v_j \\ &= 2\mathbf{v}'\mathbf{D}\mathbf{v} - 2\mathbf{v}'\mathbf{K}\mathbf{v} = 2\mathbf{v}'\mathbf{L}(\mathbf{K})\,\mathbf{v}. \end{aligned}$$

It follows that the all 1s vector $\mathbf{j}$ is an eigenvector of $\mathbf{L}(\mathbf{K})$ with eigenvalue 0 since the sum is zero if $v_i = v_j = 1$. It also implies that if the kernel matrix has positive entries then $\mathbf{L}(\mathbf{K})$ is positive semi-definite. In the statement of the following theorem we separate out λ_1 as the eigenvalue 0, while ordering the remaining eigenvalues in ascending order.

Theorem 8.32 *Let*

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$$

be a set of points with kernel matrix $\mathbf{K}$. *The visualisation problem given in Computation 8.30 is solved by computing the eigenvectors* $\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^\ell$ *with corresponding eigenvalues* $0 = \lambda_1,\ \lambda_2 \leq \ldots \leq \lambda_\ell$ *of the Laplacian matrix* $\mathbf{L}(\mathbf{K})$. *An optimal embedding* $\boldsymbol{\tau}$ *is given by* $\boldsymbol{\tau}_i = \mathbf{v}^{i+1}$, $i = 1, \ldots, k$ *and the minimal value of* $E(\boldsymbol{\tau})$ *is*

$$2\sum_{\ell=2}^{k+1} \lambda_\ell.$$

If $\lambda_{k+1} < \lambda_{k+2}$ *then the optimal embedding is unique up to orthonormal transformations in* $\mathbb{R}^k$.

Proof The criterion to be minimised is

$$\begin{aligned}\sum_{i,j=1}^{\ell} \kappa(\mathbf{x}_i, \mathbf{x}_j) \left\|\boldsymbol{\tau}(\mathbf{x}_i) - \boldsymbol{\tau}(\mathbf{x}_j)\right\|^2 &= \sum_{s=1}^{k}\sum_{i,j=1}^{\ell} \kappa(\mathbf{x}_i, \mathbf{x}_j)\left(\boldsymbol{\tau}_s(\mathbf{x}_i) - \boldsymbol{\tau}_s(\mathbf{x}_j)\right)^2 \\ &= 2\sum_{s=1}^{k} \boldsymbol{\tau}_s' \mathbf{L}(\mathbf{K})\boldsymbol{\tau}_s.\end{aligned}$$

Taking into account the normalisation and orthogonality constraints gives the solution as the eigenvectors of $\mathbf{L}(\mathbf{K})$ by the usual characterisation of the Raleigh quotients. The uniqueness again follows from the invariance under orthonormal transformations together with the need to restrict to the subspace spanned by the first k eigenvectors. □

The implementation of this visualisation technique is very straightforward.

Algorithm 8.33 [Data visualisation] Matlab code for the data visualisation algorithm is given in Code Fragment 8.4. ■

```
% original kernel matrix stored in variable K
% tau gives the embedding in k dimensions
D = diag(sum(K));
L = D - K;
[V,Lambda] = eig(L);
Lambda = diag(Lambda);
I = find(abs(Lambda) > 0.00001)
objective = 2*sum(Lambda(I(1:k)))
Tau = V(:,I(1:k));
plot(Tau(:,1), Tau(:,2), 'x')
```

Code Fragment 8.4. Matlab code to implementing low-dimensional visualisation.

8.4 Summary

- The problem of learning a ranking function can be solved using kernel methods.
- Stability analysis motivates the development of an optimisation criterion that can be efficiently solved using convex quadratic programming.
- Practical considerations suggest an on-line version of the method. A stability analysis is derived for the application of a perceptron-style algorithm suitable for collaborative filtering.
- The quality of a clustering of points can be measured by the expected within cluster distances.
- Analysis of this measure shows that it has many attractive properties. Unfortunately, minimising the criterion on the training sample is NP-hard.
- Two kernel approaches to approximating the solution are presented. The first is the dual version of the k-means algorithm. The second relies on solving exactly a relaxed version of the problem in what is known as spectral clustering.
- Data in the feature space can be visualised by finding low-dimensional embeddings satisfying certain natural criteria.

8.5 Further reading and advanced topics

The problem of identifying clusters in a set of data has been a classic topic of statistics and machine learning for many years. The book [40] gives a general introduction to many of the main ideas, particularly to the various cost functions that can be optimised, as well as presenting the many convenient properties of the cost function described in this book, the expected square distance from cluster-centre. The classical k-means algorithm greedily attempts to optimise this cost function.

On the other hand, the clustering approach based on spectral analysis is relatively new, and still the object of current research. In the NIPS conference 2001, for example, the following independent works appeared: [34], [169], [103], [12]. Note also the article [74]. The statistical stability of spectral clustering has been investigated in [126], [103].

The kernelisation of k-means is a folk algorithm within the kernel machines community, being used as an example of kernel-based algorithms for some time. The combination of spectral clustering with kernels is more recent.

Also the problem of learning to rank data has been addressed by several authors in different research communities. For recent examples, see Herbrich

et al. [56] and Crammer and Singer [30]. Our discussion in this chapter mostly follows Crammer and Singer.

Visualising data has for a long time been a vital component of data mining, as well as of graph theory, where visualising is also known as graph drawing. Similar ideas were developed in multivariate statistics, with the problem of multidimensional scaling [159]. The method presented in this chapter was developed in the chemical literature, but it has natural equivalents in graph drawing [105].

For constantly updated pointers to online literature and free software see the book's companion website: www.kernel-methods.net