

---

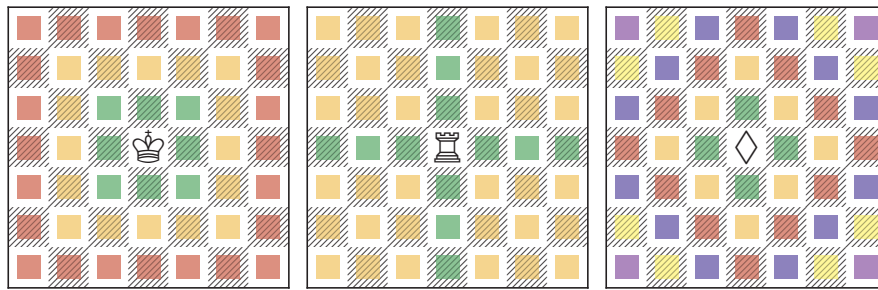
## Distance-based models

---

**M**ANY FORMS OF LEARNING are based on generalising from training data to unseen data by exploiting the similarities between the two. With grouping models such as decision trees these similarities take the form of an equivalence relation or partition of the instance space: two instances are similar whenever they end up in the same segment of this partition. In this chapter we consider learning methods that utilise more graded forms of similarity. There are many different ways in which similarity can be measured, and in [Section 8.1](#) we take a look at the most important of them. [Section 8.2](#) is devoted to a discussion of two key concepts in distance-based machine learning: neighbours and exemplars. In [Section 8.3](#) we consider what is perhaps the best-known distance-based learning method: the nearest-neighbour classifier. [Section 8.4](#) investigates *K*-means clustering and close relatives, and [Section 8.5](#) looks at hierarchical clustering by constructing dendrograms. Finally, in [Section 8.6](#) we discuss how several of these methods can be extended using the kind of kernels that we saw in the previous chapter.

### 8.1 So many roads...

It may seem odd at first that there should be many ways to measure distance. I am not referring to the fact that distance can be measured on different scales (kilometres, miles, nautical miles, and so on), as such changes of scale are simple monotonic trans-

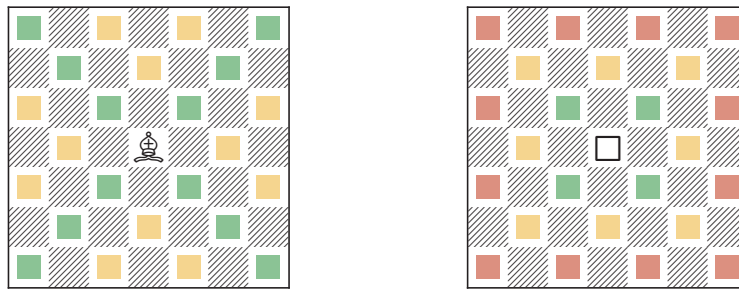


**Figure 8.1.** (left) Distance as experienced by a King on a chessboard: green squares are one move away, orange ones two moves and red ones three moves. The shape formed by equidistant squares from the current position is itself a square. (middle) A Rook can travel any number of squares in one move, but only horizontally or vertically. No square is further than two moves away. (right) The (fictional) KRook combines the restrictions of King and Rook: it can move only one square at a time, and only horizontally or vertically. Equidistant squares now form a lozenge.

formations and do not fundamentally alter the distance measure. A better intuition is obtained by taking the mode of travel into account. Clearly, when travelling from Bristol to Amsterdam by train you travel a larger distance than when travelling by plane, because planes are less restricted in their paths than trains. We will explore this a bit further by considering the game of chess.

In chess, each piece is governed by a set of rules that restrict its possible moves. These restrictions can be directional: for instance, King and Queen can move horizontally, vertically and diagonally, while a Bishop can only move diagonally, a Rook only horizontally and vertically, and pawns only upwards. King and pawn are further restricted by the fact that they can move only one square at a time, whereas Queen, Rook and Bishop can move any number of squares in a single allowed direction. Finally, a Knight moves according to a very specific pattern (one diagonal step and one horizontal or vertical step in a single move).

Although these pieces move around on the same board, they experience distances in very different ways. For example, the next square down is one move away for King, Queen and Rook; three moves away for a Knight; and unreachable for Bishop or pawn. This is, of course, very similar to our experience in the real world. Trains and cars can only move along tracks or roads, like a Bishop, which leaves remote places unreachable. A mountain range can mean large detours when travelling by car, train or on foot, but is easy to cross when flying. On an underground, two stations a few streets away may be only reachable with several changes of line, not unlike the way a Knight can reach a nearby square only in two or three moves. And on foot we are most flexible but



**Figure 8.2. (left)** The Bishop's world: squares are either one or two moves away, or else unreachable. **(right)** The fictional Bing combines the restrictions of King and Bishop: it can only move one square at a time, and only diagonally. Equidistant squares now form a punctuated square.

also slow, like a King.

Figure 8.1 visualises the distances experienced by King and Rook. Both can reach all parts of the chessboard, but a Rook can travel much faster. In fact, a Rook can reach any square in either one or two moves (assuming no other pieces are in its way). All squares one move away form a cross, and the remaining squares are one additional move away. A King will often have to travel more than two moves to reach a particular square (although there are also squares that the King can reach in one move while a Rook needs two). The squares one move away form a small square shape around the current position; those two moves away form a larger square around the smaller square; and so on. Figure 8.1 (right) shows a piece that doesn't exist in chess, but could. It combines the restrictions of King and Rook, and I therefore call it a KRook. Like a King, it can only move one square at a time; and like a Rook, it can only move horizontally and vertically. For the KRook, equidistant squares form a sort of lozenge around the current position.


Figure 8.2 (left) visualises the Bishop's moves. The Bishop is somewhat similar to the Rook in that some squares (those of the same colour as its current square) are never more than two moves away; however, the remaining squares of the other colour are unreachable. Combining the restrictions of the Bishop (only diagonal moves) with those of the King (one square per move) we obtain another fictional piece, the Bing (Figure 8.2 (right)). We could say that the world of Bishops and Bings is rotated 45 degrees, compared with the world of Rooks and KRooks.

What's the relevance of all this when trying to understand distance-based machine learning, you may ask? Well, the rank (row) and file (column) on a chessboard is not unlike a discrete or categorical feature in machine learning (in fact, since ranks and files are ordered, they are *ordinal features*, as we will further discuss in Chapter 10). We can switch to real-valued features by imagining a 'continuous' chessboard with

infinitely many, infinitesimally narrow ranks and files. Squares now become points, and distances are not expressed as the number of squares travelled, but simply as a real number on some scale. If we now look at the shapes obtained by connecting equidistant points, we see that many of these carry over from the discrete to the continuous case. For a King, for example, all points a given fixed distance away still form a square around the current position; and for a KRook they still form a square rotated 45 degrees. As it happens, these are special cases of the following generic concept.

**Definition 8.1 (Minkowski distance).** If  $\mathcal{X} = \mathbb{R}^d$ , the **Minkowski distance** of order  $p > 0$  is defined as

$$\text{Dis}_p(\mathbf{x}, \mathbf{y}) = \left( \sum_{j=1}^d |x_j - y_j|^p \right)^{1/p} = \|\mathbf{x} - \mathbf{y}\|_p$$

where  $\|\mathbf{z}\|_p = \left( \sum_{j=1}^d |z_j|^p \right)^{1/p}$  is the  **$p$ -norm** (sometimes denoted  $L_p$  norm) of the vector  $\mathbf{z}$ . We will often refer to  $\text{Dis}_p$  simply as the  $p$ -norm. 

So, the **2-norm** refers to the familiar **Euclidean distance**

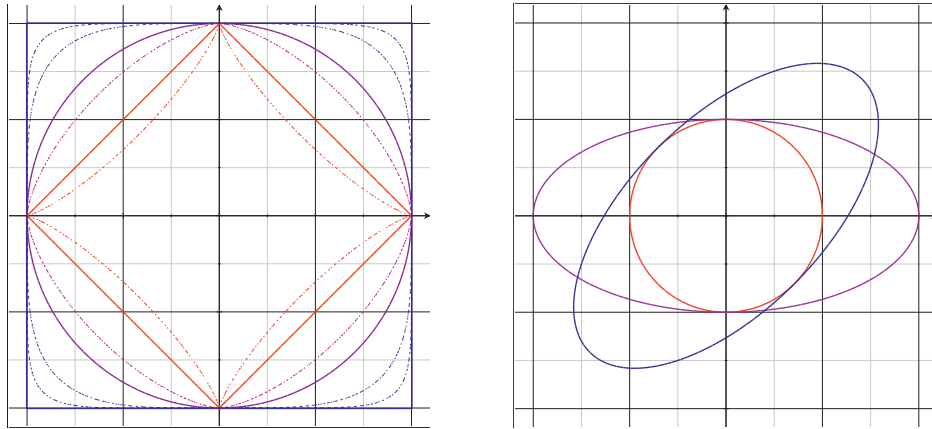
$$\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^d (x_j - y_j)^2} = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})}$$

which measures distance ‘as the crow flies’. Two other values of  $p$  can be related back to the chess example. The **1-norm** denotes **Manhattan distance**, also called **cityblock distance**:

$$\text{Dis}_1(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d |x_j - y_j|$$

This is the distance if we can only travel along coordinate axes: similar to a taxi in Manhattan or other cities whose streets follow a regular grid pattern, but also the distance experienced by our fictional KRook piece. If we now let  $p$  grow larger, the distance will be more and more dominated by the largest coordinate-wise distance, from which we can infer that  $\text{Dis}_\infty(\mathbf{x}, \mathbf{y}) = \max_j |x_j - y_j|$ . This is the distance experienced by the King on a chessboard, who can move diagonally as well as horizontally and vertically but only one step at a time; it is also called **Chebyshev distance**. Figure 8.3 (left) visualises equidistant points from the origin using Minkowski distances of various orders. It can be seen that Euclidean distance is the only Minkowski distance that is rotation-invariant – in other words, special significance is given to the directions of the coordinate axes whenever  $p \neq 2$ . Minkowski distances do not refer to a particular choice of origin and are therefore translation-invariant, but none of them are scaling-invariant.

You will sometimes see references to the **0-norm** (or  $L_0$  norm) which counts the number of non-zero elements in a vector. The corresponding distance then counts the



**Figure 8.3.** (left) Lines connecting points at order- $p$  Minkowski distance 1 from the origin for (from inside)  $p = 0.8$ ;  $p = 1$  (Manhattan distance, the **rotated square in red**);  $p = 1.5$ ;  $p = 2$  (Euclidean distance, the **violet circle**);  $p = 4$ ;  $p = 8$ ; and  $p = \infty$  (Chebyshev distance, the **blue rectangle**). Notice that for points on the coordinate axes all distances agree. For the other points, our reach increases with  $p$ ; however, if we require a rotation-invariant distance metric then Euclidean distance is our only choice. (right) The rotated ellipse  $\mathbf{x}^T \mathbf{R}^T \mathbf{S}^2 \mathbf{R} \mathbf{x} = 1/4$ ; the axis-parallel ellipse  $\mathbf{x}^T \mathbf{S}^2 \mathbf{x} = 1/4$ ; and the circle  $\mathbf{x}^T \mathbf{x} = 1/4$  ( $\mathbf{R}$  and  $\mathbf{S}$  as in Example 8.1).

number of positions in which vectors  $\mathbf{x}$  and  $\mathbf{y}$  differ. This is not strictly a Minkowski distance; however, we can define it as

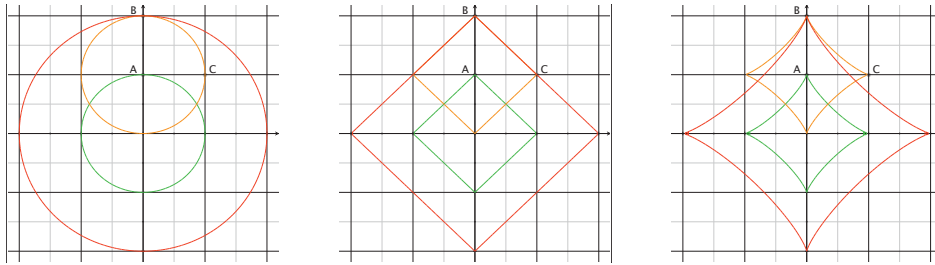
$$\text{Dis}_0(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d (x_j - y_j)^0 = \sum_{j=1}^d I[x_j \neq y_j]$$

under the understanding that  $x^0 = 0$  for  $x = 0$  and 1 otherwise. This is actually the distance experienced by a Rook on the chessboard: if both rank and file are different the square is two moves away, if only one of them is different the square is one move away. If  $\mathbf{x}$  and  $\mathbf{y}$  are binary strings, this is also called the **Hamming distance**. Alternatively, we can see the Hamming distance as the number of bits that need to be flipped to change  $\mathbf{x}$  into  $\mathbf{y}$ ; for non-binary strings of unequal length this can be generalised to the notion of **edit distance** or **Levenshtein distance**.

Do all of these mathematical constructs make sense as a notion of distance? In order to answer that question we can draw up a list of properties that a proper distance measure should have, such as non-negativity and symmetry. The generally agreed-upon list defines what is known as a metric.


**Definition 8.2 (Distance metric).** Given an instance space  $\mathcal{X}$ , a **distance metric** is a function  $\text{Dis} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  such that for any  $x, y, z \in \mathcal{X}$ :

1. distances between a point and itself are zero:  $\text{Dis}(x, x) = 0$ ;



**Figure 8.4.** (left) The green circle connects points the same Euclidean distance (i.e., Minkowski distance of order  $p = 2$ ) away from the origin as A. The orange circle shows that B and C are equidistant from A. The red circle demonstrates that C is closer to the origin than B, which conforms to the triangle inequality. (middle) With Manhattan distance ( $p = 1$ ), B and C are equally close to the origin and also equidistant from A. (right) With  $p < 1$  (here,  $p = 0.8$ ) C is further away from the origin than B; since both are again equidistant from A, it follows that travelling from the origin to C via A is quicker than going there directly, which violates the triangle inequality.

2. all other distances are larger than zero: if  $x \neq y$  then  $\text{Dis}(x, y) > 0$ ;
3. distances are symmetric:  $\text{Dis}(y, x) = \text{Dis}(x, y)$ ;
4. detours can not shorten the distance:  $\text{Dis}(x, z) \leq \text{Dis}(x, y) + \text{Dis}(y, z)$ .

If the second condition is weakened to a non-strict inequality – i.e.,  $\text{Dis}(x, y)$  may be zero even if  $x \neq y$  – the function  $\text{Dis}$  is called a **pseudo-metric**. 

The last condition is called the **triangle inequality** (or sub-additivity, as it really concerns the interaction between distance and addition). Figure 8.4 investigates this for Minkowski distances of various orders. The triangle inequality dictates that the distance from the origin to C is no more than the sum of the distances from the origin to A ( $\text{Dis}(O, A)$ ) and from A to C ( $\text{Dis}(A, C)$ ). B is at the same distance from A as C, regardless of the distance measure used; so  $\text{Dis}(O, A) + \text{Dis}(A, C)$  is equal to the distance from the origin to B. So, if we draw a circle around the origin through B, the triangle inequality dictates that C not be outside that circle. As we see in the left figure for Euclidean distance, B is the only point where the circles around the origin and around A intersect, so everywhere else the triangle inequality is a strict inequality.

The middle figure shows the same situation for Manhattan distance ( $p = 1$ ). Now, B and C are in fact equidistant from the origin, and so travelling via A to C is no longer a detour, but just one of the many shortest routes. However, if we now decrease  $p$  further, we see that C ends up outside the red shape, and is thus further away than B when seen from the origin, whereas of course the sum of the distances from the origin to A and from A to C is still equal to the distance from the origin to B. At this point, our intuition breaks down: Minkowski distances with  $p < 1$  are simply not very useful as distances since they all violate the triangle inequality.

Sometimes it is useful to use different scales for different coordinates if they are traversed with different speeds. For instance, for people horizontal distances can be traversed more easily than vertical differences, and consequently it is more realistic to use an ellipse rather than a circle to identify points that can be reached in a fixed amount of time, with the major axis of the ellipse indicating directions that can be traversed at larger speed. The ellipse can also be rotated, so that the major axis is not aligned with any of the coordinates: for instance, this could be the direction of a motorway, or the wind direction. Mathematically, while hyper-spheres (circles in  $d \geq 2$  dimensions) of radius  $r$  can be defined by the equation  $\mathbf{x}^T \mathbf{x} = r^2$ , hyper-ellipses are defined by  $\mathbf{x}^T \mathbf{M} \mathbf{x} = r^2$ , where  $\mathbf{M}$  is a matrix describing the appropriate rotation and scaling.

**Example 8.1 (Elliptical distance).** Consider the following matrices

$$\mathbf{R} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \quad \mathbf{S} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{M} = \begin{pmatrix} 5/8 & -3/8 \\ -3/8 & 5/8 \end{pmatrix}$$

The matrix  $\mathbf{R}$  describes a clockwise rotation of 45 degrees, and the diagonal matrix  $\mathbf{S}$  scales the  $x$ -axis by a factor 1/2. The equation

$$(\mathbf{S}\mathbf{R}\mathbf{x})^T (\mathbf{S}\mathbf{R}\mathbf{x}) = \mathbf{x}^T \mathbf{R}^T \mathbf{S}^T \mathbf{S} \mathbf{R} \mathbf{x} = \mathbf{x}^T \mathbf{R}^T \mathbf{S}^2 \mathbf{R} \mathbf{x} = \mathbf{x}^T \mathbf{M} \mathbf{x} = 1/4$$

describes a shape which, after clockwise rotation of 45 degrees and scaling of the  $x$ -axis by a factor 1/2, is a circle with radius 1/2 – i.e., the ‘ascending’ ellipse in Figure 8.3 (right). The ellipse equation is  $(5/8)x^2 + (5/8)y^2 - (3/4)xy = 1/2$ .

Often, the shape of the ellipse is estimated from data as the inverse of the covariance matrix:  $\mathbf{M} = \boldsymbol{\Sigma}^{-1}$ . This leads to the definition of the *Mahalanobis distance*

$$\text{Dis}_M(\mathbf{x}, \mathbf{y} | \boldsymbol{\Sigma}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})} \quad (8.1)$$

Using the covariance matrix in this way has the effect of decorrelating and normalising the features, as we saw in Section 7.1. Clearly, Euclidean distance is a special case of Mahalanobis distance with the identity matrix  $\mathbf{I}$  as covariance matrix:  $\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \text{Dis}_M(\mathbf{x}, \mathbf{y} | \mathbf{I})$ .

## 8.2 Neighbours and exemplars

Now that we understand the basics of measuring distance in instance space, we proceed to consider the key ideas underlying distance-based models. The two most

important of these are: formulating the model in terms of a number of prototypical instances or *exemplars*, and defining the decision rule in terms of the nearest exemplars or *neighbours*. We can understand these concepts by revisiting our old friend, the basic linear classifier. This classifier uses the two class means  $\mu^+$  and  $\mu^-$  as exemplars, as a summary of all we need to know about the training data in order to build the classifier. A fundamental property of the mean of a set of vectors is that it minimises the sum of squared Euclidean distances to those vectors.

**Theorem 8.1 (The arithmetic mean minimises squared Euclidean distance).** *The arithmetic mean  $\mu$  of a set of data points  $D$  in a Euclidean space is the unique point that minimises the sum of squared Euclidean distances to those data points.*

*Proof.* We will show that  $\arg \min_{\mathbf{y}} \sum_{\mathbf{x} \in D} \|\mathbf{x} - \mathbf{y}\|^2 = \mu$ , where  $\|\cdot\|$  denotes the 2-norm. We find this minimum by taking the gradient (the vector of partial derivatives with respect to  $y_i$ ) of the sum and setting it to the zero vector:

$$\nabla_{\mathbf{y}} \sum_{\mathbf{x} \in D} \|\mathbf{x} - \mathbf{y}\|^2 = -2 \sum_{\mathbf{x} \in D} (\mathbf{x} - \mathbf{y}) = -2 \sum_{\mathbf{x} \in D} \mathbf{x} + 2|D|\mathbf{y} = \mathbf{0}$$

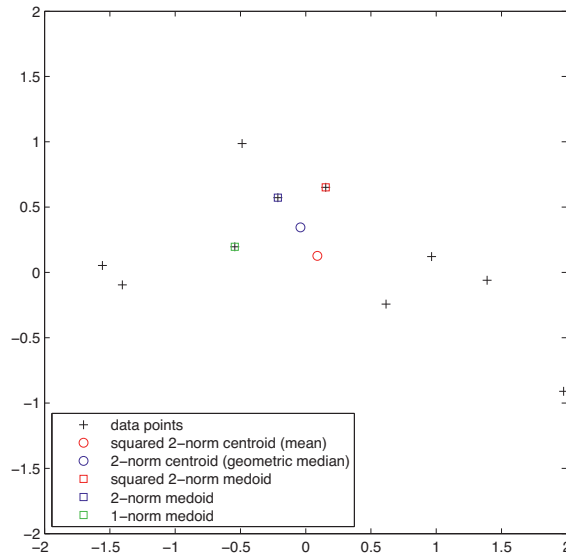
from which we derive  $\mathbf{y} = \frac{1}{|D|} \sum_{\mathbf{x} \in D} \mathbf{x} = \mu$ . 

Notice that minimising the sum of squared Euclidean distances of a given set of points is the same as minimising the *average* squared Euclidean distance. You may wonder what happens if we drop the square here: wouldn't it be more natural to take the point that minimises total Euclidean distance as exemplar? This point is known as the *geometric median*, as for univariate data it corresponds to the median or 'middle value' of a set of numbers. However, for multivariate data there is no closed-form expression for the geometric median, which needs to be calculated by successive approximation. This computational advantage is the main reason why distance-based methods tend to use squared Euclidean distance.

In certain situations it makes sense to restrict an exemplar to be one of the given data points. In that case, we speak of a *medoid*, to distinguish it from a *centroid* which is an exemplar that doesn't have to occur in the data. Finding a medoid requires us to calculate, for each data point, the total distance to all other data points, in order to choose the point that minimises it. Regardless of the distance metric used, this is an  $O(n^2)$  operation for  $n$  points, so for medoids there is no computational reason to prefer one distance metric over another. Figure 8.5 shows a set of 10 data points where the different ways of determining exemplars all give different results. In particular, the mean and squared 2-norm medoid can be overly sensitive to outliers.

Once we have determined the exemplars, the basic linear classifier constructs the decision boundary as the perpendicular bisector of the line segment connecting the two exemplars. An alternative, distance-based way to classify instances without direct

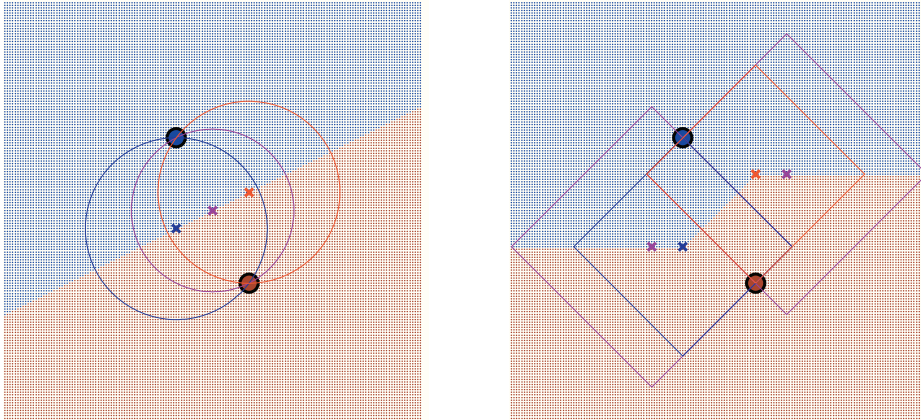




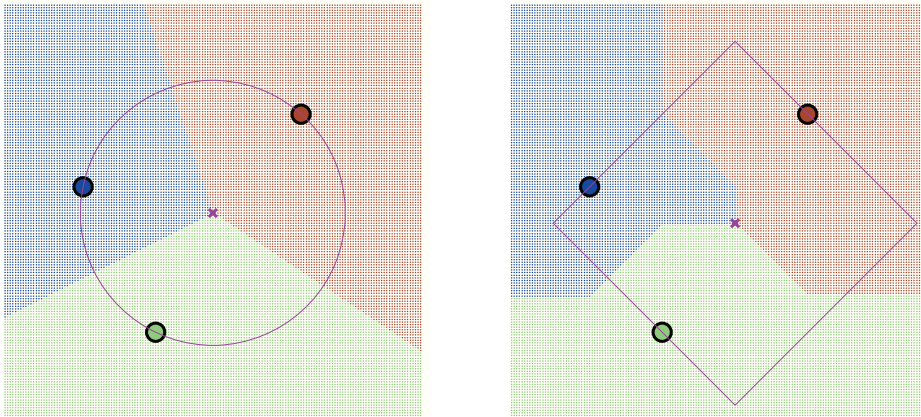
**Figure 8.5.** A small data set of 10 points, with circles indicating centroids and squares indicating medoids (the latter must be data points), for different distance metrics. Notice how the outlier on the bottom-right ‘pulls’ the mean away from the geometric median; as a result the corresponding medoid changes as well.

reference to a decision boundary is by the following decision rule: if  $\mathbf{x}$  is nearest to  $\mu^{\oplus}$  then classify it as positive, otherwise as negative; or equivalently, classify an instance to the class of the *nearest* exemplar. If we use Euclidean distance as our closeness measure, simple geometry tells us we get exactly the same decision boundary (Figure 8.6 (left)).

So *the basic linear classifier can be interpreted from a distance-based perspective as constructing exemplars that minimise squared Euclidean distance within each class, and then applying a nearest-exemplar decision rule*. This change of perspective opens up many new possibilities. For example, we can investigate what the decision boundary looks like if we use Manhattan distance for the decision rule (Figure 8.6 (right)). It turns out that the decision boundary can only run along a number of fixed angles: in two dimensions these are horizontal, vertical and at (plus or minus) 45 degrees. This can be understood as follows. Suppose the two exemplars have different  $x$ - and  $y$ -coordinates, then they span a rectangle (I’ll assume a tall rectangle, as in the figure). Imagine yourself in the centre of that rectangle, then clearly you are at equal distances from both exemplars (in fact, that same point is part of the 2-norm decision boundary). Now, imagine that you move one horizontal step, then you will move closer to one exemplar and away from the other; in order to compensate for that, you will also need to make a vertical step. So, within the rectangle, you maintain equal distance

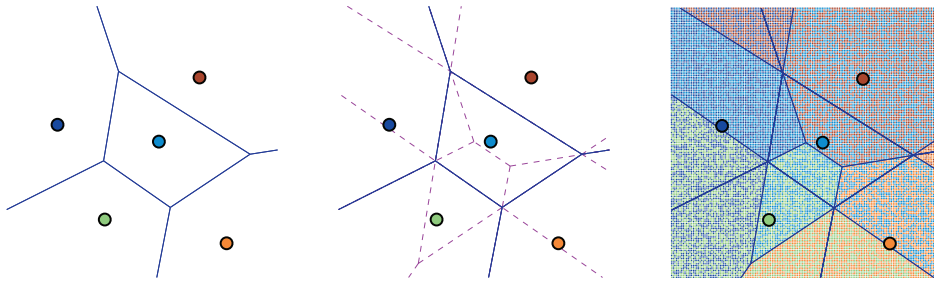


**Figure 8.6. (left)** For two exemplars the nearest-exemplar decision rule with Euclidean distance results in a linear decision boundary coinciding with the perpendicular bisector of the line connecting the two exemplars. The crosses denote different locations on the decision boundary, and the circles centred at those locations demonstrate that the exemplars are equidistant from each of them. When travelling along the decision boundary from bottom-left to top-right, these circles first shrink then grow again after passing the location halfway between the two exemplars. **(right)** Using Manhattan distance the circles are replaced by diamonds. Travelling from left to right, the diamonds shrink along the left-most horizontal segment of the decision boundary, then stay the same size along the 45-degree segment, and then grow again along the right-most horizontal segment.



**Figure 8.7. (left)** Decision regions defined by the 2-norm nearest-exemplar decision rule for three exemplars. **(right)** With Manhattan distance the decision regions become non-convex.

from the exemplars by moving at a 45 degree angle. Once you reach the perimeter of the rectangle you will walk away from both exemplars by making horizontal steps, so from there the decision boundary runs horizontally.



**Figure 8.8.** (left) Voronoi tessellation for five exemplars. (middle) Taking the two nearest exemplars into account leads to a further subdivision of each Voronoi cell. (right) The shading indicates which exemplars contribute to which cell.

Another useful consequence of switching to the distance-based perspective is that the nearest-exemplar decision rule works equally well for more than two exemplars, which gives us a multi-class version of the basic linear classifier.<sup>1</sup> Figure 8.7 (left) illustrates this for three exemplars. Each decision region is now bounded by two line segments. As you would expect, the 2-norm decision boundaries are more regular than the 1-norm ones: mathematicians say that the 2-norm decision regions are *convex*, which means that linear interpolation between any two points in the region can never go outside it. Clearly, this doesn't hold for 1-norm decision regions (Figure 8.7 (right)).

Increasing the number of exemplars further means that some of the regions become closed convex 'cells' (we are assuming Euclidean distance for the remainder of this section), giving rise to what is known as a *Voronoi tessellation*. Since the number of classes is typically much lower than the number of exemplars, decision rules often take more than one nearest exemplar into account. This increases the number of decision regions further.

**Example 8.2 (Two neighbours know more than one).** Figure 8.8 (left) gives a Voronoi tessellation for five exemplars. Each line segment is part of the perpendicular bisector of two exemplars. There are  $\binom{5}{2} = 10$  pairs of exemplars, but two of these pairs are too far away from each other so we observe only eight line segments in the Voronoi tessellation.

If we now also take the second-nearest exemplars into account, each Voronoi cell is further subdivided: for instance, since the central point has four neighbours, the central cell is divided into four subregions (Figure 8.8 (middle)). You can think of those additional line segments as being part of the Voronoi tessela-

<sup>1</sup> In information retrieval this is often called the *Rocchio classifier*.

tion that results when the central point is removed. The other exemplars have only three immediate neighbours and so their cells are divided into three subregions. We thus obtain 16 ‘2-nearest exemplar’ decision regions, each of which is defined by a different pair of nearest and second-nearest exemplars.

Figure 8.8 (right) shades each of these regions according to the two nearest exemplars spanning it. Notice that we gave each of the two exemplars the same weight, and so there are pairs of adjacent regions (across each of the original Voronoi boundaries) receiving the same shading, resulting in eight different shadings in all. This will be relevant later on, when we discuss the refinement of nearest-neighbour classifiers.

To summarise, the main ingredients of distance-based models are

- ☞ distance metrics, which can be Euclidean, Manhattan, Minkowski or Mahalanobis, among many others;
- ☞ exemplars: centroids that find a centre of mass according to a chosen distance metric, or medoids that find the most centrally located data point; and
- ☞ distance-based decision rules, which take a vote among the  $k$  nearest exemplars.

In the next sections these ingredients are combined in various ways to obtain supervised and unsupervised learning algorithms.

### 8.3 Nearest-neighbour classification

In the previous section we saw how to generalise the basic linear classifier to more than two classes, by learning an exemplar for each class and using the nearest-exemplar decision rule to classify new data. In fact, the most commonly used distance-based classifier is even more straightforward than that: it simply uses each training instance as an exemplar. Consequently, ‘training’ this classifier requires nothing more than memorising the training data. This extremely simple classifier is known as the *nearest-neighbour classifier*. Its decision regions are made up of the cells of a Voronoi tessellation, with piecewise linear decision boundaries selected from the Voronoi boundaries (since adjacent cells may be labelled with the same class).

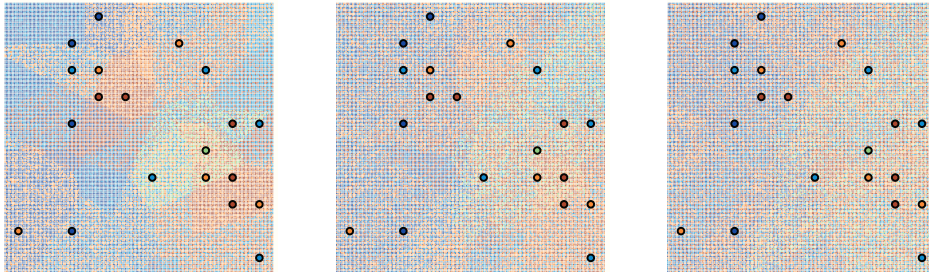
What are the properties of the nearest-neighbour classifier? First, notice that, unless the training set contains identical instances from different classes, we will be able to separate the classes perfectly on the training set – not really a surprise, as we memorised all training examples! Furthermore, by choosing the right exemplars we can more or less represent any decision boundary, or at least an arbitrarily close piecewise linear

approximation. It follows that the nearest-neighbour classifier has low bias, but also high variance: move any of the exemplars spanning part of the decision boundary, and you will also change the boundary. This suggests a risk of overfitting if the training data is limited, noisy or unrepresentative.

From an algorithmic point of view, training the nearest-neighbour classifier is very fast, taking only  $O(n)$  time for storing  $n$  exemplars. The downside is that classifying a single instance also takes  $O(n)$  time, as the instance will need to be compared with every exemplar to determine which one is the nearest. It is possible to reduce classification time at the expense of increased training time by storing the exemplars in a more elaborate data structure, but this tends not to scale well to large numbers of features.

In fact, high-dimensional instance spaces can be problematic for another reason: the infamous *curse of dimensionality*. High-dimensional spaces tend to be extremely sparse, which means that every point is far away from virtually every other point, and hence pairwise distances tend to be uninformative. However, whether or not you are hit by the curse of dimensionality is not simply a matter of counting the number of features, as there are several reasons why the effective dimensionality of the instance space may be much smaller than the number of features. For example, some of the features may be irrelevant and drown out the relevant features' signal in the distance calculations. In such a case it would be a good idea, before building a distance-based model, to reduce dimensionality by performing *feature selection*, as will be discussed in Chapter 10. Alternatively, the data may live on a *manifold* of lower dimension than the instance space (e.g., the surface of a sphere is a two-dimensional manifold wrapped around a three-dimensional object), which allows other dimensionality-reduction techniques such as *principal component analysis*, which will be explained in the same chapter. In any case, before applying nearest-neighbour classification it is a good idea to plot a histogram of pairwise distances of a sample to see if they are sufficiently varied.

Notice that the nearest-neighbour method can easily be applied to regression problems with a real-valued target variable. In fact, the method is completely oblivious to the type of target variable and can be used to output text documents, images and videos. It is also possible to output the exemplar itself instead of a separate target, in which case we usually speak of *nearest-neighbour retrieval*. Of course we can only output targets (or exemplars) stored in the exemplar database, but if we have a way of aggregating these we can go beyond this restriction by applying the *k-nearest neighbour* method. In its simplest form, the *k*-nearest neighbour classifier takes a vote between the  $k \geq 1$  nearest exemplars of the instance to be classified, and predicts the majority class. We can easily turn this into a probability estimator by returning the normalised class counts as a probability distribution over classes.

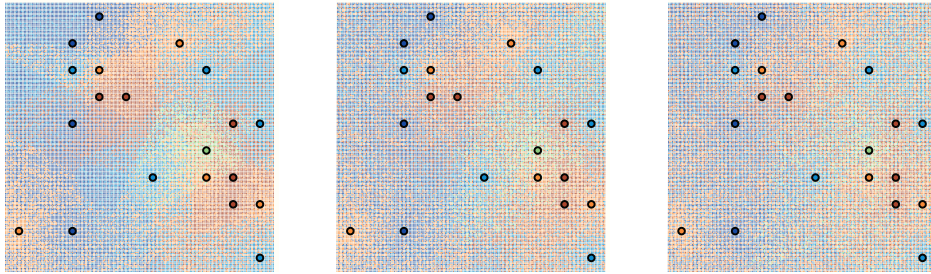


**Figure 8.9.** (left) Decision regions of a 3-nearest neighbour classifier; the shading represents the predicted probability distribution over the five classes. (middle) 5-nearest neighbour. (right) 7-nearest neighbour.

Figure 8.9 illustrates this on a small data set of 20 exemplars from five different classes, for  $k = 3, 5, 7$ . The class distribution is visualised by assigning each test point the class of a uniformly sampled neighbour: so, in a region where two of  $k = 3$  neighbours are red and one is orange, the shading is a mix of two-thirds red and one-third orange. While for  $k = 3$  the decision regions are still mostly discernible, this is much less so for  $k = 5$  and  $k = 7$ . This may seem at odds with our earlier demonstration of the increase in the number of decision regions with increasing  $k$  in Example 8.2. However, this increase is countered by the fact that the probability vectors become more similar to each other. To take an extreme example: if  $k$  is equal to the number of exemplars  $n$ , every test instance will have the same number of neighbours and will receive the same probability vector which is equal to the prior distribution over the exemplars. If  $k = n - 1$  we can reduce one of the class counts by 1, which can be done in  $c$  ways: the same number of possibilities as with  $k = 1$ !

We conclude that the refinement of  $k$ -nearest neighbour – the number of different predictions it can make – initially increases with increasing  $k$ , then decreases again. Furthermore, we can say that the bias increases and the variance decreases with increasing  $k$ . There is no easy recipe to decide what value of  $k$  is appropriate for a given data set. However, it is possible to sidestep this question to some extent by applying *distance weighting* to the votes: that is, the closer an exemplar is to the instance to be classified, the more its vote counts. Figure 8.10 demonstrates this, using the reciprocal of the distance to an exemplar as the weight of its vote. This blurs the decision boundaries, as the model now applies a combination of grouping by means of the Voronoi boundaries, and grading by means of distance weighting. Furthermore, since the weights decrease quickly for larger distances, the effect of increasing  $k$  is much smaller than with unweighted voting. In fact, with distance weighting we can simply put  $k = n$  and still obtain a model that makes different predictions in different parts of the instance space. One could say that distance weighting makes  $k$ -nearest neighbour





**Figure 8.10.** (left) 3-nearest neighbour with distance weighting on the data from Figure 8.9. (middle) 5-nearest neighbour. (right) 7-nearest neighbour.

more of a global model, while without it (and for small  $k$ ) it is more like an aggregation of local models.

If  $k$ -nearest neighbour is used for regression problems, the obvious way to aggregate the predictions from the  $k$  neighbours is by taking the mean value, which can again be distance-weighted. This would lend the model additional predictive power by predicting values that aren't observed among the stored exemplars. More generally, we can apply  $k$ -means to any learning problem where we have an appropriate 'aggregator' for multiple target values.


## 8.4 Distance-based clustering

In a distance-based context, unsupervised learning is usually taken to refer to clustering, and we will now review a number of distance-based clustering methods. The ones considered in this section are all exemplar-based and hence predictive: they naturally generalise to unseen instances (see Section 3.3 for the distinction between predictive and descriptive clustering). In the next section we consider a clustering method that is not exemplar-based and hence descriptive.

Predictive distance-based clustering methods use the same ingredients as distance-based classifiers: a distance metric, a way to construct exemplars and a distance-based decision rule. In the absence of an explicit target variable, the assumption is that the distance metric indirectly encodes the learning target, so that we aim to find clusters that are *compact* with respect to the distance metric. This requires a notion of cluster compactness that can serve as our optimisation criterion. To that end, we refer back to the scatter matrix introduced in Background 7.2 on p.200.

**Definition 8.3 (Scatter).** Given a data matrix  $\mathbf{X}$ , the **scatter matrix** is the matrix

$$\mathbf{S} = (\mathbf{X} - \mathbf{1}\mu)^T (\mathbf{X} - \mathbf{1}\mu) = \sum_{i=1}^n (\mathbf{X}_i - \mu)^T (\mathbf{X}_i - \mu)$$

where  $\boldsymbol{\mu}$  is a row vector containing all column means of  $\mathbf{X}$ . The **scatter** of  $\mathbf{X}$  is defined as  $\text{Scat}(\mathbf{X}) = \sum_{i=1}^n \|\mathbf{X}_i - \boldsymbol{\mu}\|^2$ , which is equal to the trace of the scatter matrix (i.e., the sum of its diagonal elements). 

Imagine now that we partition  $D$  into  $K$  subsets  $D_1 \uplus \dots \uplus D_K = D$ , and let  $\boldsymbol{\mu}_j$  denote the mean of  $D_j$ . Let  $\mathbf{S}$  be the scatter matrix of  $D$ , and  $\mathbf{S}_j$  be the scatter matrices of  $D_j$ . These scatter matrices then have the following relationship:

$$\mathbf{S} = \sum_{j=1}^K \mathbf{S}_j + \mathbf{B} \quad (8.2)$$

Here,  $\mathbf{B}$  is the scatter matrix that results by replacing each point in  $D$  with the corresponding  $\boldsymbol{\mu}_j$ . Each  $\mathbf{S}_j$  is called a **within-cluster scatter matrix** and describes the compactness of the  $j$ -th cluster.  $\mathbf{B}$  is the **between-cluster scatter matrix** and describes the spread of the cluster centroids. It follows that the traces of these matrices can be decomposed similarly, which gives

$$\text{Scat}(D) = \sum_{j=1}^K \text{Scat}(D_j) + \sum_{j=1}^K |D_j| \|\boldsymbol{\mu}_j - \boldsymbol{\mu}\|^2 \quad (8.3)$$

What this tells us is that minimising the total scatter over all clusters is equivalent to maximising the (weighted) scatter of the centroids. The  **$K$ -means problem** is to find a partition that minimises the total within-cluster scatter.

**Example 8.3 (Reducing scatter by partitioning data).** Consider the following five points:  $(0, 3)$ ,  $(3, 3)$ ,  $(3, 0)$ ,  $(-2, -4)$  and  $(-4, -2)$ . These points are, conveniently, centred around  $(0, 0)$ . The scatter matrix is

$$\mathbf{S} = \begin{pmatrix} 0 & 3 & 3 & -2 & -4 \\ 3 & 3 & 0 & -4 & -2 \end{pmatrix} \begin{pmatrix} 0 & 3 \\ 3 & 3 \\ 3 & 0 \\ -2 & -4 \\ -4 & -2 \end{pmatrix} = \begin{pmatrix} 38 & 25 \\ 25 & 38 \end{pmatrix}$$

with trace  $\text{Scat}(D) = 76$ . If we cluster the first two points together in one cluster and the remaining three in another, then we obtain cluster means  $\boldsymbol{\mu}_1 = (1.5, 3)$  and  $\boldsymbol{\mu}_2 = (-1, -2)$  and within-cluster scatter matrices

$$\mathbf{S}_1 = \begin{pmatrix} 0-1.5 & 3-1.5 \\ 3-3 & 3-3 \end{pmatrix} \begin{pmatrix} 0-1.5 & 3-3 \\ 3-1.5 & 3-3 \end{pmatrix} = \begin{pmatrix} 4.5 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\mathbf{S}_2 = \begin{pmatrix} 3-(-1) & -2-(-1) & -4-(-1) \\ 0-(-2) & -4-(-2) & -2-(-2) \end{pmatrix} \begin{pmatrix} 3-(-1) & 0-(-2) \\ -2-(-1) & -4-(-2) \\ -4-(-1) & -2-(-2) \end{pmatrix} = \begin{pmatrix} 26 & 10 \\ 10 & 8 \end{pmatrix}$$



with traces  $\text{Scat}(D_1) = 4.5$  and  $\text{Scat}(D_2) = 34$ . Two copies of  $\mu_1$  and three copies of  $\mu_2$  have, by definition, the same centre of gravity as the complete data set:  $(0, 0)$  in this case. We thus calculate the between-cluster scatter matrix as

$$\mathbf{B} = \begin{pmatrix} 1.5 & 1.5 & -1 & -1 & -1 \\ 3 & 3 & -2 & -2 & -2 \end{pmatrix} \begin{pmatrix} 1.5 & 3 \\ 1.5 & 3 \\ -1 & -2 \\ -1 & -2 \\ -1 & -2 \end{pmatrix} = \begin{pmatrix} 7.5 & 15 \\ 15 & 30 \end{pmatrix}$$

with trace 37.5.

Alternatively, if we treat the first three points as a cluster and put the other two in a second cluster, then we obtain cluster means  $\mu'_1 = (2, 2)$  and  $\mu'_2 = (-3, -3)$ , and within-cluster scatter matrices

$$\mathbf{S}'_1 = \begin{pmatrix} 0-2 & 3-2 & 3-2 \\ 3-2 & 3-2 & 0-2 \end{pmatrix} \begin{pmatrix} 0-2 & 3-2 \\ 3-2 & 3-2 \\ 3-2 & 0-2 \end{pmatrix} = \begin{pmatrix} 6 & -3 \\ -3 & 6 \end{pmatrix}$$

$$\mathbf{S}'_2 = \begin{pmatrix} -2-(-3) & -4-(-3) \\ -4-(-3) & -2-(-3) \end{pmatrix} \begin{pmatrix} -2-(-3) & -4-(-3) \\ -4-(-3) & -2-(-3) \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}$$

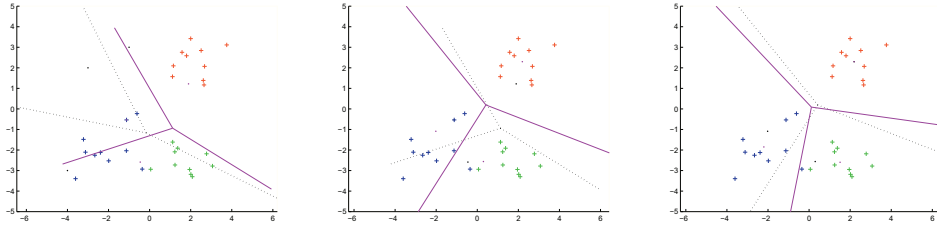
with traces  $\text{Scat}(D'_1) = 12$  and  $\text{Scat}(D'_2) = 4$ . The between-cluster scatter matrix is

$$\mathbf{B}' = \begin{pmatrix} 2 & 2 & 2 & -3 & -3 \\ 2 & 2 & 2 & -3 & -3 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \\ -3 & -3 \\ -3 & -3 \end{pmatrix} = \begin{pmatrix} 30 & 30 \\ 30 & 30 \end{pmatrix}$$

with trace 60. Clearly, the second clustering produces tighter clusters whose centroids are further apart.

### *K-means algorithm*

The  $K$ -means problem is NP-complete, which means that there is no efficient solution to find the global minimum and we need to resort to a heuristic algorithm. The best-known algorithm is usually also called  $K$ -means, although the name 'Lloyd's algorithm' is also used. The outline of the algorithm is given in [Algorithm 8.1](#). The algorithm iterates between partitioning the data using the nearest-centroid decision rule, and



**Figure 8.11.** (left) First iteration of 3-means on Gaussian mixture data. The dotted lines are the Voronoi boundaries resulting from randomly initialised centroids; the **violet solid lines** are the result of the recalculated means. (middle) Second iteration, taking the previous partition as starting point (dotted line). (right) Third iteration with stable clustering.

recalculating centroids from a partition. Figure 8.11 demonstrates the algorithm on a small data set with three clusters, and Example 8.4 gives the result on our example data set describing properties of different machine learning methods.

**Example 8.4 (Clustering MLM data).** Refer back to the MLM data set in Table 1.4 on p.39 (it is also helpful to look at its two-dimensional approximation in Figure 1.7 on p.37). When we run  $K$ -means on this data with  $K = 3$ , we obtain the clusters {Associations, Trees, Rules}, {GMM, naive Bayes}, and a larger cluster with the remaining data points. When we run it with  $K = 4$ , we get that the large cluster splits into two: {kNN, Linear Classifier, Linear Regression} and

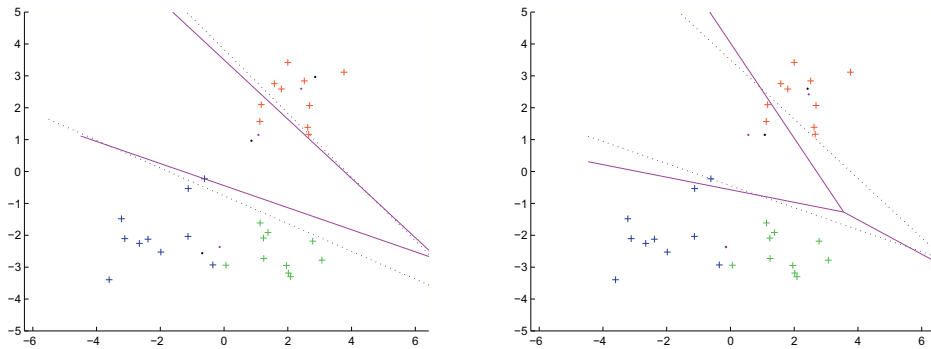
---

**Algorithm 8.1:**  $KMeans(D, K)$  –  $K$ -means clustering using Euclidean distance  $Dis_2$ .

---

**Input** : data  $D \subseteq \mathbb{R}^d$ ; number of clusters  $K \in \mathbb{N}$ .  
**Output** :  $K$  cluster means  $\mu_1, \dots, \mu_K \in \mathbb{R}^d$ .  
1 randomly initialise  $K$  vectors  $\mu_1, \dots, \mu_K \in \mathbb{R}^d$ ;  
2 **repeat**  
3     assign each  $\mathbf{x} \in D$  to  $\arg \min_j Dis_2(\mathbf{x}, \mu_j)$ ;  
4     **for**  $j = 1$  to  $K$  **do**  
5          $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;  
6          $\mu_j = \frac{1}{|D_j|} \sum_{\mathbf{x} \in D_j} \mathbf{x}$ ;  
7     **end**  
8 **until** no change in  $\mu_1, \dots, \mu_K$ ;  
9 **return**  $\mu_1, \dots, \mu_K$ ;

---



**Figure 8.12.** (left) First iteration of 3-means on the same data as Figure 8.11 with differently initialised centroids. (right) 3-means has converged to a sub-optimal clustering.

{Kmeans, Logistic Regression, SVM}; but also that GMM gets reallocated to the latter cluster, and naive Bayes ends up as a singleton.

It can be shown that one iteration of  $K$ -means can never increase the within-cluster scatter, from which it follows that the algorithm will reach a *stationary point*: a point where no further improvement is possible. It is worth noting that even the simplest data set will have many stationary points.

**Example 8.5 (Stationary points in clustering).** Consider the task of dividing the set of numbers {8, 44, 50, 58, 84} into two clusters. There are four possible partitions that 2-means can find: {8}, {44, 50, 58, 84}; {8, 44}, {50, 58, 84}; {8, 44, 50}, {58, 84}; and {8, 44, 50, 58}, {84}. It is easy to verify that each of these establishes a stationary point for 2-means, and hence will be found with a suitable initialisation. Only the first clustering is optimal; i.e., it minimises the total within-cluster scatter.

In general, while  $K$ -means converges to a stationary point in finite time, no guarantees can be given about whether the convergence point is in fact the global minimum, or if not, how far we are from it. Figure 8.12 shows how an unfortunate initialisation of the centroids can lead to a sub-optimal solution. In practice it is advisable to run the algorithm a number of times and select the solution with the smallest within-cluster scatter.

### Clustering around medoids

It is straightforward to adapt the  $K$ -means algorithm to use a different distance metric; note that this will also change the objective function being minimised. [Algorithm 8.2](#) gives the  *$K$ -medoids* algorithm, which additionally requires the exemplars to be data points. Notice that calculating the medoid of a cluster requires examining all pairs of points – whereas calculating the mean requires just a single pass through the points – which can be prohibitive for large data sets. [Algorithm 8.3](#) gives an alternative algorithm called *partitioning around medoids (PAM)* that tries to improve a clustering locally by swapping medoids with other data points. The quality of a clustering  $Q$  is calculated as the total distance over all points to their nearest medoid. Notice that there are  $k(n - k)$  pairs of one medoid and one non-medoid, and evaluating  $Q$  requires iterating over  $n - k$  data points, so the computational cost of one iteration is quadratic in the number of data points. For large data sets one can run **PAM** on a small sample but evaluate  $Q$  on the whole data set, and repeat this a number of times for different samples.

An important limitation of the clustering methods discussed in this section is that they represent clusters only by means of exemplars. This disregards the shape of the clusters, and sometimes leads to counter-intuitive results. The two data sets in [Figure 8.13](#) are identical, except for a rescaling of the  $y$ -axis. Nevertheless,  $K$ -means finds entirely different clusterings. This is not actually a shortcoming of the  $K$ -means algorithm as such, as in [Figure 8.13 \(right\)](#) the two centroids are further away than in the intended solution, and hence this represents a better solution in terms of [Equation](#)

---

**Algorithm 8.2:**  $KMedoids(D, K, Dis)$  –  $K$ -medoids clustering using arbitrary distance metric  $Dis$ .

---

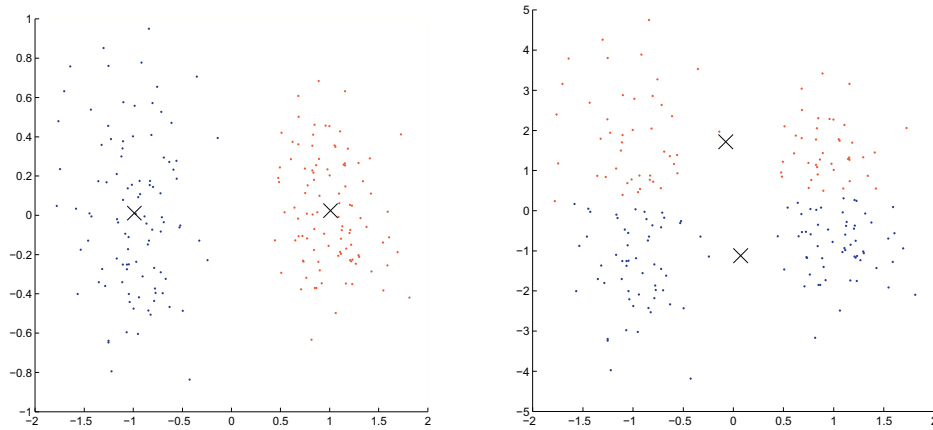
**Input** : data  $D \subseteq \mathcal{X}$ ; number of clusters  $K \in \mathbb{N}$ ;  
distance metric  $Dis : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ .

**Output** :  $K$  medoids  $\mu_1, \dots, \mu_K \in D$ , representing a predictive clustering of  $\mathcal{X}$ .

```

1 randomly pick  $K$  data points  $\mu_1, \dots, \mu_K \in D$ ;
2 repeat
3   assign each  $\mathbf{x} \in D$  to  $\arg \min_j Dis(\mathbf{x}, \mu_j)$ ;
4   for  $j = 1$  to  $k$  do
5      $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;
6      $\mu_j = \arg \min_{\mathbf{x} \in D_j} \sum_{\mathbf{x}' \in D_j} Dis(\mathbf{x}, \mathbf{x}')$ ;
7   end
8 until no change in  $\mu_1, \dots, \mu_K$ ;
9 return  $\mu_1, \dots, \mu_K$ ;
```

---



**Figure 8.13.** (left) On this data 2-means detects the right clusters. (right) After rescaling the y-axis, this configuration has a higher between-cluster scatter than the intended one.

8.3. The real issue is that in this case we want to estimate the ‘shape’ of the clusters as well as the cluster centroids, and hence take account of more than just the trace of the scatter matrices. We will discuss this further in the next chapter.

---

**Algorithm 8.3:**  $\text{PAM}(D, K, \text{Dis})$  – Partitioning around medoids clustering using arbitrary distance metric  $\text{Dis}$ .

---

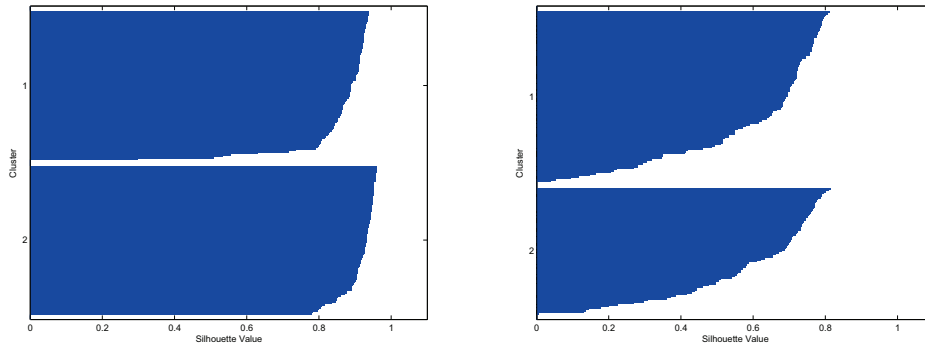
**Input** : data  $D \subseteq \mathcal{X}$ ; number of clusters  $K \in \mathbb{N}$ ;  
distance metric  $\text{Dis} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ .

**Output** :  $K$  medoids  $\mu_1, \dots, \mu_K \in D$ , representing a predictive clustering of  $\mathcal{X}$ .

```

1 randomly pick  $K$  data points  $\mu_1, \dots, \mu_K \in D$ ;
2 repeat
3   assign each  $\mathbf{x} \in D$  to  $\arg \min_j \text{Dis}(\mathbf{x}, \mu_j)$ ;
4   for  $j = 1$  to  $k$  do
5      $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;
6   end
7    $Q \leftarrow \sum_j \sum_{\mathbf{x} \in D_j} \text{Dis}(\mathbf{x}, \mu_j)$ ;
8   for each medoid  $\mathbf{m}$  and each non-medoid  $\mathbf{o}$  do
9     calculate the improvement in  $Q$  resulting from swapping  $\mathbf{m}$  with  $\mathbf{o}$ ;
10  end
11  select the pair with maximum improvement and swap;
12 until no further improvement possible;
13 return  $\mu_1, \dots, \mu_K$ ;
```

---



**Figure 8.14.** (left) Silhouette for the clustering in Figure 8.13 (left), using squared Euclidean distance. Almost all points have a high  $s(\mathbf{x})$ , which means that they are much closer, on average, to the other members of their cluster than to the members of the neighbouring cluster. (right) The silhouette for the clustering in Figure 8.13 (right) is much less convincing.

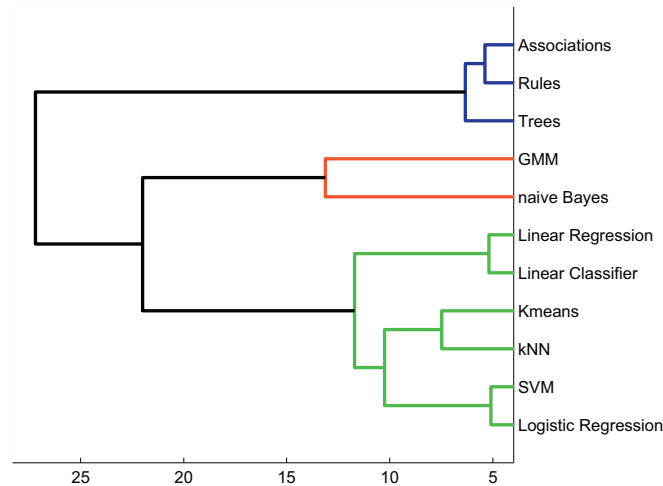
### Silhouettes

How could we detect the poor quality of the clustering in Figure 8.13 (right)? An interesting technique is the use of silhouettes. For any data point  $\mathbf{x}_i$ , let  $d(\mathbf{x}_i, D_j)$  denote the average distance of  $\mathbf{x}_i$  to the data points in cluster  $D_j$ , and let  $j(i)$  denote the index of the cluster that  $\mathbf{x}_i$  belongs to. Furthermore, let  $a(\mathbf{x}_i) = d(\mathbf{x}_i, D_{j(i)})$  be the average distance of  $\mathbf{x}_i$  to the points in its own cluster  $D_{j(i)}$ , and let  $b(\mathbf{x}_i) = \min_{k \neq j(i)} d(\mathbf{x}_i, D_k)$  be the average distance to the points in its neighbouring cluster. We would expect  $a(\mathbf{x}_i)$  to be considerably smaller than  $b(\mathbf{x}_i)$ , but this cannot be guaranteed. So we can take the difference  $b(\mathbf{x}_i) - a(\mathbf{x}_i)$  as an indication of how ‘well-clustered’  $\mathbf{x}_i$  is, and divide this by  $b(\mathbf{x}_i)$  to obtain a number less than or equal to 1.

It is, however, conceivable that  $a(\mathbf{x}_i) > b(\mathbf{x}_i)$ , in which case the difference  $b(\mathbf{x}_i) - a(\mathbf{x}_i)$  is negative. This describes the situation that, on average, the members of the neighbouring cluster are closer to  $\mathbf{x}_i$  than the members of its own cluster. In order to get a normalised value we divide by  $a(\mathbf{x}_i)$  in this case. This leads to the following definition:

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max(a(\mathbf{x}_i), b(\mathbf{x}_i))} \quad (8.4)$$

A *silhouette* then sorts and plots  $s(\mathbf{x})$  for each instance, grouped by cluster. Examples are shown in Figure 8.14 for the two clusterings in Figure 8.13. In this particular case we have used squared Euclidean distance in the construction of the silhouette, but the method can be applied to other distance metrics. We can clearly see that the first clustering is much better than the second. In addition to the graphical representation, we can compute average silhouette values per cluster and over the whole data set.



**Figure 8.15.** A dendrogram (printed left to right to improve readability) constructed by hierarchical clustering from the data in Table 1.4 on p.39.


## 8.5 Hierarchical clustering

The clustering methods discussed in the previous section use exemplars to represent a predictive clustering: a partition of the entire instance space. In this section we take a look at methods that represent clusters using trees. We previously encountered *clustering trees* in Section 5.3: those trees use features to navigate the instance space, similar to decision trees, and aren't distance-based as such. Here we consider trees called dendrograms, which are purely defined in terms of a distance measure. Because dendrograms use features only indirectly, as the basis on which the distance measure is calculated, they partition the given data rather than the entire instance space, and hence represent a descriptive clustering rather than a predictive one.


**Example 8.6 (Hierarchical clustering of MLM data).** We continue Example 8.4 on p.248. A hierarchical clustering of the MLM data is given in Figure 8.15. The tree shows that the three logical methods at the top form a strong cluster. If we wanted three clusters, we get the logical cluster, a second small cluster {GMM, naive Bayes}, and the remainder. If we wanted four clusters, we would separate GMM and naive Bayes, as the tree indicates this cluster is the least tight of the three (notice that this is slightly different from the solution found by 4-means). If we wanted five clusters, we would construct {Linear Regression, Linear Classifier} as a separate cluster. This illustrates the key

advantage of hierarchical clustering: it doesn't require fixing the number of clusters in advance.

A precise definition of a dendrogram is as follows.

**Definition 8.4 (Dendrogram).** Given a data set  $D$ , a **dendrogram** is a binary tree with the elements of  $D$  at its leaves. An internal node of the tree represents the subset of elements in the leaves of the subtree rooted at that node. The level of a node is the distance between the two clusters represented by the children of the node. Leaves have level 0. 

For this definition to work, we need a way to measure how close two clusters are. You might think that this is straightforward: just calculate the distance between the two cluster means. However, this occasionally leads to problems, as discussed later in this section. Furthermore, taking cluster means as exemplars assumes Euclidean distance, and we may want to use one of the other distance metrics discussed earlier. This has led to the introduction of the so-called linkage function, which is a general way to turn pairwise point distances into pairwise cluster distances.

**Definition 8.5 (Linkage function).** A **linkage function**  $L : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}$  calculates the distance between arbitrary subsets of the instance space, given a distance metric  $\text{Dis} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ . 

The most common linkage functions are as follows:

- Single linkage** defines the distance between two clusters as the *smallest* pairwise distance between elements from each cluster.
- Complete linkage** defines the distance between two clusters as the *largest* pointwise distance.
- Average linkage** defines the cluster distance as the *average* pointwise distance.
- Centroid linkage** defines the cluster distance as the point distance between the cluster means.

These linkage functions can be defined mathematically as follows:

$$\begin{aligned}
 L_{\text{single}}(A, B) &= \min_{x \in A, y \in B} \text{Dis}(x, y) \\
 L_{\text{complete}}(A, B) &= \max_{x \in A, y \in B} \text{Dis}(x, y) \\
 L_{\text{average}}(A, B) &= \frac{\sum_{x \in A, y \in B} \text{Dis}(x, y)}{|A| \cdot |B|} \\
 L_{\text{centroid}}(A, B) &= \text{Dis} \left( \frac{\sum_{x \in A} x}{|A|}, \frac{\sum_{y \in B} y}{|B|} \right)
 \end{aligned}$$



Clearly, all these linkage functions coincide for singleton clusters:  $L(\{x\}, \{y\}) = \text{Dis}(x, y)$ . However, for larger clusters they start to diverge. For example, suppose  $\text{Dis}(x, y) < \text{Dis}(x, z)$ , then the linkage between  $\{x\}$  and  $\{y, z\}$  is different in all four cases:

$$\begin{aligned} L_{\text{single}}(\{x\}, \{y, z\}) &= \text{Dis}(x, y) \\ L_{\text{complete}}(\{x\}, \{y, z\}) &= \text{Dis}(x, z) \\ L_{\text{average}}(\{x\}, \{y, z\}) &= (\text{Dis}(x, y) + \text{Dis}(x, z)) / 2 \\ L_{\text{centroid}}(\{x\}, \{y, z\}) &= \text{Dis}(x, (y + z) / 2) \end{aligned}$$

The general algorithm to build a dendrogram is given in Algorithm 8.4. The tree is built from the data points upwards and is hence a bottom-up or *agglomerative* algorithm. At each iteration the algorithm constructs a new partition of the data by merging the two nearest clusters together. In general, the HAC algorithm gives different results when different linkage functions are used. Single linkage is the easiest case to understand, as it effectively builds a graph by adding increasingly longer links between points, one at a time, such that ultimately there is a path between any pair of points (hence the term ‘linkage’). At any point during this process, the connected components are the clusters found at that iteration, and the linkage of the most recently found cluster is the length of the most recently added link. Hierarchical clustering using single linkage can essentially be done by calculating and sorting all pairwise distances between data points, which requires  $O(n^2)$  time for  $n$  points. The other linkage functions require at least  $O(n^2 \log n)$ . Notice that the unoptimised algorithm in Algorithm 8.4 has time complexity  $O(n^3)$ .

---

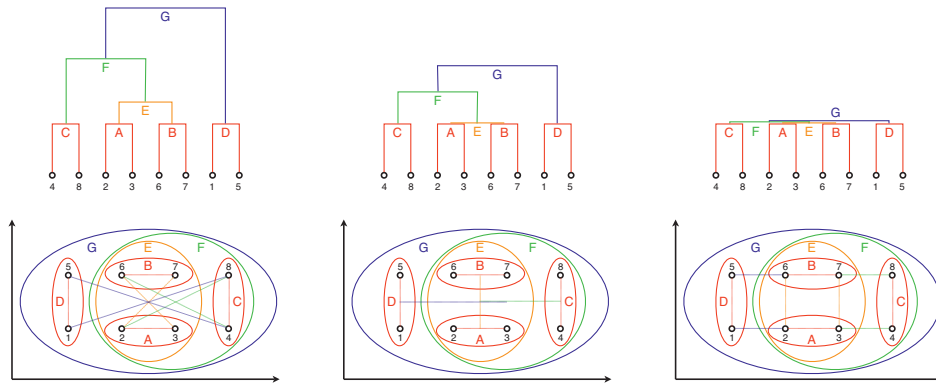
**Algorithm 8.4:** HAC( $D, L$ ) – Hierarchical agglomerative clustering.

---

**Input** : data  $D \subseteq \mathcal{X}$ ; linkage function  $L : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}$  defined in terms of distance metric.

**Output** : a dendrogram representing a descriptive clustering of  $D$ .

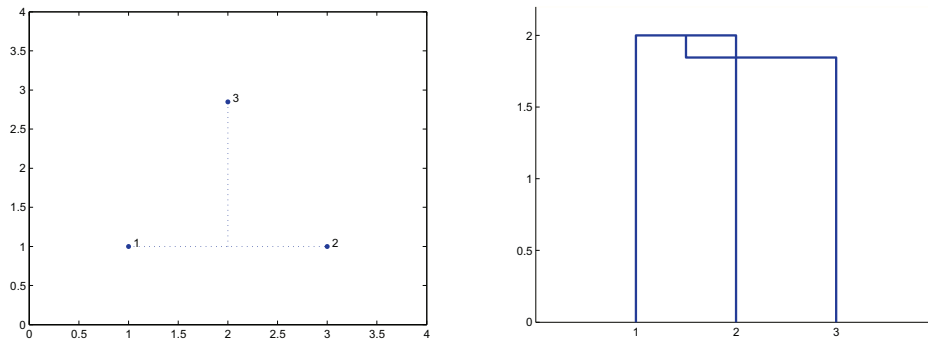
- 1 initialise clusters to singleton data points;
  - 2 create a leaf at level 0 for every singleton cluster;
  - 3 **repeat**
  - 4     find the pair of clusters  $X, Y$  with lowest linkage  $l$ , and merge;
  - 5     create a parent of  $X, Y$  at level  $l$ ;
  - 6 **until** all data points are in one cluster;
  - 7 **return** the constructed binary tree with linkage levels;
-



**Figure 8.16. (left)** Complete linkage defines cluster distance as the largest pairwise distance between elements from each cluster, indicated by the coloured lines between data points. The clustering found can be represented as nested partitions (bottom) or a dendrogram (top); the level of a horizontal connection between clusters in the dendrogram corresponds to the length of a linkage line. The example assumes that ties are broken by small irregularities in the grid. **(middle)** Centroid linkage defines the distance between clusters as the distance between their means. Notice that E obtains the same linkage as A and B, and so the latter clusters effectively disappear. **(right)** Single linkage defines the distance between clusters as the smallest pairwise distance. The dendrogram all but collapses, which means that no meaningful clusters are found in the given grid configuration.

**Example 8.7 (Linkage matters).** We consider a regular grid of 8 points in two rows of four (Figure 8.16). We assume that ties are broken by small irregularities. Each linkage function merges the same clusters in the same order, but the linkages are quite different in each case. Complete linkage gives the impression that D is far removed from the rest, whereas by moving D very slightly to the right it would have been added to E before C. With centroid linkage we see that E has in fact the same linkage as A and B, which means that A and B are not really discernible as separate clusters, even though they are found first. Single linkage seems preferable in this case, as it most clearly demonstrates that there is no meaningful cluster structure in this set of points.

Single and complete linkage both define the distance between clusters in terms of a particular pair of points. Consequently, they cannot take the shape of the cluster into account, which is why average and centroid linkage can offer an advantage. However, centroid linkage can lead to non-intuitive dendrograms, as illustrated in Figure

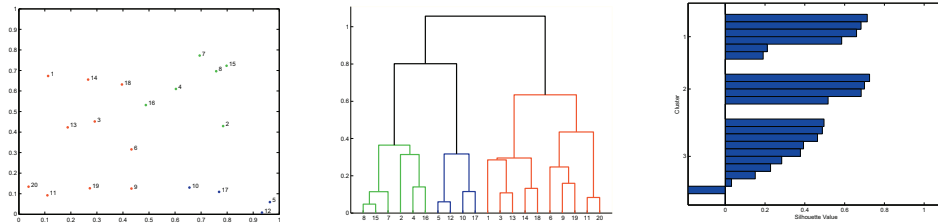


**Figure 8.17. (left)** Points 1 and 2 are closer to each other than to point 3. However, the distance between point 3 to the centroid of the other two points is less than any of the pairwise distances. **(right)** This results in a decrease in linkage when adding point 3 to cluster {1, 2}, and hence a non-monotonic dendrogram.

**8.17.** The issue here is that we have  $L(\{1\}, \{2\}) < L(\{1\}, \{3\})$  and  $L(\{1\}, \{2\}) < L(\{2\}, \{3\})$  but  $L(\{1\}, \{2\}) > L(\{1, 2\}, \{3\})$ . The first two inequalities mean that 1 and 2 are the first to be merged into a cluster; but the second inequality means that the level of cluster {1, 2, 3} in the dendrogram drops below the level of {1, 2}. Centroid linkage violates the requirement of *monotonicity*, which stipulates that  $L(A, B) < L(A, C)$  and  $L(A, B) < L(B, C)$  implies  $L(A, B) < L(A \cup B, C)$  for any clusters  $A$ ,  $B$  and  $C$ . The other three linkage functions are monotonic (the example also serves as an illustration why average linkage and centroid linkage are not the same).

Another thing to keep in mind when constructing dendrograms is that the hierarchical clustering method is deterministic and will always construct a clustering. Consider Figure 8.18, which shows a data set of 20 uniformly randomly sampled points. One would be hard-pressed to find any cluster structure in this data; yet a dendrogram constructed with complete linkage and Euclidean distance appears to indicate that there are three or four clearly discernible clusters. But if we look closer, we see that the linkage levels are very close together in the bottom of the tree, and the fact that linkages are higher towards the top comes primarily from the use of complete linkage, which concentrates on maximal pairwise distances. The silhouette in Figure 8.18 (right) confirms that the cluster structure is not very strong. Effectively, we are witnessing here a particular, clustering-related kind of overfitting, already familiar from other tree-based models discussed in Chapter 5. Furthermore, dendrograms – like other tree models – have high variance in that small changes in the data points can lead to large changes in the dendrogram.

In conclusion, hierarchical clustering methods have the distinct advantage that the number of clusters does not need to be fixed in advance. However, this advantage comes at considerable computational cost. Furthermore, we now need to choose not



**Figure 8.18.** (left) 20 data points, generated by uniform random sampling. (middle) The dendrogram generated from complete linkage. The three clusters suggested by the dendrogram are spurious as they cannot be observed in the data. (right) The rapidly decreasing silhouette values in each cluster confirm the absence of a strong cluster structure. Point 18 has a negative silhouette value as it is on average closer to the green points than to the other red points.

just the distance measure used, but also the linkage function.

## 8.6 From kernels to distances

In Section 7.5 we discussed how kernels can be used to extend the power of linear models considerably. Recall that a kernel is a function  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$  that calculates a dot product in some feature space, but without constructing the feature vectors  $\phi(\mathbf{x})$  explicitly. Any learning method that can be defined purely in terms of dot products of data points is amenable to such ‘kernelisation’. Because of the close connection between Euclidean distance and dot products we can apply the same ‘kernel trick’ to many distance-based learning methods.

The key insight is that Euclidean distance can be rewritten in terms of dot products:

$$\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{(\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})} = \sqrt{\mathbf{x} \cdot \mathbf{x} - 2\mathbf{x} \cdot \mathbf{y} + \mathbf{y} \cdot \mathbf{y}}$$

This formula clearly shows that the distance between  $\mathbf{x}$  and  $\mathbf{y}$  decreases whenever the dot product  $\mathbf{x} \cdot \mathbf{y}$  increases, which suggests that the dot product itself is a kind of similarity measure. However, it is not translation-invariant, because it depends on the location of the origin. The two terms  $\mathbf{x} \cdot \mathbf{x}$  and  $\mathbf{y} \cdot \mathbf{y}$  have the effect of making the overall expression translation-invariant. Replacing the dot product with a kernel function  $\kappa$ , we can construct the following kernelised distance:

$$\text{Dis}_\kappa(\mathbf{x}, \mathbf{y}) = \sqrt{\kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}, \mathbf{y}) + \kappa(\mathbf{y}, \mathbf{y})} \quad (8.5)$$

It turns out that  $\text{Dis}_\kappa$  defines a pseudo-metric (see Definition 8.2 on p.235) whenever  $\kappa$  is a positive semi-definite kernel.<sup>2</sup>

<sup>2</sup>It is only a metric if the feature mapping  $\phi$  is injective: suppose not, then some distinct  $\mathbf{x}$  and  $\mathbf{y}$  are mapped to the same feature vector  $\phi(\mathbf{x}) = \phi(\mathbf{y})$ , from which we derive  $\kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}, \mathbf{y}) + \kappa(\mathbf{y}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}) - 2\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) + \phi(\mathbf{y}) \cdot \phi(\mathbf{y}) = 0$ .

As an illustration, [Algorithm 8.5](#) adapts the  $K$ -means algorithm ([Algorithm 8.1](#) on [p.248](#)) to use a kernelised distance. So, the algorithm clusters according to a non-linear distance in instance space, corresponding to Euclidean distance in an implicit feature space. However, one complication arises, which is that [Theorem 8.1](#) doesn't apply to non-linear distances, and so we cannot construct cluster means in instance space. For this reason [Algorithm 8.5](#) treats the clustering as a partition rather than a set of exemplars. Consequently, assigning each data point  $\mathbf{x}$  to its nearest cluster (step 3) is now of quadratic complexity, since for each cluster we need to sum up the distances of all its members to  $\mathbf{x}$ . In contrast, this step is linear in  $|D|$  for the  $K$ -means algorithm.

There is an alternative way to turn dot products into distances. Since the dot product can be written as  $\|\mathbf{x}\| \cdot \|\mathbf{y}\| \cos \theta$ , where  $\theta$  is the angle between the vectors  $\mathbf{x}$  and  $\mathbf{y}$ , we define the *cosine similarity* as

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = \frac{\mathbf{x} \cdot \mathbf{y}}{\sqrt{(\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{y})}} \quad (8.6)$$

Cosine similarity differs from Euclidean distance in that it doesn't depend on the length of the vectors  $\mathbf{x}$  and  $\mathbf{y}$ . On the other hand, it is not translation-independent, but assigns special status to the origin: one way to think of it is as a projection onto a unit sphere around the origin, and measuring distance on that sphere. Cosine similarity is usually turned into a distance metric by taking  $1 - \cos \theta$ . Being defined entirely in terms of dot products, it is as easily kernelised as Euclidean distance.

---

**Algorithm 8.5:**  $\text{Kernel-KMeans}(D, K)$  –  $K$ -means clustering using kernelised distance  $\text{Dis}_K$ .

---

**Input** : data  $D \subseteq \mathcal{X}$ ; number of clusters  $K \in \mathbb{N}$ .  
**Output** :  $K$ -fold partition  $D_1 \uplus \dots \uplus D_K = D$ .  
1 randomly initialise  $K$  clusters  $D_1, \dots, D_K$ ;  
2 **repeat**  
3     assign each  $\mathbf{x} \in D$  to  $\arg \min_j \frac{1}{|D_j|} \sum_{\mathbf{y} \in D_j} \text{Dis}_K(\mathbf{x}, \mathbf{y})$ ;  
4     **for**  $j = 1$  to  $K$  **do**  
5          $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;  
6     **end**  
7 **until** no change in  $D_1, \dots, D_K$ ;  
8 **return**  $D_1, \dots, D_K$ ;

---

## 8.7 Distance-based models: Summary and further reading

Along with linear models, distance-based models are the second group of models with strong geometric intuitions. The literature on distance-based models is rich and diverse; in this chapter I've concentrated on getting the main intuitions across.

- ☞ In [Section 8.1](#) we reviewed the most commonly used distance metrics: the Minkowski distance or  $p$ -norm with special cases Euclidean distance ( $p = 2$ ) and Manhattan distance ( $p = 1$ ); the Hamming distance, which counts the number of bits or literals that are different; and the Mahalanobis distance, which decorrelates and normalises the features ([Mahalanobis, 1936](#)). Other distances can be taken into account, as long as they satisfy the requirements of a distance metric listed in [Definition 8.2](#).
- ☞ [Section 8.2](#) investigated the key concepts of neighbours and exemplars. Exemplars are either centroids that find a centre of mass according to a chosen distance metric, or medoids that find the most centrally located data point. The most commonly used centroid is the arithmetic mean, which minimises squared Euclidean distance to all other points. Other definitions of centroids are possible but harder to compute: e.g., the geometric median is the point minimising Euclidean distance, but does not admit a closed-form solution. The complexity of finding a medoid is always quadratic regardless of the distance metric. We then considered nearest-neighbour decision rules, and looked in particular at the difference between 2-norm and 1-norm nearest-exemplar decision boundaries, and how these get refined by switching to a 2-nearest-exemplars decision rule.
- ☞ In [Section 8.3](#) we discussed nearest-neighbour models which simply use the training data as exemplars. This is a very widely used model for classification, the origins of which can be traced back to [Fix and Hodges \(1951\)](#). Despite its simplicity, it can be shown that with sufficient training data the error rate is at most twice the optimal error rate ([Cover and Hart, 1967](#)). The 1-nearest neighbour classifier has low bias but high variance; by increasing the number of neighbours over which we aggregate we can reduce the variance but at the same time increase the bias. The nearest-neighbour decision rule can also be applied to real-valued target variables, and more generally to any task where we have an appropriate aggregator for multiple target values.
- ☞ [Section 8.4](#) considered a number of algorithms for distance-based clustering using either arithmetic means or medoids. The  $K$ -means algorithm is a simple heuristic approach to solve the  $K$ -means problem that was originally proposed

in 1957 and is sometimes referred to as Lloyd's algorithm (Lloyd, 1982). It is dependent on the initial configuration and can easily converge to the wrong stationary point. We also looked at the  $K$ -medoids and partitioning around medoids algorithms, the latter due to Kaufman and Rousseeuw (1990). These are computationally more expensive due to the use of medoids. Silhouettes (Rousseeuw, 1987) are a useful technique to check whether points are on average closer to the other members of their cluster than they are to the members of the neighbouring cluster. Much more detail about these and other clustering methods is provided by Jain, Murty and Flynn (1999).

- ☞ Whereas the previous clustering methods all result in a partition of the instance space and are therefore predictive, hierarchical clustering discussed in Section 8.5 applies only to the given data and is hence descriptive. A distinct advantage is that the clustering is constructed in the form of a dendrogram, which means that the number of clusters does not need to be specified in advance and can be chosen by inspecting the dendrogram. However, the method is computationally expensive and infeasible for large data sets. Furthermore, it is not always obvious which of the possible linkage functions to choose.
- ☞ Finally, in Section 8.6 we briefly considered how distances can be 'kernelised', and we gave one example in the form of kernel  $K$ -means. The use of a non-Euclidean distance metric leads to quadratic complexity of recalculating the clusters in each iteration.

