

## Chapter 14

# Unsupervised Learning

It would be a mistake to think that machine learning always requires examples with class labels. Far from it! Useful information can be gleaned even from examples whose classes are not known. This is sometimes called *unsupervised learning*, in contrast to the term *supervised learning* which is used when talking about induction from pre-classified examples.

While supervised learning focuses on induction of classifiers, unsupervised learning is interested in discovering useful properties of available data. Perhaps the most popular task looks for groups (called clusters) of similar examples. The centroids of these groups can then be used as gaussian centers for Bayesian or RBF classifiers, as predictors of unknown attribute values, and even as visualization tools for multidimensional data. Last but not least, techniques used in unsupervised learning can be used to create higher-level attributes from existing ones.

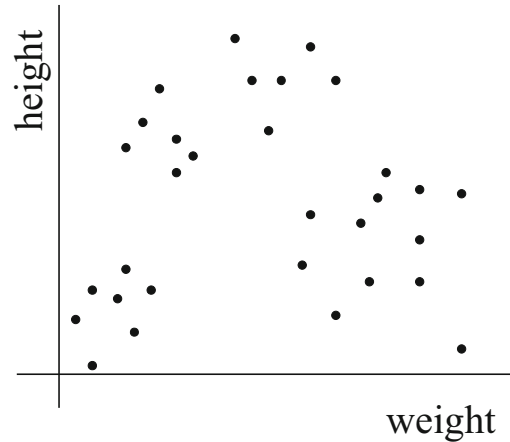
The chapter describes some practical techniques for unsupervised learning, explaining the basic algorithms, their behaviors in practical circumstances, and the benefits they offer.

### 14.1 Cluster Analysis

The fundamental task in unsupervised learning is *cluster analysis*. Here, the input is a set of examples, each described by a vector of attribute values—but no class labels. The output is a set of two or more clusters of examples.

**Identifying Groups of Similar Examples** Figure 14.1 shows a simple domain with a few examples described by two attributes: weight and height. An observer can easily see that the examples form three or four groups, depending on the subjective “level of resolution.”

**Fig. 14.1** A two-dimensional domain with clusters of examples



Visual identification of such groups in a two-dimensional space is easy, but in four or more dimensions, humans can neither visualize the data nor see the clusters. These can only be detected by cluster-analysis algorithms.

**Representing Clusters by Centroids** To begin with, we have to decide how the clusters are to be described. A few alternatives can be considered: it is possible to specify the clusters' locations, sizes, boundaries, and perhaps some other aspects. But the simplest approach relies on *centroids*.<sup>1</sup> If all attributes are numeric, the centroid is identified with the averages of the individual attributes. For instance, suppose a two-dimensional cluster consists of the following examples: (2, 5), (1, 4), (3, 6). In this case, the centroid is described by vector (2, 5) because the first attribute's average is  $\frac{2+1+3}{3} = 2$  and the second attribute's average is  $\frac{5+4+6}{3} = 5$ .

The averages can be calculated even when the attributes are discrete if we know how to turn them into numeric ones. Here is a simple way of doing so. If the attribute can acquire three or more different values, we can replace each attribute-value pair with one boolean variable (say, `season=fall`, `season=winter`, etc.). The values of the boolean attributes are then represented by 0 or 1 instead of *false* and *true*, respectively.

**What Should the Clusters Be Like?** Clusters should not overlap each other: each example must belong to one and only one cluster. Within the same cluster, the examples should be relatively close to each other, certainly much closer than to the examples from the other clusters.

An important question will ask how many clusters the data contain. In Fig. 14.1, we noticed that the human observer discerns either three or four clusters. However, the scope of existing options is not limited to these two possibilities. At one extreme, the entire training set can be thought of as forming one big cluster; at the other,

<sup>1</sup>Machine learning professionals sometimes avoid the term “center” which might imply mathematical properties that are for the specific needs of cluster analysis largely irrelevant.

each example can be seen as representing its own single-example cluster. Practical implementations often side-step the problem by asking the user to supply the number of clusters by way of an input parameter. Sometimes, however, machine-learning software is expected to determine the number automatically.

**Problems with Measuring Distances** Algorithms for cluster analysis usually need a mechanism to evaluate the distance between an example and a cluster. If the cluster is described by its centroid, the Euclidean distance between the two vectors seems to offer a good way of doing so—assuming that we are aware of situations where this can be misleading. This last statement calls for an explanation.

Euclidean distance may be inconvenient in the case of discrete attributes, but we already know how to deal with them. More importantly, we must not forget that each attribute is likely to represent a different quantity, which renders the use geometric distance rather arbitrary: a 4-year difference in `age` is hard to compare with a 4-foot difference in `height`. Also the problem of scaling plays its role: if we replace feet with miles, the distances will change considerably.

We have encountered these problems earlier, in Chap. 3. In the context of cluster analysis, however, these issues tend to be less serious than in  $k$ -NN classifiers. Most of the time, engineers get around the difficulties by normalizing all attribute values into the unit interval,  $x_i \in [0, 1]$ . We will return to normalization in Sect. 14.2.

**A More General Formula for Distances** If the examples are described by a mixture of numeric and discrete attributes, we can rely on the sum of the squared distances along corresponding attributes. More specifically, the following expression is recommended (denoting by  $n$  the number of attributes):

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n d(x_i, y_i)} \quad (14.1)$$

In this formula, we will use  $d(x_i, y_i) = (x_i - y_i)^2$  for continuous attributes. For discrete attributes (including boolean attributes), we put  $d(x_i, y_i) = 0$  if  $x_i = y_i$  and  $d(x_i, y_i) = 1$  if  $x_i \neq y_i$ .

**Which Cluster Should an Example Belong To?** Let us suppose that each example is described by an attribute vector,  $\mathbf{x}$ , and that each cluster is defined by its centroid—which, too, is an attribute vector.

Suppose there are  $N$  clusters whose centroids are denoted by  $\mathbf{c}_i$ , where  $i \in (1, N)$ . The example  $\mathbf{x}$  has a certain distance  $d(\mathbf{x}, \mathbf{c}_i)$ , from each centroid. If  $d(\mathbf{x}, \mathbf{c}_p)$  is the smallest of these distances, it is natural to expect that  $\mathbf{x}$  be placed in cluster  $\mathbf{c}_p$ .

For instance, suppose that we use the Euclidean distance, and that there are three clusters. If the centroids are  $\mathbf{c}_1 = (3, 4)$ ,  $\mathbf{c}_2 = (4, 6)$  and  $\mathbf{c}_3 = (5, 7)$ , and if the example is  $\mathbf{x} = (4, 4)$ , then the Euclidean distances are  $d(\mathbf{x}, \mathbf{c}_1) = 1$ ,  $d(\mathbf{x}, \mathbf{c}_2) = 2$ , and  $d(\mathbf{x}, \mathbf{c}_3) = \sqrt{10}$ . Since  $d(\mathbf{x}, \mathbf{c}_1)$  is the smallest of the three values, we conclude that  $\mathbf{x}$  should belong to  $\mathbf{c}_1$ .

**Benefit 1: Estimating Missing Values** The knowledge of the clusters can help us estimate missing attribute values. Returning to Fig. 14.1, the reader will notice that if `weight` is low, the example is bound to belong to the bottom-left cluster. In this case, also `height` is likely to be low because it is low in all examples found in this cluster. This aspect tends to be even more strongly pronounced in realistic data described by multiple attributes.

In example descriptions, some attribute values are sometimes unknown. As a simple way of dealing with this issue, Sect. 10.4 suggested that the missing value be estimated as the average or the most frequent value encountered in the training set. However, an estimate of the missing value as the average or the most frequent value of the given cluster is sounder because it uses more information about the nature of the domain.

**Benefit 2: Reducing the Size of RBF Networks and Bayesian Classifiers** Cluster analysis can assist such techniques as Bayesian learners and radial-basis-function networks. The reader will recall that these paradigms operate with centers.<sup>2</sup> In the simplest implementation, the centers are identified with the attribute vectors of the individual examples. In domains with millions of examples, however, this would lead to impractically big classifiers. The engineer then prefers to divide the training set into  $N$  clusters, and to identify the gaussian centers with the centroids of the clusters.

**Benefit 3: A Simple Classifier** Finally, the knowledge of data clusters may be useful in supervised learning. It is quite common that all (or almost all) examples in a cluster belong to the same class. In that case, the developer of a supervised-learning software may decide first to identify the clusters, and then label each cluster with its dominant class.

## What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- What is the main difference between unsupervised learning and supervised learning?
- What is a cluster of examples? Define some mechanisms to describe the clusters. How can we measure the distance between an example and a cluster?
- Summarize the main benefits of cluster analysis.

---

<sup>2</sup>For instance, the section on RBF networks denoted these centers by  $\mu_i$ 's.

## 14.2 A Simple Algorithm: $k$ -Means

Perhaps the simplest algorithm to detect clusters of data is known under the name  $k$ -means. The “ $k$ ” in the name denotes the requested number of clusters—a parameter whose value is supplied by the user.

**The Outline of the Algorithm** The pseudocode of the algorithm is provided in Table 14.1. The first step creates  $k$  initial clusters such that each example finds itself in one and only one cluster. After this, the coordinates of all centroids are calculated. Let us note that the problem of initialization is somewhat more complicated than that. We will return to this issue presently.

In the next step,  $k$ -means investigates one example at a time, calculating its distances from all centroids. The nearest centroid then defines the cluster to which the example should belong. If the example already *is* that cluster, nothing needs to be done; otherwise, the example is transferred from the current (wrong) cluster to the right one. After the relocation, the centroids of the two affected clusters (the one that lost the example, and the one that gained it) have to be recalculated. The procedure is graphically illustrated by the single-attribute domain from Fig. 14.2. Here, two examples find themselves in the wrong cluster and are therefore relocated. Note how, after the example relocation, the vertical bars (separating the clusters), and also the centroids, change their locations.

**Termination** The good thing about the algorithm described in Table 14.1 is that the process is guaranteed to reach a situation where each example finds itself in the nearest cluster so that, from this moment on, no further transfers are needed. The clusters do not overlap. Since this is usually achieved in a manageable number of steps, no sophisticated termination criterion is needed here.

**Table 14.1** The clustering algorithm  $k$ -means

---

Input: a set of examples without class labels  
user-set constant  $k$

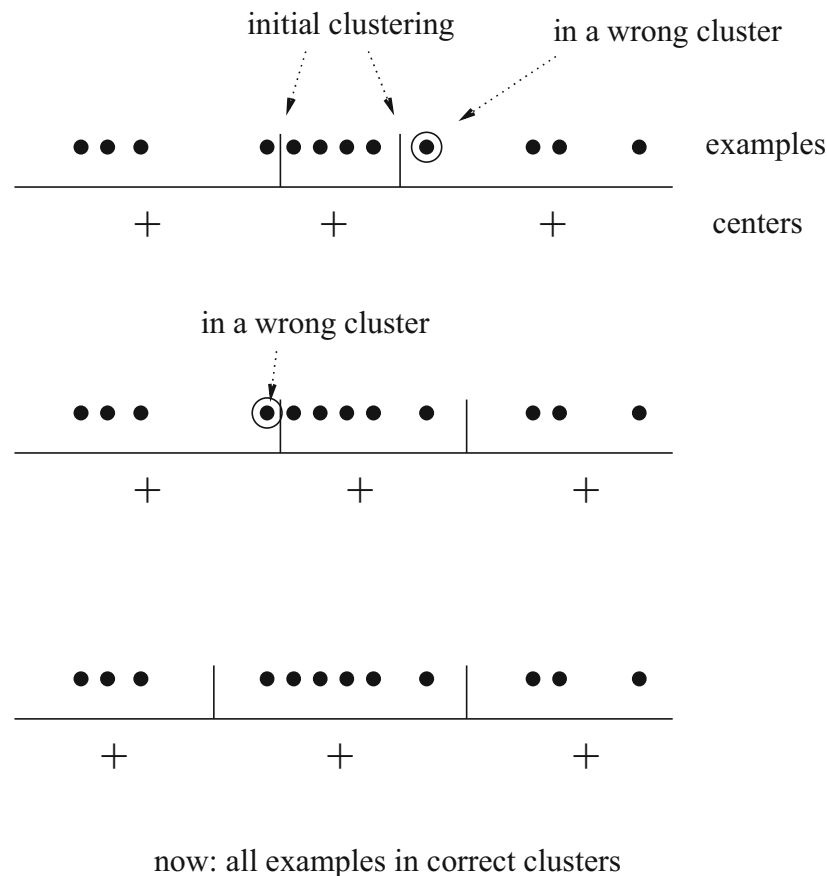
1. Create  $k$  initial clusters. For each, calculate the coordinates of its centroid,  $C_i$ , as the numeric averages of the attribute values in the examples it contains.
2. Choose an example,  $\mathbf{x}$ , and find its distances from all centroids. Let  $j$  be the index of the *nearest centroid*.
3. If  $\mathbf{x}$  already finds itself in the  $j$ -th cluster, do nothing. Otherwise, move  $\mathbf{x}$  from its current cluster to the  $j$ -th cluster and recalculate the centroids.
4. Unless a stopping criterion has been satisfied, repeat the last two steps for another example.

---

Stopping criterion: each training example already finds itself in the nearest cluster.

---

Let us consider an almost trivial domain where 13 examples are described by a single numeric attribute. Suppose the examples have been initially divided into the three groups indicated here by the *vertical bars*. The following sequence shows how two examples (marked by *circles*) are moved from one cluster to another.



After the second transfer, the clusters are perfect and the calculations can stop.

**Fig. 14.2** Illustration of the  $k$ -means procedure in a domain with one numeric attribute

**Numeric Example** In Table 14.2, a set of nine two-dimensional examples has been randomly divided into three groups (because the user specified  $k = 3$ ), each containing the same number of examples. The table also provides the centroids for each group.  $k$ -means goes through these examples systematically, one by one—in this concrete case, starting with group-2. For each example, its distance from each centroid is calculated. It turns out that the first example from group-2 already finds itself in the right cluster. However, the second example is closer to group-1 than to group-2 and, for this reason, has to be transferred from its original cluster to group-1. After this, the affected centroids are recalculated.

**Table 14.2** Illustration of the *k*-means procedure in a domain with two attributes

The table below contains three initial groups of vectors. The task is to find “ideal” clusters using the *k*-means ( $k = 3$ ).

	Group-1	Group-2	Group-3
	(2, 5)	(4, 3)	(1, 5)
	(1, 4)	(3, 7)	(3, 1)
	(3, 6)	(2, 2)	(2, 3)
Centroids:	(2, 5)	(3, 4)	(2, 3)

Let us pick the first example in group-2. The Euclidean distances between this example, (4, 3), and the centroids of the three groups are  $\sqrt{8}$ ,  $\sqrt{2}$ , and  $\sqrt{4}$ , respectively. This means that the centroid of group-2 is the one nearest to the example. Since this is where the example already is, *k*-means does not do anything.

Let us now proceed to the second example in group-2, (3, 7). In this case, the distances are  $\sqrt{5}$ ,  $\sqrt{9}$ , and  $\sqrt{17}$ , respectively. Since the centroid of group-1 has the smallest distance, the example is moved from group-2 to group-1. After this, the averages of the two affected groups are recalculated.

Here are the new clusters:

	Group-1	Group-2	Group-3
	(2, 5)	(4, 3)	(1, 5)
	(1, 4)	(2, 2)	(3, 1)
	(3, 6)		(2, 3)
	(3, 7)		
Averages:	(2.25, 5.25)	(3, 2.5)	(2, 3)

The process continues as long as any example transfers are needed.

**The Need for Normalization** The reader will recall that, when discussing *k*-NN classifiers, Chap. 3 argued that inappropriate attribute scaling will distort the distances between attribute vectors. The same concern can be raised in cluster analysis. It is therefore always a good idea to normalize the vectors so that all numeric attributes have values from the same range, say, from 0 to 1.

The simplest way of doing so is to determine for the given attribute its maximum (*MAX*) and minimum (*MIN*) value in the training set. Then, each value of this attribute is re-calculated using the following formula:

$$x = \frac{x - MIN}{MAX - MIN} \quad (14.2)$$

As for boolean attributes, their values can simply be replaced with 1 and 0 for *true* and *false*, respectively. Finally, an attribute that acquires  $n$  discrete values (such as *season*, which has four different values) can be replaced with  $n$  boolean attributes, one for each value—and, again, for the values of these boolean attributes, 1 or 0 are used.

**Computational Aspects of Initialization** To reach its goal, the *k-means* needs to go through a certain number of transfers of examples from wrong clusters to the right clusters. How many such transfers are needed depends on the contents of the initial clusters. At least in theory, it *can* happen that the randomly created initial clusters are already perfect, and not a single example needs to be moved. This, of course, is an extreme, but the reader surely understands that if the initialization is “lucky,” fewer relocations have to be carried out than otherwise. Initialization matters in the sense that a better starting point ensures that the solution is found sooner.

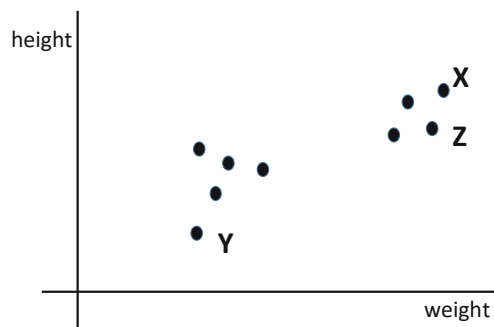
**How to Initialize** In some domains, we can take advantage of some background knowledge about the problem at hand. For instance, seeking to create initial clusters in a database of a company’s employees, the data analyst may speculate that it makes sense to group them by their age, salary, or some other intuitive criterion, and that the groups thus obtained will be good initial clusters.

In other applications, however, no such guidelines exist. The simplest procedure then picks  $k$  random training examples and regards them as *code vectors* to define initial centroids. The initial clusters are then created by associating each of the examples with its nearest code vector.

**A More Serious Problem with Initialization** There is another issue, and a more serious one than the computational costs. The thing is, also the *composition* of the resulting clusters (once the *k-means* has completed its work) may depend on initialization. Choose a different set of initial code vectors, and the technique may generate a different set of clusters.

The point is illustrated in Fig. 14.3. Suppose that the user wants two clusters ( $k = 2$ ). If he chooses as code vectors the examples denoted by **x** and **y** then the initial clusters created with the help of these two examples are already perfect.

**Fig. 14.3** Suppose  $k = 2$ . If the code vectors are **[x,y]**, the initial clusters for *k-means* will be different than when the code vectors are **[x,z]**





The situation changes when we choose for the code vectors examples  $\mathbf{x}$  and  $\mathbf{z}$ . In this event, the two initial clusters will have a very different composition, and *k-means* is likely to converge on a different set of clusters. The phenomenon will be more pronounced if there are “outliers,” examples that do not apparently belong to any of the two clusters.

**Summary of the Main Problems** The good thing about *k-means* is that it is easy to explain and easy to implement. Yet this simplicity comes at a price. The technique is sensitive to initialization; the user is expected to provide the number of clusters (though he may not know how many clusters there are); and, as we will see, some clusters can never be identified, in this manner. The next sections will take a look at some techniques to overcome these shortcomings.

## What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- Describe the principle of the *k-means* algorithm. What is the termination criterion?
- What would be the consequence if we did not normalize the training set? Write down the simple normalization formula.
- Describe some methods for the initialization of *k-means*. What are the main consequences of good or bad initialization?

## 14.3 More Advanced Versions of *k*-Means

The previous section described the baseline version of *k-means* and explained its main weaknesses: sensitivity to initialization, and lack of flexibility in deciding about the number of clusters. Let us now take a look at some improvements meant to address these shortcomings.

**Quality of the Clusters** An engineer always needs to be able to evaluate and compare alternative solutions. And thus also for the topic addressed by this section, we need a criterion to help us choose between different sets of clusters. Primarily, the criterion should reflect the fact that we want clusters that minimize the average distance between examples and the centroids of “their” clusters.

Let us denote by  $d(\mathbf{x}, \mathbf{c})$  the distance between example  $\mathbf{x}$  and the centroid,  $\mathbf{c}$ , of the cluster to which  $\mathbf{x}$  belongs. If all attributes are numeric, and if they all have been normalized, then  $d(\mathbf{x}, \mathbf{c})$  can be evaluated either by the Euclidean distance or by the more general Eq. (14.1).

The following formula (in which  $SD$  stands for *summed distances*) sums up the distances of all examples from their clusters' centroids. Here,  $\mathbf{x}_i^{(j)}$  denotes the  $i$ -th example in the  $j$ -th cluster,  $K$  is the number of clusters,  $n_j$  is the number of examples in the  $j$ -th cluster, and  $\mathbf{c}_j$  is the  $j$ -th cluster's centroid.

$$SD = \sum_{j=1}^K \sum_{i=1}^{n_j} d(\mathbf{x}_i^{(j)}, \mathbf{c}_j) \quad (14.3)$$

In cluster analysis, we seek to minimize  $SD$ . When calculating this quantity, we must not forget that the value obtained by Eq. (14.3) will go down if we increase the number of clusters (and thus decrease their average size), reaching  $SD = 0$  in the extreme case when each cluster is identified with one and only one training example. The formula is therefore useful only if we compare solutions that have similar numbers of clusters.

**Using Alternative Initializations** Knowing that the composition of the resulting clusters depends on the algorithm's initialization, we can suggest a simple improvement. We will define two or more sets of initial code vectors, and apply *k-means* separately to each of them. After this, we will evaluate the quality of all the alternative data partitionings thus obtained, using the criterion defined by Eq. (14.3). The best solution is the one for which we get the lowest value. This solution is then retained, and the others discarded.

**Experimenting with Different Values of  $k$**  One obvious weakness of *k-means* is the requirement that the user should provide the value of  $k$ . This is easier said than done because, more often than not, the engineer has no idea into how many clusters the available data naturally divide. Unless more sophisticated techniques are used (about these, see later), the only way out is to try a few different values, and then pick the best according to an appropriate criterion (such as the one defined in Eq. (14.3)). As we already know, the shortcoming of this criterion is that it tends to give preference to small clusters. For this reason, data analysts often normalize the value of  $SD$  by  $k$ , the number of clusters.

**Post-processing: Merging and Splitting Clusters** The quality of the set of clusters created by *k-means* can often be improved by post-processing techniques that either increase the number of clusters by splitting, or decrease the number by merging.

As for merging, two neighboring clusters will be merged if their mutual distance is small. To find out whether the distance merits the merging, we simply calculate the distance of two centroids, and then compare it with the average cluster-to-cluster distance calculated by the following sum, where  $\mathbf{c}_i$  and  $\mathbf{c}_j$  are centroids:

$$S = \sum_{i \neq j} d(\mathbf{c}_i, \mathbf{c}_j) \quad (14.4)$$

Conversely, splitting makes sense when the average example-to-example distance within some cluster is high. The concrete solution is not easy to formalize because once we have specified that cluster  $C$  is to be split into  $C_1$  and  $C_2$ , we need to decide which of  $C$ 's examples will go to the first cluster and which to the second. Very often, however, it is perfectly acceptable to identify in  $C$  two examples with the greatest mutual distance, and then treat them as the code vectors of newly created  $C_1$  and  $C_2$ , respectively.

**Hierarchical Application of  $k$ -Means** Another modification relies on recursive calls. The technique begins with running  $k$ -means for  $k = 2$ , obtaining two clusters. After this,  $k$ -means is applied to each of these two clusters separately, again with  $k = 2$ . The process is continued until an appropriate termination criterion has been satisfied—for instance, the maximum number of clusters the user wants to obtain, or the minimum distance between neighboring clusters.

This hierarchical version side-steps one serious shortcoming of  $k$ -means: the necessity to provide the number of clusters because, in this case, this is found automatically. This is particularly useful when the goal is to identify the gaussian centers in Bayesian classifiers or in RBF networks.

## What Have You Learned?

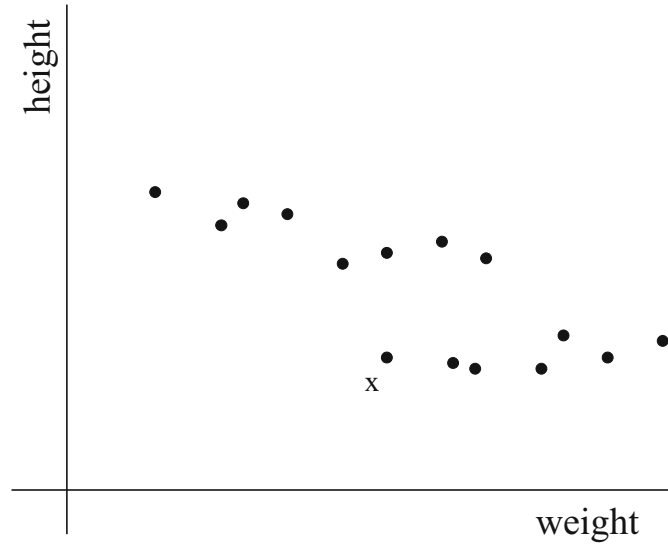
To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- Discuss some shortcomings of the  $k$ -means algorithm, and describe the simple techniques for overcoming them.
- Specify how you would implement the cluster-splitting and cluster-merging techniques described in this section.
- Explain the principle of hierarchical application of  $k$ -means.

## 14.4 Hierarchical Aggregation

Just as any other machine-learning technique,  $k$ -means has not only advantages, but also shortcomings. To avoid at least some of the latter, we need an alternative, an approach which is likely to prove useful in situations where  $k$ -means fails.

**Another Serious Limitation of  $k$ -Means** By minimizing the distance between the examples and the centroids of the clusters to which these examples belong,  $k$ -means essentially guarantees that it will discover “convex” clusters. Most of the time, this is indeed what we want. Such clusters can be useful in the context of Bayesian classifiers and the RBF networks where they reduce, sometimes very significantly, the number of employed gaussian centers.



**Fig. 14.4** Note that the leftmost examples in the “bottom” cluster are closer to the “upper” cluster’s centroid than to its own. In a domain of this kind *k-means* will not find the best solution

This approach, however, will do a poor job if the clusters are of a different (“non-convex”) nature. To see the point, consider the clusters in Fig. 14.4. Here, the leftmost example,  $\mathbf{x}$ , in the bottom cluster is closer to the centroid of the upper cluster, and *k-means* would therefore relocate it accordingly—and yet we feel that this would not do justice to the nature of the two groups.

To deal with data of this kind, we need another technique, one capable of identifying clusters such as those in Fig. 14.4.

**An Alternative Way of Measuring Inter-Cluster Distance** In the previous section, the distance between two clusters was evaluated as the Euclidean distance between their centroids. However, for the needs of the approach described below, we will suggest another mechanism: we will measure the distances between all pairs of examples,  $[\mathbf{x}, \mathbf{y}]$ , such that  $\mathbf{x}$  comes from the first cluster and  $\mathbf{y}$  from the second. The smallest value found among all these example-to-example distances then defines the distance between the two clusters.

Returning to Fig. 14.4, the reader will agree that, along this new distance metric, example  $\mathbf{x}$  is closer to the bottom cluster than to the upper clusters. This means that the limitation mentioned in the previous paragraph has been in this particular case eliminated. The price for this improvement is increased computational costs: if  $N_A$  is the number of examples in the first cluster, and  $N_B$  the number of examples in the second, then  $N_A \times N_B$  example-to-example distances have to be evaluated. Most of the time, however, the clusters are not going to so big as to make this an issue.

**Numeric Example** For the sake of illustration of how this new distance metric is calculated, consider the following two clusters,  $A$  and  $B$ .

$A$	$B$
$\mathbf{x}_1 = (1, 0)$	$\mathbf{y}_1 = (3, 3)$
$\mathbf{x}_2 = (2, 2)$	$\mathbf{y}_2 = (4, 4)$

**Table 14.3** The basic algorithm of *hierarchical aggregation*


---

Input: a set of examples without labels

1. Let each example form one cluster. For  $N$  examples, this means creating  $N$  clusters, each containing a single example.
  2. Find a pair of clusters with the smallest cluster-to-cluster distance. Merge the two clusters into one, thus reducing the total number of clusters to  $N - 1$ .
  3. Unless a termination criterion is satisfied, repeat the previous step.
- 

Using the Euclidean formula, we calculate the individual example-to-example distances as follows:

$$\begin{aligned} d(\mathbf{x}_1, \mathbf{y}_1) &= \sqrt{13}, \\ d(\mathbf{x}_1, \mathbf{y}_2) &= \sqrt{25}, \\ d(\mathbf{x}_2, \mathbf{y}_1) &= \sqrt{2}, \\ d(\mathbf{x}_2, \mathbf{y}_2) &= \sqrt{8}. \end{aligned}$$

Observing that the smallest of these values is  $d(\mathbf{x}_2, \mathbf{y}_1) = \sqrt{2}$ , we conclude that the distance between the two clusters is  $d(A, B) = \sqrt{2}$ .

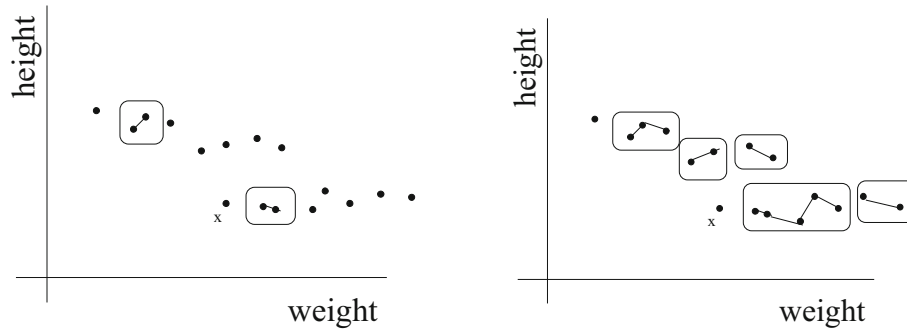
**Hierarchical Aggregation** For domains such as the one in Fig. 14.4, the clustering technique known as *hierarchical aggregation* is recommended. The principle is summarized by the pseudocode in Table 14.3.

In the first step, each example defines its own cluster. This means that in a domain with  $N$  examples, we have  $N$  initial clusters. In a series of the subsequent steps, hierarchical aggregation always identifies a pair of clusters that have the smallest mutual distance along the distance metric from the previous paragraphs. These clusters are then merged. At an early stage, this typically amounts to merging pairs of neighboring examples. Later, this results either in adding an example to its nearest cluster, or in merging two neighboring clusters. The first few steps are illustrated in Fig. 14.5.

The process continues until an appropriate termination criterion is satisfied. One possibility is to stop when the number of clusters drops below a user-specified threshold. Alternatively, one can base the stopping criterion on the cluster-to-cluster distance (finish when the smallest of these distances exceeds a certain value).

## What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.



**Fig. 14.5** Hierarchical aggregation after first two steps (*left*) and after first nine steps (*right*). Note how the clusters are gradually developed

- What kind of clusters cannot be detected by the *k-means* algorithm?
- What distance metric is used in hierarchical aggregation? What are the advantages and disadvantages of this metric?
- Describe the principle of the hierarchical-aggregation approach to clustering. For what kind of clusters is it particularly suited?

## 14.5 Self-Organizing Feature Maps: Introduction

Let us now introduce yet another approach to unsupervised learning, this time borrowing from the field of neural networks. The technique is known as a *self-organizing feature map*, *SOFM*.<sup>3</sup> Another name commonly used in this context is *Kohonen networks*, to honor its inventor.

**The Idea** Perhaps the best way to explain the nature of SOFM is to use, as a metaphor, the principle of physical attraction. A *code vector*, initially generated by a random-number generator, is subjected to the influence of a sequence of examples (attribute vectors), each “pulling” the vector in a different direction. In the long run, the code vector settles in a location that represents a compromise over all these conflicting forces.

The whole network consists of a set of *neurons* arranged in a two-dimensional matrix such as the one shown in Fig. 14.6. Each node (a neuron) in this matrix represents a code vector that has the same length (the same number of attributes) as the training examples. At the bottom is an input attribute vector that is connected to all neurons in parallel. The idea is to achieve by training a situation where neighboring neurons respond similarly to similar input vectors. For the sake of simplicity, the input vector in the picture only has two attributes,  $x_1$  and  $x_2$ . In reality, it can have a great many.

<sup>3</sup>In statistics, and in neural networks, scientists often use the term *feature* instead of *attribute*.