CHAPTER 4

CLUSTERING

Birds of a feather flock together.

INTRODUCTION

Every branch of knowledge depends on the division of its concepts into separate categories. For example, biologists have developed a taxonomy to categorize all the living things into kingdom, phylum, class, order, family, genus, and species. Music is categorized into various genres—*carnatic*, *hindustani*, *jazz*, *rock*, *bass*, *pop*, *new age*, etc. Directories on the World Wide Web such as Yahoo! categorize webpages based on their subject matter into various categories such as business, computers, etc.

In all the above examples, the objects within a category are considerably similar when compared to the objects across categories. While humans are adept at categorizing objects, the task of clustering in data mining is to automatically come up with such a categorization. Clustering is the task of organizing data into groups (known as clusters) such that the data objects that are similar to (or close to) each other are put in the same cluster. There is no one correct basis of clustering–there could be many different ways to categorize data objects. Clustering schemes are evaluated based on the similarity of objects within each cluster and the dissimilarity of objects across clusters.

To a beginner, clustering is often confused with classification. The two, however, are significantly different. Classification is a form of supervised learning, where the class labels of some training samples are given–these samples are used as examples to supervise the learning of a classification model. On the other hand, clustering is a form of unsupervised learning in which no class labels are provided. Instead, data records need to be grouped based on how similar they are to other records.

Clustering is often the first data mining task applied on a given collection of data and is used to explore if any underlying patterns exist in the data. The presence of dense well-separated clusters indicates that there are structure and patterns to be explored in the data. Examples of clustering tasks include dividing the stars in the sky into constellations, dividing a large group of people into small teams, and segmenting customers into small groups for additional analysis in marketing activities.

4.1 Basic Problem Definition

A cluster is a collection of data objects that are similar to each other and dissimilar to the data objects in other clusters. The following illustration will help to grasp the abstract definitions provided later.

Illustration 4.1 Table 4.1 shows data related to students, their native state, and their marks in various subjects.

Table 4.1 Data of students

Roll No.	Name	Gender	State	Maths	Physics	Chemistry	English	Social Science
200101001	Govind	M	AP	94	89	75	62	57
200101002	Ramakanth	M	MP	72	65	67	80	85
200101003	Karuna	F	AP	95	92	77	65	60
...	...	...	...	...	...	...	...	...

In this table, it is clear that Govind and Karuna have obtained very similar marks in all the subjects. They can, therefore, be put as parts of the same cluster. In contrast, the marks obtained by Ramakanth are very different. Therefore, he needs to be part of a different cluster. In this manner, all the students can be grouped into a few clusters. This grouping can be very useful for several purposes such as career counselling, forming tutorial groups where different groups are taught differently, and even for future employers to know which group to target.

Instance An instance is an object that is represented by a vector of numeric or categorical attributes.

The input data model for clustering consists of instances (also known as samples, cases, objects, or records) where each instance is defined by a number

of attributes (also known as features or variables), which could be numeric (such as marks in various subjects) or categorical (such as gender and state). It may be recalled that in Chapter 3, the input data model for classification also consisted of such instances. The difference is that the instances here are not labelled.

In order to determine whether two data objects are similar to each other, clustering strategies use the notion of distance function. Such a function takes two objects as input and returns a positive real number representing the distance between the two objects—the smaller the distance, the more similar the two objects are to each other.

Distance Function A distance function d is a mapping from every pair of instances (x, y) to real numbers such that,

1. $d(x, y) \geq 0$
2. $d(x, x) = 0$
3. $d(x, y) = d(y, x)$
4. $d(x, y) < d(x, z) + d(z, y)$

The first three properties are obvious. The fourth property is known as the *triangle inequality*. Several popular distance functions are available that satisfy these properties and one of them needs to be chosen for each clustering problem based on the application domain. Functions satisfying all four properties are also referred to as metrics. In some applications, distance measures are designed that do not follow the triangle inequality. Strictly speaking, such functions are not distance metrics, however, they are still useful for clustering.

Clustering Given a set of n instances $x_1, \ldots, x_n$ represented by vectors $v_1, \ldots, v_n$, and a distance function d, the clustering problem is to determine a function f such that $f(x_i) = j$, where j is a (small) natural number corresponding to a cluster. The function f is to be designed with the objective that, instances in the same cluster must be closer to each other (with respect to d) than instances across clusters.

For many practical applications, it is either very difficult or even impossible to design clustering algorithms that achieve the objective stated in the definition. Instead, algorithms are usually designed to satisfy the objective as much as possible. This means that there may be instances that are put as parts of the same cluster, but some of these are actually closer to instances in other clusters. This also means that there is usually no single correct answer for a clustering task.

4.2 Clustering: Applications

There are numerous potential applications of clustering analysis. Some of these applications are listed below, under two broad categories.

4.2.1 Targetting Similar People or Objects

Student tutorial groups Consider students completing their tenth class exams. It is theoretically possible to analyze the performance of each student in various subjects and then provide specific training which will benefit that student. Although this kind of focussed training is very effective, it is clearly infeasible due to the size of faculty and resources required. However, by dividing the students into groups based on their exam scores, entire groups can be targeted differently.

Each group of students can be characterized by the fact that they may be good in some subjects, average in some others, and bad in the remaining. Specific tutorial classes or other learning material can be designed to train these students in the subjects in which they are weak. They can also be given special advanced lectures and study material to help them advance rapidly in the subjects in which they are good.

Hobby groups Several popular chatting engines on the World Wide Web collect details regarding the hobbies and interests of their customers. By clustering the customers based on their shared interests, it is possible to identify people who might become very good friends. These hobby groups can be used as a starting point for other applications such as matchmaking or recommendation engines.

Health support groups Health expert systems are software programs that are used to diagnose the condition of patients by enquiring about their symptoms. Patients could be asked to list the symptoms of their condition or disease. For most well-known diseases, the expert system usually has no trouble in finding the right diagnosis. However, for some conditions, the expert system may not have any diagnosis to provide.

Nevertheless, for all the conditions, these expert systems can be used to enable people, sharing similar symptoms, to communicate with each other. Most patients find comfort and obtain a sense of well-being when they communicate with fellow patients. This is especially useful when the underlying condition or trauma is due to psychological reasons.

Customer groups for marketing It is useful for any company to find clusters of their customers based on characteristics such as age group, gender, occupation, etc. Such a segmentation of customers is invaluable for several reasons as given below:

- It can help in finding a suitable sample of people for surveys regarding their products–each cluster needs to be represented in the sample.
- The problems/suggestions of each group of customers might be unique.
- It can help decide on how to advertise their products based on the feedback given by each customer group.

Clustering e-mail E-mail is one of the most pervasive applications of computers. People or organizations, who receive hundreds of e-mail messages, will definitely find utility in a program that can automatically categorize their e-mail messages into different groups. If e-mail messages are previously categorized (such as *spam* or *non-spam*), then this becomes a classification task. However, the groups are often not known in advance and therefore, a clustering algorithm is needed here.

4.2.2 Deciding on the Locations for an Activity

Exam centres The addresses of the applicants for an exam can be utilized to form a map of their geographical locations. These locations can then be clustered into groups and suitable exam centres for each group can be decided. This would reduce the overall effort and the cost of travel for the numerous applicants for that exam.

Locations for a business chain In order to decide locations for setting up a supermarket chain, Automatic Teller Machines (ATM), etc., it is necessary to identify densely populated regions of prospective customers. By surveying a sizeable sample of prospective customers and determining their geographical locations, it becomes possible to cluster these locations. The places to set up the supermarkets or ATMs can then be divided appropriately in these clusters.

Planning a political strategy A political party may be weak (inactive) in some constituencies of the state or country and strong (highly active) in the others. By clustering each category of constituency (weak or strong) separately, it becomes possible to identify the entire regions that are either weak or strong. With this information, a politician could decide to form new centres of activity in the weak regions and distribute these centres appropriately. It may also be

useful to consider shifting some centres from the strong regions to the weak ones.

Alternatively, if the party is by and large not very strong, the politician may decide to concentrate the strength of a party in one particular region.

4.3 Measurement of Similarity

Clustering algorithms rely on the ability to determine whether two data objects are similar to each other. For this they use the notion of a distance function (Section 4.1) where this function takes two objects as input and returns a positive real number representing the distance between the two objects—the smaller the distance, the more similar the two objects are to each other.

Several popular distance functions are available and one of these needs to be chosen for each clustering problem depending on the data types in that specific application. Some of these functions strictly follow the criteria listed in under the definition for distance function (Section 4.1) and are referred to as metrics. Other functions are not designed so strictly due to practical reasons and may not follow some of the criteria listed in that definition.

The distance function is chosen based on the type of attributes used to describe an instance in a clustering problem. The attributes may be categorical or numerical. Each of these has its own accepted ways of measuring distance. Here the distance functions, used for instances that consist of the same types of attributes, are described. Section 4.3.1 will deal with the possibilities of measuring distance when instances consist of more than one type of attributes.

It is to be noted that what follows should be treated only as generic guidelines for deciding what distance function to use. Ultimately, the nuances of specific applications should be taken into account before making the final decision.

4.3.1 Nominal (Categorical) Variables

Categorical variables are also known as nominal variables. The values for a nominal variable are names or strings that come from a (small) set of possible names. For practical storage and processing purposes, the names may be numbers. However, normal processing of numeric variables such as addition and comparisons, such as 'less than', do not make sense. They are only useful for comparison of equality and inequality. Examples of nominal variables include gender, race, country, the make of a car, etc.

Consider two instances x and y, which consist of n nominal attributes. Then, the distance between x and y can be measured using the following function:

$$d(x, y) = (n - m)/n$$

where m is the number of matches, i.e. the number of attributes for which both x and y have the same value.

Weights In some applications, certain attributes may be more important than the others in determining distance. As an example, in deciding whether two people are similar, it may be more important to consider whether they are from the same state rather than having the same name. To take care of this, it is possible to add different weights to different attributes. The distance computation then becomes

$$d(x, y) = 1 - (\Sigma w_i / W)$$

where w_i are the weights of the attributes whose values match the values in x and y, and W is the sum of the weights of all the attributes.

Binary variables A binary variable is a special type of nominal variable that has only two possible values. Normally, gender, marital status, experimental results (success, failure), presence of a word in a document, etc. are represented as binary variables. In some applications it may be significant if two instances share one of the values of a binary variable but not the other. This is shown in the following example.

Example 4.1 Consider the task of measuring the distance between two text documents x and y. Here, each possible word is treated as a binary variable that is either present or absent in a particular document. Since there are many possible words, it is generally insignificant if any particular word is absent in both the documents. However, if a word is present in both the documents, it is considered significant for the purpose of distance computation.

It follows that higher weights are specified for the variables that are present in both the documents and lesser weights are specified for the variables that are absent in both the documents.

Note that the requirement in Example 4.1 is different from the solution discussed earlier where the weights were static—they were decided once and for all, for each variable. Here, the weight of a variable has to be changed depending on whether or not, the two instances share the same value of the weight. One way to do this is to use the *Jaccard coefficient*:

$$d(x, y) = (n - m)/(n - s)$$

where m is the number of the binary variables whose values match in x and y, s is the number of the binary variables whose values are 'absent' in both x and y, and n is the number of attributes.

The above situation may also arise in some applications with non-binary nominal variables– it may be necessary to give different weights to different attributes depending on the values they take in specific instances. For such applications, the solution is to treat a nominal variable v, which can take p different values, as p binary variables. These binary variables are set to 1 if the corresponding value of v is found in the current instance and or else, to 0.

4.3.2 Numeric Variables

Consider two instances x and y that consist of n numeric attributes. The instances can be represented as vectors of the values of these attributes as $(x_1, x_2, x_3,..., x_n)$ and $(y_1,y_2,y_3,...,y_n)$. The distance between x and y will be determined as

$$d(x, y) = (|x_1 - y_1|^q + |x_2 - y_2|^q + \ldots + |x_n - y_n|^q)^{1/q}$$

where q is a positive integer.

This is known as the *Minkowski* distance. The most common forms of this function is when $q = 1$ (in which case it is known as the *manhattan* distance) and $q = 2$ (in which case it is known as the *euclidean* distance). When q becomes infinite the distance computed is known as the *chebychev* distance. As for nominal variables, weights can be applied on each attribute to signify how important it is. In this case, the computation is

$$d(x, y) = \left(w_i |x_1 - y_1|^q + w_2 |x_2 - y_2|^q + \ldots + w_k |x_n - y_n|^q\right)^{1/q}$$

where w_i is the weight of the i^{th} dimension.

Ordinal variables

Ordinal variables are numeric attributes that are intended to represent the rank order of some measured entities. They have all the properties of nominal variables–they can be compared with respect to equality. In addition, they can be compared using 'less than', 'greater than', etc.

Examples of ordinal variables include the rank orders obtained in most competitions. These ranks can be used to determine if a person is better or worse than another, but not by how much. Two people may differ very slightly in their performance, but their ranks may be very different due to an intense competition. Hence, it is meaningless to add or subtract two ranks. Equal differences in rank values do not represent equal intervals.

Consider two instances x and y that consist of k ordinal variables. The range of these k variables may be different—some of them may have only a few ranks, while some may have many ranks. If the Minkowski distance function is applied directly, the result will heavily depend on the range of these variables. Therefore, before applying the Minkowski distance function, it is necessary to first standardize these variables.

If x has a rank r_i for an ordinal variable v whose maximum rank is M, then the standardized value of v is computed as

$$z_i = (r_i - 1)/(M - 1)$$

These standardized values are always in the range (0, 1) and therefore do not have the drawback mentioned above. Hence, they can be used in the computation of the Minkowski distance and its variants.

Interval and ratio-scaled variables

Interval-scaled variables are the numeric attributes that satisfy all the criteria of the ordinal variables (that is comparisons of 'less than', 'equals', etc. are meaningful) and, in addition, have the property of equal distances between the values representing equal intervals. As a result, operations such as addition and subtraction become meaningful. However, the ratios between the values are not meaningful. Hence, multiplication and division cannot be carried out directly. However, the ratio of differences between the values is meaningful. An example of an interval-scaled variable is the date of an event.

Ratio-scaled variables are numeric attributes that satisfy all the criteria of interval-scaled variables, and in addition also have meaningful ratios between their values. Examples of the ratio-scaled variables are weight, height, length, mass, etc. of objects.

Both interval and ratio-scaled variables can be measured in different units. For example, height can be measured in inches, feet, cm, etc. Clearly, from the above formula of the Minkowski distance, different distances will be computed depending on the unit used. This is undesirable. To avoid this, it is necessary to first standardize these variables before using the Minkowski distance functions. One way to do this is to apply the following two step procedure on each variable v:

1. First, calculate the mean absolute deviation, m,

$$m = (|v_1 - m| + |v_2 - m| + \ldots + |v_n - m|)/\mathrm{N}$$

where $v_1, v_2, \ldots, v_n$ are the values of attribute v that acutally occur in the data and m is their mean.

2. Calculate the standardized measurement, or *z-score*,

$$z_i = (v_i - \mu)/m$$

This *z-score* is devoid of units and can be used to compute the Minkowski distance and its variants.

Cosine similarity

The cosine similarity measures the opposite of distance (unlike the previous functions which were distance functions)—its value is high when the two input instances are very similar to each other. Given two instances whose attribute values are given by the vectors $x = (x_1, x_2, x_3, \ldots, x_n)$ and $y = (y_1, y_2, y_3, \ldots, y_n)$, their cosine similarity is defined as

$$x \,.\, y/|x||y|$$

where

$$x \,.\, y = (x_1y_1 + x_2y_2 + \ldots + x_ny_n)$$

and

$$|x| = \sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}$$

This similarity measure represents the cosine of the angle between the vectors x and y. It is very popular in determining the similarity between text documents. Recent research suggests that both cosine similarity and euclidean distance measures yield similar results when the number of variables is high. One advantage of using cosine similarity is that the distances are naturally normalized to be in the range (0, 1).

4.3.3 Instances with Mixed Types

The above subsections described the guidelines to compute the distance between two instances that have several attributes of the same type. We now move on to the computation of distance when instances are described by attributes of different types. The following procedure shows one way to design a good distance function for this mixed case:

1. Standardize the variables according to the previous subsections.
2. Further standardize the variables so that their values are within a common range. To standardize the value v_i of a variable v, whose range is $(\min_1, \max_1)$, to a range of $(\min_2, \max_2)$, compute

$$\min_2 + (v_i - \min_1)(\max_2 - \min_2)/(\max_1 - \min_1)$$

3. Separate out the mixture—group all the variables of each type together, and find the distances for each group separately. Utilize appropriate weights to represent the significance or importance of each component in the distance computation.
4. Finally, compute the resultant distance which will be the average of the individual distances.

4.4 Evaluation of Clustering Algorithms

The primary metric of evaluation of clustering algorithms is the quality of the clusters they produce, i.e. the extent to which they satisfy the clustering objective stated in Section 4.1. According to this objective, instances in the same cluster must be closer to each other than to instances across clusters.

Often, the quality of clustering algorithms is evaluated subjectively. Each algorithm is run over several test datasets for which the natural clusters are previously known. This is possible if the datasets consist of familiar objects. For example, we would know the correct clustering for a dataset of animals, based on biological classification into species, etc. Alternatively, the dataset may consist of only two or three numeric attributes. In that case, the instances can be plotted in a two- or three-dimensional space and the clusters can be identified visually.

Sections 4.4.1–4.4.3 describe techniques for the objective evaluation of the quality of clustering algorithms. This is followed by other metrics for the evaluation of clustering algorithms.

4.4.1 Squared Error Criterion

Instances in a cluster should be closer to the centre of that cluster than to the centre of any other cluster. One way to define the centre of a cluster is to use the mean of that cluster, which is calculated as the average along each attribute of all the points in that cluster.

Cluster Mean If there are N data points in a cluster represented by tuples of the form $(x_{i1}, x_{i2}, \ldots, x_{in})$ for $i = 1 \ldots N$, then its mean is defined as

$$\text{mean} = (\Sigma x_{i1}/N, \Sigma x_{i2}/N, \ldots, \Sigma x_{in}/N)$$

One may also use the median or mode to represent the centre of a cluster. The use of mode for defining cluster centres is especially appropriate when the data consists of categorical attributes.

If some point x is erroneously placed in a wrong cluster C_i, then the distance between x and the centre of C_i (which is $d(x, C_i)$ is going to be larger than the distance if x was placed in its correct cluster. Thus, this distance can be considered as an error that needs to be minimized over all the possible choices of C_i.

In statistics, the square of the error is normally taken as the function to be minimized because, in some applications, the error could take both positive and negative values. Thus, the square of the sum of distances of all the points from their cluster centres needs to be minimized.

This squared error criterion is expressed as

$$E = \sum_{i=1}^{N} \sum_{x \in C} d(x, \bar{x}_i)^2$$

where d is the distance function and $\bar{x}_i$ is the centre of the cluster C_i. The algorithm that finds clusters in such a way that the above sum is minimized would be a good clustering algorithm.

4.4.2 Absolute Error Criterion

The squared error criterion has one significant drawback, it is heavily affected by the presence of outliers (data points with extreme values) in the dataset. The distance of an outlier point from its cluster centre will be quite large. The square of this distance will be even larger.

To avoid the above drawback, squaring the distances of points from their cluster centres can be avoided. The resulting measure is known as the absolute error criterion. It is expressed as

$$E = \sum_{i=1}^{N} \sum_{x \in C} d(x, \bar{x}_i)$$

where d is the distance function (which should always return positive values) and $\bar{x}_i$ is the centre of the cluster C_i. This measure of the clustering quality is relatively immune to the presence of outliers.

4.4.3 Distance between Clusters

For a good clustering algorithm, the distances between its clusters will be large. To determine the distances between the clusters, several approaches exist. Given below are some of the widely used approaches.

Minimum distance This is the distance between the closest pair of points (x, y) where x belongs to one cluster and y belongs to the other.

Maximum distance This is the distance between the farthest pair of points (x, y) where x belongs to one cluster and y belongs to the other.

Average distance This is the average of the distances between every pair of points (x, y) where x belongs to one cluster and y belongs to the other.

Mean distance This is the distance between the means of the two clusters.

4.4.4 Other Performance Metrics

While cluster quality is the most important performance metric, clustering algorithms should also perform well according to the following metrics:

Clustering time This is the amount of time required by the algorithm to find clusters. In several exploratory applications, clustering needs to be done online and in real time. Hence, it is imperative that the clustering time is as small as possible.

Main memory usage The amount of main memory required during clustering is not critical, but should be within practical limits so that it can work on current machines.

Scalability It is important that the clustering algorithm is scalable to handle datasets that contain a huge number of instances and/or attributes.

In addition to the above performance metrics, it is desirable that a clustering algorithm has the following features:

Handle noise/outliers Most applications have noisy data containing data points that do not really belong to any cluster. Such a point could arise because of the errors or missing values in data. It could also be because of an extreme instance (outlier) that really does not lie within any cluster.

Data-order independence Some clustering strategies may be sensitive to the order in which the input records are presented to them and produce different clusters depending on this order. This is undesirable.

Arbitrary-shaped clusters Many clustering algorithms produce clusters whose shape is constrained in some way. For example, they may only produce spherical or convex clusters. This is undesirable. If arbitrary shaped clusters exist in the data, the algorithm should find them.

4.5 Classification of Clustering Algorithms

Clustering algorithms are classified into different categories based on the overall techniques they use and also the kind of clusters they produce. This is useful because when given a clustering problem in a new application domain, the focus can be on a particular category of algorithm, if the kinds of clusters desired are known. Following are the major categories of clustering algorithms:

Partitioning methods These algorithms partition the input data instances into *disjoint spherical clusters*. They are useful in applications where each cluster represents a prototype, and other instances in the cluster are similar to this prototype.

Hierarchical methods In these methods, clusters which are very similar to each other are grouped into larger clusters. These larger clusters may further be grouped into still larger clusters, etc. In essence, a hierarchy of clusters is produced.

Density-based methods In these methods, clusters are 'grown' starting with some data points and then including the other neighbouring points as long as the neighbourhood is sufficiently 'dense'. These methods can find clusters of arbitrary shape and are useful in applications where such a feature is desired.

Grid-based methods In these methods, the space of instances is divided into a grid structure. Clustering techniques are then applied using cells of the grid as the basic units instead of individual data points. This has the advantage of improving the processing time significantly.

Other methods for clustering have been proposed in machine learning and other areas outside the field of data mining. These methods make use of various concepts such as wavelets, neural networks, genetic algorithms, gradient descent, etc. These techniques work like 'black boxes' that are hard to interpret. The clusters produced are, therefore, hard to explain. This is not desirable of data mining techniques, whose main purpose is to produce patterns that are understandable by humans. Due to this reason such algorithms have not been covered in this book.

Sections 4.6–4.9 describe the different clustering algorithms belonging to each of the categories mentioned above. In all these cases, it is assumed that a distance function d is available to compute the distance between data instances.

4.6 Partitioning Methods

These algorithms partition the input data instances into disjoint spherical clusters. They are useful in applications where each cluster represents a prototype, and other instances in the cluster are similar to this prototype.

Partitioning methods require as input k, the number of clusters to produce. This severely limits the applicability of these methods. In most exploratory studies, the number of clusters may not be previously known. In order to know the number of clusters, it may be necessary to first run another kind of a clustering algorithm (such as a hierarchical algorithm) that does not require k as input.

4.6.1 Overall Strategy

Partitioning methods follow an iterative strategy as shown in the pseudo-code of Fig. 4.1 and explained as follows:

```
Partitioning Cluster (Data D, int k)
1. select k prototypes t1, t2...tk, from D
2. for i = 1,..., k: initialize cluster Ci = {ti}
3. repeat:
4. for each point p in D:
5.    let tj be the prototype that minimizes d(t, p)
6.    put p in cluster Cj
7.    quality = ClusteringQuality (C1, C2...Ck)
8.    Re-compute prototypes
9. until quality does not change
```

Fig. 4.1 Clustering using partitioning

A partitioning algorithm starts by selecting k prototypes or representative data points for each cluster (lines 1, 2 of Fig. 4.1). The selection of prototypes is changed at each iteration in a manner that improves the overall *clustering quality* (line 8). Once the prototypes are selected, each data point is put in the cluster whose prototype it resembles the most (lines 4–6). In line 5, the function d represents the distance function used. These iterations continue until the clustering quality does not improve any further (lines 7, 9). In some implementations, the algorithm stops after a fixed maximum number of iterations.

To handle huge datasets, the above basic strategy is applied on small samples of the dataset and the result is used for clustering the entire dataset. Specific partitioning algorithms differ in the details of the above strategy such as how to select the initial k prototypes; how to define the quality of clustering; how to re-compute the prototypes in each iteration; how to utilize sampling for huge datasets.

4.6.2 The k-Means Algorithm

By far, the most popular clustering algorithm used in scientific and industrial applications is the k-means algorithm, which is a partitioning algorithm. Its popularity stems from its simplicity and its firm foundation in statistics.

The Algorithm

In its normal version, the k-means algorithm works only for datasets that consist of numeric attributes. It takes k, the number of desired clusters, and the data points as input and produces k clusters as output. The functioning of the algorithm follows the pseudo-code for the general partitioning strategy given in Fig. 4.1. The details that characterize the algorithm are as follows:

1. First, it selects the initial k prototypes arbitrarily.
2. The squared error criterion in Section 4.4.1 is used to determine the clustering quality.
3. In each iteration, the prototype of each cluster is re-computed to be the *cluster mean* as defined in Section 4.4.1.
4. The basic version of k-means does not include any sampling techniques to scale to huge databases.

Illustration 4.2 Consider the task of clustering people into two clusters based on their heights and weights given Table 4.2.

Table 4.2 Data for k-means example

Id	Name	Height (in)	Weight (kg)
x_1	Ram	64	60
x_2	Shyam	60	61
x_3	Gita	59	70
x_4	Mohan	68	71

Since we want two clusters, $k = 2$. When plotted on a graph, the data looks as shown in Fig. 4.2(a).

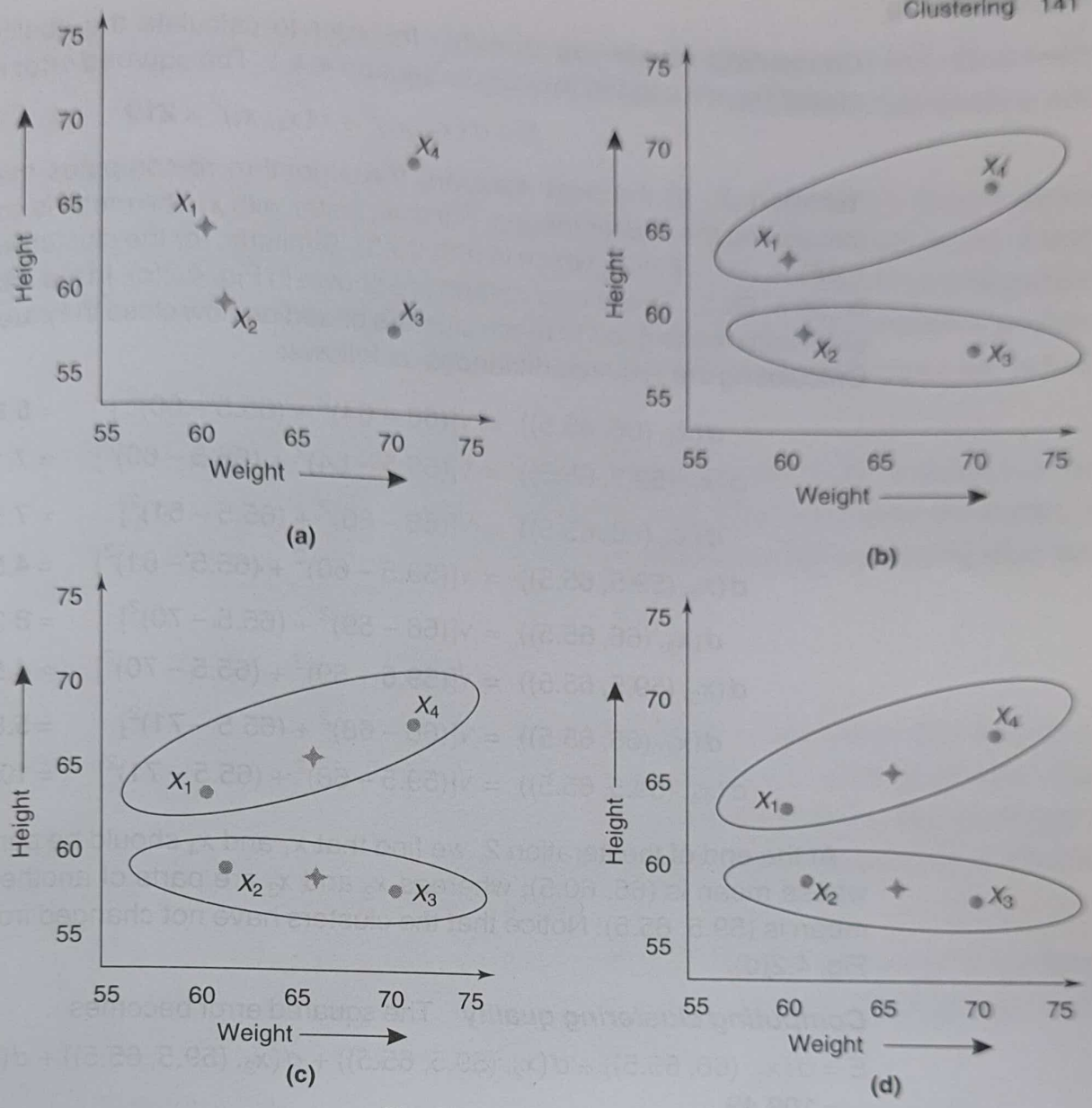

Fig 4.2 *k*-means illustrated

Iteration 1 Suppose the *k*-means algorithm initially selects points x_1 and x_2 as the cluster means [represented as diamonds in Fig. 4.2(a)]. In the next step (lines 4–6 of Fig. 4.1), the algorithm decides which clusters the remaining points (i.e., points x_3 and x_4) should fall into. This decision is based on how close they are to the prototypes. Calculating the euclidean distance, we obtain

$$d(x_3, x_1) = \sqrt{[(59-64)^2 + (70-60)^2]} = 11.18$$

$$d(x_3, x_2) = \sqrt{[(59-60)^2 + (70-61)^2]} = \mathbf{9.06}$$

$$d(x_4, x_1) = \sqrt{[(68-64)^2 + (71-60)^2]} = \mathbf{11.7}$$

$$d(x_4, x_2) = \sqrt{[(68-60)^2 + (71-61)^2]} = 12.81$$

Clearly, x_3 is closer to x_2 than x_1. Hence, it should belong to the cluster for which x_2 is a prototype. Similarly, x_4 should belong to the cluster for which x_1 is a prototype. These cluster assignments are marked in Fig. 4.2(b).

Computing clustering quality In order to calculate the quality of the above clustering, we use the formula in Section 4.4.1. The squared error is obtained as:

$$E = d(x_3, x_2)^2 + d(x_4, x_1)^2 = \mathbf{219}$$

Iteration 2 In the next iteration, the algorithm re-computes the prototypes by calculating the cluster means. For the cluster with x_1, the mean is computed as ((64 + 68)/2, (60 + 71)/2), which is (66, 65.5). Similarly, for the cluster with x_2, the mean is (59.5, 65.5). These new means are shown in Fig. 4.2(c). In the next step, the data points are reassigned to these clusters based on how close they are to their means. Calculating the required distances as follows:

$$\begin{aligned}
d(x_1, (66, 65.5)) &= \sqrt{[(66-64)^2 + (65.5-60)^2\,]} &&= \mathbf{5.85}\\
d(x_1, (59.5, 65.5)) &= \sqrt{[(59.5-64)^2 + (65.5-60)^2]} &&= 7.1\\
d(x_2, (66, 65.5)) &= \sqrt{[(66-60)^2 + (65.5-61)^2]} &&= 7.5\\
d(x_2, (59.5, 65.5)) &= \sqrt{[(59.5-60)^2 + (65.5-61)^2]} &&= \mathbf{4.53}\\
d(x_3, (66, 65.5)) &= \sqrt{[(66-59)^2 + (65.5-70)^2]} &&= 8.32\\
d(x_3, (59.5, 65.5)) &= \sqrt{[(59.5-59)^2 + (65.5-70)^2]} &&= \mathbf{4.53}\\
d(x_4, (66, 65.5)) &= \sqrt{[(66-68)^2 + (65.5-71)^2]} &&= \mathbf{5.85}\\
d(x_4, (59.5, 65.5)) &= \sqrt{[(59.5-68)^2 + (65.5-71)^2]} &&= 10.12
\end{aligned}$$

At the end of the iteration 2, we find that x_1 and x_4 should be parts of the cluster whose mean is (66, 60.5), whereas x_2 and x_3 are parts of another cluster whose mean is (59.5, 65.5). Notice that the clusters have not changed from Fig. 4.2(b) to Fig. 4.2(d).

Computing clustering quality The squared error becomes

$$E = d(x_1, (66, 65.5)) + d(x_2, (59.5, 65.5)) + d(x_3, (59.5, 65.5)) + d(x_4, (66.5, 65.5)) = \mathbf{109.49}$$

Clearly, the squared error is much smaller now than after it was iteration 1. Although the original clusters have not changed at all, the means have been re-computed and hence, the squared error has changed. Further iterations will not yield any improvements in this case.

Analysis In each iteration, the k-means algorithm computes the distances of each point from each of the prototypes. If there are n points and t iterations, then the complexity of the algorithm is $O(nkt)$. Thus, the algorithm is linearly scalable with respect to the data size.

Although the k-means algorithm is popular due to its simplicity and efficiency, it has drawbacks as follows:

Sensitive to bad initial prototypes Illustration 4.2 demonstrates this defect. Visually, the most appropriate clustering in Fig. 4.2(a) is when points 1 and 2

are put in the same cluster and points 3 and 4 are in the other cluster. However, in Illustration 4.2, due to a bad initial selection of prototypes, this result is not obtained.

Prototypes are not actual instances In this case, the computed cluster mean is unlikely to correspond to an actual instance in the dataset; in some applications this is not acceptable. For example, if the task is to categorize people based on several features and to also select a representative person from each cluster, then it is desirable that the selected prototypes are actual persons in the dataset.

Sensitive to outliers The *k*-means algorithm is known to be sensitive to the presence of outliers (points with extreme values). This is because the mean of a population can be affected by the presence of points with extreme (either too large or too small) values.

4.6.3 PAM Algorithm

The PAM (Partitioning Around Medoids) algorithm alleviates the defects of the k-means algorithm at the expense of computational time. It ensures that the prototype of each cluster is an actual instance in the dataset. The prototypes in PAM are also known as *medoids*, to differentiate them from cluster means, which need not be actual data instances.

PAM is actually one among other algorithms that use the strategy of working with medoids–prototypes that are actual data instances. As a group, such algorithms are known as *k*-medoids algorithms.

The Algorithm

Similar to the *k*-means algorithm, PAM only works for datasets that consist of numeric attributes. It takes k, the number of desired clusters, and the data points as input and produces k clusters as output. The functioning of the algorithm follows the pseudo-code for the general partitioning strategy given in Fig. 4.1. The details that characterize the algorithm are as follows:

1. First, it selects the initial k prototypes arbitrarily.
2. The absolute error criterion in Section 4.4.2 is used to determine clustering quality.
3. In each iteration, the prototype of each cluster is reassigned to an actual data point that minimizes the absolute error criterion.
4. PAM does not include any sampling techniques to scale to huge databases.

Illustration 4.3 Consider the task of clustering people into two clusters based on their heights and weights given in Table 4.2. Since we want two clusters, $k = 2$. When plotted on a graph, the data looks as shown in Fig. 4.3(a).

Iteration 1 Suppose the PAM algorithm initially selects points x_1 and x_2 as the cluster means [represented as diamonds in Fig. 4.3(a)]. In the next step (lines 4–6 in Fig. 4.1), the algorithm decides which clusters the remaining points (i.e., points x_3 and x_4) should fall into. This decision is based on how close they are to the prototypes. Calculating the euclidean distance, we obtain:

$$d(x_3, x_1) = \sqrt{[(59-64)^2 + (70-60)^2]} = 11.18$$
$$d(x_3, x_2) = \sqrt{[(59-60)^2 + (70-61)^2]} = \mathbf{9.06}$$
$$d(x_4, x_1) = \sqrt{[(68-64)^2 + (71-60)^2]} = \mathbf{11.7}$$
$$d(x_4, x_2) = \sqrt{[(68-60)^2 + (71-61)^2]} = 12.81$$

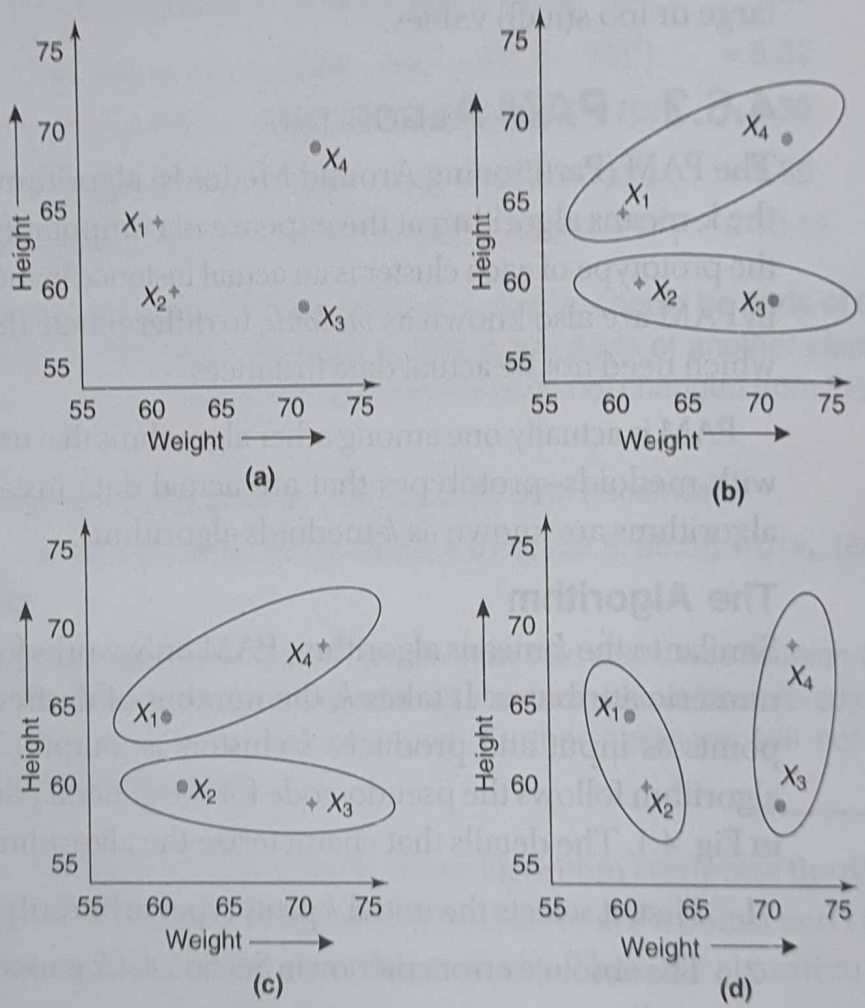

Fig. 4.3 PAM illustrated

Clearly, x_3 is closer to x_2 than x_1. Hence, it should belong to the cluster for which x_2 is a prototype. Similarly, x_4 should belong to the cluster for which x_1 is a prototype. These cluster assignments are marked in Fig. 4.3(b). So far, the algorithm is proceeding exactly as the *k*-means algorithm.

Computing clustering quality In order to calculate the quality of the above clustering, we use the formula in Section 4.4.2. The absolute error is obtained as

$$E = d(x_3, x_2) + d(x_4, x_1) = 20.76$$

Iteration 2 In the next iteration, the algorithm selects new medoids in place of the existing medoids. Either x_3 or x_4 can be selected in place of x_1 or x_2. For each of these four cases, a new clustering and its corresponding quality need to be computed. Next, the new clustering that has the best quality (i.e., minimum absolute error) should be selected.

Consider the case where x_4 is selected in place of x_1. The new medoids are shown in Fig. 4.3(c). In the next step, the data points are reassigned to new clusters based on how close they are to the medoids. Calculating the required distances

$$d(x_1, x_2) = \sqrt{[(60-64)^2 + (61-60)^2]} = 4.123$$
$$d(x_1, x_4) = \sqrt{[(68-64)^2 + (71-60)^2]} = 11.7$$
$$d(x_3, x_2) = \sqrt{[(59-60)^2 + (70-61)^2]} = 9.06$$
$$d(x_3, x_4) = \sqrt{[(59-68)^2 + (70-71)^2]} = 9.05$$

Therefore, at the end of the iteration 2, it is established that x_1 and x_2 should be parts of the same cluster, whereas x_3 and x_4 will be parts of another cluster. The new clustering is shown in Fig. 4.3(d).

Computing clustering quality The absolute error becomes

$$E = d(x_1, x_2) + d(x_3, x_4) = 13.17$$

The above procedure is repeated for each of the other three cases (of replacing x_1 or x_2 with x or x). The case, in which there is least error, is selected. If this minimum error is less than the error during iteration 1, then the corresponding change in medoids is enforced and the next iteration is started. For the current illustration, all the four cases will result in the same new clustering.

Clearly, the error (13.17) in the new clustering is much smaller than the error in the old clustering after iteration 1. The original clusters have changed to be more acceptable. Therefore, at the end of iteration 2, the new choice of medoids is retained. If the new error, at the end of iteration 2, had become worse, then the choice of medoids would have been undone and the previous selection at the beginning of iteration 1 would have been retained.

Analysis The PAM algorithm is much more time-consuming than the k-means algorithm as shown in Theorem 4.1.

Theorem 4.1 (PAM Complexity) The time-complexity of PAM is $O(k(n-k)^2)$ where n is the number of data points, k is the desired number of clusters, and t is the number of iterations in PAM.

Proof In each iteration, the PAM algorithm computes the distances of each point from each of the medoids. This does not incur any extra cost than the *k*-means algorithm (where there were means instead of medoids).

However, for the selection of new medoids, there are multiple possible cases and for each case, the remaining points need to be reallocated to the new medoids. If there are n points, then there are $k(n-k)$ ways of selecting new medoids and the remaining $(n-k)$ non-medoid points need to be reallocated to the medoid points.

Consequently, the complexity of each iteration is $O(k(n-k)^2)$. Hence, the theorem follows.

In spite of the non-linear time complexity of PAM, it is considered as an enhancement over *k*-means because as seen in Illustration 4.3, it is not sensitive to a bad initial selection of prototypes. Further, its prototypes are actual data instances. Finally, it is not so sensitive to the presence of outliers due to two reasons: (1) it does not depend on cluster means (which are sensitive to outliers); (2) it uses the absolute error criterion instead of the squared error criterion.

4.7 Hierarchical Methods

In these methods, clusters which are very similar to each other are grouped into larger clusters. These larger clusters may further be grouped into still larger clusters, etc. In essence, a hierarchy or tree of clusters is produced. This can be pictorially represented by a dendogram as shown in Fig. 4.4.

Example 4.2 Figure 4.4 shows the hierarchical clusters and their dendogram representation for the following data:

$$x_1 = (6, 8);\ x_2 = (5, 7)\ x_3 = (8, 4);\ x_4 = (11, 10);\ x_5 = (12, 8)$$

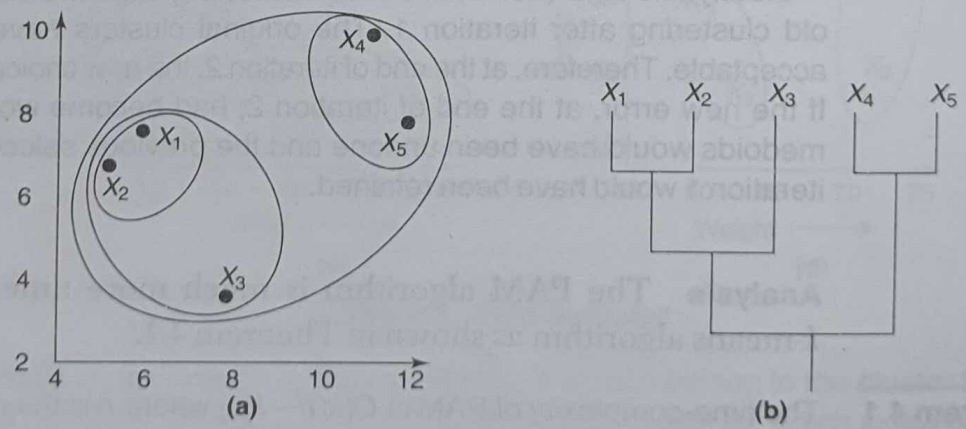

Fig. 4.4 (a) Hierarchical clusters; (b) Dendogram

It is clear from Fig. 4.4(a) that natural clusters exist at many different levels. Depending on the application, x_3 may (or may not) be put in the cluster containing x_1 and x_2.

These clusterings correspond to different levels of the dendogram shown in Fig. 4.4(b). A hierarchical clustering algorithm would output the entire dendogram. In contrast, other algorithms would be constrained to output clusters at some specific level. This would be unnatural and the resulting clustering would not reveal the inherent patterns in the data.

4.7.1 Overall Approaches to Hierarchical Clustering

Hierarchical algorithms may be designed to construct dendograms either top-down or bottom-up. These correspond to the following two basic approaches to hierarchical clustering.

Agglomerative Approach

Agglomerative clustering algorithms start with all the data objects in separate clusters and then merge 'nearby' clusters until a single cluster is formed that contains all the objects. This corresponds to starting at the top of the dendogram in Fig. 4.4(b) and generating the rest of the tree. The pseudo-code for such a procedure is shown in Fig. 4.5 and described below.

```
AgglomerativeCluster (Data D):
1. N = ∅   # dendogram being constructed
2. for each point p in D:
3.         initialize cluster C_i = {p}
4.         N = N ∪ C_i
5. Repeat:
6.         (C_i , C_j ) = closest pair of unmerged clusters in N
7.         C_k = merge(C_i , C_j)
8.         N = N ∪ C_k
9. until no unmerged clusters exist in N
```

Fig. 4.5 Agglomerative hierarchical clustering

The algorithm first initializes each data object (or point) to be in its own cluster and the current dendogram to contain all these clusters (lines 1–4 in Fig. 4.5). Then, in each iteration, it merges the closest pair of clusters (lines 6, 7) and inserts the resulting cluster into the dendogram (line 8). Specific agglomerative clustering algorithms differ in how they define the closeness

between the clusters, as required in step 6. Different approaches for this are discussed in Section 4.7.2.

Divisive Approach

Divisive hierarchical clustering algorithms do the reverse of agglomerative algorithms. They start with all the data objects in a single cluster and then divide the cluster into sub-clusters repeatedly until each object is in its own cluster. This corresponds to starting at the bottom of the dendogram in Fig. 4.4(b) and working upwards to generate the rest of the tree. The pseudo-code for such a procedure is shown in Fig. 4.6 and described below.

The divisive clustering algorithm starts with a cluster containing all the data objects (known as points) in the input dataset D (line 1 in Fig. 4.6). Then, in each iteration, it selects the cluster with largest diameter (line 4)—the diameter of a cluster is the distance between its most distant points. This selected cluster is then split into two parts using the split function shown at the bottom of the figure.

```
DivisiveCluster (Data D):

1. C = cluster containing all points in D
2. N = {C}   # dendogram being constructed
3. repeat:
4.         C_i = leaf node in N with largest diameter
5.         (C_j, C_k) = split(C_i)
6.         Insert C_j, C_k in N as children of C_i
7. until all leaf nodes in N contain only a single point

split (Cluster C_i):

1. p = point in C_i that is most distant from other points in C_i
2. C_j = set of points in C_i that are closer to p than to C_i
3. C_k = C_i − C_j
4. return (C_j, C_k)
```

Fig. 4.6 Divisive hierarchical clustering

The split function finds the point p in the cluster, which is furthest from all the other points in the cluster (line 1 of the split function). This distance is usually calculated as the average distance of each point to all the other points in the cluster. The points in the cluster are then split into two groups based on whether they are closer to the most distant point p, or to the old cluster (lines 2, 3). Specific divisive clustering algorithms may differ in how they define the closeness between a point and a cluster, as required in this step.