Unsupervised Learning

7

7.1 Introduction

In machine learning, the problem of unsupervised learning is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate the goodness of a potential solution. This distinguishes unsupervised from supervised learning. Unsupervised learning is defined as the task performed by algorithms that learn from a training set of unlabeled or unannotated examples, using the features of the inputs to categorize them according to some geometric or statistical criteria.

Unsupervised learning encompasses many techniques that seek to summarize and explain key features or structures of the data. Many methods employed in unsupervised learning are based on data mining methods used to preprocess data. Most unsupervised learning techniques can be summarized as those that tackle the following four groups of problems:

- *Clustering*: has as a goal to partition the set of examples into groups.
- *Dimensionality reduction*: aims to reduce the dimensionality of the data. Here, we encounter techniques such as Principal Component Analysis (PCA), independent component analysis, and nonnegative matrix factorization.
- *Outlier detection*: has as a purpose to find unusual events (e.g., a malfunction), that distinguish part of the data from the rest according to certain criteria.
- *Novelty detection*: deals with cases when changes occur in the data (e.g., in streaming data).

The most common unsupervised task is clustering, which we focus on in this chapter.

© Springer International Publishing Switzerland 2017

L. Igual and S. Seguí, *Introduction to Data Science*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-50017-1_7

7.2 Clustering

Clustering is a process of grouping similar objects together; i.e., to partition unlabeled examples into disjoint subsets of clusters, such that:

- Examples within a cluster are similar (in this case, we speak of *high intraclass similarity*).
- Examples in different clusters are different (in this case, we speak of *low interclass similarity*).

When we denote data as similar and dissimilar, we should define a measure for this similarity/dissimilarity. Note that grouping similar data together can help in discovering new categories in an unsupervised manner, even when no sample category labels are provided. Moreover, two kinds of inputs can be used for grouping:

(a) in *similarity-based clustering*, the input to the algorithm is an $n \times n$ *dissimilarity matrix* or *distance matrix*;
(b) in *feature-based clustering*, the input to the algorithm is an $n \times D$ *feature matrix* or *design matrix*, where n is the number of examples in the dataset and D the dimensionality of each sample.

Similarity-based clustering allows easy inclusion of domain-specific similarity, while feature-based clustering has the advantage that it is applicable to potentially noisy data.

Therefore, several questions regarding the clustering process arise.

- What is a natural grouping among the objects? We need to define the "groupness" and the "similarity/distance" between data.
- How can we group samples? What are the best procedures? Are they efficient? Are they fast? Are they deterministic?
- How many clusters should we look for in the data? Shall we state this number a priori? Should the process be completely data driven or can the user guide the grouping process? How can we avoid "trivial" clusters? Should we allow final clustering results to have very large or very small clusters? Which methods work when the number of samples is large? Which methods work when the number of classes is large?
- What constitutes a good grouping? What objective measures can be defined to evaluate the quality of the clusters?

There is not always a single or optimal answer to these questions. It used to be said that clustering is a "subjective" issue. Clustering will help us to describe, analyze, and gain insight into the data, but the quality of the partition depends to a great extent on the application and the analyst.

7.2.1 Similarity and Distances

To speak of similar and dissimilar data, we need to introduce a notion of the similarity of data. There are several ways for modeling of similarity. A simple way to model this is by means of a Gaussian kernel:

$$s(a, b) = e^{-\gamma d(a,b)}$$

where $d(a, b)$ is a metric function, and γ is a constant that controls the decay of the function. Observe that when $a = b$, the similarity is maximum and equal to one. On the contrary, when a is very different to b, the similarity tends to zero. The former modeling of the similarity function suggests that we can use the notion of distance as a surrogate. The most widespread distance metric is the *Minkowski distance*:

$$d(a, b) = (\sum_{i=1}^{d} |a_i - b_i|^p)^{1/p}$$

where $d(a, b)$ stands for the distance between two elements $a, b \in \mathbb{R}^d$, d is the dimensionality of the data, and p is a parameter.

The best-known instantiations of this metric are as follows:

- when $p = 2$, we have the *Euclidean distance*,
- when $p = 1$, we have the *Manhattan distance*, and
- when $p = \inf$, we have the *max-distance*. In this case, the distance corresponds to the component $|a_i - b_i|$ with the highest value.

7.2.2 What Constitutes a Good Clustering? Defining Metrics to Measure Clustering Quality

When performing clustering, the question normally arises: How do we measure the quality of the clustering result? Note that in unsupervised clustering, we do not have groundtruth labels that would allow us to compute the accuracy of the algorithm. Still, there are several procedures for assessing quality. We find two families of techniques: those that allow us to compare clustering techniques, and those that check on specific properties of the clustering, for example "compactness".

7.2.2.1 Rand Index, Homogeneity, Completeness and V-measure Scores

One of the best-known methods for comparing the results in clustering techniques in statistics is the *Rand index* or Rand measure (named after William M. Rand). The Rand index evaluates the similarity between two results of data clustering. Since in unsupervised clustering, class labels are not known, we use the Rand index to compare the coincidence of different clusterings obtained by different approaches or criteria. As an alternative, we later discuss the *Silhouette coefficient*: instead of

comparing different clusterings, this evaluates the compactness of the results of applying a specific clustering approach.

Given a set of n elements $S = \{o_1, \ldots, o_n\}$, we can compare two partitions of S[1]: $X = \{X_1, \ldots, X_r\}$, a partition of S into r subsets; and $Y = \{Y_1, \ldots, , Y_s\}$, a partition of S into s subsets. Let us use the annotations as follows:

- a is the number of pairs of elements in S that are in the same subset in both X and Y;
- b is the number of pairs of elements in S that are in different subsets in both X and Y;
- c is the number of pairs of elements in S that are in the same subset in X, but in different subsets in Y; and
- d is the number of pairs of elements in S that are in different subsets in X, but in the same subset in Y.

The Rand index, R, is defined as follows:

$$R = \frac{a+b}{a+b+c+d},$$

ensuring that its value is between 0 and 1.

One of the problems of the Rand index is that when given two datasets with random labelings, it does not take a constant value (e.g., zero) as expected. Moreover, when the number of clusters increases it is desirable that the upper limit tends to the unity. To solve this problem, a form of the Rand index, called the *Adjusted Rand index*, is used that adjusts the Rand index with respect to chance grouping of elements. It is defined as follows:

$$AR = \frac{\binom{n}{2}(a+d) - [(a+b)(a+c)+(c+d)(b+d)]}{\binom{n}{2}^2 [(a+b)(a+c)+(c+d)(b+d)]}.$$

Another way for comparing clustering results is the V-measure. Let us first introduce some concepts. We say that a clustering result satisfies a *homogeneity* criterion if all of its clusters contain only data points which are members of the same original (single) class. A clustering result satisfies a *completeness* criterion if all the data points that are members of a given class are elements of the same predicted cluster. Note that both scores have real positive values between 0.0 and 1.0, larger values being desirable. For example, if we consider two toy clustering sets (e.g., original and predicted) with four samples and two labels, we get:

In [1]:
```
print("%.3f" % metrics.homogeneity_score([0, 0, 1, 1],
                                         [0, 0, 0, 0]))
```

Out[1]:
```
0.000
```

[1] https://en.wikipedia.org/wiki/Rand_index.

The homogeneity is 0 since the samples in the predicted cluster 0 come from original cluster 0 and cluster 1.

In [2]:
```
print metrics.completeness_score([0, 0, 1, 1],
                                 [1, 1, 0, 0])
```

Out[2]:
```
1.0
```

The completeness is 1 since all the samples from the original cluster with label 0 go into the same predicted cluster with label 1, and all the samples from the original cluster with label 1 go into the same predicted cluster with label 0.

However, how can we define a measure that takes into account the completeness as well as the homogeneity? The *V-measure* is the harmonic mean between the homogeneity and the completeness defined as follows:

$$v = 2 * (\text{homogeneity} * \text{completeness})/(\text{homogeneity} + \text{completeness}).$$

Note that this metric is not dependent of the absolute values of the labels: a permutation of the class or cluster label values will not change the score value in any way. Moreover, the metric is symmetric with respect to switching between the predicted and the original cluster label. This is very useful to measure the agreement of two independent label assignment strategies applied to the same dataset even when the real groundtruth is not known. If class members are completely split across different clusters, the assignment is totally incomplete, hence the V-measure is null:

In [3]:
```
print("%.3f" % metrics.v_measure_score([0, 0, 0, 0],
                                       [0, 1, 2, 3]))
```

Out[3]:
```
0.000
```

In contrast, clusters that include samples from different classes destroy the homogeneity of the labeling, hence:

In [4]:
```
print("%.3f" % metrics.v_measure_score([0, 0, 1, 1],
                                       [0, 0, 0, 0]))
```

Out[4]:
```
0.000
```

In summary, we can say that the advantages of the V-measure include that it has bounded scores: 0.0 means the clustering is extremely bad; 1.0 indicates a perfect clustering result. Moreover, it can be interpreted easily: when analyzing the V-measure, low completeness or homogeneity explain in which direction the clustering is not performing well. Furthermore, we do not assume anything about the cluster structure. Therefore, it can be used to compare clustering algorithms such as *K-means*, which assume isotropic blob shapes, with results of other clustering algorithms such as spectral clustering (see Sect. 7.2.3.2), which can find clusters with "folded" shapes. As a drawback, the previously introduced metrics are not normalized with regard to random labeling. This means that depending on the number of samples, clusters and groundtruth classes, a completely random labeling will

not always yield the same values for homogeneity, completeness and hence, the V-measure. In particular, random labeling will not yield a zero score, and they will tend further from zero as the number of clusters increases. It can be shown that this problem can reliably be overcome when the number of samples is high, i.e., more than a thousand, and the number of clusters is less than 10. These metrics require knowledge of the groundtruth classes, while in practice this information is almost never available or requires manual assignment by human annotators. Instead, as mentioned before, these metrics can be used to compare the results of different clusterings.

7.2.2.2 Silhouette Score

An alternative to the former scores is to evaluate the final 'shape' of the clustering result. This is the underlying idea behind the Silhouette coefficient. It is defined as a function of the intracluster distance of a sample in the dataset, a and the nearest-cluster distance, b for each sample.[2] Later, we will discuss different ways to compute the distance between clusters. The Silhouette coefficient for a sample i can be written as follows:

$$Silhouette(i) = \frac{b - a}{\max(a, b)}.$$

Hence, if the Silhouette $s(i)$ is close to 0, it means that the sample is on the border of its cluster and the closest one from the rest of the dataset clusters. A negative value means that the sample is closer to the neighbor cluster. The average of the Silhouette coefficients of all samples of a given cluster defines the "goodness" of the cluster. A high positive value, i.e., close to 1 would mean a compact cluster, and vice versa. And the average of the Silhouette coefficients of all clusters gives idea of the quality of the clustering result. Note that the Silhouette coefficient only makes sense when the number of labels predicted is less than the number of samples clustered.

The advantage of the Silhouette coefficient is that it is bounded between -1 and $+1$. Moreover, it is easy to show that the score is higher when clusters are dense and well separated; a logical feature when speaking about clusters. Furthermore, the Silhouette coefficient is generally higher when clusters are compact.

7.2.3 Taxonomies of Clustering Techniques

Within different clustering algorithms, one can find *soft partition* algorithms, which assign a probability of the data belonging to each cluster, and also *hard partition* algorithms, where each datapoint is assigned precise membership of one cluster. A typical example of a soft partition algorithm is the Mixture of Gaussians [1], which can be viewed as a density estimator method that assigns a confidence or

[2]The intracluster distance of sample i is obtained by the distance of the sample to the nearest sample from the same class, and the nearest-cluster distance is given by the distance to the closest sample from the cluster nearest to the cluster of sample i.

probability to each point in the space. A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. The universally used generative unsupervised clustering using a Gaussian mixture model is also known as *EM Clustering*. Each point in the dataset has a soft assignment to the K clusters. One can convert this soft probabilistic assignment into membership by picking out the most likely clusters (those with the highest probability of assignment).

An alternative to soft algorithms are the *hard partition* algorithms, which assign a unique cluster value to each element in the feature space. According to the grouping process of the hard partition algorithm, there are two large families of clustering techniques:

- *Partitional algorithms*: these start with a random partition and refine it iteratively. That is why sometimes these algorithms are called "flat" clustering. In this chapter, we will consider two partitional algorithms in detail: K-means and *spectral clustering*.
- *Hierarchical algorithms*: these organize the data into hierarchical structures, where data can be agglomerated in the bottom-up direction, or split in a top-down manner. In this chapter, we will discuss and illustrate *agglomerative clustering*.

A typical hard partition algorithm is K-means clustering. We will now discuss it in some detail.

7.2.3.1 K-means Clustering

K-means algorithm is a hard partition algorithm with the goal of assigning each data point to a single cluster. K-means algorithm divides a set of n samples X into k disjoint clusters c_i, $i = 1, \ldots, k$, each described by the mean μ_i of the samples in the cluster. The means are commonly called cluster *centroids*. The K-means algorithm assumes that all k groups have equal variance.

K-means clustering solves the following minimization problem:

$$\arg min_c \sum_{j=1}^{k} \sum_{x \in c_j} d(x, \mu_j) = \arg min_c \sum_{j=1}^{k} \sum_{x \in c_j} ||x - \mu_j||_2^2 \tag{7.1}$$

where c_i is the set of points that belong to cluster i and μ_i is the center of the class c_i. K-means clustering objective function uses the square of the Euclidean distance $d(x, \mu_j) = ||x - \mu_j||^2$, that is also referred to as the *inertia* or *within-cluster sum-of-squares*. This problem is not trivial to solve (in fact, it is NP-hard problem), so the algorithm only hopes to find the global minimum, but may become stuck at a different solution.

In other words, we may wonder whether the centroids should belong to the original set of points:

$$inertia = \sum_{i=0}^{n} min_{\mu_j \in c}(||x_i - \mu_j||^2)). \tag{7.2}$$

The K-means algorithm, also known as Lloyd's algorithm, is an iterative procedure that searches for a solution of the K-means clustering problem and works as follows. First, we need to decide the number of clusters, k. Then we apply the following procedure:

1. Initialize (e.g., randomly) the k cluster centers, called *centroids*.
2. Decide the class memberships of the n data samples by assigning them to the nearest-cluster centroids (e.g., the center of gravity or mean).
3. Re-estimate the k cluster centers, c_i, by assuming the memberships found above are correct.
4. If none of the n objects changes its membership from the last iteration, then exit. Otherwise go to step 2.

Let us illustrate the algorithm in Python. First, we will create three sample distributions:

In [5]:
```
MAXN = 40
X = np.concatenate([
    1.25*np.random.randn(MAXN, 2),
    5 + 1.5*np.random.randn(MAXN, 2)])
X = np.concatenate([
    X, [8, 3] + 1.2*np.random.randn(MAXN, 2)])
```

The sample distributions generated are shown in Fig. 7.1 (left). However, the algorithm is not aware of their distribution. Figure 7.1 (right) shows what the algorithm sees. Let us assume that we expect to have three clusters ($k = 3$) and apply the K-means command from the Scikit-learn library:

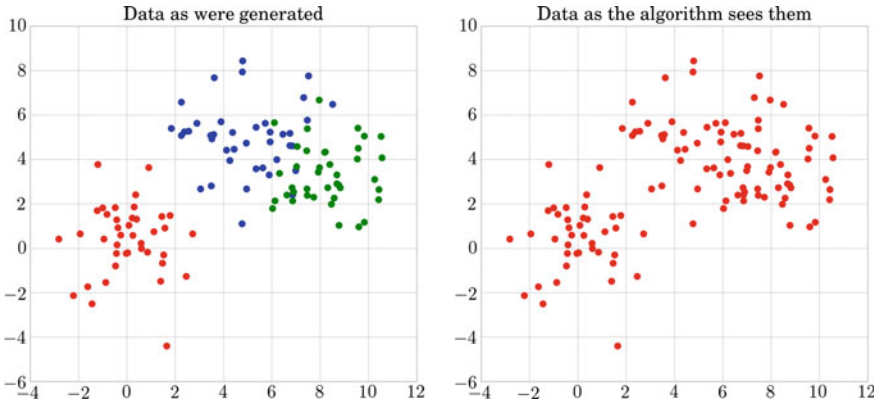

Fig. 7.1 Initial samples as generated (*left*), and samples seen by the algorithm (*right*)

In [6]:
```
from sklearn import cluster

K = 3 # Assuming we have 3 clusters!
clf = cluster.KMeans(init = 'random', n_clusters = K)
clf.fit(X)
```

Out[6]:
```
KMeans(copy_x=True, init='random', max_iter=300,
n_clusters=3, n_init=10, n_jobs=1, precompute_distances=True,
random_state=None, tol=0.0001, verbose=0)
```

Each clustering algorithm in Scikit-learn is used as follows. First, an object from the clustering technique is instantiated. Then we can use the `fit` method to adjust the learning parameters. We also find the method `predict` that, given new data, returns the cluster they belong to. For the class, the labels over the training data can be found in the `labels_` attribute or alternatively they can be obtained using the predict method.

How many "mis-clusterings" do we have? In order to see this, we tessellate the space and color all grid points from the same cluster with the same color. Then, we overlay the initial sample distributions (see Fig. 7.2). In the ideal case, we expect that in each partitioned subspace the sample points are of the same color. However, as shown in Fig. 7.2, the resulting clustering, which is represented in the figure by the color subspace in gray, does not usually coincide exactly with the initial distribution, which is represented by the color of the data. For example, in the same figure, if most of the blue points belong to the same cluster, there are a few ones that belong to the space occupied by the green data.

When computing the Rand index, we get:

In [7]:
```
print ('The Adjusted Rand index is: %.2f' %
       metrics.adjusted_rand_score(y.ravel(), clf.labels_))
```

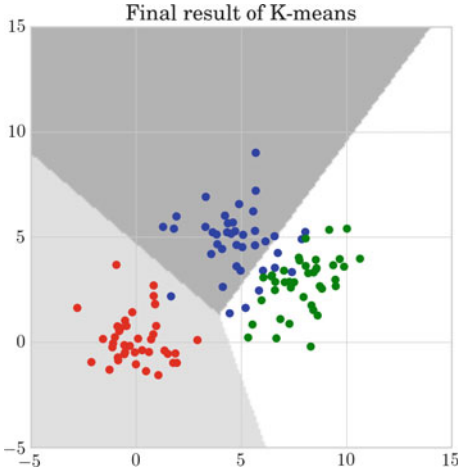

Fig. 7.2 Original samples (*dots*) generated by three distributions and the partition of the space according to the K-means clustering

```
Out[7]: The Adjusted Rand index is: 0.66
```

Taking into account that the Adjusted Rand index belongs to the interval [0, 1], the result of 0.66 in our example means that although most of the clusters were discovered, not 100% of them were; as confirmed by Fig. 7.2.

The inertia can be seen as a measure of how internally coherent the clusters are. Several issues should be taken into account:

- The inertia assumes that clusters are isotropic and convex, since the Euclidean distance is applied, which is isotropic with regard to the different dimensions of the data. However, we cannot expect that the data fulfill this assumption by default. Hence, the K-means algorithm responds poorly to elongated clusters or manifolds with irregular shapes.
- The algorithm may not ensure convergence to the global minimum. It can be shown that K-means will always converge to a local minimum of the inertia (Eq. (7.2)). It depends on the random initialization of the seeds, but some seeds can result in a poor convergence rate, or convergence to suboptimal clustering. To alleviate the problem of local minima, the K-means computation is often performed several times, with different centroid initializations. One way to address this issue is the `k-means++` initialization scheme, which has been implemented in `Scikit-learn` (use the `init='kmeans++'` parameter). This parameter initializes the centroids to be (generally) far from each other, thereby probably leading to better results than random initialization.
- This algorithm requires the number of clusters to be specified. Different heuristics can be applied to predetermine the number of seeds of the algorithm.
- It scales well to a large number of samples and has been used across a large range of application areas in many different fields.

In summary, we can conclude that K-means has the advantages of allowing the easy use of heuristics to select good seeds; initialization of seeds by other methods; multiple points to be tried. However, in contrast, it still cannot ensure that the local minima problem is overcome; it is iterative and hence slow when there are a lot of high-dimensional samples; and it tends to look for spherical clusters.

7.2.3.2 Spectral Clustering

Up to this point, the clustering procedure has been considered as a way to find data groups following a notion of compactness. Another way of looking at what a cluster is is provided by connectivity (or similarity). Spectral clustering [2] refers to a family of methods that use spectral techniques. Specifically, these techniques are related to the eigendecomposition of an affinity or similarity matrix and solve the problem of clustering according to the connectivity of the data. Let us consider an ideal similarity matrix of two clear sets.

Let us denote the similarity matrix, S, as the matrix $S_{ij} = s(x_i, x_j)$ which gives the similarity between observations x_i and x_j. Remember that we can model similarity

using the Euclidean distance, $d(x_i, x_j) = ||x_i - x_j||^2$, by means of a Gaussian Kernel as follows:

$$s(x_i, x_j) = exp(-\alpha||x_i - x_j||^2),$$

where α is a parameter. We expect two points from different clusters to be far away from each other. However, if there is a sequence of points within the cluster that forms a "path" between them, this also would lead to big distance among some of the points from the same cluster. Hence, we define an affinity matrix A based on the similarity matrix S, where A contains positive values and is symmetric. This can be done, for example, by applying a k-nearest neighbor that builds a graph connecting just the k closest data points. The symmetry comes from the fact that A_{ij} and A_{ji} give the distance between the same points. Considering the affinity matrix, the clustering can be seen as a graph partition problem, where connected graph components correspond to clusters. The graph obtained by spectral clustering will be partitioned so that graph edges connecting different clusters have low weights, and vice versa. Furthermore, we define a degree matrix D, where each diagonal value is the degree of the respective graph node and all other elements are 0. Finally, we can compute the unnormalized graph Laplacian ($U = D - A$) and/or a normalized version of the Laplacian (L), as follows:

- *Simple Laplacian*: $L = I - D^{-1}A$, which corresponds to a random walk, being D^{-1} the transition matrix. Spectral clustering obtains groups of nodes such that the random walk corresponds to seldom transitions from one group to another.
- *Normalized Laplacian*: $L = D^{-\frac{1}{2}}UD^{-\frac{1}{2}}$.
- *Generalized Laplacian*: $L = D^{-1}U$.

If we assume that there are k clusters, the next step is to find the k smallest eigenvectors, without considering the trivial constant eigenvector. Each row of the matrix formed by the k smallest eigenvectors of the Laplacian matrix defines a transformation of the data x_i. Thus, in this transformed space, we can apply K-means clustering in order to find the final clusters. If we do not know in advance the number of clusters, k, we can look for sudden changes in the sorted eigenvalues of the matrix, U, and keep the smallest ones.

7.2.3.3 Hierarchical Clustering

Another well-known clustering technique of particular interest is hierarchical clustering. Hierarchical clustering is comprised of a general family of clustering algorithms that construct nested clusters by successive merging or splitting of data. The hierarchy of clusters is represented as a tree. The tree is usually called a *dendrogram*. The root of the dendrogram is the single cluster that contains all the samples; the leaves are the clusters containing only one sample each. This is a nice tool, since it can be straightforwardly interpreted: it "explains" how clusters are formed and visualizes clusters at different scales. The tree that results from the technique shows

the similarity between the samples. Partitioning is computed by selecting a cut on the tree at a certain level.

In general, there are two types of hierarchical clustering:

- *Top-down* divisive clustering applies the following algorithm:
  - Start with all the data in a single cluster.
  - Consider every possible way to divide the cluster into two.
  - Choose the best division.
  - Recursively, it operates on both sides until a stopping criterion is met. That can be something as follows: there are as much clusters as data; the predetermined number of clusters has been reached; the maximum distance between all possible partition divisions is smaller than a predetermined threshold; etc.
- *Bottom-up* agglomerative clustering applies the following algorithm:
  - Start with each data point in a separate cluster.
  - Repeatedly join the closest pair of clusters.
  - At each step, a stopping criterion is checked: there is only one cluster; a predetermined number of clusters has been reached; the distance between the closest clusters is greater than a predetermined threshold; etc.

  This process of merging forms a binary tree or hierarchy.

When merging two clusters, a question naturally arises: How to measure the similarity of two clusters? There are different ways to define this with different results for the agglomerative clustering. The linkage criterion determines the metric used for the cluster merging strategy:

- *Maximum* or *complete* linkage minimizes the maximum distance between observations of pairs of clusters. Based on the similarity of the two least similar members of the clusters, this clustering tends to give tight spherical clusters as a final result.
- *Average* linkage averages similarity between members, i.e., minimizes the average of the distances between all observations of pairs of clusters.
- *Ward* linkage minimizes the sum of squared differences within all clusters. It is thus a variance-minimizing approach and in this sense is similar to the K-means objective function, but tackled with an agglomerative hierarchical approach.

Let us illustrate how the different linkages work with an example. Let us generate three clusters as follows:

In [8]:
```
MAXN1 = 500
MAXN2 = 400
MAXN3 = 300
X1 = np.concatenate([
     2.25 * np.random.randn(MAXN1, 2),
     4 + 1.7*np.random.randn(MAXN2, 2)])
X1 = np.concatenate([
     X1, [8, 3] + 1.9*np.random.randn(MAXN3, 2)])

y1 = np.concatenate([
     np.ones((MAXN1, 1)),
     2 * np.ones((MAXN2, 1))])
y1 = np.concatenate([
     y1, 3 * np.ones((MAXN3, 1))]).ravel()
y1 = np.int_(y1)
labels_y1 = ['+', '*', 'o']
colors = ['r', 'g', 'b']
```

Let us apply agglomerative clustering using the different linkages:

In [9]:
```
from sklearn.cluster import AgglomerativeClustering

for linkage in ('ward', 'complete', 'average'):
    clustering = AgglomerativeClustering(linkage = linkage,
                                         n_clusters =3)
    clustering.fit(X1)

    x_min , x_max = np.min (X1, axis = 0) , np.max (X1,axis
        = 0)
    X1 = (X1 - x_min ) / ( x_max - x_min )
    plt.figure ( figsize =(5 , 5))
    for i in range (X1.shape [0]) :
        plt.text(X1[i, 0], X1[i, 1], labels_y1[y1[i]-1],
                 color = colors[y1[i]-1])
    plt.title("\%s linkage " \% linkage, size  = 20)
    plt.tight_layout()

plt.show()
```

The results of the agglomerative clustering using the different linkages: complete, average, and Ward are given in Fig. 7.3. Note that agglomerative clustering exhibits "rich get richer" behavior that can sometimes lead to uneven cluster sizes, with average linkage being the worst strategy in this respect and Ward linkage giving the most regular sizes. Ward linkage is an attempt to form clusters that are as compact as possible, since it considers inter- and intra-distances of the clusters. Meanwhile, for non-Euclidean metrics, average linkage is a good alternative. Average linkage can produce very unbalanced clusters, it can even separate a single data point into a separate cluster. This fact would be useful if we want to detect outliers, but it may be undesirable when two clusters are very close to each other, since it would tend to merge them.

Agglomerative clustering can scale to a large number of samples when it is used jointly with a connectivity matrix, but it is computationally expensive when no con-

nectivity constraints are added between samples: it considers all the possible merges at each step.

7.2.3.4 Adding Connectivity Constraints

Sometimes, we are interested in introducing a connectivity constraint into the clustering process so that merging of nonadjacent points is avoided. This can be achieved by constructing a connectivity matrix that defines which are the neighboring samples in the dataset. For instance, in the example in Fig. 7.4, we want to avoid the formation of clusters of samples from the different circles. A sample code to compute agglomerative clustering with connectivity would be as follows:

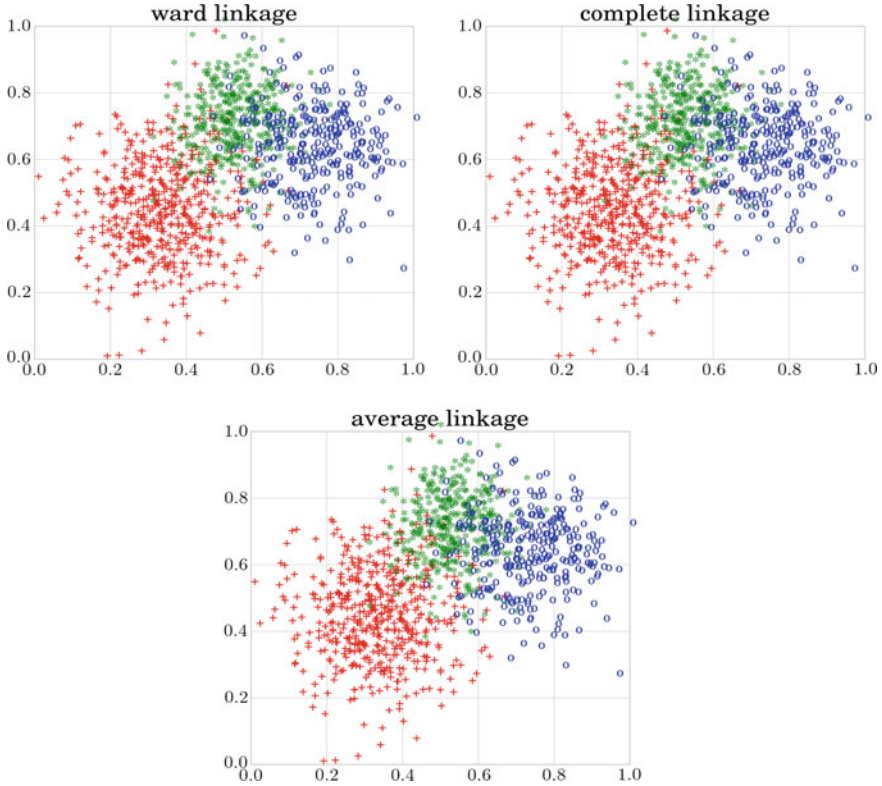

Fig. 7.3 Illustration of agglomerative clustering using different linkages: Ward, complete, and average. The symbol of each data point corresponds to the original class generated and the color corresponds to the cluster obtained

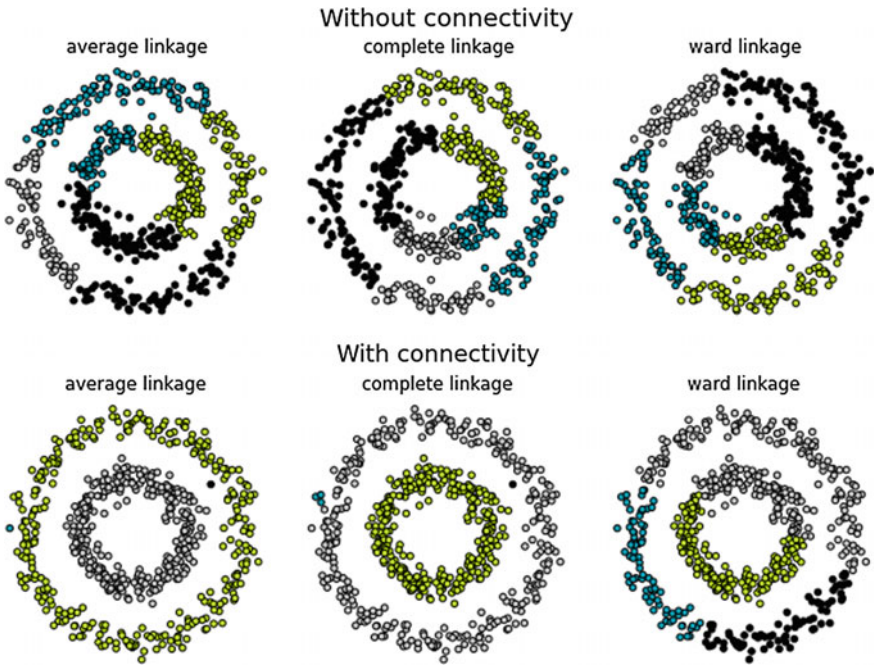

Fig. 7.4 Illustration of agglomerative clustering without (*top row*) and with (*bottom row*) a connectivity graph using the three linkages (from *left* to *right*): average, complete, and Ward. The *colors* correspond to the clusters obtained

In [10]:
```
connectivity = kneighbors_graph(X, 30)
model = AgglomerativeClustering(linkage = 'average',
    connectivity = connectivity, n_clusters = 8)
model.fit(X)
```

A connectivity constraint is useful to impose a certain local structure, but it also makes the algorithm faster, especially when the number of the samples is large. A connectivity constraint is imposed via a *connectivity matrix*: a sparse matrix that only has elements at the intersection of a row and a column with indexes of the dataset that should be connected. This matrix can be constructed from a priori information or can be learned from the data, for instance using `kneighbors_graph` to restrict merging to nearest neighbors or using `image.grid_to_graph` to limit merging to neighboring pixels in an image, both from Scikit-learn. This phenomenon can be observed in Fig. 7.4, where in the first row we see the results of the agglomerative clustering without using a connectivity graph. The clustering can join data from different circles (e.g., the black cluster). At the bottom, the three linkages use a connectivity graph and thus two of them avoid joining data points that belong to different circles (except the Ward linkage that attempts to form compact and isotropic clusters).

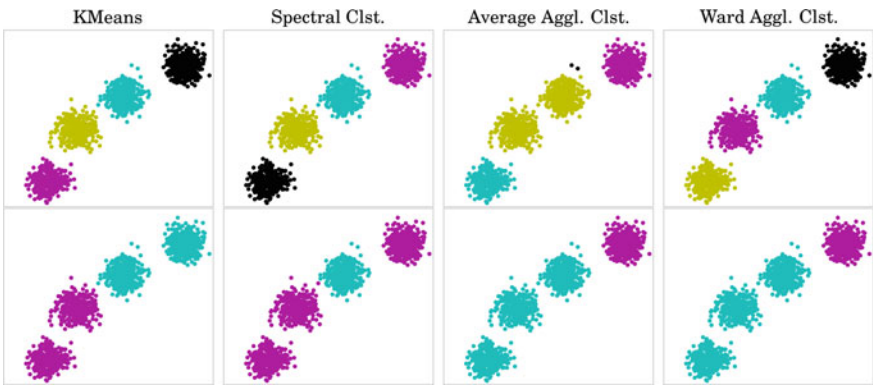

Fig. 7.5 Comparison of the different clustering techniques (from *left* to *right*): K-means, spectral clustering, and agglomerative clustering with average and Ward linkage on simple compact datasets. In the *first row*, the expected number of clusters is $k = 2$ and in the *second row*: $k = 4$

7.2.3.5 Comparison of Different Hard Partition Clustering Algorithms

Let us compare the behavior of the different clustering algorithms discussed so far. For this purpose, we generate three different datasets' configurations:

(a) 4 spherical groups of data;
(b) a uniform data distribution; and
(c) a non-flat configuration of data composed of two moon-like groups of data.

An easy way to generate these datasets is by using `Scikit-learn` that has predefined functions for it: `datasets.make_blobs()`, `datasets.ma- ke_ moons()`, etc.

We apply the clustering techniques discussed above, namely K-means, agglomerative clustering with average linkage, agglomerative clustering with Ward linkage, and spectral clustering. Let us test the behavior of the different algorithms assuming $k = 2$ and $k = 4$. Connectivity is applied in the algorithms where it is applicable.

In the simple case of separated clusters of data and $k = 4$, most of the clustering algorithms perform well, as expected (see Fig. 7.5). The only algorithm that could not discover the four groups of samples is the average agglomerative clustering. Since it allows highly unbalanced clusters, the two noisy data points that are quite separated from the closest two blobs were considered as a different cluster, while the two central blobs were merged in one cluster. In case of $k = 2$, each of the methods is obligated to join at least two blobs in a cluster.

Regarding the uniform distribution of data (see Fig. 7.6), K-means, Ward linkage agglomerative clustering and spectral clustering tend to yield even and compact clusters; while the average linkage agglomerative clustering attempts to join close points as much as possible following the "rich get richer" rule. This results in a

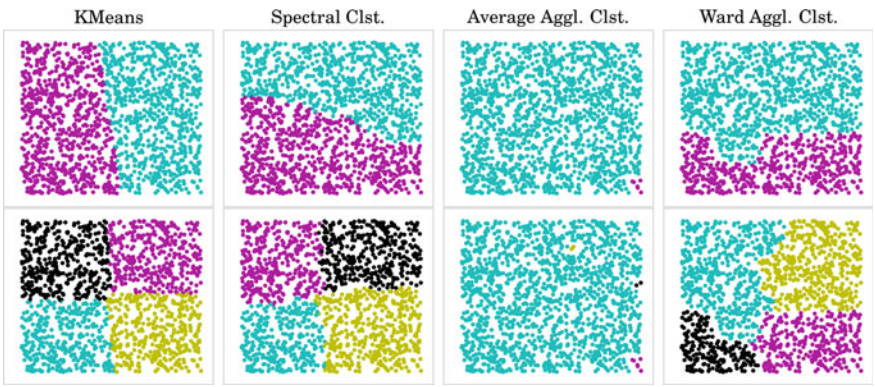

Fig. 7.6 Comparison of the different clustering techniques (from *left* to *right*): K-means, spectral clustering, and agglomerative clustering with average and Ward linkage on uniformly distributed data. In the *first row*, the number of clusters assumed is $k = 2$ and in the *second row*: $k = 4$

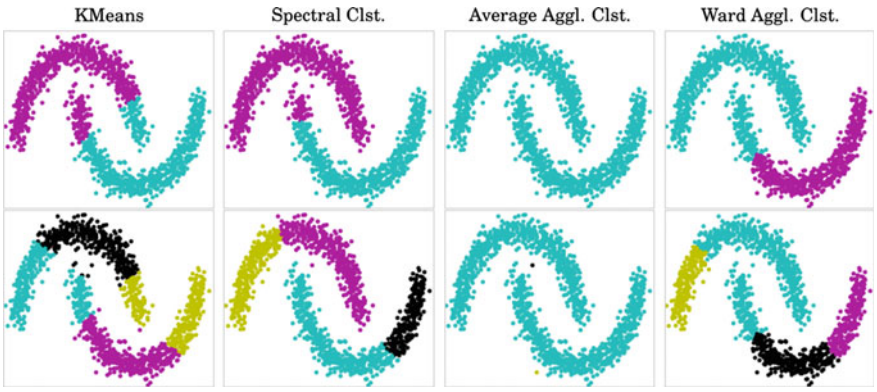

Fig. 7.7 Comparison of the different clustering techniques (from *left* to *right*): K-means, spectral clustering, and agglomerative clustering with average and Ward linkage on non-flat geometry datasets. In the *first row*, the expected number of clusters is $k = 2$ and in the *second row*: $k = 4$

second cluster of a small set of data. This behavior is observed in both cases: $k = 2$ and $k = 4$.

Regarding datasets with more complex geometry, like in the moon dataset (see Fig. 7.7), K-means and Ward linkage agglomerative clustering attempt to construct compact clusters and thus cannot separate the moons. Due to the connectivity constraint, the spectral clustering and the average linkage agglomerative clustering separated both moons in case of $k = 2$, while in case of $k = 4$, the average linkage agglomerative clustering clustered most of datasets correctly separating some of the noisy data points as two separate single clusters. In the case of spectral clustering, looking for four clusters, the method splits each of the two moon datasets into two clusters.

	0	1	2	3	4	5	6	7	8	9	10	11
GEO												
Albania	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Austria	0.52	5.25	9.98	0.64	1.22	0.61	1.01	2.64	1.63	5.89	5.90	11.20
Belgium	0.34	6.25	11.90	0.32	0.61	0.78	1.54	2.79	1.46	6.57	6.44	12.51
Bulgaria	0.63	3.35	8.96	0.74	1.99	0.92	0.80	1.76	0.61	4.10	4.18	10.95
Croatia	0.26	4.24	NaN	0.03	NaN	0.65	1.87	0.97	0.78	4.27	4.42	NaN

Fig. 7.8 Expenditure on different educational indicators for the first five countries in the Eurostat dataset

7.3 Case Study

In order to illustrate clustering with a real dataset, we will now analyze the indicators of spending on education among the European Union member states, provided by the Eurostat data bank.[3] The data are organized by year (TIME) from 2002 until 2011 and country (GEO): ('Albania', 'Austria', 'Belgium', 'Bulgaria', etc.). Twelve indicators (INDIC_ED) of financing of education with their corresponding values (Value) are given: (1) Expenditure on educational institutions from private sources as % of gross domestic product (GDP), for all levels of education combined; (2) Expenditure on educational institutions from public sources as % of GDP, for all levels of government combined, (3) Expenditure on educational institutions from public sources as % of total public expenditure, for all levels of education combined, (4) Public subsidies to the private sector as % of GDP, for all levels of education combined, (5) Public subsidies to the private sector as % of total public expenditure, for all levels of education combined, etc. We can store the 12 indicators for a given year (e.g., 2010) in a table. Figure 7.8 provides visualization of the first five countries in the table.

As we can observe, this is not a clean dataset, since there are values missing. Some countries have very limited information and should be excluded. Other countries may still not collect or have access to a few indicators. For these last cases, we can proceed in two ways: (a) fill in the gaps with some non-informative, non-biasing data; or (b) drop the features with missing values for the analysis. If we have many features and only a few have missing values, then it is not very harmful to drop them. However, if missing values are spread across most of the features, we eventually have to deal with them. In our case, both options seem reasonable, as long as the number of missing features for a country is not too large. We will proceed in both ways at the same time.

We apply both options: filling the gap with the mean value of the feature and the dropping option, ignoring the indicators with missing values. Let us now apply K-means clustering to these data in order to partition the countries according to

[3] http://ec.europa.eu/eurostat.

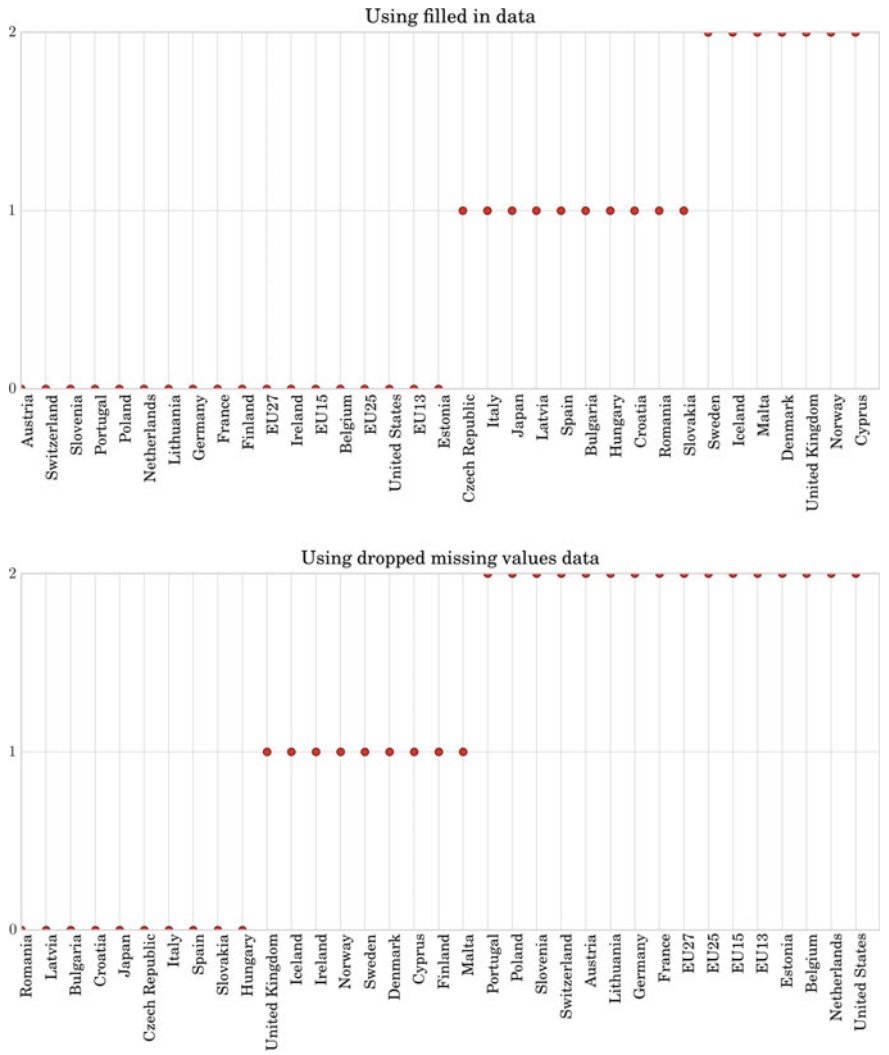

Fig. 7.9 Clustering of the countries according to their educational expenditure using filled-in (*top row*) and dropped (*bottom row*) missing values

their investment in education and check their profiles. Figure 7.9 shows the results of this K-means clustering. We have sorted the data for better visualization. At a simple glance, we can see that the partitions (top and bottom of Fig. 7.9) are different. Most countries in cluster 2 in the filled-in dataset correspond to cluster 0 in the dropped missing values dataset. Analogously, most of cluster 0 in the filled-in dataset correspond to cluster 1 in the dropped missing values dataset; and most countries from cluster 1 in the filled-in dataset correspond to cluster 2 in the dropped

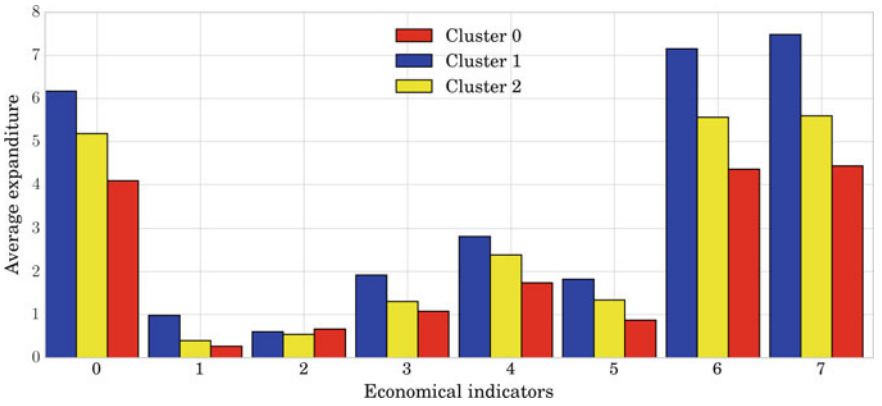

Fig. 7.10 Mean expenditure of the different clusters according to the 8 indicators of the indicators-dropped dataset

set. Still, there are some countries that do not follow this rule. That is, looking at both clusterings, they may yield similar (up to label permutation) results, but they will not necessarily always coincide. This is mainly due to two aspects: the random initialization of the K-means clustering and the fact that each method works in a different space (i.e., dropped data in 8D space *vs* filled-in data, working in 12D space). Note that we should not consider the assigned absolute cluster value, since it is irrelevant. The mean expenditure of the different clusters is shown by different colors according to the 8 indicators of the indicators-dropped dataset (see Fig. 7.10).

So, without loss of generality, we continue analyzing the set obtained by dropping missing values. Let us now check the clusters and check their profile by looking at the centroids. Visualizing the eight values of the three clusters (see Fig. 7.10), we can see that cluster 1 spends more on education for the 8 educational indicators, while cluster 0 is the one with least resources invested in education.

Let us consider a specific country, e.g., Spain and its expenditure on education. If we refine cluster 0 further and check how close members are from this cluster to cluster 1, it may give us a hint as to a possible ordering. When visualizing the distance to cluster 0 and 1, we can observe that Spain, while being from cluster 0, has a smaller distance to cluster 1 (see Fig. 7.11). This should make us realize that using 3 clusters probably does not sufficiently represent the groups of countries. So we redo the process, but applying $k = 4$: we obtain 4 clusters. This time cluster 0 includes the EU members with medium expenditure (Fig. 7.12). This reinforce the intuition about Spain being a limit case in the former clustering. The clusters obtained are as follows:

- Cluster 0: ('Austria', 'Estonia', 'EU13', 'EU15', 'EU25', 'EU27', 'France', 'Germany', 'Hungary', 'Latvia', 'Lithuania', 'Netherlands', 'Poland', 'Portugal', 'Slovenia', 'Spain', 'Switzerland', 'United Kingdom', 'United States')

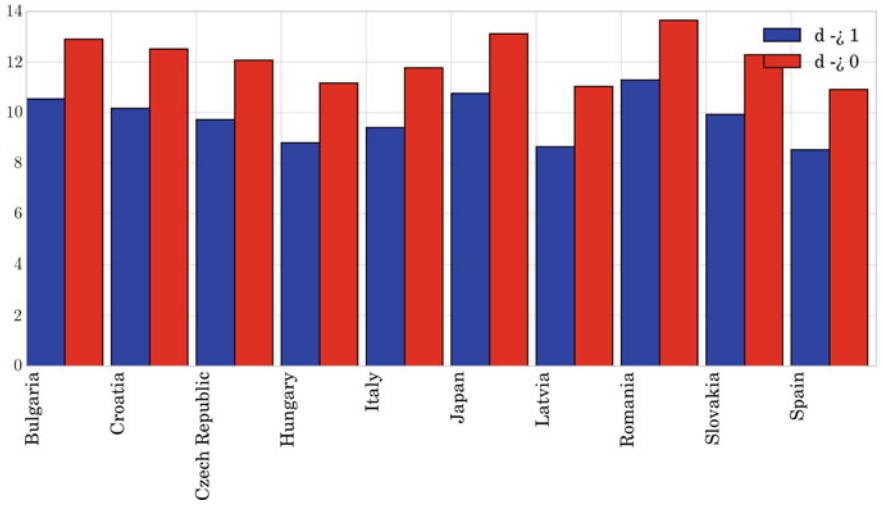

Fig. 7.11 Distance of countries in cluster 0 to centroids of cluster 0 (in *red*) and cluster 1 (in *blue*)

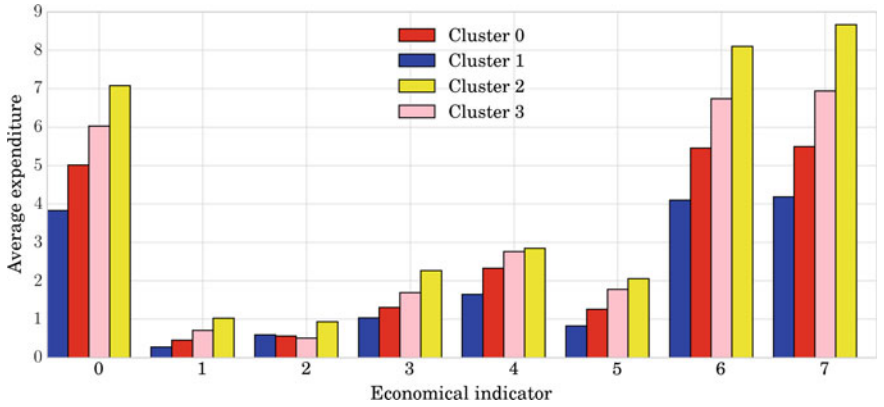

Fig. 7.12 K-means applied to the Eurostat dataset grouping the countries into four clusters

- Cluster 1: ('Bulgaria', 'Croatia', 'Czech Republic', 'Italy', 'Japan', 'Romania', 'Slovakia')
- Cluster 2: ('Cyprus', 'Denmark', 'Iceland')
- Cluster 3: ('Belgium', 'Finland', 'Ireland', 'Malta', 'Norway', 'Sweden')

We can repeat the process using the alternative clustering techniques and compare their results. Let us first apply spectral clustering. The corresponding code will be as follows:

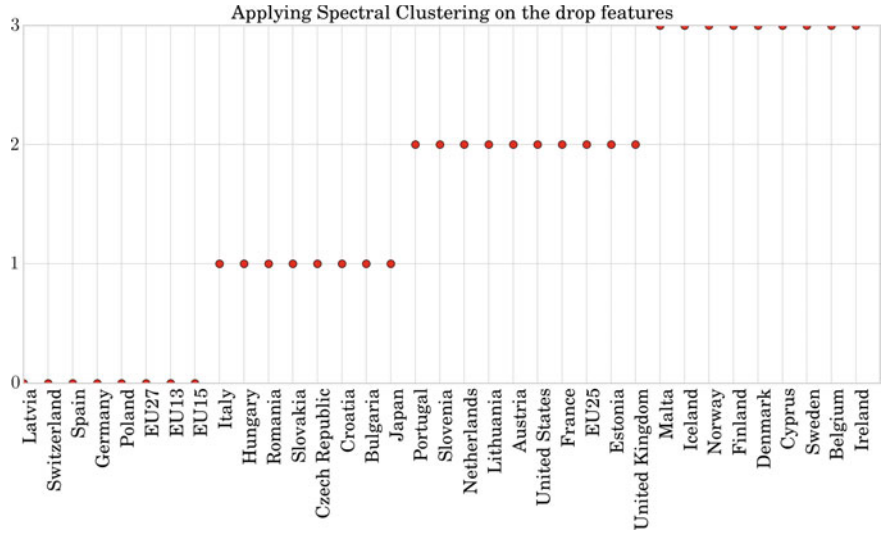

Fig. 7.13 Spectral clustering applied to the European countries according to their expenditure on education

In [11]:
```
X = StandardScaler().fit_transform(edudrop.values)
distances = euclidean_distances(edudrop.values)
spectral = cluster.SpectralClustering(
    n_clusters = 4, affinity = "nearest_neighbors")
spectral.fit(edudrop.values)
y_pred = spectral.labels_.astype(np.int)
```

The result of this spectral clustering is shown in Fig. 7.13. Note that in general, the aim of spectral clustering is to obtain more balanced clusters. In this way, the predicted cluster 1 merges clusters 2 and 3 of the K-means clustering, cluster 2 corresponds to cluster 1 of the K-means clustering, cluster 0 mainly shifts to cluster 2, and cluster 3 corresponds to cluster 0 of the K-means.

Applying agglomerative clustering, not only we do obtain different clusters, but also we can see how different clusters are obtained. Thus, in some way it is giving us information on which the most similar pairs of countries and clusters are. The corresponding code that applies the agglomerative clustering will be as follows:

In [12]:
```
from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.spatial.distance import pdist

X_train = edudrop.values
dist = pdist(X_train, 'euclidean')
linkage_matrix = linkage(dist, method = 'complete');

plt.figure(figsize = (11.3, 11.3))
dendrogram(linkage_matrix, orientation="right",
           color_threshold = 3,
           labels = wrk_countries_names,
           leaf_font_size = 20);
plt.tight_layout()
```

In `Scikit-learn`, the parameter *color_threshold* of the command dendrogram() colors all the descendent links below a cluster node k the same color if k is the first node below the color_threshold. All links connecting nodes with distances greater than or equal to the threshold are colored blue. Hence, using *color_threshold* = 3, the clusters obtained are as follows:

- Cluster 0: ('Cyprus', 'Denmark', 'Iceland')
- Cluster 1: ('Bulgaria', 'Croatia', 'Czech Republic', 'Italy', 'Japan', 'Romania', 'Slovakia')
- Cluster 2: ('Belgium', 'Finland', 'Ireland', 'Malta', 'Norway', 'Sweden')
- Cluster 3: ('Austria', 'Estonia', 'EU13', 'EU15', 'EU25', 'EU27', 'France', 'Germany', 'Hungary', 'Latvia', 'Lithuania', 'Netherlands', 'Poland', 'Portugal', 'Slovenia', 'Spain', 'Switzerland', 'United Kingdom', 'United States')
  Note that, to a high degree, they correspond to the clusters obtained by the K-means (except for permutation of cluster labels, which is irrelevant).

Figure 7.14 shows the construction of the clusters using complete linkage agglomerative clustering. Different cuts at different levels of the dendrogram allow us to obtain different numbers of clusters.

To summarize, we can compare the results of the three clustering approaches. We cannot expect the results to coincide, since the different approaches are based on different criteria for constructing clusters. Nonetheless, we can still observe that in this case, K-means and the agglomerative approaches gave the same results (up to a permutation of the number of cluster, which is irrelevant); while spectral clustering gave more evenly distributed clusters. This later approach fused clusters 0 and 2 of the agglomerative clustering in cluster 1, and split cluster 3 of the agglomerative clustering into its clusters 0 and 3. Note that these results could change when using different distances among data.

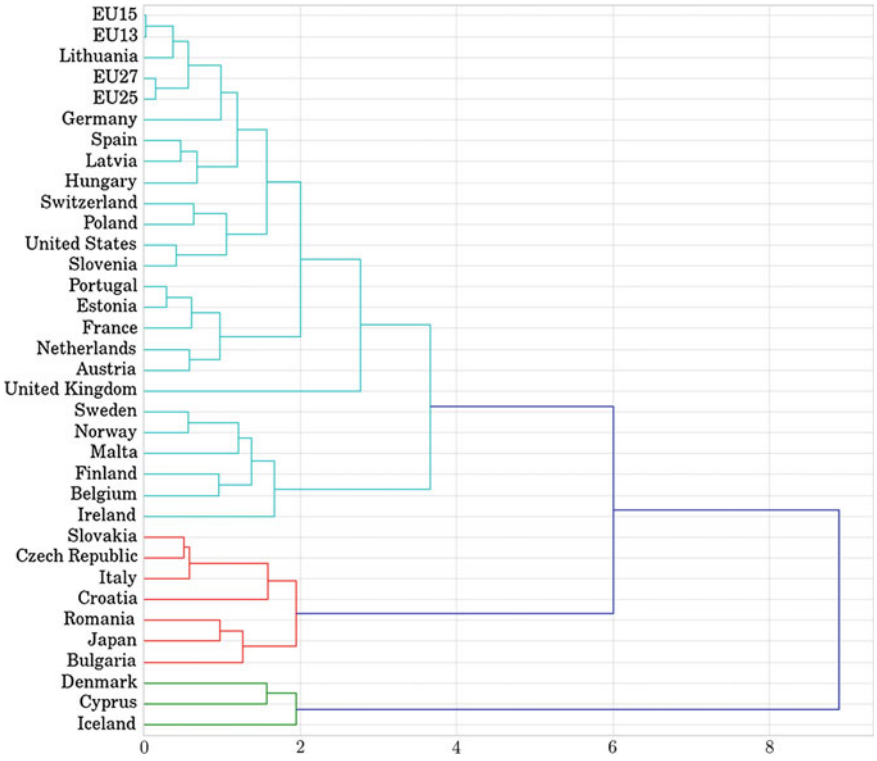

Fig. 7.14 Agglomerative clustering applied to cluster European countries according to their expenditure on education

7.4 Conclusions

In this chapter, we have introduced the unsupervised learning problem as a problem of knowledge or structure discovery from a set of unlabeled data. We have focused on clustering as one of the main problems in unsupervised learning. Basic concepts such as distance, similarity, connectivity, and the quality of the clustering results have been discussed as the main elements to be determined before choosing a specific clustering technique. Three basic clustering techniques have been introduced: K-means, agglomerative clustering, and spectral clustering. We have discussed their advantages and disadvantages and compared them through different examples. One of the important parameters for most clustering techniques is the number of clusters expected.

Regarding scalability, K-means can be applied to very large datasets, but the number of clusters should be as much as medium value, due to its iterative procedure. Spectral clustering can manage datasets that are not very large and a reasonable number of clusters, since it is based on computing the eigenvectors of the affinity matrix. In this aspect, the best option is hierarchical clustering, which allows large

numbers of samples and clusters to be tackled. Regarding uses, K-means is best suited to data with a flat geometry (isotropic and compact clusters), while spectral clustering and agglomerative clustering, with either average or complete linkage, are able to detect patterns in data with non-flat geometry. The connectivity graph is especially helpful in such cases. At the end of the chapter, a case study using a Eurostat database has been considered to show the applicability of the clustering in real problems (with real datasets).

Acknowledgements This chapter was co-written by Petia Radeva and Oriol Pujol.

References

1. Press, WH; Teukolsky, SA; Vetterling, W.T.; Flannery, B.P. (2007). "Section 16.1. Gaussian Mixture Models and k-Means Clustering". Numerical Recipes: The Art of Scientific Computing (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
2. Meilă, M.; Shi, J. (2001); "Learning Segmentation by Random Walks", Neural Information Processing Systems 13 (NIPS 2000), 2001, pp. 873–879.
3. Székely, G.J.; Rizzo, M.L. (2005). "Hierarchical clustering via Joint Between-Within Distances: Extending Ward's Minimum Variance Method", Journal of Classification 22, 151–183.