10
Unsupervised Learning

Most of this book concerns *supervised learning* methods such as regression and classification. In the supervised learning setting, we typically have access to a set of p features $X_1, X_2, \ldots, X_p$, measured on n observations, and a response Y also measured on those same n observations. The goal is then to predict Y using $X_1, X_2, \ldots, X_p$.

This chapter will instead focus on *unsupervised learning*, a set of statistical tools intended for the setting in which we have only a set of features $X_1, X_2, \ldots, X_p$ measured on n observations. We are not interested in prediction, because we do not have an associated response variable Y. Rather, the goal is to discover interesting things about the measurements on $X_1, X_2, \ldots, X_p$. Is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations? Unsupervised learning refers to a diverse set of techniques for answering questions such as these. In this chapter, we will focus on two particular types of unsupervised learning: *principal components analysis*, a tool used for data visualization or data pre-processing before supervised techniques are applied, and *clustering*, a broad class of methods for discovering unknown subgroups in data.

10.1 The Challenge of Unsupervised Learning

Supervised learning is a well-understood area. In fact, if you have read the preceding chapters in this book, then you should by now have a good

grasp of supervised learning. For instance, if you are asked to predict a binary outcome from a data set, you have a very well developed set of tools at your disposal (such as logistic regression, linear discriminant analysis, classification trees, support vector machines, and more) as well as a clear understanding of how to assess the quality of the results obtained (using cross-validation, validation on an independent test set, and so forth).

In contrast, unsupervised learning is often much more challenging. The exercise tends to be more subjective, and there is no simple goal for the analysis, such as prediction of a response. Unsupervised learning is often performed as part of an *exploratory data analysis*. Furthermore, it can be hard to assess the results obtained from unsupervised learning methods, since there is no universally accepted mechanism for performing cross-validation or validating results on an independent data set. The reason for this difference is simple. If we fit a predictive model using a supervised learning technique, then it is possible to *check our work* by seeing how well our model predicts the response Y on observations not used in fitting the model. However, in unsupervised learning, there is no way to check our work because we don't know the true answer—the problem is unsupervised.

exploratory data analysis

Techniques for unsupervised learning are of growing importance in a number of fields. A cancer researcher might assay gene expression levels in 100 patients with breast cancer. He or she might then look for subgroups among the breast cancer samples, or among the genes, in order to obtain a better understanding of the disease. An online shopping site might try to identify groups of shoppers with similar browsing and purchase histories, as well as items that are of particular interest to the shoppers within each group. Then an individual shopper can be preferentially shown the items in which he or she is particularly likely to be interested, based on the purchase histories of similar shoppers. A search engine might choose what search results to display to a particular individual based on the click histories of other individuals with similar search patterns. These statistical learning tasks, and many more, can be performed via unsupervised learning techniques.

10.2   Principal Components Analysis

Principal components are discussed in Section 6.3.1 in the context of principal components regression. When faced with a large set of correlated variables, principal components allow us to summarize this set with a smaller number of representative variables that collectively explain most of the variability in the original set. The principal component directions are presented in Section 6.3.1 as directions in feature space along which the original data are *highly variable*. These directions also define lines and subspaces that are *as close as possible* to the data cloud. To perform

principal components regression, we simply use principal components as predictors in a regression model in place of the original larger set of variables.

Principal component analysis (PCA) refers to the process by which principal components are computed, and the subsequent use of these components in understanding the data. PCA is an unsupervised approach, since it involves only a set of features $X_1, X_2, \ldots, X_p$, and no associated response Y. Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data visualization (visualization of the observations or visualization of the variables). We now discuss PCA in greater detail, focusing on the use of PCA as a tool for unsupervised data exploration, in keeping with the topic of this chapter.

principal component analysis

10.2.1 *What Are Principal Components?*

Suppose that we wish to visualize n observations with measurements on a set of p features, $X_1, X_2, \ldots, X_p$, as part of an exploratory data analysis. We could do this by examining two-dimensional scatterplots of the data, each of which contains the n observations' measurements on two of the features. However, there are $\binom{p}{2} = p(p-1)/2$ such scatterplots; for example, with $p = 10$ there are 45 plots! If p is large, then it will certainly not be possible to look at all of them; moreover, most likely none of them will be informative since they each contain just a small fraction of the total information present in the data set. Clearly, a better method is required to visualize the n observations when p is large. In particular, we would like to find a low-dimensional representation of the data that captures as much of the information as possible. For instance, if we can obtain a two-dimensional representation of the data that captures most of the information, then we can plot the observations in this low-dimensional space.

PCA provides a tool to do just this. It finds a low-dimensional representation of a data set that contains as much as possible of the variation. The idea is that each of the n observations lives in p-dimensional space, but not all of these dimensions are equally interesting. PCA seeks a small number of dimensions that are as interesting as possible, where the concept of *interesting* is measured by the amount that the observations vary along each dimension. Each of the dimensions found by PCA is a linear combination of the p features. We now explain the manner in which these dimensions, or *principal components*, are found.

The *first principal component* of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \ldots + \phi_{p1}X_p \tag{10.1}$$

that has the largest variance. By *normalized*, we mean that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. We refer to the elements $\phi_{11}, \ldots, \phi_{p1}$ as the *loadings* of the first principal

loading

component; together, the loadings make up the principal component loading vector, $\phi_1 = (\phi_{11} \ \phi_{21} \ \ldots \ \phi_{p1})^T$. We constrain the loadings so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance.

Given a $n \times p$ data set $\mathbf{X}$, how do we compute the first principal component? Since we are only interested in variance, we assume that each of the variables in $\mathbf{X}$ has been centered to have mean zero (that is, the column means of $\mathbf{X}$ are zero). We then look for the linear combination of the sample feature values of the form

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \ldots + \phi_{p1}x_{ip} \qquad (10.2)$$

that has largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. In other words, the first principal component loading vector solves the optimization problem

$$\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1}x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} \phi_{j1}^2 = 1. \qquad (10.3)$$

From (10.2) we can write the objective in (10.3) as $\frac{1}{n}\sum_{i=1}^{n} z_{i1}^2$. Since $\frac{1}{n}\sum_{i=1}^{n} x_{ij} = 0$, the average of the $z_{11},\ldots,z_{n1}$ will be zero as well. Hence the objective that we are maximizing in (10.3) is just the sample variance of the n values of z_{i1}. We refer to $z_{11},\ldots,z_{n1}$ as the *scores* of the first principal component. Problem (10.3) can be solved via an eigen decomposition, a standard technique in linear algebra, but details are outside of the scope of this book.

score

There is a nice geometric interpretation for the first principal component. The loading vector ϕ_1 with elements $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$ defines a direction in feature space along which the data vary the most. If we project the n data points $x_1, \ldots, x_n$ onto this direction, the projected values are the principal component scores $z_{11}, \ldots, z_{n1}$ themselves. For instance, Figure 6.14 on page 230 displays the first principal component loading vector (green solid line) on an advertising data set. In these data, there are only two features, and so the observations as well as the first principal component loading vector can be easily displayed. As can be seen from (6.19), in that data set $\phi_{11} = 0.839$ and $\phi_{21} = 0.544$.

After the first principal component Z_1 of the features has been determined, we can find the second principal component Z_2. The second principal component is the linear combination of $X_1, \ldots, X_p$ that has maximal variance out of all linear combinations that are *uncorrelated* with Z_1. The second principal component scores $z_{12}, z_{22}, \ldots, z_{n2}$ take the form

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \ldots + \phi_{p2}x_{ip}, \qquad (10.4)$$

	PC1	PC2
Murder	0.5358995	−0.4181809
Assault	0.5831836	−0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

TABLE 10.1. *The principal component loading vectors, ϕ_1 and ϕ_2, for the* USArrests *data. These are also displayed in Figure 10.1.*

where ϕ_2 is the second principal component loading vector, with elements $\phi_{12}, \phi_{22}, \ldots, \phi_{p2}$. It turns out that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction ϕ_2 to be orthogonal (perpendicular) to the direction ϕ_1. In the example in Figure 6.14, the observations lie in two-dimensional space (since $p = 2$), and so once we have found ϕ_1, there is only one possibility for ϕ_2, which is shown as a blue dashed line. (From Section 6.3.1, we know that $\phi_{12} = 0.544$ and $\phi_{22} = -0.839$.) But in a larger data set with $p > 2$ variables, there are multiple distinct principal components, and they are defined in a similar manner. To find ϕ_2, we solve a problem similar to (10.3) with ϕ_2 replacing ϕ_1, and with the additional constraint that ϕ_2 is orthogonal to ϕ_1.[1]

Once we have computed the principal components, we can plot them against each other in order to produce low-dimensional views of the data. For instance, we can plot the score vector Z_1 against Z_2, Z_1 against Z_3, Z_2 against Z_3, and so forth. Geometrically, this amounts to projecting the original data down onto the subspace spanned by ϕ_1, ϕ_2, and ϕ_3, and plotting the projected points.

We illustrate the use of PCA on the USArrests data set. For each of the 50 states in the United States, the data set contains the number of arrests per $100,000$ residents for each of three crimes: Assault, Murder, and Rape. We also record UrbanPop (the percent of the population in each state living in urban areas). The principal component score vectors have length $n = 50$, and the principal component loading vectors have length $p = 4$. PCA was performed after standardizing each variable to have mean zero and standard deviation one. Figure 10.1 plots the first two principal components of these data. The figure represents both the principal component scores and the loading vectors in a single *biplot* display. The loadings are also given in Table 10.1.

biplot

In Figure 10.1, we see that the first loading vector places approximately equal weight on Assault, Murder, and Rape, with much less weight on

---

[1]On a technical note, the principal component directions ϕ_1, ϕ_2, $\phi_3, \ldots$ are the ordered sequence of eigenvectors of the matrix $\mathbf{X}^T \mathbf{X}$, and the variances of the components are the eigenvalues. There are at most $\min(n-1, p)$ principal components.
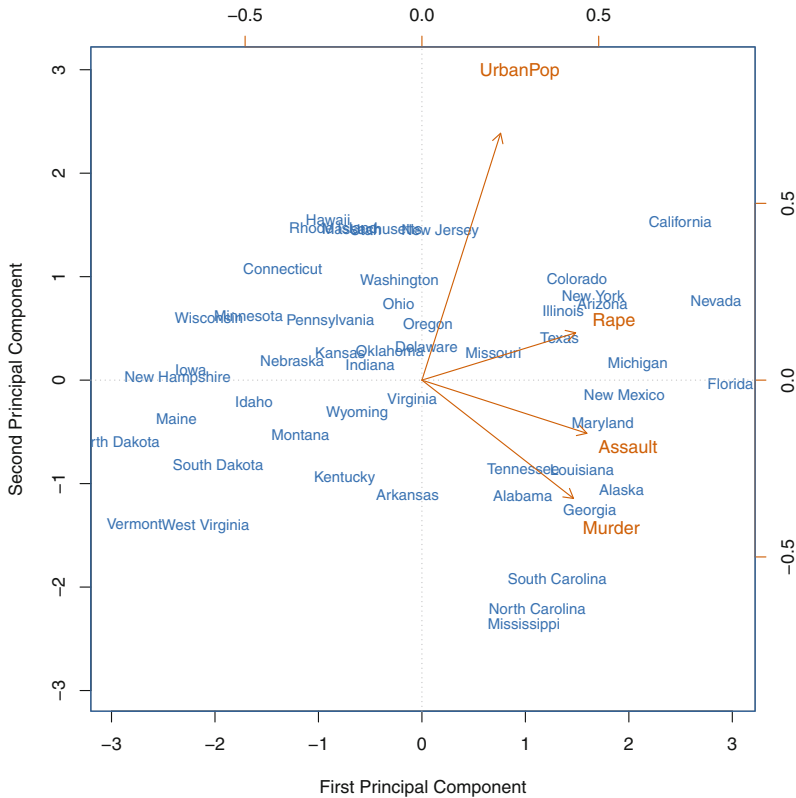
FIGURE 10.1. *The first two principal components for the* `USArrests` *data. The blue state names represent the scores for the first two principal components. The orange arrows indicate the first two principal component loading vectors (with axes on the top and right). For example, the loading for* `Rape` *on the first component is 0.54, and its loading on the second principal component 0.17 (the word* `Rape` *is centered at the point (0.54, 0.17)). This figure is known as a biplot, because it displays both the principal component scores and the principal component loadings.*

`UrbanPop`. Hence this component roughly corresponds to a measure of overall rates of serious crimes. The second loading vector places most of its weight on `UrbanPop` and much less weight on the other three features. Hence, this component roughly corresponds to the level of urbanization of the state. Overall, we see that the crime-related variables (`Murder`, `Assault`, and `Rape`) are located close to each other, and that the `UrbanPop` variable is far from the other three. This indicates that the crime-related variables are correlated with each other—states with high murder rates tend to have high assault and rape rates—and that the `UrbanPop` variable is less correlated with the other three.

We can examine differences between the states via the two principal component score vectors shown in Figure 10.1. Our discussion of the loading vectors suggests that states with large positive scores on the first component, such as California, Nevada and Florida, have high crime rates, while states like North Dakota, with negative scores on the first component, have low crime rates. California also has a high score on the second component, indicating a high level of urbanization, while the opposite is true for states like Mississippi. States close to zero on both components, such as Indiana, have approximately average levels of both crime and urbanization.

10.2.2 Another Interpretation of Principal Components

The first two principal component loading vectors in a simulated three-dimensional data set are shown in the left-hand panel of Figure 10.2; these two loading vectors span a plane along which the observations have the highest variance.

In the previous section, we describe the principal component loading vectors as the directions in feature space along which the data vary the most, and the principal component scores as projections along these directions. However, an alternative interpretation for principal components can also be useful: principal components provide low-dimensional linear surfaces that are *closest* to the observations. We expand upon that interpretation here.

The first principal component loading vector has a very special property: it is the line in p-dimensional space that is *closest* to the n observations (using average squared Euclidean distance as a measure of closeness). This interpretation can be seen in the left-hand panel of Figure 6.15; the dashed lines indicate the distance between each observation and the first principal component loading vector. The appeal of this interpretation is clear: we seek a single dimension of the data that lies as close as possible to all of the data points, since such a line will likely provide a good summary of the data.

The notion of principal components as the dimensions that are closest to the n observations extends beyond just the first principal component. For instance, the first two principal components of a data set span the plane that is closest to the n observations, in terms of average squared Euclidean distance. An example is shown in the left-hand panel of Figure 10.2. The first three principal components of a data set span the three-dimensional hyperplane that is closest to the n observations, and so forth.

Using this interpretation, together the first M principal component score vectors and the first M principal component loading vectors provide the best M-dimensional approximation (in terms of Euclidean distance) to the ith observation x_{ij}. This representation can be written
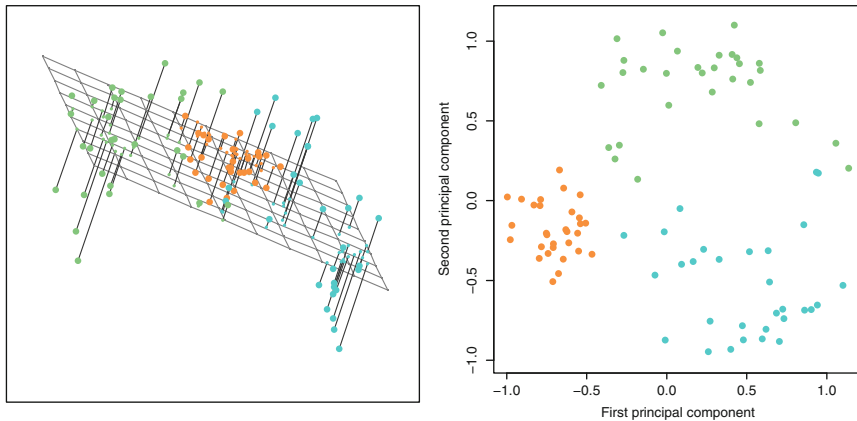
FIGURE 10.2. *Ninety observations simulated in three dimensions.* Left: *the first two principal component directions span the plane that best fits the data. It minimizes the sum of squared distances from each point to the plane.* Right: *the first two principal component score vectors give the coordinates of the projection of the 90 observations onto the plane. The variance in the plane is maximized.*

$$x_{ij} \approx \sum_{m=1}^{M} z_{im}\phi_{jm} \tag{10.5}$$

(assuming the original data matrix $\mathbf{X}$ is column-centered). In other words, together the M principal component score vectors and M principal component loading vectors can give a good approximation to the data when M is sufficiently large. When $M = \min(n-1, p)$, then the representation is exact: $x_{ij} = \sum_{m=1}^{M} z_{im}\phi_{jm}$.

10.2.3   More on PCA

Scaling the Variables

We have already mentioned that before PCA is performed, the variables should be centered to have mean zero. Furthermore, *the results obtained when we perform PCA will also depend on whether the variables have been individually scaled* (each multiplied by a different constant). This is in contrast to some other supervised and unsupervised learning techniques, such as linear regression, in which scaling the variables has no effect. (In linear regression, multiplying a variable by a factor of c will simply lead to multiplication of the corresponding coefficient estimate by a factor of $1/c$, and thus will have no substantive effect on the model obtained.)

For instance, Figure 10.1 was obtained after scaling each of the variables to have standard deviation one. This is reproduced in the left-hand plot in Figure 10.3. Why does it matter that we scaled the variables? In these data,
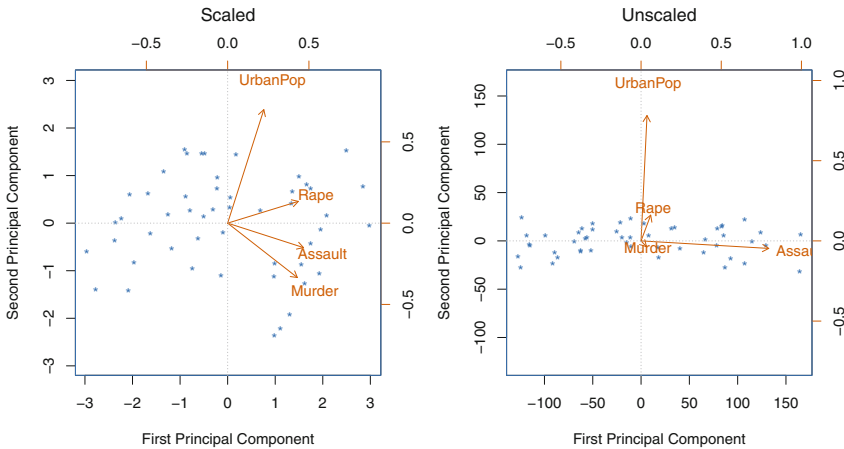
FIGURE 10.3. *Two principal component biplots for the* `USArrests` *data.* Left: *the same as Figure 10.1, with the variables scaled to have unit standard deviations.* Right: *principal components using unscaled data.* `Assault` *has by far the largest loading on the first principal component because it has the highest variance among the four variables. In general, scaling the variables to have standard deviation one is recommended.*

the variables are measured in different units; `Murder`, `Rape`, and `Assault` are reported as the number of occurrences per 100,000 people, and `UrbanPop` is the percentage of the state's population that lives in an urban area. These four variables have variance 18.97, 87.73, 6945.16, and 209.5, respectively. Consequently, if we perform PCA on the unscaled variables, then the first principal component loading vector will have a very large loading for `Assault`, since that variable has by far the highest variance. The right-hand plot in Figure 10.3 displays the first two principal components for the `USArrests` data set, without scaling the variables to have standard deviation one. As predicted, the first principal component loading vector places almost all of its weight on `Assault`, while the second principal component loading vector places almost all of its weight on `UrpanPop`. Comparing this to the left-hand plot, we see that scaling does indeed have a substantial effect on the results obtained.

However, this result is simply a consequence of the scales on which the variables were measured. For instance, if `Assault` were measured in units of the number of occurrences per 100 people (rather than number of occurrences per 100,000 people), then this would amount to dividing all of the elements of that variable by 1,000. Then the variance of the variable would be tiny, and so the first principal component loading vector would have a very small value for that variable. Because it is undesirable for the principal components obtained to depend on an arbitrary choice of scaling, we typically scale each variable to have standard deviation one before we perform PCA.

In certain settings, however, the variables may be measured in the same units. In this case, we might not wish to scale the variables to have standard deviation one before performing PCA. For instance, suppose that the variables in a given data set correspond to expression levels for p genes. Then since expression is measured in the same "units" for each gene, we might choose not to scale the genes to each have standard deviation one.

Uniqueness of the Principal Components

Each principal component loading vector is unique, up to a sign flip. This means that two different software packages will yield the same principal component loading vectors, although the signs of those loading vectors may differ. The signs may differ because each principal component loading vector specifies a direction in p-dimensional space: flipping the sign has no effect as the direction does not change. (Consider Figure 6.14—the principal component loading vector is a line that extends in either direction, and flipping its sign would have no effect.) Similarly, the score vectors are unique up to a sign flip, since the variance of Z is the same as the variance of $-Z$. It is worth noting that when we use (10.5) to approximate x_{ij} we multiply z_{im} by ϕ_{jm}. Hence, if the sign is flipped on both the loading and score vectors, the final product of the two quantities is unchanged.

The Proportion of Variance Explained

In Figure 10.2, we performed PCA on a three-dimensional data set (left-hand panel) and projected the data onto the first two principal component loading vectors in order to obtain a two-dimensional view of the data (i.e. the principal component score vectors; right-hand panel). We see that this two-dimensional representation of the three-dimensional data does successfully capture the major pattern in the data: the orange, green, and cyan observations that are near each other in three-dimensional space remain nearby in the two-dimensional representation. Similarly, we have seen on the `USArrests` data set that we can summarize the 50 observations and 4 variables using just the first two principal component score vectors and the first two principal component loading vectors.

We can now ask a natural question: how much of the information in a given data set is lost by projecting the observations onto the first few principal components? That is, how much of the variance in the data is *not* contained in the first few principal components? More generally, we are interested in knowing the *proportion of variance explained* (PVE) by each principal component. The *total variance* present in a data set (assuming that the variables have been centered to have mean zero) is defined as

proportion of variance explained

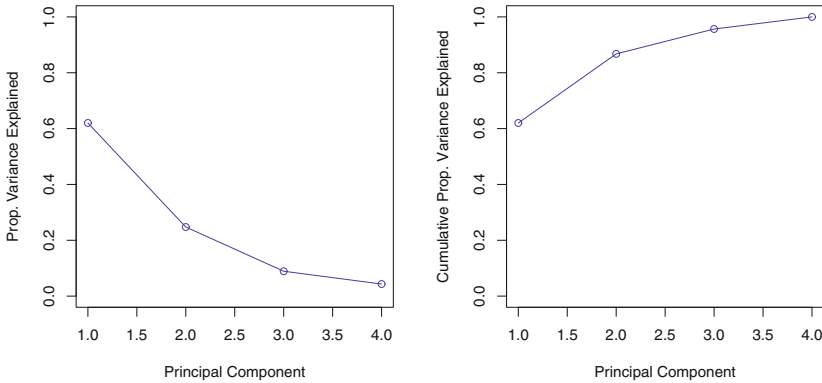$$\sum_{j=1}^{p} \text{Var}(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2, \qquad (10.6)$$

FIGURE 10.4. Left: *a scree plot depicting the proportion of variance explained by each of the four principal components in the* `USArrests` *data.* Right: *the cumulative proportion of variance explained by the four principal components in the* `USArrests` *data.*

and the variance explained by the mth principal component is

$$\frac{1}{n}\sum_{i=1}^{n} z_{im}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p} \phi_{jm}x_{ij}\right)^2. \tag{10.7}$$

Therefore, the PVE of the mth principal component is given by

$$\frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{jm}x_{ij}\right)^2}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2}. \tag{10.8}$$

The PVE of each principal component is a positive quantity. In order to compute the cumulative PVE of the first M principal components, we can simply sum (10.8) over each of the first M PVEs. In total, there are $\min(n-1, p)$ principal components, and their PVEs sum to one.

In the `USArrests` data, the first principal component explains 62.0 % of the variance in the data, and the next principal component explains 24.7 % of the variance. Together, the first two principal components explain almost 87 % of the variance in the data, and the last two principal components explain only 13 % of the variance. This means that Figure 10.1 provides a pretty accurate summary of the data using just two dimensions. The PVE of each principal component, as well as the cumulative PVE, is shown in Figure 10.4. The left-hand panel is known as a *scree plot*, and will be discussed next.

scree plot

Deciding How Many Principal Components to Use

In general, a $n \times p$ data matrix $\mathbf{X}$ has $\min(n-1, p)$ distinct principal components. However, we usually are not interested in all of them; rather,

we would like to use just the first few principal components in order to
visualize or interpret the data. In fact, we would like to use the smallest
number of principal components required to get a *good* understanding of the
data. How many principal components are needed? Unfortunately, there is
no single (or simple!) answer to this question.

We typically decide on the number of principal components required
to visualize the data by examining a *scree plot*, such as the one shown
in the left-hand panel of Figure 10.4. We choose the smallest number of
principal components that are required in order to explain a sizable amount
of the variation in the data. This is done by eyeballing the scree plot, and
looking for a point at which the proportion of variance explained by each
subsequent principal component drops off. This is often referred to as an
elbow in the scree plot. For instance, by inspection of Figure 10.4, one
might conclude that a fair amount of variance is explained by the first
two principal components, and that there is an elbow after the second
component. After all, the third principal component explains less than ten
percent of the variance in the data, and the fourth principal component
explains less than half that and so is essentially worthless.

However, this type of visual analysis is inherently *ad hoc*. Unfortunately,
there is no well-accepted objective way to decide how many principal com-
ponents are *enough*. In fact, the question of how many principal compo-
nents are enough is inherently ill-defined, and will depend on the specific
area of application and the specific data set. In practice, we tend to look
at the first few principal components in order to find interesting patterns
in the data. If no interesting patterns are found in the first few principal
components, then further principal components are unlikely to be of inter-
est. Conversely, if the first few principal components are interesting, then
we typically continue to look at subsequent principal components until no
further interesting patterns are found. This is admittedly a subjective ap-
proach, and is reflective of the fact that PCA is generally used as a tool for
exploratory data analysis.

On the other hand, if we compute principal components for use in a
supervised analysis, such as the principal components regression presented
in Section 6.3.1, then there is a simple and objective way to determine how
many principal components to use: we can treat the number of principal
component score vectors to be used in the regression as a tuning parameter
to be selected via cross-validation or a related approach. The comparative
simplicity of selecting the number of principal components for a supervised
analysis is one manifestation of the fact that supervised analyses tend to
be more clearly defined and more objectively evaluated than unsupervised
analyses.

10.2.4   Other Uses for Principal Components

We saw in Section 6.3.1 that we can perform regression using the principal component score vectors as features. In fact, many statistical techniques, such as regression, classification, and clustering, can be easily adapted to use the $n \times M$ matrix whose columns are the first $M \ll p$ principal component score vectors, rather than using the full $n \times p$ data matrix. This can lead to *less noisy* results, since it is often the case that the signal (as opposed to the noise) in a data set is concentrated in its first few principal components.

10.3   Clustering Methods

Clustering refers to a very broad set of techniques for finding *subgroups*, or *clusters*, in a data set. When we cluster the observations of a data set, we seek to partition them into distinct groups so that the observations within each group are quite similar to each other, while observations in different groups are quite different from each other. Of course, to make this concrete, we must define what it means for two or more observations to be *similar* or *different*. Indeed, this is often a domain-specific consideration that must be made based on knowledge of the data being studied.

clustering

For instance, suppose that we have a set of n observations, each with p features. The n observations could correspond to tissue samples for patients with breast cancer, and the p features could correspond to measurements collected for each tissue sample; these could be clinical measurements, such as tumor stage or grade, or they could be gene expression measurements. We may have a reason to believe that there is some heterogeneity among the n tissue samples; for instance, perhaps there are a few different *unknown* subtypes of breast cancer. Clustering could be used to find these subgroups. This is an unsupervised problem because we are trying to discover structure—in this case, distinct clusters—on the basis of a data set. The goal in supervised problems, on the other hand, is to try to predict some outcome vector such as survival time or response to drug treatment.

Both clustering and PCA seek to simplify the data via a small number of summaries, but their mechanisms are different:

- PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance;

- Clustering looks to find homogeneous subgroups among the observations.

Another application of clustering arises in marketing. We may have access to a large number of measurements (e.g. median household income, occupation, distance from nearest urban area, and so forth) for a large

number of people. Our goal is to perform *market segmentation* by identifying subgroups of people who might be more receptive to a particular form of advertising, or more likely to purchase a particular product. The task of performing market segmentation amounts to clustering the people in the data set.

Since clustering is popular in many fields, there exist a great number of clustering methods. In this section we focus on perhaps the two best-known clustering approaches: *K-means clustering* and *hierarchical clustering*. In *K*-means clustering, we seek to partition the observations into a pre-specified number of clusters. On the other hand, in hierarchical clustering, we do not know in advance how many clusters we want; in fact, we end up with a tree-like visual representation of the observations, called a *dendrogram*, that allows us to view at once the clusterings obtained for each possible number of clusters, from 1 to n. There are advantages and disadvantages to each of these clustering approaches, which we highlight in this chapter.

K-means clustering

hierarchical clustering

dendrogram

In general, we can cluster observations on the basis of the features in order to identify subgroups among the observations, or we can cluster features on the basis of the observations in order to discover subgroups among the features. In what follows, for simplicity we will discuss clustering observations on the basis of the features, though the converse can be performed by simply transposing the data matrix.

10.3.1 *K*-Means Clustering

K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters. To perform *K*-means clustering, we must first specify the desired number of clusters K; then the *K*-means algorithm will assign each observation to exactly one of the K clusters. Figure 10.5 shows the results obtained from performing *K*-means clustering on a simulated example consisting of 150 observations in two dimensions, using three different values of K.

The *K*-means clustering procedure results from a simple and intuitive mathematical problem. We begin by defining some notation. Let $C_1, \ldots, C_K$ denote sets containing the indices of the observations in each cluster. These sets satisfy two properties:

1. $C_1 \cup C_2 \cup \ldots \cup C_K = \{1, \ldots, n\}$. In other words, each observation belongs to at least one of the K clusters.

2. $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$. In other words, the clusters are non-overlapping: no observation belongs to more than one cluster.

For instance, if the ith observation is in the kth cluster, then $i \in C_k$. The idea behind *K*-means clustering is that a *good* clustering is one for which the *within-cluster variation* is as small as possible. The within-cluster variation
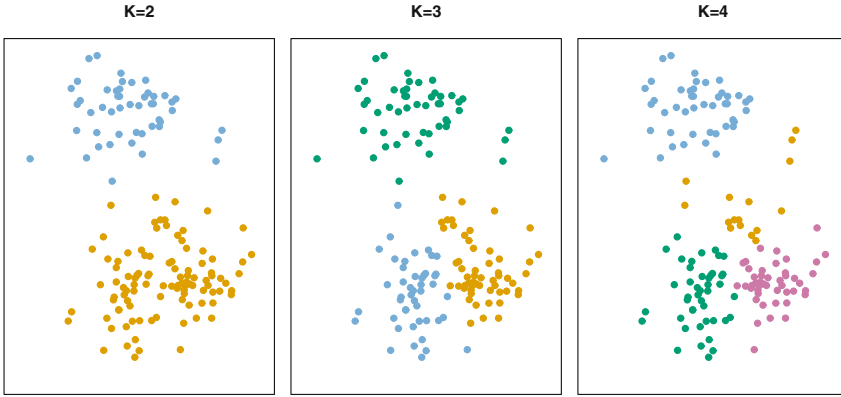
FIGURE 10.5. *A simulated data set with 150 observations in two-dimensional space. Panels show the results of applying K-means clustering with different values of K, the number of clusters. The color of each observation indicates the cluster to which it was assigned using the K-means clustering algorithm. Note that there is no ordering of the clusters, so the cluster coloring is arbitrary. These cluster labels were not used in clustering; instead, they are the outputs of the clustering procedure.*

for cluster C_k is a measure $W(C_k)$ of the amount by which the observations within a cluster differ from each other. Hence we want to solve the problem

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}. \tag{10.9}$$

In words, this formula says that we want to partition the observations into K clusters such that the total within-cluster variation, summed over all K clusters, is as small as possible.

Solving (10.9) seems like a reasonable idea, but in order to make it actionable we need to define the within-cluster variation. There are many possible ways to define this concept, but by far the most common choice involves *squared Euclidean distance*. That is, we define

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2, \tag{10.10}$$

where $|C_k|$ denotes the number of observations in the kth cluster. In other words, the within-cluster variation for the kth cluster is the sum of all of the pairwise squared Euclidean distances between the observations in the kth cluster, divided by the total number of observations in the kth cluster. Combining (10.9) and (10.10) gives the optimization problem that defines K-means clustering,

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}. \tag{10.11}$$

Now, we would like to find an algorithm to solve (10.11)—that is, a method to partition the observations into K clusters such that the objective of (10.11) is minimized. This is in fact a very difficult problem to solve precisely, since there are almost K^n ways to partition n observations into K clusters. This is a huge number unless K and n are tiny! Fortunately, a very simple algorithm can be shown to provide a local optimum—a *pretty good solution*—to the K-means optimization problem (10.11). This approach is laid out in Algorithm 10.1.

Algorithm 10.1 *K-Means Clustering*

---

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.

2. Iterate until the cluster assignments stop changing:

   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

Algorithm 10.1 is guaranteed to decrease the value of the objective (10.11) at each step. To understand why, the following identity is illuminating:

$$\frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = 2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2, \qquad (10.12)$$

where $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$ is the mean for feature j in cluster C_k. In Step 2(a) the cluster means for each feature are the constants that minimize the sum-of-squared deviations, and in Step 2(b), reallocating the observations can only improve (10.12). This means that as the algorithm is run, the clustering obtained will continually improve until the result no longer changes; the objective of (10.11) will never increase. When the result no longer changes, a *local optimum* has been reached. Figure 10.6 shows the progression of the algorithm on the toy example from Figure 10.5. K-means clustering derives its name from the fact that in Step 2(a), the cluster centroids are computed as the mean of the observations assigned to each cluster.

Because the K-means algorithm finds a local rather than a global optimum, the results obtained will depend on the initial (random) cluster assignment of each observation in Step 1 of Algorithm 10.1. For this reason, it is important to run the algorithm multiple times from different random
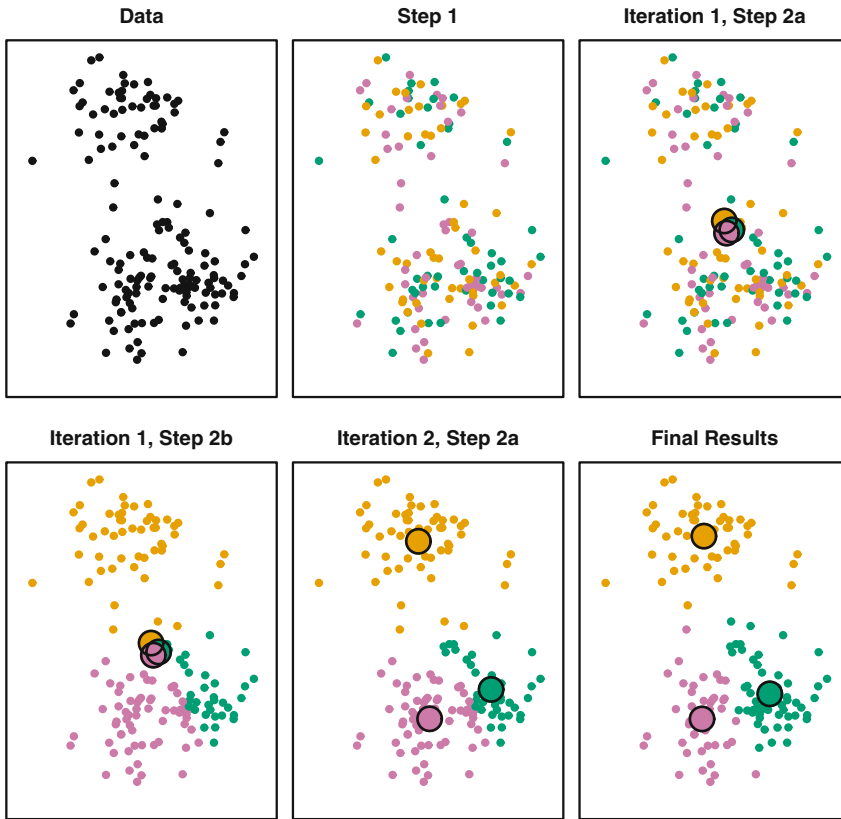
FIGURE 10.6. *The progress of the K-means algorithm on the example of Figure 10.5 with K=3. Top left: the observations are shown. Top center: in Step 1 of the algorithm, each observation is randomly assigned to a cluster. Top right: in Step 2(a), the cluster centroids are computed. These are shown as large colored disks. Initially the centroids are almost completely overlapping because the initial cluster assignments were chosen at random. Bottom left: in Step 2(b), each observation is assigned to the nearest centroid. Bottom center: Step 2(a) is once again performed, leading to new cluster centroids. Bottom right: the results obtained after ten iterations.*

initial configurations. Then one selects the *best* solution, i.e. that for which the objective (10.11) is smallest. Figure 10.7 shows the local optima obtained by running K-means clustering six times using six different initial cluster assignments, using the toy data from Figure 10.5. In this case, the best clustering is the one with an objective value of 235.8.

As we have seen, to perform K-means clustering, we must decide how many clusters we expect in the data. The problem of selecting K is far from simple. This issue, along with other practical considerations that arise in performing K-means clustering, is addressed in Section 10.3.3.

FIGURE 10.7. *K-means clustering performed six times on the data from Figure 10.5 with K = 3, each time with a different random assignment of the observations in Step 1 of the K-means algorithm. Above each plot is the value of the objective (10.11). Three different local optima were obtained, one of which resulted in a smaller value of the objective and provides better separation between the clusters. Those labeled in red all achieved the same best solution, with an objective value of 235.8.*

10.3.2   Hierarchical Clustering

One potential disadvantage of K-means clustering is that it requires us to pre-specify the number of clusters K. *Hierarchical clustering* is an alternative approach which does not require that we commit to a particular choice of K. Hierarchical clustering has an added advantage over K-means clustering in that it results in an attractive tree-based representation of the observations, called a *dendrogram*.

In this section, we describe *bottom-up* or *agglomerative* clustering. This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram (generally depicted as an upside-down tree; see
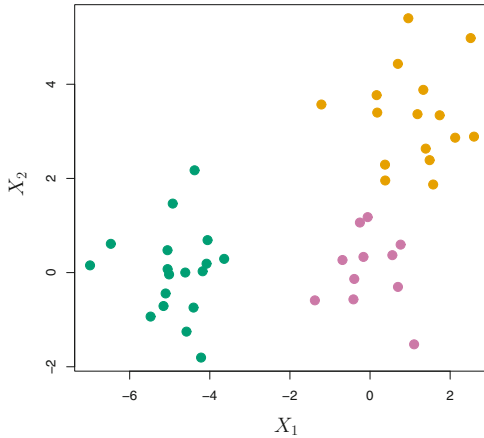
bottom-up

agglomerative

FIGURE 10.8. *Forty-five observations generated in two-dimensional space. In reality there are three distinct classes, shown in separate colors. However, we will treat these class labels as unknown and will seek to cluster the observations in order to discover the classes from the data.*

Figure 10.9) is built starting from the leaves and combining clusters up to the trunk. We will begin with a discussion of how to interpret a dendrogram and then discuss how hierarchical clustering is actually performed—that is, how the dendrogram is built.

Interpreting a Dendrogram

We begin with the simulated data set shown in Figure 10.8, consisting of 45 observations in two-dimensional space. The data were generated from a three-class model; the true class labels for each observation are shown in distinct colors. However, suppose that the data were observed without the class labels, and that we wanted to perform hierarchical clustering of the data. Hierarchical clustering (with complete linkage, to be discussed later) yields the result shown in the left-hand panel of Figure 10.9. How can we interpret this dendrogram?

In the left-hand panel of Figure 10.9, each *leaf* of the dendrogram represents one of the 45 observations in Figure 10.8. However, as we move up the tree, some leaves begin to *fuse* into branches. These correspond to observations that are similar to each other. As we move higher up the tree, branches themselves fuse, either with leaves or other branches. The earlier (lower in the tree) fusions occur, the more similar the groups of observations are to each other. On the other hand, observations that fuse later (near the top of the tree) can be quite different. In fact, this statement can be made precise: for any two observations, we can look for the point in the tree where branches containing those two observations are first fused. The height of this fusion, as measured on the vertical axis, indicates how
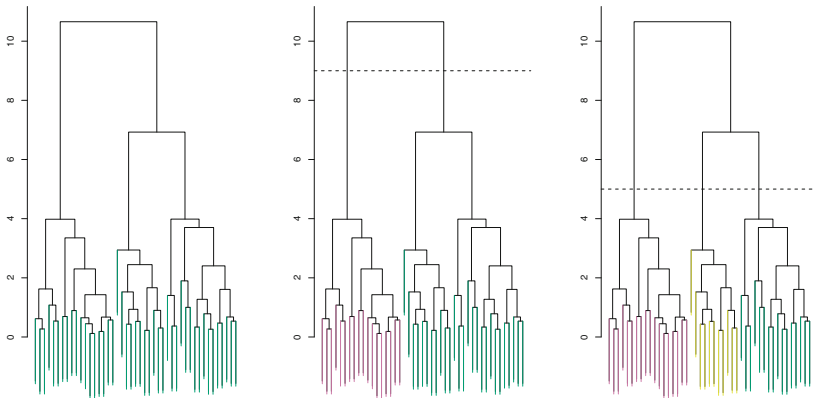
FIGURE 10.9. Left: *dendrogram obtained from hierarchically clustering the data from Figure 10.8 with complete linkage and Euclidean distance.* Center: *the dendrogram from the left-hand panel, cut at a height of nine (indicated by the dashed line). This cut results in two distinct clusters, shown in different colors.* Right: *the dendrogram from the left-hand panel, now cut at a height of five. This cut results in three distinct clusters, shown in different colors. Note that the colors were not used in clustering, but are simply used for display purposes in this figure.*

different the two observations are. Thus, observations that fuse at the very bottom of the tree are quite similar to each other, whereas observations that fuse close to the top of the tree will tend to be quite different.

This highlights a very important point in interpreting dendrograms that is often misunderstood. Consider the left-hand panel of Figure 10.10, which shows a simple dendrogram obtained from hierarchically clustering nine observations. One can see that observations 5 and 7 are quite similar to each other, since they fuse at the lowest point on the dendrogram. Observations 1 and 6 are also quite similar to each other. However, it is tempting but incorrect to conclude from the figure that observations 9 and 2 are quite similar to each other on the basis that they are located near each other on the dendrogram. In fact, based on the information contained in the dendrogram, observation 9 is no more similar to observation 2 than it is to observations 8, 5, and 7. (This can be seen from the right-hand panel of Figure 10.10, in which the raw data are displayed.) To put it mathematically, there are 2^{n-1} possible reorderings of the dendrogram, where n is the number of leaves. This is because at each of the $n-1$ points where fusions occur, the positions of the two fused branches could be swapped without affecting the meaning of the dendrogram. Therefore, we cannot draw conclusions about the similarity of two observations based on their proximity along the *horizontal axis*. Rather, we draw conclusions about the similarity of two observations based on the location on the *vertical axis* where branches containing those two observations first are fused.
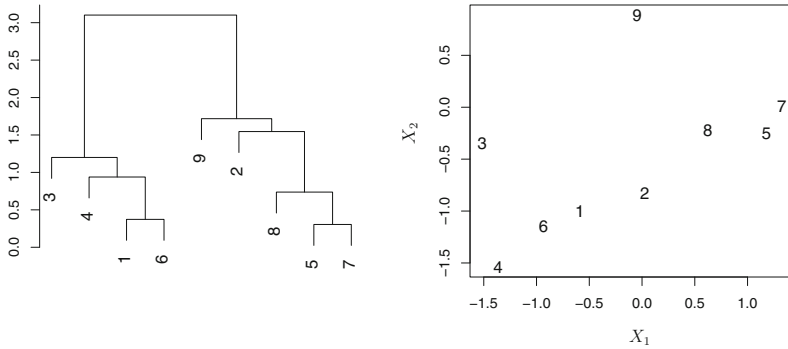
FIGURE 10.10. *An illustration of how to properly interpret a dendrogram with nine observations in two-dimensional space.* Left: *a dendrogram generated using Euclidean distance and complete linkage. Observations* 5 *and* 7 *are quite similar to each other, as are observations* 1 *and* 6. *However, observation* 9 *is* no more *similar to observation* 2 *than it is to observations* 8, 5, *and* 7, *even though observations* 9 *and* 2 *are close together in terms of horizontal distance. This is because observations* 2, 8, 5, *and* 7 *all fuse with observation* 9 *at the same height, approximately* 1.8. Right: *the raw data used to generate the dendrogram can be used to confirm that indeed, observation* 9 *is* no more *similar to observation* 2 *than it is to observations* 8, 5, *and* 7.

Now that we understand how to interpret the left-hand panel of Figure 10.9, we can move on to the issue of identifying clusters on the basis of a dendrogram. In order to do this, we make a horizontal cut across the dendrogram, as shown in the center and right-hand panels of Figure 10.9. The distinct sets of observations beneath the cut can be interpreted as clusters. In the center panel of Figure 10.9, cutting the dendrogram at a height of nine results in two clusters, shown in distinct colors. In the right-hand panel, cutting the dendrogram at a height of five results in three clusters. Further cuts can be made as one descends the dendrogram in order to obtain any number of clusters, between 1 (corresponding to no cut) and n (corresponding to a cut at height 0, so that each observation is in its own cluster). In other words, the height of the cut to the dendrogram serves the same role as the K in K-means clustering: it controls the number of clusters obtained.

Figure 10.9 therefore highlights a very attractive aspect of hierarchical clustering: one single dendrogram can be used to obtain any number of clusters. In practice, people often look at the dendrogram and select by eye a sensible number of clusters, based on the heights of the fusion and the number of clusters desired. In the case of Figure 10.9, one might choose to select either two or three clusters. However, often the choice of where to cut the dendrogram is not so clear.

The term *hierarchical* refers to the fact that clusters obtained by cutting the dendrogram at a given height are necessarily nested within the clusters obtained by cutting the dendrogram at any greater height. However, on an arbitrary data set, this assumption of hierarchical structure might be unrealistic. For instance, suppose that our observations correspond to a group of people with a 50–50 split of males and females, evenly split among Americans, Japanese, and French. We can imagine a scenario in which the best division into two groups might split these people by gender, and the best division into three groups might split them by nationality. In this case, the true clusters are not nested, in the sense that the best division into three groups does not result from taking the best division into two groups and splitting up one of those groups. Consequently, this situation could not be well-represented by hierarchical clustering. Due to situations such as this one, hierarchical clustering can sometimes yield *worse* (i.e. less accurate) results than K-means clustering for a given number of clusters.

The Hierarchical Clustering Algorithm

The hierarchical clustering dendrogram is obtained via an extremely simple algorithm. We begin by defining some sort of *dissimilarity* measure between each pair of observations. Most often, Euclidean distance is used; we will discuss the choice of dissimilarity measure later in this chapter. The algorithm proceeds iteratively. Starting out at the bottom of the dendrogram, each of the n observations is treated as its own cluster. The two clusters that are most similar to each other are then *fused* so that there now are $n-1$ clusters. Next the two clusters that are most similar to each other are fused again, so that there now are $n-2$ clusters. The algorithm proceeds in this fashion until all of the observations belong to one single cluster, and the dendrogram is complete. Figure 10.11 depicts the first few steps of the algorithm, for the data from Figure 10.9. To summarize, the hierarchical clustering algorithm is given in Algorithm 10.2.

This algorithm seems simple enough, but one issue has not been addressed. Consider the bottom right panel in Figure 10.11. How did we determine that the cluster $\{5,7\}$ should be fused with the cluster $\{8\}$? We have a concept of the dissimilarity between pairs of observations, but how do we define the dissimilarity between two clusters if one or both of the clusters contains multiple observations? The concept of dissimilarity between a pair of observations needs to be extended to a pair of *groups of observations*. This extension is achieved by developing the notion of *linkage*, which defines the dissimilarity between two groups of observa- linkage
tions. The four most common types of linkage—*complete*, *average*, *single*, and *centroid*—are briefly described in Table 10.2. Average, complete, and single linkage are most popular among statisticians. Average and complete

---

Algorithm 10.2 *Hierarchical Clustering*

---

1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.

2. For $i = n, n-1, \ldots, 2$:

   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.

   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

Linkage	*Description*
Complete	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *largest* of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *smallest* of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *average* of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable *inversions*.

TABLE 10.2. *A summary of the four most commonly-used types of linkage in hierarchical clustering.*

linkage are generally preferred over single linkage, as they tend to yield more balanced dendrograms. Centroid linkage is often used in genomics, but suffers from a major drawback in that an *inversion* can occur, whereby two clusters are fused at a height *below* either of the individual clusters in the dendrogram. This can lead to difficulties in visualization as well as in interpretation of the dendrogram. The dissimilarities computed in Step 2(b) of the hierarchical clustering algorithm will depend on the type of linkage used, as well as on the choice of dissimilarity measure. Hence, the resulting
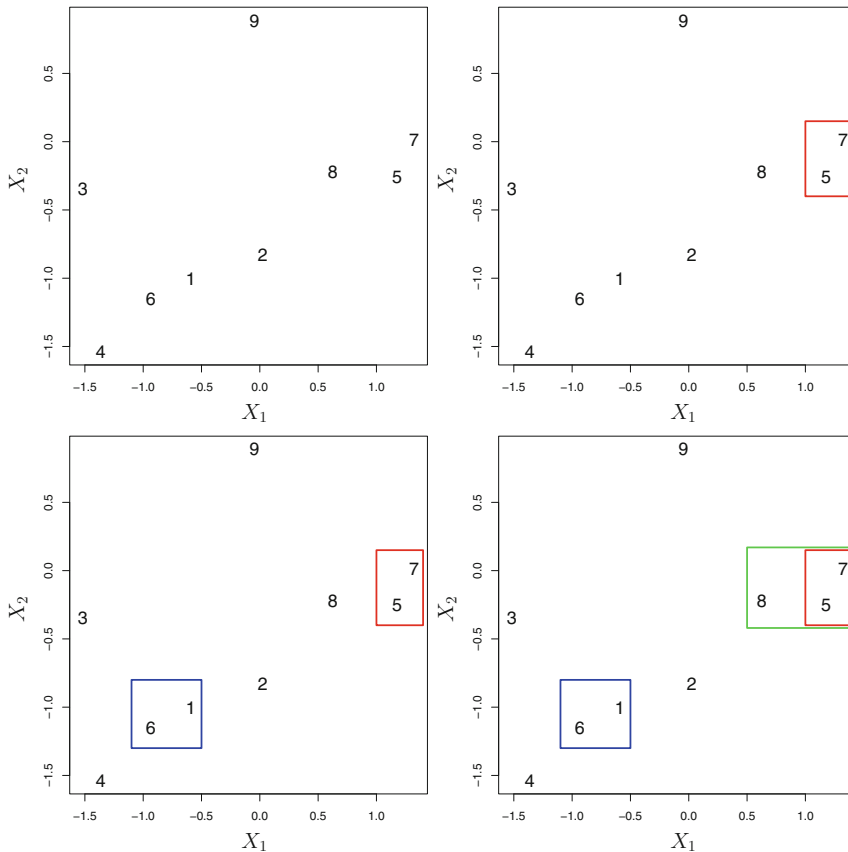
inversion

FIGURE 10.11. *An illustration of the first few steps of the hierarchical clustering algorithm, using the data from Figure 10.10, with complete linkage and Euclidean distance.* Top Left: *initially, there are nine distinct clusters,* $\{1\}, \{2\}, \ldots, \{9\}$. Top Right: *the two clusters that are closest together,* $\{5\}$ *and* $\{7\}$, *are fused into a single cluster.* Bottom Left: *the two clusters that are closest together,* $\{6\}$ *and* $\{1\}$, *are fused into a single cluster.* Bottom Right: *the two clusters that are closest together using* complete linkage, $\{8\}$ *and the cluster* $\{5, 7\}$, *are fused into a single cluster.*

dendrogram typically depends quite strongly on the type of linkage used, as is shown in Figure 10.12.

Choice of Dissimilarity Measure

Thus far, the examples in this chapter have used Euclidean distance as the dissimilarity measure. But sometimes other dissimilarity measures might be preferred. For example, *correlation-based distance* considers two observations to be similar if their features are highly correlated, even though the observed values may be far apart in terms of Euclidean distance. This is
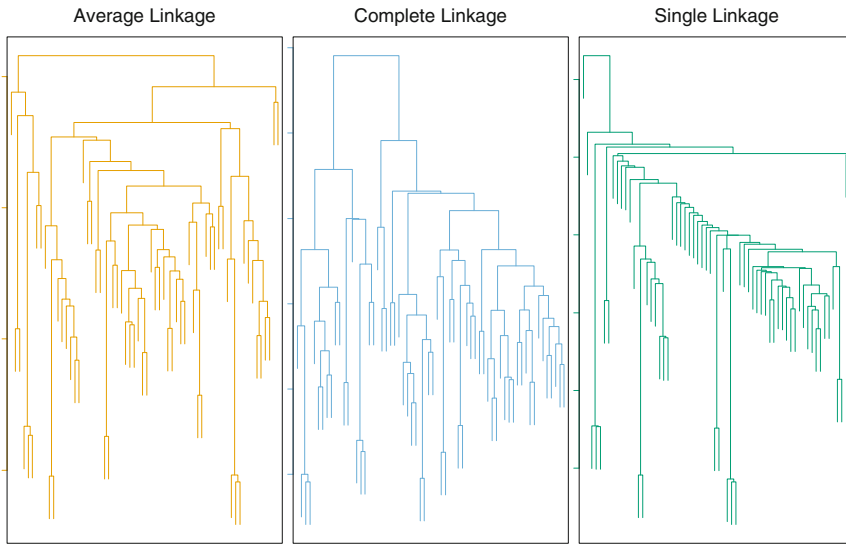
Average Linkage	Complete Linkage	Single Linkage

FIGURE 10.12. *Average, complete, and single linkage applied to an example data set. Average and complete linkage tend to yield more balanced clusters.*

an unusual use of correlation, which is normally computed between variables; here it is computed between the observation profiles for each pair of observations. Figure 10.13 illustrates the difference between Euclidean and correlation-based distance. Correlation-based distance focuses on the shapes of observation profiles rather than their magnitudes.

The choice of dissimilarity measure is very important, as it has a strong effect on the resulting dendrogram. In general, careful attention should be paid to the type of data being clustered and the scientific question at hand. These considerations should determine what type of dissimilarity measure is used for hierarchical clustering.

For instance, consider an online retailer interested in clustering shoppers based on their past shopping histories. The goal is to identify subgroups of *similar* shoppers, so that shoppers within each subgroup can be shown items and advertisements that are particularly likely to interest them. Suppose the data takes the form of a matrix where the rows are the shoppers and the columns are the items available for purchase; the elements of the data matrix indicate the number of times a given shopper has purchased a given item (i.e. a 0 if the shopper has never purchased this item, a 1 if the shopper has purchased it once, etc.) What type of dissimilarity measure should be used to cluster the shoppers? If Euclidean distance is used, then shoppers who have bought very few items overall (i.e. infrequent users of the online shopping site) will be clustered together. This may not be desirable. On the other hand, if correlation-based distance is used, then shoppers with similar preferences (e.g. shoppers who have bought items A and B but
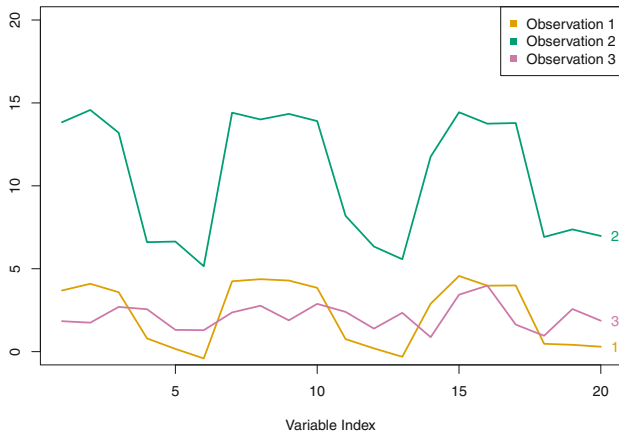
FIGURE 10.13. *Three observations with measurements on 20 variables are shown. Observations 1 and 3 have similar values for each variable and so there is a small Euclidean distance between them. But they are very weakly correlated, so they have a large correlation-based distance. On the other hand, observations 1 and 2 have quite different values for each variable, and so there is a large Euclidean distance between them. But they are highly correlated, so there is a small correlation-based distance between them.*

never items C or D) will be clustered together, even if some shoppers with these preferences are higher-volume shoppers than others. Therefore, for this application, correlation-based distance may be a better choice.

In addition to carefully selecting the dissimilarity measure used, one must also consider whether or not the variables should be scaled to have standard deviation one before the dissimilarity between the observations is computed. To illustrate this point, we continue with the online shopping example just described. Some items may be purchased more frequently than others; for instance, a shopper might buy ten pairs of socks a year, but a computer very rarely. High-frequency purchases like socks therefore tend to have a much larger effect on the inter-shopper dissimilarities, and hence on the clustering ultimately obtained, than rare purchases like computers. This may not be desirable. If the variables are scaled to have standard deviation one before the inter-observation dissimilarities are computed, then each variable will in effect be given equal importance in the hierarchical clustering performed. We might also want to scale the variables to have standard deviation one if they are measured on different scales; otherwise, the choice of units (e.g. centimeters versus kilometers) for a particular variable will greatly affect the dissimilarity measure obtained. It should come as no surprise that whether or not it is a good decision to scale the variables before computing the dissimilarity measure depends on the application at hand. An example is shown in Figure 10.14. We note that the issue of whether or not to scale the variables before performing clustering applies to K-means clustering as well.
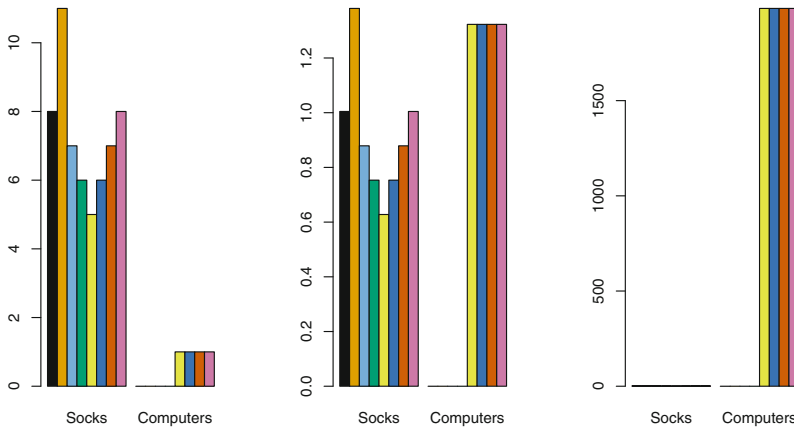
FIGURE 10.14. *An eclectic online retailer sells two items: socks and computers. Left: the number of pairs of socks, and computers, purchased by eight online shoppers is displayed. Each shopper is shown in a different color. If inter-observation dissimilarities are computed using Euclidean distance on the raw variables, then the number of socks purchased by an individual will drive the dissimilarities obtained, and the number of computers purchased will have little effect. This might be undesirable, since (1) computers are more expensive than socks and so the online retailer may be more interested in encouraging shoppers to buy computers than socks, and (2) a large difference in the number of socks purchased by two shoppers may be less informative about the shoppers' overall shopping preferences than a small difference in the number of computers purchased. Center: the same data is shown, after scaling each variable by its standard deviation. Now the number of computers purchased will have a much greater effect on the inter-observation dissimilarities obtained. Right: the same data are displayed, but now the y-axis represents the number of dollars spent by each online shopper on socks and on computers. Since computers are much more expensive than socks, now computer purchase history will drive the inter-observation dissimilarities obtained.*

10.3.3   Practical Issues in Clustering

Clustering can be a very useful tool for data analysis in the unsupervised setting. However, there are a number of issues that arise in performing clustering. We describe some of these issues here.

Small Decisions with Big Consequences

In order to perform clustering, some decisions must be made.

- Should the observations or features first be standardized in some way? For instance, maybe the variables should be centered to have mean zero and scaled to have standard deviation one.

- In the case of hierarchical clustering,

  - What dissimilarity measure should be used?
  - What type of linkage should be used?
  - Where should we cut the dendrogram in order to obtain clusters?

- In the case of K-means clustering, how many clusters should we look for in the data?

Each of these decisions can have a strong impact on the results obtained. In practice, we try several different choices, and look for the one with the most useful or interpretable solution. With these methods, there is no single right answer—any solution that exposes some interesting aspects of the data should be considered.

Validating the Clusters Obtained

Any time clustering is performed on a data set we will find clusters. But we really want to know whether the clusters that have been found represent true subgroups in the data, or whether they are simply a result of *clustering the noise.* For instance, if we were to obtain an independent set of observations, then would those observations also display the same set of clusters? This is a hard question to answer. There exist a number of techniques for assigning a p-value to a cluster in order to assess whether there is more evidence for the cluster than one would expect due to chance. However, there has been no consensus on a single best approach. More details can be found in Hastie et al. (2009).

Other Considerations in Clustering

Both K-means and hierarchical clustering will assign each observation to a cluster. However, sometimes this might not be appropriate. For instance, suppose that most of the observations truly belong to a small number of (unknown) subgroups, and a small subset of the observations are quite different from each other and from all other observations. Then since K-means and hierarchical clustering force *every* observation into a cluster, the clusters found may be heavily distorted due to the presence of outliers that do not belong to any cluster. Mixture models are an attractive approach for accommodating the presence of such outliers. These amount to a *soft* version of K-means clustering, and are described in Hastie et al. (2009).

In addition, clustering methods generally are not very robust to perturbations to the data. For instance, suppose that we cluster n observations, and then cluster the observations again after removing a subset of the n observations at random. One would hope that the two sets of clusters obtained would be quite similar, but often this is not the case!

A Tempered Approach to Interpreting the Results of Clustering

We have described some of the issues associated with clustering. However, clustering can be a very useful and valid statistical tool if used properly. We mentioned that small decisions in how clustering is performed, such as how the data are standardized and what type of linkage is used, can have a large effect on the results. Therefore, we recommend performing clustering with different choices of these parameters, and looking at the full set of results in order to see what patterns consistently emerge. Since clustering can be non-robust, we recommend clustering subsets of the data in order to get a sense of the robustness of the clusters obtained. Most importantly, we must be careful about how the results of a clustering analysis are reported. These results should not be taken as the absolute truth about a data set. Rather, they should constitute a starting point for the development of a scientific hypothesis and further study, preferably on an independent data set.

10.4   Lab 1: Principal Components Analysis

In this lab, we perform PCA on the `USArrests` data set, which is part of the base `R` package. The rows of the data set contain the 50 states, in alphabetical order.

```
> states=row.names(USArrests)
> states
```

The columns of the data set contain the four variables.

```
> names(USArrests)
[1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

We first briefly examine the data. We notice that the variables have vastly different means.

```
> apply(USArrests, 2, mean)
  Murder  Assault UrbanPop     Rape
    7.79   170.76    65.54    21.23
```

Note that the `apply()` function allows us to apply a function—in this case, the `mean()` function—to each row or column of the data set. The second input here denotes whether we wish to compute the mean of the rows, 1, or the columns, 2. We see that there are on average three times as many rapes as murders, and more than eight times as many assaults as rapes. We can also examine the variances of the four variables using the `apply()` function.

```
> apply(USArrests, 2, var)
  Murder  Assault UrbanPop     Rape
    19.0   6945.2    209.5     87.7
```

Not surprisingly, the variables also have vastly different variances: the `UrbanPop` variable measures the percentage of the population in each state living in an urban area, which is not a comparable number to the number of rapes in each state per 100,000 individuals. If we failed to scale the variables before performing PCA, then most of the principal components that we observed would be driven by the `Assault` variable, since it has by far the largest mean and variance. Thus, it is important to standardize the variables to have mean zero and standard deviation one before performing PCA.

We now perform principal components analysis using the `prcomp()` function, which is one of several functions in `R` that perform PCA.

prcomp()

```
> pr.out=prcomp(USArrests, scale=TRUE)
```

By default, the `prcomp()` function centers the variables to have mean zero. By using the option `scale=TRUE`, we scale the variables to have standard deviation one. The output from `prcomp()` contains a number of useful quantities.

```
> names(pr.out)
[1] "sdev"      "rotation" "center"    "scale"     "x"
```

The `center` and `scale` components correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```
> pr.out$center
  Murder   Assault  UrbanPop      Rape
    7.79    170.76     65.54     21.23
> pr.out$scale
  Murder   Assault  UrbanPop      Rape
    4.36     83.34     14.47      9.37
```

The `rotation` matrix provides the principal component loadings; each column of `pr.out$rotation` contains the corresponding principal component loading vector.[2]

```
> pr.out$rotation
             PC1     PC2     PC3     PC4
Murder    -0.536   0.418  -0.341   0.649
Assault   -0.583   0.188  -0.268  -0.743
UrbanPop  -0.278  -0.873  -0.378   0.134
Rape      -0.543  -0.167   0.818   0.089
```

We see that there are four distinct principal components. This is to be expected because there are in general $\min(n-1, p)$ informative principal components in a data set with n observations and p variables.

---

[2]This function names it the rotation matrix, because when we matrix-multiply the **X** matrix by `pr.out$rotation`, it gives us the coordinates of the data in the rotated coordinate system. These coordinates are the principal component scores.

Using the `prcomp()` function, we do not need to explicitly multiply the data by the principal component loading vectors in order to obtain the principal component score vectors. Rather the 50×4 matrix `x` has as its columns the principal component score vectors. That is, the kth column is the kth principal component score vector.

```
> dim(pr.out$x)
[1] 50   4
```

We can plot the first two principal components as follows:

```
> biplot(pr.out, scale=0)
```

The `scale=0` argument to `biplot()` ensures that the arrows are scaled to represent the loadings; other values for `scale` give slightly different biplots with different interpretations.

`biplot()`

Notice that this figure is a mirror image of Figure 10.1. Recall that the principal components are only unique up to a sign change, so we can reproduce Figure 10.1 by making a few small changes:

```
> pr.out$rotation=-pr.out$rotation
> pr.out$x=-pr.out$x
> biplot(pr.out, scale=0)
```

The `prcomp()` function also outputs the standard deviation of each principal component. For instance, on the `USArrests` data set, we can access these standard deviations as follows:

```
> pr.out$sdev
[1] 1.575 0.995 0.597 0.416
```

The variance explained by each principal component is obtained by squaring these:

```
> pr.var=pr.out$sdev^2
> pr.var
[1] 2.480 0.990 0.357 0.173
```

To compute the proportion of variance explained by each principal component, we simply divide the variance explained by each principal component by the total variance explained by all four principal components:

```
> pve=pr.var/sum(pr.var)
> pve
[1] 0.6201 0.2474 0.0891 0.0434
```

We see that the first principal component explains $62.0\,\%$ of the variance in the data, the next principal component explains $24.7\,\%$ of the variance, and so forth. We can plot the PVE explained by each component, as well as the cumulative PVE, as follows:

```
> plot(pve, xlab="Principal Component", ylab="Proportion of
    Variance Explained", ylim=c(0,1),type='b')
> plot(cumsum(pve), xlab="Principal Component", ylab="
    Cumulative Proportion of Variance Explained", ylim=c(0,1),
    type='b')
```

The result is shown in Figure 10.4. Note that the function `cumsum()` computes the cumulative sum of the elements of a numeric vector. For instance:

`cumsum()`

```
> a=c(1,2,8,-3)
> cumsum(a)
[1]   1   3 11   8
```

10.5   Lab 2: Clustering

10.5.1   K-Means Clustering

The function `kmeans()` performs K-means clustering in `R`. We begin with a simple simulated example in which there truly are two clusters in the data: the first 25 observations have a mean shift relative to the next 25 observations.

`kmeans()`

```
> set.seed(2)
> x=matrix(rnorm(50*2), ncol=2)
> x[1:25,1]=x[1:25,1]+3
> x[1:25,2]=x[1:25,2]-4
```

We now perform K-means clustering with $K = 2$.

```
> km.out=kmeans(x,2,nstart=20)
```

The cluster assignments of the 50 observations are contained in `km.out$cluster`.

```
> km.out$cluster
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1
[30] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The K-means clustering perfectly separated the observations into two clusters even though we did not supply any group information to `kmeans()`. We can plot the data, with each observation colored according to its cluster assignment.

```
> plot(x, col=(km.out$cluster+1), main="K-Means Clustering
    Results with K=2", xlab="", ylab="", pch=20, cex=2)
```

Here the observations can be easily plotted because they are two-dimensional. If there were more than two variables then we could instead perform PCA and plot the first two principal components score vectors.

In this example, we knew that there really were two clusters because we generated the data. However, for real data, in general we do not know the true number of clusters. We could instead have performed K-means clustering on this example with $K = 3$.

```
> set.seed(4)
> km.out=kmeans(x,3,nstart=20)
> km.out
K-means clustering with 3 clusters of sizes 10, 23, 17
```

```
Cluster means:
        [,1]           [,2]
1   2.3001545  -2.69622023
2  -0.3820397  -0.08740753
3   3.7789567  -4.56200798

Clustering vector:
 [1] 3 1 3 1 3 3 3 1 3 1 3 1 3 1 3 1 3 3 3 3 3 3 1 3 3 3 2 2 2 2
     2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2

Within cluster sum of squares by cluster:
[1] 19.56137 52.67700 25.74089
 (between_SS / total_SS =   79.3 %)

Available components:

[1] "cluster"      "centers"       "totss"           "withinss"
    "tot.withinss" "betweenss"     "size"
> plot(x, col=(km.out$cluster+1), main="K-Means Clustering
    Results with K=3", xlab="", ylab="", pch=20, cex=2)
```

When $K = 3$, K-means clustering splits up the two clusters.

To run the `kmeans()` function in `R` with multiple initial cluster assign-ments, we use the `nstart` argument. If a value of `nstart` greater than one is used, then K-means clustering will be performed using multiple random assignments in Step 1 of Algorithm 10.1, and the `kmeans()` function will report only the best results. Here we compare using `nstart=1` to `nstart=20`.

```
> set.seed(3)
> km.out=kmeans(x,3,nstart=1)
> km.out$tot.withinss
[1] 104.3319
> km.out=kmeans(x,3,nstart=20)
> km.out$tot.withinss
[1] 97.9793
```

Note that `km.out$tot.withinss` is the total within-cluster sum of squares, which we seek to minimize by performing K-means clustering (Equation 10.11). The individual within-cluster sum-of-squares are contained in the vector `km.out$withinss`.

We *strongly* recommend always running K-means clustering with a large value of `nstart`, such as 20 or 50, since otherwise an undesirable local optimum may be obtained.

When performing K-means clustering, in addition to using multiple ini-tial cluster assignments, it is also important to set a random seed using the `set.seed()` function. This way, the initial cluster assignments in Step 1 can be replicated, and the K-means output will be fully reproducible.

10.5.2 *Hierarchical Clustering*

The `hclust()` function implements hierarchical clustering in `R`. In the fol-
lowing example we use the data from Section 10.5.1 to plot the hierarchical
clustering dendrogram using complete, single, and average linkage cluster-
ing, with Euclidean distance as the dissimilarity measure. We begin by
clustering observations using complete linkage. The `dist()` function is used
to compute the 50×50 inter-observation Euclidean distance matrix.

hclust()

dist()

```
> hc.complete =hclust(dist(x), method="complete")
```

We could just as easily perform hierarchical clustering with average or
single linkage instead:

```
> hc.average =hclust(dist(x), method="average")
> hc.single =hclust(dist(x), method="single")
```

We can now plot the dendrograms obtained using the usual `plot()` function.
The numbers at the bottom of the plot identify each observation.

```
> par(mfrow=c(1,3))
> plot(hc.complete,main="Complete Linkage", xlab="", sub="",
    cex=.9)
> plot(hc.average, main="Average Linkage", xlab="", sub="",
    cex=.9)
> plot(hc.single, main="Single Linkage", xlab="", sub="",
    cex=.9)
```

To determine the cluster labels for each observation associated with a
given cut of the dendrogram, we can use the `cutree()` function:

cutree()

```
> cutree(hc.complete, 2)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
[30] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
> cutree(hc.average, 2)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
[30] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2
> cutree(hc.single, 2)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[30] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

For this data, complete and average linkage generally separate the observa-
tions into their correct groups. However, single linkage identifies one point
as belonging to its own cluster. A more sensible answer is obtained when
four clusters are selected, although there are still two singletons.

```
> cutree(hc.single, 4)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 3 3 3 3
[30] 3 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3
```

To scale the variables before performing hierarchical clustering of the
observations, we use the `scale()` function:

scale()

```
> xsc=scale(x)
> plot(hclust(dist(xsc), method="complete"), main="Hierarchical
    Clustering with Scaled Features")
```

Correlation-based distance can be computed using the `as.dist()` function, which converts an arbitrary square symmetric matrix into a form that the `hclust()` function recognizes as a distance matrix. However, this only makes sense for data with at least three features since the absolute correlation between any two observations with measurements on two features is always 1. Hence, we will cluster a three-dimensional data set.

`as.dist()`

```
> x=matrix(rnorm(30*3), ncol=3)
> dd=as.dist(1-cor(t(x)))
> plot(hclust(dd, method="complete"), main="Complete Linkage
    with Correlation-Based Distance", xlab="", sub="")
```

10.6 Lab 3: NCI60 Data Example

Unsupervised techniques are often used in the analysis of genomic data. In particular, PCA and hierarchical clustering are popular tools. We illustrate these techniques on the `NCI60` cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines.

```
> library(ISLR)
> nci.labs=NCI60$labs
> nci.data=NCI60$data
```

Each cell line is labeled with a cancer type. We do not make use of the cancer types in performing PCA and clustering, as these are unsupervised techniques. But after performing PCA and clustering, we will check to see the extent to which these cancer types agree with the results of these unsupervised techniques.

The data has 64 rows and 6,830 columns.

```
> dim(nci.data)
[1]   64 6830
```

We begin by examining the cancer types for the cell lines.

```
> nci.labs[1:4]
[1] "CNS"   "CNS"    "CNS"    "RENAL"
> table(nci.labs)
nci.labs
     BREAST          CNS        COLON K562A-repro K562B-repro
         7            5            7            1            1
   LEUKEMIA MCF7A-repro MCF7D-repro     MELANOMA         NSCLC
         6            1            1            8            9
     OVARIAN     PROSTATE        RENAL      UNKNOWN
         6            2            9            1
```

10.6.1 PCA on the NCI60 Data

We first perform PCA on the data after scaling the variables (genes) to
have standard deviation one, although one could reasonably argue that it
is better not to scale the genes.

```
> pr.out=prcomp(nci.data, scale=TRUE)
```

We now plot the first few principal component score vectors, in order to
visualize the data. The observations (cell lines) corresponding to a given
cancer type will be plotted in the same color, so that we can see to what
extent the observations within a cancer type are similar to each other. We
first create a simple function that assigns a distinct color to each element
of a numeric vector. The function will be used to assign a color to each of
the 64 cell lines, based on the cancer type to which it corresponds.

```
Cols=function(vec){
+     cols=rainbow(length(unique(vec)))
+     return(cols[as.numeric(as.factor(vec))])
+   }
```

Note that the `rainbow()` function takes as its argument a positive integer,
and returns a vector containing that number of distinct colors. We now can
plot the principal component score vectors.

`rainbow()`

```
> par(mfrow=c(1,2))
> plot(pr.out$x[,1:2], col=Cols(nci.labs), pch=19,
    xlab="Z1",ylab="Z2")
> plot(pr.out$x[,c(1,3)], col=Cols(nci.labs), pch=19,
    xlab="Z1",ylab="Z3")
```

The resulting plots are shown in Figure 10.15. On the whole, cell lines
corresponding to a single cancer type do tend to have similar values on the
first few principal component score vectors. This indicates that cell lines
from the same cancer type tend to have pretty similar gene expression
levels.

   We can obtain a summary of the proportion of variance explained (PVE)
of the first few principal components using the `summary()` method for a
`prcomp` object (we have truncated the printout):

```
> summary(pr.out)
Importance of components:
                          PC1     PC2     PC3     PC4     PC5
Standard deviation     27.853 21.4814 19.8205 17.0326 15.9718
Proportion of Variance  0.114  0.0676  0.0575  0.0425  0.0374
Cumulative Proportion   0.114  0.1812  0.2387  0.2812  0.3185
```

Using the `plot()` function, we can also plot the variance explained by the
first few principal components.

```
> plot(pr.out)
```

Note that the height of each bar in the bar plot is given by squaring the
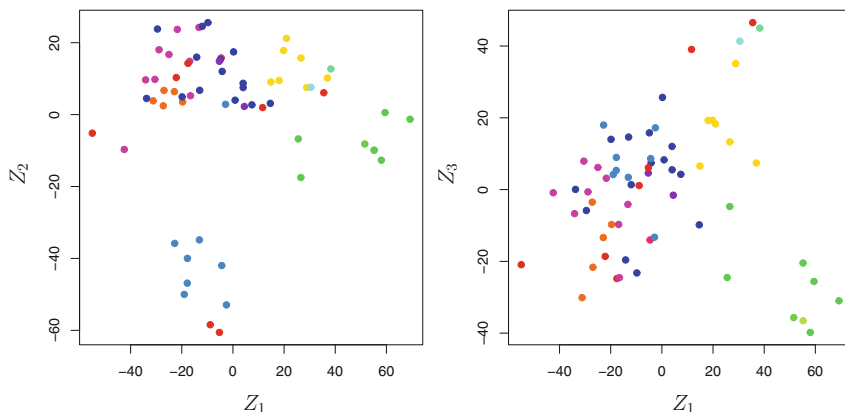corresponding element of `pr.out$sdev`. However, it is more informative to

FIGURE 10.15. *Projections of the* `NCI60` *cancer cell lines onto the first three principal components (in other words, the scores for the first three principal components). On the whole, observations belonging to a single cancer type tend to lie near each other in this low-dimensional space. It would not have been possible to visualize the data without using a dimension reduction method such as PCA, since based on the full data set there are $\binom{6,830}{2}$ possible scatterplots, none of which would have been particularly informative.*

plot the PVE of each principal component (i.e. a scree plot) and the cumulative PVE of each principal component. This can be done with just a little work.

```
> pve=100*pr.out$sdev^2/sum(pr.out$sdev^2)
> par(mfrow=c(1,2))
> plot(pve, type="o", ylab="PVE", xlab="Principal Component",
    col="blue")
> plot(cumsum(pve), type="o", ylab="Cumulative PVE", xlab="
    Principal Component", col="brown3")
```

(Note that the elements of `pve` can also be computed directly from the summary, `summary(pr.out)$importance[2,]`, and the elements of `cumsum(pve)` are given by `summary(pr.out)$importance[3,]`.) The resulting plots are shown in Figure 10.16. We see that together, the first seven principal components explain around 40 % of the variance in the data. This is not a huge amount of the variance. However, looking at the scree plot, we see that while each of the first seven principal components explain a substantial amount of variance, there is a marked decrease in the variance explained by further principal components. That is, there is an *elbow* in the plot after approximately the seventh principal component. This suggests that there may be little benefit to examining more than seven or so principal components (though even examining seven principal components may be difficult).
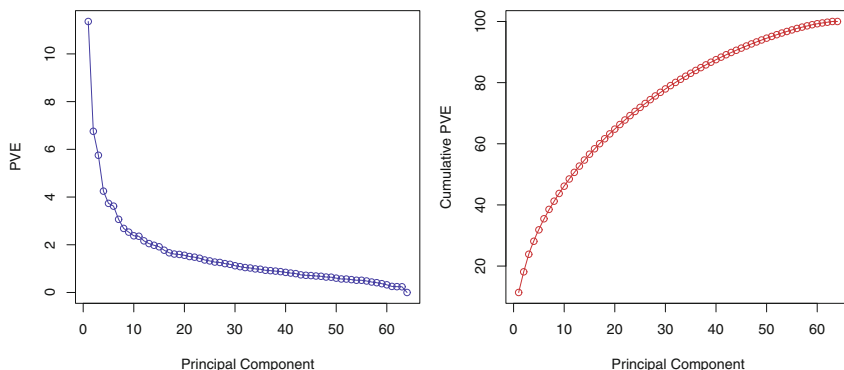
FIGURE 10.16. *The PVE of the principal components of the* `NCI60` *cancer cell line microarray data set.* Left: *the PVE of each principal component is shown.* Right: *the cumulative PVE of the principal components is shown. Together, all principal components explain 100 % of the variance.*

10.6.2   Clustering the Observations of the NCI60 Data

We now proceed to hierarchically cluster the cell lines in the `NCI60` data, with the goal of finding out whether or not the observations cluster into distinct types of cancer. To begin, we standardize the variables to have mean zero and standard deviation one. As mentioned earlier, this step is optional and should be performed only if we want each gene to be on the same *scale*.

```
> sd.data=scale(nci.data)
```

We now perform hierarchical clustering of the observations using complete, single, and average linkage. Euclidean distance is used as the dissimilarity measure.

```
> par(mfrow=c(1,3))
> data.dist=dist(sd.data)
> plot(hclust(data.dist), labels=nci.labs, main="Complete
    Linkage", xlab="", sub="",ylab="")
> plot(hclust(data.dist, method="average"), labels=nci.labs,
    main="Average Linkage", xlab="", sub="",ylab="")
> plot(hclust(data.dist, method="single"), labels=nci.labs,
    main="Single Linkage", xlab="", sub="",ylab="")
```

The results are shown in Figure 10.17. We see that the choice of linkage certainly does affect the results obtained. Typically, single linkage will tend to yield *trailing* clusters: very large clusters onto which individual observations attach one-by-one. On the other hand, complete and average linkage tend to yield more balanced, attractive clusters. For this reason, complete and average linkage are generally preferred to single linkage. Clearly cell lines within a single cancer type do tend to cluster together, although the
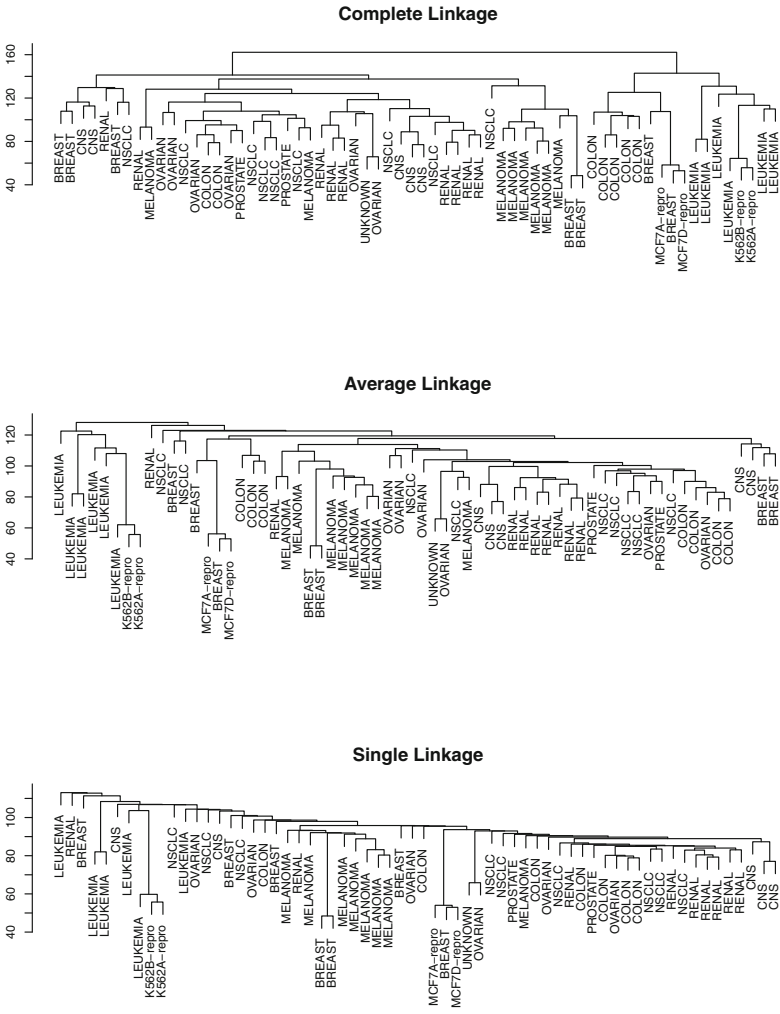
FIGURE 10.17. *The* `NCI60` *cancer cell line microarray data, clustered with average, complete, and single linkage, and using Euclidean distance as the dissimilarity measure. Complete and average linkage tend to yield evenly sized clusters whereas single linkage tends to yield extended clusters to which single leaves are fused one by one.*

clustering is not perfect. We will use complete linkage hierarchical clustering for the analysis that follows.

We can cut the dendrogram at the height that will yield a particular number of clusters, say four:

```
> hc.out=hclust(dist(sd.data))
> hc.clusters=cutree(hc.out,4)
> table(hc.clusters,nci.labs)
```

There are some clear patterns. All the leukemia cell lines fall in cluster 3, while the breast cancer cell lines are spread out over three different clusters. We can plot the cut on the dendrogram that produces these four clusters:

```
> par(mfrow=c(1,1))
> plot(hc.out, labels=nci.labs)
> abline(h=139, col="red")
```

The `abline()` function draws a straight line on top of any existing plot in R. The argument `h=139` plots a horizontal line at height 139 on the dendrogram; this is the height that results in four distinct clusters. It is easy to verify that the resulting clusters are the same as the ones we obtained using `cutree(hc.out,4)`.

Printing the output of `hclust` gives a useful brief summary of the object:

```
> hc.out

Call:
hclust(d = dist(dat))

Cluster method   : complete
Distance         : euclidean
Number of objects: 64
```

We claimed earlier in Section 10.3.2 that K-means clustering and hierarchical clustering with the dendrogram cut to obtain the same number of clusters can yield very different results. How do these NCI60 hierarchical clustering results compare to what we get if we perform K-means clustering with $K = 4$?

```
> set.seed(2)
> km.out=kmeans(sd.data, 4, nstart=20)
> km.clusters=km.out$cluster
> table(km.clusters,hc.clusters)
           hc.clusters
km.clusters  1  2  3  4
          1 11  0  0  9
          2  0  0  8  0
          3  9  0  0  0
          4 20  7  0  0
```

We see that the four clusters obtained using hierarchical clustering and K-means clustering are somewhat different. Cluster 2 in K-means clustering is identical to cluster 3 in hierarchical clustering. However, the other clusters

differ: for instance, cluster 4 in K-means clustering contains a portion of
the observations assigned to cluster 1 by hierarchical clustering, as well as
all of the observations assigned to cluster 2 by hierarchical clustering.

Rather than performing hierarchical clustering on the entire data matrix,
we can simply perform hierarchical clustering on the first few principal
component score vectors, as follows:

```
> hc.out=hclust(dist(pr.out$x[,1:5]))
> plot(hc.out, labels=nci.labs, main="Hier. Clust. on First
    Five Score Vectors")
> table(cutree(hc.out,4), nci.labs)
```

Not surprisingly, these results are different from the ones that we obtained
when we performed hierarchical clustering on the full data set. Sometimes
performing clustering on the first few principal component score vectors
can give better results than performing clustering on the full data. In this
situation, we might view the principal component step as one of denois-
ing the data. We could also perform K-means clustering on the first few
principal component score vectors rather than the full data set.

10.7 Exercises

Conceptual

1. This problem involves the K-means clustering algorithm.
   (a) Prove (10.12).

   (b) On the basis of this identity, argue that the K-means clustering
       algorithm (Algorithm 10.1) decreases the objective (10.11) at
       each iteration.

2. Suppose that we have four observations, for which we compute a
   dissimilarity matrix, given by

$$
\begin{bmatrix}
 & 0.3 & 0.4 & 0.7 \\
0.3 & & 0.5 & 0.8 \\
0.4 & 0.5 & & 0.45 \\
0.7 & 0.8 & 0.45 &
\end{bmatrix}.
$$

   For instance, the dissimilarity between the first and second obser-
   vations is 0.3, and the dissimilarity between the second and fourth
   observations is 0.8.

   (a) On the basis of this dissimilarity matrix, sketch the dendrogram
       that results from hierarchically clustering these four observa-
       tions using complete linkage. Be sure to indicate on the plot the
       height at which each fusion occurs, as well as the observations
       corresponding to each leaf in the dendrogram.

(b) Repeat (a), this time using single linkage clustering.

(c) Suppose that we cut the dendogram obtained in (a) such that two clusters result. Which observations are in each cluster?

(d) Suppose that we cut the dendogram obtained in (b) such that two clusters result. Which observations are in each cluster?

(e) It is mentioned in the chapter that at each fusion in the dendrogram, the position of the two clusters being fused can be swapped without changing the meaning of the dendrogram. Draw a dendrogram that is equivalent to the dendrogram in (a), for which two or more of the leaves are repositioned, but for which the meaning of the dendrogram is the same.

3. In this problem, you will perform K-means clustering manually, with $K = 2$, on a small example with $n = 6$ observations and $p = 2$ features. The observations are as follows.

Obs.	X_1	X_2
1	1	4
2	1	3
3	0	4
4	5	1
5	6	2
6	4	0

(a) Plot the observations.

(b) Randomly assign a cluster label to each observation. You can use the `sample()` command in `R` to do this. Report the cluster labels for each observation.

(c) Compute the centroid for each cluster.

(d) Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

(e) Repeat (c) and (d) until the answers obtained stop changing.

(f) In your plot from (a), color the observations according to the cluster labels obtained.

4. Suppose that for a particular data set, we perform hierarchical clustering using single linkage and using complete linkage. We obtain two dendrograms.

(a) At a certain point on the single linkage dendrogram, the clusters $\{1, 2, 3\}$ and $\{4, 5\}$ fuse. On the complete linkage dendrogram, the clusters $\{1, 2, 3\}$ and $\{4, 5\}$ also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

(b) At a certain point on the single linkage dendrogram, the clusters {5} and {6} fuse. On the complete linkage dendrogram, the clusters {5} and {6} also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

5. In words, describe the results that you would expect if you performed K-means clustering of the eight shoppers in Figure 10.14, on the basis of their sock and computer purchases, with $K = 2$. Give three answers, one for each of the variable scalings displayed. Explain.

6. A researcher collects expression measurements for 1,000 genes in 100 tissue samples. The data can be written as a $1,000 \times 100$ matrix, which we call $\mathbf{X}$, in which each row represents a gene and each column a tissue sample. Each tissue sample was processed on a different day, and the columns of $\mathbf{X}$ are ordered so that the samples that were processed earliest are on the left, and the samples that were processed later are on the right. The tissue samples belong to two groups: control (C) and treatment (T). The C and T samples were processed in a random order across the days. The researcher wishes to determine whether each gene's expression measurements differ between the treatment and control groups.

As a pre-analysis (before comparing T versus C), the researcher performs a principal component analysis of the data, and finds that the first principal component (a vector of length 100) has a strong linear trend from left to right, and explains 10 % of the variation. The researcher now remembers that each patient sample was run on one of two machines, A and B, and machine A was used more often in the earlier times while B was used more often later. The researcher has a record of which sample was run on which machine.

(a) Explain what it means that the first principal component "explains 10 % of the variation".

(b) The researcher decides to replace the (j, i)th element of $\mathbf{X}$ with

$$x_{ji} - \phi_{j1} z_{i1}$$

where z_{i1} is the ith score, and ϕ_{j1} is the jth loading, for the first principal component. He will then perform a two-sample t-test on each gene in this new data set in order to determine whether its expression differs between the two conditions. Critique this idea, and suggest a better approach. (The principal component analysis is performed on $\mathbf{X}^T$).

(c) Design and run a small simulation experiment to demonstrate the superiority of your idea.

Applied

7. In the chapter, we mentioned the use of correlation-based distance and Euclidean distance as dissimilarity measures for hierarchical clustering. It turns out that these two measures are almost equivalent: if each observation has been centered to have mean zero and standard deviation one, and if we let r_{ij} denote the correlation between the ith and jth observations, then the quantity $1 - r_{ij}$ is proportional to the squared Euclidean distance between the ith and jth observations.

   On the `USArrests` data, show that this proportionality holds.

   Hint: The Euclidean distance can be calculated using the `dist()` *function, and correlations can be calculated using the* `cor()` *function.*

8. In Section 10.2.3, a formula for calculating PVE was given in Equation 10.8. We also saw that the PVE can be obtained using the `sdev` output of the `prcomp()` function.

   On the `USArrests` data, calculate PVE in two ways:

   (a) Using the `sdev` output of the `prcomp()` function, as was done in Section 10.2.3.

   (b) By applying Equation 10.8 directly. That is, use the `prcomp()` function to compute the principal component loadings. Then, use those loadings in Equation 10.8 to obtain the PVE.

   These two approaches should give the same results.

   Hint: You will only obtain the same results in (a) and (b) if the same data is used in both cases. For instance, if in (a) you performed `prcomp()` *using centered and scaled variables, then you must center and scale the variables before applying Equation 10.3 in (b).*

9. Consider the `USArrests` data. We will now perform hierarchical clustering on the states.

   (a) Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

   (b) Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which clusters?

   (c) Hierarchically cluster the states using complete linkage and Euclidean distance, *after scaling the variables to have standard deviation one.*

   (d) What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.

10. In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

   (a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

   Hint: There are a number of functions in R *that you can use to generate data. One example is the* `rnorm()` *function;* `runif()` *is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.*

   (b) Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors.

   (c) Perform K-means clustering of the observations with $K = 3$. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

   Hint: You can use the `table()` *function in* R *to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.*

   (d) Perform K-means clustering with $K = 2$. Describe your results.

   (e) Now perform K-means clustering with $K = 4$, and describe your results.

   (f) Now perform K-means clustering with $K = 3$ on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60×2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results.

   (g) Using the `scale()` function, perform K-means clustering with $K = 3$ on the data *after scaling each variable to have standard deviation one.* How do these results compare to those obtained in (b)? Explain.

11. On the book website, www.StatLearning.com, there is a gene expression data set (`Ch10Ex11.csv`) that consists of 40 tissue samples with measurements on 1,000 genes. The first 20 samples are from healthy patients, while the second 20 are from a diseased group.

(a) Load in the data using `read.csv()`. You will need to select `header=F`.

(b) Apply hierarchical clustering to the samples using correlation-based distance, and plot the dendrogram. Do the genes separate the samples into the two groups? Do your results depend on the type of linkage used?

(c) Your collaborator wants to know which genes differ the most across the two groups. Suggest a way to answer this question, and apply it here.