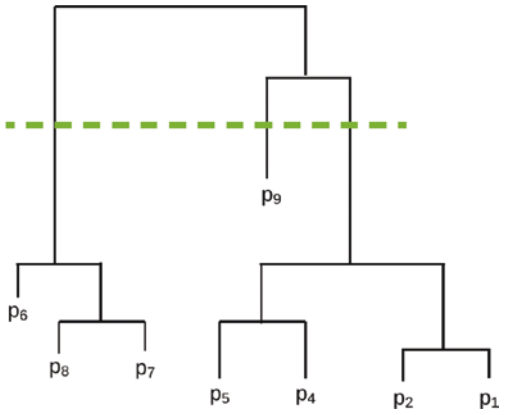

Figure 25-9. *Dendrogram cut*

Hierarchical Clustering with the SciPy Package

This example implements hierarchical or agglomerative clustering with SciPy. The 'scipy.cluster.hierarchy' package has simple methods for performing hierarchical clustering and plotting dendrograms. This example uses the 'complete' linkage method. The plot of the dendrogram is shown in Figure 25-10.

```
# import packages
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster import hierarchy

Z = hierarchy.linkage(X, method='complete')

plt.figure()
dn = hierarchy.dendrogram(Z, truncate_mode='lastp')
```

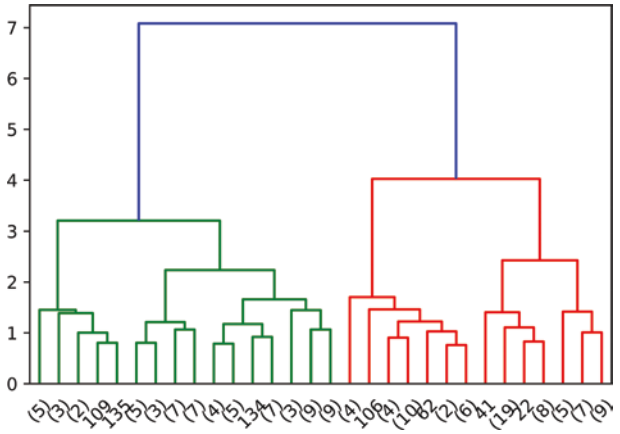

Figure 25-10. *Dendrogram produced by hierarchical clustering*

This chapter reviewed the pros and cons of *K*-means and hierarchical clustering. Both hierarchical and *K*-means are susceptible to perturbations in the dataset and can give very different results if a few data points are removed or added. Also, it is crucial to standardize the dataset features (i.e., to subtract each element in the feature from its mean and divide by its standard deviation or by the range) before performing clustering. This ensures that the features are within similar numeric bounds and have tempered or measured distances in the feature space.

The results of these clustering algorithms also depend on a wide range of considerations such as the choice of *K* for *K*-means, and for hierarchical clustering, the choice of dissimilarity measure, the type of linkage, and where to cut the dendrogram all affect the final result of the clusters. Hence, to get the best out of clustering, it is best to perform a grid search and try out all these different configurations in order to get a measured view on the robustness of the results before applying into your learning pipeline or using as a model to explain the dataset.

In the next chapter, we will discuss principal component analysis (PCA) as an unsupervised machine learning algorithm for finding low-dimensional feature subspaces that capture the variability in the dataset.

CHAPTER 26

Principal Component Analysis (PCA)

Principal component analysis (PCA) is an essential algorithm in machine learning. It is a mathematical method for evaluating the principal components of a dataset. The principal components are a set of vectors in high-dimensional space that capture the variance (i.e., spread) or variability of the feature space.

The goal of computing principal components is to find a low-dimensional feature sub-space that captures as much information as possible from the original higher-dimensional features of the dataset.

PCA is particularly useful for simplifying data visualization of high-dimensional features by reducing the dimensions of the dataset to a lower sub-space. For example, since we can easily visualize relationships on a 2-D plane using scatter diagrams, it will be useful to condense an n-dimensional space into two dimensions that retain as much information as possible in the n-dimensional dataset. This technique is popularly called dimensionality reduction.

How Are Principal Components Computed

The mathematical details for computing principal components are somewhat involved. This section will instead provide a conceptual but solid overview of this process.

The first step is to find the covariance matrix of the dataset. The covariance matrix captures the linear relationship between variables or features in the dataset. In a covariance matrix, an increasingly positive number represents a growing relationship, while the converse is represented by an increasingly negative number. Numbers around zero indicate a non-linear relationship between the variables. The covariance matrix is a square matrix (that means it has the same rows and columns). Hence, given a dataset with m rows and p columns, the covariance matrix will be a $m \times p$ matrix.

E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_26