Python机器学习基础教程

Stephen CUI[1]

February 17, 2023

[1]cuixuanStephen@gmail.com

Chapter 1

无监督学习与预处理

无监督学习包括没有已知输出、没有老师指导学习算法的各种机器学习。在无监督学习中，学习算法只有输入数据，并需要从这些数据中提取知识。

1.1 无监督学习的类型

这里将研究两种类型的无监督学习：数据集变换与聚类。

数据集的**无监督变换**（unsupervised transformation）是创建数据新的表示的算法，与数据的原始表示相比，新的表示可能更容易被人或其他机器学习算法所理解。无监督变换的一个常见应用是降维（dimensionality reduction），它接受包含许多特征的数据的高维表示，并找到表示该数据的一种新方法，用较少的特征就可以概括其重要特性。降维的一个常见应用是为了可视化将数据降为二维。

无监督变换的另一个应用是找到“构成”数据的各个组成部分。这方面的一个例子就是对文本文档集合进行主题提取。

聚类算法（clustering algorithm）将数据划分成不同的组，每组包含相似的物项。

1.2 无监督学习的挑战

无监督学习的一个主要挑战就是评估算法是否学到了有用的东西。无监督学习算法一般用于不包含任何标签信息的数据，所以我们不知道正确的输出应该是什么。因此很难判断一个模型是否“表现很好”。通常来说，评估无监督算法结果的唯一方法就是人工检查。

因此，如果数据科学家想要更好地理解数据，那么无监督算法通常可用于探索性的目的，而不是作为大型自动化系统的一部分。无监督算法的另一个常见应用是作为监督算法的预处理步骤。学习数据的一种新表示，有时可以提高监督算法的精度，或者可以减少内存占用和时间开销。

虽然预处理和缩放通常与监督学习算法一起使用，但缩放方法并没有用到与“监督”有关的信息，所以它是无监督的。

1.3 预处理与缩放

一些算法（如神经网络和 SVM）对数据缩放非常敏感。因此，通常的做法是对特征进行调节，使数据表示更适合于这些算法。通常来说，这是对数据的一种简单的按特征的缩放和移动。Figure 1.1 给出了一个简单的例子：

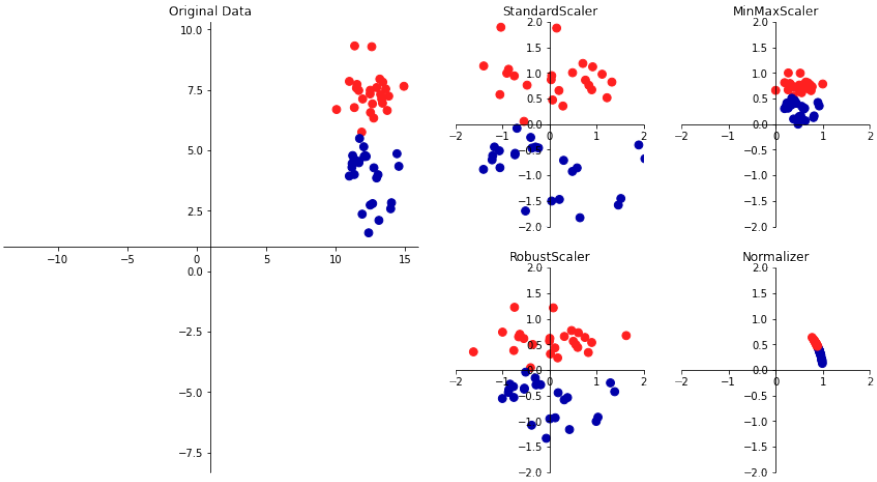

Figure 1.1: Different ways to rescale and preprocess a dataset

1.3.1 不同类型的预处理

Table 1.1: 一些缩放方式

缩放方式	作用位置	描述
StandardScaler	特征	确保每个特征的平均值为 0、方差为 1，使所有特征都位于同一量级。但这种缩放不能保证特征任何特定的最大值和最小值。
RobustScaler	特征	与StandardScaler 类似，确保每个特征的统计属性都位于同一范围。但 RobustScaler 使用的是中位数和四分位数 1，而不是平均值和方差。这样 RobustScaler 会忽略与其他点有很大不同的数据点（比如测量误差）。这些与众不同的数据点也叫异常值（outlier），可能会给其他缩放方法造成麻烦。
MinMaxScaler	特征	移动数据，使所有特征都刚好位于 0 到 1 之间。
Normalizer	样本	它对每个数据点进行缩放，使得特征向量的欧式长度等于 1。换句话说，它将一个数据点投射到半径为 1 的圆上（对于更高维度的情况，是球面）。这意味着每个数据点的缩放比例都不相同（乘以其长度的倒数）。如果只有数据的方向（或角度）是重要的，而特征向量的长度无关紧要，那么通常会使用这种归一化。

1.3.2 应用数据变换

常在应用监督学习算法之前使用预处理方法（比如缩放）。首先加载数据集并将其分为训练集和测试集（我们需要分开的训练集和数据集来对预处理后构建的监督模型进行评估）。

与之前构建的监督模型一样，我们首先导入实现预处理的类，然后将其实例化，然后，使用 fit 方法拟合缩放器（scaler），并将其应用于训练数据。对于 MinMaxScaler 来说，fit 方法计算训练集中每个特征的最大值和最小值。在对缩放器调用 fit 时只提供了 `X_train`，而不用 `y_train`。为了应用刚刚学习的变换（即对训练数据进行实际缩放），我们使用缩放器的 transform方法。在 scikit-learn 中，每当模型返回数据的一种新表示时，都可以使用 transform 方法。变换后的数据形状与原始数据相同，特征只是

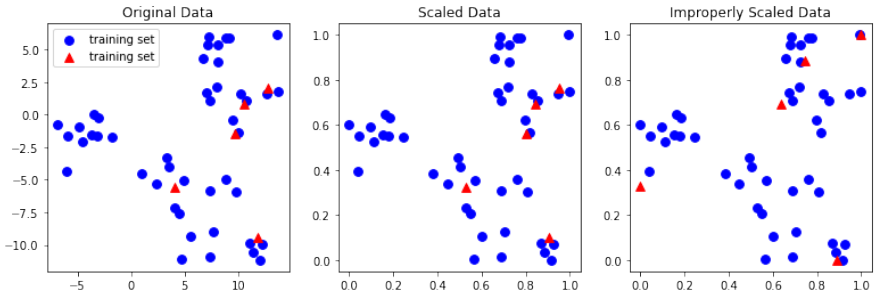

Figure 1.2: Effect of scaling training and test data

发生了移动和缩放。

除了对训练数据做变换，还需要对测试集进行变换。

你可以发现，对测试集缩放后的最大值和最小值不是 1 和 0，这或许有些出乎意料。有些特征甚至在 0 1 的范围之外！对此的解释是，MinMaxScaler（以及其他所有缩放器）总是对训练集和测试集应用完全相同的变换。也就是说，transform 方法总是减去训练集的最小值，然后除以训练集的范围，而这两个值可能与测试集的最小值和范围并不相同。这个是显然的，因为 scaler 在 fit 的时候使用的训练数据

1.3.3 对训练数据和测试数据进行相同的缩放

在Figure 1.2中，第一张图是未缩放的二维数据集，其中训练集用圆形表示，测试集用三角形表示。第二张图中是同样的数据，但使用 MinMaxScaler 缩放。这里我们调用 fit 作用在训练集上，然后调用 transform 作用在训练集和测试集上。你可以发现，第二张图中的数据集看起来与第一张图中的完全相同，只是坐标轴刻度发生了变化。现在所有特征都位于 0 到 1 之间。你还可以发现，测试数据（三角形）的特征最大值和最小值并不是 1 和 0。

第三张图展示了如果我们对训练集和测试集分别进行缩放会发生什么。在这种情况下，对训练集和测试集而言，特征的最大值和最小值都是 1 和 0。但现在数据集看起来不一样。测试集相对训练集的移动不一致，因为它们分别做了不同的缩放。我们随意改变了数据的排列。这显然不是我们想要做的事情。

再换一种思考方式，想象你的测试集只有一个点。对于一个点而言，无法将其正确地缩放以满足 MinMaxScaler 的最大值和最小值的要求。但是，测试集的大小不应该对你的处理方式有影响。

快捷方式与高效的替代方法

通常来说，你想要在某个数据集上 fit 一个模型，然后再将其 transform。这是一个非常常见的任务，通常可以用比先调用 fit 再调用 transform 更高效的方法来计算。对于这种使用场景，所有具有 transform 方法的模型也都具有一个 `fit_transform` 方法。

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit(X).transform(X)
X_scaled_d = scaler.fit_transform(X_train)
y_scaled_d = scaler.transform(X_test)
```

虽然 `fit_transform` 不一定对所有模型都更加高效，但在尝试变换训练集时，使用这一方法仍然是很好的做法。

1.3.4 预处理对监督学习的作用

我们回到 cancer 数据集，观察使用 MinMaxScaler 对学习 SVC 的作用。首先，为了对比，我们再次在原始数据上拟合 SVC。

正如我们上面所见，数据缩放的作用非常显著。虽然数据缩放不涉及任何复杂的数学，但良好的做法仍然是使用 scikit-learn 提供的缩放机制，而不是自己重新实现它们，因为即使在这些简单的计算中也容易犯错。

你也可以通过改变使用的类将一种预处理算法轻松替换成另一种，因为所有的预处理类都具有相同的接口，都包含 fit 和 transform 方法

1.4 降维、特征提取与流形学习

利用无监督学习进行数据变换可能有很多种目的。最常见的目的就是可视化、压缩数据，以及寻找信息量更大的数据表示以用于进一步的处理。

为了实现这些目的，最简单也最常用的一种算法就是主成分分析。也将介绍另外两种算法：非负矩阵分解（NMF）和 t-SNE，前者通常用于特征提取，后者通常用于二维散点图的可视化。

1.4.1 主成分分析

主成分分析（principal component analysis，PCA）是一种旋转数据集的方法，旋转后的特征在统计上不相关。在做完这种旋转之后，通常是根据新特征对解释数据的重要性来选择它的一个子集。下面的例子Figure 1.3展示了 PCA 对一个模拟二维数据集的作用。

在Figure 1.3中，第一张图（左上）显示的是原始数据点，用不同颜色加以区分。算法首先找到方差最大的方向，将其标记为“成分 1”（Component 1）。这是数据中包含最多信息的方向（或向量），换句话说，沿着这个方向的特征之间最为相关。然后，算法找到与第一个方向正交（成直角）且包含最多信息的方向。在二维空间中，只有一个成直角的方向，但在更高维的空间中会有（无穷）多的正交方向。虽然这两个成分都画成箭头，但其头尾的位置并不重要。我们也可以将第一个成分画成从中心指向左上，而不是指向右下。利用这一过程找到的方向被称为主成分（principal component），因为它们是数据方差的主要方向。一般来说，主成分的个数与原始特征相同。

第二张图（右上）显示的是同样的数据，但现在将其旋转，使得第一主成分与 x 轴平行且第二主成分与 y 轴平行。在旋转之前，从数据中减去平均值，使得变换后的数据以零为中心。在 PCA 找到的旋转表示中，两个坐标轴是不相关的，也就是说，对于这种数据表示，除了对角线，相关矩阵全部为零。

我们可以通过仅保留一部分主成分来使用 PCA 进行降维。在这个例子中，我们可以仅保留第一个主成分，正如图 3-3 中第三张图所示（左下）。这将数据从二维数据集降为一维数据集。但要注意，我们没有保留原始特征之一，而是找到了最有趣的方向（第一张图中从左上到右下）并保留这一方向，即第一主成分。

最后，我们可以反向旋转并将平均值重新加到数据中。这样会得到图 3-3 最后一张图中的数据。这些数据点位于原始特征空间中，但我们仅保留了第一主成分中包含的信息。这种变换有时用于去除数据中的噪声影响，或者将主成分中保留的那部分信息可视化。

将 PCA 应用于 cancer 数据集并可视化

我们为每个特征创建一个直方图，计算具有某一特征的数据点在特定范围内（叫作bin）的出现频率。这样我们可以了解每个特征在两个类别中的分布情况，也可以猜测哪些特征能够更好地区分良性样本和恶性样本。例如，“smoothness error”特征似乎没有什么信息量，因为两个直方图大部分都重叠在一起，而“worst concave points”特征看起来信息量相当大，因为两个直方图的交集很小。

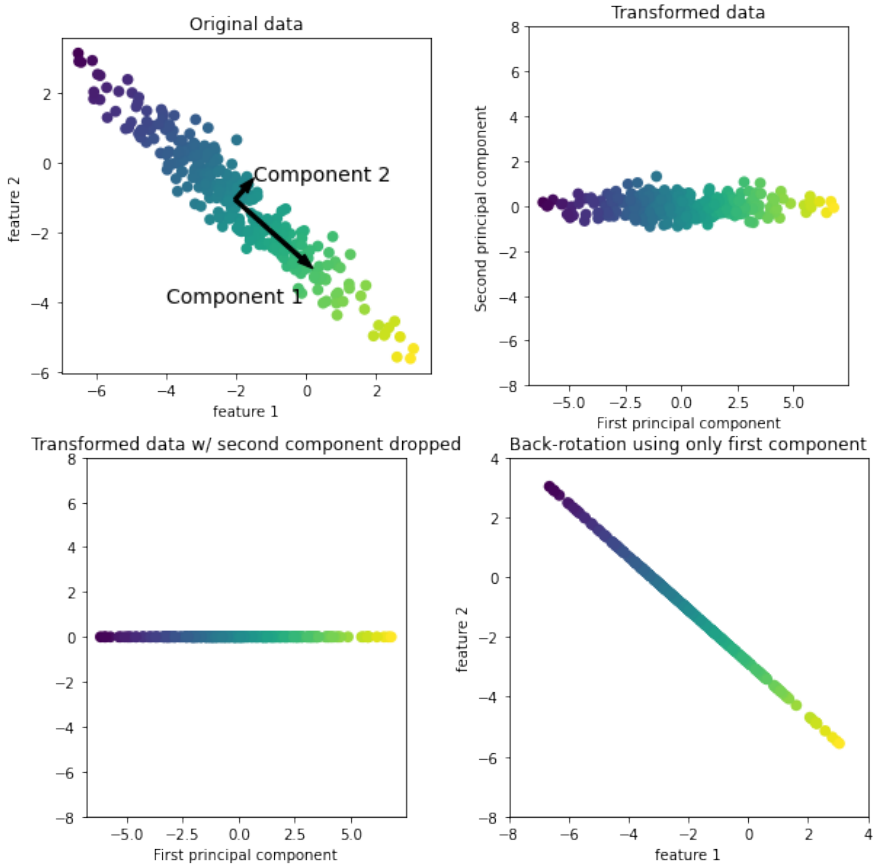

Figure 1.3: Transformation of data with PCA

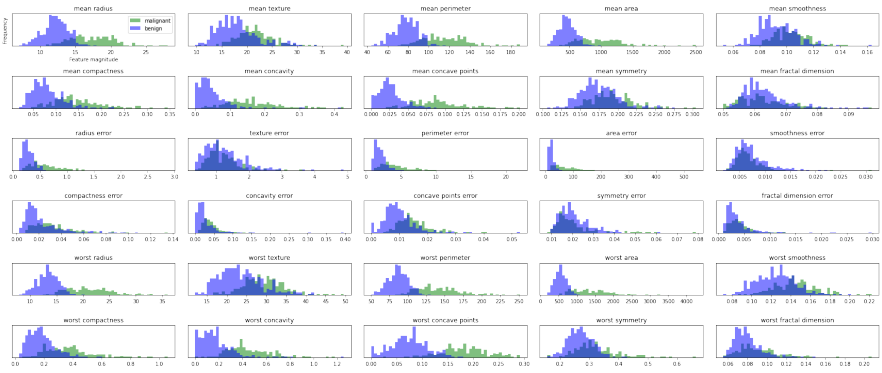

Figure 1.4: Per-class feature histograms on the Breast Cancer dataset

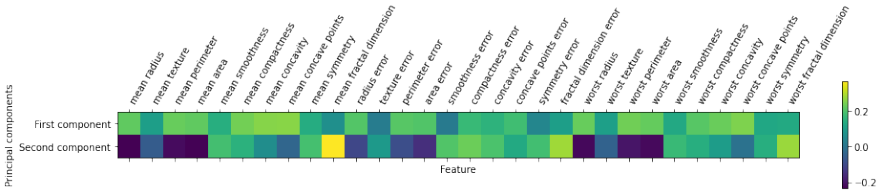

Figure 1.5: Heat map of the first two principal components on the Breast Cancer dataset

但是，这种图无法向我们展示变量之间的相互作用以及这种相互作用与类别之间的关系。利用 PCA，我们可以获取到主要的相互作用，并得到稍为完整的图像。我们可以找到前两个主成分，并在这个新的二维空间中用散点图将数据可视化。

学习并应用 PCA 变换与应用预处理变换一样简单。我们将 PCA 对象实例化，调用 fit 方法找到主成分，然后调用 transform 来旋转并降维。默认情况下，PCA 仅旋转（并移动）数据，但保留所有的主成分。为了降低数据的维度，我们需要在创建 PCA 对象时指定想要保留的主成分个数。

重要的是要注意，PCA 是一种无监督方法，在寻找旋转方向时没有用到任何类别信息。它只是观察数据中的相关性。对于这里所示的散点图，我们绘制了第一主成分与第二主成分的关系，然后利用类别信息对数据点进行着色。你可以看到，在这个二维空间中两个类别被很好地分离。这让我们相信，即使是线性分类器（在这个空间中学习一条直线）也可以在区分这个两个类别时表现得相当不错。

PCA 的一个缺点在于，通常不容易对图中的两个轴做出解释。主成分对应于原始数据中的方向，所以它们是原始特征的组合。但这些组合往往非常复杂。在拟合过程中，主成分被保存在 PCA 对象的 `components_` 属性中，`components_` 中的每一行对应于一个主成分，它们按重要性排序（第一主成分排在首位，以此类推）。列对应于 PCA 的原始特征属性。

在第一个主成分中，所有特征的符号相同（均为正，但前面我们提到过，箭头指向哪个方向无关紧要）。这意味着在所有特征之间存在普遍的相关性。。第二个主成分的符号有正有负，而且两个主成分都包含所有 30 个特征。这种所有特征的混合使得解释Figure 1.5中的坐标轴变得十分困难。

特征提取的特征脸

前面提到过，PCA 的另一个应用是特征提取。特征提取背后的思想是，可以找到一种数据表示，比给定的原始表示更适合于分析。特征提取很有用，它的一个很好的应用实例就是图像。图像由像素组成，通常存储为红绿蓝（RGB）强度。图像中的对象通常由上千个像素组成，它们只有放在一起才有意义。

我们得到的精度为 14.0%。对于包含 62 个类别的分类问题来说，这实际上不算太差（随机猜测的精度约为 1/62=1.5%），但也不算好。

这里就可以用到 PCA。想要度量人脸的相似度，计算原始像素空间中的距离是一种相当糟糕的方法。用像素表示来比较两张图像时，我们比较的是每个像素的灰度值与另一张图像对应位置的像素灰度值。这种表示与人们对人脸图像的解释方式有很大不同，使用这种原始表示很难获取到面部特征。例如，如果使用像素距离，那么将人脸向右移动一个像素将会发生巨大的变化，得到一个完全不同的表示。我们希望，使用沿着主成分方向的距离可以提高精度。这里我们启用 PCA 的白化（whitening）选项，它将主成分缩放到相同的尺度。变换后的结果与使用 StandardScaler 相同。

虽然我们肯定无法理解这些成分的所有内容，但可以猜测一些主成分捕捉到了人脸图像的哪些方面。第一个主成分似乎主要编码的是人脸与背景的对比，第二个主成分编码的是人脸左半部分和右半部分的明暗程度差异，如此等等。虽然这种表示比原始像素值的语义稍强，但它仍与人们感知人脸的方式

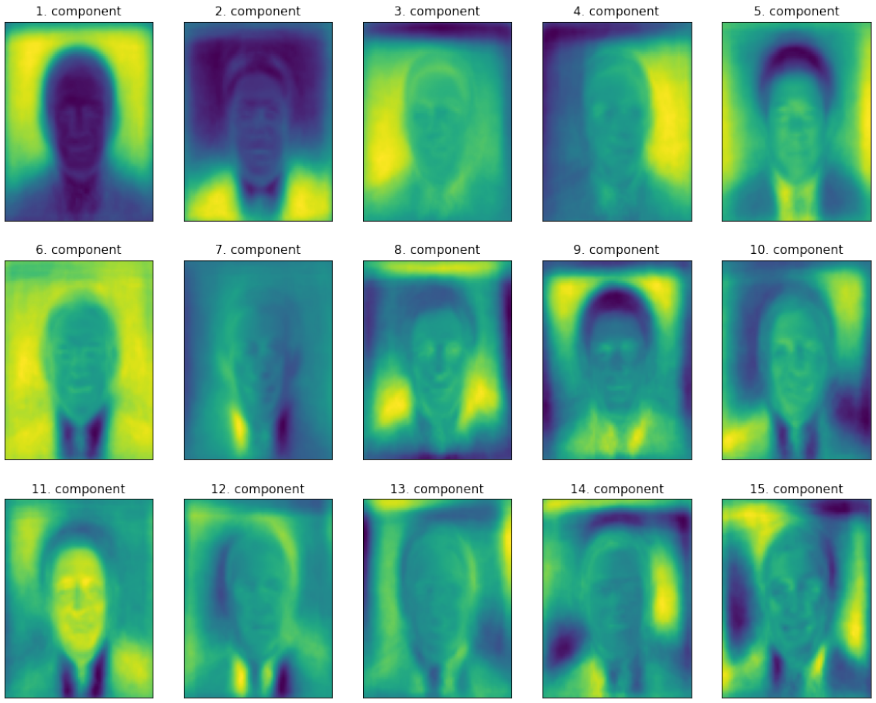

Figure 1.6: Component vectors of the first 15 principal components of the faces dataset

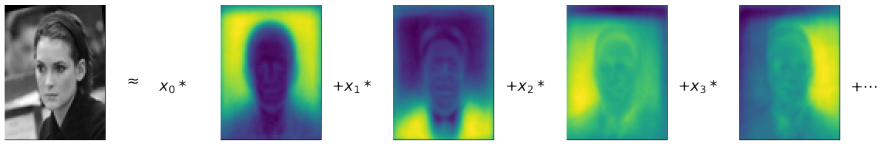

Figure 1.7: Schematic view of PCA as decomposing an image into a weighted sum of components

相去甚远。由于 PCA 模型是基于像素的，因此人脸的相对位置（眼睛、下巴和鼻子的位置）和明暗程度都对两张图像在像素表示中的相似程度有很大影响。但人脸的相对位置和明暗程度可能并不是人们首先感知的内容。在要求人们评价人脸的相似度时，他们更可能会使用年龄、性别、面部表情和发型等属性，而这些属性很难从像素强度中推断出来。重要的是要记住，算法对数据（特别是视觉数据，比如人们非常熟悉的图像）的解释通常与人类的解释方式大不相同。

我们对 PCA 变换的介绍是：先旋转数据，然后删除方差较小的成分。另一种有用的解释是尝试找到一些数字（PCA 旋转后的新特征值），使我们可以将测试点表示为主成分的加权求和（见Figure 1.7）。

这里 x_0、x_1 等是这个数据点的主成分的系数，换句话说，它们是图像在旋转后的空间中的表示。

我们还可以用另一种方法来理解 PCA 模型，就是仅使用一些成分对原始数据进行重建。在Figure 1.3 中，在去掉第二个成分并来到第三张图之后，我们反向旋转并重新加上平均值，这样就在原始空间中获得去掉第二个成分的新数据点，正如最后一张图所示。我们可以对人脸做类似的变换，将数据降维到只包含一些主成分，然后反向旋转回到原始空间。回到原始特征空间可以通过 `inverse_transform` 方法来实现。这里我们分别利用 10 个、50 个、100 个和 500 个成分对一些人脸进行重建并将其可视。

在仅使用前 10 个主成分时，仅捕捉到了图片的基本特点，比如人脸方向和明暗程度。随着使用的主成分越来越多，图像中也保留了越来越多的细节。这对应于Figure 1.7的求和中包含越来越多的项。如

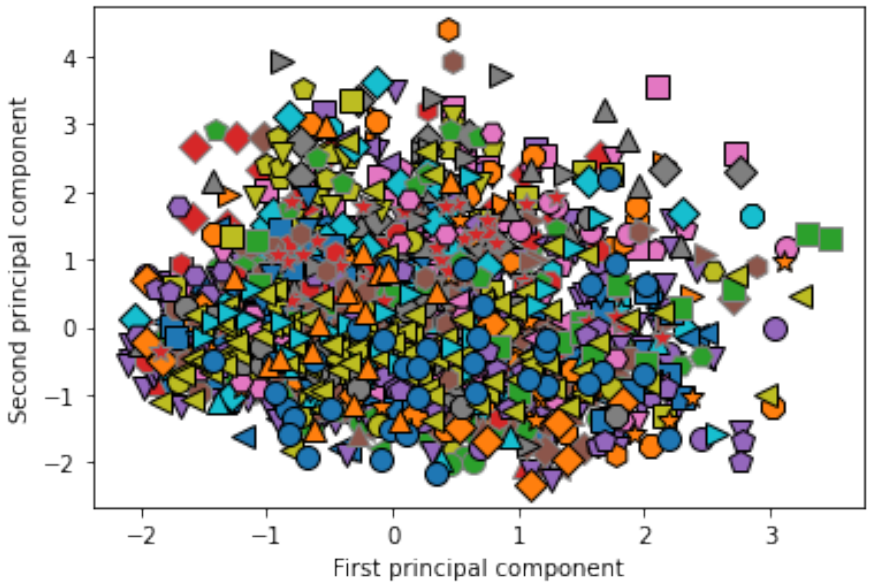

Figure 1.8: Scatter plot of the faces dataset using the first two principal components

果使用的成分个数与像素个数相等，意味着我们在旋转后不会丢弃任何信息，可以完美重建图像。

我们还可以尝试使用 PCA 的前两个主成分，将数据集中的所有人脸在散点图中可视化（Figure 1.8），其类别在图中给出。

1.4.2 非负矩阵分解

非负矩阵分解（non-negative matrix factorization，NMF）是另一种无监督学习算法，其目的在于提取有用的特征。它的工作原理类似于 PCA，也可以用于降维。与 PCA 相同，我们试图将每个数据点写成一些分量的加权求和。但在 PCA 中，我们想要的是正交分量，并且能够解释尽可能多的数据方差；而在 NMF 中，我们希望分量和系数均为非负，也就是说，我们希望分量和系数都大于或等于 0。因此，这种方法只能应用于每个特征都是非负的数据，因为非负分量的非负求和不可能变为负值。

将数据分解成非负加权求和的这个过程，对由多个独立源相加（或叠加）创建而成的数据特别有用，比如多人说话的音轨或包含多种乐器的音乐。在这种情况下，NMF 可以识别出组成合成数据的原始分量。总的来说，与 PCA 相比，NMF 得到的分量更容易解释，因为负的分量和系数可能会导致难以解释的抵消效应（cancellation effect）。举个例子，图3-9 中的特征脸同时包含正数和负数，我们在 PCA 的说明中也提到过，正负号实际上是任意的。

将 NMF 应用于模拟数据

与使用 PCA 不同，我们需要保证数据是正的，NMF 能够对数据进行操作。这说明数据相对于原点 (0,0) 的位置实际上对 NMF 很重要。因此，你可以将提取出来的非负分量看作是从 (0,0) 到数据的方向。

对于两个分量的 NMF（如左图所示），显然所有数据点都可以写成这两个分量的正数组合。如果有足够多的分量能够完美地重建数据（分量个数与特征个数相同），那么算法会选择指向数据极值的方向。

如果我们仅使用一个分量，那么 NMF 会创建一个指向平均值的分量，因为指向这里可以对数据做出最好的解释。你可以看到，与 PCA 不同，减少分量个数不仅会删除一些方向，而且会创建一组完全

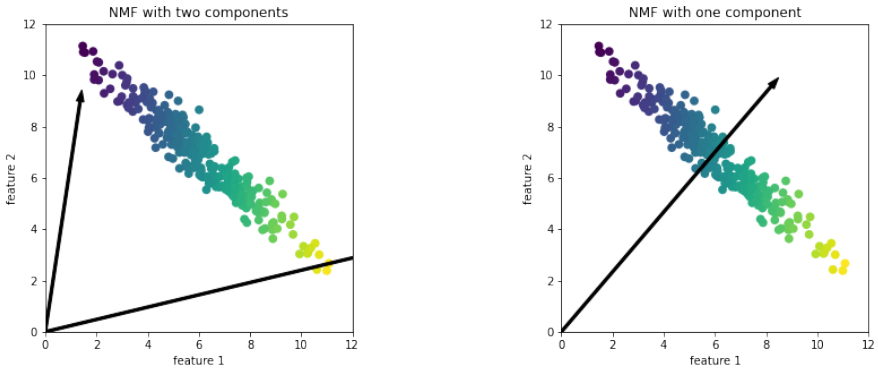

Figure 1.9: Components found by non-negative matrix factorization

不同的分量！ NMF 的分量也没有按任何特定方法排序，所以不存在“第一非负分量”：所有分量的地位平等。

NMF 使用了随机初始化，根据随机种子的不同可能会产生不同的结果。在相对简单的情况下（比如两个分量的模拟数据），所有数据都可以被完美地解释，那么随机性的影响很小（虽然可能会影响分量的顺序或尺度）。在更加复杂的情况下，影响可能会很大。

还有许多其他算法可用于将每个数据点分解为一系列固定分量的加权求和，正如 PCA 和NMF 所做的那样。讨论所有这些算法已超出了本书的范围，而且描述对分量和系数的约束通常要涉及概率论。如果你对这种类型的模式提取感兴趣，我们推荐你学习 scikit-learn 用户指南中关于独立成分分析（ICA）、因子分析（FA）和稀疏编码（字典学习）等的内容，所有这些内容都可以在关于分解方法的页面中找到。

1.4.3 用t-SNE进行流形学习

虽然 PCA 通常是用于变换数据的首选方法，使你能够用散点图将其可视化，但这一方法的性质（先旋转然后减少方向）限制了其有效性。有一类用于可视化的算法叫作**流形学习算法**（manifold learning algorithm），它允许进行更复杂的映射，通常也可以给出更好的可视化。其中特别有用的一个就是 t-SNE 算法。

流形学习算法主要用于可视化，因此很少用来生成两个以上的新特征。其中一些算法（包括 t-SNE）计算训练数据的一种新表示，但不允许变换新数据。这意味着这些算法不能用于测试集：更确切地说，它们只能变换用于训练的数据。流形学习对探索性数据分析是很有用的，但如果最终目标是监督学习的话，则很少使用。t-SNE 背后的思想是找到数据的一个二维表示，尽可能地保持数据点之间的距离。t-SNE 首先给出每个数据点的随机二维表示，然后尝试让在原始特征空间中距离较近的点更加靠近，原始特征空间中相距较远的点更加远离。t-SNE 重点关注距离较近的点，而不是保持距离较远的点之间的距离。换句话说，它试图保存那些表示哪些点比较靠近的信息。

t-SNE 的结果非常棒。所有类别都被明确分开。数字 1 和 9 被分成几块，但大多数类别都形成一个密集的组。要记住，这种方法并不知道类别标签：它完全是无监督的。但它能够找到数据的一种二维表示，仅根据原始空间中数据点之间的靠近程度就能够将各个类别明确分开。

t-SNE 算法有一些调节参数，虽然默认参数的效果通常就很好。你可以尝试修改 `perplexity` 和 `early_exaggeration`，但作用一般很小。

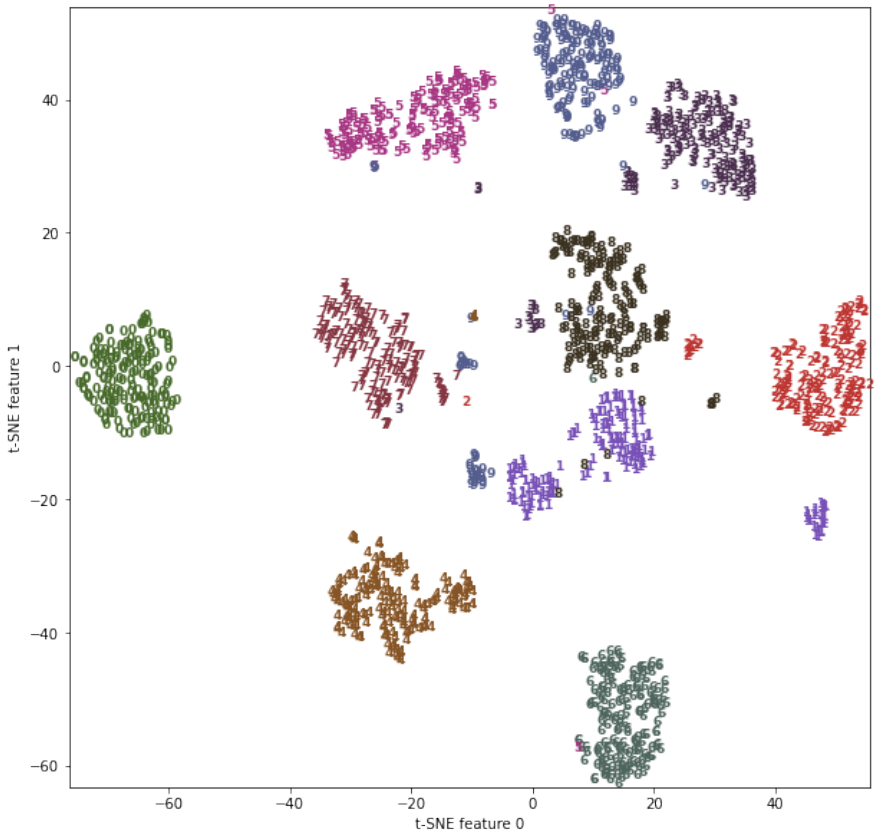

Figure 1.10: Scatter plot of the digits dataset using two components found by t-SNE

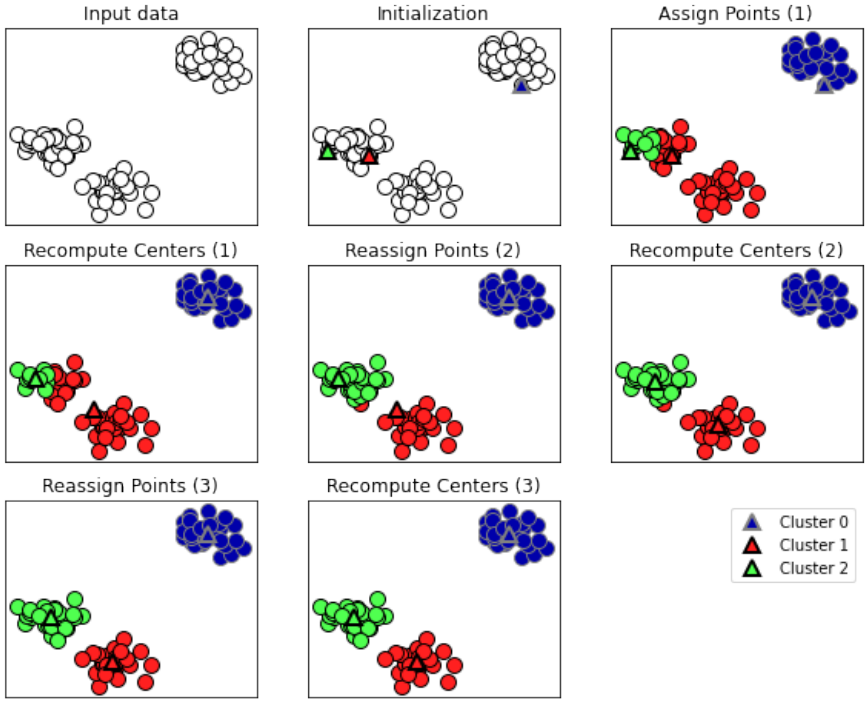

Figure 1.11: Input data and three steps of the k-means algorithm

1.5 聚类

聚类（clustering）是将数据集划分成组的任务，这些组叫作簇（cluster）。其目标是划分数据，使得一个簇内的数据点非常相似且不同簇内的数据点非常不同。与分类算法类似，聚类算法为每个数据点分配（或预测）一个数字，表示这个点属于哪个簇。

1.5.1 k均值聚类

k 均值聚类是最简单也最常用的聚类算法之一。它试图找到代表数据特定区域的簇中心（cluster center）。算法交替执行以下两个步骤：将每个数据点分配给最近的簇中心，然后将每个簇中心设置为所分配的所有数据点的平均值。如果簇的分配不再发生变化，那么算法结束。 scikit-learn 实现 k 均值相当简单。下面我们将其应用于上图中的模拟数据。我们将KMeans 类实例化，并设置我们要寻找的簇个数[1] 。然后对数据调用 fit 方法。

算法运行期间，为 X 中的每个训练数据点分配一个簇标签。你可以在 `kmeans.labels_` 属性中找到这些标签。

你也可以用 predict 方法为新数据点分配簇标签。预测时会将最近的簇中心分配给每个新数据点，但现有模型不会改变。聚类算法与分类算法有些相似，每个元素都有一个标签。但并不存在真实的标签，因此标签本身并没有先验意义。

对于刚刚在二维演示数据集上运行的聚类算法，这意味着我们不应该为其中一组的标签是 0、另一组的标签是 1 这一事实赋予任何意义。再次运行该算法可能会得到不同的簇编号，原因在于初始化的随机性质。

[1] 如果不指定，它的默认值是 8。使用这个值并没有什么特别的原因。

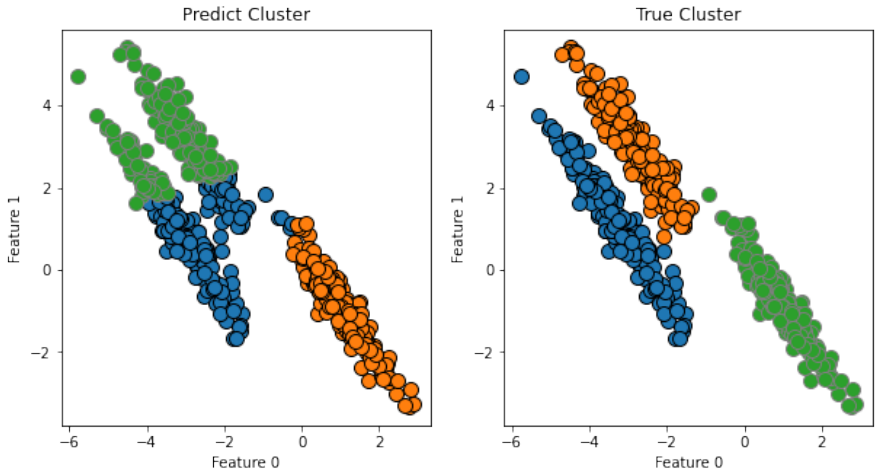

Figure 1.12: k-means fails to identify nonspherical clusters

k均值的失败案例

即使你知道给定数据集中簇的“正确”个数，k 均值可能也不是总能找到它们。每个簇仅由其中心定义，这意味着每个簇都是凸形（convex）。因此，k 均值只能找到相对简单的形状。k 均值还假设所有簇在某种程度上具有相同的“直径”，它总是将簇之间的边界刚好画在簇中心的中间位置。有时这会导致令人惊讶的结果。

k 均值还假设所有方向对每个簇都同等重要，见Figure 1.12。

如果簇的形状更加复杂，比如 `two_moons` 数据，那么 k 均值的表现也很差（见Figure 1.13）

这里我们希望聚类算法能够发现两个半月形。但利用 k 均值算法是不可能做到这一点的。

矢量量化，或者将k均值看作分解

虽然 k 均值是一种聚类算法，但在 k 均值和分解方法（比如之前讨论过的 PCA 和 NMF）之间存在一些有趣的相似之处。你可能还记得，PCA 试图找到数据中方差最大的方向，而NMF 试图找到累加的分量，这通常对应于数据的“极值”或“部分”。两种方法都试图将数据点表示为一些分量之和。与之相反，k 均值则尝试利用簇中心来表示每个数据点。你可以将其看作仅用一个分量来表示每个数据点，该分量由簇中心给出。这种观点将 k 均值看作是一种分解方法，其中每个点用单一分量来表示，这种观点被称为矢量量化（vector quantization）。

利用 k 均值做矢量量化的一个有趣之处在于，可以用比输入维度更多的簇来对数据进行编码。让我们回到 `two_moons` 数据。利用 PCA 或 NMF，我们对这个数据无能为力，因为它只有两个维度。使用 PCA 或 NMF 将其降到一维，将会完全破坏数据的结构。但通过使用更多的簇中心，我们可以用 k 均值找到一种更具表现力的表示（见Figure 1.14）。

我们使用了 10 个簇中心，也就是说，现在每个点都被分配了 0 到 9 之间的一个数字。我们可以将其看作 10 个分量表示的数据（我们有 10 个新特征），只有表示该点对应的簇中心的那个特征不为 0，其他特征均为 0。利用这个 10 维表示，现在可以用线性模型来划分两个半月形，而利用原始的两个特征是不可能做到这一点的。将到每个簇中心的距离作为特征，还可以得到一种表现力更强的数据表示。可以利用 kmeans 的 transform 方法来完成这一点。

k 均值是非常流行的聚类算法，因为它不仅相对容易理解和实现，而且运行速度也相对较快。k 均值可以轻松扩展到大型数据集，scikit-learn 甚至在 MiniBatchKMeans 类中包含了一种更具可扩展性的变

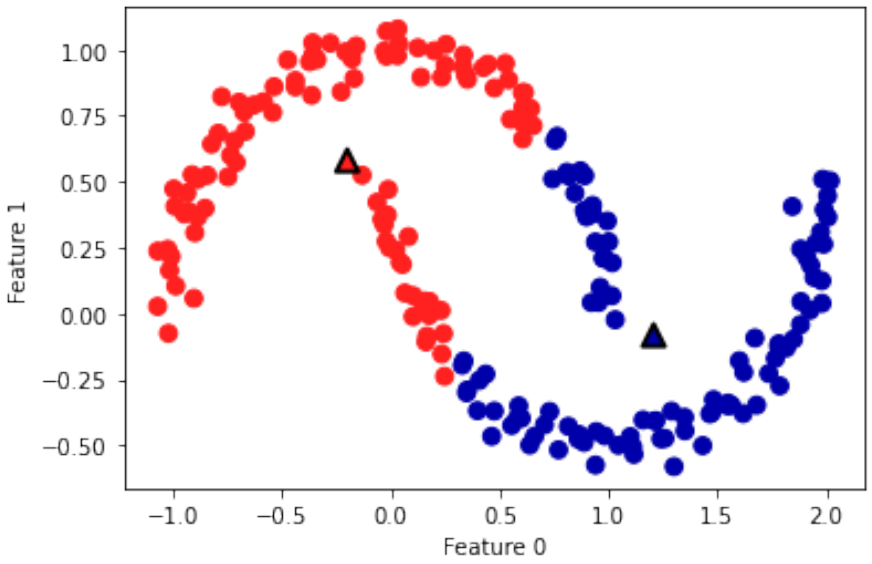

Figure 1.13: k-means fails to identify clusters with complex shapes

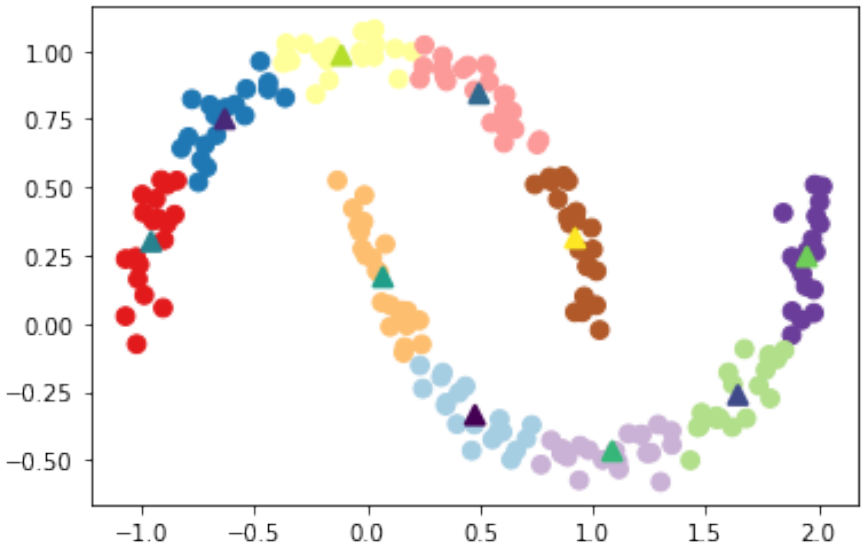

Figure 1.14: Using many k-means clusters to cover the variation in a complex dataset

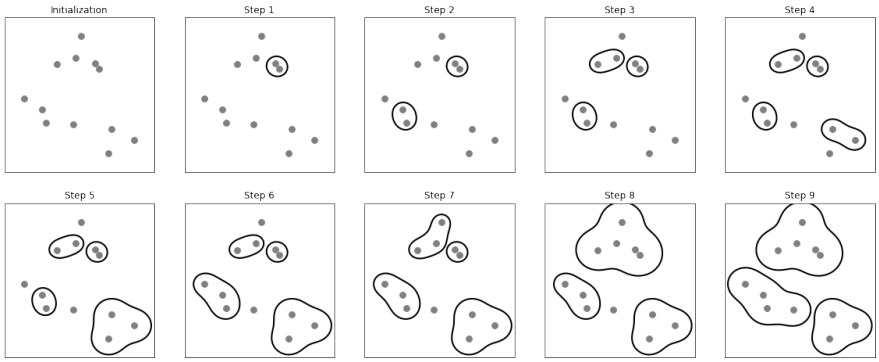

Figure 1.15: Agglomerative clustering iteratively joins the two closest clusters

体，可以处理非常大的数据集。

k 均值的缺点之一在于，它依赖于随机初始化，也就是说，算法的输出依赖于随机种子。默认情况下，scikit-learn 用 10 种不同的随机初始化将算法运行 10 次，并返回最佳结果[2]。k 均值还有一个缺点，就是对簇形状的假设的约束性较强，而且还要求指定所要寻找的簇的个数（在现实世界的应用中可能并不知道这个数字）。

1.5.2 凝聚聚类

凝聚聚类（agglomerative clustering）指的是许多基于相同原则构建的聚类算法，这一原则是：算法首先声明每个点是自己的簇，然后合并两个最相似的簇，直到满足某种停止准则为止。scikit-learn 中实现的停止准则是簇的个数，因此相似的簇被合并，直到仅剩下指定个数的簇。还有一些链接（linkage）准则，规定如何度量“最相似的簇”。这种度量总是定义在两个现有的簇之间。

scikit-learn 中实现了以下三种选项。

1. ward 默认选项。ward 挑选两个簇来合并，使得所有簇中的方差增加最小。这通常会得到大小差不多相等的簇。
2. average 链接将簇中所有点之间平均距离最小的两个簇合并。
3. complete 链接（也称为最大链接）将簇中点之间最大距离最小的两个簇合并。

ward 适用于大多数数据集，如果簇中的成员个数非常不同（比如其中一个比其他所有都大得多），那么 average 或 complete 可能效果更好。

由于算法的工作原理，凝聚算法不能对新数据点做出预测。因此 AgglomerativeClustering 没有 predict 方法。为了构造模型并得到训练集上簇的成员关系，可以改用 `fit_predict` 方法。结果如 Figure 1.16 所示。

虽然凝聚聚类的 scikit-learn 实现需要你指定希望算法找到的簇的个数，但凝聚聚类方法为选择正确的个数提供了一些帮助，我们将在下面讨论。

层次聚类与树状图

凝聚聚类生成了所谓的层次聚类（hierarchical clustering）。聚类过程迭代进行，每个点都从一个单点簇变为属于最终的某个簇。每个中间步骤都提供了数据的一种聚类（簇的个数也不相同）。有时候，同时查看所有可能的聚类是有帮助的。下一个例子（Figure 1.17）叠加显示了Figure 1.15 中所有可能的聚类，有助于深入了解每个簇如何分解为较小的簇。

[2]在这种情况下，“最佳”的意思是簇的方差之和最小。

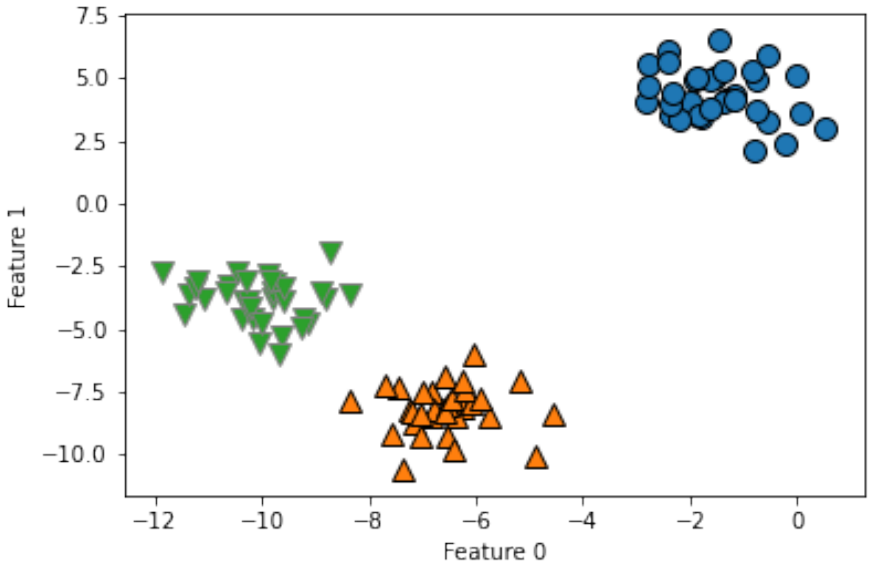

Figure 1.16: Cluster assignment using agglomerative clustering with three clusters

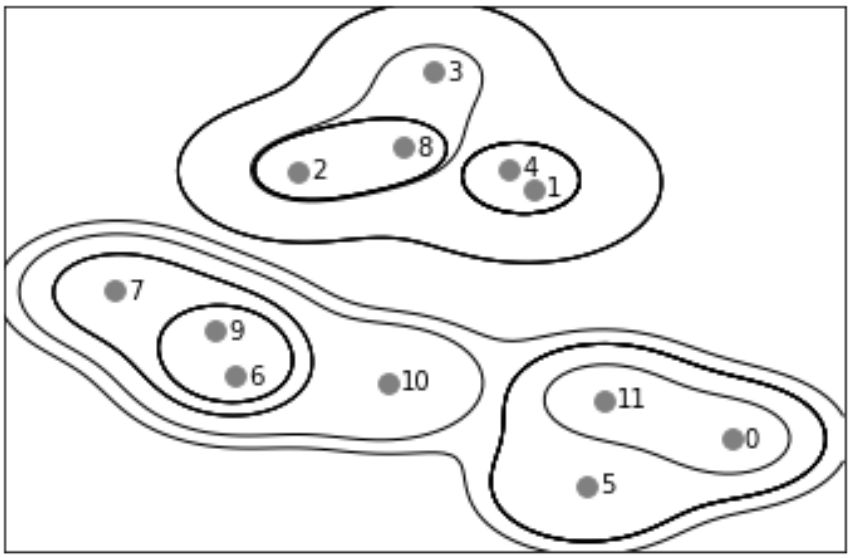

Figure 1.17: Hierarchical cluster assignment generated with agglomerative clustering

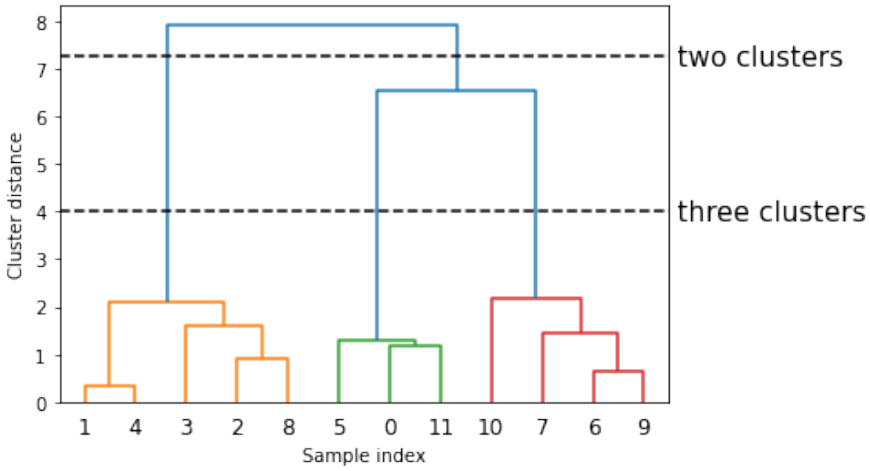

Figure 1.18: Dendrogram of the clustering

虽然这种可视化为层次聚类提供了非常详细的视图，但它依赖于数据的二维性质，因此不能用于具有两个以上特征的数据集。但还有另一个将层次聚类可视化的工具，叫作树状图（dendrogram），它可以处理多维数据集。

不幸的是，目前 scikit-learn 没有绘制树状图的功能。但你可以利用 SciPy 轻松生成树状图。SciPy 的聚类算法接口与 scikit-learn 的聚类算法稍有不同。SciPy 提供了一个函数，接受数据数组 X 并计算出一个链接数组（linkage array），它对层次聚类的相似度进行编码。然后我们可以将这个链接数组提供给 scipy 的 dendrogram 函数来绘制树状图（图 Figure 1.18）。

状图在底部显示数据点（编号从 0 到 11）。然后以这些点（表示单点簇）作为叶节点绘制一棵树，每合并两个簇就添加一个新的父节点。

从下往上看，数据点 1 和 4 首先被合并。接下来，点 6 和 9 被合并为一个簇，以此类推。在顶层有两个分支，一个由点 11、0、5、10、7、6 和 9 组成，另一个由点 1、4、3、2 和 8 组成。这对应于图中左侧两个最大的簇。

树状图的 y 轴不仅说明凝聚算法中两个簇何时合并，每个分支的长度还表示被合并的簇之间的距离。在这张树状图中，最长的分支是用标记为“three clusters”（三个簇）的虚线表示的三条线。它们是最长的分支，这表示从三个簇到两个簇的过程中合并了一些距离非常远的点。我们在图像上方再次看到这一点，将剩下的两个簇合并为一个簇也需要跨越相对较大的距离。

不幸的是，凝聚聚类仍然无法分离像 `two_moons` 数据集这样复杂的形状。但我们要学习的下一个算法 DBSCAN 可以解决这个问题。

1.5.3 DBSCAN

1.5.4 聚类算法的对比与评估

1.5.5 聚类方法小结

1.6 小结与展望