第3章

无监督学习与数据预处理

本章内容

- ◇ 无监督学习的类型
- ◇ 无监督学习的挑战
- ◇ 数据预处理
- ◇ 降维算法及其应用
- ◇ 聚类算法及其应用

3.1 无监督学习

现实生活中常常会有这样的问题：缺乏足够的先验知识，因此难以人工标注类别或进行人工类别标注的成本太高。很自然地，我们希望计算机能代替我们完成这些工作，或至少提供一些帮助。根据类别未知（没有被标记）的训练样本解决机器学习中的各种问题，称为无监督学习。常见的应用背景包括：

（1）从庞大的样本集合中选出一些具有代表性的加以标注用于分类器的训练。

（2）先将所有样本自动分为不同的类别，再由人对这些类别进行标注。

（3）在无类别信息的情况下，寻找好的特征。

这一章将讨论机器学习算法中的无监督学习算法。无监督学习的学习算法只有输入数据，需要从这些数据中提取知识。

微课视频

3.1.1 无监督学习的类型

本章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，从而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提取主题。这里的任务是找到每个文档中讨论的未知主题，并学习每个文档中出现了哪些主题。这可以用于追踪社交媒体上的话题讨论，比如流行歌手等话题。

与之相反，聚类算法（Clustering Algorithm）将数据划分成不同的组，每组包含相似的项目。思考向社交媒体网站上传照片的例子。为了方便整理照片，网站可能想要将同一个人的照片分在一组。但网站并不知道每张照片是谁，也不知道照片集中出现了多少个人。明智的做法是提取所有的人脸，并将看起来相似的人脸分在一组。

3.1.2　无监督学习的挑战


微课视频

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，所以算法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”。例如，假设聚类算法已经将所有的侧脸照片和所有的正面照片进行了分组。这肯定是人脸照片集合的一种可能的划分方法，但这并不是我们想要的方法。然而，我们没有办法“告诉”算法我们要的是什么，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，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。

在开始学习“真正的”无监督算法之前，先简要讨论几种简单又常用的预处理方法。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。

3.2　数据预处理

据统计，数据预处理相关的工作时间占据了整个机器学习项目的70%以上。数据的质量直接决定了模型的预测和泛化能力的好坏。它涉及很多因素，包括准确性、完整性、一致性、时效性、可信性和解释性。而真实数据可能包含了大量的缺失值，可能包含大量的噪声，也可能因为人工录入错误导致存在异常点，这些都非常不利于算法模型的训练。数据清洗的结果是对各种脏数据进行对应方式的处理，得到标准的、干净的、连续的数据，供数据统计、数据挖掘等。

原始数据很少以能满足学习算法最佳性能所需要的理想形式出现。因此数据的预处理是任何机器学习应用中最关键的步骤之一。以鸢尾花数据集为例，可以把原始数据看

成一系列的花朵图像，要从中提取有意义的特征。有意义的特征可能是颜色、色调、强度、高度、长度、宽度。许多机器学习算法也要求所选择特征的测量结果具有相同的单位，以获得最佳性能。通常可以通过把特征数据变换为[0,1]的取值范围或者均值为0和方差为1的标准正态分布来实现。

某些选定的特征可能高度相关，因此在某种程度上是多余的。在这种情况下，降维技术对于将特征压缩到低维子空间非常有价值。降低特征空间维数的优点是减少存储空间、提高算法运行的速度。在某些情况下，如果数据集包含大量无用的特征或噪声(即数据集具有较低的信噪比)，那么降维也可以提高模型预测的性能。

为了确定机器学习算法不仅能在训练集上表现良好，对新数据也有很好的适应性，我们希望将数据集随机分成单独的训练集和测试集。用训练集来训练和优化机器学习模型，同时把测试集保留到最后，用以评估最终的模型。

数据预处理的主要步骤分为数据清洗、数据变换、数据集成、数据规约。下面将从这四个方面详细介绍具体的方法。在一个项目中，如果这几方面的数据处理做得都很不错，将对之后的建模具有极大的帮助，能快速达到一个不错的结果。


微课视频

3.2.1 数据清洗

数据清洗(Data Cleaning)的主要思想是通过填补缺失值，光滑噪声数据，平滑或删除离群点，并解决数据的不一致问题来“清洗”数据。如果用户认为数据是“脏乱”的，他们不太会相信基于这些数据的挖掘结果，即输出的结果是不可靠的。

1. 缺失值的处理

由于现实世界获取信息和数据的过程中，会存在各类的原因导致数据丢失和空缺。针对这些缺失值的处理方法，主要是基于变量的分布特性和变量的重要性(信息量和预测能力)采用不同的方法。主要分为以下几种。

(1) **删除变量**：若变量的缺失率较高(大于80%)、覆盖率较低且重要性较低，可以直接将变量删除。

(2) **定值填充**：工程中常用−9999来替代缺失值。

(3) **统计量填充**：若缺失率较低(小于95%)且重要性也较低，则根据数据分布的情况进行填充。若数据符合均匀分布，则用该变量的均值填补缺失；若数据存在倾斜分布的情况，则采用中位数进行填补。

(4) **插值法填充**：包括随机插值法、多重差补法、热平台插补法、拉格朗日插值法、牛顿插值法等。

(5) **模型填充**：使用回归、贝叶斯、随机森林、决策树等模型对缺失数据进行预测。

(6) **哑变量填充**：若变量是离散型，且不同值较少，可转换成哑变量。例如性别SEX变量，存在male,female,NA三个不同的值，可将该列转换成IS_SEX_MALE，IS_SEX_FEMALE，IS_SEX_NA。若某个变量存在十几个不同的值，可根据每个值的频数，将频数较小的值归为一类other，降低维度。此做法可最大化保留变量的信息。

总之，常用的做法是：先用pandas.isnull.sum()检测出变量的缺失比例，考虑删除

或者填充。若需要填充的变量是连续型，一般采用均值法和随机插值进行填充；若变量是离散型，通常采用中位数或哑变量进行填充。

注意：若对变量进行分箱离散化，一般会将缺失值单独作为一个箱子（离散变量的一个值）。

2. 离群点处理

存在异常值是数据分布的常态，处于特定分布区域或范围之外的数据通常被定义为异常或噪声。异常分为两种：①“伪异常”是由于特定的业务运营动作产生的，是正常反应业务的状态，而不是数据本身的异常；②“真异常”不是由于特定的业务运营动作产生的，而是数据本身分布异常，即离群点。主要有以下检测离群点的方法。

（1）简单统计分析：根据箱线图、各分位点判断是否存在异常，例如 pandas 的 describe()函数可以快速发现异常值。

（2）3σ 原则：若数据存在正态分布，偏离均值的 3σ 之外，通常定义范围在 $P(|x-u|>3\sigma)\leqslant 0.003$ 内的点为离群点。

（3）基于绝对离差中位数（MAD）：这是一种稳健对抗离群数据的距离值方法，采用计算各观测值与平均值的距离总和的方法，放大离群值的影响。

（4）基于距离：通过定义对象之间的临近性度量，根据距离判断异常对象是否远离其他对象，缺点是计算复杂度较高，不适用于大数据集和存在不同密度区域的数据集。

（5）基于密度：离群点的局部密度显著低于大部分近邻点，适用于非均匀的数据集。

（6）基于聚类：利用聚类算法，丢弃远离其他簇的小簇。

总之，在数据处理阶段将离群点作为影响数据质量的异常点考虑，而不是作为通常所说的异常检测目标点，因而一般采用较为简单直观的方法，结合箱线图和 MAD 的统计方法判断变量的离群点。

具体的处理手段如下。

（1）根据异常点的数量和影响，考虑是否将该条记录删除，这样信息损失多。

（2）若对数据做了 log-scale 对数变换后消除了异常值，则此方法生效，且不损失信息。

（3）使用平均值或中位数替代异常点，简单高效，信息的损失较少。

（4）在训练树模型时，树模型对离群点的鲁棒性较高，无信息损失，不影响模型训练效果。

3. 噪声处理

噪声是变量的随机误差和方差，是观测点和真实点之间的误差，即 obs$=x\varepsilon$。通常的处理办法：对数据进行分箱操作，等频或等宽分箱，然后用每个箱的平均数、中位数或者边界值（对不同数据分布，处理方法不同）代替箱中所有的数，起到平滑数据的作用。另外一种做法是建立该变量和预测变量的回归模型，根据回归系数和预测变量，反解出自变量的近似值。


微课视频

3.2.2 数据变换

数据变换包括对数据进行规范化、离散化、稀疏化处理，达到适用于挖掘的目的。

1. 规范化处理

数据中不同特征的量纲可能不一致，数值间的差别可能很大，不进行处理可能会影响到数据分析的结果，因此，需要对数据按照一定比例进行缩放，使之落在一个特定的区域，便于进行综合分析。特别是基于距离的挖掘方法：聚类、KNN、SVM 一定要进行规范化处理，如式(3-1)。

$$x_{\text{norm}}^{(i)}=\frac{x^{(i)}-x_{\min}}{x_{\max}-x_{\min}} \tag{3-1}$$

其中，$x^{(i)}$ 为某个特定样本，$x_{\min}$ 为特征列的最小值，$x_{\max}$ 为特征列的最大值。

通过最大、最小比例的调整实现归一化是一种常用的技术，这对有界区间值的问题很有用。标准化对于许多机器学习算法来说更为实用，特别是梯度下降等优化算法。因为许多线性模型(如第 2 章中的逻辑回归和支持向量机)把权重值初始化为 0 或接近 0 的随机值。使用标准化，可以把特征列的中心设在均值为 0 且标准差为 1 的位置，这样特征列呈正态分布，可以使学习权重更容易。此外，标准化保持了关于离群值的有用信息，使算法对离群值不敏感，这与最小、最大比例的调整刚好相反，它把数据调整到有限的值域。标准化的过程为：

$$x_{\text{std}}^{(i)}=\frac{x^{(i)}-\mu_x}{\sigma_x} \tag{3-2}$$

其中，μ_x 是某个特定特征列的样本均值，σ_x 为对应的标准差。而在时间序列数据中，对于数据量级相差较大的变量，通常进行 log 函数的变换(注：由于 Python 中，默认 log 函数底数为 e，如无特殊说明，本书 log 也默认底数为 e)：

$$x_{\text{new}}=\log(x) \tag{3-3}$$

2. 离散化处理

数据离散化是指将连续的数据进行分段，使其变为一段段离散化的区间。分段的原则有基于等距离、等频率或优化的方法。数据离散化的原因主要有以下几点：

① 模型需要：比如决策树、朴素贝叶斯等算法，都是基于离散型的数据展开的。如果要使用该类算法，必须将离散型的数据进行分段。有效的离散化能减小算法的时间和空间开销，提高系统对样本的分类聚类能力和抗噪声能力。

② 离散化的特征相对于连续型特征更容易理解。

③ 可以有效地克服数据中隐藏的缺陷，使模型结果更加稳定。

(1) **等频法**：使得每个箱中的样本数量相等，例如总样本 $n=100$，分成 $k=5$ 个箱，则分箱原则是保证落入每个箱的样本量为 20。

(2) **等宽法**：使得属性的箱宽度相等，例如年龄变量(0～100)，可分成 [0,20)、[20,40)、[40,60)、[60,80)、[80,100]五个等宽的箱。

（3）**聚类法**：根据聚类出来的簇，每个簇中的数据为一个箱，簇的数量模型给定。

3. 稀疏化处理

针对离散型且标称变量，无法进行有序的 LabelEncoder 时，通常考虑将变量进行 0/1 哑变量的稀疏化处理，例如动物类型变量中含有猫、狗、猪、羊四个不同值，将该变量转换成 is_猪、is_猫、is_狗、is_羊四个哑变量。若变量的不同值较多，则根据频数，将出现次数较少的值统一归为一类 rare。稀疏化处理既有利于模型快速收敛，又能提升模型的抗噪能力。

3.2.3　数据集成


微课视频

数据分析任务大多涉及数据集成。数据集成将多个数据源中的数据结合、存放在一个一致的数据存储，如数据仓库。这些源可能包括多个数据库、数据提供方或一般文件。

（1）实体识别问题：例如，数据分析者或计算机如何才能确信一个数据库中的 customer_id 和另一个数据库中的 cust_number 指的是同一实体？通常，数据库和数据仓库有元数据——关于数据的数据。这种元数据可以帮助避免模式集成中的错误。

（2）冗余问题：如果一个属性能由另一个表“导出”，如年薪，那么这个属性是冗余的。属性或维度命名的不一致也可能导致数据集中的冗余问题。用相关性检测冗余：数值型变量可计算相关系数矩阵，标称型变量可计算卡方检验。

（3）数据值的冲突和处理：不同数据源在统一合并时，保持规范化，去重。

3.2.4　数据规约


微课视频

数据规约技术可以用来得到数据集的规约表示，它小得多，但仍接近地保持原数据的完整性。这样在规约后的数据集上挖掘将更有效，并产生相同（或几乎相同）的分析结果。一般有如下策略。

1. 维度规约

用于数据分析的数据可能包含数以百计的属性，其中大部分属性与挖掘任务不相关，是冗余的。维度规约通过删除不相关的属性来减少数据量，并保证信息的损失最小。

属性子集选择：目标是找出最小属性集，使得数据类的概率分布尽可能地接近使用所有属性的原分布。在压缩的属性集上挖掘还有其他的优点：它减少了出现在发现模式上的属性的数目，使得模式更易于理解。

（1）逐步向前选择：该过程由空属性集开始，选择原属性集中最好的属性，并将它添加到该集合中。其后的每次迭代将原属性集剩下的属性中的最好的属性添加到该集合中。

（2）逐步向后删除：该过程由整个属性集开始。每步删除尚在属性集中的最坏属性。

（3）向前选择和向后删除的结合：向前选择和向后删除方法可以结合在一起，每步

选择一个最好的属性,并在剩余属性中删除一个最坏的属性。

Python Scikit-learn 中的递归特征消除(Recursive Feature Elimination,RFE)算法,就是利用这样的思想进行特征子集筛选的,一般考虑建立 SVM 或回归模型。

单变量重要性:分析单变量和目标变量的相关性,删除预测能力较低的变量。这种方法不同于属性子集选择,通常从统计学和信息的角度去分析。

(1) Pearson(皮尔逊)相关系数和卡方检验,分析目标变量和单变量的相关性。

(2) 回归系数:训练线性回归或逻辑回归,提取每个变量的表决系数,进行重要性排序。

(3) 树模型的 Gini 指数:训练决策树模型,提取每个变量的重要度(即 Gini 指数)进行排序。

(4) Lasso 正则化:训练回归模型时,加入 L1 正则化参数,将特征向量稀疏化。

(5) Ⅳ指标:风控模型中,通常求解每个变量的Ⅳ值来定义变量的重要度,一般将阈值设定在 0.02 以上。

以上提到的方法,没有讲解具体的理论知识和实现方法,需要自己去熟悉掌握。通常的做法是根据业务需求来定采用的方法;如果基于业务的用户或商品特征,需要较多的解释性,考虑采用统计上的一些方法,如变量的分布曲线、直方图等,再计算相关性指标,最后考虑一些模型方法;如果建模需要,则通常采用模型方法去筛选特征;如果用一些更为复杂的 GBDT、DNN 等模型,建议不做特征选择,而做特征交叉。

2. 维度变换

维度变换是将现有数据降低到更小的维度,尽量保证数据信息的完整性。下面介绍几种常用的有损失的维度变换方法,它们将大幅提高实践中建模的效率。

(1) 主成分分析(PCA)和因子分析(FA):PCA 通过空间映射的方式,将当前维度映射到更低的维度,使得每个变量在新空间的方差最大;FA 则是找到当前特征向量的公因子(维度更小),用公因子的线性组合来描述当前的特征向量。

(2) 奇异值分解(SVD):SVD 的降维导致可解释性较低,且计算量比 PCA 大,一般用在稀疏矩阵上降维,例如图片压缩、推荐系统。

(3) 聚类:将某一类具有相似性的特征聚到单个变量,从而大大降低维度。

(4) 线性组合:将多个变量做线性回归,根据每个变量的表决系数,赋予变量权重,可将该类变量根据权重组合成一个变量。

(5) 流行学习:流行学习中的一些复杂的非线性方法,可参考 sklearn 的 LLE Example。

3.3 降维

下面开始学习第一种类型的无监督学习问题——降维。有几个不同的原因使你可能想要做降维:一是数据压缩,数据压缩允许我们压缩数据,因而可以让算法使用更少的计算机内存或磁盘空间,提高算法学习速度;二是数据可视化,数据可视化会帮助机器学习工程师们更好地处理数据、设计算法。

3.3.1　数据压缩


微课视频

首先，需要讨论降维是什么。图 3.1 是利用收集得到的数据集中的两个特征绘制的。

假设已知两个特征：x_1 为长度，用厘米表示；x_2 是同一物体的长度，用英寸表示。

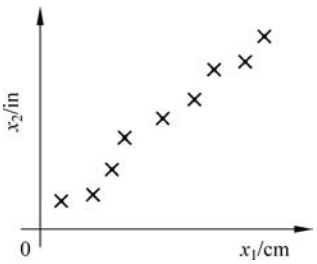

图 3.1　二维压缩到一维

所以，这两个特征含有高度冗余的信息，也许不是两个分开的特征 x_1 和 x_2，而是两个基本长度度量的单位，我们想要做的是减少数据到一维，只保留一个测量长度的特征。

将数据从二维降至一维：假使要采用两种不同的仪器来测量一些物体的尺寸，其中一个仪器测量结果的单位是英寸，另一个仪器测量的结果是厘米，希望将测量的结果作为机器学习的特征。现在的问题是，两种仪器对同一个物体测量的结果不完全相等（由于误差、精度等），而将两者都作为特征有些重复，因而，希望将这个二维的数据降至一维。

从这个例子可以看到，在工业上做特征搜集时经常产生这种高度冗余的数据集。如果有几百个或成千上万的特征，而需要的特征可能不会很多。有时可能有几个不同的工程团队，也许第一个工程团队给了二百个特征，第二个工程团队给了另外三百个特征，第三个工程团队给了五百个特征，合计一千多个特征都在一起，这样实际上会让跟踪那些需要的特征变得非常困难。

将数据从三维降至二维：如图 3.2 所示，将一个三维的特征向量降至一个二维的特征向量。过程是与上面类似的，将三维向量投射到一个二维的平面上，强迫使得所有的数据都在同一个平面上，降为二维的特征向量。

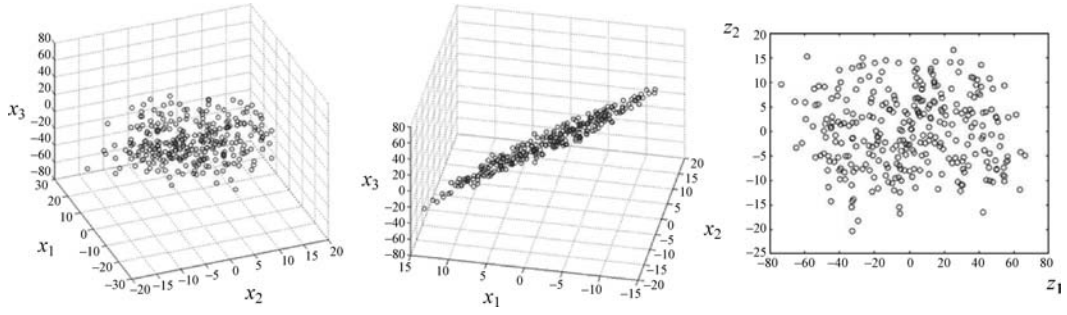

图 3.2　三维压缩到二维

这样的处理过程可以被用于把任何维度的数据降到任何想要的维度，例如将 1000 维的特征降至 100 维。

正如上面所看到的，这也将使得机器学习的算法运行更快。

3.3.2 数据可视化

在许多机器学习问题中，如果能将数据可视化，便能寻找到一个更好的解决方案。降维可以帮助实现数据可视化，如图 3.3 所示。

国家	GDP(万亿美元)	人均GDP(千美元)	人类发展指数	平均寿命	贫困指数(基尼百分比)	家庭平均收入(千美元)	...
加拿大	1.577	39.17	0.908	80.7	32.6	67.793	...
中国	5.878	7.54	0.687	73	46.9	10.22	...
印度	1.632	3.41	0.547	64.7	36.8	0.735	...
俄罗斯	1.48	19.84	0.755	65.5	39.9	0.72	...
新加坡	0.223	56.69	0.866	80	42.5	67.1	...
美国	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

图 3.3 各国人口数据

假设有关于许多不同国家的数据，每一个特征向量都有 50 个特征(如 GDP、人均 GDP、平均寿命等)。如果要将这个 50 维的数据可视化是不可能的。使用降维的方法将其降至二维，便可以将其可视化了，如图 3.4 所示。

国家	z_1	z_2
加拿大	1.6	1.2
中国	1.7	0.3
印度	1.6	0.2
俄罗斯	1.4	0.5
新加坡	0.5	1.7
美国	2	1.5
...	...	...

图 3.4 压缩到二维

这样做的问题在于，降维的算法只负责减少维数，新产生的特征 z_1 和 z_2 的意义就必须由我们自己去发现了。可视化后的效果如图 3.5 所示。

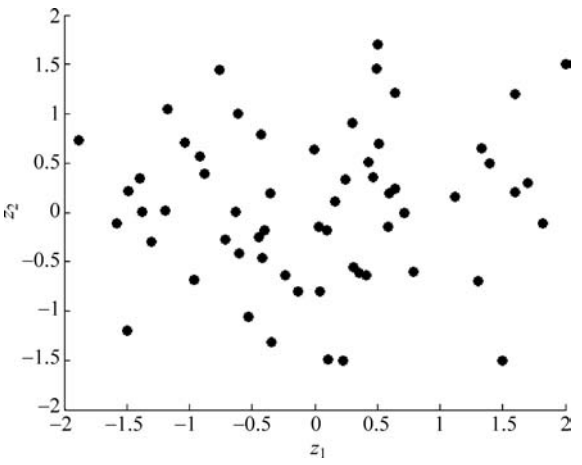

图 3.5 降维后可视化效果

3.3.3　降维的主要方法


微课视频

在深入了解具体的降维算法之前，先来看看降维的两种主要方法：投影和流形学习。

1. 投影

在大多数现实世界的问题里，训练实例在所有维度上并不是均匀分布的。许多特征几乎是不变的，也有许多特征是高度相关联的（如前面讨论的 MNIST 数据集）。因此，高维空间的所有训练实例实际上（或近似于）受一个低得多的低维子空间影响。这听起来很抽象，下面来看一个例子。在图 3.6 中，你可以看到一个由圆圈表示的 3D 数据集。

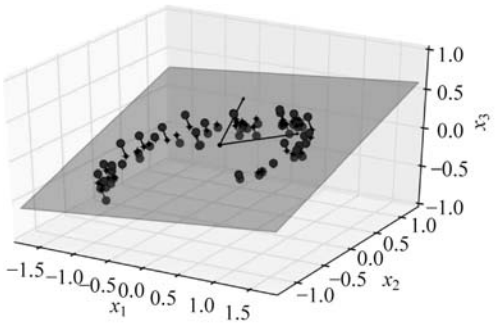

图 3.6　3D 数据集和 2D 子空间（见彩插）

注意看，所有的训练实例都紧挨着一个平面：这就是高维（3D）空间的低维（2D）子空间。现在，如果将每个训练实例垂直投影到这个子空间（如图 3.6 中实例到平面之间的短线所示），将得到如图 3.7 所示的新 2D 数据集，它已经将数据集维度从三维降到了二维。注意，图中的轴对应的是新特征 z_1 和 z_2（平面上投影的坐标）。

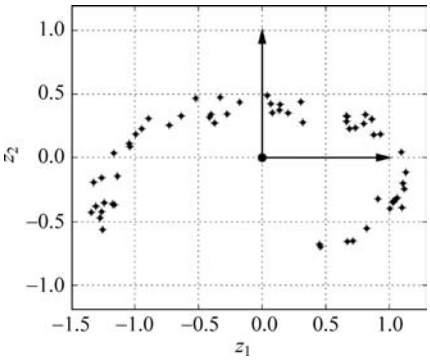

图 3.7　投影后产生的新 2D 数据集

不过投影并不总是降维的最佳方法。在许多情况下，子空间可能会弯曲或转动，比如图 3.8 所示的著名的瑞士卷数据集。

简单地进行平面投影（例如放弃 x_3）会直接将瑞士卷的不同层压扁在一起，如图 3.9(a)所示。但是你真正想要的是将整个瑞士卷展开铺平以后的 2D 数据集，如图 3.9(b)所示。

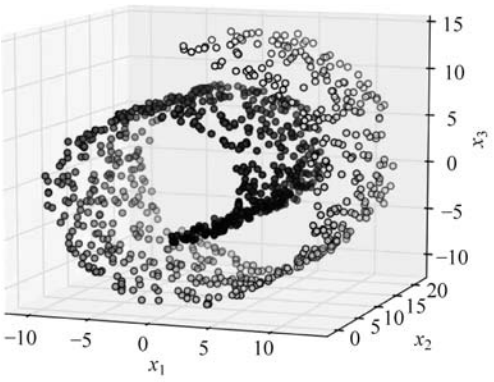

图 3.8　瑞士卷数据集(见彩插)

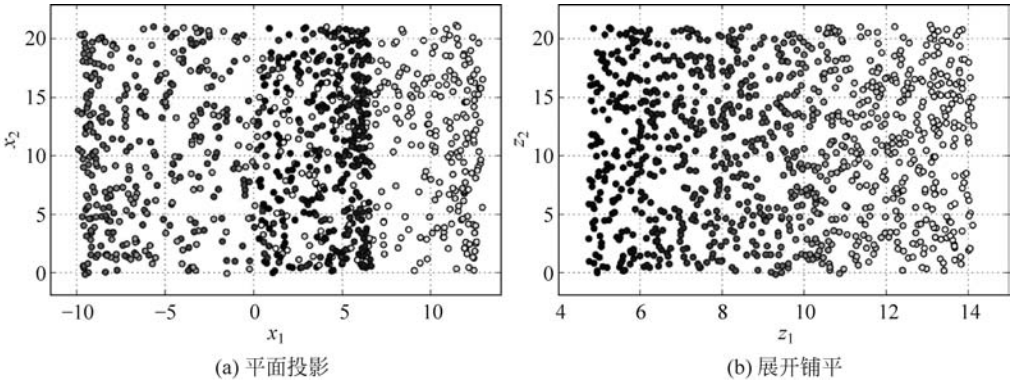

图 3.9　瑞士卷数据集处理后

2. 流形学习

瑞士卷就是一个二维流形的例子。简单地说,二维流形就是一个能够在更高维空间里面弯曲和扭转的二维形状。更概括地说,d 维流形就是 n(其中,$d<n$)维空间的一部分,局部类似于一个 d 维超平面。在瑞士卷的例子中,$d=2$,$n=3$:它局部类似于一个二维平面,但是在第三个维度上卷起。

许多降维算法是通过对训练实例进行流形建模来实现的,这被称为流形学习。它所依赖的流形假设(也称为流形假说)认为大多数现实世界的高维度数据集都可以用一个低维度的流形来重新表示。这个假设通常是凭经验观察得到的。

再次说到 MNIST 数据集:所有手写的数字图像都有一些相似之处。它们由相连的线条组成,边界都是白色的,或多或少是居中的,等等。如果随机生成图像,只有少到不能再少的一部分可能看起来像手写数字。也就是说,创建一个数字图像拥有的自由度要远远低于创建任意图像的自由度。而这些限制正倾向于将数据集挤压成更低维度的流形。

流形假设通常还伴随着一个隐含的假设:如果能用低维空间的流形表示,手头的任务(例如分类或者回归)将变得更简单。如图 3.10(a)和图 3.10(b)所示,瑞士卷被分为两

类：3D 空间中(图 3.10(a))决策边界将会相当复杂，但是在展开的 2D 流形空间(图 3.10(b))，决策边界是一条简单的直线。

但是，这个假设并不总是成立。如图 3.10(c)和图 3.10(d)所示，决策边界在 $x_1=5$ 处，在原始的 3D 空间中，这个边界看起来非常简单(一个垂直的平面)，但是在展开的流形中，决策边界看起来反而更为复杂(四条独立线段的集合)。

简而言之，在训练模型之前降低训练集的维度，肯定可以加快训练速度，但这并不总是会导致更好或更简单的解决方案，它取决于数据集。

希望现在你对于维度的诅咒有了很好地理解，也知道降维算法是怎么对付它的，特别是当流形假设成立的时候，应该怎么处理。本章剩余部分将逐一介绍几个最流行的算法。

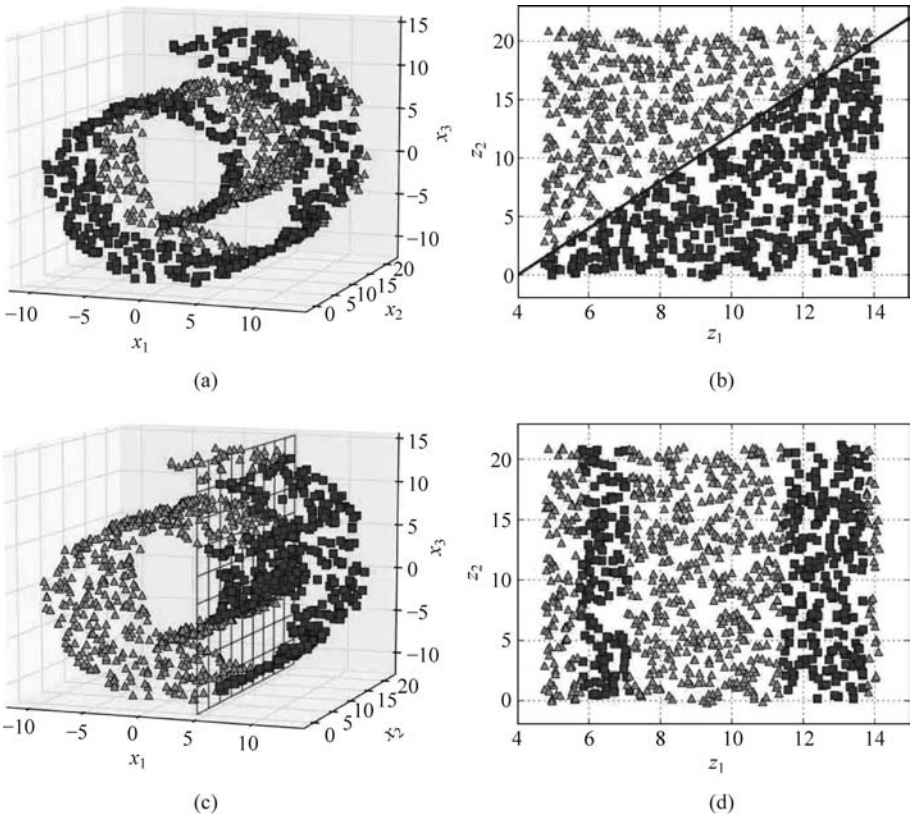

图 3.10　决策边界不总是维度越低越简单(见彩插)

3.3.4　PCA


微课视频

主成分分析(Principal Component Analysis，PCA)：通俗理解，就是找出一个最主要的特征，然后进行分析。

主成分分析是一种分析、简化数据集的技术。主成分分析经常用于减少数据集的维度，同时保持数据集中对方差贡献最大的特征。这是通过保留低阶主成分、忽略高阶主成分做到的。这样低阶成分往往能够保留住数据的最重要方面。但是，这也不是一定的，要

视具体应用而定。由于主成分分析依赖所给数据，所以数据的准确性对分析结果影响很大。

主成分分析由卡尔·皮尔逊于1901年发明，用于分析数据及建立数理模型。其方法主要是通过对协方差矩阵进行特征分解，以得出数据的主成分(即特征向量)与它们的权值(即特征值)。PCA是最简单的以特征量分析多元统计分布的方法。其结果可以理解为对原数据中的方差做出解释——哪一个方向上的数据值对方差的影响最大。换而言之，PCA提供了一种降低数据维度的有效办法；如果分析者在原数据中除掉最小的特征值所对应的成分，那么所得的低维度数据必定是最优的(这样降低维度必定是失去信息最少的方法)。主成分分析在分析复杂数据时尤为有用，比如人脸识别。

如果一个多元数据集能够在一个高维数据空间坐标系中被显示出来，那么PCA就能够提供一幅低维度的图像，这幅图像即为在信息最多的点上原对象的一个“投影”。这样就可以利用少量的主成分使得数据的维度降低。

PCA的主要步骤包括：

- 去除平均值。
- 计算协方差矩阵。
- 计算协方差矩阵的特征值和特征向量。
- 将特征值排序。
- 保留前 N 个最大的特征值对应的特征向量。
- 将数据转换到上面得到的 N 个特征向量构建的新空间中(实现特征压缩)。


微课视频

3.3.5 利用PCA实现半导体制造数据降维

半导体是在一些极为先进的工厂中制造出来的。设备的生命周期有限，并且花费巨大。虽然通过早期测试和频繁测试已发现部分有瑕疵的产品，但仍有一些存在瑕疵的产品通过了测试。如果通过机器学习技术用于发现瑕疵产品，那么它就会为制造商节省大量的资金。具体来讲，Secom公司的半导体数据集拥有590个特征。看看能否对这些特征进行降维处理。对于数据的缺失值的问题，将缺失值NaN(Not a Number缩写)，全部用平均值来替代(如果用0来处理就太差了)。

代码清单3-1：半导体制造数据降维

```
#收集数据:提供文本文件,文件名:secom.data
[3030.93  2564  2187.7333  1411.1265  1.3602  100  97.6133  0.1242  1.5005  0.0162
 -0.0034  0.9455  202.4396  0  7.9558  414.871  10.0433  0.968  192.3963  12.519  1.4026
 -5419  2916.5  -4043.75  751  0.8955  1.773  3.049  64.2333  2.0222  0.1632  3.5191
83.3971  9.5126  50.617  64.2588  49.383  66.3141  86.9555  117.5132  61.29  4.515  70
352.7173  10.1841  130.3691  723.3092  1.3072  141.2282  1  624.3145  218.3174  0  4.592]
'''将 NaN 替换成平均值函数'''
def replaceNanWithMean():
    datMat = loadDataSet('./secom.data', ' ')
    numFeat = shape(datMat)[1]
    for i in range(numFeat):
```

```
            # 对 value 不为 NaN 的求均值
            # .A 返回矩阵基于的数组
            meanVal = mean(datMat[nonzero(~isnan(datMat[:, i].A))[0], i])
            # 将 value 为 NaN 的值赋值为均值
            datMat[nonzero(isnan(datMat[:, i].A))[0],i] = meanVal
    return datMat
```

对数据进行数据预处理之后，再运行程序，看看中间结果如何，分析数据代码如下。

```
'''分析数据'''
def analyse_data(dataMat):
    meanVals = mean(dataMat, axis=0)
    meanRemoved = dataMat-meanVals
    covMat = cov(meanRemoved, rowvar=0)
    eigvals, eigVects = linalg.eig(mat(covMat))
    eigValInd = argsort(eigvals)
    topNfeat = 20
    eigValInd = eigValInd[:-(topNfeat+1):-1]
    cov_all_score = float(sum(eigvals))
    sum_cov_score = 0
    for i in range(0, len(eigValInd)):
        line_cov_score = float(eigvals[eigValInd[i]])
        sum_cov_score += line_cov_score
    print('主成分:%s, 方差占比:%s%%, 累积方差占比:%s%%' % (format(i+1, '2.0f'), format(line_cov_score/cov_all_score*100, '4.2f'), format(sum_cov_score/cov_all_score*100, '4.1f')))
#去均值化的特征值结果显示
[ 5.34151979e+07  2.17466719e+07  8.24837662e+06  2.07388086e+06
  1.31540439e+06  4.67693557e+05  2.90863555e+05  2.83668601e+05
  2.37155830e+05  2.08513836e+05  1.96098849e+05  1.86856549e+05
  1.52422354e+05  1.13215032e+05  1.08493848e+05  1.02849533e+05
  1.00166164e+05  8.33473762e+04  8.15850591e+04  7.76560524e+04
  …
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
]
```

```
33  #数据分析结果主成分: 1, 方差占比:59.25%, 累积方差占比:59.3%
34  主成分: 2, 方差占比:24.12%, 累积方差占比:83.4%
35  主成分: 3, 方差占比:9.15%, 累积方差占比:92.5%
36  主成分: 4, 方差占比:2.30%, 累积方差占比:94.8%
37  主成分: 5, 方差占比:1.46%, 累积方差占比:96.3%
38  主成分: 6, 方差占比:0.52%, 累积方差占比:96.8%
39  主成分: 7, 方差占比:0.32%, 累积方差占比:97.1%
40  主成分: 8, 方差占比:0.31%, 累积方差占比:97.4%
41  主成分: 9, 方差占比:0.26%, 累积方差占比:97.7%
42  主成分:10, 方差占比:0.23%, 累积方差占比:97.9%
43  主成分:11, 方差占比:0.22%, 累积方差占比:98.2%
44  主成分:12, 方差占比:0.21%, 累积方差占比:98.4%
45  主成分:13, 方差占比:0.17%, 累积方差占比:98.5%
46  主成分:14, 方差占比:0.13%, 累积方差占比:98.7%
47  主成分:15, 方差占比:0.12%, 累积方差占比:98.8%
48  主成分:16, 方差占比:0.11%, 累积方差占比:98.9%
49  主成分:17, 方差占比:0.11%, 累积方差占比:99.0%
50  主成分:18, 方差占比:0.09%, 累积方差占比:99.1%
51  主成分:19, 方差占比:0.09%, 累积方差占比:99.2%
52  主成分:20, 方差占比:0.09%, 累积方差占比:99.3%
```

发现其中有超过20%的特征值都是0。这就意味着这些特征都是其他特征的副本，也就是说，它们可以通过其他特征来表示，而本身并没有提供额外的信息。最前面值的数量级大于10^5，实际上它之后的值都变得非常小。这就相当于告诉我们只有部分重要特征，重要特征的数目也很快就会下降。最后，注意到有一些绝对值较小的负值，它们主要源自数值误差，应该四舍五入成0。

根据实验结果绘制半导体数据中前7个主成分所占的方差百分比如表3.1所示。

表3.1 前7个主成分所占的方差百分比

主成分	方差百分比/%	累积方差百分比/%	主成分	方差百分比/%	累积方差百分比/%
1	59.25	59.3	5	1.46	96.3
2	24.12	83.4	6	0.52	96.8
3	9.15	92.5	7	0.32	97.1
4	2.30	94.8			

3.4 聚类

聚类是一种机器学习技术，它涉及数据点的分组。给定一组数据点，可以使用聚类算法将每个数据点划分至一个特定的组。理论上，同一组中的数据点应该具有相似的属性和/或特征，而不同组中的数据点应该具有高度不同的属性和/或特征。聚类是一种无监

督学习的方法，是许多领域中常用的统计数据分析技术。

在数据科学中，可以使用聚类分析从数据中获得一些有价值的见解。本节将研究5种流行的聚类算法以及它们的优缺点。

聚类算法可以解决一些非常重要的商业应用，比如市场分割：在数据库中存储了许多客户的信息，希望将他们分至不同的客户群，这样可以对不同类型的客户分别销售产品或者分别提供更适合的服务。再比如社交网络分析：事实上有许多研究人员正在研究这样一些内容，他们关注一群人，关注社交网络，或者是其他的一些信息，比如经常跟哪些人联系，而这些人又经常给哪些人发邮件，由此找到关系密切的人群。因此，这可能需要另一个聚类算法，用它发现社交网络中关系密切的朋友。使用聚类算法可以更好地组织计算机集群，或者更好地管理数据中心。因为如果知道数据中心中，哪些计算机经常协同工作，那么可以重新分配资源、重新布局网络，由此优化数据中心、优化数据通信。

最后，实际上还可以研究如何利用聚类算法了解星系的形成，然后用这个知识，了解一些天文学上的细节问题。

3.4.1　*K*-Means 聚类


微课视频

K-Means 聚类(K 均值聚类)算法可能是大家最熟悉的聚类算法。它出现在很多介绍性的数据科学和机器学习课程中。在代码中很容易理解和实现，如图 3.11 所示。

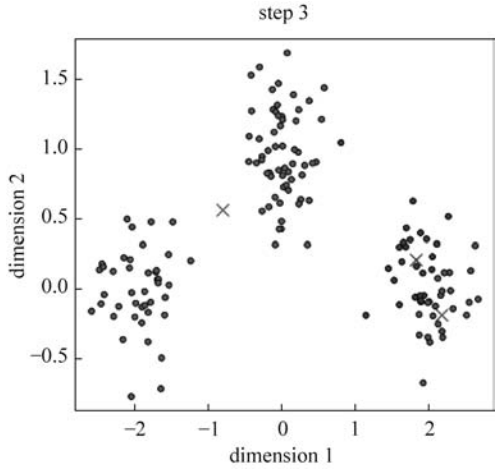

图 3.11　*K*-Means 聚类(见彩插)

K-Means 聚类的一般步骤如下。

(1) 首先，选择一些类/组并随机地初始化它们各自的中心点。若想知道要使用的类的数量，最好快速地查看一下数据，并尝试识别任何不同的分组。中心点是与每个数据点向量相同长度的向量，在图 3.11 中是“×”。

(2) 每个数据点通过计算点和每个组中心之间的距离进行分类，然后将这个点分类为最接近它的组。

(3) 基于这些分类点，通过取组中所有向量的均值来重新计算组中心。

(4) 对每组迭代重复这些步骤。你还可以选择随机初始化组中心几次,然后选择那些看起来对它提供了最好结果的来运行,如图 3.12 所示。

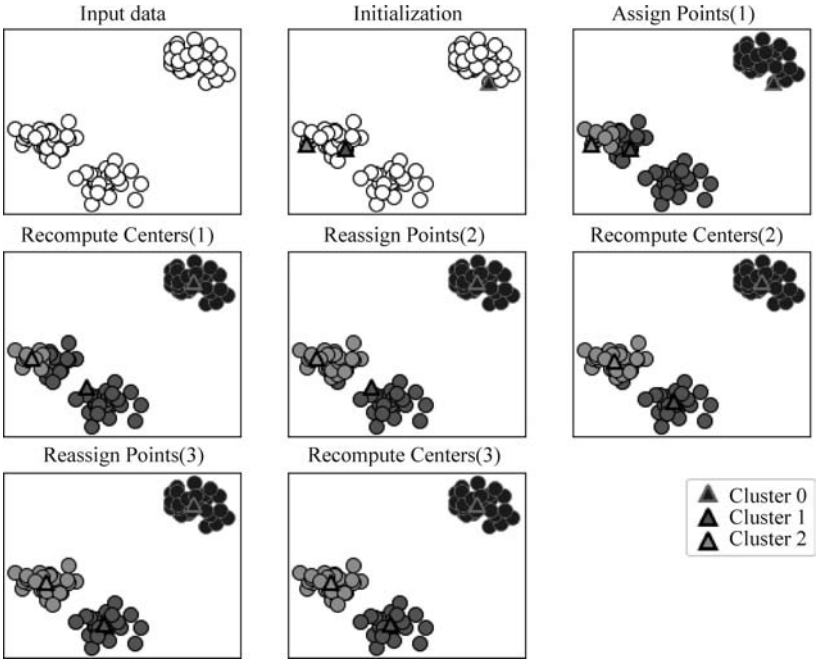

图 3.12 K 均值聚类算法步骤(见彩插)

图 3.12 中,簇中心用三角形表示,而数据点用圆形表示。颜色表示簇成员。指定要寻找三个簇,所以通过声明三个随机数据点为簇中心来将算法初始化(见图 3.12 中 Initialization,初始化)。然后开始迭代算法。首先,每个数据点被分配给距离最近的簇中心(见图 3.12 中 Assign Points(1),分配数据点(1))。接下来,将簇中心修改为所分配点的平均值(见图 3.12 中 Recompute Centers(1),重新计算中心(1))。然后将这一过程再重复两次。在第三次迭代之后,为簇中心分配的数据点保持不变,因此算法结束。

给定新的数据点,K 均值聚类会将其分配给最近的簇中心。下一个例子(图 3.13)展示了图 3.12 学到的簇中心的边界。

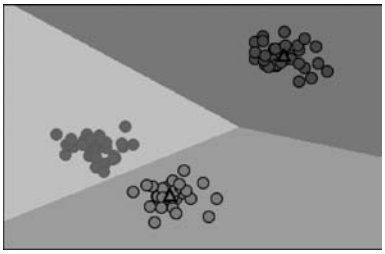

图 3.13 K 均值聚类算法找到的簇中心和簇边界(见彩插)

K-Means 聚类算法的优势在于它的速度非常快,因为所做的只是计算点和群中心之间的距离,它有一个线性复杂度 $O(n)$。

另一方面，K-Means 聚类也有几个缺点。首先，你必须选择有多少组/类。这并不是不重要的事，理想情况下，希望它能帮助解决这些问题，因为它的关键在于从数据中获得一些启示。K-Means 聚类也从随机选择的聚类中心开始，因此在不同的算法运行中可能产生不同的聚类结果。因此，结果可能是不可重复的，并且缺乏一致性。而其他聚类方法的聚类结果更加一致。

K-Medians 是另一种与 K-Means 有关的聚类算法，除了使用均值的中间值来重新计算组中心点以外，这种方法对离群值的敏感度较低（因为使用中值），但对于较大的数据集来说，它要慢得多，因为在计算中值向量时，每次迭代都需要进行排序。

3.4.2　均值偏移聚类


微课视频

均值偏移（Mean Shift）聚类算法是一种基于滑动窗口（Sliding-Window）的算法，它试图找到密集的数据点。而且，它还是一种基于中心的算法，它的目标是定位每组群/类的中心点，通过更新中心点的候选点来实现滑动窗口中的点的平均值。这些候选窗口在后期处理阶段被过滤，以消除几乎重复的部分，形成最后一组中心点及其对应的组，如图 3.14 所示。

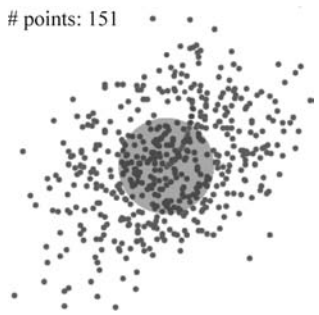

图 3.14　基于滑动窗口的均值偏移聚类（见彩插）

均值偏移聚类的一般步骤如下。

（1）考虑使用二维空间中的一组点（就像上面的例子）。从一个以点 C（随机选择）为中心的圆形滑窗开始，内核半径为 r。均值偏移是一种爬山算法（Hill Climbing Algorithm），它需要在每个步骤中反复地将这个内核移动到一个更高的密度区域，直到收敛。

（2）在每次迭代中，滑动窗口会移向密度较高的区域，将中心点移动到窗口内的点的平均值（因此得名）。滑动窗口中的密度与它内部的点的数量成比例。自然地，通过移向窗口中点的平均值，它将逐渐向更高的点密度方向移动。

（3）继续根据均值移动滑动窗口，直到没有方向移动可以容纳内核中的更多点。如图 3.14 所示，一直移动这个圆，直到密度（也就是窗口中的点数）不再增加。

（4）步骤（1）到（3）的过程是用许多滑动窗口完成的，直到所有的点都位于一个窗口内。当多个滑动窗口重叠的时候，包含最多点的窗口会被保留。然后，数据点根据它们所在的滑动窗口聚类。

与 K-Means 聚类相比，均值偏移聚类不需要选择聚类的数量，因为它会自动地发现这一点。这是一个巨大的优势。聚类中心收敛于最大密度点的事实也是非常可取的，因为它可以被非常直观地理解并适合于一种自然数据驱动。缺点是选择窗口大小/半径 r 是非常关键的，所以不能疏忽。


微课视频

3.4.3 DBSCAN

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)是一个比较有代表性的基于密度的聚类算法，类似于均值偏移聚类算法，但它有几个显著的优点。下面展示了利用 DBSCAN 实现笑脸数据集的聚类，如图 3.15 所示。

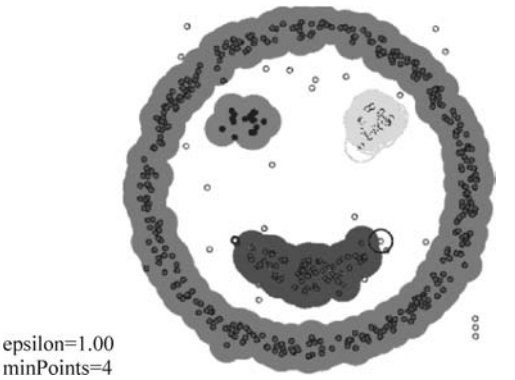

图 3.15 DBSCAN 实现笑脸聚类(见彩插)

DBSCAN 聚类的一般步骤如下。

(1) DBSCAN 以一个从未访问过的任意起始数据点开始。这个点的邻域是用距离 ε(所有在 ε 距离内的点都是邻点)来提取的。

(2) 如果在这个邻域中有足够数量的点(根据 minPoints)，那么聚类过程就开始了，并且当前的数据点成为新聚类中的第一个点。否则，该点将被标记为噪声(稍后这个噪声点可能会成为聚类的一部分)。在这两种情况下，这一点都被标记为“访问(Visited)”。

(3) 对于新聚类中的第一个点，其 ε 距离附近的点也会成为同一聚类的一部分。这一过程使在 ε 邻近的所有点都属于同一个聚类，然后将此过程重复应用到所有刚刚添加到聚类组的新点。

(4) 步骤(2)和步骤(3)的过程将重复，直到聚类中的所有点都被确定，就是说在聚类附近的所有点都已被访问和标记。

(5) 一旦完成了当前的聚类，就会检索并处理一个新的未访问点，这将导致进一步的聚类或噪声的发现。这个过程不断地重复，直到所有的点被标记为已被访问。因为在所有的点都被访问过之后，每个点都被标记为属于一个聚类或者是噪声。

DBSCAN 比其他聚类算法有一些优势。首先，它不需要一个预设定的聚类数量。它还将异常值识别为噪声，而不像均值偏移聚类算法，即使数据点非常不同，后者也会将它们放入一个聚类中。此外，它还能很好地找到任意大小和任意形状的聚类。

DBSCAN 的主要缺点是，当聚类具有不同的密度时，它的性能不像其他聚类算法那

样好。这是因为当密度变化时，距离阈值 ε 和识别邻近点的 minPoints 的设置会随着聚类的不同而变化。这种缺点也会出现在非常高维的数据中，因为距离阈值 ε 会变得难以估计。

3.4.4　高斯混合模型的期望最大化(EM)聚类


微课视频

K-Means 聚类的一个主要缺点是它对聚类中心的平均值的使用很简单、幼稚。可以通过图 3.16 来了解为什么这不是最好的方法。图 3.16(a)看起来很明显有两个圆形的聚类，不同的半径以相同的平均值为中心。此时 *K*-Means 聚类无法处理，因为聚类的均值非常接近。在聚类不是循环的情况下，*K*-Means 聚类也会失败，这也是使用均值作为聚类中心的结果，如图 3.16(b)所示。

图 3.16　*K*-Means 聚类两个失败案例(见彩插)

高斯混合模型(GMM)比 *K*-Means 更具灵活性。使用高斯混合模型，可以假设数据点是高斯分布的；比起说它们是循环的，这是一个不那么严格的假设。这样，就有两个参数来描述聚类的形状：平均值和标准差。以二维的例子为例，这意味着聚类可以采用任何形式的椭圆形状(因为在 x 和 y 方向上都有标准差)。因此，每个高斯分布可归属于一个单独的聚类。

为了找到每个聚类的高斯分布的参数(例如平均值和标准差)，将使用一种叫作期望最大化(EM)的优化算法。如图 3.17 所示，就可以看到高斯混合模型是被拟合到聚类上的。然后，可以继续进行期望的过程——使用高斯混合模型实现最大化聚类。

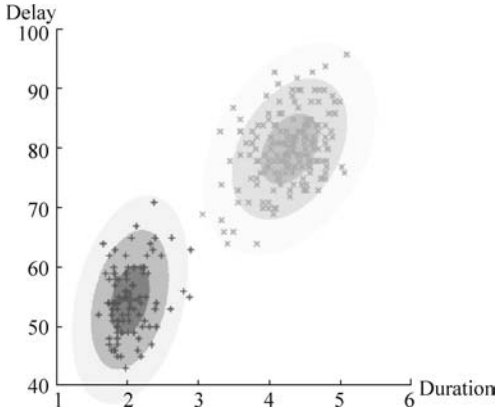

图 3.17　使用高斯混合模型的期望最大化聚类(见彩插)

高斯混合模型的期望最大化(EM)聚类的一般步骤如下。

(1) 首先选择聚类的数量(如 K-Means),然后随机初始化每个聚类的高斯分布参数。通过快速查看数据,可以尝试为初始参数提供良好的猜测。注意,在图 3.17 中可以看到,这并不是必要的,虽然高斯混合模型开始时的表现非常不好,但是很快就被优化了。

(2) 给定每个聚类的高斯分布,计算每个数据点属于特定聚类的概率。一个点离高斯混合模型中心越近,它就越有可能属于那个聚类。这应该是很直观的,因为有一个高斯分布,假设大部分的数据都离聚类中心很近。

(3) 基于这些概率,为高斯分布计算一组新的参数,这样就能最大限度地利用聚类中的数据点。使用数据点位置的加权和来计算这些新参数,权重是属于该特定聚类的数据点的概率。为了解释这一点,可以看一下图 3.17,特别是黄色的聚类作为例子。分布在第一次迭代中是随机的,但是可以看到大多数的黄色点都在这个分布的右边。当计算一个由概率加权的和,即使在中心附近有一些点,它们中的大部分都在右边,因此,自然分布的均值更接近于这些点。还可以看到,大多数点都是"从右上角到左下角"分布的。因此,标准差的变化是为了创造一个更符合这些点的椭圆,从而使概率的总和最大化。

(4) 步骤(2)和(3)被迭代、重复,直到收敛。分布不会在迭代这个过程中变化很多。

使用高斯混合模型有两个关键的优势。首先,高斯混合模型在聚类协方差方面比 K-Means 聚类要灵活得多;根据标准差参数,聚类可以采用任何椭圆形状,而不是局限于圆形。K-Means 聚类实际上是高斯混合模型的一个特例,每个聚类在所有维度上的协方差都接近 0。其次,根据高斯混合模型的使用概率,每个数据点可以有多个聚类。因此,如果一个数据点位于两个重叠的聚类的中间,通过说 $x\%$ 属于 1 类,而 $y\%$ 属于 2 类,可以简单地定义它的类。


微课视频

3.4.5 层次聚类

层次聚类算法实际上分为两类:自上而下或自下而上。自下而上的算法在一开始就将每个数据点视为一个单一的聚类,然后依次合并(或聚集)类,直到所有类合并成一个包含所有数据点的单一聚类。因此,自下而上的层次聚类称为合成聚类或 HAC。聚类的层次结构用一棵树(或树状图)表示。树的根是收集所有样本的唯一聚类,而叶子是只有一个样本的聚类,如图 3.18 所示。

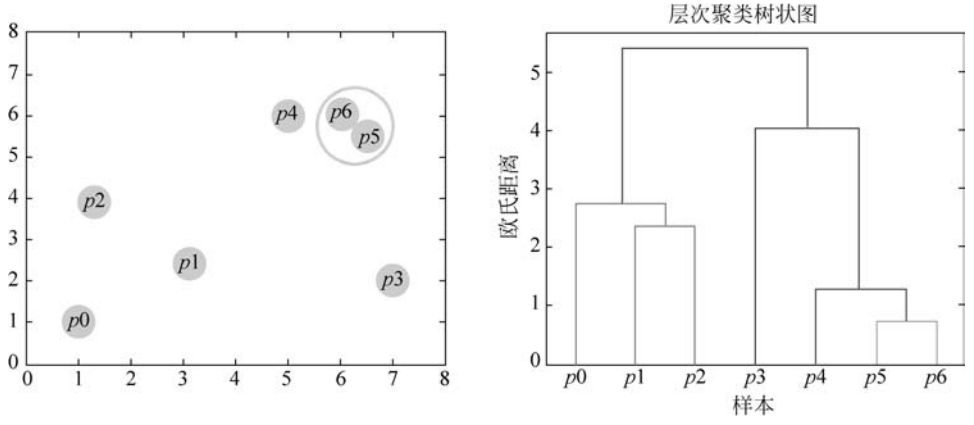

图 3.18 层次聚类(见彩插)

层次聚类的一般步骤如下。

(1) 首先将每个数据点作为一个单独的聚类进行处理。如果数据集有 X 个数据点，那么就有了 X 个聚类。然后选择一个度量两个聚类之间距离的距离度量。作为一个示例，将使用平均连接(Average Linkage)聚类，它定义了两个聚类之间的距离，即第一个聚类中的数据点和第二个聚类中的数据点之间的平均距离。

(2) 在每次迭代中，将两个聚类合并为一个。将两个聚类合并为具有最小平均连接的组。比如根据选择的距离度量，这两个聚类之间的距离最小，因此是最相似的，应该组合在一起。

(3) 重复步骤(2)直到到达树的根。只有一个包含所有数据点的聚类。通过这种方式，可以选择最终需要多少个聚类，只需选择何时停止合并聚类，也就是停止建造这棵树的时候。

层次聚类算法不要求指定聚类的数量，甚至可以选择哪个聚类看起来最好。此外，该算法对距离度量的选择不敏感；它们的工作方式都很好，而对于其他聚类算法，距离度量的选择是至关重要的。层次聚类方法的一个特别好的用例是，当底层数据具有层次结构时，可以恢复层次结构；而其他的聚类算法无法做到这一点。层次聚类的优点是以低效率为代价的，因为它具有 $O(n^3)$ 的时间复杂度，与 K-Means 聚类和高斯混合模型的线性复杂度不同。

下面给出 K-Means 聚类的实现代码，并对比不同簇的效果。最后再利用 100 个分量的 K-Means、PCA 和 NMF 进行图像重建的对比。

代码清单 3-2：*K*-Means

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# 生成模拟的二维数据
X, y = make_blobs(random_state = 1)
# 构建聚类模型
kmeans = KMeans(n_clusters = 3)
kmeans.fit(X)

# 在 kmeans.labels_属性中找到这些标签
print("Cluster memberships:\n{}".format(kmeans.labels_))

# 对训练集运行 predict 会返回与 labels_相同的结果
print(kmeans.predict(X))

# 用三角形表示
mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers = 'o')
mglearn.discrete_scatter(
kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], [0, 1, 2], markers = '^',
markeredgewidth = 2)
# 使用更多或更少的簇中心
fig, axes = plt.subplots(1, 2, figsize = (10, 5))
# 使用 2 个簇中心:
```

```
kmeans = KMeans(n_clusters = 2)
kmeans.fit(X)
assignments = kmeans.labels_
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax = axes[0])
# 使用 5 个簇中心
kmeans = KMeans(n_clusters = 5)
kmeans.fit(X)
assignments = kmeans.labels_
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax = axes[1])

# 使用 2 个簇和 5 个簇的 K 均值算法找到簇分配
X_varied, y_varied = make_blobs(n_samples = 200,
cluster_std = [1.0, 2.5, 0.5], random_state = 170)
y_pred = KMeans(n_clusters = 3, random_state = 0).fit_predict(X_varied)
mglearn.discrete_scatter(X_varied[:, 0], X_varied[:, 1], y_pred)
plt.legend(["cluster 0", "cluster 1", "cluster 2"], loc = 'best')
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")

# 生成一些随机分组数据
X, y = make_blobs(random_state = 170, n_samples = 600)
rng = np.random.RandomState(74)
# 变换数据使其拉长
transformation = rng.normal(size = (2, 2))
X = np.dot(X, transformation)
# 将数据聚类成 3 个簇
kmeans = KMeans(n_clusters = 3)
kmeans.fit(X)
y_pred = kmeans.predict(X)
# 画出簇分配和簇中心
plt.scatter(X[:, 0], X[:, 1], c = y_pred, cmap = mglearn.cm3)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker =
'^', c = [0, 1, 2], s = 100, linewidth = 2, cmap = mglearn.cm3)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")

# 生成模拟的 two_moons 数据(这次的噪声较小)
from sklearn.datasets import make_moons
X, y = make_moons(n_samples = 200, noise = 0.05, random_state = 0)
# 将数据聚类成 2 个簇
kmeans = KMeans(n_clusters = 2)
kmeans.fit(X)
y_pred = kmeans.predict(X)
# 画出簇分配和簇中心
plt.scatter(X[:, 0], X[:, 1], c = y_pred, cmap = mglearn.cm2, s = 60)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
marker = '^', c = [mglearn.cm2(0), mglearn.cm2(1)], s = 100, linewidth = 2)
plt.xlabel("Feature 0")
```

```
plt.ylabel("Feature 1")
# 并排比较 PCA、NMF 和 K 均值
X_train, X_test, y_train, y_test = train_test_split( X_people, y_people, stratify = y_people,
random_state = 0)
nmf = NMF(n_components = 100, random_state = 0)
nmf.fit(X_train)
pca = PCA(n_components = 100, random_state = 0)
pca.fit(X_train)
kmeans = KMeans(n_clusters = 100, random_state = 0)
kmeans.fit(X_train)
X_reconstructed_pca = pca.inverse_transform(pca.transform(X_test))
X_reconstructed_kmeans = kmeans.cluster_centers_[kmeans.predict(X_test)]
X_reconstructed_nmf = np.dot(nmf.transform(X_test), nmf.components_)

fig, axes = plt.subplots(3, 5, figsize = (8, 8),
subplot_kw = {'xticks': (), 'yticks': ()})
fig.suptitle("Extracted Components")
for ax, comp_kmeans, comp_pca, comp_nmf in zip(
axes.T, kmeans.cluster_centers_, pca.components_, nmf.components_):
ax[0].imshow(comp_kmeans.reshape(image_shape))
ax[1].imshow(comp_pca.reshape(image_shape), cmap = 'viridis')
ax[2].imshow(comp_nmf.reshape(image_shape))
axes[0, 0].set_ylabel("kmeans")
axes[1, 0].set_ylabel("pca")
axes[2, 0].set_ylabel("nmf")
fig, axes = plt.subplots(4, 5, subplot_kw = {'xticks': (), 'yticks': ()},
figsize = (8, 8))
fig.suptitle("Reconstructions")
for ax, orig, rec_kmeans, rec_pca, rec_nmf in zip(
axes.T, X_test, X_reconstructed_kmeans, X_reconstructed_pca, X_reconstructed_nmf):
ax[0].imshow(orig.reshape(image_shape))
ax[1].imshow(rec_kmeans.reshape(image_shape))
ax[2].imshow(rec_pca.reshape(image_shape))
 ax[3].imshow(rec_nmf.reshape(image_shape))
 axes[0, 0].set_ylabel("original")
 axes[1, 0].set_ylabel("kmeans")
 axes[2, 0].set_ylabel("pca")
 axes[3, 0].set_ylabel("nmf")

 # 对比 K 均值的簇中心与 PCA 和 NMF 找到的分量
 X, y = make_moons(n_samples = 200, noise = 0.05, random_state = 0)
 kmeans = KMeans(n_clusters = 10, random_state = 0)
 kmeans.fit(X)
 y_pred = kmeans.predict(X)
 plt.scatter(X[:, 0], X[:, 1], c = y_pred, s = 60, cmap = 'Paired')
 plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 60,
 marker = '^', c = range(kmeans.n_clusters), linewidth = 2, cmap = 'Paired')
```

```
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
print("Cluster memberships:\n{}".format(y_pred))

# 利用 kmeans 的 transform 方法
distance_features = kmeans.transform(X)
print("Distance feature shape: {}".format(distance_features.shape))
print("Distance features:\n{}".format(distance_features))
```

3.5 本章小结

本章的内容主要讨论了无监督学习的数据预处理技术以及两个最重要的无监督学习应用：降维和聚类。

降维是为了数据压缩或者数据可视化，具体的实现方法有投影和流形学习，最重要的降维算法是主成分分析(PCA)。本章详细讨论了 PCA 实现的思想和实现的步骤，并利用 PCA 算法进行了一个案例：半导体制造数据降维。

聚类的应用与评估是一个非常定性的过程，通常在数据分析的探索阶段很有帮助。学习了 5 种聚类算法：K-Means 聚类、高斯混合模型的期望最大化(EM)聚类、DBSCAN、均值偏移聚类和层次聚类。这 5 种算法都可以控制聚类的粒度(Granularity)。K 均值允许指定想要的簇的数量，而 DBSCAN 允许用 ε 参数定义接近程度，从而间接影响簇的大小。5 种方法都可以用于大型的现实世界数据集，都相对容易理解，也都可以聚类成多个簇。

5 种聚类算法的优点稍有不同。K 均值可以用簇的平均值来表示簇；它还可以被看作一种分解方法，每个数据点都由其簇中心表示；DBSCAN 可以检测到没有分配任何簇的“噪声点”，还可以帮助自动判断簇的数量，与其他方法不同，它允许簇具有复杂的形状。DBSCAN 有时会生成大小差别很大的簇，这可能是它的优点，也可能是缺点；层次聚类可以提供数据的可能划分的整个层次结构，可以通过树状图轻松查看。

习题

1. 降低数据集维度的主要动机是什么？主要弊端有哪些？
2. 什么是维度的诅咒？
3. 一旦数据集被降维，是否还有可能逆转？如果有，需要怎么做？如果没有，请说明原因。
4. PCA 可以用来给高度非线性数据集降维吗？
5. 假设在一个 1000 维数据集上执行 PCA，方差解释比设为 95%。产生的结果数据集维度是多少？
6. 如何在数据集上评估降维算法的性能？
7. 聚类算法有哪些？
8. 分别说明 K-Means、DBSCAN、层次聚类的优缺点。