第十二章无监督学习

监督学习和无监督学习的特点：

- 监督学习通常使用一组特征 $X_1, X_2, \ldots, X_P$ 来预测目标变量 Y
- 无监督学习没有目标变量 Y，我们的目的是找到 $X_1, X_2, \ldots, X_p$ 之间的一些潜在关系

无监督学习尝试可视化，发现特征 $X_1, X_2, \ldots, X_p$ 中的子组。本章讨论主成分分析和聚类两种无监督学习模式

12.1 无监督学习的挑战

无监督学习相比于监督学习面临更多挑战：

- 目标变得主观，因为我们没有具体的预测变量 Y
- 无法检测我们的结果，这与监督学习可以用交叉检验或者测试集不同

无监督学习通常作为**探索性数据分析 (exploratory data analysis)** 的一部分。它的应用包括：

1. 在若干名癌症患者中找到一些子组，以便更好地对这种疾病进行分类
2. 根据用户的购物习惯，为他推荐喜欢的商品
3. 将相似浏览记录的人识别出来，进行搜索推广

12.2 主成分分析

我们之前讨论过主成分回归。现在我们讨论的**主成分分析 (Principal components analysis)** 是一种无监督学习方法，它从一组特征 $X_1, X_2, \ldots, X_p$ 中找到衍生的特征作为主成分。它的应用包括：

- 进行数据可视化
- 数据插值，为有空缺的矩阵填充值
- 探寻潜在的特征

12.2.1 什么是主成分

对于一组特征 $X_1, X_2, \ldots, X_p$，我们可能希望查看它们的数据分布。两两特征之间的散点图可以实现这一点，但是当 p 过多时将会很不方便。现在，我们希望用更少的特征尽可能多地反应原来的信息，以可视化这些数据

PCA 找到包含尽可能多的数据信息的低维表示，它通过观测值沿每个维度的变化量来衡量更低的有意义的维度。所有的降维都是原来 p 个特征的线性组合。一组特征的**第一主成分 (first principal component)** 是标准化的 (normalized) $X_1, X_2, \ldots, X_p$ 的线性组合：

$$X_1 = \Phi_{11}X_1 + \Phi_{21}X_2 + \cdots + \Phi_{p1}X_p$$

这个线性组合拥有最大的方差。此处标准化是指 $\sum_{i=1}^{p}\Phi_{j1}^2 = 1$，我们称 $\Phi_{i1}, i = 1, 2, \ldots, p$ 为主成分的**载荷 (loading)**，载荷组成了组成分的载荷向量 $\Phi_1 = (\Phi_{11}, \Phi_{21}, \ldots \Phi_{p1})^T$。对载荷平方和的约束限制了任意的方差变化

由于我们只关心方差，我们假设设计矩阵 X 的每一列均值为 0，然后对于每个样本：

$$z_{i1} = \Phi_{11}x_{i1} + \Phi_{21}x_{i2} + \cdots + \Phi_{p1}x_{ip}$$

我们寻找这样的一组载荷使它们的样本方差最大。换句话说，第一组成分向量是下面这个优化式子的解：

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

对于上式，由于 $\frac{1}{n}\sum_{i=1}^{n}x_{ij} = 0$，$z_{11}, \ldots z_{n1}$ 的均值也是 0，且我们要优化的式子即是最大化 n 对样本方差 z_{i1}，我们称 $z_{11}, \ldots, z_{n1}$ 为第一主成分的**得分 (score)**。最大化上式可用**特征分解 (eigen decomposition)** 的方法解决，但超出本书范围

关于主成分的几何解释如下：载荷向量 Φ_1 的元素 $\Phi_{11}, \Phi_{21}, \ldots, \Phi_{p1}$ 定义了特征空间中的一个方向，在此方向上的数据变化最大。如果我们将 n 对数据投影到这个方向，投影值 $z_{11}, \ldots, z_{n1}$ 就是它们的主成分得分，在几何上即是在这条主成分轴 Z_1 上的坐标位置。载荷向量中的元素大小表示特征在投影方向上的权重大小

找到第一个主成分 Z_1 后，我们可以找到第二个主成分 Z_2，它也是 $X_1, X_2, \ldots, X_p$ 的线性组合，但是在所有与 Z_1 不相关的线性组合中，这个组合的方差最大。第二个主成分的分数形式如下：

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip}$$

这里 Φ_2 是第二主成分的载荷向量。事实上，为了约束 Z_2 与 Z_1 不相关，Z_2 必须是与 Z_1 正交也即垂直的。为了找到 Φ_2，我们求解一个类似的优化问题，但是添加条件是 Φ_2 与 Φ_1 正交

计算完主成分后，我们可以将他们相互绘制，以获得低维度的可视化图。例如，绘制分数向量 Z_1, Z_2, Z_3 的两两组合，在几何上相当于将原始数据投影到 Φ_1, Φ_2, Φ_3 的子空间上并绘制投影点

12.2.2 主成分的另一种解释

上一节我们认为主成分就是原始数据变动最大的方向。下面提供另一种解释：主成分提供了一个能最接近观测值的低维表面

> 这里说是“表面”实际上是比原始维度低的图形的面。例如三维空间的低维表面是一个二维平面

第一个主成分的载荷向量的一个重要属性是：它是 p 维空间中最接近 n 个观测值的直线。将此概念推广，则前两个主成分是最接近 n 个观测值的平面；前三个主成分是三维超平面

使用这个解释，则前 M 个主成分得分向量 Z 和前 M 个主成分载荷向量 Φ 提供了一个最优的 M 维估计，它在欧氏距离上最优。对第 i 个观测值 x_{ij}：

$$x_{ij} \approx \sum_{m=1}^{m} z_{im}\Phi jm$$

正式来看，给定原始数据矩阵 $\mathbf{X}$，为了找到对它们的估计，使得 $x_{ij} \approx \sum_{m=1}^{M} a_{im}b_{jm}$，我们最小化残差平方和：

$$\underset{\mathbf{A}\in\mathbb{R}^{n\times M},\mathbf{B}\in\mathbb{R}^{p\times M}}{\text{minimize}} \left\{ \sum_{j=1}^{p}\sum_{i=1}^{n}\left(x_{ij}-\sum_{m=1}^{M}a_{im}b_{jm}\right)^2 \right\}$$

其中 $\mathbf{A}$ 是一个 $n\times M$ 的矩阵，位置 (i,m) 的元素就是 a_{im}，$\mathbf{B}$ 是一个 $p\times m$ 的矩阵，位置 (j,M) 的元素就是 b_{jm}。可以证明对于任意的 M，$\mathbf{A}$ 和 $\mathbf{B}$ 实际上列分别对应主成分得分 $Z_1,Z_2,\ldots,Z_M$ 和主成分载荷 $\Phi_1,\Phi_2,\ldots,\Phi_M$。优化问题的最优解得到的估计矩阵就是最终答案。这意味这我们得到的矩阵要能最小化：

$$\sum_{i,j}\left(x_{ij}-\sum_{m=1}^{M}z_{im}\Phi_{jm}^2\right)$$

总的来说，M 个主成分得分和载荷向量在 M 足够大时能通过对原始数据的很好近似，当 $M=\min(n-1,p)$ 时，$x_{ij}=\sum_{m=1}^{m}z_{im}\Phi_{jm}$

12.2.3 解释方程的比例

在对原始数据投影后，我们感兴趣的问题是对于每个组成分，它**解释的方差比例 (proportion of variance explained, PVE)** 由多少。对于已经标准化的数据，总体方差：

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

第 m 个主成分解释的方差：

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

因此，PVE 的计算如下：

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

每个主成分的 PVE 都是正数，计算累计 PVE 可对前 M 个主成分的 PVE 进行求和，它们的 PVE 之和为 1

总体方差可以分解为：

$$\underbrace{\sum_{j=1}^{p}\frac{1}{n}\sum_{i=1}^{n}x_{ij}^2}_{\text{Var. of data}}=\underbrace{\sum_{m=1}^{M}\frac{1}{n}\sum_{i=1}^{n}z_{im}^2}_{\text{Var. of first } M \text{ PCs}}+\underbrace{\frac{1}{n}\sum_{j=1}^{p}\sum_{i=1}^{n}\left(x_{ij}-\sum_{m=1}^{M}z_{im}\phi_{jm}\right)^2}_{\text{MSE of } M\text{-dimensional approximation}}$$

通过最大化前面 M 个主成分的方差，我们就能最小化 M 维近似的均方误差，反之亦然。

此外，我们通过上式可以发现 PVE 的定义等价于：

$$1-\frac{\sum_{j=1}^{p}\sum_{i=1}^{n}\left(x_{ij}-\sum_{m=1}^{M}z_{im}\phi_{jm}\right)^2}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2}=1-\frac{\text{RSS}}{\text{TSS}},$$

其中 TSS 表示 X 的总平方和，RSS 表示 M 个主成分维度重构给出的残差平方和，则上式表示保留的方差比例。这与 R^2 的定义非常类似，我们可以将 PVE 解释为前 M 个主成分给出的 R^2

通过绘制累积 PVE 的图像，我们得到了**碎石图 (scree plot)**，它能逐步反应各主成分解释的方差比例

12.2.4 关于 PCA 的更多信息

变量尺度问题：
我们之前执行 PCA 时都将变量标准化了。与线性回归不同，对变量乘以系数将会影响 PCA 的结果，量纲也会影响 PCA 结果，这是因为载荷向量需要赋予方差大的特征更高的系数。然而，对于同量纲的问题，我们不应该对其进行标准化

> 对于同量纲且方差差异有意义的数据，我们不需要标准化；但若它们数值范围差异过大，或者仅需要分析相关性结构，我们可以标准化
> 标准化会消除原始方差差异，仅衡量相关性

主成分的唯一性：
主成分的方向通常唯一 (除非矩阵的特征值相同，实际中很少出现)。载荷向量的符号正负不影响其方向性，因此我们分析时应当只考虑载荷的绝对值大小

决定主成分数量：
决定主成分数量是一个主观的过程，取决于我们希望探索多少感兴趣的因子。如果前面几个因子我们感兴趣，则我们很可能继续选择更多的因子来分析，反之我们不会继续关注后面的因子。这是 PCA 被称为探索性分析的原因

一种决定主成分数量的方法是观察碎石图，当图像出现拐弯时，后续的主成分解释的方差将相当小，我们不会选择；然而，这也是主观的过程

> 用于监督学习的 PCA 可以通过交叉检验等方式来决定主成分数量

12.2.5 主成分的其他用处

我们之前学过的主成分回归实际上用到了主成分分数向量作为特征来执行回归，这是因为 $M << p$ 个主成分分数向量通常能减少噪声影响

12.3 缺失值和矩阵补全

数据集通常不是完整的，包含一些随机的缺失值，一些基础的处理方式包括：

- 删除包含缺失值的行。这可能导致一些浪费，尤其是损失比例很大时
- 若 x_{ij} 缺失，用第 j 列的均值替换它

本节介绍如何用主成分来**估算 (impute)** 缺失值，这种方案叫做**矩阵补全 (matrix completion)**。补全后的矩阵可以用于统计学，例如线性回归或者 LDA。估算缺失值基于**随机缺失 (missing at random)** 的数据集，如果数据的缺失本身提供信息 (如删失数据中的超过刻度等)，则不适用这个方法

事实上，数据会不可避免地丢失。例如考察 n 个客户对 Netfilx 的 p 部电影的评分，大多数用户不可能看过全部电影。如果我们能对矩阵进行补全，则可以支持我们的**推荐系统 (recommender system)**

用主成分补充缺失值
补全矩阵缺失值和获得主成分可以同时进行。考察下面的优化问题：

$$\min_{\mathbf{A}\in\mathbb{R}^{n\times M},\mathbf{B}\in\mathbb{R}^{p\times M}}\left\{\sum_{(i,j)\in\mathcal{O}}\left(x_{ij}-\sum_{m=1}^{M}a_{im}b_{jm}\right)^2\right\}$$

其中 $\mathcal{O}$ 是是所有观察到的观测值 (i,j) 的集合，是原始矩阵 $n\times p$ 对的一个子集。求解出估计的矩阵 $\mathbf{A}$ 和 $\mathbf{B}$ 后，我们有：

- 可以估计缺失值 x_{ij}，即 $\hat{x}_{ij}=\sum_{m=1}^{M}\hat{a}_{im}\hat{b}_{jm}$，其中 $\hat{a}_{im},\hat{b}_{jm}$ 分别是 $\mathbf{A},\mathbf{B}$ 的第 (i,m) 和第 (j,m) 个值
- 可以估算前 M 个主成分得分和载荷矩阵，即用估算的缺失值来获取

事实证明与完整的情况不同，求解上面的矩阵不适用特征分解的方法，但是下面的迭代算法提供了一个估算方案：

1. 首先创建一个 $n\times p$ 的完整数据矩阵 $\widetilde{X}$，它的元素 (i,j) 满足：

   $$\tilde{x}_{ij}=\begin{cases}x_{ij} & \text{if } (i,j)\in\mathcal{O}\\ \bar{x}_j & \text{if } (i,j)\notin\mathcal{O},\end{cases}$$

   其中 $\bar{x}_j$ 是第 j 列观测值的平均值。此处我们得到 $\mathcal{O}$ 就是未缺失的观测值的集合。接着，当优化目标式子 (上式) 未满足时，执行：
2. 用补全的完整数据矩阵 $\widetilde{X}$ 求解分解的矩阵 $\mathbf{A}$ 和 $\mathbf{B}$，即优化下面的式子：

   $$\min_{\mathbf{A}\in\mathbb{R}^{n\times M},\mathbf{B}\in\mathbb{R}^{p\times M}}\left\{\sum_{j=1}^{p}\sum_{i=1}^{n}\left(\tilde{x}_{ij}-\sum_{m=1}^{M}a_{im}b_{jm}\right)^2\right\}$$
3. 对于不在 $\mathcal{O}$ 的元素 (i,j)，使 $\tilde{x}_{ij}\leftarrow\sum_{m=1}^{M}\hat{a}_{im}\hat{b}_{jm}$
4. 计算目标

$$\sum_{(i,j)\in\mathcal{O}}\left(x_{ij}-\sum_{m=1}^{M}\hat{a}_{im}\hat{b}_{jm}\right)^2$$

执行循环完毕后，返回缺失值 $\tilde{x}_{ij},(i,j)\notin\mathcal{O}$

> 上面算法在每次循环都会减小优化目标，但是不能保证该算法能够实现全局最优

人为删去随机数据的实验表明，使用插补法得到真实值和插补值之间存在一定的相关性。虽然插补矩阵计算的模型结果不如真是矩阵，但是它依然提供了一个相当好的近似，考虑到实际中的数据大多存在缺失，插补法依然有其存在价值

> 此实验中的 `USErrests` 数据集特征 $p=4$ 相对较少，可能影响估算算法的稳健性

在使用算法估算时，我们常常需要考虑合适的主成分数量 M，这可以通过类似验证集的方法实现：我们筛选出一个验证集，人为删去其中的一些变量，然后挑选 M 来估计并评价不同的 M 的效果，最后再泛化到整个数据集中

推荐系统
流媒体播放网站的推荐系统依赖于各种算法。以 Netfilx 为例，为了推荐用户喜欢的电影，它要求用户对 p 部电影中看过的进行评分，产生了一个非常大的矩阵 $n\times p$，一个早期的例子是 $n=480,189$ 和 $p=17770$，其中 99% 的元素为空。为了补全这些矩阵，Netflix 基于这样一个思想：第 i 位用户观看的电影集合将与其他用户观看的电影有重叠；此外，其他用户与第 i 位用户有相似偏好。因此可用第 i 位用户未看过电影的类似用户的评分来预测第 i 位用户是否喜欢这个电影

在数学上，这相当于用上面的算法估算 $\hat{x}_{ij}=\sum_{m=1}^{M}\hat{a}_{im}\hat{b}_{jm}$，且 M 主成分的具体意义是某个团体分类：

- $\hat{a}_{im}$ 表示第 i 个用户属于第 m 个团体的强度 (喜欢某类电影用户的集合)
- $\hat{b}_{jm}$ 表示第 j 个电影属于第 m 类别的强度

类似于上面算法的主成分模型是许多推荐系统的核心

12.4 聚类方法

聚类 (clustering) 是一类包含广阔的算法，它用于寻找数据集的子组或**簇 (clusters)**。聚类分割各种簇，使得相似的数据靠的越近，不同的数据离得越远。如何衡量数据的相似与不同取决于我们分析的数据集的背景知识，例如对于 n 名癌症病人的基因表达数据，我们希望找到癌症的部分亚群

> 市场细分需要用到聚类方法。例如我们希望识别哪些人群容易接受哪种类型的广告

PCA 与聚类的区别如下：

- PCA 寻找原始观测值的低维表示，以尽可能地解释数据的方差
- 聚类在观测值中寻找同质的子组

本章介绍两种最著名的聚类算法：**K-means 聚类 (K-means clustering)** 和 **层次聚类 (hierarchical clustering)**。前者在给定聚类数目的情况下聚类，后者则是将从 1 到 n 的所有可能聚类以**聚类树图 (dendrogram)** 的形式展现。二者各有优劣

一般而言，我们可以根据特征对观测进行聚类，以识别观测之间的子群，也可以根据观测对特征进行聚类，以发现特征之间的子群。在接下来的内容中，为了简单起见，我们将根据特征讨论聚类观测

> 转置矩阵即可改变聚类模式

12.4.1 K-means 聚类

K-means 聚类在指定聚类数目 K 之后，将观测数据完全分配到这些簇中。这个过程非常直观。首先定义一些记号：

1. $C_1 \cup C_2 \cup \cdots \cup C_K = \{1, \ldots, n\}$ 。即所有观测值都在 K 个簇中
2. $C_k \cap C_{k'} = \emptyset, \forall k \neq k'$。即所有簇互斥，没有属于两个簇的元素

K-means 聚类的核心思想是类内差异尽可能小的聚类是好的聚类。记第 i 个观测值属于第 k 个簇为 $i \in C_k$，则 K-means 实际上是一个优化问题：

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

其中 $W(C_k)$ 衡量了类内差异。类内差异有许多定义方式，一种最常见的是**欧式平方距离 (squared Euclidean distance)**：

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

其中 $|C_k|$ 表示第 k 个簇内观测值数量。上式即类内差异表示为第 k 个簇钟所有观测值成对之间的欧氏距离平方和处于观测数。由上面两个式子得到：

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

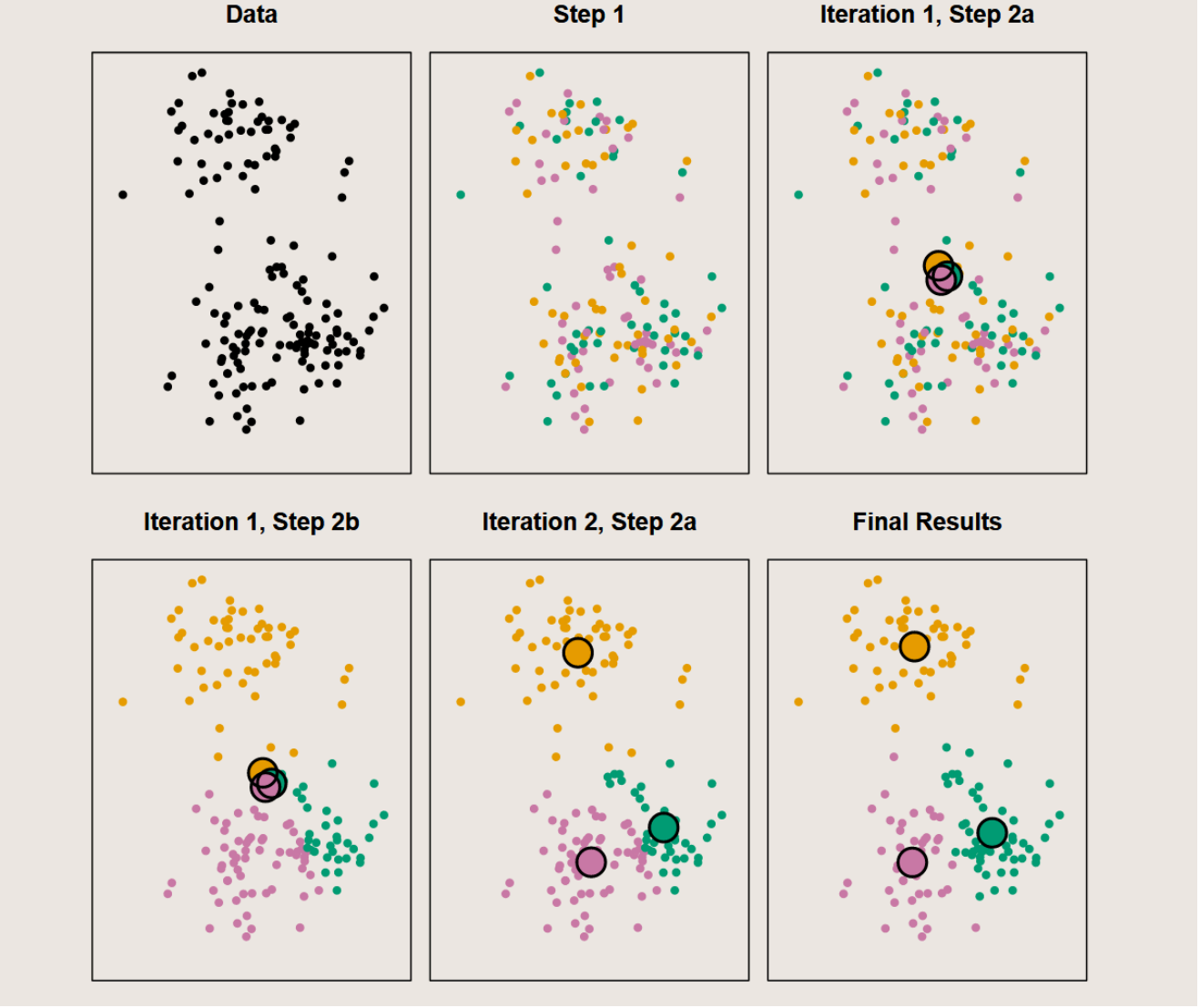
这个优化目标的精确求解很不容易，因为有 K^n 种可能的解法。下面这个算法可以给出局部最优解：

1. 对 $\forall$ 观测值，分配一个从 1 到 k 的数字作为初始聚类分配
2. 不断迭代，直到停止分配簇
   2.1 对于 K 个聚类中的每个簇，计算簇的**质心 (centroid)**。第 k 个簇的质心是第 k 个簇所有观测值特征均值的向量
   2.2 将每个观测值分配给距离质心最近的簇，其距离用欧氏距离衡量

下面的式子有助于我们理解上面算法能优化目标的原因：

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

其中 $\overline{x}_{kj}=\frac{1}{|C_k|}\sum_{i\in C_k}x_{ij}$ 表示特征 j 在簇 C_k 中的均值。等式右边实际上是簇内观测值到均值的平方距离和的两倍，即我们只需要优化簇内观测值到其质心的距离。当结果不再发生变化时，达到最优



K-means 聚类的最终结果依赖于初始的簇分类质心，因此需要多运行几次取最优的结果。这是因为 K-means 寻找的是局部最优解

> 簇的数量选取很重要，将在后面讲解

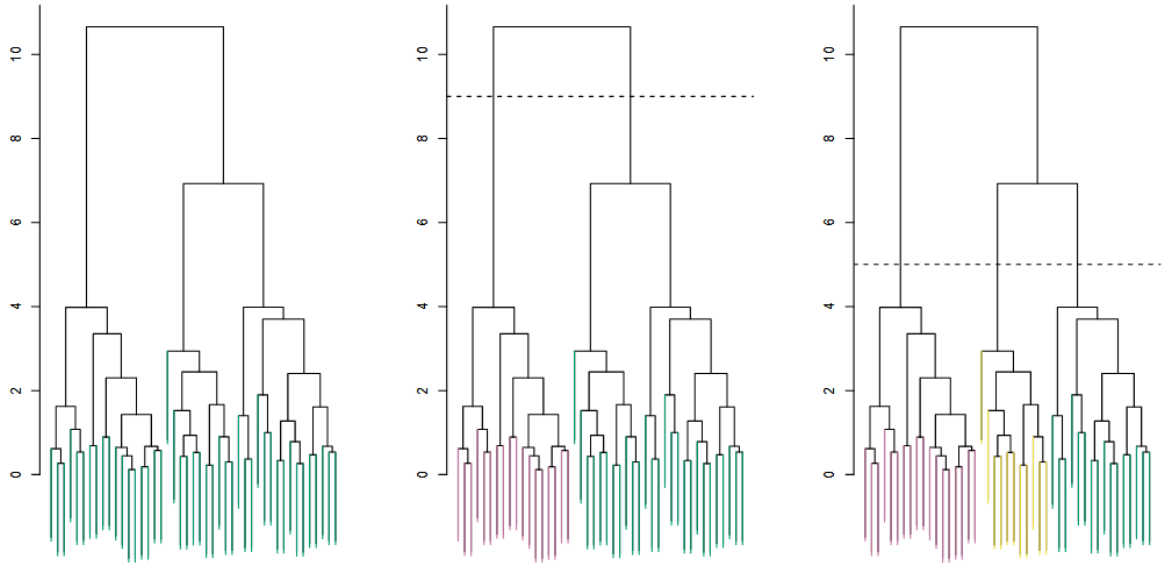
12.4.2 层次聚类

K-means 聚类的一个劣势是需要指定 K 的大小，因此层次聚类提供了一个更吸引人的方案，而且它还能生成直观的聚类树图。本节介绍**自下而上 (bottom-up)** 和 **凝聚式 (agglomerative)** 两种聚类方案

解释聚类树图

聚类树图是一颗倒着的树 (就像决策树)，每个叶子节点表示观测值之一，它们向树根逐渐合并成枝干。不难看出，越早融合的观测值相似程度越高，接近树根融合的枝干则可能完全不相同

值得注意的是，我们不能仅仅根据叶子节点的相近程度来判断它们的相似性，而应该结合它们位于的层次 (即从下往上融合的时机)。通过对树状图水平切割，我们能得到对应的聚类；不同的切割高度对应不同的聚类，相当于 K-means 中的 K



> n 个观测值可以产生 2^{n-1} 中不同的排列顺序，因此不能仅仅根据距离判断相似度

肉眼观察树状图可以直接让我们选择需要划分的聚类数量，然而，有时候这种划分并不容易抉择。层次聚类看上去是“嵌套”的，这也就是说更高处的切割将会包含更低处的切割 (子集的关系)；但是对于没有嵌套关系的数据集，这种划分往往不那么合适，这时候层次聚类的效果不如 K-means

> 合适的划分：切割成两个簇划分为“动物”和“植物“，切割为三个簇在植物中细分出”花草“和”树木“
> 不合适的划分：不同国籍人群的数据集，切割成两个簇划分为“男人”和“女人”，切割成三个簇则在女人中划分出三个国家，无层次嵌套关系

层次聚类算法
层次聚类的算法如下：

1. 从 n 个观测值和某个不相似度测量值 (例如欧氏距离) 开始，将观测值划分为 C_n^2 个不想似对，每个观测值为一个聚类
2. 从 $i = n, n-1, \ldots, 2$：
   2.1 检查所有不相似对的不相似度，选择不相似度最小的 (即最相似的) 进行合并
   2.2 对于新的对，重新计算不相似度

我们的不相似度仅仅度量了两个观测值之间的不相似情况，现在拓展到度量两组之间，这里引入了一个新的概念**连接 (linkage)**。连接包括完全 (complete)、平均（average)、单个 (single) 和质心 (centroid) 四种，前两者往往优于单个类型的连接，因为能产生更均平衡的树状图；质心连接常用在几何中，它可能产生一种称为**反转 (inversion)** 的现象，即后合并的节点高度可能低于先合并的，这是因为质心 (几何中心) 的计算是动态的

> 最终得到的树状图取决于连接方式

链接（Linkage）	描述（Description）
完全（Complete）	最大的聚类间不相似度。计算聚类 A 中的观测值与聚类 B 中的观测值之间的所有成对不相似度，并记录这些不相似度中的最大值。
单一（Single）	最小的聚类间不相似度。计算聚类 A 中的观测值与聚类 B 中的观测值之间的所有成对不相似度，并记录这些不相似度中的最小值。单一链接可能导致延伸的、拖尾的聚类，其中单个观测值会逐一融合。
平均（Average）	平均的聚类间不相似度。计算聚类 A 中的观测值与聚类 B 中的观测值之间的所有成对不相似度，并记录这些不相似度的平均值。
质心（Centroid）	聚类 A 的质心（长度为 p 的均值向量）与聚类 B 的质心之间的不相似度。质心链接可能导致不良的反转（inversions）。

不相似度度量的选择
通常，我们的不相似度度量量是欧氏距离。有时候，我们又会使用相关性来度量特征的相似性，即如果两个观测值的特征高度相关，则它们被认为相似，即时它们在空间上相隔很远

> 基于相关性的度量将注重观测值的轮廓 (profile) 和变化趋势而非绝对距离

选择欧氏距离或者相关性取决于我们研究的数据。对于购买商品的用户，如果用欧氏距离我们可能将购买数量很少 (即很少上网) 的用户聚类到一起，而不是他们对产品的偏好

用相关性度量相似性时，应当对数据标准化。这种标准化也适用于 K-means。若不标准化，则可能因为数值大小 (量纲大小) 错误聚类

12.4.3 聚类中的实际问题

小决定和大后果
执行聚类前我们需要的决定：

- 是否标准化数据
- 对于层次聚类
  - 不相似性的度量
  - 连接的选取
  - 剪切聚类树图的位置
- 对于 K-means 聚类，选择合适的 K

实践中，常常进行多种不同选择，选取最容易解决或者可解释的方案

> 聚类是主观性很强的

验证获得的簇
聚类后，我们获得若干数量的簇。然而，这些簇的划分是基于实际情况还是受到噪声影响的，我们无法知道。有一些用 p 值来决定是否接受这些簇的方法，但是本书不涉及，可参阅*The Elements of Statistical Learning, ESL*

其他在聚类中应当考虑的问题

不论是 K-means 还是层次聚类都会将 n 个观测值分配到所有可能的簇中。然而，这里面可能有些噪声，它们并不属于任何簇 (甚至属于多个簇)，但是强制被分类到各个簇中了。**混合模型 (mixture model)** 是一种软分类方案，它允许一个观测值被分配到多个簇中，参见 ESL

此外，聚类方法对数据的扰动很敏感。例如，我们对 n 个数据进行聚类，若删除一些观测值再聚类，很可能得到不同的聚类模型

对聚类结果的解读采取审慎的态度

前文提到聚类的一些小决定可能导致差异很大的结果，聚类的稳健性不高。因此，有必要使用多种参数聚类以及对数据集的子集进行聚类，查看哪些模式反复出现。最后，谨慎地分析结果，它们容易被误解或者过度解释

12.5 实验：无监督学习

基础引入：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.datasets import get_rdataset
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from ISLP import load_data
```

本章新引入：

```
from sklearn.cluster import \
    (KMeans,
     AgglomerativeClustering)
from scipy.cluster.hierarchy import \
    (dendrogram,
     cut_tree)
from ISLP.cluster import compute_linkage
```

12.5.1 主成分分析

`USArrests` 是 `R` 中的数据集，用 `get_rdataset()` 可以得到：

```
USArrests = get_rdataset('USArrests').data
print(USArrests.columns)
print(USArrests.mean()) #输出列均值
print(USArrests.var()) #输出列方差
```

> 注意：实际运行过程中，获取 `R` 数据集的过程比较慢

注意到由于单位引起的 `UrbanPop` 列的大方差，需要进行缩放为标准差为1；通常，均值也被设置为0

使用 `StandardScaler()` 生成一个标准化器，然后再对数据集应用：

```
scaler = StandardScaler(with_std=True,
                        with_mean=True)
USArrests_scaled = scaler.fit_transform(USArrests)
```

用 `PCA()` 建立一个转换器，它来自 `sklearn.decomposition` 包：

```
pcaUS = PCA()
```

通过 `fit()` ， `pcaUS` 可以获得数据集的主成分得分。拟合后， `mean_` 属性将返回列的均值(已被事先标准化为0)：

```
pcaUS.fit(USArrests_scaled)
print(pcaUS.mean_)
```

得到模型的主成分得分后，可应用于数据集(包括新的数据集)：

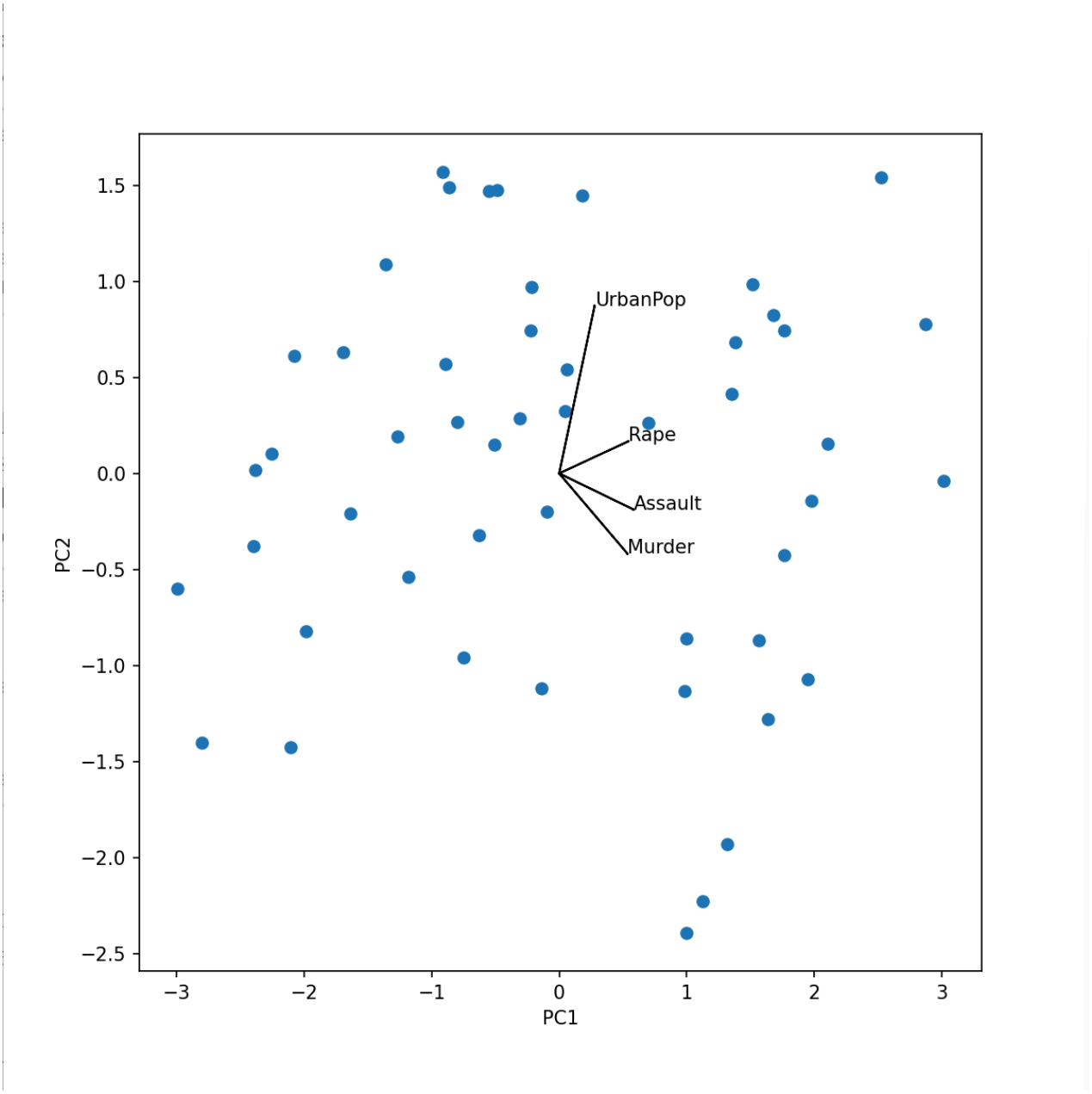
```
scores = pcaUS.transform(USArrests_scaled)
print(pcaUS.components_) #输出载荷向量
```

其中 `components_` 属性包括了主成分的载荷向量

这里手动实现一个 `bibplot` 用于可视化降维数据：

```
i, j = 0, 1 #指定主成分序号
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.scatter(scores[:,0], scores[:,1])
ax.set_xlabel('PC%d' % (i+1))
ax.set_ylabel('PC%d' % (j+1))
for k in range(pcaUS.components_.shape[1]): #对于原始的每个特征，绘制空间中的方向
    ax.arrow(0, 0, pcaUS.components_[i,k], pcaUS.components_[j,k])
    ax.text(pcaUS.components_[i,k],
            pcaUS.components_[j,k],
            USArrests.columns[k])
```

翻转符号得到的载荷向量实际意义一样；此处箭头长度表示载荷强度

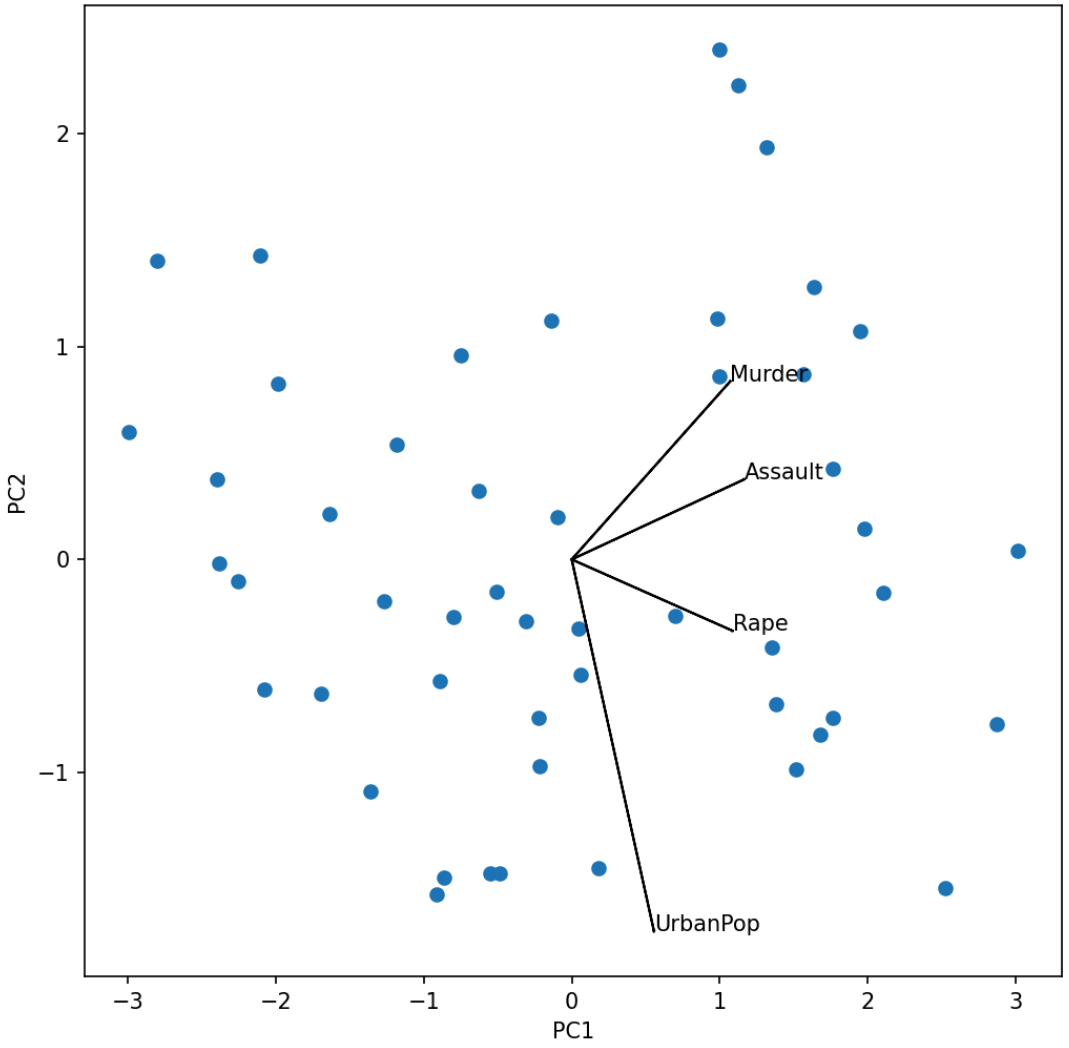


翻转符号：

```
#翻转y轴，显示不同挂的原始方向指向
scale_arrow = s_ = 2
scores[:,1] *= -1
pcaUS.components_[1] *= -1 #翻转y轴
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.scatter(scores[:,0], scores[:,1])
ax.set_xlabel('PC%d' % (i+1))
ax.set_ylabel('PC%d' % (j+1))
for k in range(pcaUS.components_.shape[1]):
    ax.arrow(0, 0, s_*pcaUS.components_[i,k], s_*pcaUS.components_[j,k])
    ax.text(s_*pcaUS.components_[i,k],
            s_*pcaUS.components_[j,k],
            USArrests.columns[k])
```

这里：

- `ax.arrow(x, y, dx, dy, **kwards)` 的前四个参数分别表示起始(x,y)坐标和终点坐标偏移量
- `ax.text(x, y, s, **kwards)` 的前三个参数分别表示起始(x,y)坐标和显示的文本



下面是一些关于主成分的参数：

```
print(scores.std(0, ddof=1)) #主成分得分的标准差
print(pcaUS.explained_variance_) #主成分得分的方差
print(pcaUS.explained_variance_ratio_) #主成分解释的方差比例PVE
```

根据PVE，我们可以画出每个PVE数值的图像：

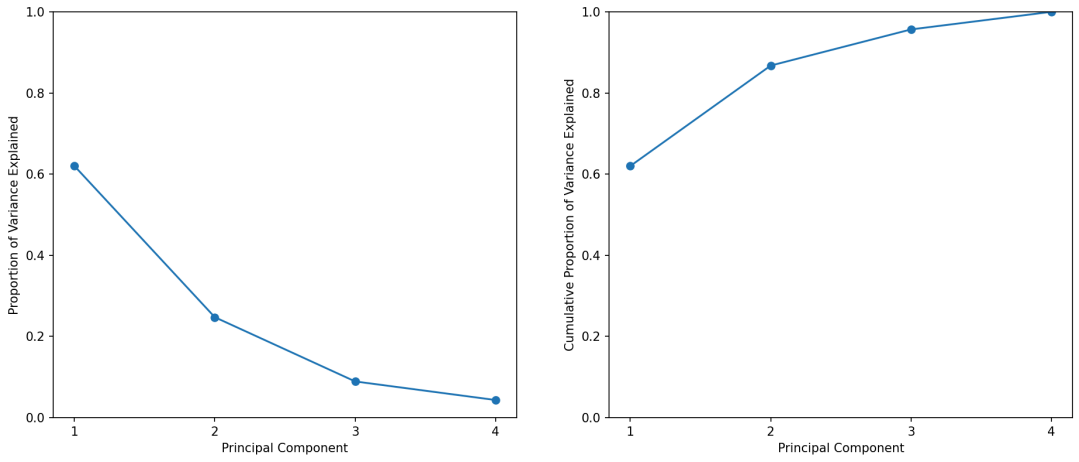
```
fig, axes = plt.subplots(1, 2, figsize=(15, 6))
ticks = np.arange(pcaUS.n_components_)+1
ax = axes[0]
ax.plot(ticks,
        pcaUS.explained_variance_ratio_,
        marker='o')
```

```
ax.set_xlabel('Principal Component');
ax.set_ylabel('Proportion of Variance Explained')
ax.set_ylim([0,1])
ax.set_xticks(ticks)
```

然后可以画累积PVE：

```
ax = axes[1]
ax.plot(ticks,
        pcaUS.explained_variance_ratio_.cumsum(), #累计和
        marker='o')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Cumulative Proportion of Variance Explained')
ax.set_ylim([0, 1])
ax.set_xticks(ticks)
```

这里的 `.cumsum()` 用于计算列表的累计和



12.5.2 矩阵补全

之前讨论过解决一个最优化问题以解出前M个主成分，一种算法是**奇异值分解(singular value decomposition, SVD)**，下面令标准化后的数据集为$\mathbf{X}$：

```
X = USArrests_scaled
U, D, V = np.linalg.svd(X, full_matrices=False)
print(U.shape, D.shape, V.shape) #输出原始矩阵和分解后的矩阵
```

函数 `np.linalg.svd()` 返回三个组分，即 `U, D, V` 。其中 `V` 等于前M个主成分的载荷矩阵，使用 `full_matrices=False` 保证tall矩阵 `U` 的形状和 `X` 一样

对比大小

```
print(V) #奇异值分解的V矩阵
```

```
print(pcaUS.components_) #pca的主成分矩阵
```

结果显示两个矩阵数值完全相同

矩阵`U`表示PCA得分矩阵的标准化版本。如果将它乘以原来的初始矩阵`D`，则可以恢复原始的PCA得分：

```
print((U * D[None ,:])[:3]) #恢复原始PCA得分
print(scores[:3]) #原始PCA得分
```

下面随机选取20行，对4个特征之一进行删除：

```
n_omit = 20
np.random.seed(15)
r_idx = np.random.choice(np.arange(X.shape[0]), #选择行标签
                         n_omit,
                         replace=False)
c_idx = np.random.choice(np.arange(X.shape[1]), #选择列标签
                         n_omit,
                         replace=True)
Xna = X.copy()
Xna[r_idx, c_idx] = np.nan
```

下面的函数实现了用PCA补全矩阵的算法，其中关键步骤是用`svd()`函数进行奇异值分解：

```
def low_rank(X, M=1): #M是一个默认参数, 默认值为1(可更改)
    U, D, V = np.linalg.svd(X)
    L = U[:,:M] * D[None,:M]
    return L.dot(V[:M])
```

这个函数是算法的第二步

`np.nanmean()`能够计算矩阵的均值(按行或列方向)。注意这里创建矩阵的副本，防止对原始矩阵直接进行修改：

```
Xhat = Xna.copy() #创建矩阵副本
Xbar = np.nanmean(Xhat, axis=0) #按列求矩阵
Xhat[r_idx, c_idx] = Xbar[c_idx] #对矩阵缺失值进行填充
```

开始第二步前，需要衡量迭代的进度：

```
thresh = 1e-7 #迭代误差阈值, 小于此值认为收敛
rel_err = 1 #初始误差, 1表示尚未开始迭代
count = 0 #迭代次数计数器
ismiss = np.isnan(Xna) #返回缺失值的索引矩阵
```

```
mssold = np.mean(Xhat[~ismiss]**2) #计算初始迭代矩阵非缺失值的平方均值
mss0 = np.mean(Xna[~ismiss]**2) #计算原始矩阵非缺失值的平方均值
```

这里希望算法持续迭代，直到`mssold - mss / mss0 < thresh`

在算法的2.1步中，我们用`low_rank()`估计`Xhat`，它被称为`Xapp`，在2.2步中，用`Xapp`更新在`Xna`中缺失位置的在`Xhat`中的估计值，在第2.3步中计算相对误差：

```
while rel_err > thresh:
    count += 1
    # Step 2(a)
    Xapp = low_rank(Xhat, M=1)
    # Step 2(b)
    Xhat[ismiss] = Xapp[ismiss]
    # Step 2(c)
    mss = np.mean(((Xna - Xapp)[~ismiss])**2)
    rel_err = (mssold - mss) / mss0
    mssold = mss
    print("Iteration: {0}, MSS:{1:.3f}, Rel.Err {2:.2e}"
          .format(count, mss, rel_err))
```

每次迭代输出的`MSS`表示非缺失值的均方误差

显示补充缺失值和原始值的相关性：

```
print(np.corrcoef(Xapp[ismiss], X[ismiss])[0,1])
```

12.5.3 聚类

K-means聚类

`sklearn.cluster.KMeans()`能够执行Python中的k-means聚类任务，下面使用模拟的缺失具有两类簇的数据来演示：

```
#生成随机数据
np.random.seed(0);
X = np.random.standard_normal((50,2));
X[:25,0] += 3;
X[:25,1] -= 4
```

执行聚类并查看聚类标签：

```
kmeans = KMeans(n_clusters=2,
                random_state=2, #随机选取点的随机数种子
                n_init=20).fit(X)
print(kmeans.labels_) #聚类标签
```

为聚类绘图，对点上色：

```
fig, ax = plt.subplots(1, 1, figsize=(8,8))
ax.scatter(X[:,0], X[:,1], c=kmeans.labels_)
ax.set_title("K-Means Clustering Results with K=2")
```

因为我们的数据是二维的，所以可以很容易绘制；对于高维数据，我们需要先用PCA获取前两个主成分再绘制二维图

实际上，我们常常不知道K的大小，例如我们设置$K=3$：

```
kmeans = KMeans(n_clusters=3,
                random_state=3,
                n_init=20).fit(X)
fig, ax = plt.subplots(figsize=(8,8))
ax.scatter(X[:,0], X[:,1], c=kmeans.labels_)
ax.set_title("K-Means Clustering Results with K=3")
```

也能得到聚类结果。一般情况下，`n_init`表示聚类次数，默认为10次，将会输出聚类次数中最优(簇最紧密)的结果

显示不同聚类次数得到的结果差异：

```
kmeans1 = KMeans(n_clusters=3,
                 random_state=3,
                 n_init=1).fit(X)
kmeans20 = KMeans(n_clusters=3,
                  random_state=3,
                  n_init=20).fit(X);
print(kmeans1.inertia_, kmeans20.inertia_)
```

这里的`kmeans_inertia`表示簇内距离平方和，我们希望k-means能减小这个值。指定`random_state`参数以获得可重复结果

> 作者建议重复次数在20次~50次左右

层次聚类

`sklearn.clustering`中的`AgglomerativeClustering()`提供了层次聚类，本节缩写为`HClust`。下面的示例中，不相似度度量为欧几里得距离，连接方式为“完全”：

```
HClust = AgglomerativeClustering
hc_comp = HClust(distance_threshold=0,
                 n_clusters=None,
                 linkage='complete')
hc_comp.fit(X)
```

然后使用单一和平均作为连接方式：

```
hc_avg = HClust(distance_threshold=0, #平均
                n_clusters=None,
                linkage='average');
hc_avg.fit(X)
hc_sing = HClust(distance_threshold=0, #单一
                 n_clusters=None,
                 linkage='single');
hc_sing.fit(X)
```

为了使用预先计算的距离矩阵，我们指定 `meetric="precomputed"`；下面代码的前四行计算一个50×50的对与对之间距离矩阵：

```
D = np.zeros((X.shape[0], X.shape[0])) #一个行×行大小的矩阵
for i in range(X.shape[0]): #对于矩阵中的每个元素(观测值)
    x_ = np.multiply.outer(np.ones(X.shape[0]), X[i]) #计算与其他所有点的欧氏距离平方
    D[i] = np.sqrt(np.sum((X - x_)**2, 1)) #存储到距离矩阵中
hc_sing_pre = HClust(distance_threshold=0, #仅合并距离为零的矩阵
                     n_clusters=None,
                     metric='precomputed',
                     linkage='single')
hc_sing_pre.fit(D)
```

对于上面的距离计算部分：

- `D` 是一个初始化的矩阵，用于存储点i与点j之间的欧氏距离
- `x_ = np.multiply.outer(np.ones(X.shape[0]), X[i])` 生成了一个矩阵，每行均为样本 `i` 的特征向量，使其与原始数据 `X` 形状相同
- `X - x_` 计算所有样本与样本 `i` 的差值。
- `np.sqrt(np.sum((X - x_)**2, 1))` 对差值平方求和后开平方，得到欧氏距离。
- 结果存入 `D[i]`，最终 `D` 为对称的距离矩阵，其中 `D[i][j]` 表示样本 `i` 与 `j` 的距离

下面使用 `scipy.cluster.hierarchy` 中的 `dendrogram()` 函数来绘制聚类树图。它需要连接矩阵表示在 `AgglomerativeClustering()` 中没有，但是我们可以计算。函数 `compute_linkage`，来自 `ISLP.cluster` 提供了帮助
`dendrogram()` 函数的一个默认设置是为所有不同的分支着色，我们一般设置为无限来覆盖掉这个设置；这个设置被存储在 `cargs` 字典中，将被传递到函数的附加参数中：

```
cargs = {'color_threshold':-np.inf, #剪切阈值为负无穷
         'above_threshold_color':'black'} #高于阈值的颜色设置为黑色
linkage_comp = compute_linkage(hc_comp) #ISLP中计算连矩阵的轮子
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
dendrogram(linkage_comp,
```

```
            ax=ax,
            **cargs)
```

如果我们希望对剪切不为以下的树进行着色，这时候应该更改 `color_threshold` 参数，下面设置剪切高度为4：

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
dendrogram(linkage_comp,
            ax=ax,
            color_threshold=4,
            above_threshold_color='black')
```

输出聚类标签：

```
print(cut_tree(linkage_comp, n_clusters=4).T)
```

这可以通过向 `HCLust()` 提供 `n_clusters` 参数实现，但是需要重新计算聚类；也可以通过 `HClust()` 的参数 `distance_threshold` 或者 `cut_tree()` 的 `height()` 参数实现：

```
print(cut_tree(linkage_comp , height =5))
```

为了在对观测值进行分层聚类前对变量进行标准化，需要使用 `StandardScaler()`：

```
scaler = StandardScaler()
X_scale = scaler.fit_transform(X) #标准化矩阵
hc_comp_scale = HClust(distance_threshold=0,
                       n_clusters=None,
                       linkage='complete').fit(X_scale)
linkage_comp_scale = compute_linkage(hc_comp_scale)
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
dendrogram(linkage_comp_scale, ax=ax, **cargs)
ax.set_title("Hierarchical Clustering with Scaled Features")
plt.show()
```

相关性可用于聚类，不相关度即可用一减去相关度来实现。注意使用相关性度量必须对至少三个特征的数据有意义，因为在平面中，两个点之间的线性关系必然是± 1：

```
X = np.random.standard_normal((30, 3))
corD = 1 - np.corrcoef(X)
hc_cor = HClust(linkage='complete',
                distance_threshold=0,
                n_clusters=None,
                metric='precomputed')
hc_cor.fit(corD)
linkage_cor = compute_linkage(hc_cor)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
dendrogram(linkage_cor, ax=ax, **cargs)
ax.set_title("Complete Linkage with Correlation-Based Dissimilarity")
```

12.5.4 NCI60数据集示例

无监督学习技术常用于基因组分析，下面以数据集 `NCI60` 为例，该数据集包括64种癌细胞系的6830个基因表达测量值：

```
#加载数据集
NCI60 = load_data('NCI60')
nci_labs = NCI60['labels']
nci_data = NCI60['data']
```

这里面每个细胞系都用一种癌症类型标记。后续进行PCA和聚类不使用这些标签，而是检查我们得到结果与标签的一致性

一些关于数据的检查：

```
print(nci_data.shape) #数据集形状64×6830
print(nci_labs.value_counts()) #各种标签计数
```

将变量标准化后进行PCA：

```
scaler = StandardScaler()
nci_scaled = scaler.fit_transform(nci_data)
nci_pca = PCA()
nci_scores = nci_pca.fit_transform(nci_scaled)
```

> 这里有理由不标准化，因为测量尺度一致

绘制前几个主成分分数向量以可视化数据。对于确定类型癌症的观察值以相同颜色绘制，我们可观察到癌症类型中的观测值在多大程度上彼此相似：

```
cancer_types = list(np.unique(nci_labs)) #癌症种类标签
nci_groups = np.array([cancer_types.index(lab) #将每个样本癌症类型标签转换为对应的索引值
                       for lab in nci_labs.values])
fig, axes = plt.subplots(1, 2, figsize=(15,6))
ax = axes[0]
ax.scatter(nci_scores[:,0], #绘制关于1, 2主成分的二维图
           nci_scores[:,1],
           c=nci_groups,
           marker='o',
           s=50)
ax.set_xlabel('PC1'); ax.set_ylabel('PC2')
```

```
ax = axes[1]
ax.scatter(nci_scores[:,0], #绘制关于1, 3主成分的二维图
           nci_scores[:,2],
           c=nci_groups,
           marker='o',
           s=50)
ax.set_xlabel('PC1'); ax.set_ylabel('PC3')
```

图像整体上显示对于单一癌症类型的细胞系，在前几个主成分分数向量上具有相似的值，表明同一癌症类型细胞系往往具有相似的基因表达水平

绘制主成分解释的百分比方差和累积方差：

```
fig, axes = plt.subplots(1, 2, figsize=(15,6))
ax = axes[0]
ticks = np.arange(nci_pca.n_components_)+1
ax.plot(ticks,
        nci_pca.explained_variance_ratio_,
        marker='o')
ax.set_xlabel('Principal Component');
ax.set_ylabel('PVE')
ax = axes[1]
ax.plot(ticks,
        nci_pca.explained_variance_ratio_.cumsum(),
        marker='o');
ax.set_xlabel('Principal Component')
ax.set_ylabel('Cumulative PVE')
```

下面使用完全、单一和平均连接来进行分层聚类，不相似度度量为欧氏距离。编写一个函数，生成三个聚类树图：

```
def plot_nci(linkage, ax, cut=-np.inf): #参数包括连接方式, 绘制图像的图片, 剪切位置
    cargs = {'above_threshold_color':'black',
             'color_threshold':cut} #聚类树图默认字典
    hc = HClust(n_clusters=None,
                distance_threshold=0,
                linkage=linkage.lower()).fit(nci_scaled)
    linkage_ = compute_linkage(hc)
    dendrogram(linkage_,
               ax=ax,
               labels=np.asarray(nci_labs),
               leaf_font_size=10,
               **cargs)
    ax.set_title('%s Linkage' % linkage)
    return hc
```

使用函数绘制结果：

```
fig, axes = plt.subplots(3, 1, figsize=(15,30))
ax = axes[0]; hc_comp = plot_nci('Complete', ax)
ax = axes[1]; hc_avg = plot_nci('Average', ax)
ax = axes[2]; hc_sing = plot_nci('Single', ax)
```

可以看到不同的连接方式会产生不同的聚类树图。单一连接会产生拖尾效应，即很大的数量的树枝联系到一起。因此，平均和完全产生的图像通常优于单一

基于完整的连接方式，我们产生一个4个簇的高度切割：

```
linkage_comp = compute_linkage(hc_comp)
comp_cut = cut_tree(linkage_comp, n_clusters=4).reshape(-1)
pd.crosstab(nci_labs['label'],
            pd.Series(comp_cut.reshape(-1), name='Complete'))
```

这将生成一个表格，表明标签和它们被分类到簇的数量。容易发现，所有白血病细胞系都属于一个簇，而乳腺癌细胞系分布在三个不同的簇中。

可以在树状图上绘制我们切割的部位，这用到`axhline()`方法，它将会在某个高度绘制水平线：

```
fig, ax = plt.subplots(figsize=(10,10))
plot_nci('Complete', ax, cut=140)
ax.axhline(140, c='r', linewidth=4)
```

使用分层聚类得到的聚类簇与K-means不同：

```
nci_kmeans = KMeans(n_clusters=4,
                    random_state=0,
                    n_init=20).fit(nci_scaled)
pd.crosstab(pd.Series(comp_cut, name='HClust'),
            pd.Series(nci_kmeans.labels_, name='K-means'))
```

实际上，与其对所有的数据进行分层聚类，在前几个主成分上进行聚类可以看作是噪声较小的数据版本：

```
hc_pca = HClust(n_clusters=None,
                distance_threshold=0,
                linkage='complete'
                ).fit(nci_scores[:,:5])
linkage_pca = compute_linkage(hc_pca)
fig, ax = plt.subplots(figsize=(8,8))
dendrogram(linkage_pca,
```

```
           labels=np.asarray(nci_labs),
           leaf_font_size=10,
           ax=ax,
           **cargs)
ax.set_title("Hier. Clust. on First Five Score Vectors")
pca_labels = pd.Series(cut_tree(linkage_pca,
                                n_clusters=4).reshape(-1),
                       name='Complete-PCA')
pd.crosstab(nci_labs['label'], pca_labels)
```