第 4 章　维数灾难与降维

维数与个数是数据的两个基本指标

维数是数学上的一个基本概念，随着维数的增加，会出现新奇的现象、会带来几何想象的困难、会带来计算上的根本困难。对于机器学习模型而言，快速增加的维数将导致“维数灾难”，从而使得模型不再可行。因此，降维是机器学习的基本任务之一。

按照所采用的变换类型的不同，可以将降维方法划分为线性降维和非线性降维两大类。本章只涉及线性降维方法，将详细介绍主成分分析和奇异值分解两种经典方法。在此基础上，读者可以进一步探索更多的线性降维方法（比如多维尺度分析）和各种非线性降维方法（比如等距特征映射、局部线性嵌套、tSNE 等）。

4.1　基本概念

几何上，维数的概念是很直观的。如图 4.1 所示，一个点可视为 0 维，一个点向右运动所形成的线段就是 1 维，而一条线段向下移动所形成的正方形就是 2 维，依次类推，可以得到 3 维的立方体、4 维的超立方体等。把这个过程反过来，那么 n 维空间中的边界就是 n-1 维，据此可以递归地确定维数。

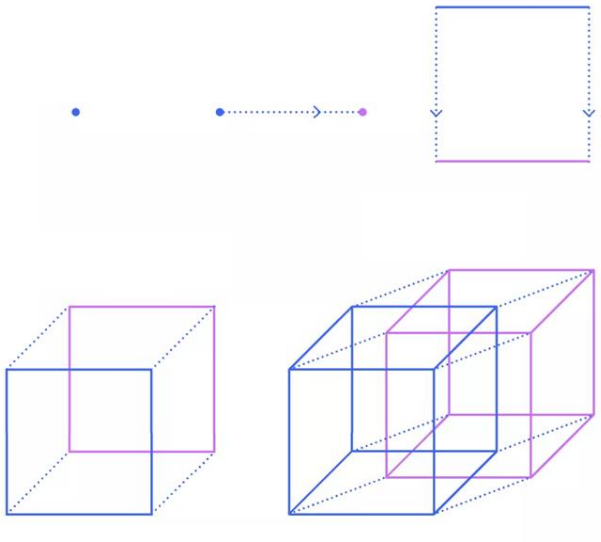

图 4.1　递归确定维数

随着维数的增加，会带来什么问题呢？如图 4.2 所示，想象要把边长为 3 的空间填满，则维数 D=1 时，只需要 $3=3^1$ 个单元就解决问题；当 D=2 时，需要的单元数增加到 $3=3^2$；而当 D=3 时，需要的单元数增加到 $3=3^3$。可见，维数的增加将导致需要的单元数呈现指数增长，对于机器学习而言，这就意味着需要的数据样本将随着维数的增加而呈指数增长。这就是所谓的“维数灾难”。实际的机器学习任务一般都具有较高的维数（比如 1 幅 500 万像素的图片，其维数为 500 万维），因此降维是机器学习的基本任务之一。

降维可以带来如下一些好处：压缩与简化数据，降低计算复杂度；减小或消除数据中的噪声；帮助理解数据，比如找出重要的数据。另外，为了实现数据的可视化，也常常需要将

高维数据降维到 3 维或更低。值得特别注意的是，降维的过程可能会导致丢失有用信息，因此如何减少甚至避免这一点，是需要仔细考虑和权衡的。

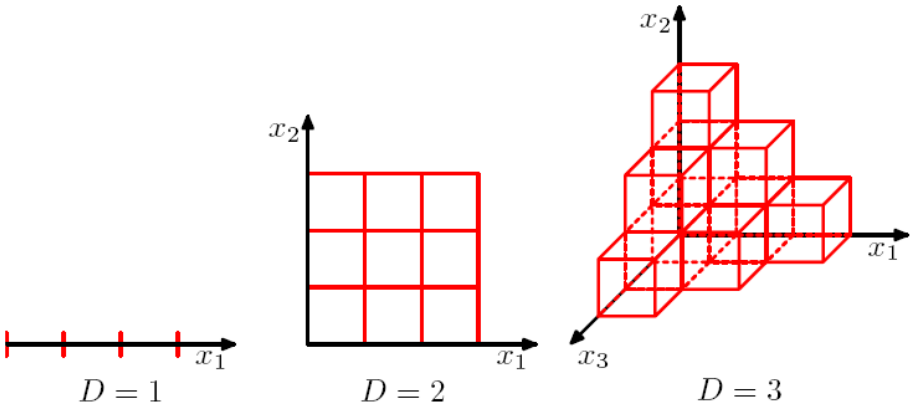

图 4.2 维数灾难

下面举两个例子。以图 2.5.2 为例，这个三层的全连接神经网络从降维的角度来看，就是将原始 784 维（28x28 灰度图像）的输入数据先通过隐藏层降到 15 维，再通过输出层进一步降到 10 维（对应 10 个类别）。实际上，神经网络的根本优势之一就在于通过端对端的方式学习降维。

世上的事物都有两面性，那么维数增加是否可能带来好处呢？回答是肯定的，2.4 节介绍的 SVM 就是这样一个例子：特征变换函数$\boldsymbol{\varphi(x)}$可以将低维空间中线性不可分的数据$\boldsymbol{x}$升维为高维空间中线性可分的数据$\boldsymbol{\varphi(x)}$。这也进一步启发我们，在设计神经网络时，可以把“升维”也考虑进去[1]。

4.2 主成分分析

回顾线性代数里讲到的特征分解$\boldsymbol{Ax}=\lambda\boldsymbol{x}$，主成分分析（Principal Component Analysis）其实就是实对称阵$\boldsymbol{A}$的特征分解。所谓主成分（PC），就是指对应较大特征值λ的特征向量$\boldsymbol{x}$。

具体来讲，设$\boldsymbol{A}$为$F\times F$实对称阵（元素都为实数的对称方阵），则$\boldsymbol{A}$有实且正交的特征分解，即特征值λ都为实数，对应λ的实特征向量$\boldsymbol{x}$相互之间正交。这样，$\boldsymbol{A}$通过特征分解就能够得到K个（$K\leq F$）对应较大特征值的特征向量，这些特征向量反映了$\boldsymbol{A}$中较重要的信息（所以称为主成分）。如果$K<F$，就达到了降维的目的；如果$K=F$，就仅仅是进行了投影和坐标旋转（从而与特征向量对齐）。

那么，为什么说对应较大特征值的特征向量就更重要呢？这要从降维期望达到的目标说起。一般而言，降维的目标是：保留重要的关系，同时减少信息损失、保持数据区分度。那么，数据中存在一些什么关系？哪些关系是数据中重要的关系？如何度量信息的损失？什么又是数据的区分度？如何保持数据的区分度？

不同的降维方法看待和处理这些问题的角度往往有所不同。PCA 试图保持数据的一些重要维度的信息，这些维度相互是正交的关系，其衡量重要性的准则是特定维度数据的方差，而对于数据的区分度也是通过不同维度的方差不同来刻画。因为，一般来讲，数据的方差越大，其信息熵也越大（见式（2.2.1）），有用的数据往往具有较大的方差，而噪声则具有较小的方差。因此，PCA 保持具有较大方差的维度，能够起到保留有用信息、降低噪声的作用。用线性代数的术语来讲，PCA 定义了一个正交变换（正交投影），通过这个变换将高维多元数据投影到一个低维的坐标系统，并使得原始数据的第一大方差投影到第一个坐标上，第二大方差投影到第二个坐标上，以此类推。换句话说，PCA 将原始数据投影到了一个能更好刻

注 1：实际上，先降维（编码）再升维（解码）也是最常见且应用最广泛的神经网络之一。

画数据特征的正交坐标系中。

如图 4.3 所示，原始 2 维笛卡儿坐标系中的数据点$\boldsymbol{x}_i = (x_1^{(i)}, x_2^{(i)})^T (i = 1, \ldots, N)$通过投影到其最大方差方向$\boldsymbol{u}_1$而被降为 1 维，这个降维过程保持了重要的信息——方差。如果进一步考虑与$\boldsymbol{u}_1$正交的方向$\boldsymbol{u}_2$，$\boldsymbol{u}_2$就对应第二大方差方向，数据投影到$\boldsymbol{u}_2$就同样保持了第二大方差信息。当然，如果把$\boldsymbol{u}_1$和$\boldsymbol{u}_2$两个方向的投影信息都保持下来，就不再有降维的作用了，而仅仅只是投影加上坐标旋转——将原始坐标轴与两个主成分（$\boldsymbol{u}_1$和$\boldsymbol{u}_2$）对齐。

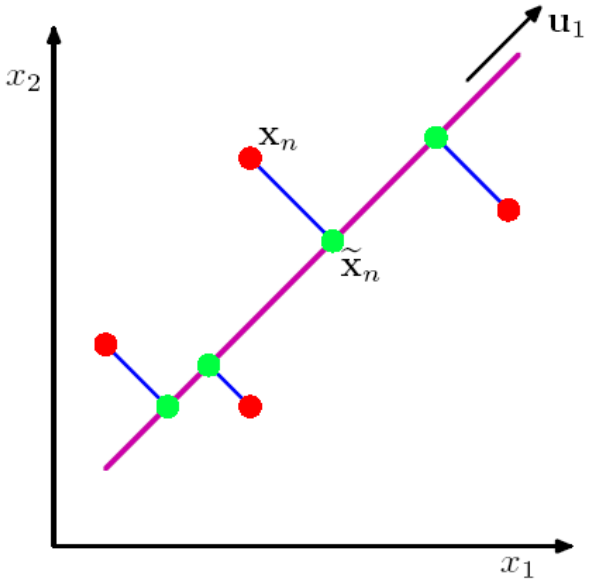

图 4.3　PCA

4.2.1　最大化投影方差推导

明白了 PCA 的降维思想，接下来就可以运用线性代数知识来详细地推导整个过程。设样本点$\boldsymbol{x}_i (i = 1, \ldots, N)$为$F$维向量，定义一个$F$维单位方向向量$\boldsymbol{u}_1$，从而有$\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$。$\boldsymbol{x}_i$在$\boldsymbol{u}_1$上的投影为一个标量值$\boldsymbol{u}_1^T \boldsymbol{x}_i$。

首先得到所有样本点的均值向量：

$$\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \tag{4.1}$$

则投影样本点的方差为：

$$\frac{1}{N} \sum_{i=1}^{N} \{\boldsymbol{u}_1^T \boldsymbol{x}_i - \boldsymbol{u}_1^T \overline{\boldsymbol{x}}\}^2 = \boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 \tag{4.2}$$

其中，$\{\boldsymbol{u}_1^T \boldsymbol{x}_i - \boldsymbol{u}_1^T \overline{\boldsymbol{x}}\}^2 = \boldsymbol{u}_1^T (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{u}_1^T (\boldsymbol{x}_i - \overline{\boldsymbol{x}}))^T = \boldsymbol{u}_1^T (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T \boldsymbol{u}_1$，$\boldsymbol{S}$是协方差矩阵（$F \times F$实对称阵）：

$$\boldsymbol{S} = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T \tag{4.3}$$

我们的目标是相对于$\boldsymbol{u}_1$最大化投影数据的方差$\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$，且满足约束$\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$。由此，我们引入 Lagrange 乘子$\lambda_1$并定义一个无约束最大化问题：

$$\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1) \tag{4.4}$$

式（4.4）相对于$\boldsymbol{u}_1$求导并将其置为 0 得到：

$$\boldsymbol{u}_1^T \boldsymbol{S} - \lambda_1 \boldsymbol{u}_1^T = 0 => \boldsymbol{u}_1^T \boldsymbol{S} = \lambda_1 \boldsymbol{u}_1^T => \boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1 \tag{4.5}$$

$$\boldsymbol{S} \boldsymbol{u}_1 - \lambda_1 \boldsymbol{u}_1 = 0 => \boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1 \tag{4.5'}$$

式（4.5）和（4.5）’是一个东西，只不过为了说明向量求导我们特意写了两遍，意在说

明对于求导$\boldsymbol{u}_1^T$与$\boldsymbol{u}_1$并无区别，最后的结果$\boldsymbol{u}_1^T\boldsymbol{S}=\lambda_1\boldsymbol{u}_1^T$与$\boldsymbol{S}\boldsymbol{u}_1=\lambda_1\boldsymbol{u}_1$并无区别，因为$\boldsymbol{S}$是对称阵，一个式子两边同时转置即得到另一个式子，两者定义的方程组完全一致。

（4.5）式表明$\boldsymbol{u}_1$是$\boldsymbol{S}$的一个特征向量。（4.5）式两边左乘$\boldsymbol{u}_1^T$并注意到$\boldsymbol{u}_1^T\boldsymbol{u}_1=1$，我们得到方差$\boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1=\lambda_1$，这意味着只要我们选取对应最大特征值$\lambda_1$的那个特征向量$\boldsymbol{u}_1$就能保证投影后的数据具有最大方差。问题得到完美解决。

上面我们考虑的是$K=1$的情况。对于$K>1$的一般情况，显然，第二个主分量就是对应第二大特征值的那个特征向量，第三个主分量就是对应第三大特征值的那个特征向量，依此类推。由于协方差矩阵$\boldsymbol{S}$是$F\times F$维实对称阵，故其有F个实特征值，分别对应F个两两正交的特征向量，这些两两正交的特征向量构成一个正交坐标系统。PCA 的实质就是将原始数据投影到这个能更好刻画数据特征的正交坐标系中。当然，如果我们取$K=F$，就不再有降维的效果，而只是简单的坐标旋转，这一点我们前面在说明图 4.3 的时候已经提到过。

还要强调一点，协方差矩阵$\boldsymbol{S}$的表达式（见式（4.3））中，涉及到每个数据向量$\boldsymbol{x}_i$减去均值向量$\bar{\boldsymbol{x}}$从而得到 0 均值数据向量（方差也称为二阶中心矩，这个中心就是指的均值，减掉均值就得到 0 中心，这也是一种常见的标准化手段）。注意，均值向量$\bar{\boldsymbol{x}}$是所有数据向量$\boldsymbol{x}_i(i=1,\dots,N)$的均值（见式（4.1））！很清楚，对吧。但是在实践中，还是有很多细节容易犯错。比如设$\boldsymbol{x}_i$是 224×224×3 的 RGB 彩色图像，那么均值向量$\bar{\boldsymbol{x}}$应该如何考虑呢？所有$\boldsymbol{x}_i$的所有像素得到一个均值标量？所有$\boldsymbol{x}_i$的对应通道的像素得到一个均值标量，从而形成一个具有三个分量的均值向量？还是所有$\boldsymbol{x}_i$的对应通道的对应像素得到一个均值标量，从而形成一个具有 224×224×3 个分量的均值向量？应该说最后这一种才是跟均值向量的定义完全一致的（数据向量与均值向量具有相同的维数）。但是实践中，为了方便更多时候采用的是第二种，即每个通道计算一个均值。与此相关的一种常见的错误是，将数据向量$\boldsymbol{x}_i$减去其自身每个通道的均值，这是完全违背定义的，需要特别引起注意！均值作为最基本的一个统计量，一定是针对所有数据样本而言的。由这个例子，大家应该能体会到工程实现上的细节恰如绣女绣花，不能有半点马虎，要做到精益求精就更是要斟酌每一个细节，不断改进，不断完善。

4.2.2 最小化投影误差推导

PCA 的推导还有另一种从本质上讲完全等价的观点——最小化投影后的误差。如图 4.3 中到$\boldsymbol{u}_1$的投影线段所示，PCA 实际上也是在最小化原始数据与投影数据之间的均方差（MSE）。

为此，首先引入一个完备标准正交基向量集合$\{\boldsymbol{u}_i,i=1,\dots,F\}$，其中每个$\boldsymbol{u}_i$为$F$维向量。由于两两标准正交，所以有：

$$\boldsymbol{u}_i^T\boldsymbol{u}_j=\delta_{ij} \tag{4.6}$$

其中，δ_{ij}是指示函数，即$i=j$时，值为 1，反之则为 0。

又由于是完备基，那么每一个数据点$\boldsymbol{x}_i$都可以表示为基向量$\boldsymbol{u}_j$的线性组合：

$$\boldsymbol{x}_i=\sum_{j=1}^{F}a_{ij}\,\boldsymbol{u}_j \tag{4.7}$$

由此，$\boldsymbol{x}_i$的F个分量$\{x_1^{(i)},\dots,x_F^{(i)}\}$被替换为一个等价的集合$\{a_{i1},\dots,a_{iF}\}$。$\boldsymbol{x}_i$的转置与$\boldsymbol{u}_j$求内积，并使用（4.6）式即可得到$a_{ij}=\boldsymbol{x}_i^T\boldsymbol{u}_j$，将其代入（4.7）式就得到：

$$\boldsymbol{x}_i=\sum_{j=1}^{F}(\boldsymbol{x}_i^T\boldsymbol{u}_j)\boldsymbol{u}_j \tag{4.8}$$

我们的目标是用$K<F$个基向量得到的表示来逼近$\boldsymbol{x}_i$。这K个基向量是原始F个基向量在一个低维子空间上的投影。由此，$\boldsymbol{x}_i$的逼近可表示为：

$$\tilde{\boldsymbol{x}}_i=\sum_{j=1}^{K}z_{ij}\boldsymbol{u}_j+\sum_{j=K+1}^{F}b_j\boldsymbol{u}_j \tag{4.9}$$

其中$\{z_{ij}\}$依赖于特定的数据点$\boldsymbol{x}_i$，而b_j是对所有数据点都相同的常量。采用 MSE 来度量$\boldsymbol{x}_i$与$\tilde{\boldsymbol{x}}_i$之间的误差，目标是最小化这个误差：

$$J = \frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{x}_i - \widetilde{\boldsymbol{x}}_i\|^2 \tag{4.10}$$

首先考虑相对于$\{z_{ij}\}$来最小化J。将（4.8）和（4.9）式代入（4.10）式，并利用（4.6）式，则有：

$$\begin{aligned}&\|\boldsymbol{x}_i - \widetilde{\boldsymbol{x}}_i\|^2\\ &= \left\|\sum_{j=1}^{F}(\boldsymbol{x}_i^T\boldsymbol{u}_j)\boldsymbol{u}_j - \sum_{j=1}^{K} z_{ij}\boldsymbol{u}_j - \sum_{j=K+1}^{F} b_j\boldsymbol{u}_j\right\|^2\\ &= \sum_{j=1}^{K}(\boldsymbol{x}_i^T\boldsymbol{u}_j - z_{ij})^2 + \sum_{j=K+1}^{F}(\boldsymbol{x}_i^T\boldsymbol{u}_j - b_j)^2\end{aligned} \tag{4.11}$$

注意：$\|.\|$表示 L2 范数，详见式（3.7）。

由此，J对z_{ij}求导并置为 0，得到：

$$\frac{dJ}{dz_{ij}} = -2\left(\boldsymbol{x}_i^T\boldsymbol{u}_j - z_{ij}\right) = 0 \Rightarrow z_{ij} = \boldsymbol{x}_i^T\boldsymbol{u}_j \tag{4.12}$$

其中$j = 1, \dots, K$。

类似地，J可以对b_j求导并置为 0，利用（4.6）式，可得到：

$$\frac{dJ}{db_j} = -2\left(\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i^T\boldsymbol{u}_j - b_j\right) = 0 \Rightarrow b_j = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i^T\boldsymbol{u}_j = \overline{\boldsymbol{x}}^T\boldsymbol{u}_j \tag{4.13}$$

注意：对b_j求导需要对所有数据点进行累加。此处$j = K+1, \dots, F$。

将（4.12）、（4.13）式代入（4.9）式，可得：

$$\begin{aligned}\widetilde{\boldsymbol{x}}_i &= \sum_{j=1}^{K}(\boldsymbol{x}_i^T\boldsymbol{u}_j)\boldsymbol{u}_j + \sum_{j=K+1}^{F}(\overline{\boldsymbol{x}}^T\boldsymbol{u}_j)\boldsymbol{u}_j\\ &= \sum_{j=1}^{F}(\overline{\boldsymbol{x}}^T\boldsymbol{u}_j)\boldsymbol{u}_j + \sum_{j=1}^{K}(\boldsymbol{x}_i^T\boldsymbol{u}_j - \overline{\boldsymbol{x}}^T\boldsymbol{u}_j)\boldsymbol{u}_j\\ &= \overline{\boldsymbol{x}} + \sum_{j=1}^{K}(\boldsymbol{x}_i^T\boldsymbol{u}_j - \overline{\boldsymbol{x}}^T\boldsymbol{u}_j)\boldsymbol{u}_j\end{aligned} \tag{4.14}$$

这个式子表明，$\boldsymbol{x}_i$的逼近值$\widetilde{\boldsymbol{x}}_i$是$\boldsymbol{x}_i$的一个$K$维压缩表示（只需存储$K$个系数$(\boldsymbol{x}_i^T\boldsymbol{u}_j - \overline{\boldsymbol{x}}^T\boldsymbol{u}_j)$），$K$值越小，压缩率越大。

由（4.8）、（4.9）、（4.12）、（4.13）式易得：

$$\boldsymbol{x}_i - \widetilde{\boldsymbol{x}}_i = \sum_{j=K+1}^{F}\{(\boldsymbol{x}_i^T - \overline{\boldsymbol{x}}^T)\boldsymbol{u}_j\}\boldsymbol{u}_j \tag{4.15}$$

由上式可见，从$\boldsymbol{x}_i$到$\widetilde{\boldsymbol{x}}_i$的位移向量$\boldsymbol{x}_i - \widetilde{\boldsymbol{x}}_i$是$\{\boldsymbol{u}_j, j = K+1, \dots, F\}$的线性组合（注意上式中$(\boldsymbol{x}_i^T - \overline{\boldsymbol{x}}^T)\boldsymbol{u}_j$是标量），而$\{\boldsymbol{u}_j, j = K+1, \dots, F\}$所定义的空间是与$\{\boldsymbol{u}_j, j = 1, \dots, K\}$所定义的空间（称为主子空间）正交的空间，因此位移向量$\boldsymbol{x}_i - \widetilde{\boldsymbol{x}}_i$所在的空间与主子空间正交。比如图 4.3 中的$\boldsymbol{u}_1$对应主子空间，而与$\boldsymbol{u}_1$正交的投影线段就是位移向量所在的空间。直观上看，到$\boldsymbol{u}_1$的正交投影能够得到最小误差。

将（4.15）式代入（4.10）式，易得到：

$$\begin{aligned}J &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=K+1}^{F}\left(\boldsymbol{x}_i^T\boldsymbol{u}_j - \overline{\boldsymbol{x}}^T\boldsymbol{u}_j\right)^2\\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=K+1}^{F}\left\{(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T\boldsymbol{u}_j\right\}^2\end{aligned}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=K+1}^{F} \boldsymbol{u}_j^T(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T\boldsymbol{u}_j$$

$$= \sum_{j=K+1}^{F} \boldsymbol{u}_j^T\left\{\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T\right\}\boldsymbol{u}_j$$

$$= \sum_{j=K+1}^{F} \boldsymbol{u}_j^T\boldsymbol{S}\boldsymbol{u}_j \tag{4.16}$$

这个推导结果意味着，投影误差J就是位移向量所在$F-K$维空间（该空间与K维主子空间正交）各维方差的和。这说明，要使得投影误差J最小，就等价于K维主子空间上投影方差最大。

因此，类似（4.4）、（4.5），可得到：

$$\boldsymbol{S}\boldsymbol{u}_j = \lambda_j\boldsymbol{u}_j \tag{4.17}$$

于是J的值即为：

$$J = \sum_{j=K+1}^{F}\lambda_j \tag{4.18}$$

也就是与主子空间正交的那些特征向量所对应特征值的和，这些特征值就是$F-K$个较小的特征值。

4.2.3 核心代码实现

为了后面代码测试的方便，先用 Numpy 的 random.standard_normal()函数生成 10 个 2 维数据点，如图 4.4 所示。(为了看得更清楚，只生成 10 个点)

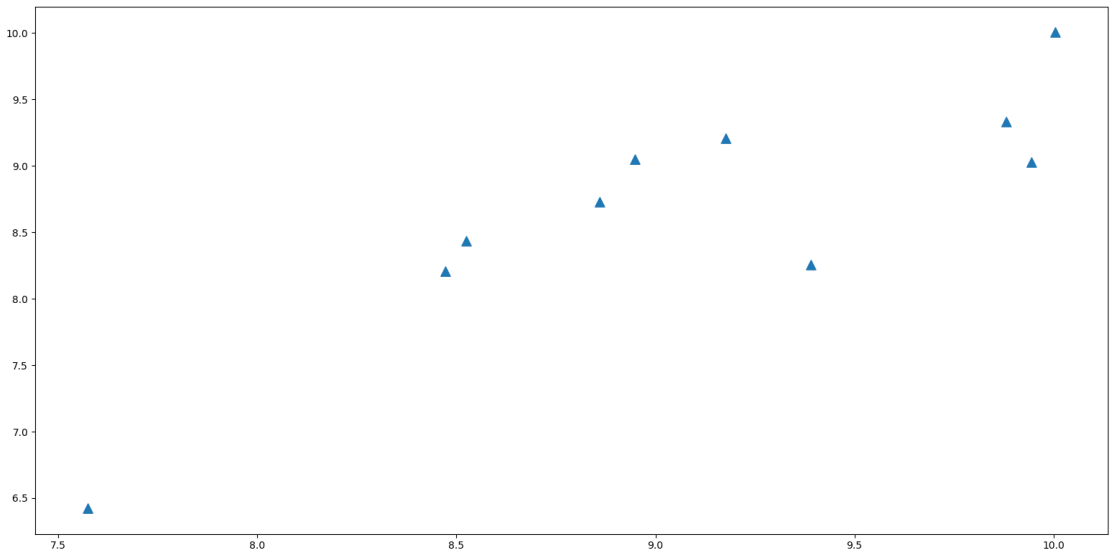

图 4.4 一个随机生成的 2 维简单数据集

接下来，定义函数 loadDataSet()用来从生成的数据文件中读入数据。

```
def loadDataSet(fileName, delim='\t'):
    with open(fileName) as fr:
        strList = [line.strip().split(delim) for line in fr.readlines()]
    datList = [list(map(float, line)) for line in strList]
    return np.mat(datList)
```

这段代码用到了 map()函数完成从字符串类型到浮点类型的转换，最后返回 Numpy 的矩阵类型。

```
def pca(dataMat, topNfeat=1):
```

```
    meanVals = np.mean(dataMat, axis=0) #列对应轴 0，即属性
    meanRemoved = dataMat - meanVals #减均值
    covMat = np.cov(meanRemoved, rowvar=0) #指定 rowvar=0，即列为属性
    eigVals, eigVects = np.linalg.eig(np.mat(covMat)) #特征向量 eigVects 是规范化的（即单位长度）
    eigValInd = np.argsort(eigVals)              #特征值从小到大排序
    eigValInd = eigValInd[:-(topNfeat+1):-1]   #得到最大的 topNfeat 个特征值的索引
    redEigVects = eigVects[:, eigValInd]         #得到按特征值从大到小排列的对应特征向量
    lowDDataMat = meanRemoved * redEigVects #得到降维后的数据
    reconMat = (lowDDataMat * redEigVects.T) + meanVals #得到重构数据
    return lowDDataMat, reconMat
```

函数 pca()完成数据矩阵 dataMat 的 PCA 变换，默认参数 topNfeat=1 指定变换后的数据维数。

（1）首先计算 dataMat 的均值 meanVals。注意列对应特征，所以需要指定 axis=0（列对应轴 0）。

（2）然后 dataMat 减掉均值得到矩阵 meanRemoved。

（3）调用 Numpy 的函数 cov()得到协方差矩阵 covMat。同样，需要指定 rowvar=0（列为特征）。

（4）调用 Numpy 的线性代数库函数 linalg.eig ()得到 covMat 的特征分解：特征值 eigVals 和特征向量 eigVects。注意 eigVects 的第 i 列特征向量对应 eigVals 的第 i 个特征值。

（5）接下来的三行代码根据特征值的大小完成对特征向量的间接排序。由于 Numpy 函数 argsort()是从小到大排序，所以采用负索引 eigValInd[:-(topNfeat+1):-1]得到最大 topNfeat 个特征值的索引值，然后通过 eigVects[:, eigValInd]就得到了对应的 topNfeat 个特征向量。

（6）接下来的两行代码就是套公式$\left(\boldsymbol{x}_i^T-\bar{\boldsymbol{x}}^T\right)\boldsymbol{u}_j$和式（4.14）分别得到降维后的数据 lowDDataMat 和重构数据 reconMat（即式（4.14）中的逼近值$\tilde{\boldsymbol{x}}_i$）。

用图 4.4 的 10 个 2 维数据点来实际验证一下。

```
lowDMat, reconMat = pca(dataMat, 1)
```

将 topNfeat 指定为 1 调用 pca()，并用 Matplotlib 的 scatter()函数将 dataMat、lowDMat 和 reconMat 都可视化出来，结果如图 4.5 所示。其中，三角形的点表示数据矩阵 dataMat，跟三角形点挨着的圆点表示重构数据 reconMat，而左下角的圆点就是降维后的数据 lowDDataMat。正如所期望的，2 维的 dataMat 降维后得到 1 维的 lowDDataMat，而且其跟 x 轴对齐。reconMat 是基于 lowDDataMat、对应最大特征值的特征向量 redEigVects.T 和均值 meanVals 重构出来的 2 维数据。可以看到，redEigVects.T 方向（图中带箭头的直线所示）是 dataMat 的最大方差方向，也是最小投影误差方向。reconMat 相对于 dataMat 所丢失的信息主要是与 redEigVects.T 垂直的方向，这应证了式（4.15）。

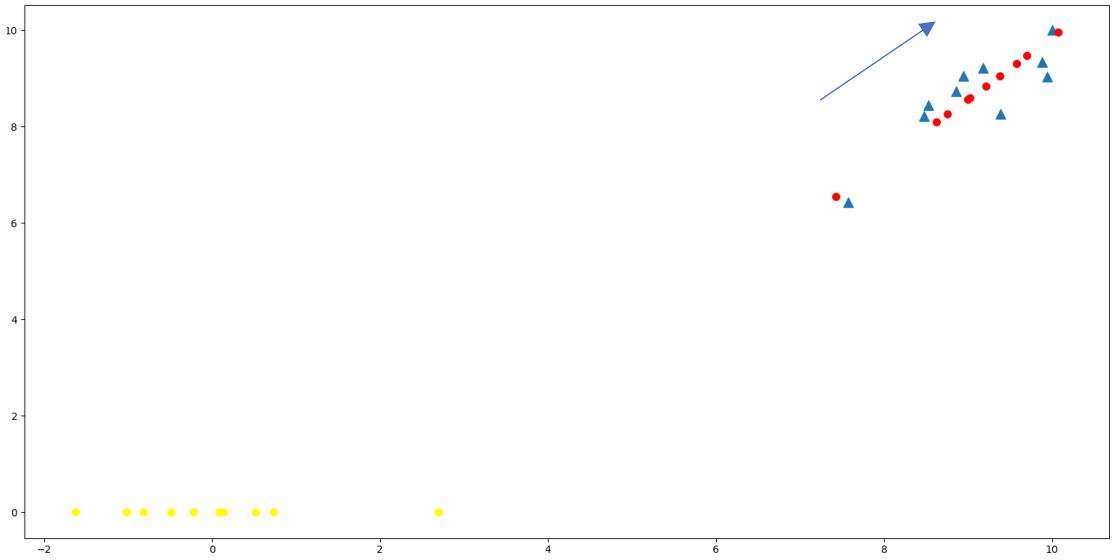

图 4.5 降到 1 维的可视化结果

如果不降维又是什么效果呢？

lowDMat, reconMat = pca(dataMat, 2)

图 4.6 给出了对应的可视化结果。由该图可见，reconMat 和 dataMat 完全重合。lowDDataMat 没有损失什么信息，仅仅只是作了些几何变换（习题 4.6）而已。

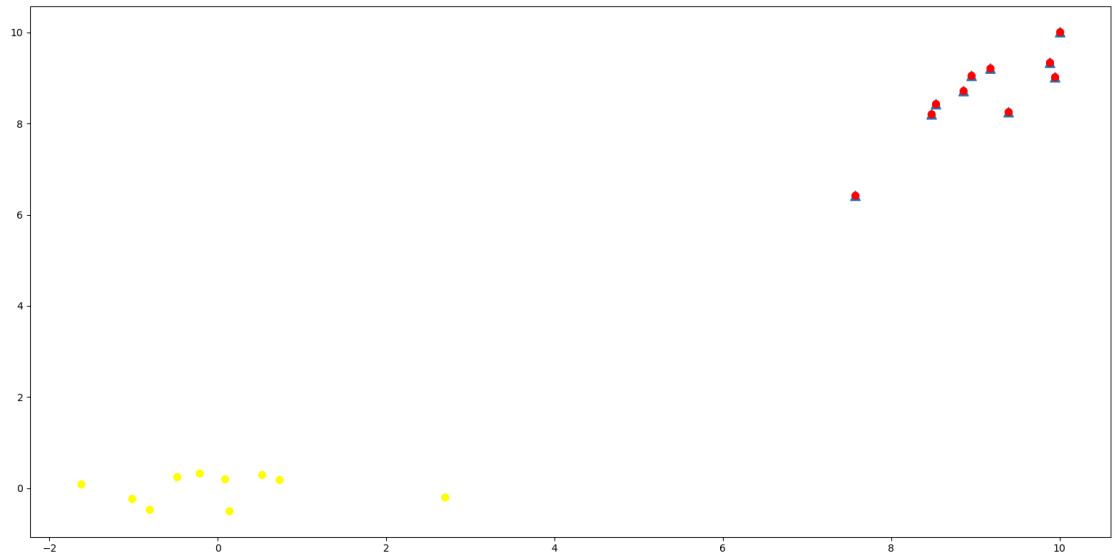

图 4.6 不降维的可视化结果

最后，生成 1000 个 2 维数据点，并类似图 4.5 和图 4.6 给出可视化结果，如图 4.7 和图 4.8 所示。

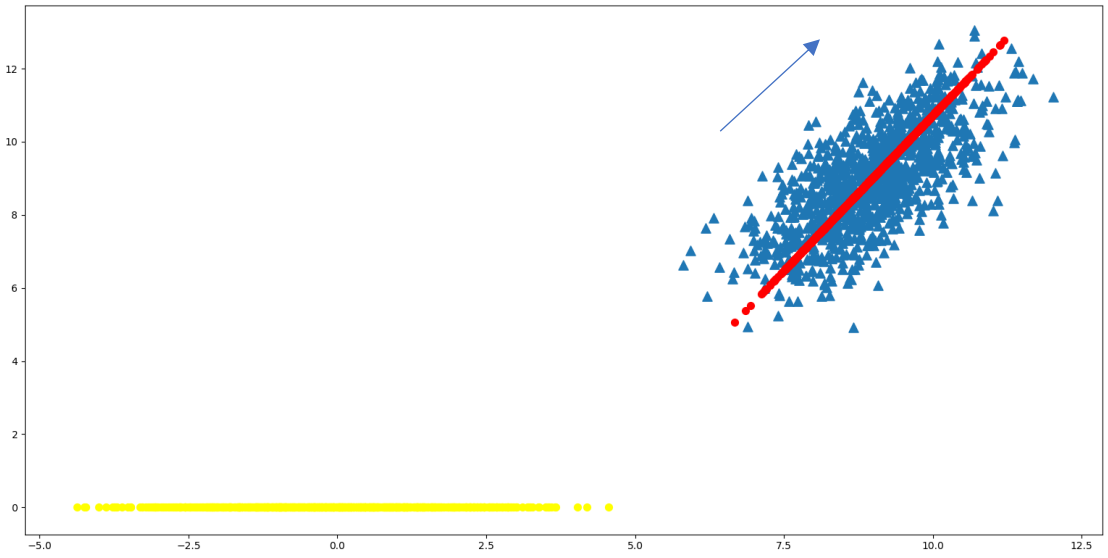

图 4.7 降到 1 维的可视化结果

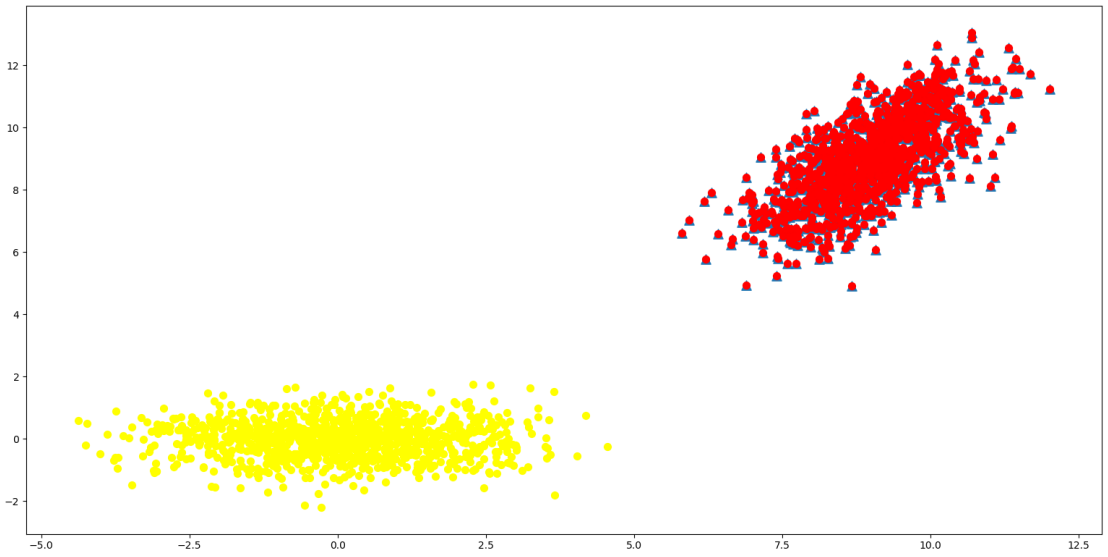

图 4.8 不降维的可视化结果

4.3 奇异值分解

上一节介绍了主成分分析，本节将接着介绍另一个常用的降维方法：奇异值分解(Singular value decomposition, SVD)。SVD 在许多领域都有广泛的应用，包括图像压缩、数据降维、矩阵逆的计算、奇异值软阈值处理等。它还在推荐系统中被用于协同过滤算法，以及在自然语言处理中用于词嵌入等任务。

4.3.1 奇异值分解的公式

实际上，SVD 是一种矩阵分解方法，可将任意矩阵分解为三个矩阵的乘积，用以代表原矩阵中最本质的变换。如式（4.19）所示，利用 SVD 可以将任意矩阵$\boldsymbol{A}$分解为$\boldsymbol{U}$、$\boldsymbol{\Sigma}$和$\boldsymbol{V}^T$三个矩阵的乘积：

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{4.19}$$

其中，$\boldsymbol{A}$的维度为$m \times n$。$\boldsymbol{U}$为左奇异矩阵，其维度为$m \times m$，是一个标准正交矩阵，即

$\boldsymbol{UU^T = E}$，$\boldsymbol{E}$为单位矩阵。$\boldsymbol{V}$为右奇异矩阵，其维度为$n \times n$，也是一个正交矩阵，即$\boldsymbol{VV^T = E}$。$\boldsymbol{\Sigma}$为奇异值矩阵，其为一个$m \times n$的对角矩阵，仅在主对角线上有值，其它元素均为 0。$\boldsymbol{\Sigma}$的主对角线上的值称为$\boldsymbol{A}$的奇异值（记为σ_i），该奇异值同时也是$\boldsymbol{A^T A}$或$\boldsymbol{AA^T}$的特征值（记为λ_i）的平方根，即$\sigma_i = \sqrt{\lambda_i}(i = 1,2,\ldots,r)$，注意$r$为$m$和$n$中的最小者。

基于奇异值分解的矩阵降维，就是要找到一个比较小的值k，保留$\boldsymbol{\Sigma}$中前k个奇异值，其中$\boldsymbol{U}$的维度从$m \times m$变成了$m \times k$，$\boldsymbol{V}$的维度从$n \times n$ 变成了$k \times n$，$\boldsymbol{\Sigma}$的维度从$m \times n$变成了$k \times k$的方阵，从而达到降维效果。

4.3.2 奇异值分解的原理

类比方阵的特征分解，对于任意矩阵$\boldsymbol{A}$（维度为$m \times n$），可以类似定义其特征分解$\boldsymbol{Ax} = \lambda \boldsymbol{x}$，我们来推导其与方阵特征分解的关系。

与式（4.19）的记法一致，取$\boldsymbol{V}$的单位向量$\boldsymbol{v}_i$，则有：

$$\begin{aligned} \|\boldsymbol{Av}_i\|^2 &= (\boldsymbol{Av}_i)^T \boldsymbol{Av}_i \\ &= \boldsymbol{v}_i{}^T \boldsymbol{A^T A v}_i \\ &= \boldsymbol{v}_i{}^T \lambda_i \boldsymbol{v}_i \\ &= \lambda_i \end{aligned} \tag{4.20}$$

可见，$\boldsymbol{A}$的右奇异向量$\boldsymbol{v}_i$就是$\boldsymbol{A^T A}$的特征向量，$\boldsymbol{A}$的奇异值σ_i就是特征值λ_i的平方根。

类似地，$\boldsymbol{A}$的左奇异向量$\boldsymbol{u}_i$就是$\boldsymbol{AA^T}$的特征向量，$\boldsymbol{A}$的奇异值σ_i就是特征值λ_i的平方根。

因此，任意矩阵$\boldsymbol{A}$的 SVD 分解过程如图 4.9 所示：

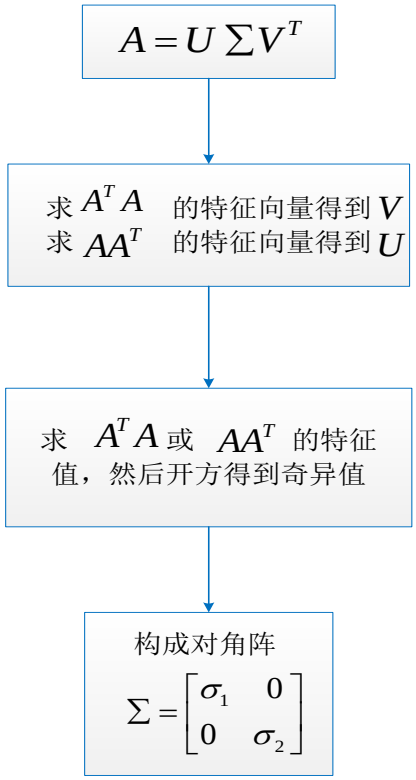

图 4.9 矩阵的 SVD 分解过程

4.3.3 矩阵的 SVD 层级分解

设矩阵$\boldsymbol{A} = \begin{bmatrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 2 & 6 \\ 5 & 3 & 5 & 8 \\ 9 & 7 & 9 & 3 \end{bmatrix}$，其 SVD 分解结果为$\boldsymbol{A} = \boldsymbol{U\Sigma V^T}$，如图 4.10 所示。

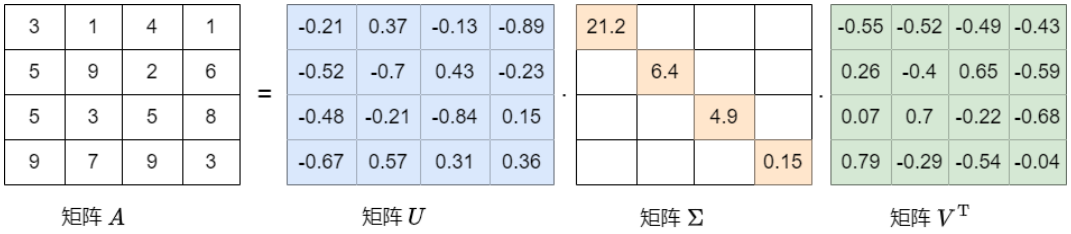

图 4.10 矩阵$\boldsymbol{A}$的 SVD 分解结果

按照 4 个奇异值从大到小的顺序，可将矩阵$\boldsymbol{A}$分解为四个层级。当奇异值取最大值$\sigma_1 = 21.2$时，矩阵$\boldsymbol{A}$的分解层级为：

$$\boldsymbol{L}_1 = \sigma_1 \cdot \boldsymbol{U}(:,0)\boldsymbol{V}^T(0,:) \tag{4.21}$$

当奇异值取次大值$\sigma_2 = 6.4$时，矩阵$\boldsymbol{A}$的分解层级为：

$$\boldsymbol{L}_2 = \sigma_2 \cdot \boldsymbol{U}(:,1)\boldsymbol{V}^T(1,:) \tag{4.22}$$

当奇异值取值$\sigma_3 = 4.9$时，矩阵$\boldsymbol{A}$的分解层级为：

$$\boldsymbol{L}_3 = \sigma_3 \cdot \boldsymbol{U}(:,2)\boldsymbol{V}^T(2,:) \tag{4.23}$$

当奇异值取最小值$\sigma_4 = 0.15$时，矩阵$\boldsymbol{A}$的分解层级为：

$$\boldsymbol{L}_4 = \sigma_4 \cdot \boldsymbol{U}(:,3)\boldsymbol{V}^T(3,:) \tag{4.24}$$

这四个层级体现了矩阵$\boldsymbol{A}$的四个不同结构层面，将其累加起来，即为矩阵$\boldsymbol{A}$的原始取值，如图 4.11 所示。

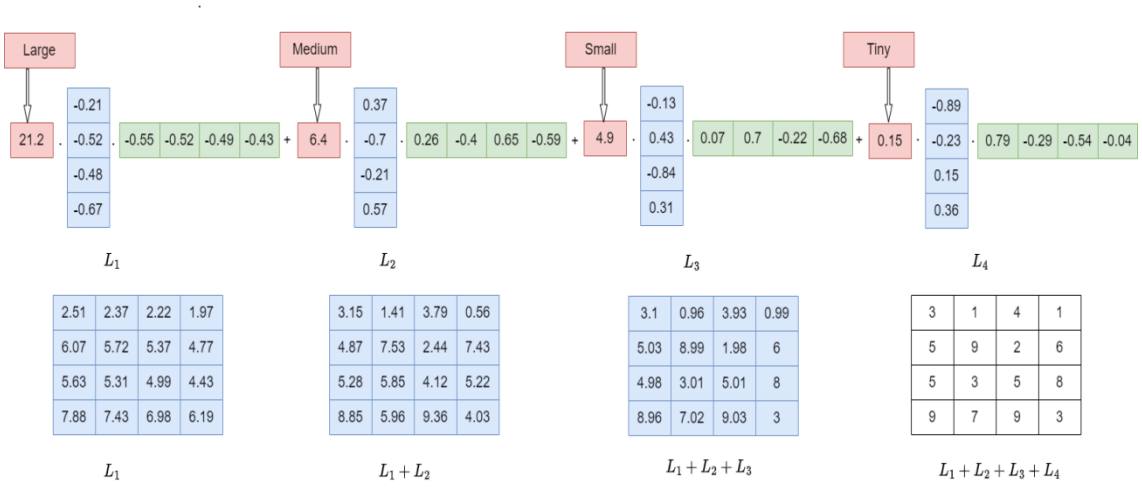

图 4.11 矩阵$\boldsymbol{A}$的 SVD 层级分解结果及其累加结果

4.3.4 SVD 的核心代码实现

本节将介绍 SVD 算法的 Python 代码实现，并给出代码的简单运行实例。相信读者在此基础上能够在实验课中进一步完善和改进代码，完成更具挑战性和实用性的应用任务。

（1）矩阵的 SVD 分解

有很多软件包可以实现 SVD，Numpy 的线性代数工具箱 linalg 就是其中之一。例如，对矩阵$\begin{bmatrix}1 & 1\\ 7 & 7\end{bmatrix}$实现 SVD 分解，可在 Python 控制台下输入如下命令：

```
>>> import numpy.linalg as la
>>> U, Sigma, VT = la.svd([[1, 1], [7, 7]])
>>> U
```

```
array([[-0.14142136, -0.98994949],
       [-0.98994949,  0.14142136]])
>>> Sigma
array([1.00000000e+01, 2.82797782e-16])
>>> VT
array([[-0.70710678, -0.70710678],
       [ 0.70710678, -0.70710678]])
```

注意，对角阵 Sigma 以行向量 array([1.00000000e+01, 2.82797782e-16])返回，可以有效地节省内存空间。

（2）基于 SVD 分解的图片压缩

基于 SVD 分解的图片压缩利用的是矩阵的“低秩近似”性质。所谓矩阵的低秩近似，实际上就是矩阵的一种稀疏表示形式，即利用一个秩较低的矩阵来近似表达原矩阵，不但能保留原矩阵的主要特征，而且可以降低数据的存储空间和计算开销。具体来讲，就是对图像矩阵进行 SVD 分解，并保留最重要的奇异值和对应的奇异向量，从而实现对图像的压缩。当然，基于保留的奇异值和奇异向量还可以方便地将压缩的图像重构出来。下面是具体的代码实现。

首先导入相关的库，并定义三个变量：

```
import numpy as np
from numpy.linalg import svd
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
vmin = 0 #图像最小像素值
vmax = 1 #图像最大像素值
image_bias = 1 #反转显示
```

其次，定义函数 plot_svd()。该函数首先将输入矩阵进行 SVD 分解，得到 U、S、V 三个矩阵。然后计算输入矩阵的层级分解结果，存放于列表 imgs 中。接着，将层级分解结果按照奇异值的索引标号 i 进行累加，存放于列表 combined_imgs 中。combined_imgs[i]表示 imgs 的前 i 项之和，即原始图像按照前 i 项特征值的压缩结果。最后，将输入图像的层级分解结果及压缩结果利用 plt.show()函数进行可视化。

```
def plot_svd(A):
    n = len(A)
    imshow(image_bias - A, cmap='gray', vmin=vmin, vmax=vmax)
    plt.show()
    U, S, V = svd(A)
    imgs = []
    for i in range(n):
        imgs.append(S[i]*np.outer(U[:, i], V[i])) #见 4.3.3 节公式
    combined_imgs = []
    for i in range(n):
        img = sum(imgs[:i+1])
        combined_imgs.append(img)
    fig, axes = plt.subplots(figsize=(n*n, n), nrows=1, ncols=n, sharex=True, sharey=True)
    for num, ax in zip(range(n), axes):
        ax.imshow(image_bias - imgs[num], cmap='gray', vmin=vmin, vmax=vmax)
```

```
            ax.set_title(np.round(S[num], 2), fontsize=50)
        plt.show()
        fig, axes = plt.subplots(figsize=(n*n,n), nrows=1, ncols=n, sharex=True, sharey=True)
        for num, ax in zip(range(n), axes):
            ax.imshow(image_bias - combined_imgs[num], cmap='gray', vmin=vmin,
vmax=vmax)
        plt.show()
        return U, S, V
```

定义 D 为一张原始二值图片对应的矩阵：

```
D = np.array([ [0, 1, 1, 0, 1, 1, 0],
               [1, 1, 1, 1, 1, 1, 1],
               [1, 1, 1, 1, 1, 1, 1],
               [0, 1, 1, 1, 1, 1, 0],
               [0, 0, 1, 1, 1, 0, 0],
               [0, 0, 0, 1, 0, 0, 0],
                ])
```

接着，执行语句 U, S, V = plot_svd(D)。

图 4.12 为 D 的可视化结果：

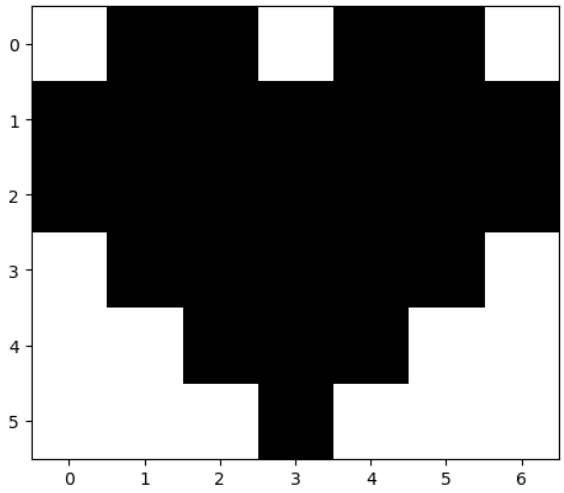

图 4.12 原始图像 D

原始图像 D 的层级分解结果如图 4.13 所示。其中，第一行的数字为原始图像矩阵 D 的奇异值大小，其中$\lambda_1 = 4.74$，$\lambda_2 = \lambda_3 = 1.41$，$\lambda_4 = 0.73$，$\lambda_5 = \lambda_6 = 0$。第二行的图片为原始图像分别按照对应奇异值进行层级分解的结果。注意，由于$\lambda_5 = \lambda_6 = 0$，其对应的图像层级分解结果为零矩阵，如图 4.13 第二行的后两列所示。

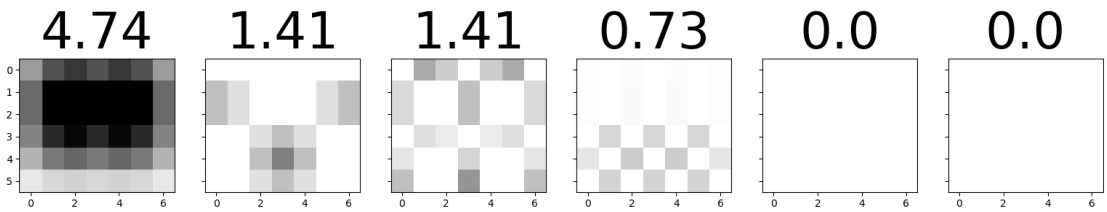

图 4.13 原始图像的层级分解结果

图 4.14 给出了不同层级分解结果的累加图像，即原始图像按照前 i 项特征值的压缩结果。注意，由于$\lambda_5 = \lambda_6 = 0$，其对应的累加图像和$\lambda_4$对应的累加图像完全一致，如图 4.14 的后三列所示。

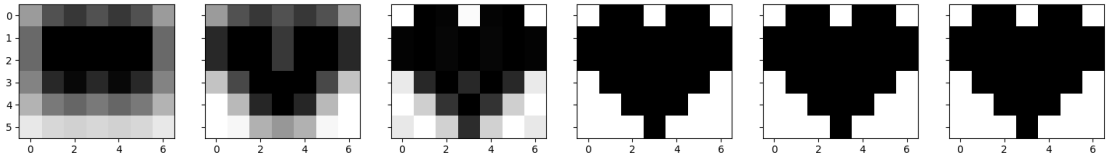

图 4.14 不同层级分解结果的累加图像

（3）基于 SVD 分解的餐馆菜品推荐系统

推荐系统的工作过程是：给定一个用户，系统为此用户推荐 N 个评分最高的菜品。为此，需要做到：

①找到用户没有评分的菜品；

②在用户没有评分的所有菜品中，对每个菜品估计一个评分；

③对这些菜品的评分从高到低进行排序，返回前 N 个菜品。

菜品推荐数据集如图 4.15 所示，可看作一个“用户-菜品”矩阵，其中有很多菜品没有评分（即图中为 0 分的菜品）。

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	三文鱼汉堡	鲁宾三明治	印度烤鸡	麻婆豆腐	宫保鸡丁	印度奶酪咖喱	俄式汉堡
Brett	2	0	0	4	4	0	0	0	0	0	0
Rob	0	0	0	0	0	0	0	0	0	0	5
Drew	0	0	0	0	0	0	0	1	0	4	0
Scott	3	3	4	0	3	0	0	2	2	0	0
Mary	5	5	5	0	0	0	0	0	0	0	0
Brent	0	0	0	0	0	0	5	0	0	5	0
Kyle	4	0	4	0	0	0	0	0	0	0	5
Sara	0	0	0	0	0	4	0	0	0	0	4
Shaney	0	0	0	0	0	0	5	0	0	5	0
Brendan	0	0	0	3	0	0	0	0	4	5	0
Leanna	1	1	2	1	1	2	1	0	4	5	0

图 4.15 菜品推荐数据集

新建文件 svdRec.py，定义如下函数：

```
def loadExData2(): #图 4.15 的菜品推荐数据集
return[ [2, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0],
        [3, 3, 4, 0, 3, 0, 0, 2, 2, 0, 0],
        [5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 5, 0, 0, 5, 0],
        [4, 0, 4, 0, 0, 0, 0, 5, 0, 0, 5],
        [0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 4],
        [0, 0, 0, 0, 0, 0, 5, 0, 0, 5, 0],
```

```
        [0, 0, 0, 3, 0, 0, 0, 0, 4, 5, 0],
        [1, 1, 2, 1, 1, 2, 1, 0, 4, 5, 0]]
```

在 Python 控制台下输入如下命令：

```
>>> import svdRec
>>> import numpy as np
>>> U, Sigma, VT = np.linalg.svd(np.mat(svdRec.loadExData2()))
>>> np.set_printoptions(suppress=True) #设置浮点数的显示方式，禁止科学计数法
>>> Sigma
```

array([13.65574047, 12.09426471, 8.39491738, 6.87317307, 5.32788293, 4.70763385, 3.2008274, 2.5168136, 1.9890208, 0.6710918, 0.])

可见，11 个奇异值依次减小，只有最后一个奇异值为 0。进一步，可以通过求前 k 个奇异值的平方和得到其“能量”，然后看 k 取多大时能达到总能量的 90%，具体命令行执行结果如下：

```
>>> Sig2=Sigma**2
>>> sum(Sig2) #总能量：全部 11 个奇异值的平方和
522.0000000000003
>>> sum(Sig2)*0.9 #总能量的 90%
469.8000000000003
>>> sum(Sig2[:2]) #前 2 个奇异值的能量
332.7504865882234
>>> sum(Sig2[:3]) #前 3 个奇异值的能量
403.2251244091107
>>> sum(Sig2[:4]) #前 4 个奇异值的能量
450.46563239414655
>>> sum(Sig2[:5]) #前 5 个奇异值的能量
478.85196886454156
```

可见，当 k=5 时高于总能量的 90%。于是，可以仅保留前 5 个奇异值及其对应的左右奇异向量，损失的能量仅有 10%。

据此，可以将所有的菜品映射到一个低维空间，再通过相似度计算（见 2.1.2 节）进行菜品推荐。定义函数 svdEst()，对给定用户、给定菜品估计一个评分值，其参数包括数据矩阵 dataMat，用户编号 user，相似度计算方法 simMeas，菜品编号 item，返回值为用户对菜品的评分。

```
def svdEst(dataMat, user, simMeas, item):
    n = np.shape(dataMat)[1]
    simTotal = 0.0; ratSimTotal = 0.0
    U, Sigma, VT = np.linalg.svd(dataMat)
    Sig5 = np.mat(np.eye(5) * Sigma[:5]) #Sig5 是一个对角阵
    xformedItems = dataMat.T * U[:, :5] * Sig5.I #得到变换后的数据项
    for j in range(n):
        userRating = dataMat[user, j]
        if (userRating == 0) or (j==item): #用户对菜品 j 没有评分或菜品 j 是用户给定的菜品
            continue
        similarity = simMeas(xformedItems[item, :].T, xformedItems[j, :].T)
```

```
            print ('the {} and {} similarity is: {}'.format(item, j, similarity))
            simTotal += similarity
            ratSimTotal += similarity * userRating
        if simTotal == 0: return 0
        else: return ratSimTotal/simTotal
```

该函数的第三行代码对数据集 dataMat 进行了 SVD 分解。然后，选取前 5 个奇异值（超过总能量的 90%），构建对角矩阵 Sig5，以便进行矩阵运算。函数的第五行代码利用矩阵 U 和对角矩阵 Sig5 的逆矩阵将“用户-菜品”矩阵（11x11 维）转换到了低维（11x5 维）空间中。

对给定的用户 user 和菜品 item，for 循环在用户对应行的所有元素（菜品）上进行遍历，并在低维空间中计算相似度。然后，对相似度求和，对相似度和用户评分值（dataMat[user, j]）的乘积求和，二者的比值（ratSimTotal/simTotal）作为返回值。

相似度计算定义了 ecludSim()、pearsSim()、cosSim()三种，分别是欧氏距离法、皮尔逊相关系数法、余弦相似度法。这三个函数的输入参数 inA 和 inB 都是列向量，返回值为两个列向量的相似度。代码如下：

```
def ecludSim(inA, inB):
    return 1.0/(1.0 + np.linalg.norm(inA - inB)) #取值范围为(0, 1]

def pearsSim(inA, inB):
    if len(inA) < 3 : return 1.0
    return 0.5+0.5*np.corrcoef(inA, inB, rowvar = 0)[0][1] #取值范围为[0, 1]

def cosSim(inA, inB):
    num = float(inA.T*inB)
    denom = np.linalg.norm(inA) * np.linalg.norm(inB)
    return 0.5+0.5*(num/denom) #取值范围为[0, 1]
```

定义函数 recommend()，实现推荐引擎的功能，会调用 svdEst()函数。该函数产生评分最高的 N 个推荐结果，如果不指定 N 的大小，则默认取 3。该函数的参数还包括数据矩阵 dataMat，用户编号 user，相似度计算方法 simMeas，估计方法 estMethod。

首先，对给定的用户 user 建立一个未评分的菜品列表，这个功能通过第一行代码“unratedItems = np.nonzero(dataMat[user, :].A==0)[1]”实现。如果不存在未评分菜品，则直接返回；否则，在所有的未评分菜品上进行循环。对每个未评分菜品，通过调用 estMethod()来产生该菜品的估计分值，并保存在列表 itemScores 中。最后按照估计的分值，对列表 itemScores 进行从大到小的逆向排序，返回得分最高的 N 个菜品。代码如下：

```
def recommend(dataMat, user, N=3, simMeas=cosSim, estMethod=standEst):
    unratedItems = np.nonzero(dataMat[user, :].A==0)[1] #找到未评分的菜品
    if len(unratedItems) == 0: return 'you rated everything' #没有未评分菜品
    itemScores = []
    for item in unratedItems: #只在未评分菜品里进行推荐
        estimatedScore = estMethod(dataMat, user, simMeas, item)
        itemScores.append((item, estimatedScore))
    return sorted(itemScores, key=lambda jj: jj[1], reverse=True)[:N]
```

接下来看看实际运行效果：

```
>>> import svdRec
```

```
>>> import numpy as np
>>> myMat=np.mat(svdRec.loadExData2())
>>> svdRec.recommend(myMat, 1, estMethod=svdRec.svdEst)
the 0 and 10 similarity is: 0.5440638439162508
the 1 and 10 similarity is: 0.3109563679388968
the 2 and 10 similarity is: 0.5323440877050434
the 3 and 10 similarity is: 0.4946502823155839
the 4 and 10 similarity is: 0.4531874380554362
the 5 and 10 similarity is: 0.7831450773172727
the 6 and 10 similarity is: 0.48492673307147904
the 7 and 10 similarity is: 0.8704435221137724
the 8 and 10 similarity is: 0.47895909338892634
the 9 and 10 similarity is: 0.5004306037986824
[(0, 5.0), (1, 5.0), (2, 5.0)]
```

上面对用户 1（Rob）进行了菜品推荐。由图 4.15 可知，用户 1 只对菜品 10（俄式汉堡）进行了评分，因此 recommend()函数将基于 SVD 和菜品 10 的评分，估计该用户对其余 10 份菜品的评分，然后从这些菜品中推荐评分最高的三个菜品：菜品 0（鳗鱼饭）、菜品 1（日式炸鸡排）及菜品 2（寿司饭），对应的估计分值都为 5.0。

类似地，可以对用户 3（Scott）进行菜品推荐：

```
>>> svdRec.recommend(myMat, 3, estMethod=svdRec.svdEst)
the 3 and 0 similarity is: 0.582287755604963
the 3 and 1 similarity is: 0.3306573409743566
the 3 and 2 similarity is: 0.37688226884110326
the 3 and 4 similarity is: 0.9528412570168112
the 3 and 7 similarity is: 0.5570891816169508
the 3 and 8 similarity is: 0.6572710915280217
the 5 and 0 similarity is: 0.3667300422408521
the 5 and 1 similarity is: 0.4764394146521668
the 5 and 2 similarity is: 0.5360103459607823
the 5 and 4 similarity is: 0.289825882298011
the 5 and 7 similarity is: 0.5086261351958441
the 5 and 8 similarity is: 0.8501988459532437
the 6 and 0 similarity is: 0.5624609729795953
the 6 and 1 similarity is: 0.44186020176575314
the 6 and 2 similarity is: 0.48752886088256747
the 6 and 4 similarity is: 0.5172024395918892
the 6 and 7 similarity is: 0.6387294399074961
the 6 and 8 similarity is: 0.23720977749330718
the 9 and 0 similarity is: 0.45030821482967975
the 9 and 1 similarity is: 0.5141713201901604
the 9 and 2 similarity is: 0.515476767819365
the 9 and 4 similarity is: 0.46138747814868947
the 9 and 7 similarity is: 0.46273673933664433
the 9 and 8 similarity is: 0.7319059579940872
```

the 10 and 0 similarity is: 0.5440638439162508
the 10 and 1 similarity is: 0.3109563679388968
the 10 and 2 similarity is: 0.5323440877050434
the 10 and 4 similarity is: 0.4531874380554362
the 10 and 7 similarity is: 0.8704435221137724
the 10 and 8 similarity is: 0.47895909338892634
[(6, 2.86536863953136), (9, 2.7834282978555747), (3, 2.7577463106038547)]
把相似度度量方法修改为皮尔逊系数法，运行结果如下：
>>> svdRec.recommend(myMat, 1, estMethod=svdRec.svdEst, simMeas=svdRec.pearsSim)
……
[(0, 5.0), (1, 5.0), (2, 5.0)]
>>> svdRec.recommend(myMat, 3, estMethod=svdRec.svdEst, simMeas=svdRec.pearsSim)
……
[(6, 2.88186477783302), (9, 2.775509091499505), (3, 2.7400646925129157)]

习题 4

4.1 SVD 适用于以下哪些任务？( )
(a) 图像压缩 (b) 数据降维
(c) 矩阵逆的计算 (d) 以上所有选项

4.2 SVD 可以用于数据降维的方法是什么？( )
(a) 选取最大的 k 个奇异值和对应的奇异向量
(b) 选取最小的 k 个奇异值和对应的奇异向量
(c) 选取所有奇异值和对应的奇异向量
(d) 选取中间的 k 个奇异值和对应的奇异向量

4.3 在 SVD 中，矩阵$\boldsymbol{A}$的奇异值是指：( )
(a) $\boldsymbol{A}$的特征值 (b) $\boldsymbol{A}$的特征向量
(c) $\boldsymbol{A}^T\boldsymbol{A}$的特征值的平方根 (d) $\boldsymbol{A}$的零空间的维度

4.4 SVD 分解后的矩阵$\boldsymbol{\Sigma}$是一个对角矩阵，其对角线上的元素是：( )
(a) 矩阵$\boldsymbol{A}$的特征值 (b) 矩阵$\boldsymbol{A}$的奇异值
(c) 矩阵$\boldsymbol{A}$的特征向量 (d) 矩阵$\boldsymbol{A}$的奇异向量

4.5 在 SVD 中，正交矩阵$\boldsymbol{U}$和$\boldsymbol{V}$的作用分别是：( )
(a) $\boldsymbol{U}$用于表示$\boldsymbol{A}$的特征向量，$\boldsymbol{V}$用于表示$\boldsymbol{A}$的特征值
(b) $\boldsymbol{U}$用于表示$\boldsymbol{A}$的奇异值，$\boldsymbol{V}$用于表示$\boldsymbol{A}$的奇异向量
(c) $\boldsymbol{U}$用于表示$\boldsymbol{A}$的左奇异向量，$\boldsymbol{V}$用于表示$\boldsymbol{A}$的右奇异向量
(d) $\boldsymbol{U}$用于表示$\boldsymbol{A}$的左奇异向量，$\boldsymbol{V}$用于表示$\boldsymbol{A}$的右特征向量

4.6 “对于 1 幅 500 万像素的图片，其维数为 500 万维。”这句话应该如何理解？这里所说的 2 维的图片是 500 万维，看似自相矛盾，那究竟是什么意思呢？

4.7 如图 4.3 所示，原始 2 维笛卡儿坐标系中的数据点$\boldsymbol{x}_n(n=1,\dots,N)$通过正交投影到其最大方差方向$u_1$而被降为 1 维，这个降维过程保持了重要的信息——方差。如果进一步考虑与u_1正交的方向u_2，u_2就对应第二大方差方向，数据正交投影到u_2就同样保持了第二大方差信息。请在图 4.3 中画出u_2，及各数据点在u_2上的正交投影。

4.8 请给出式（4.11）的详细推导过程。

4.9 请说明式（4.14）每一步推导的理由。

4.10 请推导式（4.15）。

4.11 图 4.6 中，lowDDataMat 相对于 dataMat 作了哪些几何变换呢？为什么需要作这些几何变换？

4.12 试证明：由任意$m \times n$矩阵$\boldsymbol{A}$构造的对称方阵$\boldsymbol{AA^T}$和$\boldsymbol{A^TA}$具有完全一样的非 0 特征值。

4.13 函数 recommend()仅在“未评分菜品”unratedItems 里进行推荐，你认为这样做有局限性吗？如果有，请你尝试进行改进，先说明改进思路，再进行代码实现和实际验证。

4.14 请解释函数 svdEst()中代码行“xformedItems = dataMat.T * U[:, :5] * Sig5.I #得到变换后的数据项”的作用，分析其背后的原理，并给出证明。