

降维

在维度灾难、冗余，这些在数据处理中常见的场景，不得不需要我们进一步处理，为了得到更精简更有价值的信息，我们所用的各种方法的统称就是降维。

降维有两种方式：

(1) 特征抽取：我觉得叫做特征映射更合适。因为它的思想即把高维空间的数据映射到低维空间。比如马上要提到的 PCA 即为一种特征映射的方法。还有基于神经网络的降维等。

(2) 特征选择：

a. 过滤式（打分机制）：过滤，指的是通过某个阈值进行过滤。比如经常会看到但可能并不会去用的，根据方差、信息增益、互信息

（在某个特定类别出现频率高，但在其他类别出现频率比较低的词条与该类的互信息比较大。通常用互信息作为特征词和类别之间的测度，如果特征词属于该类的话，它们的互信息量最大）

$$I(Y; X) = \sum_{y_i \in Y} \sum_{x_i \in X} P(X = x_i, Y = y_i) \log_2 \frac{P(X = x_i, Y = y_i)}{P(X = x_i)P(Y = y_i)}$$

、相关系数、卡方检验、F 检验来选择特征。

b. 包裹式：每次迭代产生一个特征子集，评分。

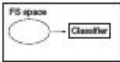


c. 嵌入式：先通过机器学习模型训练来对每个特征得到一个权值。接下来和过滤式相似，通过设定某个阈值来筛选特征。区别在于，嵌入式使用机器学习训练；过滤式采用统计特征。

➤ 过滤式方法运用统计指标来为每个特征打分并筛选特征，其聚焦于数据本身的特点。其优点是计算快，不依赖于具体的模型，缺点是选择的统计指标不是为特定模型定制的，因而最后的准确率可能不高。而且因为进行的是单变量统计检验，没有考虑特征间的相互关系。

➤ 包裹式方法使用模型来筛选特征，通过不断地增加或删除特征，在验证集上测试模型准确率，寻找最优的特征子集。包裹式方法因为有模型的直接参与，因而通常准确性较高，但是因为每变动一个特征都要重新训练模型，因而计算开销大，其另一个缺点是容易过拟合。

➤ 嵌入式方法利用了模型本身的特性，将特征选择嵌入到模型的构建过程中。典型的如 Lasso 和树模型等。准确率较高，计算复杂度介于过滤式和包裹式方法之间，但缺点是只有部分模型有这个功能。

Table 1. A taxonomy of feature selection techniques. For each feature selection type, we highlight a set of characteristics which can guide the choice for a technique suited to the goals and resources of practitioners in the field

Model search	Advantages	Disadvantages	Examples
Filter 	Univariate		
	Fast Scalable Independent of the classifier	Ignores feature dependencies Ignores interaction with the classifier	χ^2 Euclidean distance t -test Information gain, Gain ratio (Ben-Bassat, 1982)
	Multivariate		
	Models feature dependencies Independent of the classifier Better computational complexity than wrapper methods	Slower than univariate techniques Less scalable than univariate techniques Ignores interaction with the classifier	Correlation-based feature selection (CFS) (Hall, 1999) Markov blanket filter (MBF) (Koller and Sahami, 1996) Fast correlation-based feature selection (FCBF) (Yu and Liu, 2004)
Wrapper 	Deterministic		
	Simple Interacts with the classifier Models feature dependencies Less computationally intensive than randomized methods	Risk of over fitting More prone than randomized algorithms to getting stuck in a local optimum (greedy search) Classifier dependent selection	Sequential forward selection (SFS) (Kittler, 1978) Sequential backward elimination (SBE) (Kittler, 1978) Plus q take-away r (Ferri <i>et al.</i> , 1994) Beam search (Siedelecky and Sklansky, 1988)
	Randomized		
	Less prone to local optima Interacts with the classifier Models feature dependencies	Computationally intensive Classifier dependent selection Higher risk of overfitting than deterministic algorithms	Simulated annealing Randomized hill climbing (Skalak, 1994) Genetic algorithms (Holland, 1975) Estimation of distribution algorithms (Inza <i>et al.</i> , 2000)
Embedded 	Interacts with the classifier Better computational complexity than wrapper methods Models feature dependencies	Classifier dependent selection	Decision trees Weighted naive Bayes (Duda <i>et al.</i> , 2001) Feature selection using the weight vector of SVM (Guyon <i>et al.</i> , 2002; Weston <i>et al.</i> , 2003)

Principal components analysis 主成分分析在压缩消除冗余和数据噪音消除等有广泛应用。

特征值和特征向量

$$A\xi = \lambda\xi$$

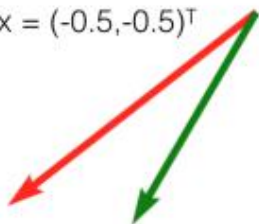
首先要理解矩阵线性变换，即矩阵乘法：

“矩阵乘法对应了一个变换，是把任意一个向量变成另一个方向或长度都大多不同的新向量。在这个变换的过程中，原向量主要发生旋转、伸缩的变化。如果矩阵对某一个向量或某些向量只发生伸缩变换，不对这些向量产生旋转的效果，那么这些向量就称为这个矩阵的特征向量，伸缩的比例就是特征值。”

← 映射前的向量x
← 映射后的向量x'

$$A = \begin{bmatrix} 0.1 & 0.4 \\ 0.3 & 0.8 \end{bmatrix}$$

$$x = (-0.5, -0.5)^T$$



$$x' = (-0.25, -0.55)^T$$

$$x = (-0.4289165, -0.90334414)^T$$



$$x' = (-0.404229306, -0.851350262)^T$$

通过求出 A 的特征值和对应的特征向量，就能找到旋转后正确的坐标轴，这个就是特征值和特征向量的一个实际应用：“得出使数据在各个维度区分达到最大的坐标轴”。（区分大：方差大）

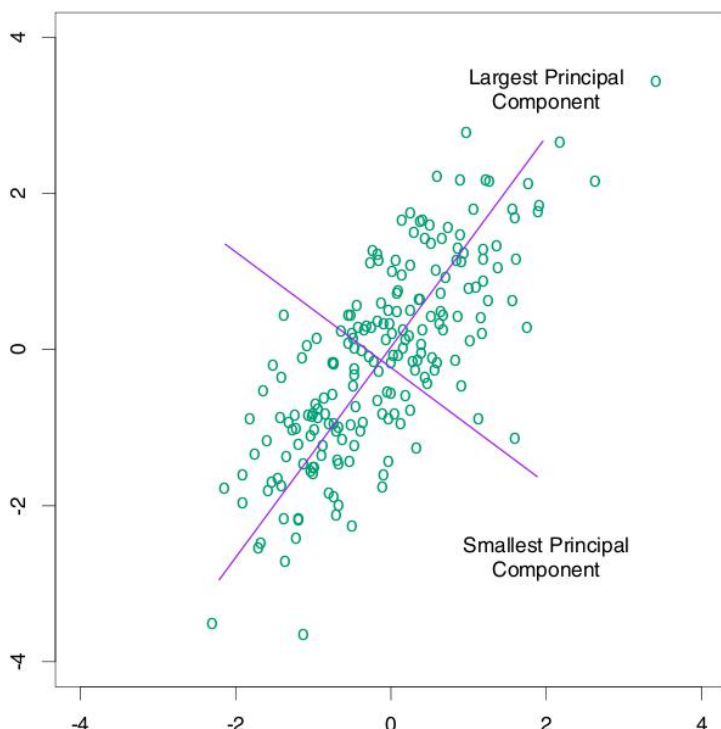
而在数据挖掘中，就会直接用特征值来描述对应特征向量方向上包含的信息量，而某一特征值除以所有特征值的和的值就为：该特征向量的方差贡献率。（在该维度下蕴含的信息的比例）

经过特征向量变换下的数据称为变量的主成分，当前 m 各主成分累计的方差贡献率达到 85% 以上就保留这个 m 个主成分的数据。实现了对数据进行降维的目的。

PCA 的目标

最小投影距离，最大投影方差。降维后不同维度的相关性为 0；

最大投影方差



显然数据离散性最大，代表数据在所投影的维度有越高的区分度，这个区分度就是信息量。应该考虑新的坐标轴。

将坐标轴进行旋转就能正确降维了。而这个旋转的操作其实就是矩阵变换

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{\text{矩阵变换}} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = Y$$

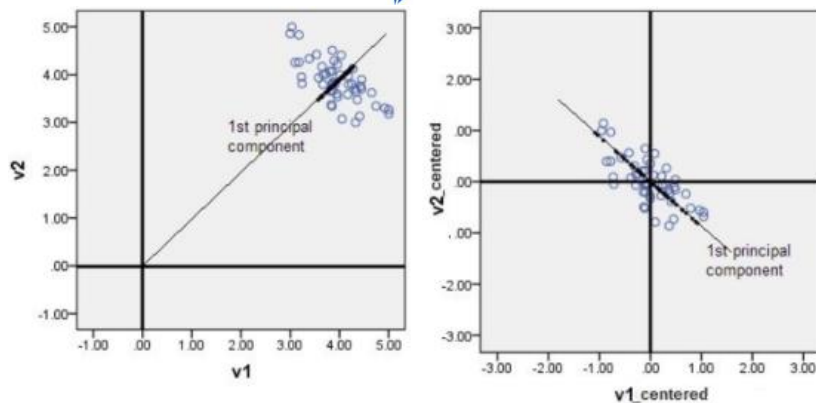
$$AX = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = Y$$

通过矩阵 A 对坐标系 X 进行旋转。经过一些数学推导，其实就可以得知，特征值对应的特征向量就是理想中想取得的正确坐标轴，而特征值就等于数据在旋转后的坐标上对应维度的方差（沿对应的特征向量的数据的方差）。而 A 其实即为我们想求得的那个降维特征空间，Y 则是我们想要的降维后的数据。

PCA 的过程

中心化

采用了中心化，均值为 0。未中心化，可能第一主成分的方向有误



这里可以看出主成分分析的目的是最小投影距离，最大投影方差。如果不中心化就达不到上述目的。

在中心化后，由于特征的均值变为 0，所以数据的协方差矩阵 C 可以用 $E(XX^T)$ 或者 $\frac{1}{m}XX^T$ 来表示， X^T 为矩阵的转置。这里 X 每一行为一个特征。当然也可以表示为 $E(X^TX)$ 或者 $\frac{1}{m}X^TX$ ，这时每一行为一个样本。

$$\text{cov}(X, Y) = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

vs.

$$\text{var}(X) = \frac{\sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

标准化数据

为什么要标准化，因为等下要算特征值和特征向量，特征值对应的特征向量就是理想中想取得正确坐标轴，而特征值就等于数据在变换后的坐标上对应维度的方差（沿对应的特征向量的数据的方差）。

而考虑这样一个例子，一个特征表示对象的长度（米为单位），而第二个特征表示对象的宽度（厘米为单位）。如果数据没有被标准化，那么最大方差及最大特征向量将隐式地由第一个特征定义。

算协方差矩阵

为什么要算协方差矩阵呢？这时就要想起我们之前的目的：在降维后的每一维度上，方差最大。而方差最大，则容易想到的就是协方差矩阵，去中心化后，协方差矩阵的对角线上的值正好就是各个数据维度的方差。原始数据的协方差矩阵 $X[n \times n]$ 对角线，对应的就是原始数据的方差；降维后的数据的协方差矩阵 $X' [n' \times n']$ ，对应的就是降维后的数据的方差。而我们的目的，则是使方差最大，这就又想到了另一个概念，迹，因为迹就是对角线上所有元素之和吗，而协方差矩阵的迹，不就是方差之和吗。这样我们构建损失函数，不就是 $\text{argmax}(\text{协方差矩阵 } X' \text{ 的迹})$!!!

首先设新的坐标系为 $W [n' \times n']$

$$\{w_1, w_2, \dots, w_{n'}\}$$

显然 w 为标准正交基

$$\|w\|_2 = 1, w_i^T w_j = 0$$

在新的坐标系的投影为 $Z = W^T X$ ，其中 Z 为 $\{z_1, z_2, \dots, z_{n'}\}$ 。

向量 X_i 在 w 上的投影坐标可以表示为 $(x_i, w) = x_i^T w$

易知中心化后，投影之后的均值为 0，即

$$\mu' = \frac{1}{n} \sum_{i=1}^n x_i^T w = \left(\frac{1}{n} \sum_{i=1}^n x_i^T \right) w = 0$$

投影之后的方差可以表示为

$$\begin{aligned} D(x) &= \frac{1}{n} \sum_{i=1}^n (x_i^T w)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (x_i^T w)^T (x_i^T w) \\ &= \frac{1}{n} \sum_{i=1}^n w^T x_i x_i^T w \\ &= w^T \left(\frac{1}{n} \sum_{i=1}^n x_i x_i^T \right) w \end{aligned}$$

$$\frac{1}{n} \sum_{i=1}^n x_i x_i^T$$

其实就是样本协方差矩阵，所以问题可以转化为

$$\begin{cases} \max \{w^T \Sigma w\}, \\ s.t. \quad w^T w = 1. \end{cases}$$

结合目标函数与条件，用 Σ 代替协方差矩阵，通过拉格朗日函数可以得到：

$$J(w) = w^T \Sigma w + \lambda (w^T w - 1)$$

对 w 进行求导：

$$\Sigma w = -\lambda w$$

此时求出来方差 $D(x)$ 为

$$D(x) = w^T \Sigma w = -\lambda w^T w = -\lambda$$

$-\lambda$ 是由特征值构成的对角阵，且特征值在对角线上，其余位置为 0。很显然， W 为标

准正交基，我们假如把特征值以及对应的 W 的那一列拿出来和相乘，不就是一个求特征值和特征向量的过程嘛。而 W 的维度为 $n' \times n'$ ，显然我们就是求前 n' 个最大的特征值以及对应的特征向量（同时也可以设定一个阈值，当前 n' 个特征向量达到总和的多少占比时，一般是 85%，就提取，这样可能在尽可能减少信息量损失的情况下更多地降低维度）。这 n' 个特征向量组成的矩阵 W 即为我们需要的矩阵。通过将原始数据矩阵与该矩阵进行矩阵乘法，便可以将 n 维的数据降到 n' 维。也就是说，一条 n 维的数据 $\{1 \times n\}$ ，假设选了前 m 个特征向量，通过乘以特征矩阵 $\{n \times m\}$ ，就变成了 m 维的数据 $\{1 \times m\}$ ，我们就达到了降维的目的。

SVD 分解

刚刚说了需求是前 n' 个最大的特征值以及对应的特征向量。于是就很容易想到对协方差矩阵进行 SVD 分解，或者也可以说特征值分解，选取最大的几个特征值（或者设置阈值来选最大的几个特征值），对应的特征向量进行标准化（使其成为标准正交基）后组成特征矩阵，这些特征向量都是正交的。

假设 M 是一个 $m \times n$ 阶矩阵，其中的元素全部属于域 K ，也就是实数域或复数域。如此则存在一个分解使得

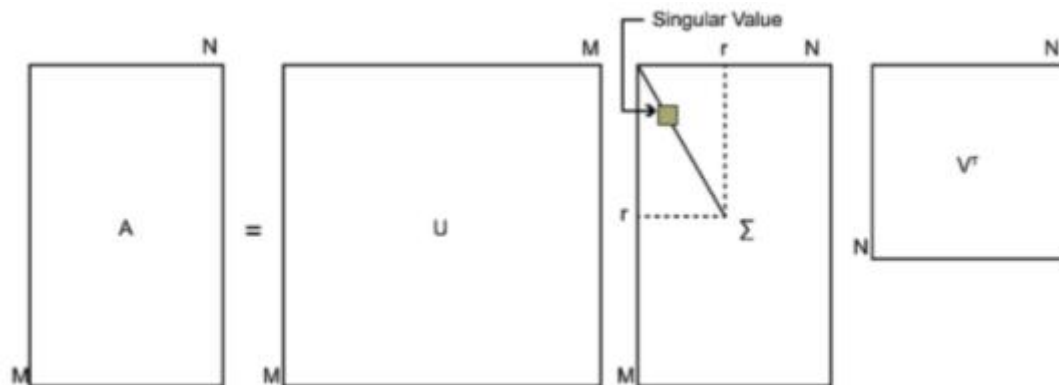
$$M = U \Sigma V^*,$$

其中 U 是 $m \times m$ 阶酉矩阵； Σ 是 $m \times n$ 阶非负实数对角矩阵；而 V^* ，即 V 的共轭转置，是 $n \times n$ 阶酉矩阵。这样的分解就称作 M 的奇异值分解。 Σ 对角线上的元素 Σ_{ii} 即为 M 的奇异值。

常见的做法是将奇异值由大而小排列。如此 Σ 便能由 M 唯一确定了。（虽然 U 和 V 仍然不能确定。）

$$\begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_1 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{bmatrix}$$

主对角线上的每个元素都称为奇异值， V 是一个 $n \times n$ 的矩阵。 U 和 V 都是酉矩阵，即满足 $U^T U = I, V^T V = I$ 。下图可以很形象的看出上面 SVD 的定义：



直观的解释 [编辑]

在矩阵 M 的奇异值分解中

$$M = U \Sigma V^*,$$

- V 的列 (rows) 组成一套对 M 的"正交"输入"或"分析"的基向量。这些向量是 $M^* M$ 的特征向量。
- U 的列 (rows) 组成一套对 M 的"正交"输出"的基向量。这些向量是 $M M^*$ 的特征向量。
- Σ 对角线上的元素是奇异值, 可视作为是在输入与输出间进行的标量的"膨胀控制"。这些是 $M M^*$ 及 $M^* M$ 的特征值的非负平方根, 并与 U 和 V 的行向量相对应。

$M M^T$ 就是一个 $m \times m$ 阶的矩阵, 这个矩阵的所有特征向量组成的矩阵就是 U 。这里面每个特征向量就是左奇异向量。

所以 $M^T M$ 就是一个 $n \times n$ 阶的矩阵, 这个矩阵的所有特征向量组成的矩阵就是 V 。这里面每个特征向量就是右奇异向量。而我们要找的, 也就是 $M^T M$ 的特征向量。

$$A = U \Sigma V^T$$

$$A^T = V \Sigma^T U^T$$

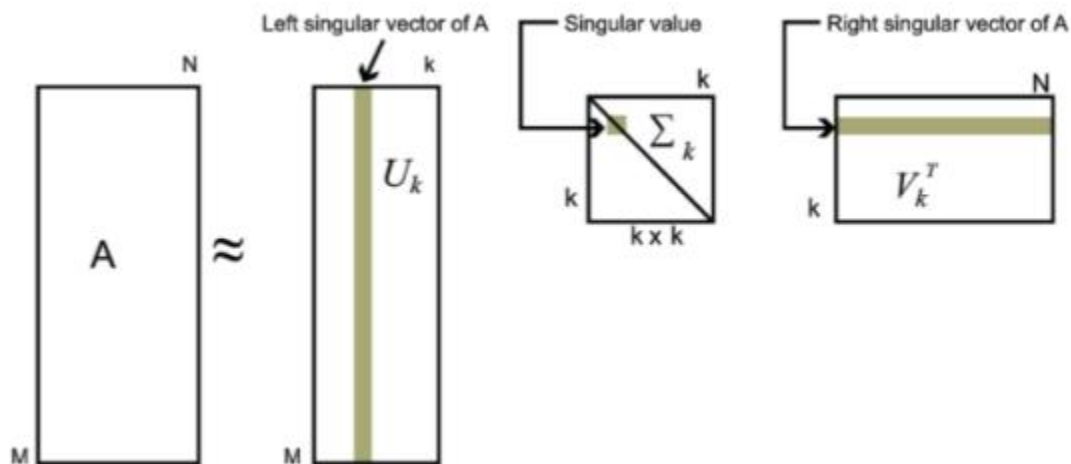
$$A^T A = V \Sigma^T U^T U \Sigma V^T$$

$$= V \Sigma^2 V^T$$

上式证明使用了: $U^T U = I, \Sigma^T \Sigma = \Sigma^2$ 。可以看出 $A^T A$ 的特征向量组成的就是 SVD 中的 V 矩阵。类似的方法可以得到 $A A^T$ 的特征向量组成的就是 SVD 中的 U 矩阵。

进一步我们还可以看出我们的特征值矩阵等于奇异值矩阵的平方, 也就是说特征值和奇异值满足如下关系: $\sigma_i = \sqrt{\lambda_i}$

这样也就是说, 我们可以不用 $\sigma_i = A v_i / u_i$ 来计算奇异值, 也可以通过求出 $A^T A$ 的特征值取平方根来求奇异值。



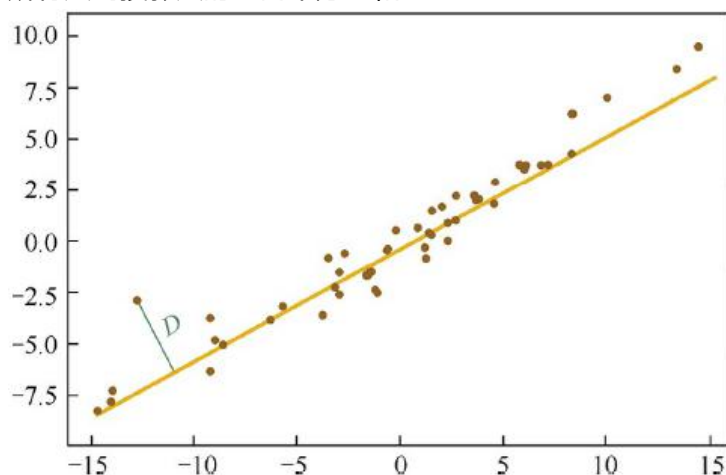
所以通过 SVD 分解就能求出我们需要的对应的特征值, 以及需要的特征空间, 而实际上在 sklearn 中 PCA 背后的算法就是用的 SVD, 而不是暴力特征分解。

定义降维后的空间信息占比为

$$\eta = \sqrt{\frac{\sum_{i=1}^d \lambda_i^2}{\sum_{i=1}^n \lambda_i^2}}$$

最小投影距离

通过最小投影距离其实是和线性回归的原理类似。我们希望投影后到 d 维超平面，最小化所有点到投影点的距离平方之和。



\mathbf{x}_k 为原始点经过变换到新的特征空间的点。数据集每个点到投影点的距离为

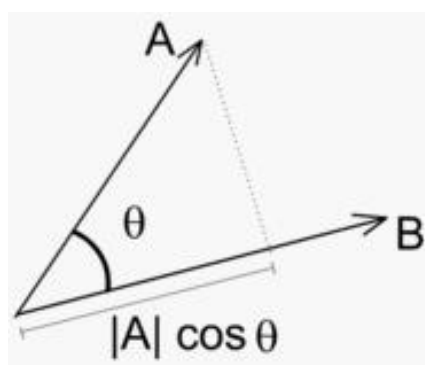
$$\text{distance}(\mathbf{x}_k, D) = \|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|_2$$

$\tilde{\mathbf{x}}_k$ 表示 \mathbf{x}_k 在超平面 D 上的投影点（投影向量），该超平面由 d 个标准正交基

$$W = \{\omega_1, \omega_2, \dots, \omega_d\}$$

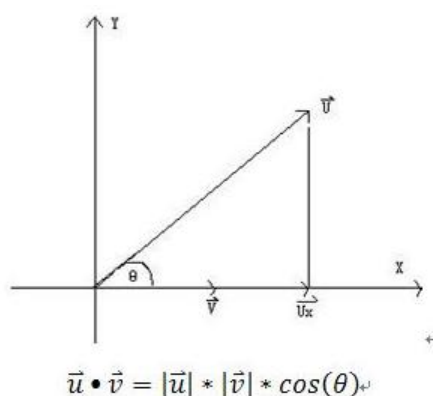
构成，于是投影向量可以表示为

$$\tilde{\mathbf{x}}_k = \sum_{i=1}^d (\omega_i^T \mathbf{x}_k) \omega_i$$



$$\omega_i^T \mathbf{x}_k$$

表示 \mathbf{x}_k 在 ω_i 方向上投影的长度。PCA 要优化的方法目标为



$$\begin{cases} \arg \min_{\omega_1, \dots, \omega_d} \sum_{k=1}^n \| \mathbf{x}_k - \widetilde{\mathbf{x}}_k \|_2^2, \\ s.t. \quad \omega_i^T \omega_j = \delta_{ij} = \begin{cases} 1, i = j; \\ 0, i \neq j. \end{cases} \end{cases}$$

其中损失函数可以展开

$$\begin{aligned} \| \mathbf{x}_k - \widetilde{\mathbf{x}}_k \|_2^2 &= (\mathbf{x}_k - \widetilde{\mathbf{x}}_k)^T (\mathbf{x}_k - \widetilde{\mathbf{x}}_k) \\ &= \mathbf{x}_k^T \mathbf{x}_k - \mathbf{x}_k^T \widetilde{\mathbf{x}}_k - \widetilde{\mathbf{x}}_k^T \mathbf{x}_k + \widetilde{\mathbf{x}}_k^T \widetilde{\mathbf{x}}_k \\ &= \mathbf{x}_k^T \mathbf{x}_k - 2\mathbf{x}_k^T \widetilde{\mathbf{x}}_k + \widetilde{\mathbf{x}}_k^T \widetilde{\mathbf{x}}_k. \end{aligned}$$

第一项的 $\mathbf{x}_k^T \mathbf{x}_k$ 是一个常数，将之前表示的 $\widetilde{\mathbf{x}}_k$ 代入上式第二三项，注意 $\omega_i^T \mathbf{x}_k$ 和 $\omega_j^T \mathbf{x}_k$ 表示投影长度，都是常数

$$\begin{aligned} \mathbf{x}_k^T \widetilde{\mathbf{x}}_k &= \mathbf{x}_k^T \sum_{i=1}^d (\omega_i^T \mathbf{x}_k) \omega_i \\ &= \sum_{i=1}^d (\omega_i^T \mathbf{x}_k) \mathbf{x}_k^T \omega_i \\ &= \sum_{i=1}^d \omega_i^T \mathbf{x}_k \mathbf{x}_k^T \omega_i, \\ \widetilde{\mathbf{x}}_k^T \widetilde{\mathbf{x}}_k &= \left(\sum_{i=1}^d (\omega_i^T \mathbf{x}_k) \omega_i \right)^T \left(\sum_{j=1}^d (\omega_j^T \mathbf{x}_k) \omega_j \right) \\ &= \sum_{i=1}^d \sum_{j=1}^d ((\omega_i^T \mathbf{x}_k) \omega_i)^T ((\omega_j^T \mathbf{x}_k) \omega_j). \end{aligned}$$

由于当 i 不等于 j 时，

$$\omega_i^T \omega_j = 0$$

因此只剩下 1

$$\begin{aligned} \widetilde{\mathbf{x}}_k^T \widetilde{\mathbf{x}}_k &= \sum_{i=1}^d ((\omega_i^T \mathbf{x}_k) \omega_i)^T ((\omega_i^T \mathbf{x}_k) \omega_i) = \sum_{i=1}^d (\omega_i^T \mathbf{x}_k) (\omega_i^T \mathbf{x}_k) \\ &= \sum_{i=1}^d (\omega_i^T \mathbf{x}_k) (\mathbf{x}_k^T \omega_i) = \sum_{i=1}^d \omega_i^T \mathbf{x}_k \mathbf{x}_k^T \omega_i. \end{aligned}$$

$$\sum_{i=1}^d \omega_i^T \mathbf{x}_k \mathbf{x}_k^T \omega_i \quad \text{实际是矩阵 } W^T \mathbf{x}_k \mathbf{x}_k^T W \text{ 的迹，将其代入得到}$$

$$\begin{aligned}\| \mathbf{x}_k - \widetilde{\mathbf{x}}_k \|_2^2 &= -\sum_{i=1}^d \omega_i^T \mathbf{x}_k \mathbf{x}_k^T \omega_i + \mathbf{x}_k^T \mathbf{x}_k \\ &= -\text{tr}(\mathbf{W}^T \mathbf{x}_k \mathbf{x}_k^T \mathbf{W}) + \mathbf{x}_k^T \mathbf{x}_k.\end{aligned}$$

因此损失函数可以写为

$$\begin{aligned}\arg \min_W \sum_{k=1}^n \| \mathbf{x}_k - \widetilde{\mathbf{x}}_k \|_2^2 &= \sum_{k=1}^n (-\text{tr}(\mathbf{W}^T \mathbf{x}_k \mathbf{x}_k^T \mathbf{W}) + \mathbf{x}_k^T \mathbf{x}_k) \\ &= -\sum_{k=1}^n \text{tr}(\mathbf{W}^T \mathbf{x}_k \mathbf{x}_k^T \mathbf{W}) + C.\end{aligned}$$

又由于矩阵乘法性质

$$\sum_k \mathbf{x}_k \mathbf{x}_k^T = \mathbf{X} \mathbf{X}^T$$

所以问题可以转化为

$$\begin{cases} \arg \max_W \text{tr}(\mathbf{W}^T \mathbf{X} \mathbf{X}^T \mathbf{W}), \\ s.t. \quad \mathbf{W}^T \mathbf{W} = \mathbf{I}. \end{cases}$$

这其实就已经和上面用最大方差理论等价了，再使用同样的步骤就可以求得最终的投影空间，最佳投影方向就是最大特征值所对应的特征向量。

核主成分分析

Kernelized PCA

有时我们的数据并不是可以投影到线性超平面的，这时候就不能直接进行 PCA 降维，这里就需要用到支持向量机一样的核函数的思想，先把数据从 n 维映射到线性可分的高维 $N > n$ ，然后再从 N 维降维到一个低维度 n' ，这里的维度之间满足 $n' < n < N$ 。

通过映射 ϕ 将 n 维映射到 N 维。

对于 n 维空间的特征分解：

$$\sum_{i=1}^m \mathbf{x}^{(i)} \mathbf{x}^{(i)T} \mathbf{W} = \lambda \mathbf{W}$$

映射为：

$$\sum_{i=1}^m \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^T \mathbf{W} = \lambda \mathbf{W}$$

通常 ϕ 不是显式的，计算的时候通过核函数完成。

其中核函数有

$$K = XX^T = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix} [\phi(x_1), \dots, \phi(x_N)] = \begin{bmatrix} \kappa(x_1, x_1) & \dots & \kappa(x_1, x_N) \\ \vdots & \ddots & \vdots \\ \kappa(x_N, x_1) & \dots & \kappa(x_N, x_N) \end{bmatrix}$$

PCA 总结

为了克服 PCA 的一些缺点，也衍生出了很多 PCA 的变种，包括上述提到的 KPCA。

PCA 的主要优点有：

- (1) 仅通过方差衡量信息量，不受数据集以外因素影响。
- (2) 各主成分之间正交，消除可能出现的低秩、或者原始数据成分间相关的可能。
- (3) 计算方法简单，易于实现。

主要缺点：

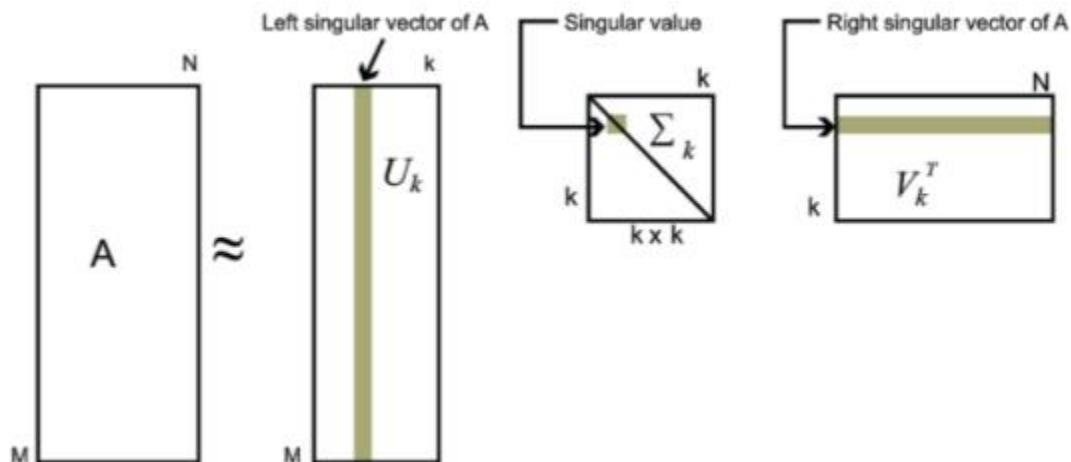
- (1) 降维后各个特征维度的含义具有一定的模糊性，数据的可解释性没有原始样本的特征强。（可解释性变弱）
- (2) 方差小的非主成分可能含有对样本差异的重要信息（可能丢失强力特征），因降维丢弃后可能对后续处理有影响。

SVD 应用

奇异值

sigma 是一个对角矩阵，但通常不是方阵。sigma 的对角元素被称为奇异值，与特征值类似。因此与 PCA 类似，我们可以取 sigma 中最大的 k 个，来简化数据：

$$u' = m * k ; \sigma' = k * k ; v'' = k * n$$



重构矩阵

利用新的三个矩阵 u' , sigma' , v'' 相乘仍然得到一个 m * n 的矩阵。如果你选择的 k 个奇

异值所占的所有奇异值比例足够大，那么新得到的 $m * n$ 的矩阵将与原矩阵 C 非常接近。

矩阵压缩

```
# -*- coding: utf-8 -*-
"""
arr =
    [[0, 0, 0, 2, 2],
     [0, 0, 0, 3, 3],
     [0, 0, 0, 1, 1],
     [1, 1, 1, 0, 0],
     [2, 2, 2, 0, 0],
     [5, 5, 5, 0, 0],
     [1, 1, 1, 0, 0]]

u = 7 * 7
sigma = 7 * 5, 只返回了对角元素, 其余 0 元素被省略
V = 5 * 5
"""

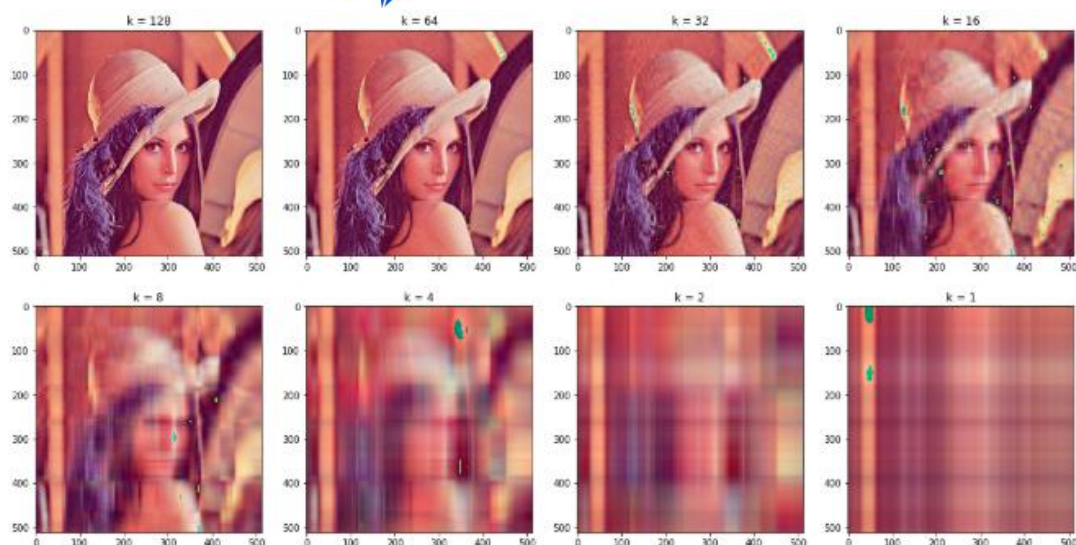
import numpy as np

arr = np.array([[0, 0, 0, 2, 2], [0, 0, 0, 3, 3], [0, 0, 0, 1, 1],
                [1, 1, 1, 0, 0], [2, 2, 2, 0, 0], [5, 5, 5, 0, 0], [1, 1, 1, 0, 0]])

# 1. 分解
u, sigma, v = np.linalg.svd(arr)

# 2. 重构
new_arr = np.mat(u[:, 0:2]) * np.mat(np.diag(sigma[0:2])) * np.mat(v[0:2, :])
```

`new_arr` 与 `arr` 非常接近，几乎相等。这其实是类似于图像压缩，只保留图像分解后的两个方阵和一个对角阵的对角元素，就可以恢复原图像。



数据降维

之所以能进行数据降维，原理与 PCA 一样，SVD 计算出的三个矩阵对应的是：

u ： CC' 的特征向量矩阵； σ ：奇异值矩阵，其中每个元素为特征值开方； v' ： $C'C$ 的特征向量矩阵。

你会发现 σ 与 $C'C$ 恰好是主成分分析所需要的两个量。因此 SVD 降维与 PCA 是一致的，尤其是事先对数据进行了中心化，再奇异值分解，则 PCA 降维和 SVD 降维完全一样。

利用 SVD 来实现 PCA 的代码：

```
# -*- coding: utf-8 -*-
""" svd 应用 2 - 降维 """

import numpy as np
import matplotlib.pyplot as plt

class PCA:
    """ 通过 SVD 分解来实现 PCA
    1. 训练数据 train_x 必须一行代表一个样本，一列代表一个特征
    2. 能够同时压缩 train_x 的行和列
    3. 可以选择在压缩前，是否对数据进行中心化
    """
    def __init__(self, dimension, centered=True, compression="cols"):
```

```

dimension:      降维后的维度
centered:       是否事先对数据进行中心化
compression:    压缩行, 还是压缩列
"""

self.dimension = dimension
self.centered = centered
self.compression = compression

def _centered(self, train_x):
    """ 数据中心化 """
    return train_x - np.mean(train_x, axis=0)

def _svd(self, train_x):
    """ 奇异值分解 """
    return np.linalg.svd(train_x)

def transform(self, train_x):
    """ 数据转化(降维)
    train_x:      训练数据, 一行代表一个样本
    u, sigma, v:   奇异值分解结果
    result:        降维后的数据
    """

    # 1. 数据中心化
    if self.centered == True:
        train_x = self._centered(train_x)

    # 2. 奇异值分解
    u, sigma, v = self._svd(train_x)
    v = v.T

    # 3. 降维
    if self.compression == "cols":
        result = np.dot(train_x, v[:, 0:self.dimension])
    elif self.compression == "rows":
        result = np.dot(u[:, 0:self.dimension], train_x[0:self.dimension, :])
    else:
        raise(Exception("parameter error."))
    
```

return result

压缩行 - 压缩记录

SVD 分解得到的三个矩阵分别称为：左奇异向量，奇异值矩阵，右奇异向量。左奇异向量用于压缩行，右奇异向量压缩列。压缩方法均是取奇异值较大的左奇异向量或者右奇异向量与原数据 C 相乘。

压缩列 - 压缩特征

```
def load_data():
    with open("../SVD/data/Iris.txt", "r") as f:
        iris = []
        for line in f.readlines():
            temp = line.strip().split(",")
            if temp[4] == "Iris-setosa":
                temp[4] = 0
            elif temp[4] == "Iris-versicolor":
                temp[4] = 1
            elif temp[4] == "Iris-virginica":
                temp[4] = 2
            else:
                raise(Exception("data error. "))
            iris.append(temp)
        iris = np.array(iris, np.float)
    return iris

def draw_result(new_trainX, iris):
    """
    new_trainX:    降维后的数据
    iris:          原数据
    """
    plt.figure()
    # Iris-setosa
    setosa = new_trainX[iris[:, 4] == 0]
    plt.scatter(setosa[:, 0], setosa[:, 1], color="red", label="Iris-setosa")

    # Iris-versicolor
    versicolor = new_trainX[iris[:, 4] == 1]
```

```
plt.scatter(versicolor[:, 0], versicolor[:, 1], color="orange",
label="Iris-versicolor")

# Iris-virginica
virginica = new_trainX[iris[:, 4] == 2]
plt.scatter(virginica[:, 0], virginica[:, 1], color="blue", label="Iris-virginica")
plt.legend()
plt.show()

def main(dimension, centered, compression):
    # 导入数据
    iris = load_data()

    # 降维
    clf = PCA(2, centered, compression)
    new_iris = clf.transform(iris[:, 0:4])

    # 降维结果可视化
    draw_result(new_iris, iris)
```

数据进行中心化后降维的结果，与 PCA 一文结果一致

协同过滤

协同过滤包含基于用户的协同过滤，基于物品的协同过滤。这两种方式本身是不需要 SVD 就可以进行的；但是加入了 SVD 之后，可以减少计算量同时还能提高推荐效果。

基于用户的协同过滤

比如补充下表当中 Jim 对日式鸡排，寿司的打分：

	鳗鱼饭	日式炸鸡排	寿司	烤牛肉	手撕猪肉
Jim	2	0	0	4	4
John	5	5	5	3	3
sally	2	4	2	1	2
Tom	1	1	1	5	5

可以直接计算 Jim 与其余三个用户的相似度，然后选最相似的样本来为 Jim 的两个空位打分。但是这样，如果一旦样本、特征过多，计算量就猛增。而事实上，我们不一定需要那么

多特征，因此可以使用 SVD 分解把样本映射到低维空间。（事实上，容易能从数据中看出来映射 2 维空间，左边三个和右边两个明显不一样）

```
food = np.mat([[2, 0, 0, 4, 4], [5, 5, 5, 3, 3], [2, 4, 2, 1, 2], [1, 1, 1, 5, 4]])
```

```
u, sigma, v = np.linalg.svd(food)
```

```
simple_food = np.mat(u[:, 0:2]) * np.mat(np.diag(sigma[0:2])) * np.mat(v[0:2, :])
```