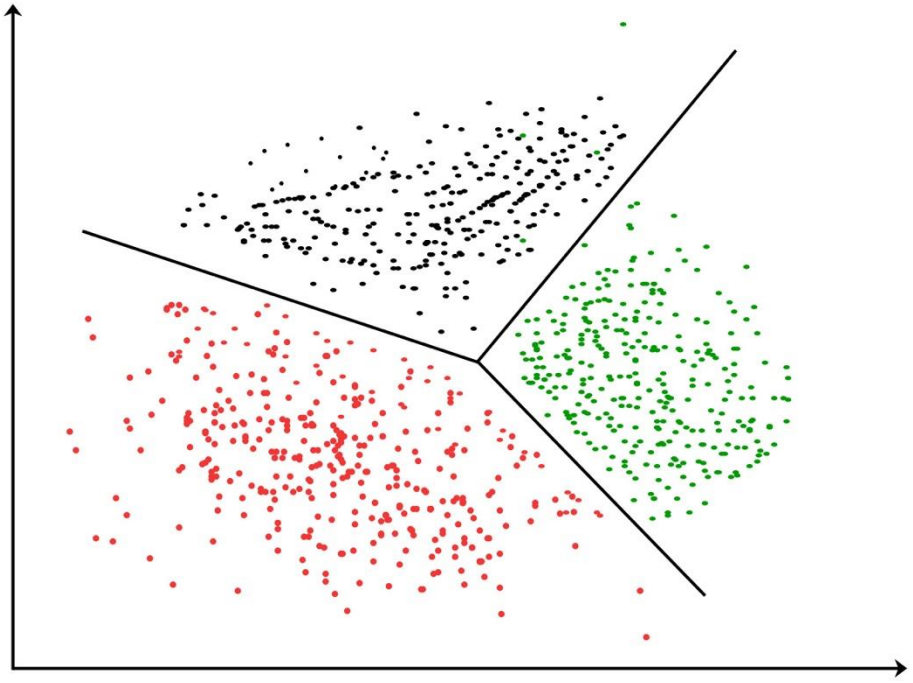
聚 类

聚类问题

- 无监督学习里典型例子就是**聚类**。
- 聚类是**将样本集合中相似的样本（实例）分配到相同的类，不相似的样本分配到不同的类。**与分类不同，聚类并不关心分的这一类是什么。
- 聚类的核心概念是**相似度**（similarity）或**距离**（distance）



- 聚类算法涉及到两个基本问题：**距离计算，性能度量**

聚类问题：距离计算

- 因为相似度直接影响聚类的结果，所以**距离和相似度选择是聚类的根本问题。**
- 有多种相似度或距离的定义，也可根据需要自己定义，但一般需满足如下条件:

非负性： $\mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geqslant 0$

同一性： $\mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$ 当且仅当 $\boldsymbol{x}_i = \boldsymbol{x}_j$

对称性： $\mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathrm{dist}(\boldsymbol{x}_j, \boldsymbol{x}_i)$

直递性： $\mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \leqslant \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_k) + \mathrm{dist}(\boldsymbol{x}_k, \boldsymbol{x}_j)$

聚类问题：性能度量

性能度量的作用：

- 评估聚类效果的好坏
- 作为聚类的优化目标

足够“好的”聚类？

- 同一簇的样本尽可能彼此相似。（“簇内相似度”高）
- 不同簇的样本尽可能地不同。（“簇间相似度”低）

聚类问题：性能度量

Distance Matrix

	A	B	C	D	E	F
A	0	16	47	72	77	79
B	16	0	37	57	65	66
C	47	37	0	40	30	35
D	72	57	40	0	31	23
E	77	65	30	31	0	10
F	79	66	35	23	10	0

指标

簇C中的所有样本的平均距离（簇内距离）：

$$\mathrm{avg}(C) = \frac{2}{|C|(|C|-1)} \sum_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

簇C_i与簇C_j中的中心点间距离（簇间距离）：

$$d_{\mathrm{cen}}(C_i, C_j) = \mathrm{dist}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)$$

由簇间距离与簇内距离定义的DB指数（Davies-Bouldin Index）：

（DBI值越小，聚类结果越好）

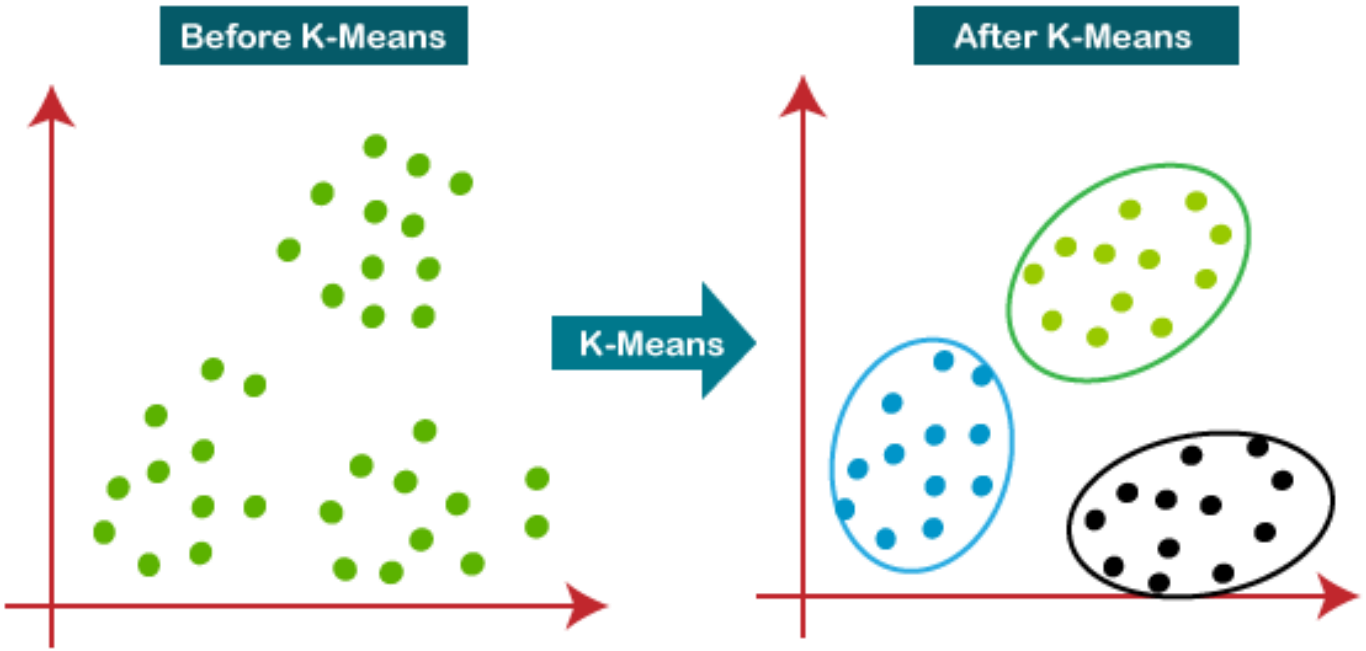
$$\mathrm{DBI} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\mathrm{avg}(C_i) + \mathrm{avg}(C_j)}{d_{\mathrm{cen}}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)} \right)$$

k-means算法

k-means算法（k均值聚类）

模型

- k-means 是最常用的**基于欧式距离**的聚类算法。
- k均值聚类是硬“ 聚类”。因为每个样本只能聚为1个类。
- k-means的目标是将n个样本分到k个不同的类或簇中，这里假设$k < n$。 k个类$G_1, G_2, \ldots, G_k$形成对样本集合X的划分，其中$G_i \cap G_j = \emptyset, \bigcup_{i=1}^{k} G_i = D$ 。



k-means算法

$k-means$算法将欧氏距离平方作为样本间距离

策略

给定样本集$D=\{x_1,x_2,\ldots,x_m\}$, “k均值”$(k-means)$算法针对聚类所得簇划分$C=\{C_1,C_2,\ldots,C_k\}$最小化平方误差

$$E=\sum_{i=1}^{k}\sum_{x\in C_i}\|x-\mu_i\|_2^2$$

该式是K-means算法的目标函数

其中$\mu_{\mathrm{i}}=\frac{1}{|C_i|}\sum_{x\in C_i}x$是簇$C_i$的均值向量。

在 k-means 聚类中，算法的策略就是最小化这个误差平方和，使得同一个簇内的点彼此尽量相似（距离簇中心更近）。

k -means算法

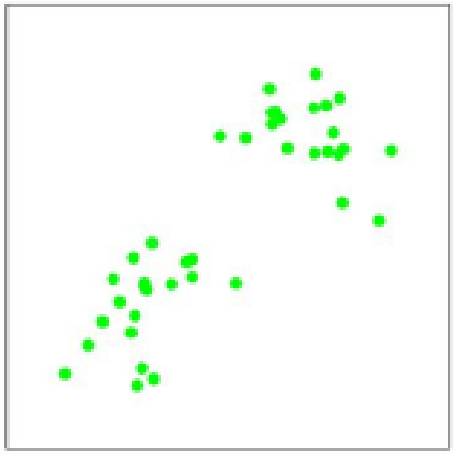
算法步骤

1. 选择初始化的 k 个样本作为初始聚类中心 $a = a_1, a_2, \ldots, a_k$;
2. 针对数据集中每个样本x_i，计算它到这 k个聚类中心的距离并将其分到距离最小的聚类中心所对应的类中;
3. 针对每个类别a_j，重新计算它的聚类中心（即属于该类的所有样本的质心）;
4. 重复上面 2 3 两步操作，直到达到某个中止条件（迭代次数、最小误差变化等）。

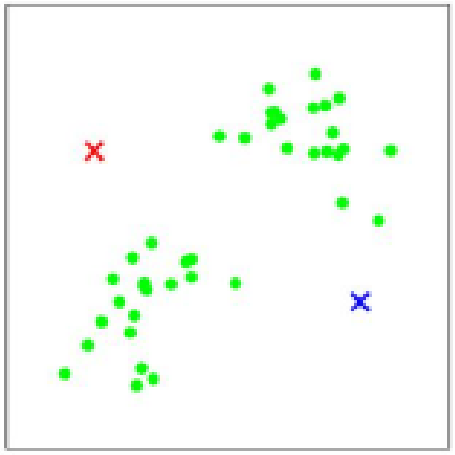
k-means算法

算法例

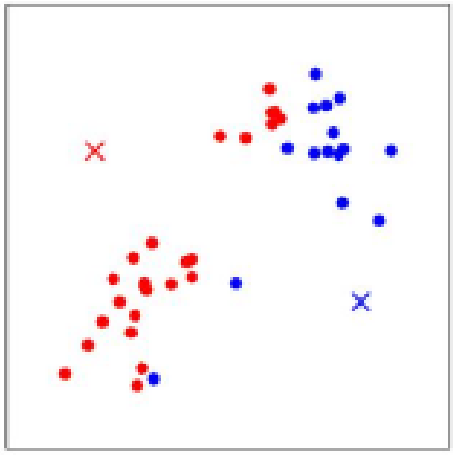
- 假设k=2。



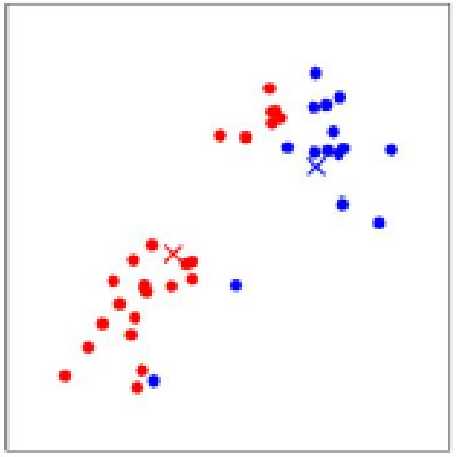
(a)



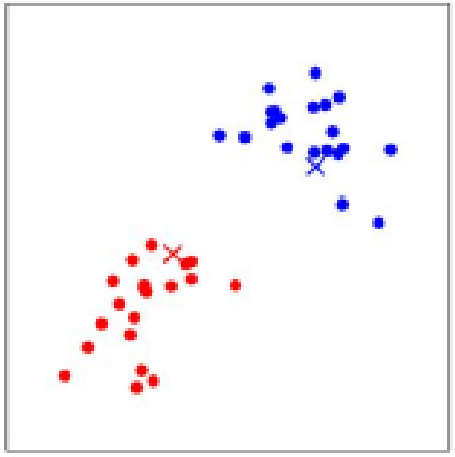
(b)



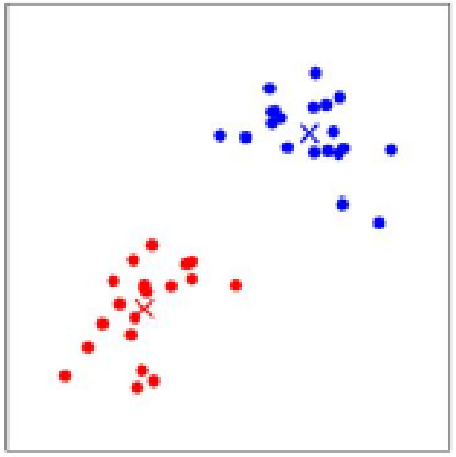
(c)



(d)



(e)



(f)

k-means算法-例

给定含有5个样本的集合

$$X = \begin{bmatrix} 0 & 0 & 1 & 5 & 5 \\ 2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

试用k均值聚类算法将样本聚到2个类中

质心	距离（欧式距离平方）				
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (0,2)$	0	4	5	29	25
$x_{c2} = (0,0)$	4	0	1	25	29

$$\Rightarrow \quad \begin{matrix} G_1 = \{x_1, x_5\} \\ G_2 = \{x_2, x_3, x_4\} \end{matrix} \quad \Rightarrow \quad \begin{matrix} x_{c1} = (2.5,2) \\ x_{c2} = (2,0) \end{matrix}$$

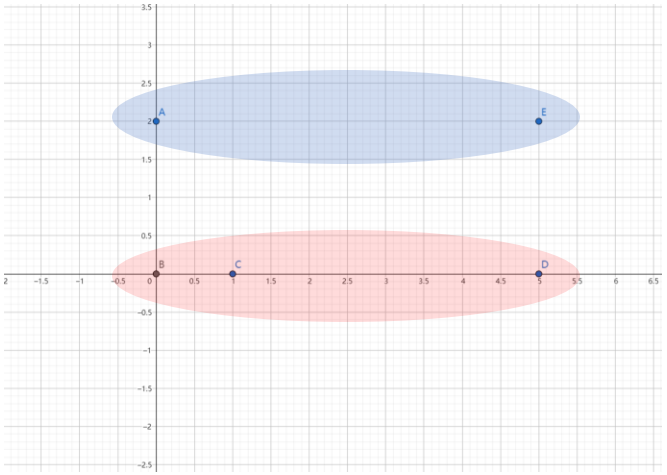
k-means算法-例

质心	距离（欧式距离平方）				
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (2.5,2)$	6.25	10.25	4.25	10.25	6.25
$x_{c2} = (2,0)$	8	4	1	9	13

$$\Rightarrow \quad \begin{matrix} G_1 = \{x_1, x_5\} \\ G_2 = \{x_2, x_3, x_4\} \end{matrix} \quad \Rightarrow \quad \begin{matrix} x_{c1} = (2.5,2) \\ x_{c2} = (2,0) \end{matrix}$$

由于得到的新的类没有改变，聚类停止。得到最终的聚类结果：

$$\begin{matrix} G_1 = \{x_1, x_5\} \\ G_2 = \{x_2, x_3, x_4\} \end{matrix}$$



k-means算法-例

给定含有5个样本的集合

$$X = \begin{bmatrix} 0 & 0 & 1 & 5 & 5 \\ 2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

试用k均值聚类算法将样本聚到2个类中

质心	距离（欧式距离平方）				
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (0,2)$					
$x_{c2} = (5,2)$					

k-means算法-例

第一次迭代

质心	距离（欧式距离平方）				
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (0,2)$	0	4	5	29	25
$x_{c2} = (5,2)$	25	29	20	4	0

$$\Rightarrow \quad \begin{matrix} G_1 = \{x_1, x_2, x_3\} \\ G_2 = \{x_4, x_5\} \end{matrix} \quad \Rightarrow \quad \begin{matrix} x_{c1} = (0.33, 0.67) \\ x_{c2} = (5,1) \end{matrix}$$

第二次迭代

质心	距离（欧式距离平方）				
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (0.33,\ 0.67)$	0.19	0.56	0.89	22.22	23.56
$x_{c2} = (5,1)$	26	26	17	1	1

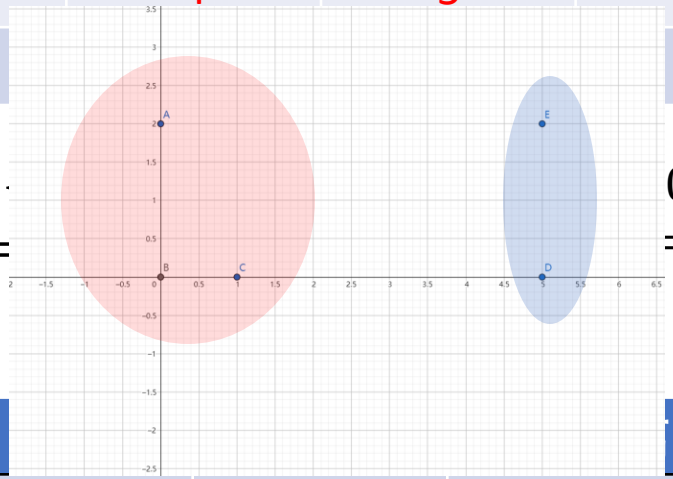
$$\Rightarrow \quad \begin{matrix} G_1 = \{x_1, x_2, x_3\} \\ G_2 = \{x_4, x_5\} \end{matrix}$$

同样一组数据，因为初始点的不同，导致聚类结果不一样

k-means算法-例

第一次迭代

质心	距离（欧式距离平方）				
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (0,2)$	0	4	5	29	25
$x_{c2} = (5,2)$	25			4	0



$G_1 =$ $0.33,0.67)$

$G_2 =$ $= (5,1)$

第二次迭代

质心		平方）			
	$x_1 = (0,2)$	$x_2 = (0,0)$	$x_3 = (1,0)$	$x_4 = (5,0)$	$x_5 = (5,2)$
$x_{c1} = (0.33,\ 0.67)$	0.19	0.56	0.89	22.22	23.56
$x_{c2} = (5,1)$	26	26	17	1	1

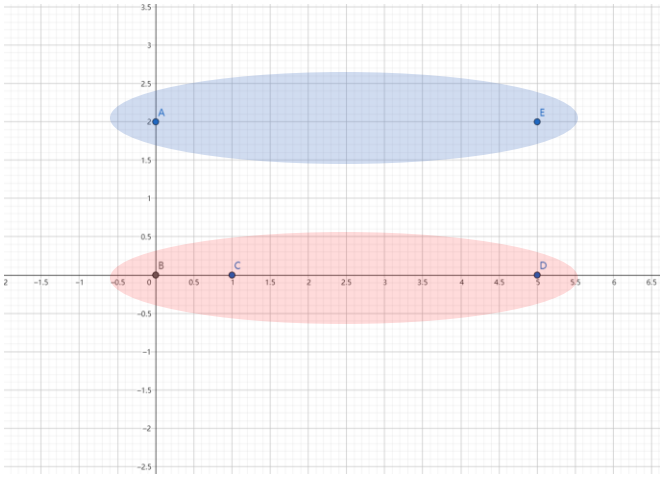
$G_1 = \{x_1, x_2, x_3\}$

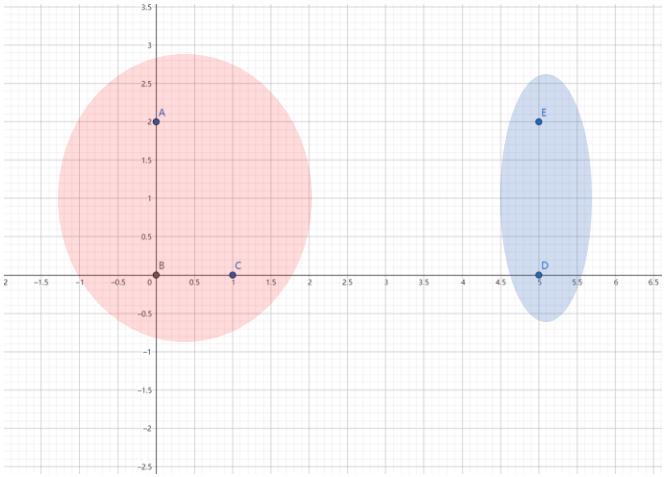
$G_2 = \{x_4, x_5\}$

同样一组数据，因为初始点的不同，导致聚类结果不一样

初始质心的选择

k均值聚类算法在每次迭代中都是寻求局部最优解。换句话说，算法会沿着初始质心附近的方向收敛，所以不同的初始质心可能导致收敛到不同的局部最优解，从而影响最终的聚类效果。

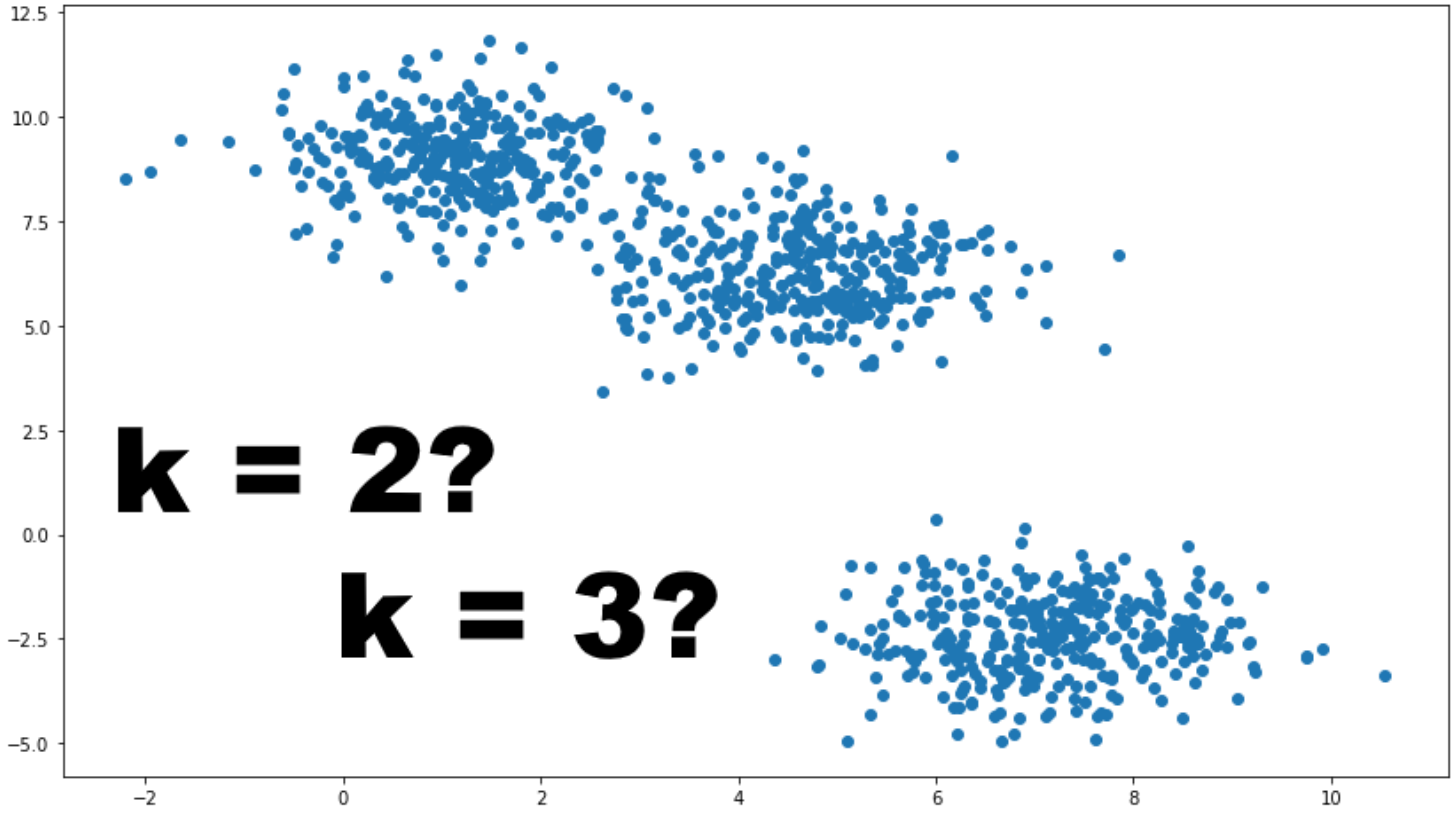




同样的一组数据，因为初始中心选择的不同，得到不同的聚类结果

k值的选择

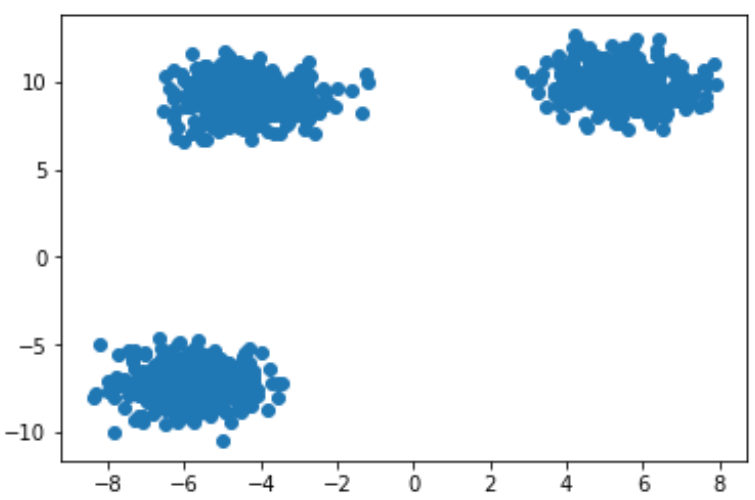
- K的取值需要事先确定，然而在无监督聚类任务上，由于并不知道数据集究竟有多少类别，所以很难确定合适的K的取值。

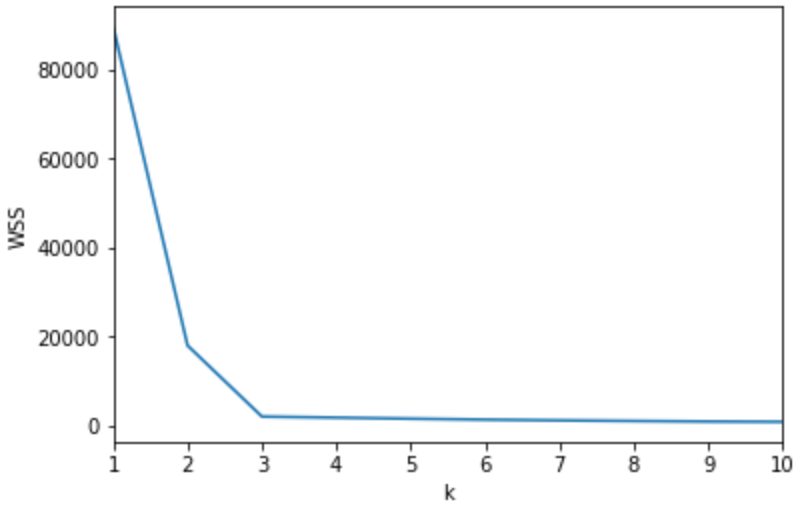


k值的选择

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||_2^2$$

- 能量函数法选择K值（肘部法）





- 分别令 k取不同的值（例如 2、3、4、...），运行 k-means，得到不同的聚类结果，每一个聚类结果都对应一个E值。
- 随着 k增大，E会显著下降，但当 k增大到一定程度后，E的下降幅度开始变小。
- 在这个“变化曲线”上寻找一个拐点（“肘部”），该拐点往往对应一个相对合理的 k 值。图中当K大于3时，E值就不怎么降低了，所以K=3是一个合理的K值。

k-means算法特性

缺点：

1. **要确定K的值**：K-Means需要事先确定聚类数目K，这在无监督聚类任务中可能会很困难，因为通常无法事先知道数据集应该被分为多少个类别。
2. **对异常点敏感**：K-Means很容易受到异常点（离群值）的影响，因为它使用簇内样本均值来更新质心，异常点可能会导致质心偏移。
3. **不适合非球形数据集**：K-Means 算法之所以更适合“球形”分布的簇，主要是因为它的聚类过程和使用的度量方式（欧几里得距离）使得每个簇的边界都呈现“离中心最近”的形状，这往往对应**近似球形**的区域。这种天然倾向于得到球形或相对均匀的簇的特点，使得K-means对于明显的非球形的数据集表现不佳。

k-means算法特性

优点：

1. **简单易实现**
   算法原理直观、流程清晰，便于编程实现和理解。
2. **计算速度快**
   由于每次迭代只涉及数据点与聚类中心的距离计算，算法在处理大规模数据集时能快速收敛。
3. **适用于大规模数据**
   k-means 的计算复杂度较低（通常为线性增长），因此能有效处理大数据集，具有良好的扩展性。

降维

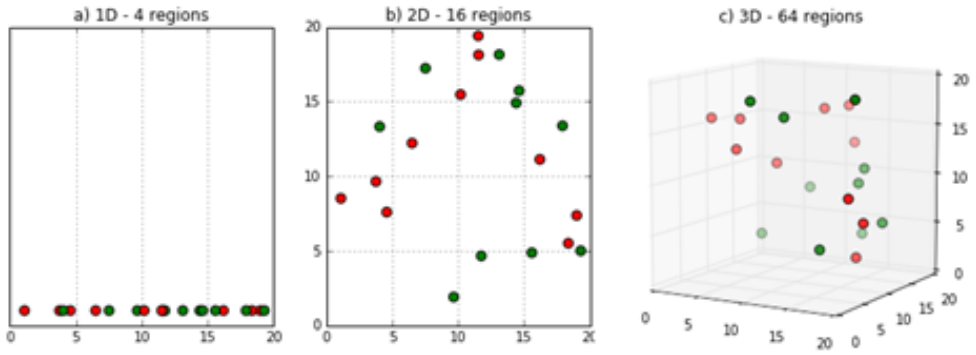
为什么需要降维?

- 数据压缩
- **维数灾难**

  -降低时间复杂度,

  -降低空间复杂度

  -模型有更好的可解释性
- 去噪声
- 可视化



curse of dimensionality

降维方法的分类

线性的

- **principle component analysis (PCA)**
- single value decomposition (SVD)

非线性的

- **MDS**
- **Isomap**
- **Locally linear embedding**
- t-SNE/SNE
- **Autoencoder** （基于神经网络的）
- Kernel PCA

…

其它的分类：带**少量参数**的/带有**大量参数**的/**不带参数**的
可**重建**的/**不可重建**的

降维的指导思想

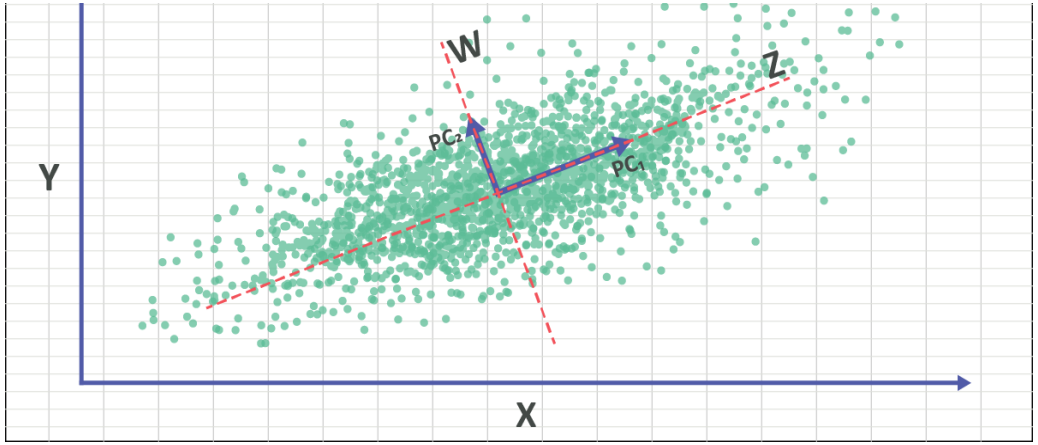
降维方法的主要目标是将高维空间中的信息映射到低维空间，以减少维度，同时尽量保持原始数据的关键特征。

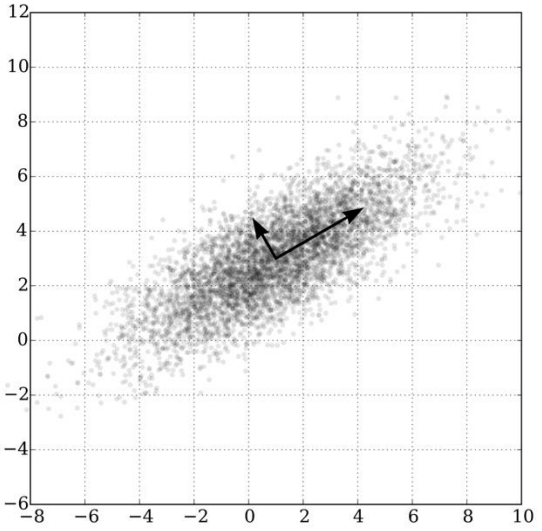
不同的降维方法在实现这一目标时有各自的侧重点：

- PCA（主成分分析）最大程度地**保留原始数据在各个方向上的方差**。
- MDS（多维缩放）在低维空间中**保持原始数据点之间的欧氏距离**。
- Isomap试图**保持原始数据空间中低维流形上的距离关系**。
- LLE（局部线性嵌入）则注重**保留局部数据点之间的线性关系**。
- t-SNE（t-分布随机邻域嵌入）在低维空间中**保持原始数据点之间的局部相似性关系**。
- 。。。

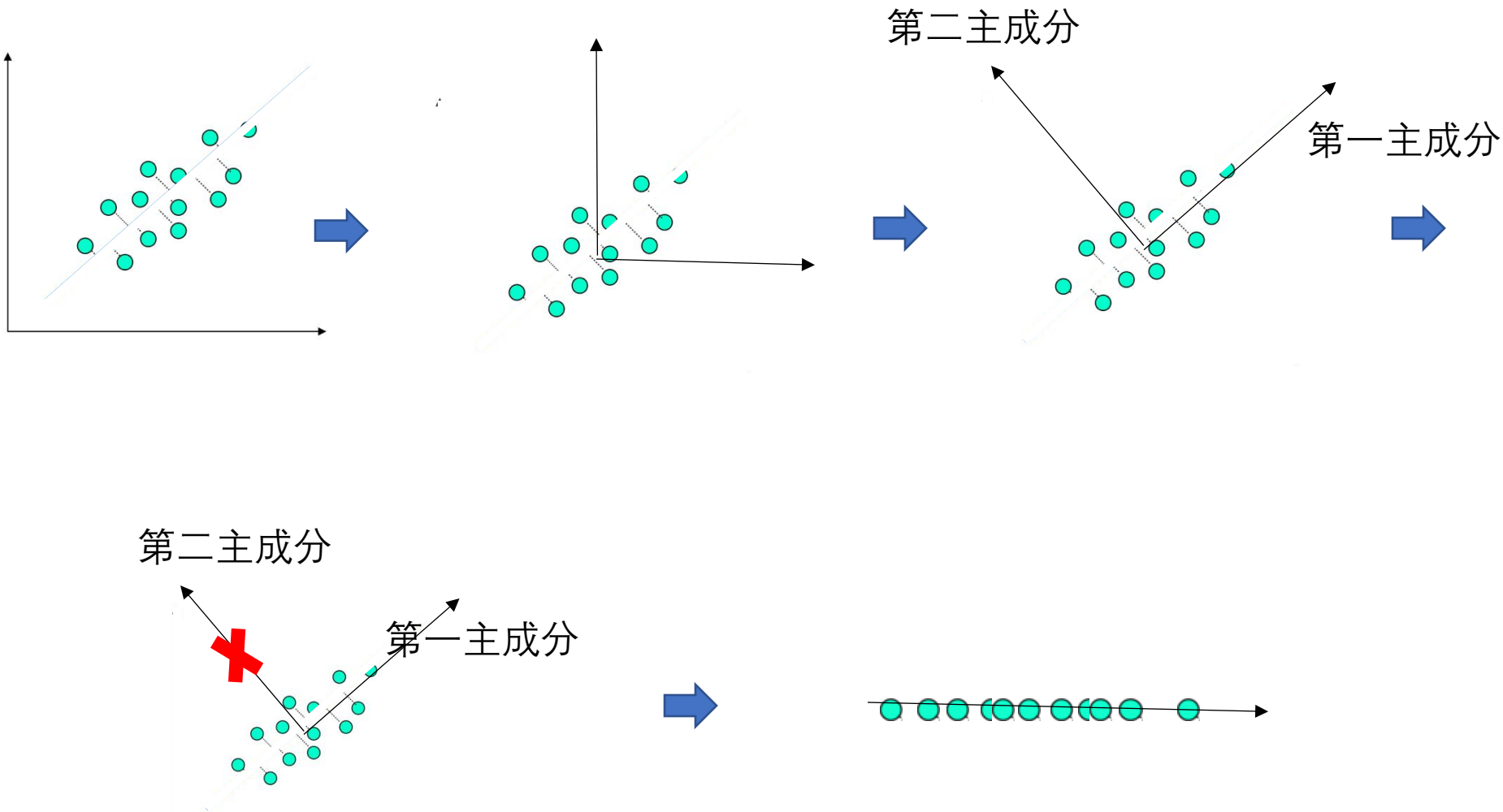
线性降维（PCA）

- 线性降维是通过对原始特征空间的坐标系进行**线性变换**，将数据映射到一个低维子空间中，从而减少特征的数量而保留尽可能多的信息。
- 线性降维可以理解为坐标系做旋转，平移（线性变换），然后去掉一些无关紧要的坐标轴（信息量比较少的那些坐标轴）来达到降低维度的目的。
- PCA是线性降维的代表性方法之一，它的优化目标是**找到一组新的基底（主成分），使得数据在这些基底上的投影方差最大**。
- 第一主成分，第二主成分，。。。





线性降维（PCA）



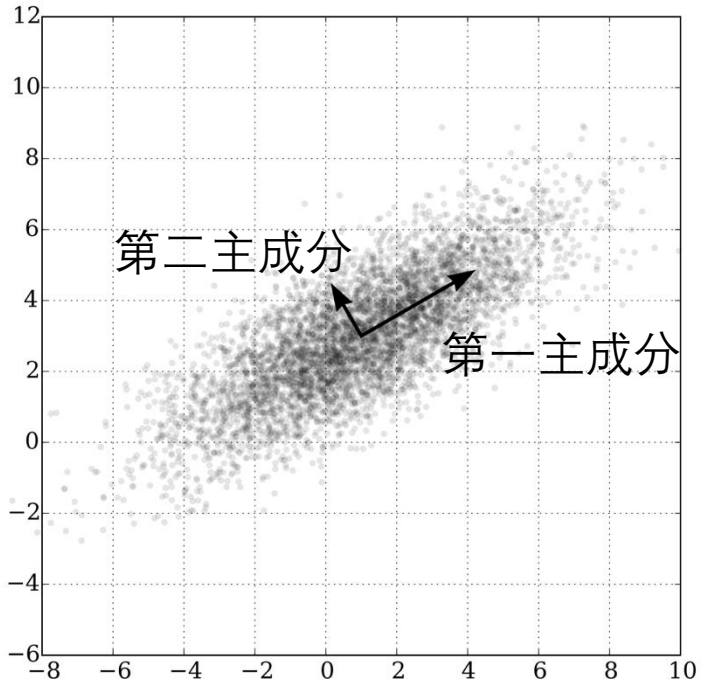
线性降维(PCA)

1.协方差矩阵的计算：为了找到主成分，首先需要计算原始数据的协方差矩阵。

2.特征值分解：通过对协方差矩阵进行特征值分解来找到主成分，以及它们对应的特征值。

3.特征值排序：一旦特征值和对应的主成分被计算出来，通常会按特征值的大小进行排序，以选择最重要的主成分。这可以帮助确定要保留的维度数量。

4.投影操作：PCA的最后一步是将数据投影到选定的主成分上，从而实现维度降低。

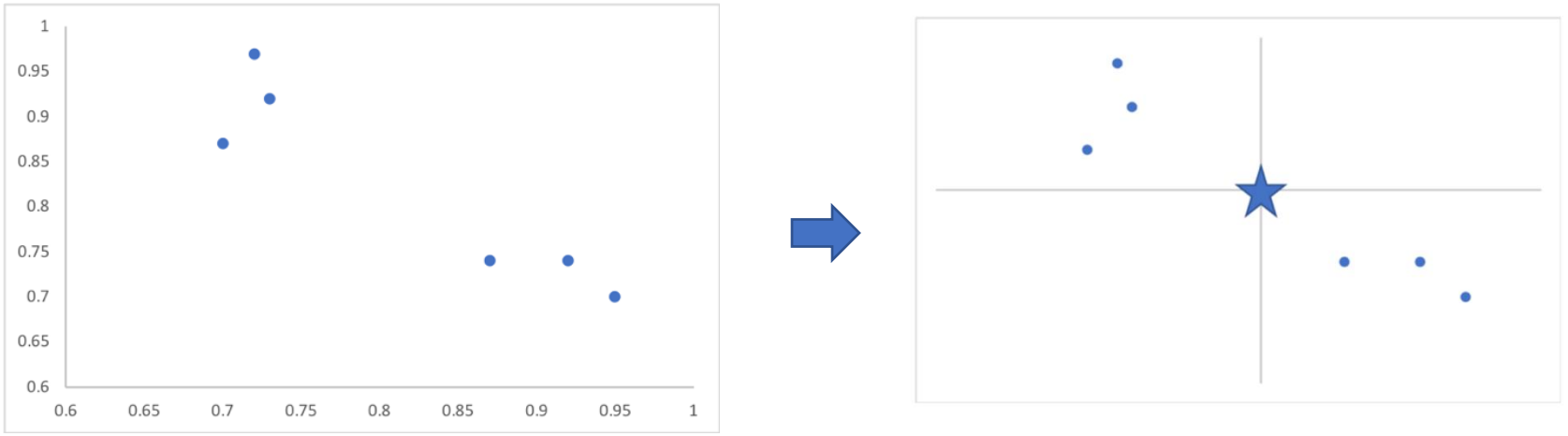


线性降维(PCA)

Step 1:

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN} \end{bmatrix} \qquad \bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{i=n} \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN} \end{bmatrix}$$

$$X = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix}$$



线性降维(PCA)

Step 2:

$$Q = XX^T = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix} \begin{bmatrix} (\mathbf{x}_1 - \bar{\mathbf{x}})^T \\ (\mathbf{x}_2 - \bar{\mathbf{x}})^T \\ \vdots \\ (\mathbf{x}_n - \bar{\mathbf{x}})^T \end{bmatrix}$$

备注:
1.Q是方阵。
2.Q是对称矩阵。
3.Q是协方差矩阵。

线性降维(PCA)

Step 3:

- 计算矩阵Q的特征向量和特征值。其中，最大的特征值对应的特征向量就是PCA中的第一主成分（对应着数据方差变化最大的方向），第二大的特征向量对应着第二主成分，以此类推。
- 如果希望将数据降维到n维，可以选择前n个特征向量作为数据投影的基底向量。通过投影数据点到这些基底向量上，就可以实现数据的降维操作。

线性降维(PCA)

例子：

x	1	1	2	2	4
y	-1	1	1	2	2

该数据集的协方差矩阵为$\begin{bmatrix} 6 & 4 \\ 4 & 6 \end{bmatrix}$

求矩阵特征值：λ = 10，λ = 2，分别对应的特征向量：$\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$　$\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$

第一主元　　第二主元

非线性降维

Multidimensional Scaling (MDS)

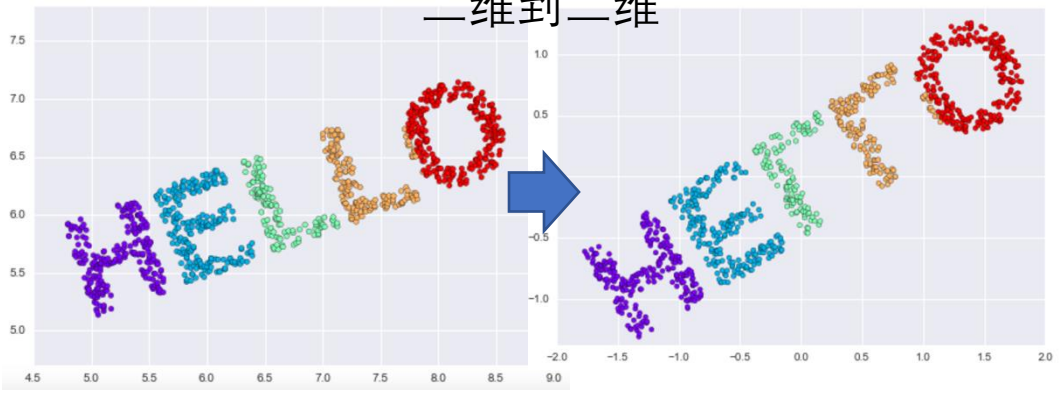
给定一个**距离矩阵**（包含数据集中两两样本之间的欧式距离）和一个预先选定的**维度*N***

MDS

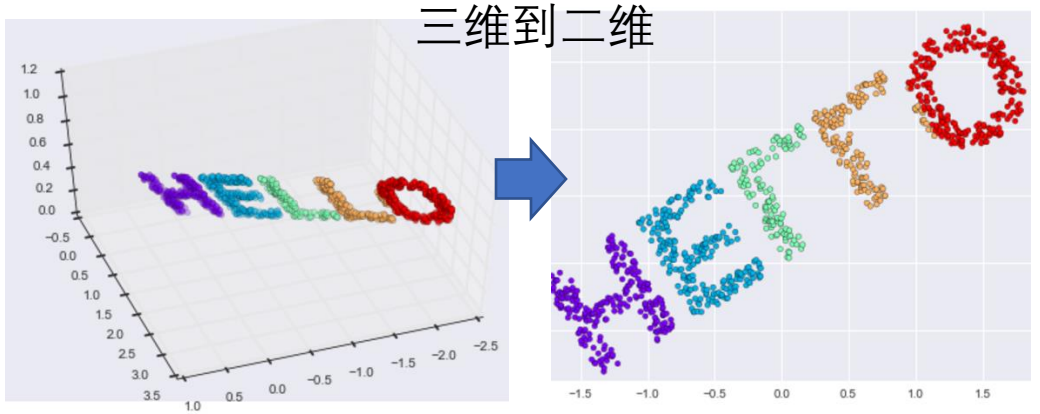
MDS将每一个样本放到***N* 维空间***中，并尽可能地保证样本两两之间的**距离不变**

例子：

二维到二维



三维到二维



Multidimensional Scaling (MDS)

从Distance matrix 开始

$$D := \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,M} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,M} \\ \vdots & \vdots & & \vdots \\ d_{M,1} & d_{M,2} & \cdots & d_{M,M} \end{pmatrix}.$$

The goal of MDS is, given D, to find M vectors $x_1, \ldots, x_M \in \mathbb{R}^N$ such that

$$\|x_i - x_j\| \approx d_{i,j} \text{ for all } i, j \in 1, \ldots, M,$$

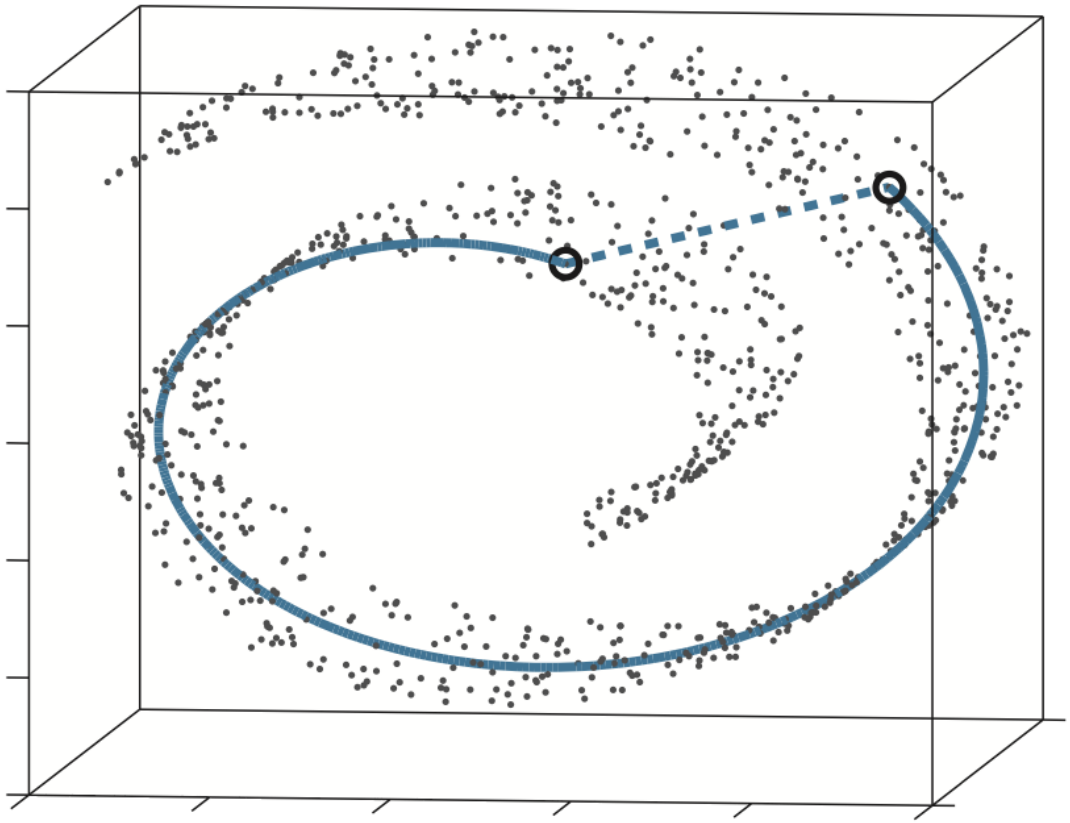
转化为最优化问题 ⟹ 数值方法求解：

Cost function： $$\min_{x_1,\ldots,x_M} \sum_{i<j} (\|x_i - x_j\| - d_{i,j})^2.$$

Isomap

- 高维空间中两点的欧氏距离有时候不能反映数据的内在关系
- 流形上的geodesic distance （测地距离）可以更好的反应数据两两间的距离

A



- 测地距离（Geodesic Distance）是描述在曲面或流形上两点之间最短路径的距离。。

Isomap

算法的步骤

Step1： 在流形上构建一个**nearest neighbor graph**

Step 2: 计算graph上两点之间最短距离（geodesic distance），构建一个**距离矩阵**

Step 3: 利用MDS方法映射到**低维空间**，**保留距离信息**