目录

降维

问题描述

【参考】

- csdn - 主成分分析（PCA）原理详解
- 知乎 - EYD与机器学习八：Dimensionality Reduction
- A Tutorial on Principal Component Analysis

在实践中我们经常会遇到含有大量属性的数据集，即数据集的维度特别的高，而且有些属性还有一定的关联性，这就导致分析起来特别的麻烦。我们可以只选择其中的一些属性进行分析，但这些属性该如何选择呢，不能拍拍脑子就决定用那些属性哪些不用，我们需要一个工具能够找出最主要的属性。这时候我们就需要找到一种方法，它能够降低数据的维度，但又不会失去数据集数据的信息。常用的降维方法可以分为两大类：`映射（projection）`和`流行学习（Manifold Learning）`。

映射方法中最常用的就是主成分分析法，主成分分析（Principal Component Analysis PCA）是最常用的一种降维方法。

下表1是某些学生的语文、数学、物理、化学成绩统计：

表 1

学生编号	语文	数学	物理	化学
1	90	140	99	100
2	90	97	88	92
3	90	110	79	83
. . .	. . .	. . .	. . .	. . .

首先，假设这些科目成绩不相关，也就是说某一科目考多少分与其他科目没有关系。那么一眼就能看出来，数学、物理、化学这三门课的成绩构成了这组数据的主成分（很显然，数学作为第一主成分，因为数学成绩拉的最开）。为什么一眼能看出来？因为坐标轴选对了！下面再看一组学生的数学、物理、化学、语文、历史、英语成绩统计，见表2，还能不能一眼看出来：

表 2

学生编号	数学	物理	化学	语文	历史	英语
1	65	61	72	84	81	79
2	77	77	76	64	70	55
3	67	63	49	65	67	57
4	80	69	75	74	74	63
5	74	70	80	84	82	74
6	78	84	75	62	72	64
7	66	71	67	52	65	57
8	77	71	57	72	86	71
9	83	100	79	41	67	50
. . .	. . .	. . .	. . .	. . .	. . .	. . .

数据太多了，以至于看起来有些凌乱！也就是说，无法直接看出这组数据的主成分，因为在坐标系下这组数据分布的很散乱。如果把这些数据在相应的空间中表示出来，也许你就能换一个观察角度找出主成分。如下图1所示：

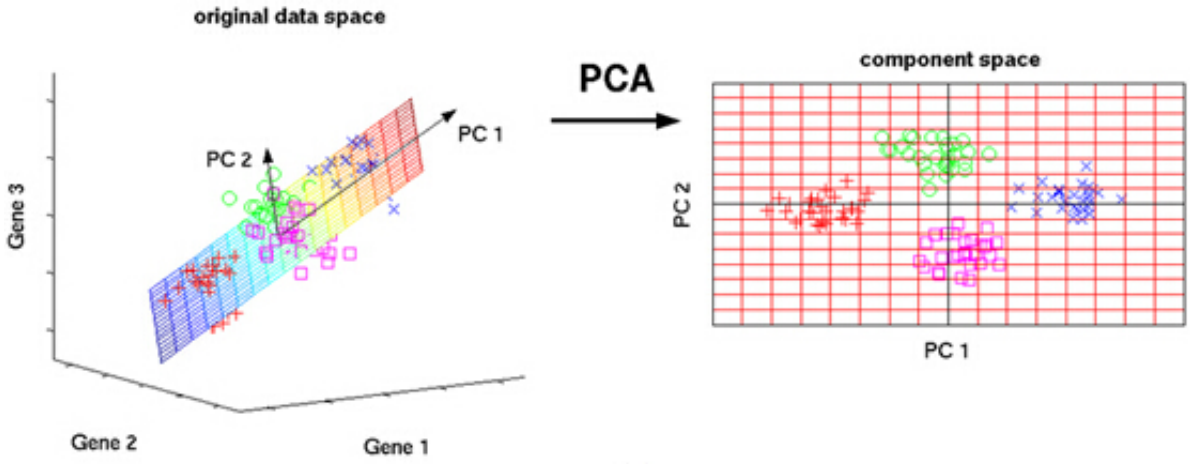
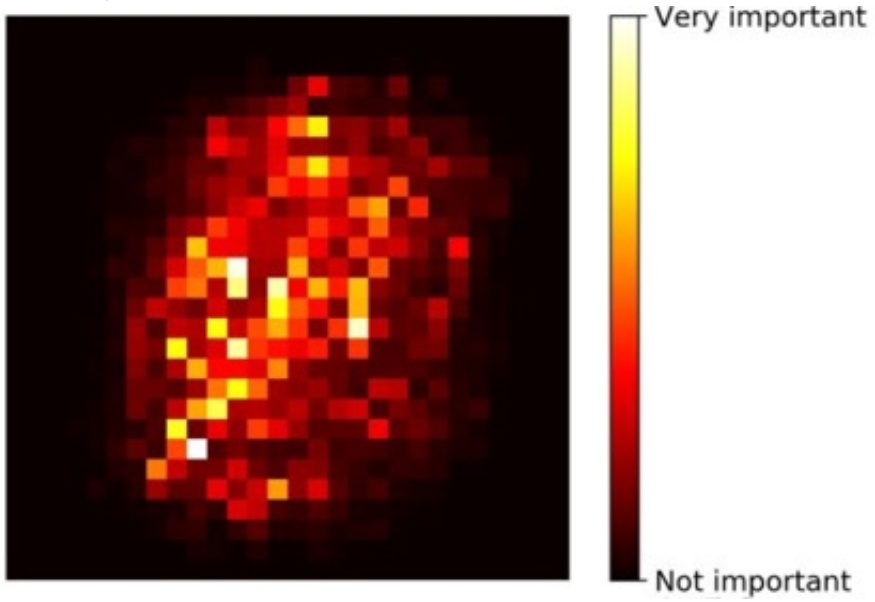


图 1

再如MNIST数据集，每个样本拥有$28 * 28 = 784$维特征，但并不是每一维对于我们的分类目标都是重要的。下图展示了样本中各像素点对分类任务的重要性，显然，图片边缘的大部分像素点都对分类问题贡献甚微，我们可以通过降维去除这部分像素点的特征：



但是，对于更高维的数据，能想象其分布吗？就算能描述分布，如何精确地找到这些主成分的轴？如何衡量你提取的主成分到底占了整个数据的多少信息？所以，我们就要用到主成分分析的处理方法。

线性代数基础

【参考】

- PCA的数学原理

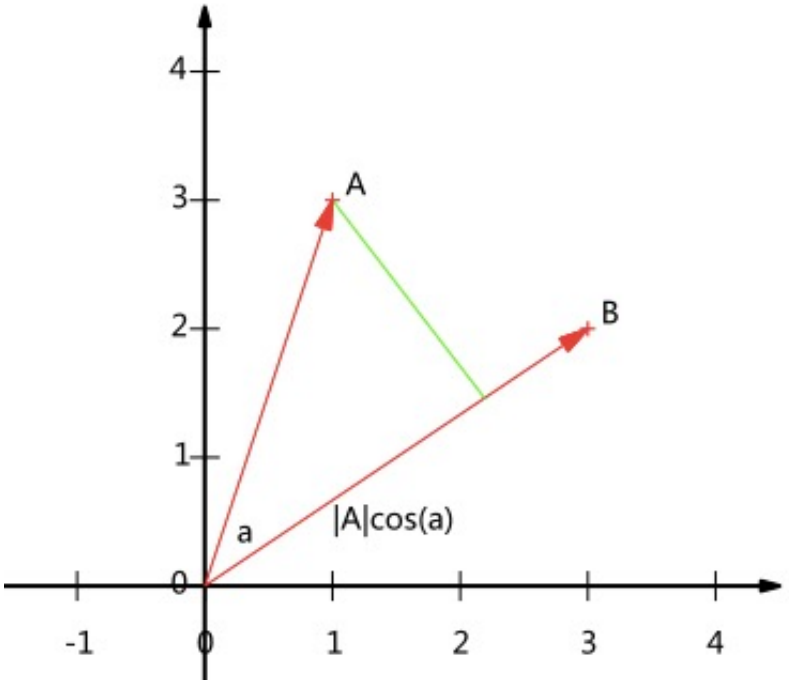
PCA的思想是将n维特征映射到k维上（k<n），这k维是全新的正交特征。这k维特征称为主成分，是重新构造出来的k维特征，而不是简单地从n维特征中去除其余n-k维特征。

推导需要用到线性代数相关的基础知识

内积与投影

内积定义为：$(a_1, a_2, \cdots, a_n)^{\mathsf{T}} \cdot (b_1, b_2, \cdots, b_n)^{\mathsf{T}} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

内积运算将两个向量映射为一个实数。他的几何意义如下，令$A = (x_1, y_1), B = (x_2, y_2)$，在二维平面上的表示如下：



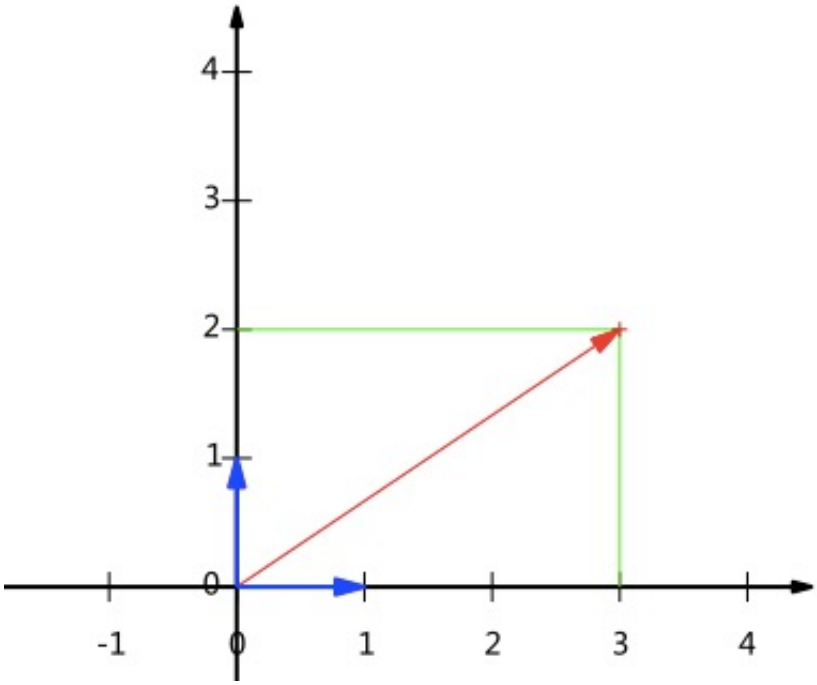
内积哈可以表示为如下形式：$A \cdot B = |A||B| \cos a$

如果令 B 的模长为 1 的话，那么上式可以写为：$A \cdot B = |A| \cos a$ 也就是说，设向量B的模为1，则A与B的内积值等于A向B所在直线投影的矢量长度。

基

以二维空间为例，其中的任何一个向量都可以表示为单位向量的线性组合：$x(1,0)^{\mathsf{T}} + y(0,1)^{\mathsf{T}}$

其中的(1,0)，(0,1) 就是基向量，如下图：



即向量(3,2) 可以表示为：$3(1,0)^{\mathsf{T}} + 2(0,1)^{\mathsf{T}}$。

所以，**要准确描述向量，首先要确定一组基，然后给出在基所在的各个直线上的投影值，就可以了**。只不过我们经常省略第一步，而默认以(1,0)和(0,1)为基。

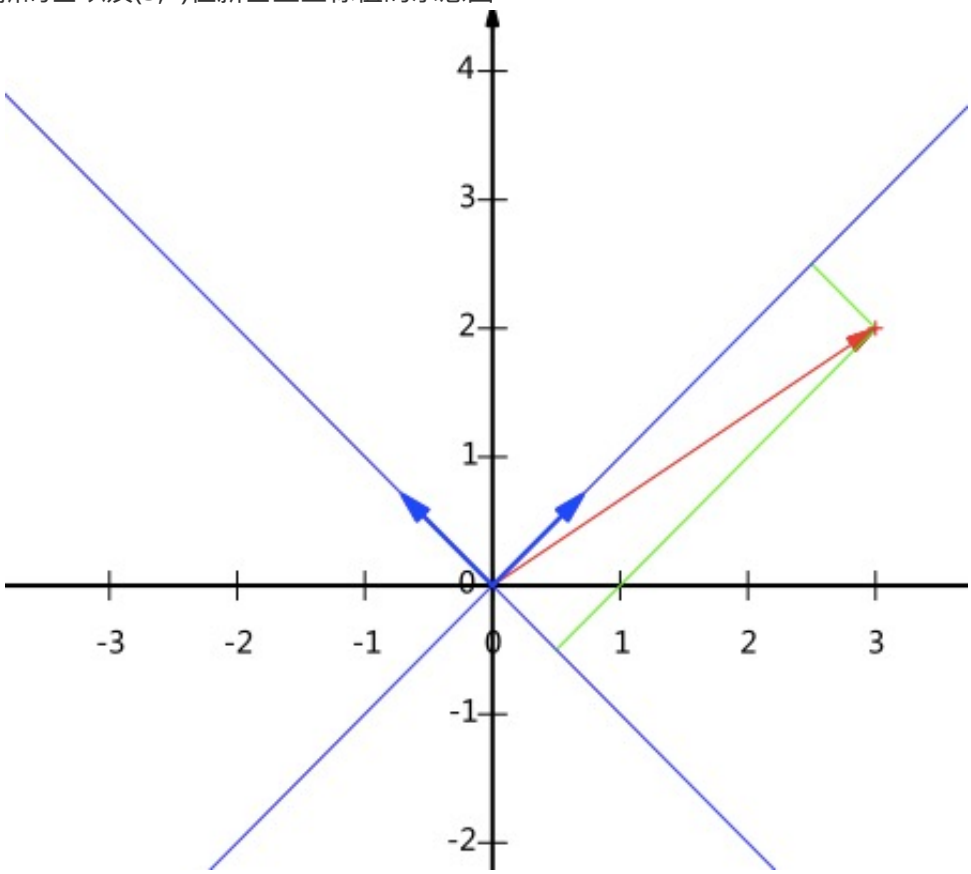
我们之所以默认选择(1,0)和(0,1)为基，当然是比较方便，因为它们分别是x和y轴正方向上的单位向量，因此就使得二维平面上点坐标和向量一一对应，非常方便。但实际上任何两个线性无关的二维向量都可以成为一组基，所谓线性无关在二维平面内可以直观认为是两个不在一条直线上的向量。

例如，(1,1)和(-1,1)也可以成为一组基。一般来说，我们希望基的模是1，因为从内积的意义可以看到，如果基的模是1，那么就可以方便的用向量点乘基而直接获得其在新基上的坐标了！实际上，对应任何一个向量我们总可以找到其同方向上模为1的向量，只要让两个分量分别除以模就好了。例如，上面的基可以变为$(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})$和$(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})$。

现在，我们想获得(3,2)在新基上的坐标，即在两个方向上的投影矢量值，那么根据内积的几何意义，我们只要分别计算(3,2)和两个基的内积，不难得到新的坐标为$(\frac{5}{\sqrt{2}},-\frac{1}{\sqrt{2}})$。

$$(3,2)(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})^T=\frac{5}{\sqrt{2}}$$
$$(3,2)(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})^T=-\frac{1}{\sqrt{2}}$$

下图给出了新的基以及(3,2)在新基上坐标值的示意图：



另外这里要注意的是，我们列举的例子中基是正交的（即内积为0，或直观说相互垂直），但可以成为一组基的唯一要求就是线性无关，非正交的基也是可以的。不过因为正交基有较好的性质，所以一般使用的基都是正交的。

基变换的矩阵表示

上面的(3,2)变换还可以使用矩阵的形式表示：

$$\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 5/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$$

其中矩阵的两行分别为两个基，乘以原向量，其结果刚好为新基的坐标。可以稍微推广一下，如果我们有m个二维向量，只要将二维向量按列排成一个两行m列矩阵，然后用“基矩阵”乘以这个矩阵，就得到了所有这些向量在新基下的值。例如(1,1)，(2,2)，(3,3)，想变换到刚才那组基上，则可以这样表示：

$$\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 2/\sqrt{2} & 4/\sqrt{2} & 6/\sqrt{2} \\ 0 & 0 & 0 \end{pmatrix}$$

于是一组向量的基变换被干净的表示为矩阵的相乘。

一般的，如果我们有M个N维向量，想将其变换为由R个N维向量表示的新空间中，那么首先将R个基按行组成矩阵A，然后将向量按列组成矩阵B，那么两矩阵的乘积AB就是变换结果，其中AB的第m列为A中第m列变换后的结果。

数学表示为：

$$\begin{pmatrix} \vec{\boldsymbol{p}}_1 \\ \vec{\boldsymbol{p}}_2 \\ \vdots \\ \vec{\boldsymbol{p}}_R \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_M \end{pmatrix} = \begin{pmatrix} \vec{\boldsymbol{p}}_1\boldsymbol{a}_1 & \vec{\boldsymbol{p}}_1\boldsymbol{a}_2 & \cdots & \vec{\boldsymbol{p}}_1\boldsymbol{a}_M \\ \vec{\boldsymbol{p}}_2\boldsymbol{a}_1 & \vec{\boldsymbol{p}}_2\boldsymbol{a}_2 & \cdots & \vec{\boldsymbol{p}}_2\boldsymbol{a}_M \\ \vdots & \vdots & \ddots & \vdots \\ \vec{\boldsymbol{p}}_R\boldsymbol{a}_1 & \vec{\boldsymbol{p}}_R\boldsymbol{a}_2 & \cdots & \vec{\boldsymbol{p}}_R\boldsymbol{a}_M \end{pmatrix}$$

其中p_i是一个行向量，表示第i个基，a_j是一个列向量，表示第j个原始数据记录。

特别要注意的是，这里R可以小于N，而R决定了变换后数据的维数。也就是说，我们可以将一N维数据变换到更低维度的空间中去，变换后的维度取决于基的数量。因此这种矩阵相乘的表示也可以表示降维变换。

最后，上述分析同时给矩阵相乘找到了一种物理解释：**两个矩阵相乘的意义是将右边矩阵中的每一列列向量变换到左边矩阵中每一行行向量为基所表示的空间中去。**

协方差矩阵及优化目标

上面我们讨论了选择不同的基可以对同样一组数据给出不同的表示，而且如果基的数量少于向量本身的维数，则可以达到降维的效果。但是我们还没有回答一个最最关键的问题：如何选择基才是最优的。或者说，如果我们有一组N维向量，现在要将其降到K维（K小于N），那么我们应该如何选择K个基才能最大程度保留原有的信息？

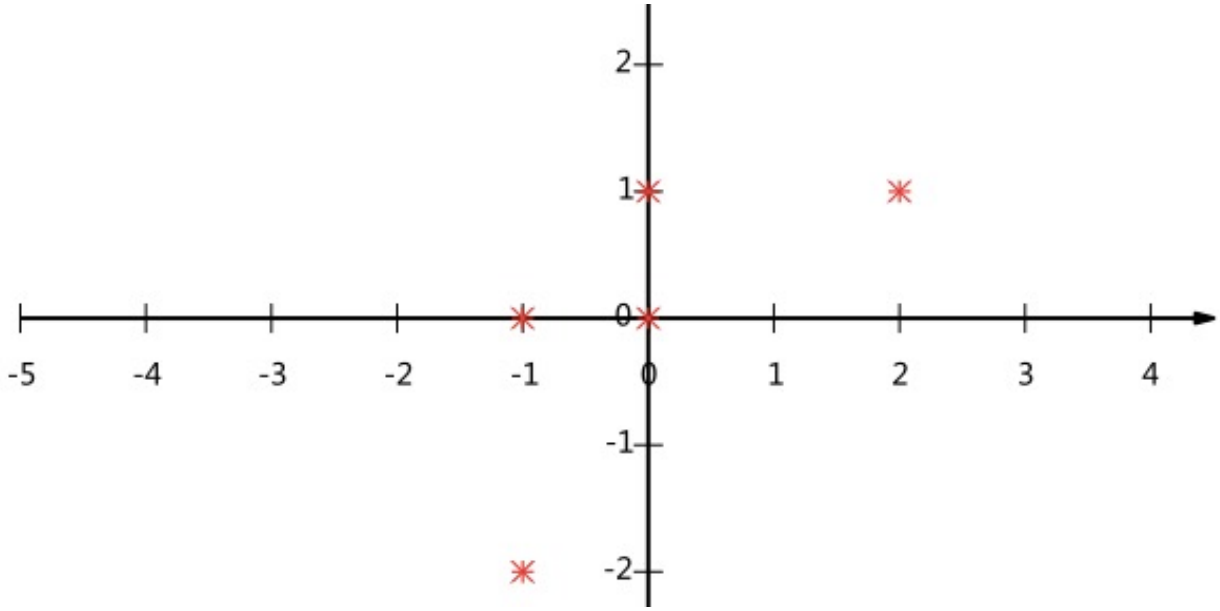
为了避免过于抽象的讨论，我们仍以一个具体的例子展开。假设我们的数据由五条记录组成，将它们表示成矩阵形式：

$$\begin{pmatrix} 1 & 1 & 2 & 4 & 2 \\ 1 & 3 & 3 & 4 & 4 \end{pmatrix}$$

其中每一列为一条数据记录，而一行为一个字段。为了后续处理方便，我们首先将每个**字段内**所有值都减去字段均值，其结果是将每个字段都变为均值为0（这样做的道理和好处后面会看到）。我们看上面的数据，第一个字段均值为2，第二个字段均值为3，所以变换后：

$$\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$$

我们可以看下五条数据在平面直角坐标系内的样子：



现在问题来了：如果我们必须使用一维来表示这些数据，又希望尽量保留原始的信息，你要如何选择？

通过上一节对基变换的讨论我们知道，这个问题实际上是要在二维平面中选择一个方向，将所有数据都投影到这个方向所在直线上，用投影值表示原始记录。这是一个实际的二维降到一维的问题。

那么如何选择这个方向（或者说基）才能尽量保留最多的原始信息呢？一种直观的看法是：**希望投影后的投影值尽可能分散**。

以上图为例，可以看出如果向x轴投影，那么最左边的两个点会重叠在一起，中间的两个点也会重叠在一起，于是本身四个各不相同的二维点投影后只剩下两个不同的值了，这是一种严重的信息丢失，同理，如果向y轴投影最上面的两个点和分布在x轴上的两个点也会重叠。所以看来x和y轴都不是最好的投影选择。我们直观目测，如果向通过第一象限和第三象限的斜线投影，则五个点在投影后还是可以区分的。

方差

【参考】

- 方差var、协方差cov、协方差矩阵（浅谈）-（一）

在信号处理中认为信号具有较大的方差，噪声有较小的方差，信噪比就是信号与噪声的方差比，越大越好。

上文说到，我们希望投影后投影值尽可能分散，而这种**分散程度**，可以用数学上的方差来表述。此处，一个字段的方差可以看做是每个元素与字段均值的差的平方和的均值，即：

$$Var(a) = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu)^2$$

由于上面我们已经将每个字段的均值都化为0了，因此方差可以直接用每个元素的平方和除以元素个数表示：

$$Var(a) = \frac{1}{m}\sum_{i=1}^{m}a_i^2$$

于是上面的问题被形式化表述为：**寻找一个一维基，使得所有数据变换为这个基上的坐标表示后，方差值最大**。

协方差

在概率论和统计学中，协方差用于衡量两个变量的总体误差，是一个衡量线性独立的无量纲的数。

对于上面二维降成一维的问题来说，找到那个使得方差最大的方向就可以了。不过对于更高维，还有一个问题需要解决。考虑三维降到二维问题。与之前相同，首先我们希望找到一个方向使得投影后方差最大，这样就完成了第一个方向的选择，继而我们选择第二个投影方向。

如果我们还是单纯只选择方差最大的方向，很明显，这个方向与第一个方向应该是“几乎重合在一起”，显然这样的维度是没有用的，因此，应该有其他约束条件。从直观上说，让两个字段尽可能表示更多的原始信息，我们是不希望它们之间存在（线性）相关性的，因为相关性意味着两个字段不是完全独立，必然存在重复表示的信息。

数学上可以用两个字段的协方差表示其相关性，由于已经让每个字段均值为0，则：

$$Cov(a,b) = \frac{1}{m}\sum_{i=1}^{m}a_i b_i$$

可以看到，在字段均值为0的情况下，两个字段的协方差简洁的表示为其**内积**除以元素数m。也可以看到方差是协方差的特例，方差是协方差在两个变量相同的特例。

当协方差为0时，表示两个字段完全独立。为了让协方差为0，我们选择第二个基时只能在与第一个基正交的方向上选择。因此最终选择的两个方向一定是正交的。

至此，我们得到了降维问题的优化目标：**将一组N维向量降为K维（K大于0，小于N），其目标是选择K个单位（模为1）正交基，使得原始数据变换到这组基上后，各字段两两间协方差为0，而字段的方差则尽可能大（在正交的约束下，取最大的K个方差）。**

协方差矩阵

前面我们导出了优化目标，但是这个目标似乎不能直接作为操作指南（或者说算法），因为它只说要什么，但根本没有说怎么做。所以我们要继续在数学上研究计算方案。

我们看到，最终要达到的目的与**字段内方差**及**字段间协方差**有密切关系。因此我们希望能将两者统一表示，仔细观察发现，两者均可以表示为内积的形式，而内积又与矩阵相乘密切相关。于是我们来了灵感：

假设我们只有a和b两个字段，那么我们将它们按行组成矩阵X：

$$X = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \\ b_1 & b_2 & \cdots & b_m \end{pmatrix}$$

然后我们用X乘以X的转置，并乘上系数1/m：

$$\frac{1}{m}XX^{\mathsf{T}} = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix}$$

这个矩阵对角线上的两个元素分别是两个字段的方差，而其它元素是a和b的协方差。两者被统一到了一个矩阵的。

根据矩阵相乘的运算法则，这个结论很容易被推广到一般情况：

设我们有m个n维数据记录，将其按列排成n乘m的矩阵X，设$C = \frac{1}{m}XX^{\mathsf{T}}$，则C是一个对称矩阵，其对角线分别个各个字段的方差，而第i行j列和j行i列元素相同，表示i和j两个字段的协方差。

协方差矩阵对角化

根据上述推导，我们发现要达到优化目前，等价于将协方差矩阵对角化：**即除对角线外的其它元素化为0，并且在对角线上将元素按大小从上到下排列**，这样我们就达到了优化目的。这样说可能还不是很明晰，我们进一步看下原矩阵与基变换后矩阵协方差矩阵的关系：

设原始数据矩阵X对应的协方差矩阵为C，而P是一组基按行组成的矩阵，设Y=PX，则Y为X对P做基变换后的数据。设Y的协方差矩阵为D，我们推导一下D与C的关系：

$$\begin{aligned} D &= \frac{1}{m}YY^{\mathsf{T}} \\ &= \frac{1}{m}(PX)(PX)^{\mathsf{T}} \\ &= \frac{1}{m}PXX^{\mathsf{T}}P^{\mathsf{T}} \\ &= P(\frac{1}{m}XX^{\mathsf{T}})P^{\mathsf{T}} \\ &= PCP^{\mathsf{T}} \end{aligned}$$

我们要找的P不是别的，而是能让原始协方差矩阵对角化的P。换句话说，**优化目标变成了寻找一个矩阵P，满足PCP^{T}是一个对角矩阵，并且对角元素按从大到小依次排列，那么P的前K行就是要寻找的基，用P的前K行组成的矩阵乘以X就使得X从N维降到了K维并满足上述优化条件**。

至此，我们离“发明”PCA还有仅一步之遥！现在所有焦点都聚焦在了协方差矩阵对角化问题上.

由上文知道，协方差矩阵C是一个是对称矩阵，在线性代数上，实对称矩阵有一系列非常好的性质：

1）实对称矩阵不同特征值对应的特征向量必然正交。

2）设特征向量λ**重数**为r，则必然存在r个线性无关的特征向量对应于λ，因此可以将这r个特征向量单位正交化。

由上面两条可知，一个n行n列的实对称矩阵(协方差矩阵)一定可以找到n个单位正交特征向量，设这n个特征向量为$e_1, e_2, \cdots, e_n$，我们将其按列组成矩阵：

$$E = \begin{pmatrix} e_1 & e_2 & \cdots & e_n \end{pmatrix}$$

则对协方差矩阵C有如下结论：

$$E^{\mathsf{T}} C E = \Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

其中Λ为对角矩阵，其对角元素为各特征向量对应的特征值（可能有重复）。到这里，我们发现我们已经找到了需要的矩阵P：

$$P = E^{\mathsf{T}}$$

P是协方差矩阵的特征向量单位化后按行排列出的矩阵，其中每一行都是C的一个特征向量。如果设P按照Λ中特征值的从大到小，将特征向量从上到下排列，则用P的前K行组成的矩阵乘以原始数据矩阵X，就得到了我们需要的降维后的数据矩阵Y。

映射类降维

PCA 算法流程

总结一下PCA的算法步骤：

设有m条n维数据。

1）将原始数据按列组成n行m列矩阵X

2）将X的每一行（代表一个属性字段）进行零均值化，即减去这一行的均值

3）求出协方差矩阵$C = \frac{1}{m} X X^{\mathsf{T}}$

4）求出协方差矩阵的特征值及对应的特征向量

5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前k行组成矩阵P

6）Y=PX即为降维到k维后的数据

算法示例

这里以上文提到的

$$\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$$

为例，我们用PCA方法将这组二维数据其降到一维。

因为这个矩阵的每行已经是零均值，这里我们直接求协方差矩阵：

$$C = \frac{1}{5}\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} -1 & -2 \\ -1 & 0 \\ 0 & 0 \\ 2 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{6}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{6}{5} \end{pmatrix}$$

然后求其特征值和特征向量，求解后特征值为：

$$\lambda_1 = 2, \lambda_2 = 2/5$$

其对应的特征向量分别是：

$$c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix}, c_2 \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

其中对应的特征向量分别是一个通解，c_1和c_2可取任意实数。那么标准化后的特征向量为：

$$\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}, \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$$

因此我们的矩阵P是：

$$P = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

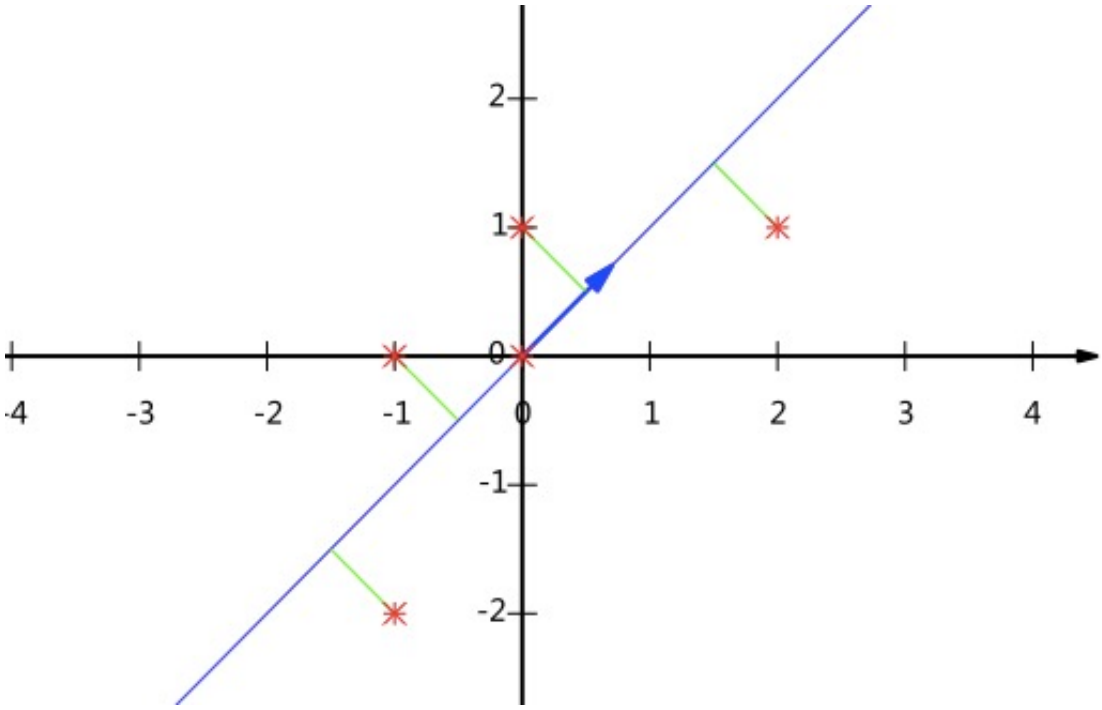
可以验证协方差矩阵C的对角化：

$$PCP^{\mathsf{T}} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 6/5 & 4/5 \\ 4/5 & 6/5 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2/5 \end{pmatrix}$$

最后我们用P的第一行乘以数据矩阵，就得到了降维后的表示：

$$Y = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -3/\sqrt{2} & -1/\sqrt{2} & 0 & 3/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$$

降维投影结果如下图：



PCA 总结

【参考】

- PCA（主成分分析）

总结

根据上面对PCA的数学原理的解释，我们可以了解到一些PCA的能力和限制。PCA本质上是将方差最大的方向作为主要特征，并且在各个正交方向上将数据“离相关”，也就是让它们在不同正交方向上没有相关性。

因此，PCA也存在一些限制，例如它可以很好的解除线性相关，但是对于高阶相关性就没有办法了，对于存在高阶相关性的数据，可以考虑Kernel PCA，通过Kernel函数将非线性相关转为线性相关。另外，PCA假设数据各主特征是分布在正交方向上，如果在非正交方向上存在几个方差较大的方向，PCA的效果就大打折扣了。

最后需要说明的是，PCA是一种无参数技术，也就是说面对同样的数据，如果不考虑清洗，谁来做结果都一样，没有主观参数的介入，所以PCA便于通用实现，但是本身无法个性化的优化。

由于 PCA 减小了特征维度，因而也有可能带来过拟合的问题。PCA 不是必须的，在机器学习中，一定谨记不要提前优化，只有当算法运行效率不尽如人意时，再考虑使用 PCA 或者其他特征降维手段来提升训练速度。

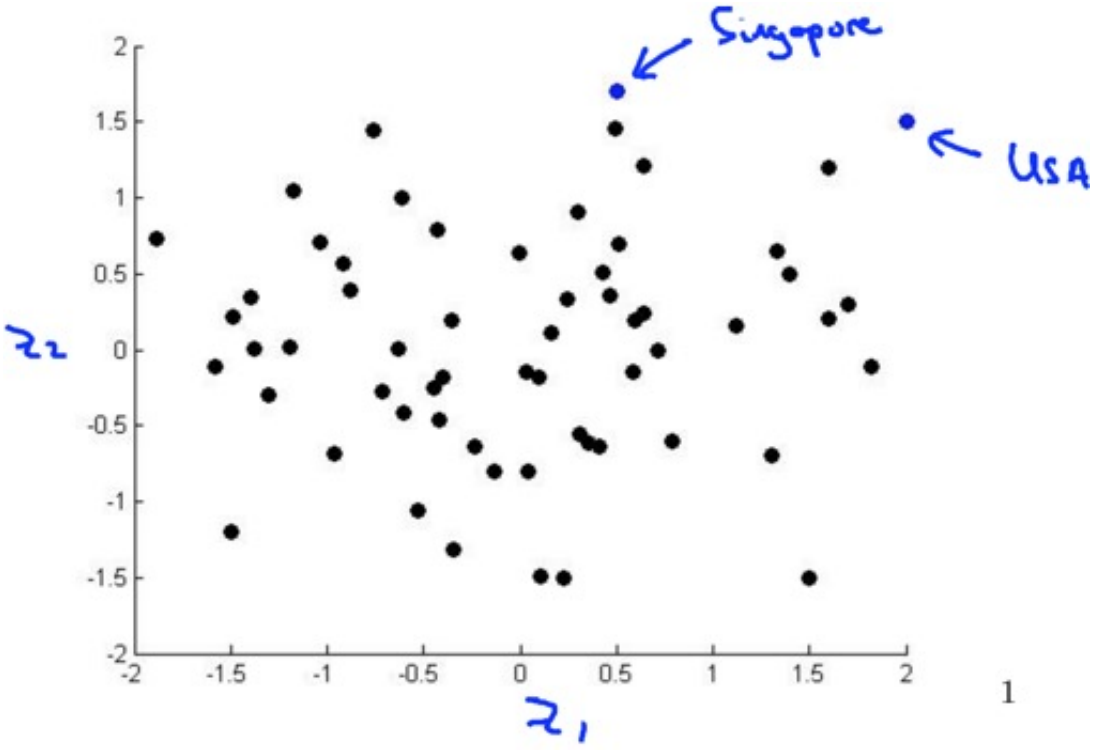
其他应用

可视化
降低特征维度不只能加速模型的训练速度，还能帮我们在低维空间分析数据，例如，一个在三维空间完成的聚类问题，我们可以通过 PCA 将特征降低到二维平面进行可视化分析。

例如下表中，我们有将近几十个特征来描述国家的经济水平，但是你仔细观察发现，我们很难直观的看出各个国家的经济差异。

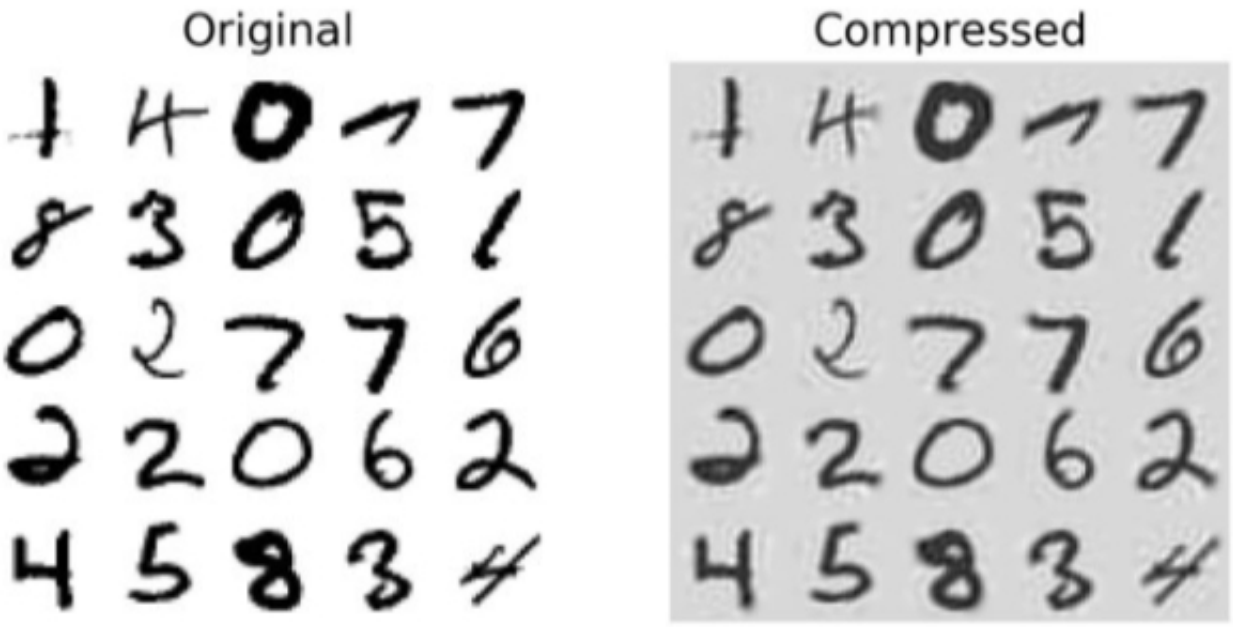
Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)
Canada	1.577	39.17	0.908	80.7	32.6	67.293
China	5.878	7.54	0.687	73	46.9	10.22
India	1.632	3.41	0.547	64.7	36.8	0.735
Russia	1.48	19.84	0.755	65.5	39.9	0.72
Singapore	0.223	56.69	0.866	80	42.5	67.1
USA	14.527	46.86	0.91	78.3	40.8	84.3

借助于 PCA，我们将特征降低到了二维，并在二维空间进行观察，很清楚的就能发现美国和新加坡具有很高的经济水平：



数据压缩

通过PCA的处理，数据集的将维度大幅度下降，为我们节省许多存储空间，同时也将加速后续处理过程。例如，如果将方差比限制为95%（通过 sklearn 的 PCA 函数实现 `pca = PCA(n_components = 0.95)`），MNIST数据集通过PCA处理后将变为154维，节省了超过80%的存储空间。如果基于压缩后的154维数据集，重构原来的784维数据，我们势必不能得到原始数据（因为损失了5%的方差），重构数据和原始数据的差异我们称之为重构误差。重构的操作可以通过 `inverse_transform()` 实现。



增量 PCA

【参考】

- Incremental Sparse-PCA Feature Extraction For Data Streams

【附加】

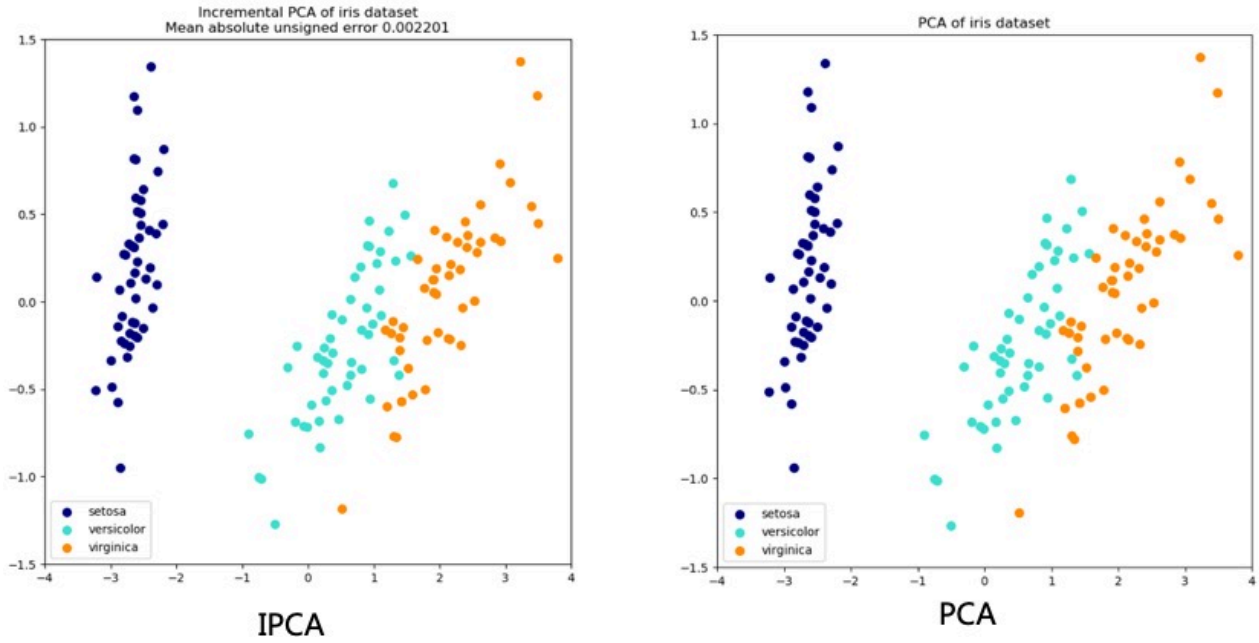
- Incremental Learning for Robust Visual Tracking

简介

虽然 PCA 可以很好地进行数据的降维，但原始的 PCA 有一个非常大的缺点，就是计算时需要用全部的数据集。在数据量不是特别大的情况下且内存足够，PCA 还可以处理，但是在数据量非常大，以至于当前内存无法容纳，就会造成内存溢出。

这时候就出现了`增量 PCA（Incremental PCA，IPCA）`，他先将训练集分成小批次，再每一批次的数据使用IPCA算法，虽然只使用了部分数据集，但精度上基本与全量 PCA 保持一致。当数据集较大，或者需要在线计算时，IPCA更有优势。

两者的对比图如下：[sklearn - Incremental PCA]



IPCA 算法流程

在 sklearn 中 IPCA 是将数据分成几个批次，批次的大小是通过`batch_size`来控制，对批次进行循环使用 PCA 算法，更新相应的参数。下面以在线为例，即每次更新参数只使用一个样本（可以与梯度下降算法、随机梯度下降和最小批次下降算法进行对比）：

假设当前有 N 个训练样本 $\boldsymbol{a}^{(i)}$ 来初始化 PCA，其中$\boldsymbol{a}^{(i)} \in \mathbb{R}^n (i = 1, \ldots, N)$，对这些训练样本应用 PCA 算法那，产生如下的特征空间模型：

$$\Omega = (\bar{\boldsymbol{a}}, \boldsymbol{U}_k, \boldsymbol{\Lambda}_k, \boldsymbol{N})$$

其中：

- $\bar{\boldsymbol{a}}$ 是$\boldsymbol{a}^{(i)} (i = 1, \ldots, N)$的均值向量
- $\boldsymbol{U}_k$ 是列向量与特征向量相关的 n×k 的矩阵
- $\boldsymbol{\Lambda}_k = diag\{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ 是一个非零特征值作为对角元素的 k×k 的矩阵

ICPA 现在假设又提拱了一个样本，即第 $N+1$ 个训练样本，即$(\boldsymbol{N}+\boldsymbol{1})^{th}$，如下：

$$\boldsymbol{a}^{N+1} = \boldsymbol{y} \in R^n$$

新增的样本产生了在`均值向量(mean vector)`、`协方差矩阵(convariance matrix)`的变化，需要更新特征空间模型Ω，新的特征空间模型Ω'定义如下：

$$\Omega' = (\bar{\boldsymbol{a}}', \boldsymbol{U}'_{k'}, \boldsymbol{\Lambda}'_{k'}, \boldsymbol{N}+\boldsymbol{1})$$

特征共空间维度 k' 是 k 还是 k+1 依赖于 $\boldsymbol{y}$是否包含足够的能量在互补特征空间。特征轴被旋转以适应数据分布的变化通过三步：均值向量更新、特征空间增强、特征轴的旋转。

1.均值向量更新（mean vector update）
可以通过如下公式：

$$\bar{\boldsymbol{a}}' = \frac{1}{N+1}(\boldsymbol{N}\bar{\boldsymbol{a}} + \boldsymbol{y}) \in R^n$$

2.特征空间增强

有两个标准可以帮助决定是否维度增强有必要。

其中一个是基于残差向量的范数，定义如下：

$$\boldsymbol{h} = (\boldsymbol{y} - \bar{\boldsymbol{a}}) - \boldsymbol{U}_k^T \boldsymbol{g}$$

其中 $\boldsymbol{g} = U_k^T(y - \bar{a})$

另一个是基于`累积率（accumulation ratio）`，定义如下：

$$A(\boldsymbol{U}_k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j} = \frac{(N+1)\sum_{i=1}^{k} \lambda_i + \|\boldsymbol{U}_k^T(\boldsymbol{y} - \bar{a})\|\|\boldsymbol{U}_k^T(\boldsymbol{y} - \bar{a})\|}{(N+1)\sum_{j=1}^{k} \lambda_j + \|(\boldsymbol{y} - \bar{a})\|\|(\boldsymbol{y} - \bar{a})\|}$$

其中，λ_i 是第 i 个最大的特征值，n 表示输入空间的维度，k 表示当前特征空间的维度。

在特征空间增强中分别表示为如下形式：

- 残差向量范数

$$\hat{\boldsymbol{h}} = \begin{cases} \frac{h}{\|h\|}, & if\ \|h\| > \eta \\ 0, & otherwise \end{cases}$$

- 累积率

$$\hat{\boldsymbol{h}} = \begin{cases} \frac{h}{\|h\|}, & if\ A(\boldsymbol{U}_k) < \theta \\ 0, & otherwise \end{cases}$$

3.特征空间旋转

如果上面的残差向量范数或者累积率被被满足，那么特征空间的维度就从 k 增长到 $k+1$。一个新的特征轴 $\hat{\boldsymbol{h}}$ 被添加到特征向量矩阵 $\boldsymbol{U}_k$ 中。如果上面的条件没有被满足，那么维度保持不变。特征轴需要进行旋转以适应新的数据分布。如果旋转通过旋转矩阵 R 给出，那么特征空间的更新通过下面的方式给出：

- 如果新的特征轴不加入

$$\boldsymbol{U}'_{k+1} = [\boldsymbol{U}'_k, \hat{\boldsymbol{h}}]\boldsymbol{R}$$

- 其他

$$\boldsymbol{U}'_k = \boldsymbol{U}_k\boldsymbol{R}$$

算法的实现可以参考 sklearn.decomposition.IncrementalPCA 用法，源码为 incremental_pca.py

随机 PCA

【附加】

- A randomized algorithm for principal component analysis

在增量 PCA 我们讲到，IPCA 是为了解决数据量过大时内存放不下的问题，而随机 PCA（Randomized PCA，RPCA）要解决的则是当数据的维度过大时，进行奇异值分解较难，时间复杂度较高的问题。

之所以叫随机 PCA 是因为使用的 SVD 与标准的 PCA 不同，随机 PCA 使用的是 `Randomized SVD` 进行奇异值分解，因此该 PCA 被叫做随机 PCA。要了解随机 PCA，首先就要弄清楚什么是 `Randomized SVD`。

标准 SVD

【参考】

- facebook - Fast Randomized SVD

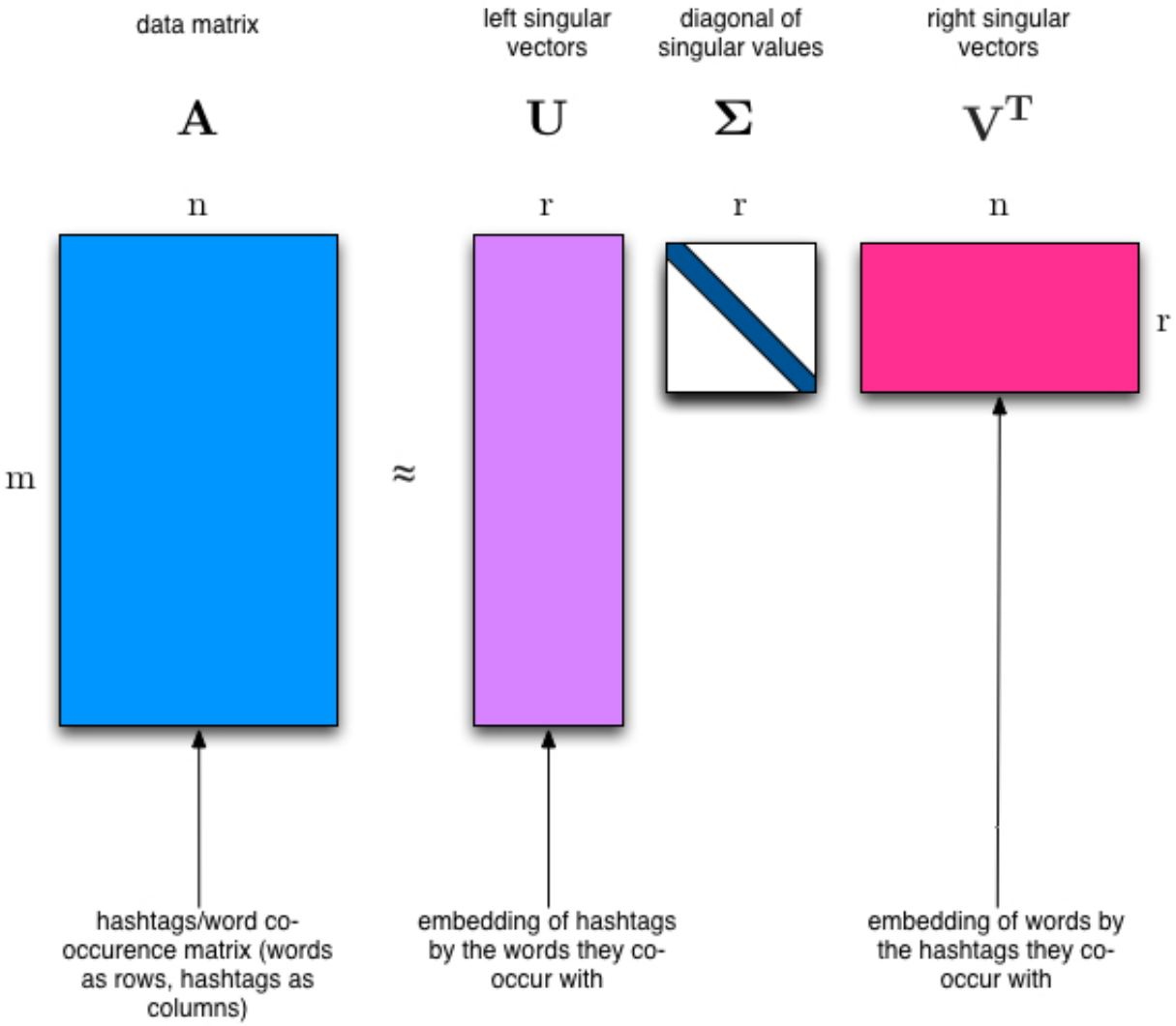
SVD（Singular Value Decomposition） 在机器学习、统计学、信号处理等领域都有应用。通常 SVD 是将一个 $m \times n$ 的实矩阵 A 分解为 $A = U\Sigma V^T$ 的形式，其中：

- U 是一个 $m \times m$ 的左奇异向量的正交矩阵，
- Σ 是一个 $m \times n$ 的奇异值对角矩阵，
- V^T 是一个 $n \times n$ 的右奇异向量的正交矩阵。

SVD 提供了对于 A 的低秩近似计算的方法，找到一个矩阵 A_r 他的秩 $r < m, n$，最终让 $\|A_r - A\|$ 最小化。通过下面的流程可以得到 A 的近似：

- 降序排列 Σ 中非零的值
- 取出 Σ 中前 r 个元素（记做Σ_r，是一个 $r \times r$ 的对角矩阵），U 的相关列（U_r 是一个 $m \times r$ 的正交矩阵）和 V 的相关列（V_r 是一个 $n \times r$ 的正交矩阵）
- 得到 $A_r = U_r \Sigma_r V_r^T$

可视化如下：



虽然 SVD 可以对矩阵进行低秩近似，但她有一个很大的缺点即使在大尺度问题上，时间消耗也是特别的大。因此我们需要转向随机化方法，他们提供显著的加速在经典的方法之上。

Randomized SVD

【参考】

- tencent - Randomized SVD 算法介绍与实现
- sklearn - PCA using randomized SVD

Facebook 这片文章主要关注的是由Nathan Halko, Per-Gunnar Martinsson and Joel A. Tropp于2009年发表的《 Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions》论文中提到的随机化矩阵近似方法。

该方法主要有两大步骤，首先使用随机化技术计算 A 范围的近似值，即我们知道一个 Q 他有 r 个正交列，且满足 $A \approx QQ^TA$，假设我们找到了这样的低维近似基 Q，我们就可以按照下列方式计算 A 的 SVD：

- 构建 $B = Q^TA$，现在 B 相对来说较小（因为 Q 只有很小的列数量），这样我们就可以按照标准方法计算 B 的 SVD，因此有 $B = S\Sigma V^T$，其中 S, V 是正交的，Σ 是对角矩阵。
- 之后，因为 $A \approx QQ^TA = Q(S\Sigma V^T)$，可以看到 $U = QS$，这样我们就得到了 A 的低秩近似 $A \approx U\Sigma V^T$

上述过程的关键在于在 $A \approx QQ^TA$ 中 Q 的求解，在上面的论文中给出了具体的方法，即采用随机采样的方式构建矩阵 Q，这里的假设 A 是一个 $m \times n$的矩阵：

- 通过设置需要获取的奇异值个数 k 以及过采样参数 p，投建一个由 $k+p$ 个 n 为随机列向量组成的矩阵 Ω，要求 $(k+p) \leq min\{m,n\}$，每一个列向量的值均采样自标准正态分布，因此这些采样的列向量线性独立
- 进行矩阵乘积运算$Y = A\Omega$，由于向量集合 Y 也是线性独立的，因此，Y 形成了矩阵 A 的列向量空间
- 通过求取向量集合 Y 的正交基，从而得到 A 的近似基 Q

通过上面的随机采样之后得到的 A 的近似基 Q，由$B = S\Sigma V^T$可以知道，B 是一个 $(k+p) \times n$的矩阵，相比初始的矩阵 $A(m \times n)$，B 的行数少了很多，更易于进行 SVD 分解。

从上面的过程也可以看到，随机 SVD 比标准的 SVD 多了一个随机构建向量的过程，其他都相同。

在 sklearn 中可以通过将 `PCA` 类中的 `svd_solver='randomized'` 来启用随机化 PCA。

Randomized SVD改进

当矩阵的奇异值衰减较快的时候，上面的表现非常好。然而，当我们遇到较大的矩阵，或者矩阵的奇异值下降缓慢时，上面的结果往往不准确。这主要是由于较小的奇异值对应的奇异值向量干扰了计算的结果。为此，Randomized SVD算法提出了改进算法，主要是通过矩阵的多次(the power of matrix)乘积运算来减小这些奇异值向量相对于主要奇异值向量的权重。

另外，为了避免上述的Power迭代过程中数值较小的奇异值所携带的信息在计算过程中丢失，上面的论文中进一步提供了改进策略，即每次进行 A 或者 A^T 的乘积运算时，都对采样矩阵的列向量进行正交化

核化 PCA

【参考】

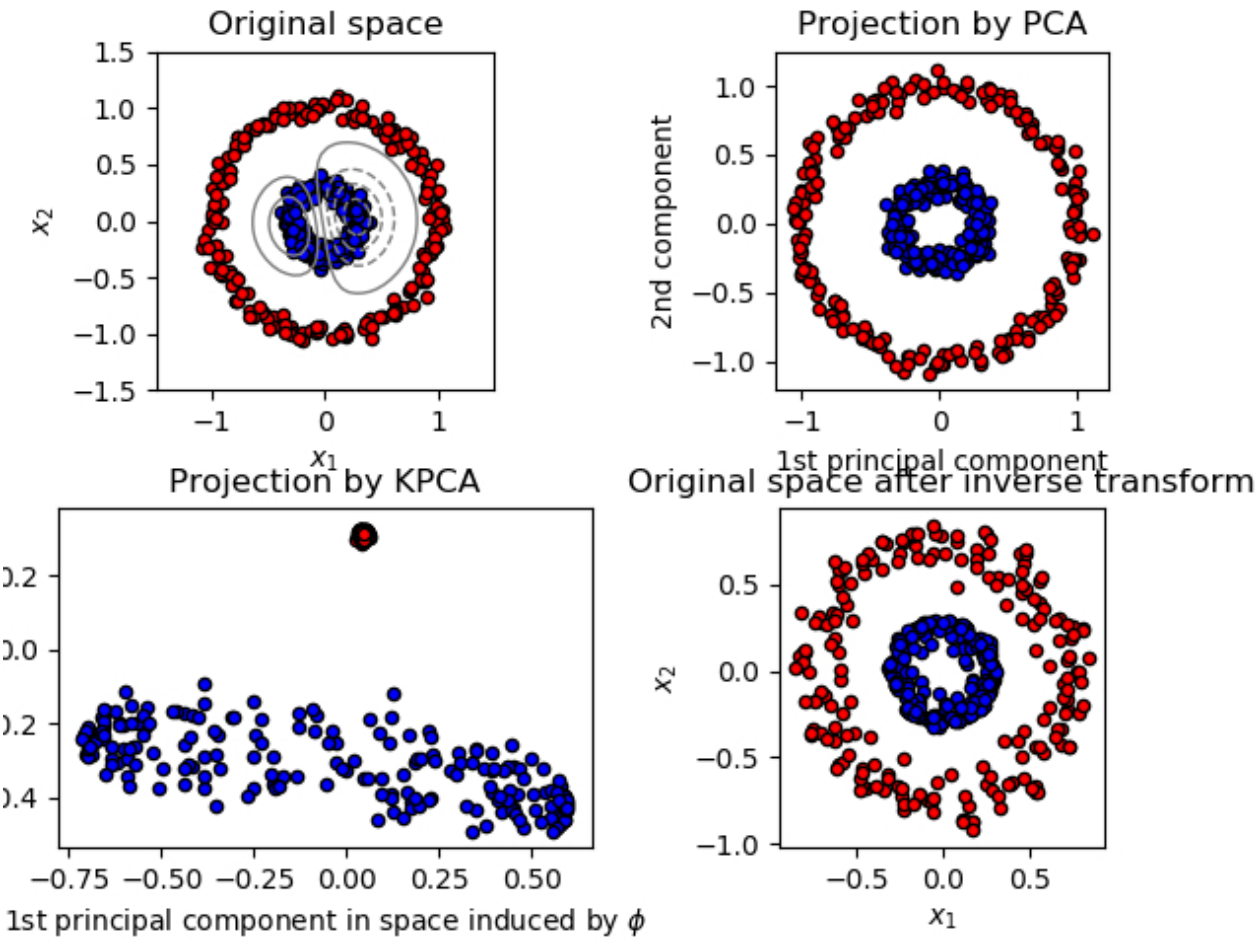
- Dimensionality reduction. PCA. Kernel PCA.
- 知乎 - Kernel PCA 原理和演示
- github - Sklearn 与 TensorFlow 机器学习实用指南

【附加】

- LECTURE :KERNEL PCA
- Kernel PCA vs PCA vs ICA in Tensorflow/sklearn

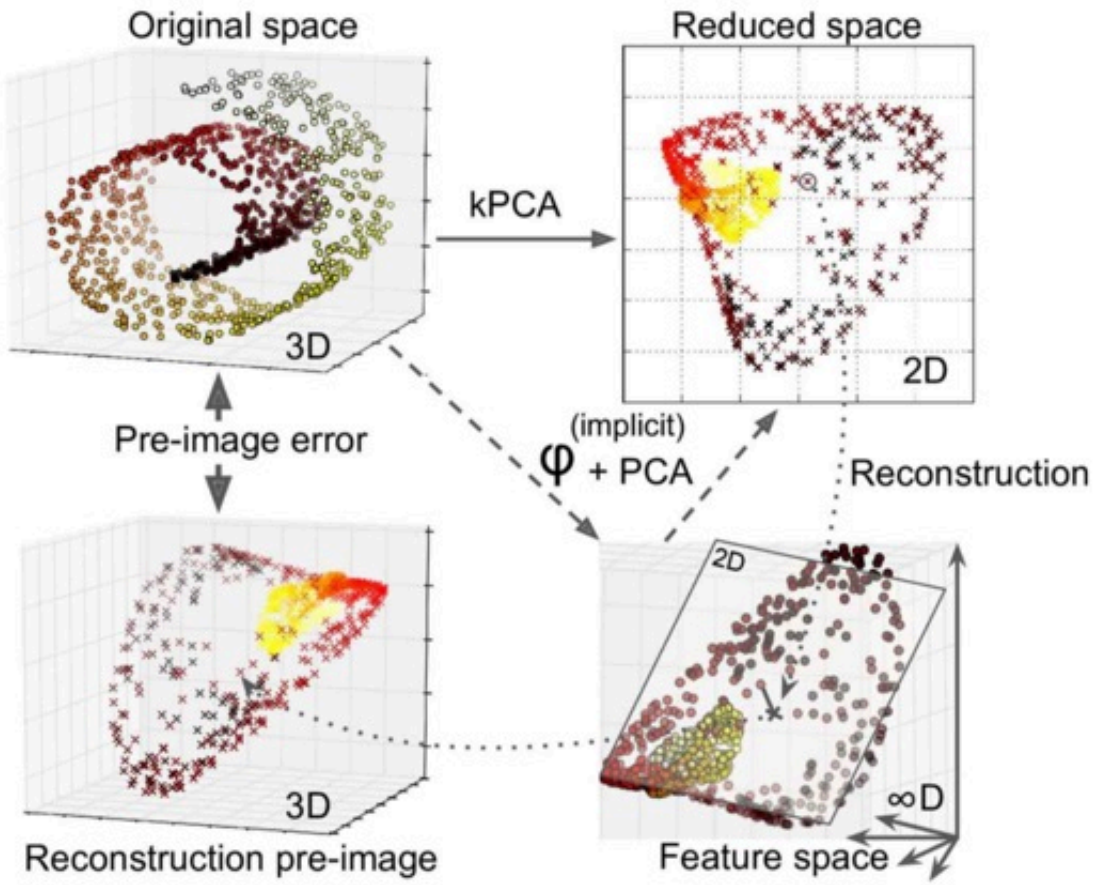
可以看到上面介绍的 PCA 方法都是线性降维的方法。在 SVM 章节我们介绍了`核函数`，他可以将低维样本空间不可分的点，映射到高维空间中进而变得线性可分，且映射不会增加计算的复杂度。同样我们可以将`核函数`应用于PCA，就是所谓的`核化 PCA(kernel pca,KPCA)`，这样 PCA 就可以执行复杂的非线性投影以降低维度。他通常可以在映射后很好的保留样本点之间的聚类关系。

核化 PCA 可以参考 sklearn 的`kernelPCA`类，通过设置 `kernel` 选择不同的核函数。如下图使用高斯核：



由于 KPCA 是无监督学习算法，因此没有明显的性能指标可以帮助您选择最佳的核方法和超参数值。但是，降维通常是监督学习任务（例如分类）的准备步骤，因此您可以简单地使用网格搜索来选择可以让该任务达到最佳表现的核方法和超参数。例如，下面的代码创建了一个两步的流水线，首先使用 kPCA 将维度降至两维，然后应用 Logistic 回归进行分类。然后它使用 `GridSearchCV()` 为 KPCA 找到最佳的核和`gamma`值，以便在最后获得最佳的分类准确性.

另一种完全为非监督的方法，是选择产生最低重建误差的核和超参数。但是，重建并不像线性 PCA 那样容易。这里是原因：下图显示了原始瑞士卷 3D 数据集（左上角），并且使用 RBF 核应用 KPCA 后生成的二维数据集（右上角）。由于核技巧，这在数学上等同于使用特征映射`φ`将训练集映射到无限维特征空间（右下），然后使用线性 PCA 将变换的训练集投影到 2D。请注意，如果我们可以在缩减空间中对给定实例实现反向线性 PCA 步骤，则重构点将位于特征空间中，而不是位于原始空间中（例如，如图中由x表示的那样）。由于特征空间是无限维的，我们不能找出重建点，因此我们无法计算真实的重建误差。幸运的是，可以在原始空间中找到一个贴近重建点的点。这被称为重建前图像（reconstruction pre-image）。一旦你有这个前图像，你就可以测量其与原始实例的平方距离。然后，您可以选择最小化重建前图像错误的核和超参数。



您可能想知道如何进行这种重建。一种解决方案是训练一个监督回归模型，将预计实例作为训练集，并将原始实例作为训练目标。如果您设置了`fit_inverse_transform = True`，sklearn 将自动执行此操作，代码如下所示：

```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf",
gamma=0.0433,fit_inverse_transform=True)
X_reduced = rbf_pca.fit_transform(X)
X_preimage = rbf_pca.inverse_transform(X_reduced)
```

这样就可以计算重构误差：

```
from sklearn.metrics import mean_squared_error
mean_squared_error(X, X_preimage) #32.786308795766132
```

这样就可以使用叉验证的方格搜索来寻找可以最小化重建前图像误差的核方法和超参数。

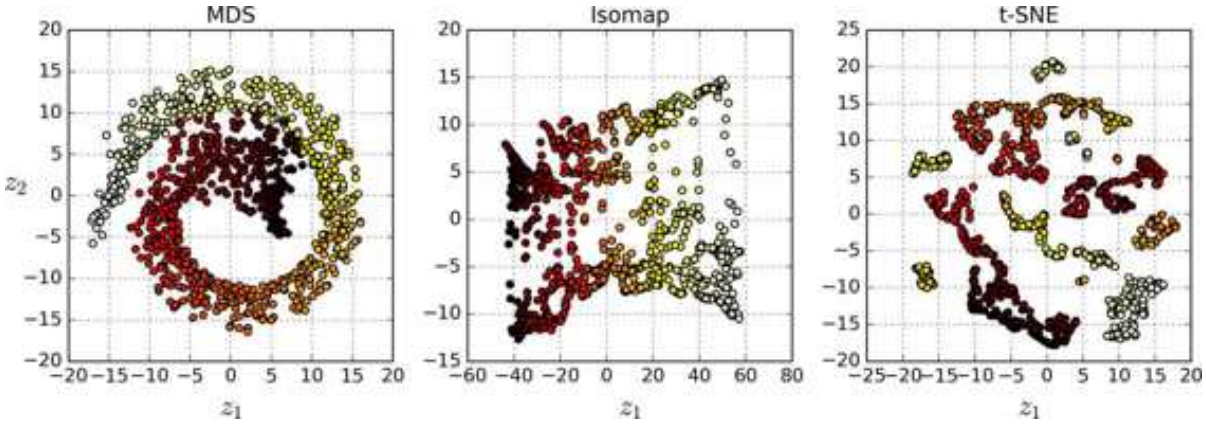
其他

其他几种降维方法：

- 多维缩放（Multidimensional Scaling,MDS）在尝试保持实例之间距离的同时降低了维度
- 等度量映射（Isometric Mapping,Isomap）:等度量映射测算的距离是测地线距离，而不 是传统的欧式距离。其基本出发是人为低维流形嵌入高维空间后，直接在高维空间中计算具有误导性，因为在高维空间中的支线距离在低维空间中是不可到达的
- t分布领域嵌入方法（t-Distributed Stochastic Neighbor Embedding,t-SNE）:该算法的理念是在

降低维度的同时，试图聚合相似的实例以及区分不相似的实例。如下图，处理后的数据按类进行了聚合，有较好的可视性。

- 线性判别分析（Linear Discriminant Analysis，LDA）实际上是一种分类算法，但在训练过程中，它会学习类之间最有区别的轴，然后使用这些轴来定义用于投影数据的超平面。LDA 的好处是投影会尽可能地保持各个类之间距离，所以在运行另一种分类算法（如 SVM 分类器）之前，LDA 是很好的降维技术。



流形学习

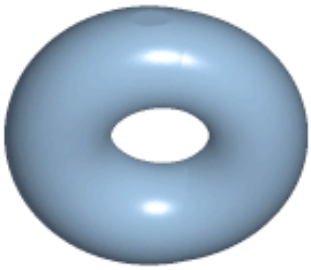
【参考】

- 个站 - 浅谈流形学习
- 流形学习 (Manifold Learning)
- 知乎 - 求简要介绍一下流形学习的基本思想
- academia - 流形学习

简介

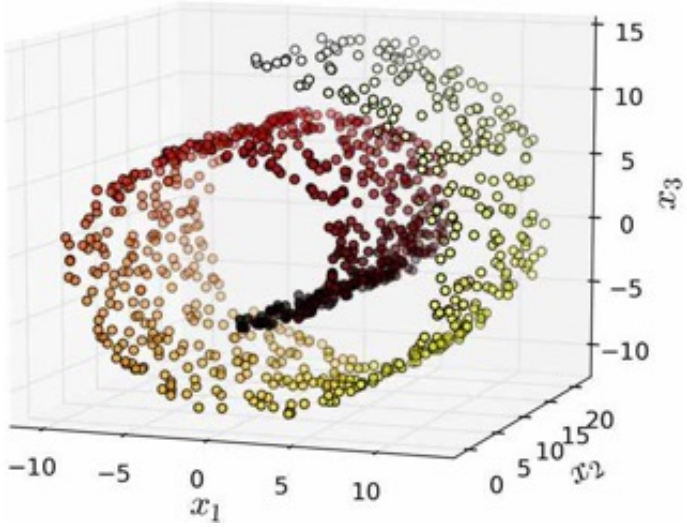
流行学习（manifold learning）是借鉴了拓扑流行概念的降维方法，'流形'是在局部与欧式空间`同胚`的空间, 换言之, 他在局部空间具有欧式空间的性质, 能用欧式距离来进行距离计算。流形并不是一个“形状”，而是一个“空间”。

对于同胚, 如果拓扑空间是一个几何物体，同胚就是把物体连续延展和弯曲，使其成为一个新的物体。因此，正方形和圆是同胚的，但球面和环面就不是。用一幅图形象的理解同胚，例如下图所示的咖啡杯和甜甜圈:



这给降维带来了很大的启发: 若低维流嵌入到高维空间中, 则数据样本在高维空间的分布虽然看上去非常复杂, 但在局部上仍然具有欧式空间的性质, 因此, 可以容易地在局部建立降维映射关系, 然后再设法将局部映射关系推广到全局. 当维数被降到二维或者三维时, 能对数据进行可视化展示, 因此流行学习也可以用于可视化.

这么说有点抽象, 以著名的瑞士卷数据集为例:



瑞士卷一个是二维流形的例子。简而言之，二维流形是一种二维形状，它可以在更高维空间中弯曲或扭曲。更一般地，一个`d`维流形是类似于`d`维超平面的`n`维空间（其中`d < n`）的一部分。在我们瑞士卷这个例子中，`d = 2`，`n = 3`：它有些像 2D 平面，但是它实际上是在第三维中卷曲。

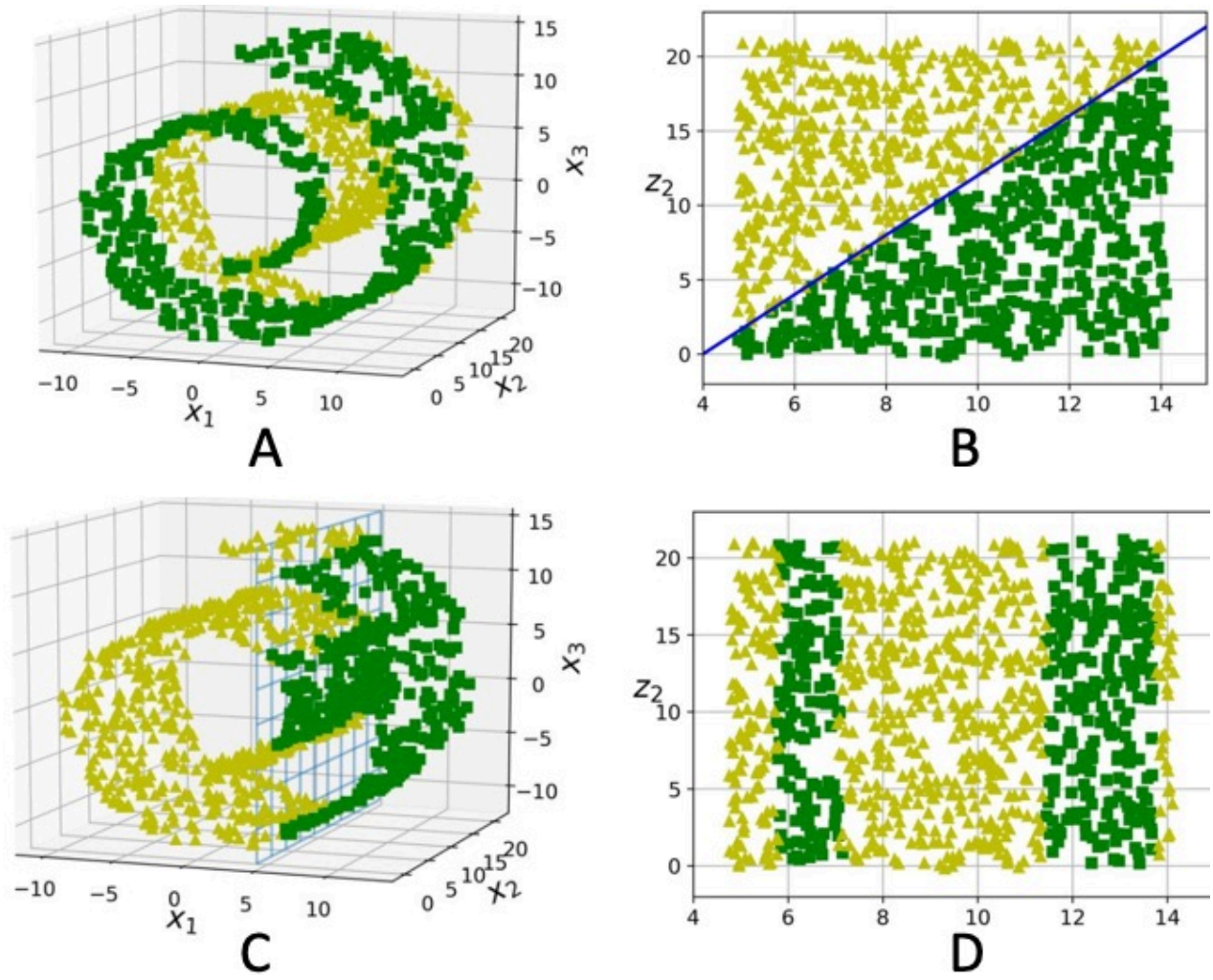
流行学习依赖于`流形猜想（manifold assumption）`，也被称为`流形假设（manifold hypothesis）`，它认为大多数现实世界的高维数据集大都靠近一个更低维的流形。这种假设经常在实践中被证实。

以 MNIST 数据集为例: 所有手写数字图像都有一些相似之处。它们由连线组成，边界是白色的，大多是在图片中中间的，等等。如果你随机生成图像，只有一小部分看起来像手写数字。换句话说，如果您尝试创建数字图像，那么您的自由度远低于您生成任何随便一个图像时的自由度。这些约束往往会将数据集压缩到较低维流形中。

流形假设通常包含着另一个隐含的假设：你现在的手上的工作（例如分类或回归）如果在流形的较低维空间中表示，那么它们会变得更简单。例如，下图的第一行中，瑞士卷被分为两类：在三维空间中（A），分类边界会相当复杂，但在二维展开的流形空间中（B），分类边界是一条简单的直线。

但是，这个假设并不总是成立。例如，下图的最下面一行，决策边界位于$x_1 = 5$（C）。这个决策边界在原始三维空间（一个垂直平面）看起来非常简单，但在展开的流形中却变得更复杂了（四个独立线段的集合）（D）。

简而言之，**如果在训练模型之前降低训练集的维数，那训练速度肯定会加快，但并不总是会得出更好的训练效果；这一切都取决于数据集。**



流形学习可以看做是对想 PCA 这样的线性框架泛化到非线性数据结构. 虽然有监督变体的存在, 典型的流形学习问题是非监督的，她从数据自身学习数据的高维结构，不适用预定义的类别。

目前，把流形引入到机器学习领域来主要有两种用途：

- 一是将原来在欧氏空间中适用的算法加以改造，使得它工作在流形上，直接或间接地对流形的结构和性质加以利用；
- 二是直接分析流形的结构，并试图将其映射到一个欧氏空间中，再在得到的结果上运用以前适用于欧氏空间的算法来进行学习

`Isomap` 就是这样的一个例子，因为它实际上是通过“改造一种原本适用于欧氏空间的算法”，达到了“将流形映射到一个欧氏空间”的目的。 `Isomap` 所改造的这个方法叫做 `MDS`，他是一种降维方法，它的目的就是使得降维之后的点两两之间的距离尽量不变（也就是和在原是空间中对应的两个点之间的距离要差不多）。只是 `MDS` 是针对欧氏空间设计的，对于距离的计算也是使用欧氏距离来完成的。如果数据分布在一个流形上的话，欧氏距离就不适用了。

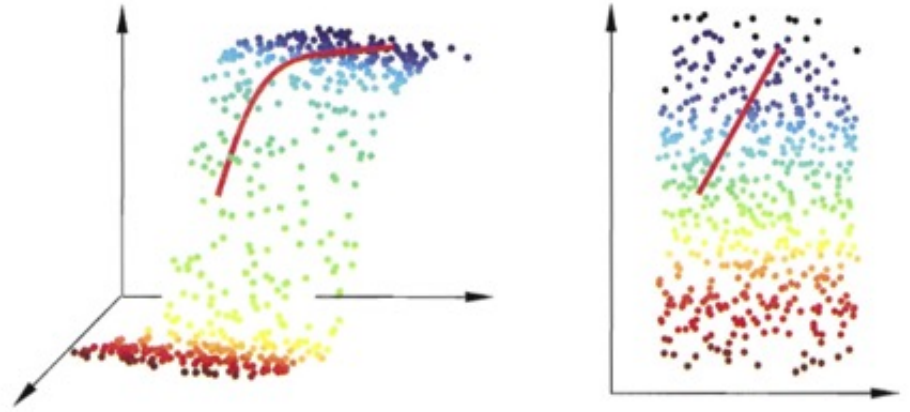
常见的流形学习主要有以下几种：多维缩放（Multidimensional Scaling, MDS）、等度量映射（Isometric Mapping, Isomap）、t分布领域嵌入方法（t-Distributed Stochastic Neighbor Embedding, t-SNE）、局部线性嵌入 (Locally Linear Embedding, LLE)等。

t-SNE原理 可以参考《t-SNE完整笔记》

MDS

简介

在很多时候，人们观测到或者搜集到的数据样本虽然是高维的，但与学习任务密切相关的也许仅是某个低维分布，即在高维空间中的一个`低维嵌入(embedding)`。如下图，原始的高维空间样本点，在这个低维嵌入子空间中更容易进行学习：



(a) 三维空间中观察到的样本点　　(b) 二维空间中的曲面

若要求**原始空间中样本之间的距离在低维空间中得以保持**，如上图所示，即得到"多维缩放"（Multidimensional Scaling, MDS）算法。

根据距离的度量方式不同可以将其分为度量型 (metric) MDS 和 非度量型 (non-metric) MDS。度量型 MDS 通过计算不同样本之间距离的度量值进行降维，而非度量型则仅考虑距离的排序信息。下面介绍的是度量型的 MDS。

推导

假定 m 个样本在原始空间中的距离矩阵为 $\boldsymbol{D} \in \mathbb{R}^{m \times m}$，其第 i 行 j 列的元素 $dist_{ij}$ 为样本 $\boldsymbol{x}_i$ 到 $\boldsymbol{x}_j$ 的距离。我们的目标是获得样本在 d' 维空间的表示 $\boldsymbol{Z} \in \mathbb{R}^{d' \times m}, d' \leq d$，且任意两个样本在 d' 维空间的距离等于原始空间中的距离，即$\|z_i - z_j\| = dist_{ij}$.

令 $\boldsymbol{B} = \boldsymbol{Z}^T \boldsymbol{Z} \in \mathbb{R}^{m \times m}$，其中 $\boldsymbol{B}$ 为降维后样本的内积矩阵，$b_{ij} = z_i^T z_j$，有：

$$\begin{aligned} dist_{ij}^2 &= \|z_i\|^2 + \|z_j\|^2 - 2z_i^T z_j \\ &= b_{ii} + b_{jj} - 2b_{ij} \end{aligned} \tag{D1}$$

为了方便讨论，令降维后的样本 $\boldsymbol{Z}$ 被中心化，即 $\sum_{i=1}^{m} z_i = 0$。显然，矩阵 $\boldsymbol{B}$ 的行与列之和均为零，即 $\sum_{i=1}^{m} b_{ij} = 0, \sum_{j=1}^{m} b_{ij} = 0$，有：

$$\begin{aligned} \sum_{i=1}^{m} dist_{ij}^2 &= tr(\boldsymbol{B}) + mb_{jj} \\ \sum_{j=1}^{m} dist_{ij}^2 &= tr(\boldsymbol{B}) + mb_{ii} \\ \sum_{i=1}^{m}\sum_{j=1}^{m} dist_{ij}^2 &= 2m\, tr(\boldsymbol{B}) \end{aligned} \tag{D2}$$

其中 tr(·) 表示矩阵的迹（trace），$tr(\boldsymbol{B}) = \sum_{i=1}^{m} \|z_i\|^2$，令：

$$
\begin{aligned}
dist_{i\cdot}^2 &= \frac{1}{m}\sum_{j=1}^{m} dist_{ij}^2 \\
dist_{j\cdot}^2 &= \frac{1}{m}\sum_{i=1}^{m} dist_{ij}^2 \\
dist_{\cdot\cdot}^2 &= \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} dist_{ij}^2
\end{aligned}
\tag{D3}
$$

由式（D1）~（D3）可以得到：

$$
b_{ij} = -\frac{1}{2}(dist_{ij}^2 - dist_{i\cdot}^2 - dist_{j\cdot}^2 + dist_{\cdot\cdot}^2) \tag{D4}
$$

由此，即可通过降维前后保持不变的距离矩阵 $\boldsymbol{D}$ 求取内积矩阵 $\boldsymbol{B}$。

对矩阵 $\boldsymbol{B}$ 做特征分解，$\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$，其中 $\boldsymbol{\Lambda} = diag(\lambda_1, \lambda_2, \ldots, \lambda_d)$ 为特征值组成的对角矩阵，且$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$，$\boldsymbol{V}$ 为特征向量矩阵。假定其中有 d^* 个非零特征值，他们构成对角矩阵 $\boldsymbol{\Lambda}_* = diag(\lambda_1, \lambda_2, \ldots, \lambda_{d^*})$，令$\boldsymbol{V}_*$表示相应的特征向量矩阵，则 $\boldsymbol{Z}$ 可以表示为：

$$
\boldsymbol{Z} = \sqrt{\boldsymbol{\Lambda}_*}\boldsymbol{V}_*^T \in \mathbb{R}^{d^* \times m} \tag{D5}
$$

在现实应用中为了有效降维，往往仅需要降维后的距离与原始空间的距离尽可能的接近，而不必严格的相等。此时可以取 $d' \ll d$ 个最大值特征构成对角矩阵 $\tilde{\boldsymbol{\Lambda}} = diag(\lambda_1, \lambda_2, \ldots, \lambda_{d'})$，令 $\tilde{\boldsymbol{V}}$表示相应的特征向量矩阵，则$\boldsymbol{Z}$ 可以表示为：

$$
\boldsymbol{Z} = \sqrt{\tilde{\boldsymbol{\Lambda}}}\tilde{\boldsymbol{V}}^T \in \mathbb{R}^{d' \times m} \tag{D6}
$$

算法流程

输入： 距离矩阵 $\boldsymbol{D} \in \mathrm{m} \times \mathrm{m}$，其元素 $dist_{ij}$ 为样本 $\boldsymbol{x}_i$ 到 $\boldsymbol{x}_j$ 的距离
低维空间维数 d'

过程：
根据式（D3）计算 $dist_{i\cdot}^2, dist_{j\cdot}^2, dist_{\cdot\cdot}^2$
根据式（D4）计算矩阵 $\boldsymbol{B}$　对矩阵 $\boldsymbol{B}$ 进行特征分解
取 $\tilde{\boldsymbol{\Lambda}}$ 为 d' 个最大特征值所构成的对角矩阵，$\tilde{\boldsymbol{V}}$为相应的特征向量矩阵

输出： 矩阵 $\sqrt{\tilde{\boldsymbol{\Lambda}}_*}\tilde{\boldsymbol{V}}_*^T \in \mathbb{R}^{m \times d'}$，每一行是一个样本的低维坐标

相应的实现可以参考 sklearn Multi-dimensional Scaling (MDS)

一般来说，与获得低维子空间，最简单的是对原始高维空间进行线性变换。给定 d 维空间中的样本 $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m) \in \mathbb{R}^{d \times m}$，变换之后得到 $d' \leq d$ 维度的空间样本：

$$
\boldsymbol{Z} = \boldsymbol{W}^T\boldsymbol{X}
$$

其中 $\boldsymbol{W} \in \mathbb{R}^{d \times d'}$ 是变换矩阵， $\boldsymbol{Z} \in \mathrm{d}' \times \mathrm{m}$ 是样本在新空间的表达。这个过程可以参考 PCA。

对降维效果的评估，通常是比较降维前后学习器的性能，若性能有所提高，则认为降维起到了作用。若将维数将至二维或三维，则可通过可视化技术直观的判读降维的效果。

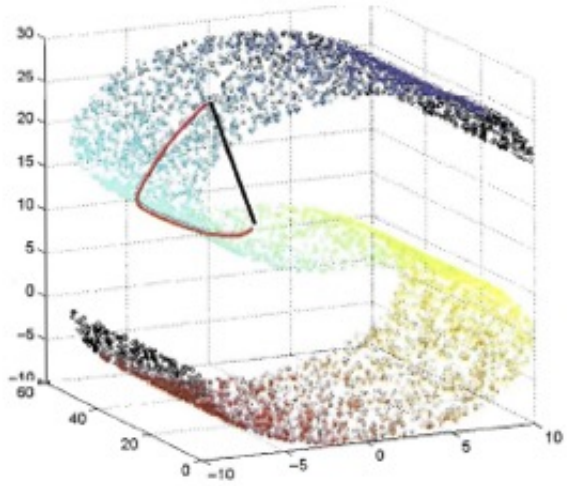
Isomap

【参考】

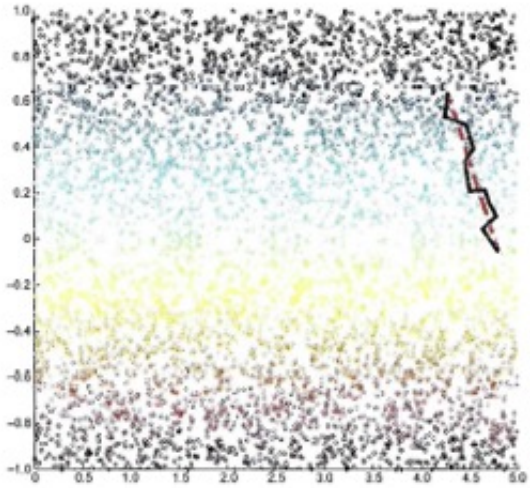
- A Global Geometric Framework for Nonlinear Dimensionality Reduction
- Isomap Isometric feature mapping
- sklearn - isomap

简介

等度量映射(Isometric Mapping, Isomap) 的基本出发点，是认为低维流形嵌入到高维之后，直接在高维空间计算直线距离具有误导性，因为高维空间中的直线距离在低维嵌入流形上是不可达的。



(a)测地距离与高维直线距离



(b)测地距离与近邻距离

如上图（a）所示，低维嵌入流形上两点间的距离是`测地线（geodesic）距离`：如果香想象一只虫子从一点爬到另一点，如果他不脱离曲面行走，那么（a）中的红色曲线是最短的路径，即 S 曲面上的`测地线`，**测地线距离是两点之间的`本真距离`**。显然，直接在高维空间计算直线的距离是不恰当的。

低维嵌入流行上的测地线距离（红色）虽然不能用公安为空间的直线距离计算，但能用近邻距离来近似

Isomap 可以看做是对 MDS 或者核化 PCA 的扩展。

测地线距离的计算

那么如何计算测地线距离呢？这时候我们可以利用流形在局部上与欧式空间同胚的这个性质，对每个点基于欧式距离找出其近邻点，然后就能建立一个`近邻连接图`，图中近邻点之间存在连接，而非近邻带你之间不存在连接，于是，计算两点之间的测地线距离的问题，就转变成为计算近邻连接图上两点之间的最短路径的问题。如上图(b) 可以看出基于近邻距离逼近能获得低维流行上测地线距离很好的近似。

在近邻连接图上计算两点之间的最短距离，可以采用著名的`Dijkstra 算法`或`Floyd 算法`，在得到任意两点之间的距离之后，就可以通过 MDS 算法来获取样本点在低维空间的坐标。

对近邻图的构建通常有两种办法：

- 一种是指定近邻点的个数，例如欧式距离最近 k 个点位紧邻点，这样得到的紧邻图称为`k 近邻图`；
- 另一种是指定距离阈值 ϵ ，距离小于 ϵ 的点被认为是近邻点，这样得到的近邻图称为`ϵ 近邻图`

两种方式均有不足，例如若近邻范围指定的较大，则距离很远的点可能被认为近邻，这样就出现了`短路`问题；近邻范围指定较小，则图中有些区域可能与其他区域不存在连接，这样就出现了`断路`问题。短路和断路都会给后续的最短路径计算造成误导。

算法流程

输入： $D = \{x_1, x_2, \ldots, x_m\}$
近邻参数 k
低维空间维数d'

过程:
for i=1,2,...,m do
确定x_i 的 k 近邻
x_i 与 k 近邻点之间的距离设置为欧式距离，与其他点之间的距离设置为无限大；
end for
调用最短路径算法计算任意两样本点之间的距离$dist(x_i, x_j)$
将$dist(x_i, x_j)$ 作为 MDS 算法的输入
return MDS 算法那的输出

输出： 样本即 D 在低维空间的投影 $Z = \{z_1, z_2, \ldots, z_m\}$

此处可以参考 sklearn 的 Isomap 实现。

新样本映射问题

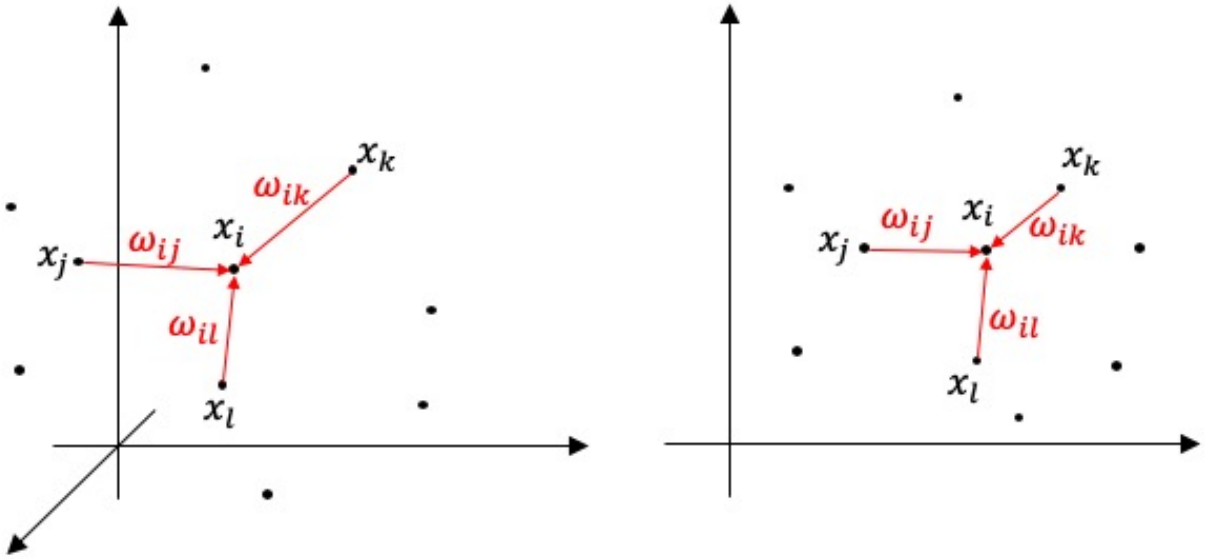
需要注意的是，Isomap 仅是得到了**训练样本** 在低维空间的坐标，对于**新样本**，入和将其映射到低维空间？这个问题的解决方案，是将**训练样本的高维空间坐标作为输入、低维空间坐标作为输出，训练一个回归学习器来对新样本的低维空间坐标进行预测**。

这仅是一个权宜之计，但目前也没有更好的办法。

LLE

简介

局部线性嵌入 (Locally Linear Embedding, LLE)与 Isomap 试图**保持近邻样本的距离**不同，LLE 视图**保持邻域内样本之间的线性关系**。



高维空间中的样本重构关系咋低维空间中得以保持

如上图，假定样本点 $\boldsymbol{x}_i$ 的坐标能通过它的邻域样本 $\boldsymbol{x}_j$、$\boldsymbol{x}_k$、$\boldsymbol{x}_l$ 的坐标通过下行组合重构出来，即：

$$\boldsymbol{x}_i = \omega_{ij}\boldsymbol{x}_j + \omega_{ik}\boldsymbol{x}_k + \omega_{il}\boldsymbol{x}_l \tag{E1}$$

LLE 希望上式的关系得以在低维空间保持。

推导

LLE 先为每一个样本 $\boldsymbol{x}_i$ 找到其近邻的下标集合 Q_i ，然后计算出基于 Q_i 中的每个样本点对 $\boldsymbol{x}_i$ 进行线性组合的重构系数 $\boldsymbol{\omega}_i$：

$$\begin{aligned} \min_{\boldsymbol{\omega}_1,\boldsymbol{\omega}_2,\ldots,\boldsymbol{\omega}_m} & \sum_{i=1}^{m} \left\| \boldsymbol{x}_i - \sum_{j\in Q_i} \omega_{ij}\boldsymbol{x}_j \right\|_2^2 \\ s.t. & \sum_{j\in Q_i} \omega_{ij} = 1 \end{aligned} \tag{E2}$$

其中 $\boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$ 均为已知，令 $C_{jk} = (\boldsymbol{x}_i - \boldsymbol{x}_j)^T(\boldsymbol{x}_i - \boldsymbol{x}_k)$， ω_{ij} 有闭试解：

$$\omega_{ij} = \frac{\sum_{k\in Q_i} C_{jk}^{-1}}{\sum_{l,s\in Q_i} C_{ls}^{-1}} \tag{E3}$$

LLE 在低维空间中保持 $\boldsymbol{\omega}_i$ 不变，于是 $\boldsymbol{x}_i$ 对应的低维空间坐标$\boldsymbol{z}_i$ 可以通过下式求解：

$$\min_{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_m} \sum_{i=1}^{m} \left\| \boldsymbol{z}_i - \sum_{j\in Q_i} \omega_{ij}\boldsymbol{z}_j \right\|_2^2 \tag{E4}$$

式（E2）与（E4）的优化目标相同，唯一区别是式（E2）中需要确定的是 $\boldsymbol{\omega}_i$ ，而（E4）需要确定的是 $\boldsymbol{x}_i$ 对应的低维空间坐标 $\boldsymbol{z}_i$ 。

令 $Z=(z_1,z_2,\ldots,z_m)\in\mathbb{R}^{d'\times m},(W)_{ij}=\omega ij$：

$$\boldsymbol{M}=(\boldsymbol{I}-\boldsymbol{W})^T(\boldsymbol{I}-\boldsymbol{W}) \tag{E5}$$

那么（E4）可以重写为：

$$\begin{aligned}&\min_Z tr(ZMZ^T)\\&s.t.\ \ ZZ^T=I\end{aligned} \tag{E6}$$

上式可以通过特征值分解求解，M 最小的 d' 个特征值对应的特征向量组成的矩阵即为 Z^T 。

LLE算法

输入： $D=\{x_1,x_2,\ldots,x_m\}$
近邻参数 k
低维空间维数d'

过程:
for i=1,2,...,m do
确定x_i 的 k 近邻
从式（E2）中求得$\omega_{ij},j\in Q_i$
对于 $j\notin Q_i$，令 $\omega_{ij}=0$
end for
从式（E5）中求得 M
对 M 进行特征值分解
return M 的最小 d' 个特征值对应的特征向量

输出： 样本即 D 在低维空间的投影 $Z=\{z_1,z_2,\ldots,z_m\}$

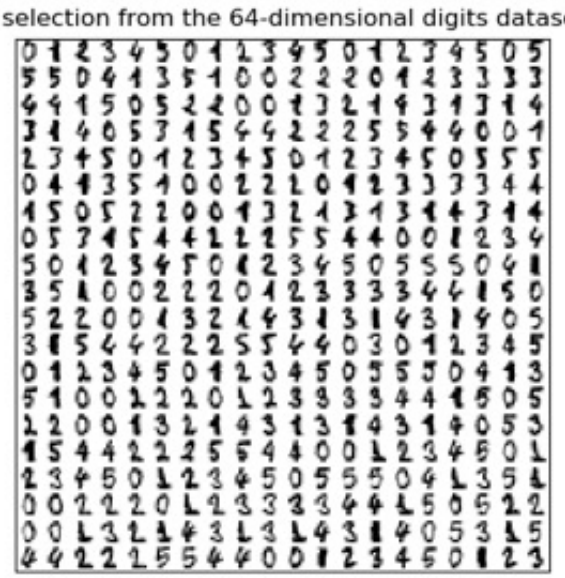
上面算法中循环的最后一部分显示出：对于不在样本 $\boldsymbol{x}_i$ 邻域区域的样本 $\boldsymbol{x}_j$ ，无论其如何变化都对 $\boldsymbol{x}_i$ 和$\boldsymbol{z}_i$ 没有任何影响；这种将变动限制在局部的思想在很多地方都有用。

总结

降维方法效果对比

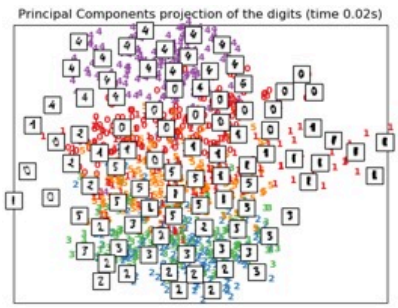
- sklearn - Manifold learning on handwritten digits

原始数据为：

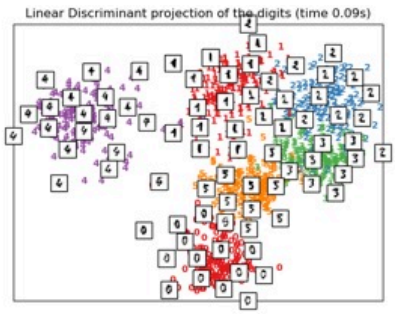


origin

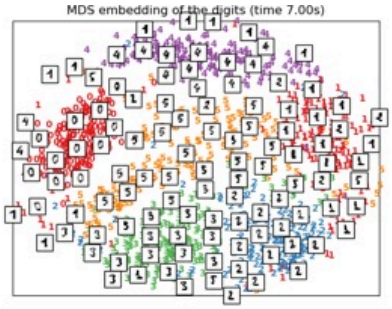
使用不同降维方法得到的结果：



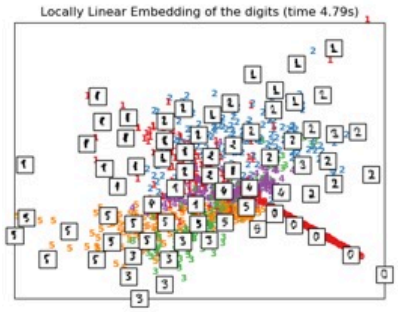
PCA



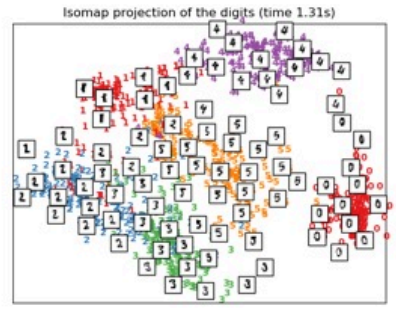
LDA



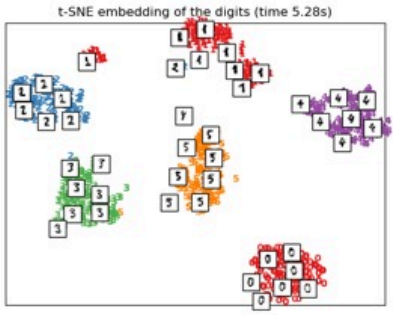
MDS



LLE



Isomap



t-SNE

降维优缺点

降维的优点：

- 通过降维，可以加速后续的训练算法（在某些情况下，它甚至可以去除噪声和冗余特征，使训练算法性能更好）；
- 降维后，数据更能够可视化，更容易直观地观测到最重要的特征；
- 节省存储空间。

降维的缺点：

- 降维势必伴随着信息的丢失，这可能会降低后续算法的性能；
- 增加了任务的支线，使任务更为复杂；
- 特征转化后变得难以直观解释。