PCA 主成分分析法

Dezeming Family

Dezeming

Art. No 0
ISBN 000–00–0000–00–0
Edition 0.0

Cover design by Dezeming Family

Published by Dezeming
Printed in China

目录

前言及简介

DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列书，可以从我们的网站 *[https://dezeming.top/]* 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

0.1 本书前言

数据降维在机器学习中非常重要。它的重要性不只是可以将复杂数据简化，还可以凝聚数据的最重要特征，从而更好地应用于样本训练和聚类。

降维有很多种方法，包括传统方法和深度学习方法。本书着重讲解 PCA 算法，该算法原理并没有那么复杂，但是非常有用。我们会对里面涉及到的多个概念和细节进行详细解释。

20210925：完成第一版。

1. 主成分分析的意义

本章从数学的角度来分析我们的降维目标和一些条件。

1.1 降维的目标

当我们要考虑把样本用来聚类或者训练分类回归模型时，如果样本的维度过高，可能效果会很不理想。例如一些文本或者图像，可能有几千几万维，因此有时候我们希望能够总结一些特征。

但通常我们不知道如何提取特征（对于图像，可以应用特征点检测、线条检测等方法；语音的波形频率特征），我们希望能找一个通用的方法，来帮我们把高维度的数据降低维度。

同时，降维可能会减少噪声的影响，因为降维可能只保留数据最有意义的特征，因此一些噪声便会被去除。

我们可能希望最后降维的结果在得到模型以后能够被解释，因此这时或许我们就得从数据本身的特性来分析如何降维，而不是套用统一的工具和方法。不过我们本文还是以研究通用的降维为主。

我们希望降维以后，数据能够比较明显地区分开，而不是都聚集在一起特别相似。如果降维以后数据值都很相近，降维也就失去了其意义。或许我们不能解释降维得到的结果，但如果对于不同的数据，降维以后的分布非常分散，我们就可以认为这个降维的算法是符合我们预期的。考虑人工简易降维方法：如果一个青蛙主要以颜色来进行降维，在降维后可能更接近于绿色的螳螂，但从类别上作为两栖动物它更应该和棕色的娃娃鱼相近。当总体样本分布更广的时候，青蛙与螳螂更相近的概率就会变低（大概是一种直觉）。

1.2 投影与基

在笛卡尔空间中，向量内积可以理解为投影：

$$\boldsymbol{x} \cdot \boldsymbol{y} = |\boldsymbol{x}||\boldsymbol{y}|cos(\alpha) \tag{1.2.1}$$

如果 $\boldsymbol{y}$ 是单位向量，则内积值就是 $\boldsymbol{x}$ 在 $\boldsymbol{y}$ 方向上的投影值。

在描述向量值时，是按照基来描述的。我们默认笛卡尔坐标系的基是 $(1,0)$ 和 $(0,1)$。想换一组基来描述，则只需要用向量跟这些基做内积，得到的值即为每个基的分量值。

我们设一个 $m \times n$ 矩阵 P 的行向量为 P_i，则对于某个 n 维向量 $\boldsymbol{a}$：

$$\begin{bmatrix} P_1 \\ P_2 \\ \dots \\ P_m \end{bmatrix} \boldsymbol{a} = \begin{bmatrix} P_1 \cdot \boldsymbol{a} \\ P_2 \cdot \boldsymbol{a} \\ \dots \\ P_m \cdot \boldsymbol{a} \end{bmatrix} \tag{1.2.2}$$

因此矩阵乘以某个向量，就可以看做是把这个向量变换到以矩阵行向量为基的空间中。但是为了变换以后更好描述，以及确保后面的运算的简单性，一般会使用标准正交基作为新坐标系的坐标向量。

1.3 协方差矩阵与最大可分性

举个例子，如果我们降维以后的目标空间是 3 维，也就是基是 3 维，但样本的维度是 10 维，我们就可以考虑使用某种映射关系将 10 维的样本变换到 3 维的空间上，其实就是用一个 3×10 的矩阵即可。

但我们最好不要随便找一个矩阵就开始降维，我们希望这个矩阵乘以样本以后能让样本更分散——也就是让信息量更大。

我们一般会用方差来表示样本的分散程度，对于一维样本向量 $\boldsymbol{x} = (x_1)$，其方差为表示如下。

$$Var(x_i) = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2 \tag{1.3.1}$$

当我们求出样本均值 μ 以后，我们可以将样本去均值化，也就是都减去均值，这样方差的计算方法就是：

$$x_i \leftarrow x_i - \mu \tag{1.3.2}$$

$$Var(x_i) = \frac{1}{m} \sum_{i=1}^{m} (x_i)^2 \tag{1.3.3}$$

因此我们可以把目标简化为，寻找一组基，使得映射以后的方差最大。

但多维空间中，样本协方差可以表示向量中每个变量的之间的线性相关性。第 i 个样本的第 j 个变量定义为 $x_{i,j}$：

$$\boldsymbol{x} = (x_1, x_2, \dots, x_n) \tag{1.3.4}$$

$$Cov(x_i, x_j) = \frac{1}{m} \sum_{k=1}^{m} (x_{k,i} - \mu_i)(x_{k,j} - \mu_j) \tag{1.3.5}$$

$$x_i \leftarrow x_i - \mu_i, \quad i = 1, 2, \dots, n \tag{1.3.6}$$

$$Cov(x_i, x_j) = \frac{1}{m} \sum_{i=1}^{m} (x_{k,i} \cdot x_{k,j}) \tag{1.3.7}$$

我们可以构建出协方差矩阵。注意上面对样本方差进行的是有偏估计，我们默认使用的样本就是总体（比如当你就是把这些样本降维以后再用来聚类）。当使用样本训练深度学习模型时，我们或许认为全部样本只是总体的一部分，这个时候可以使用无偏估计，即分母是 $m-1$（详情请见《样本估计》）。

我们希望投影以后各个变量之间不具有线性相关性，即协方差尽量为 0。

我们把 m 个 n 维样本进行排列，注意每个样本都已经去均值化了。其中，第 i 个样本的第 j 个变量定义为 $x_{i,j}$：

$$X = \begin{bmatrix} x_{1,1} & x_{2,1} & ... & x_{m,1} \\ x_{1,2} & x_{2,2} & ... & x_{m,2} \\ . & . & ... & . \\ x_{1,n} & x_{2,n} & ... & x_{m,n} \end{bmatrix} \tag{1.3.8}$$

所以可以得到协方差矩阵 Cov 为：

$$Cov = \frac{1}{m}XX^T = \frac{1}{m}\begin{bmatrix} \sum_{i=1}^m x_{i,1}^2 & \sum_{i=1}^m x_{i,1}x_{i,2} & ... & \sum_{i=1}^m x_{i,1}x_{i,n} \\ \sum_{i=1}^m x_{i,2}x_{i,1} & \sum_{i=1}^m x_{i,2}^2 & ... & \sum_{i=1}^m x_{i,2}x_{i,n} \\ . & . & ... & . \\ \sum_{i=1}^m x_{i,n}x_{i,1} & \sum_{i=1}^m x_{i,n}x_{i,2} & ... & \sum_{i=1}^m x_{i,n}x_{i,n} \end{bmatrix} \tag{1.3.9}$$

因为协方差为 0，也就是说除了对角线上的元素，其他元素都尽量为 0。由 DezemingFamily 的《矩阵的相似对角化》和《特征分解与奇异值分解》，我们可以将这个实对称矩阵进行相似对角化，即：

$$Cov = P\Lambda P^T \tag{1.3.10}$$

这里的 P 是酉矩阵，其转置等于逆矩阵。这里我们将 Λ 矩阵的对角线元素按照大小设为从上到下排列，即上面的方差大，下面的方差小。

我们设原始数据 X 对应的协方差矩阵为 Cov；使用某个基矩阵 B 映射以后的数据 BX 协方差矩阵为 Cov_B：

$$Cov_B = \frac{1}{m}(BX)(BX)^T \tag{1.3.11}$$

$$= B(\frac{1}{m}XX^T)B^T \tag{1.3.12}$$

$$= BCovB^T \tag{1.3.13}$$

即，$BCovB^T$ 会得到一个对角阵，B 就是上面的 P^T。

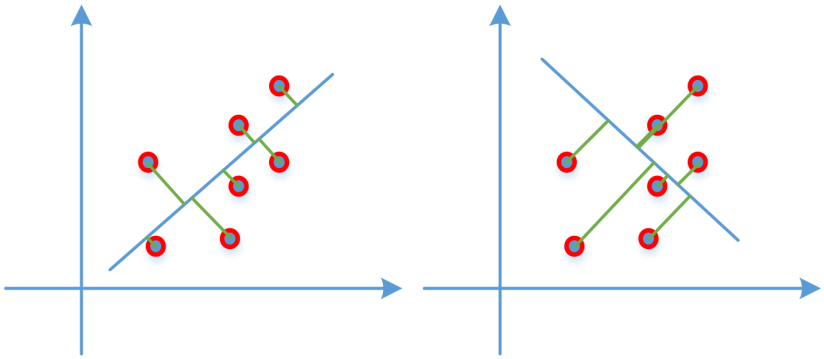
1.4 最近重构性

除了最大可分性理论，也可以使用最近重构性。由于并不常用，所以这里只简单提一下。

可以理解为拟合一个超平面，让这个超平面更好地拟合样本位置，对于 n 维样本，超平面为：

$$w_1x_1 + w_2x_2 + ... + w_nx_n + b = 0 \tag{1.4.1}$$

投影距离越短，说明信息丢失越少，以二维为例，左边的投影距离小，投影以后分布更广；右边投影距离大，投影以后分布更密集，甚至有些不同的样本被投影到了同一个位置。



在这种条件下，我们的优化目标是使平方误差尽可能小。

1.5 拉格朗日乘子法解法

很多函数优化问题都可以用拉格朗日乘子法去求解。这个问题用拉格朗日乘子法去思考会有更清晰的认识（见 DezemingFamily 的《拉格朗日乘子法》）。

我们先说一下结论：为了让方差更大，最好的投影方向就是最大特征值对应的特征向量。第二好的投影方向就是第二大特征值对应的特征向量。如果我们只想把数据投影到一维，那么只需要知道最大特征值对应的特征向量即可，但有时候样本协方差矩阵有好几个比较大的特征向量，说明这几个方向的投影都很重要。

把一个去均值化的样本点 $\boldsymbol{x}$ 投影到一个基上可以写为 $\boldsymbol{x}^T\boldsymbol{w}$，投影以后的方差就是：

$$D(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}_i^T\boldsymbol{w})^T(\boldsymbol{x}_i^T\boldsymbol{w}) \tag{1.5.1}$$

$$= \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{w}^T\boldsymbol{x}_i\boldsymbol{x}_i^T\boldsymbol{w} \tag{1.5.2}$$

$$= \boldsymbol{w}^T\Big(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^T\Big)\boldsymbol{w} \tag{1.5.3}$$

我们知道 $\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^T$ 是协方差矩阵，对于单位基，应该满足 $\boldsymbol{w}^T\boldsymbol{w}=1$，因此最终得到优化函数：

$$max\{\boldsymbol{w}^T\Big(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^T\Big)\boldsymbol{w}\} \tag{1.5.4}$$

$$s.t.\quad \boldsymbol{w}^T\boldsymbol{w}=1 \tag{1.5.5}$$

构造拉格朗日函数并对 $\boldsymbol{w}$ 求偏导：

$$L(\boldsymbol{w},\lambda)\boldsymbol{w}^T\Big(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^T\Big)\boldsymbol{w}+\lambda(1-\boldsymbol{w}^T\boldsymbol{w}) \tag{1.5.6}$$

$$\frac{\partial L(\boldsymbol{w},\lambda)}{\partial \boldsymbol{w}} = \Big(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^T\Big)\boldsymbol{w}-\lambda\boldsymbol{w}=0 \tag{1.5.7}$$

可以看到，取最大值的时候，$\boldsymbol{w}$ 恰好是特征向量，而且我们代入到方差中就能得到：特征值越大，方差越大。

$$D(\boldsymbol{x}) = \boldsymbol{w}^T\Big(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x_i}\boldsymbol{x_i}^T\Big)\boldsymbol{w} \tag{1.5.8}$$

$$= \lambda \boldsymbol{w}^T\boldsymbol{w} = \lambda \tag{1.5.9}$$

2. 主成分分析算法

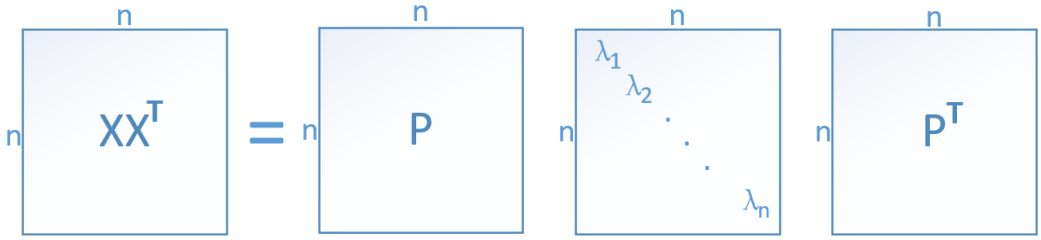
本章介绍主成分分析的实际运算过程。

2.1 PCA 求解步骤

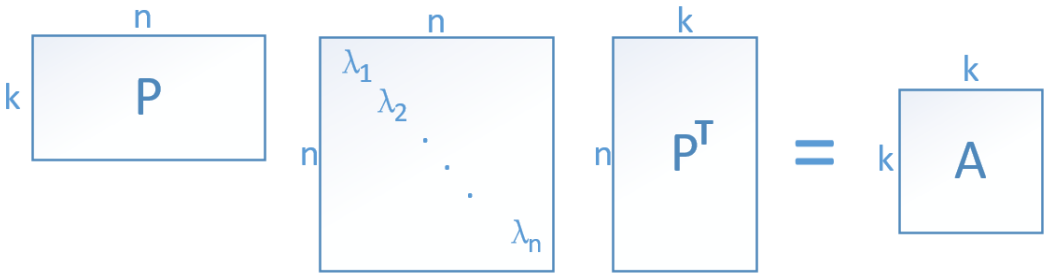
步骤一：将 m 个 n 维样本构建 $n \times m$ 矩阵 X，并求每一行的均值，然后去均值化。

步骤二：计算协方差矩阵 $Cov = \frac{1}{m} X X^T$，矩阵大小为 $n \times n$。

步骤三：求协方差矩阵的特征值和特征向量，进行相似对角化，对角矩阵对角线上的值是原矩阵的特征值，且从上到下递减，即 $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$：



步骤四：选出前 k 的特征值的特征向量作为矩阵 P，该矩阵相乘相当于只投影到特征值比较大的特征向量位置：



步骤五：用矩阵 P 乘以 X 的新的维度的样本矩阵，该矩阵一共有 k 维。

注意均值必须是从训练集中计算的，而不能是从全部数据中计算得到（因为除了聚类，很多时候我们并不能得到全部数据），得到的降维矩阵也要同时作用于验证集和测试集。

2.2 PCA 的好处和坏处

好处主要是两个方面：

- 缓解维度灾难：对于一些信息量较少的维度会被舍去，使维度降低。
- 是一种去噪手段：特征值小的特征向量可能是由于数据噪声波动带来的。

坏处体现在，有可能舍去的信息是重要信息，只是我们在训练数据中给它们的权重比较低或者不合适的归一化造成的，比如人的身高体重年龄等变量与人的体质之间的关系，如果把所有变量都进行归一化，那么有的明显更重要的信息就没法被突出，因此会被当做不重要的信息。

2.3 PCA 与 SVD

关于奇异值分解 SVD 的内容可以参考 DezemingFamily 的《特征分解与奇异值分解》。

对某个矩阵 A（不必要求是方阵）做奇异值分解以后，就能得到：

$$A_{n,m} = U_{n,n}\Sigma_{n,m}V^T_{m,m} \approx U_{n,k}\Lambda_{k,k}V^T_{k,m} \tag{2.3.1}$$

其中，$\Sigma_{n,m}$ 除了主对角线上其他元素都是 0，$\Lambda_{k,k}$ 取最大的 k 个特征值。

SVD 中，$U_{n,n}$ 和 $V_{m,m}$ 分别是 AA^T 和 A^TA 的特征向量构成的矩阵，有迭代求解的高效算法，避免计算 AA^T 和 A^TA 的特征值。因为 SVD 的优化迭代算法，方阵的特征分解计算比 SVD 要慢很多，因此很多时候，PCA 都会使用 SVD 进行降维。

我们把样本矩阵用这种方法分解以后，只保留最重要的 k 维，我们将左边的矩阵 $U_{k,n}$ 转置以后乘以原数据：

$$U_{k,n}A_{n,m} = A'_{k,m} \tag{2.3.2}$$

就将数据降维到了 k 维。

SVD 得到的矩阵有时候不具有可解释性，很多时候仅作为一种辅助提高训练效果的方法。

最后我们需要注意的是，有人用左右奇异向量来描述降维，但 PCA 中对角化矩阵是左奇异矩阵还是右奇异矩阵是与数据的排列有关的。如果 m 个 n 维数据排列为下面的方式，则 U 矩阵等同于 PCA 中的对角化矩阵 P。

$$X = \begin{bmatrix} x_{1,1} & x_{2,1} & \dots & x_{m,1} \\ x_{1,2} & x_{2,2} & \dots & x_{m,2} \\ . & . & \dots & . \\ x_{1,n} & x_{2,n} & \dots & x_{m,n} \end{bmatrix} \tag{2.3.3}$$

如果数据排列的方式是下面的方式，则 V 矩阵等同于 PCA 中的对角化矩阵 P。

$$X = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \dots \\ \boldsymbol{x_m}^T \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ . & . & \dots & . \\ x_{m,1} & x_{m,2} & \dots & x_{m,n} \end{bmatrix} \tag{2.3.4}$$

3. 降维在图像中的应用

本章介绍实际的图像降维测试效果。

3.1 测试代码

因为 PCA 都是些数学运算，这里就不再自己写底层代码测试了，而是使用现成的库。

```
1  import numpy as np
2  from sklearn.decomposition import PCA
3  import cv2
4
5  img = cv2.imread('image.jpg')
6  #[宽,高,通道]，这里采用G通道，注意OpenCV的通道顺序是BGR。
7  data = img[:,:,1]
8  dataMat = np.mat(data)
9  #降维到100维
10 pca = PCA(n_components=100).fit(data)
11 # 降维
12 x_new = pca.transform(data)
13 # 还原降维后的数据到原空间
14 recdata = pca.inverse_transform(x_new)
15 cv2.imshow('recdata', np.array(recdata, dtype='uint8'))
16 cv2.waitKey(0)
```

其中，*n_components* 表示降维的维数。

计算 PCA 信息丢失率的方式如下，就是用差值矩阵每个元素的平方和除以原矩阵每个元素

的平方和，A 是原矩阵，B 是降维以后再还原以后的矩阵：

$$\frac{\sum_{i=1}^{row}(A_i - B_i)(A_i - B_i)^T}{\sum_{i=1}^{row} A_i A_i^T} \tag{3.1.1}$$

程序写为：

```
# 输入：原数据，降维然后恢复后的数据
def calLossRate(data, recdata):
    sum_denominator = 0
    sum_numerator  = 0
    Diff_value = data - recdata # 计算两幅图像之间的差值矩阵
    # 计算两幅图像之间的误差率，即信息丢失率
    for i in range(data.shape[0]):
        sum_denominator += np.dot(data[i],data[i])
        sum_numerator += np.dot(Diff_value[i], Diff_value[i])
    print('loss rate: ', sum_numerator/sum_denominator)
# 计算信息丢失率
calLossRate(np.array(data, dtype='double'), np.array(recdata, dtype='
    double'))
```

我们令 *n_components* 的值从 400 变到 1，来查看效果：

可以看到，PCA 在降维的时候会带来一定的精度损失，但大致能保留主要成分，这是因为用来降维的矩阵会保证降维以后的数据分布尽可能大，从而丢失更少的细节。这张 400×400 的图像降维到 30，也能比较清晰地看到大致概貌。

Bibliography

[1] https://zhuanlan.zhihu.com/p/77151308

[2] https://blog.csdn.net/lanyuelvyun/article/details/82384179

[3] https://my.oschina.net/findbill/blog/535044

[4] https://blog.codinglabs.org/articles/pca-tutorial.html

[5] https://zhuanlan.zhihu.com/p/29846048

[6] https://blog.csdn.net/zhy1887996acm/article/details/104917269