Principal Component Analysis

上一篇文章介绍了SVD分解，本文承接上文，介绍数据处理中的另一个重要技巧——主成分分析（PCA）。

首先不着急研究PCA的原理，我们先来看看PCA是干什么的。

PCA是一种数据降维的手段，将所获得的数据，通过PCA处理后，就可以消除其中的冗余信息，同时将数据中的重要部分暴露出来。这个方法可以去掉数据中具有强烈相关性的部分，使得不同类别的数据尽可能不相关。另外在降维后，删去无关项后再升维，可以使得重建误差最小，这就是优化目标。

听起来很酷，我们是不是可以在所有的数据处理过程中，做一次PCA，就可以找到数据中最本质的部分呢？

首先我们来考虑一个例子，假设目前有一个弹簧球在二维平面上沿着某条直线运动，那么这个弹簧球的运动在该条直线上应该符合一个非常简洁的运动方程。此时我们用数个摄像头来记录这个弹簧球的运动，最后每个摄像头记录下一组运动数据。可想而知，由于摄像头不一定沿着这条直线，因此每个摄像头以自己为坐标原点，记录运动数据，得到的结果不会符合那个非常简洁的运动方程，其次多个摄像头记录同一个运动，不同摄像头间的数据也具有冗余性。同时摄像头记录信息也会存在噪声干扰。我们将所有的数据集合起来，组成一个矩阵X，希望通过PCA，得到最简洁的运动形式Y。

$$\boldsymbol{PX = Y}$$

由上式可得，矩阵X经过一个矩阵乘法（也就是线性变换），就得到了一个新的矩阵Y。而这个新的数据表示方式Y，应当具有哪些性质呢？消除冗余性：也就是说新的坐标表示Y中，矩阵的各行之间，应当不相关，亦即它们是正交的。

$$\begin{aligned}\boldsymbol{Y &= PX}\\ \boldsymbol{Y^T &= X^T P^T}\\ \boldsymbol{YY^T &= PXX^T P^T}\end{aligned}$$

因为Y是正交矩阵，因此上式的结果应当是一个对角矩阵，那么式子的右边如何能变成一个对角矩阵呢？回想起上篇文章的内容：

> 对称矩阵可以被特征值分解。

$$\boldsymbol{XX^T = Q\Sigma Q^T}$$

同时Q是个正交矩阵，$\boldsymbol{\Sigma}$是个对角阵。

则：

$$YY^T = PQ\Sigma Q^T P^T$$

如果我们令$P = Q^T$，那么$YY^T = \Sigma$，这样我们就消除了数据中的冗余性。

$$Y^T Y = X^T P^T P X = X^T X$$

当然上式没有什么意义。

同时上述操作后还有一个非常好的性质，我们知道，特征值分解得到的矩阵Σ的对角线上，奇异值从大到小排列。

$$\begin{aligned} XX^T &= Q\Sigma Q^T \\ &= \sigma_1 qq^T + \ldots + \sigma_n qq^T \end{aligned}$$

如果我们省略后面比较小的σ，就可以在数据降维的同时，尽可能使得重建结果和原数据相近。

对于PCA来讲，就是寻找一个新的数据表示方式，使得信噪比（signal-to-noise ratio）最大。

$$SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}}$$

在PCA中假设噪声信号的方差应当较小，而有意义信号的方差应当较大，特征值分解的结果，可以反映方差大小的顺序。

统计知识复习

回顾一下统计知识，方差的定义以及协方差的定义：

$$A = \{a_1, a_2, \ldots, a_n\} \quad B = \{b_1, b_2, \ldots, b_n\}$$
$$\sigma_A^2 = \frac{1}{n}\sum_i (a_i - \bar{a})^2 \quad \sigma_B^2 = \frac{1}{n}\sum_i (b_i - \bar{b})^2$$
$$\sigma_{AB}^2 = \frac{1}{n}\sum_i (a_i - \bar{a})(b_i - \bar{b})$$

但是实际中σ的系数为$n-1$，这是因为样本均值和实际均值有误差，因此为了得到方差的无偏（unbiased）估计，所以要调整计算方式。

$$\begin{aligned} S &= \frac{1}{n}\sum_i (a_i - \bar{a})^2 \\ &= \frac{1}{n}\sum_i (a_i - \mu - (\bar{a} - \mu))^2 \\ &= \frac{1}{n}\sum_i ((a_i - \mu)^2 - 2(a_i - \mu)(\bar{a} - \mu) + (\mu - \bar{a})^2) \\ &= \frac{1}{n}(n * var(X) - n * Var(\bar{X})) \\ &= \frac{n-1}{n}\sigma^2 \end{aligned}$$

因此需要对S乘以$\frac{1}{n-1}$来进行无偏估计。

为了简单起见，我们将所有数据进行中心化处理，使得均值为0。

我们可以将方差和协方差的计算方式表达成矩阵形式：

$$\sigma_{ab}^2 = \frac{1}{n}ab^T$$

对于矩阵形式的数据：

$$X = [x_1, \ldots, x_m]^T$$
$$C_X = \frac{1}{n}XX^T$$

C_X就是协方差矩阵，对角线上的元素都是方差，其他位置都是协方差。协方差反映了数据中的冗余和噪声（即两个系数非常具有相关性，那么更加适合使用一个系数来表示，这也是降维的本质）。

在对角线上，较大的数值表示了较大的方差，同时也代表这个测量值比较重要。在非对角线上，较大的数值代表较大的相关性，即冗余性质。

降维的目的是：

1、最小化冗余性质；

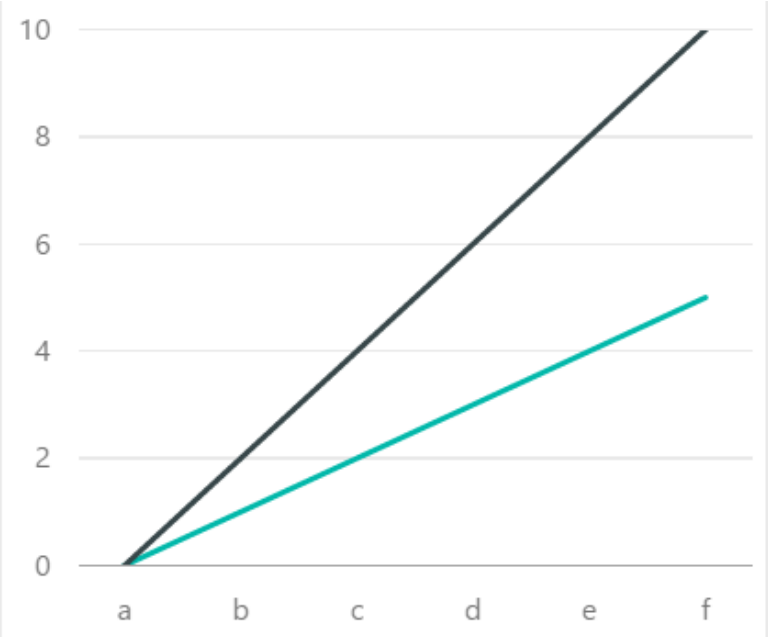
2、最大化信号。

那么理想化的协方差矩阵，应当是非对角项均是0，因此必须是个对角阵，而对角线应当从大到小排列，这和第一部分的结论一致。

PCA不是万能的

PCA做了非常多的假设：

- 通过线性变换，可以得到一个更好表示数据性质的基底。（现在有的工作在讨论非线性变换的基底，即Kernel PCA，因为线性变换并不足够，有些数据之间有非常复杂的关系。）
- 大的方差表示了更重要的信息，也是我们更想保留的。（这一点有时是不正确的。）
- 理想的新基地之间相互正交。有的时候，真实的数据并不相互正交，如下图所示。假设这两组数据统计无关，然而并不正交，那么PCA就会失效。



PCA步骤

因此整个PCA，被转换为一个矩阵分解的问题，整个PCA的步骤为：

1、对数据减去均值，进行中心化；

2、计算$\boldsymbol{C_X}$的特征向量；

3、得到特征向量组成的矩阵P，并计算新的数据表示$\boldsymbol{Y = PX}$。

而在上一篇文章中，我们知道，对矩阵A进行SVD分解，会求出两个特征向量矩阵，分别对应$\boldsymbol{AA^T}$和$\boldsymbol{A^TA}$的特征向量，在这里，我们想要$\frac{1}{n}\boldsymbol{XX^T}$的特征向量，因此可以直接对$\frac{1}{\sqrt{n}}\boldsymbol{X}$进行SVD分解即可。

总结

PCA是一种非参数化的方法，得到的结果是线性变化基底，各个基底之间相互正交。

它在各变量线性相关的情况下很有效，但是在非线性和有用的数据非正交的情况下无效。

非线性的情况下，我们可以使用流形学习和Kernel PCA解决。

非正交的情况下，我们要求分解的结果必须统计无关，可以使用ICA来解决，但是计算量更大。

代码

```
import numpy as np
import math
from scipy import linalg
from sklearn import decomposition
from numpy import linalg as LA

def PCA_by_SVD(data):
    m, n = data.shape
    mn = np.mean(data, axis = 1)
    mean_data = []
    mean_data_son = []
    for i in range(m):
        for j in range(n):
            mean_data_son.append(mn[i])
        mean_data.append(mean_data_son)
        mean_data_son = []
    mean_data = (data - np.array(mean_data))
    u, s, v = linalg.svd(mean_data.T / math.sqrt(n - 1))
    print np.dot(v, mean_data)

def PCA_by_Eigenvector(data):
    m, n = data.shape
    mn = np.mean(data, axis = 1)
    mean_data = []
    mean_data_son = []
    for i in range(m):
        for j in range(n):
            mean_data_son.append(mn[i])
        mean_data.append(mean_data_son)
        mean_data_son = []
    mean_data = (data - np.array(mean_data))
    w, v = LA.eig(np.dot(mean_data, mean_data.T) / (n -
1))
    indices = sorted(range(len(w)), key = lambda k :
w[k])
    PC = []
    for i in range(len(indices)):
        PC.append(v[indices[i]])
    PC = np.array(PC)
    print np.dot(PC.T, mean_data)

#Small Ball's Motion
data = np.array([[1, 2, 3, 4], [1, 2, 3, 4]])

#Get result from PCA method of sklearn
pca = decomposition.PCA()
data_trans = pca.fit_transform(data.T)

print data_trans
```

```
print data_trans

#Perform PCA Using SVD
PCA_by_SVD(data)

#Perform PCA Using Eigen decomposition
PCA_by_Eigenvector(data)
```

最终得到的结果是：

```
[[ -2.12132025e+00   4.64292107e-08]
 [ -7.07106829e-01  -1.26625146e-08]
 [  7.07106829e-01   1.26625128e-08]
 [  2.12132049e+00   3.79875367e-08]]
[[-2.12132034 -0.70710678  0.70710678  2.12132034]
 [ 0.          0.          0.          0.        ]]
[[ -2.12132034e+00  -7.07106781e-01   7.07106781e-01
2.12132034e+00]
 [  2.22044605e-16   5.55111512e-17  -5.55111512e-17
-2.22044605e-16]]
```

还原了最初简单的运动形式。