2020 年春季学期

计算学部《机器学习》课程

Lab 4 实验报告

姓名	梅智敏
学号	1183710118
班号	1837101
电子邮件	1044388658@qq.com
手机号码	13385658102

PCA模型实验

一、实验目的

目标

实现一个PCA模型，能够对给定数据进行降维（即找到其中的主成分）

测试

1. 首先人工生成一些数据（如三维数据），让它们主要分布在低维空间中，如首先让某个维度的方差远小于其它唯独，然后对这些数据旋转。生成这些数据后，用你的PCA方法进行主成分提取。
2. 找一个人脸数据（小点样本量），用你实现PCA方法对该数据降维，找出一些主成分，然后用这些主成分对每一副人脸图像进行重建，比较一些它们与原图像有多大差别（用信噪比衡量）。

二、实验环境

- win10
- python 3.7.4
- pycharm 2020.2

三、数学原理

3.1 PCA的目的

$PCA(Principal Component Analysis)$是一种常用的数据分析方法。它通过线性变换将原始数据用一组线性无关的$base$来表示，可用于**提取数据的主要特征分量**，常用于高维数据的**降维**，以减少资源消耗。

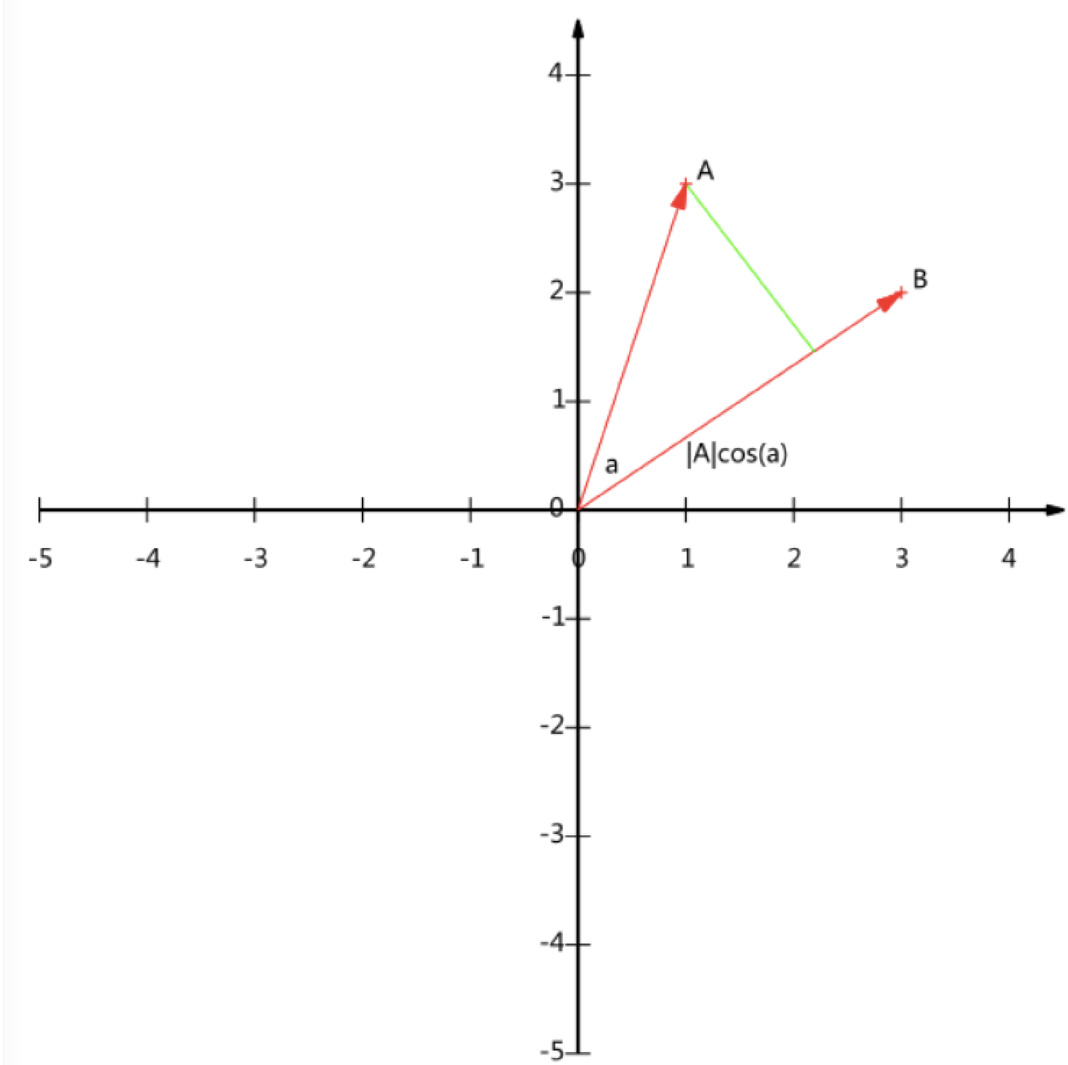
3.2 内积与投影

向量的内积被定义为

$$(a_1, a_2 \dots a_n)^T \cdot (b_1, b_2 \dots b_n)^T = a_1 b_1 + a_2 b_2 + \dots a_n b_n$$

内积运算将两个向量映射为一个实数

它的几何意义在于：相当于一个向量在另一个向量方向上的投影乘以另一个向量的模长



例如，上图中$A=(a_1,a_2),B=(b_1,b_2)$

则$A\cdot B=|A|cos\alpha|B|$，现欲求A向量在B向量方向上的投影（即以该方向上的坐标），即$|A|cos\alpha=\frac{A\cdot B}{|B|}$，倘若$|B|=1$，则简化为$|A|cos\alpha=A\cdot B$

3.3 基变换

> 我们想要准确描述一个向量，首先需要一组基；然后给出该向量在各个基$base$方向上的投影即可。而我们对于基的要求就是线性无关且能"张成"对应的向量空间，通常使用正交基，因为正交基拥有一些良好的性质，但是非正交基也可以。

现在介绍基变换，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：

$$\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \cdots & a_M \end{pmatrix} = \begin{pmatrix} p_1a_1 & p_1a_2 & \cdots & p_1a_M \\ p_2a_1 & p_2a_2 & \cdots & p_2a_M \\ \vdots & \vdots & \ddots & \vdots \\ p_Ra_1 & p_Ra_2 & \dots & p_Ra_M \end{pmatrix}$$

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。

3.4 最大可分性

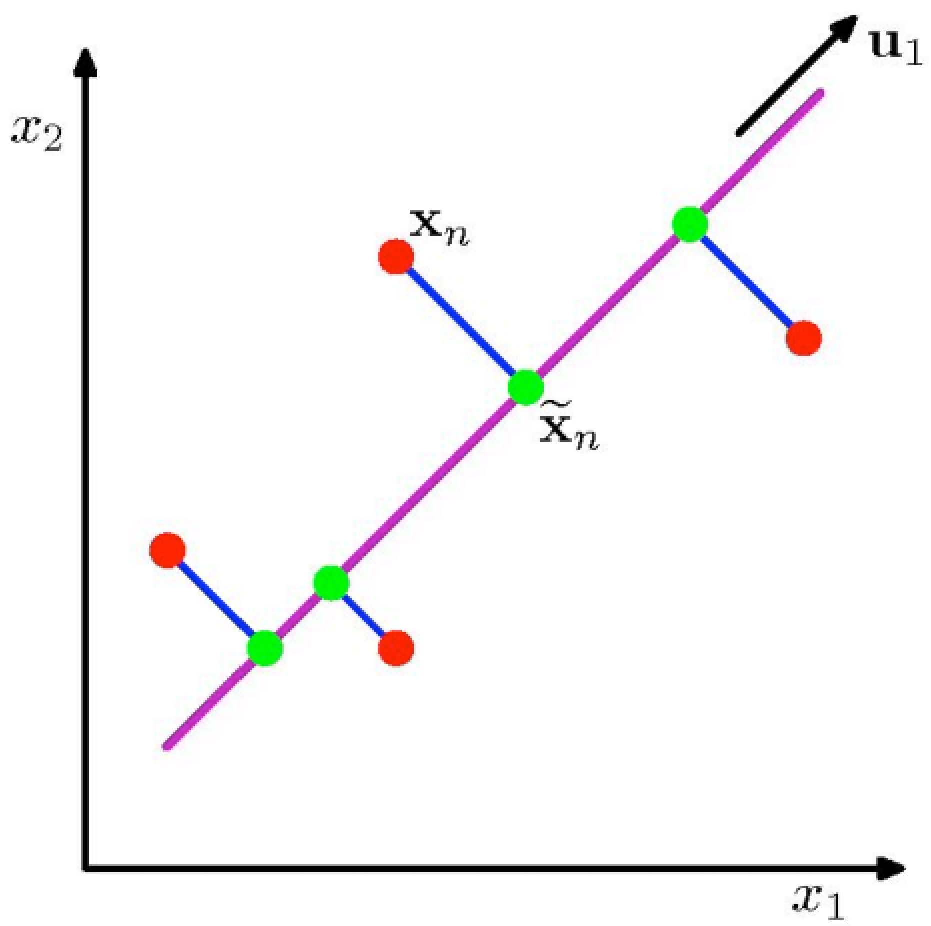
前面我们已经介绍了内积与投影的关系，以及基变换的知识。那么我们如何才能找到最合适的一组基，并使用它来降维呢？

前面已经得到结论：

我们可以将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。

我们想要获得某个向量在一组基上的坐标，只需分别求出该向量在各个基方向上的投影值即可。

而我们希望降维后的数据所保存的信息尽可能多，即降维后**各个维度数据内**的信息熵越大越好，而信息熵往往和方差有着正相关关系。同时，我们希望降维后**各个维度数据之间**相互独立，不存在相关关系。



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，其目标是选择 K 个单位正交基，使得原始数据变换到这组基上后，各变量两两间协方差为 0，而变量方差则尽可能大（在正交的约束下，取最大的 K 个方差）。**

3.5 矩阵对角化

我们现在拥有数据矩阵$X_{m\times n}$，则$C_{n\times n}=\frac{1}{m}XX^T$就是$X$的协方差矩阵

根据我们的优化条件，我们需要将**降维后的数据集的的协方差矩阵**对角线外的其他元素化为0，并且在对角线上将元素按从大到小进行排列，再选择最大的前K行即可。

设原始数据集X对应的协方差矩阵为C，P是一组基按行组成的矩阵，设Y为X做基变换之后的数据矩阵，则$Y=PX$，再设D为Y对应的协方差矩阵，有

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

故，我们的**优化目标转化为：**

- **需要寻找P，满足PCP^T是个对角矩阵，并且对角元素按大小依次排列，那么P的前的前 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。**

接下来，我们就去寻找这样的P

原始数据X对应的协方差矩阵C是一个对称矩阵，它在线性代数中有一系列非常好的性质：

- 实对称矩阵不同特征值对应的特征向量必然正交。
- 设特征向量λ的重数为 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 λ 个特征向量单位正交化（可利用施密特正交化）。

那么根据上面2条性质，我们可推出定理：

一个 n 行 n列的实对称矩阵一定可以找到 n个单位正交特征向量

将上述定理应用到我们的协方差矩阵C上，得到n个单位正交特征向量$e_1,e_2...e_n$，我们将其按列排成矩阵：

$$E=(e_1,e_2...e_n)$$

依据线性代数的知识，我们可以得到：

$$
E^TCE=\Lambda=\begin{pmatrix}
\lambda_1 & & & \\
& \lambda_2 & & \\
& & \ddots & \\
& & & \lambda_n
\end{pmatrix} \tag{2}
$$

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）。

将（2）式和（1）式进行对比，这样，我们就发现了需要的矩阵$P=E^T$。

- **我们若是想要降到K维，只需要取P的前K行作为K组基即可，也就是C的特征值最大的K个特征向量（已经单位正交化）。**
- **$Y=PX$就是降维之后的数据矩阵**

四、实验过程

4.1 自己生成数据测试

- 生成数据

为了便于可视化，我自己产生的数据就是2维及3维高斯分布数据集。

且为了让这些数据集主要分布在低维空间中，只需让某个维度的方差远远小于其他维度即可。

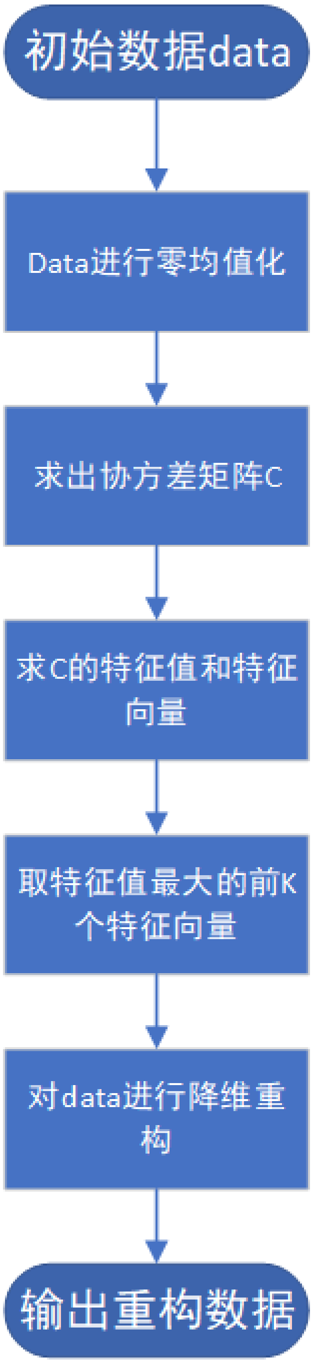
```
    if data_dimension is 2:
        mean = [-3, 4]
        # 让某个维度的方差远小于其他维度
        cov = [[1, 0], [0, 0.01]]
    elif data_dimension is 3:
        mean = [2, 8, -5]
        cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
    else:
        assert False

    # 产生shape = (D,M)的数据矩阵
    data = np.random.multivariate_normal(mean, cov, size=number).T
    if data_dimension is 3:
        # 绕z轴旋转数据点
        data = rotate(data, 40 * np.pi / 180, 'z')
```

注意，在产生三位数据时，会额外添加一个绕Z轴旋转的操作

- PCA算法

数学原理部分已经在前面叙述过，算法的思路为：



代码如下:

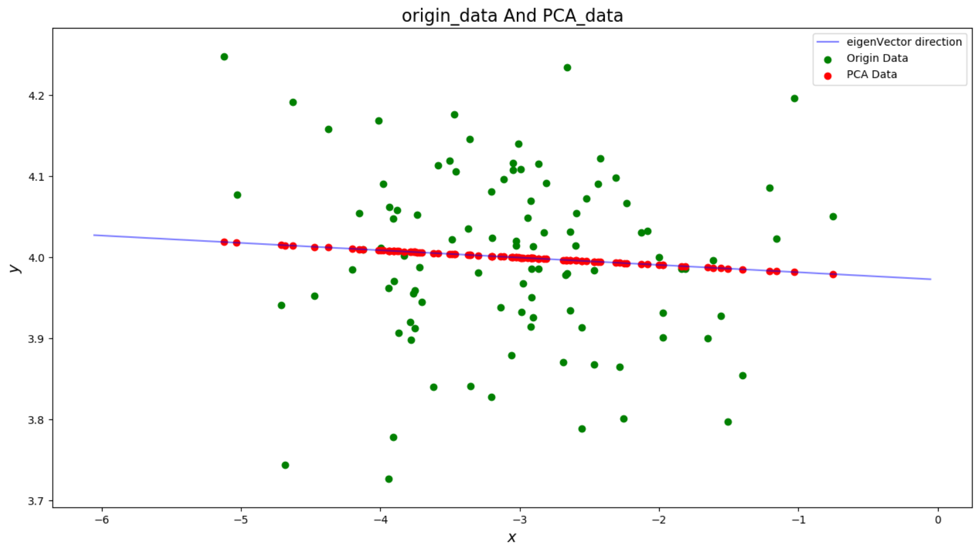
```
def PCA(data, k):
    """
    将数据data从D维降到k维
    :param data: 数据矩阵(D*N),D表示维度, N表示样本点的个数
    :param k: 把数据降到目标维数
    :return: 零均值化之后的数据矩阵, 特征值矩阵, 均值矩阵, 重构之后的数据矩阵 (仍然 D*N)
    """
    dim = data.shape[0]
    mean = np.mean(data, axis=1)
    c_data = np.zeros(data.shape)
    for i in range(dim):
        # 零均值化后得到c_data (D*N)
        c_data[i] = data[i] - mean[i]
    # 求出协方差矩阵
    covMat = np.dot(c_data, c_data.T)
    # 对协方差矩阵covMat(D*D)求特征值和特征向量
```

```
    # eigenVectors的每一列对应一个特征向量
    eigenValues, eigenVectors = np.linalg.eig(covMat)
    # 特征值排序
    eigValIndex = np.argsort(eigenValues)
    # 取前k个特征值对应的特征向量 shape = (D*k)
    rightEigenVector = eigenVectors[:, eigValIndex[:-(k + 1):-1]]
    # 一旦降维维度超过某个值，特征向量矩阵将出现复向量，对其保留实部
    rightEigenVector = np.real(rightEigenVector)
    # 计算降维后的数据(K*N)
    tmp_data = np.dot(rightEigenVector.T, c_data)
    # 重构之后的数据
    recon_data = np.zeros(data.shape)
    for i in range(dim):
        recon_data[i] = np.dot(rightEigenVector[i], tmp_data) + mean[i]
    return c_data, rightEigenVector, mean, recon_data
```

- 降维前后对比

二维情况：

绿色的点表示**原始数据**，红色的点表示将数据降维后的**重构数据**，图中直线标明了投影方向，即PCA寻找到的**主成分**。



可见我们将二维数据降到一维后，依然能够大致反应出原先各个数据点的位置分布情况，图中蓝色的直线标明了**特征值最大的那个特征向量**的方向，也就是我们选择作为投影方向的向量。

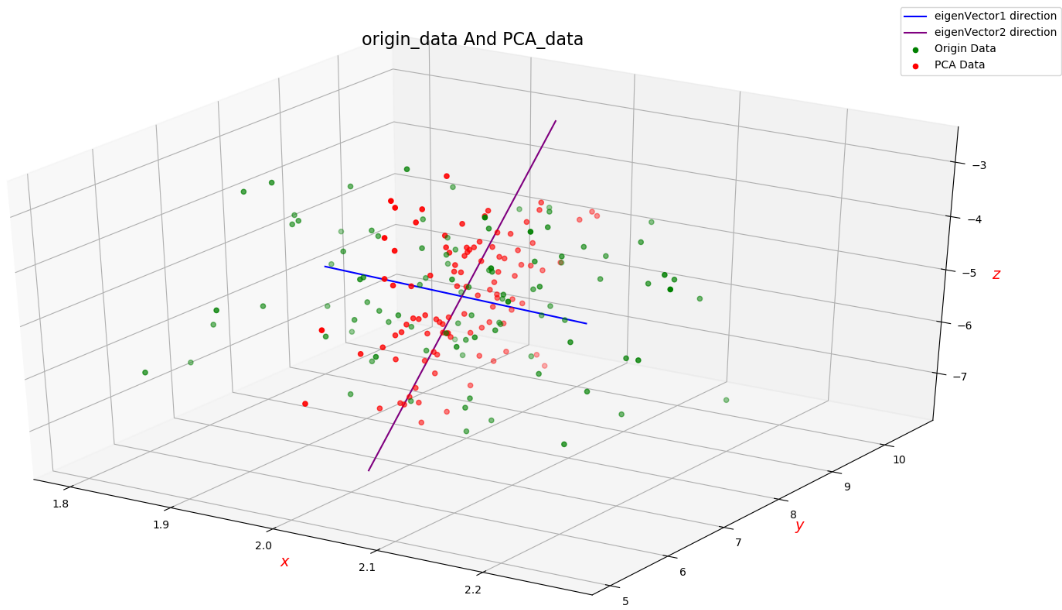
```
main
E:\Anaconda3\python.exe D:/PyCharmWorkSpace/ML/PCA/main.py
特征值第1大的特征向量:
[ 0.99995899 -0.00905645]
Mean vector:
[-3.05269739  4.00039925]

Process finished with exit code 0
```

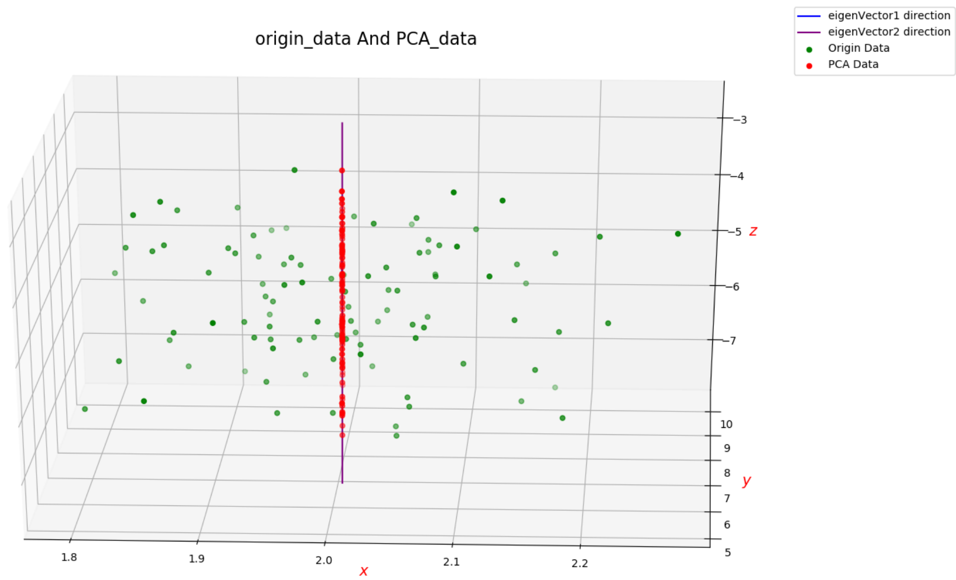
通过上图的输出情况，可见利用PCA求出的特征值最大的特征向量与真实值$[1,0]$极为接近

三维情况:

绿色的点表示**原始数据**，红色的点表示**重构数据**，2条直线分别标明了2个特征值最大的特征向量的方向，也就是PCA**所找到的**。



我们将上图进行旋转，以便于看清重构数据的分布



可见，我们重构的数据就是处于这2个特征向量所$span$的线性空间中，这也验证了我们的PCA算法的原理。

4.2 图像降维测试

- 人脸信息读取

主要利用cv2模块中的方法来进行图像信息读取，再将png图片中的RBG值转换为灰度值，最后将图像数据拉平即可完成。

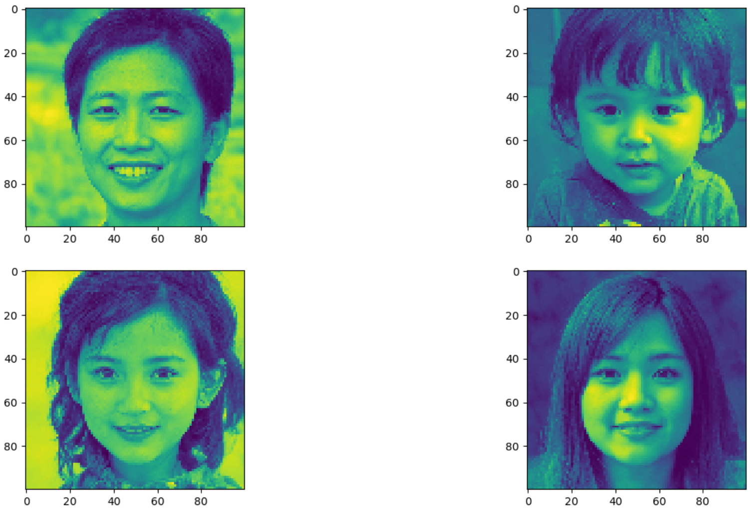
注意：为了便于后续利用PCA时计算协方差更加方便，在此处读取图像信息时就对图像进行压缩处理。

```
    for file in file_list:
        path = os.path.join(file_path, file)
        plt.subplot(2, 2, i)
        with open(path) as f:
            # 读取图像
            img = cv2.imread(path)
            # 压缩图像至size大小
            img = cv2.resize(img, size)
            # RBG图转换为灰度图
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            # 展示灰度值图像
            plt.imshow(img_gray)
            h, w = img_gray.shape
            # 对(h,w)的图像数据拉平
            img_col = img_gray.reshape(h * w)
            data.append(img_col)
        i += 1
```

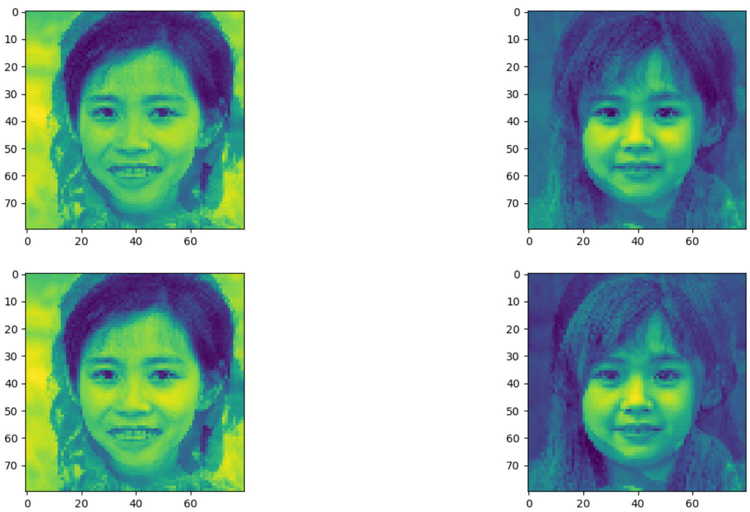
本实验所采用的图片均是1024*1024的png格式

- 降维前后对比

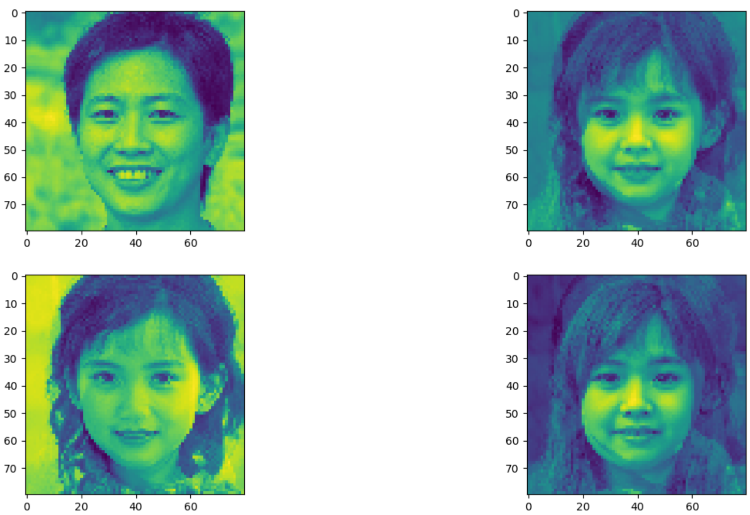
降维前（已经将RBG值转换为灰度值）：



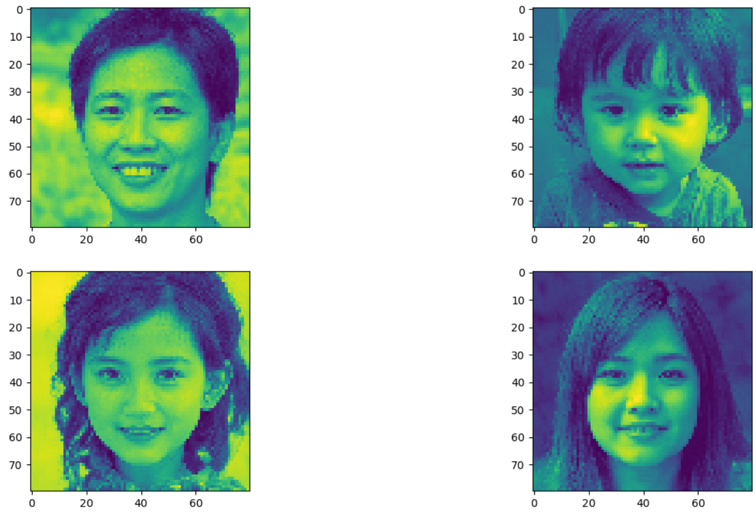
降到1维时，可见各个压缩图片之间很接近：



降到2维时，可见能勉强表示出原图像的特征， 但还是有部分图像之间比较接近：



提高到6维时，可见能够较好地表示出原图像地特征：



通过上述图片之间地比较，我们可以发现随着：**保留的维度越高，压缩后的图像与原图越接近，也就是说保留了更多的信息。**

- 信噪比变化

当然，前面只是从定性角度去分析，下面将**从定量角度去分析图像所保留信息的多少和压缩维度之间的关系。**

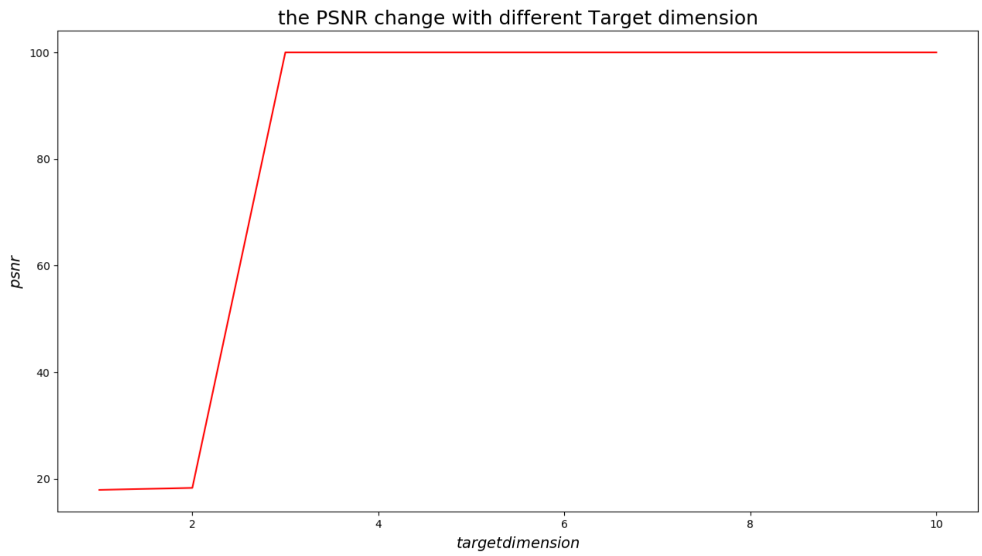
峰值信噪比$PSNR$经常用作图像压缩等领域中信号重建质量的测量方法，它常简单地通过均方差（**MSE**）进行定义。两个$m \times n$单色图像I和K，如果一个为另外一个的噪声近似，那么它们的的均方差定义为：

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \|I(i,j) - K(i,j)\|^2$$

峰值信噪比定义为：

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

下面我将展示信噪比随着所降低到地维度变化



可见，随着维度的增大，信噪比同样在增大，说明所保留的信息在增多。

另外，在维度大于4之后，信噪比变化很小，**说明我们的图像数据的前4个主成分提供了绝大多数的信息**，其余的“次要成分”所能提供的信息十分有限，这也证明了PCA的重要性。

五、实验结论

- PCA算法通过寻找协方差矩阵的特征值最大的K个特征向量，以作为新的一组基，将原数据映射到这一组新的基上来完成数据的降维，也就是说PCA算法可以找到前K个主成分。
- PCA算法提高了样本的采样密度，并且由于较小特征值对应的特征向量往往容易受到噪声的影响，PCA算法舍弃了这部分“次要成分”，一定程度上起到了降噪的效果。
- PCA降低了是在训练集的基础上提取主成分，舍弃“次要成分”； 但是对于测试集而言，被舍弃的也许正好是重要的信息，也就是说PCA可能会加剧过拟合
- PCA可以应用到图像的降维压缩等领域，以提高效率，避免“维度灾难”

六、附录（源代码）

main.py

```
import basicOperation as Bo
import drawImage as dI
import numpy as np

# 自己生成数据的测试
dimension = 3
N = 100
data = Bo.generate_data(dimension, number=N)
c_data, rightEigenVector, mean, recon_data = Bo.PCA(data, dimension - 1)
for i in range(dimension-1):
    print("特征值第"+str(i+1)+"大的特征向量:")
    print(rightEigenVector[:, i])
print("Mean vector:")
print(mean)
dI.originVsPCA(dimension, data, recon_data, mean, rightEigenVector)
```

```
# 图像处理测试
size = (80, 80)
targetDim = 6
data = Bo.read_faces('Image', size=size)
c_data, rightEigenVector, mean, recon_data = Bo.PCA(data, targetDim)
for i in range(targetDim):
    print("特征值第"+str(i+1)+"大的特征向量:")
    print(rightEigenVector[:, i])
print("Mean vector:")
print(mean)
dI.drawFace(recon_data, recon_data.shape[1], size)
# 观测降低到不同维度时的psnr变化
dimRange = np.arange(1, 11, 1)
print(data.shape)
dI.psnrChange(data, dimRange)
```

drawImage.py

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

import basicOperation as Bo


def originVsPCA(dimension, origin_data, recon_data, mean, rightEigenVector):
    """ 将PCA前后的数据进行可视化对比 """
    if dimension == 2:
        fig, ax = plt.subplots()
        ax.scatter(origin_data[0], origin_data[1], facecolor="green",
label="Origin Data")
        ax.scatter(recon_data[0], recon_data[1], facecolor='r', label='PCA
Data')
        x = [mean[0] - 3 * rightEigenVector[0], mean[0] + 3 *
rightEigenVector[0]]
        y = [mean[1] - 3 * rightEigenVector[1], mean[1] + 3 *
rightEigenVector[1]]
        ax.plot(x, y, color='blue', label='eigenVector direction', alpha=0.5)
        ax.set_title('origin_data And PCA_data', fontsize=16)
        ax.set_xlabel('$x$', fontdict={'size': 14, 'color': 'black'})
        ax.set_ylabel('$y$', fontdict={'size': 14, 'color': 'black'})
    elif dimension == 3:
        fig = plt.figure()
        ax = Axes3D(fig)
        ax.scatter(origin_data[0], origin_data[1], origin_data[2],
facecolor='green', label='Origin Data')
        ax.scatter(recon_data[0], recon_data[1], recon_data[2], facecolor='r',
label='PCA Data')
        # 画出2条eigen Vector 方向直线
        x = [mean[0] - 3 * rightEigenVector[0, 0], mean[0] + 3 *
rightEigenVector[0, 0]]
        y = [mean[1] - 3 * rightEigenVector[1, 0], mean[1] + 3 *
rightEigenVector[1, 0]]
```

```
        z = [mean[2] - 3 * rightEigenVector[2, 0], mean[2] + 3 *
rightEigenVector[2, 0]]
        ax.plot(x, y, z, color='blue', label='eigenVector1 direction', alpha=1)
        x2 = [mean[0] - 3 * rightEigenVector[0, 1], mean[0] + 3 *
rightEigenVector[0, 1]]
        y2 = [mean[1] - 3 * rightEigenVector[1, 1], mean[1] + 3 *
rightEigenVector[1, 1]]
        z2 = [mean[2] - 3 * rightEigenVector[2, 1], mean[2] + 3 *
rightEigenVector[2, 1]]
        ax.plot(x2, y2, z2, color='purple', label='eigenVector2 direction',
alpha=1)

        ax.set_title('origin_data And PCA_data', fontsize=16)
        ax.set_zlabel('$z$', fontdict={'size': 14, 'color': 'red'})
        ax.set_ylabel('$y$', fontdict={'size': 14, 'color': 'red'})
        ax.set_xlabel('$x$', fontdict={'size': 14, 'color': 'red'})
    else:
        assert False

    plt.legend()
    plt.show()


def drawFace(recon_data, N, size):
    """
    画出降维重构之后的图像
    """
    plt.figure(figsize=size)
    for i in range(N):
        plt.subplot(2, 2, i + 1)
        plt.imshow(recon_data[:, i].reshape(size))
    plt.show()


def psnrChange(origin_data, dimRange):
    psnrList = []
    for dim in dimRange:
        c_data, rightEigenVector, mean, recon_data = Bo.PCA(origin_data, dim)
        a = Bo.psnr(origin_data[:, 1], recon_data[:, 1])
        psnrList.append(a)
    fig, ax = plt.subplots()
    ax.plot(dimRange, np.array(psnrList), color='r')
    ax.set_title('the PSNR change with different Target dimension', fontsize=18)
    ax.set_xlabel('$target dimension$', fontdict={'size': 14, 'color': 'black'})
    ax.set_ylabel('$psnr$', fontdict={'size': 14, 'color': 'black'})
    plt.show()
```

basicOperation.py

```
from PIL import Image

import cv2
import numpy as np
import math
import os
```

```
import matplotlib.pyplot as plt


def generate_data(data_dimension, number=100):
    """
    自己生成2维或者3维高斯分布的数据集
    :param data_dimension: 数据的维度
    :param number: 样本点的数目
    :return: D * N 的数据矩阵, D是维度，M是样本数目
    """
    if data_dimension is 2:
        mean = [-3, 4]
        # 让某个维度的方差远小于其他维度
        cov = [[1, 0], [0, 0.01]]
    elif data_dimension is 3:
        mean = [2, 8, -5]
        cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
    else:
        assert False

    # 产生shape = (D,M)的数据矩阵
    data = np.random.multivariate_normal(mean, cov, size=number).T
    # if data_dimension is 3:
    #     # 绕z轴旋转数据点
    #     data = rotate(data, 40 * np.pi / 180, 'z')
    return data


def rotate(X, theta=0, axis='x'):
    """
    :param X: 数据矩阵 X.shape = (D, N)
    :param theta: 旋转的弧度
    :param axis: 旋转轴，合法值为'x','y'或'z'
    :return:
    """
    if axis == 'x':
        rotate_matrix = [[1, 0, 0], [0, np.cos(theta), -np.sin(theta)], [0, 
np.sin(theta), np.cos(theta)]]
        return np.dot(rotate_matrix, X)
    elif axis == 'y':
        rotate_matrix = [[np.cos(theta), 0, np.sin(theta)], [0, 1, 0], [-
np.sin(theta), 0, np.cos(theta)]]
        return np.dot(rotate_matrix, X)
    elif axis == 'z':
        rotate_matrix = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), 
np.cos(theta), 0], [0, 0, 1]]
        return np.dot(rotate_matrix, X)
    else:
        assert False


def PCA(data, k):
    """
    将数据data从D维降到k维
    :param data: 数据矩阵(D*N),D表示维度，N表示样本点的个数
    :param k: 把数据降到目标维数
    :return: 零均值化之后的数据矩阵，特征值矩阵，均值矩阵， 重构之后的数据矩阵（仍然 D*N）
    """
```

```python
    dim = data.shape[0]
    mean = np.mean(data, axis=1)
    c_data = np.zeros(data.shape)
    for i in range(dim):
        # 零均值化后得到c_data (D*N)
        c_data[i] = data[i] - mean[i]
    # 求出协方差矩阵
    covMat = np.dot(c_data, c_data.T)
    # 对协方差矩阵covMat(D*D)求特征值和特征向量
    # eigenVectors的每一列对应一个特征向量
    eigenValues, eigenVectors = np.linalg.eig(covMat)
    # 特征值排序
    eigValIndex = np.argsort(eigenValues)
    # 取前k个特征值对应的特征向量 shape = (D*k)
    rightEigenVector = eigenVectors[:, eigValIndex[:-(k + 1):-1]]
    # 一旦降维度超过某个值，特征向量矩阵将出现复向量，对其保留实部
    rightEigenVector = np.real(rightEigenVector)
    # 计算降维后的数据(K*N)
    tmp_data = np.dot(rightEigenVector.T, c_data)
    # 重构之后的数据
    recon_data = np.zeros(data.shape)
    for i in range(dim):
        recon_data[i] = np.dot(rightEigenVector[i], tmp_data) + mean[i]
    return c_data, rightEigenVector, mean, recon_data


def read_faces(file_path, size):
    """
    从图像文件中读取人脸数据
    :param file_path: 文件路径
    :param size: 压缩读取的大小
    :return: 人脸数据矩阵 （D*N）D表示维度，N表示样本点数目
    """
    file_list = os.listdir(file_path)
    data = []
    i = 1
    plt.figure(figsize=size)
    for file in file_list:
        path = os.path.join(file_path, file)
        plt.subplot(2, 2, i)
        with open(path) as f:
            # 读取图像
            img = cv2.imread(path)
            # 压缩图像至size大小
            img = cv2.resize(img, size)
            # RBG图转换为灰度图
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            # 展示灰度值图像
            plt.imshow(img_gray)
            h, w = img_gray.shape
            # 对(h,w)的图像数据拉平
            img_col = img_gray.reshape(h * w)
            data.append(img_col)
        i += 1
    plt.show()
    return np.array(data).T
```

```python
def psnr(img1Data, img2Data):
    mse = np.mean((img1Data / 255. - img2Data / 255.) ** 2)
    if mse < 1.0e-10:
        return 100
    PIXEL_MAX = 1
    # 使用的信噪比公式为20 log_10^(MAX/sqrt(MSE))
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
```