哈尔滨工业大学

计算机科学与技术学院

实验报告

课程名称:机器学习

课程类型:选修

实验题目：PCA

学号：1170300511

姓名：易 亚 玲

一、实验目的

- 掌握PCA的数学原理及代码实现
- 能够用PCA实现一些简单的应用
- 了解PCA的意义和可能存在的问题

二、实验要求及环境

实验要求

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）

（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维数，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）。

实验环境

- Win10, x86-64
- Pycharm 2019.1
- python 3.7.1
- numpy 1.17.4
- Pillow 6.2.1
- matplotlib 3.1.2

三、设计思想

3.1算法原理

3.1.1从内积角度看待向量基变换

以二维向量的表示为例，比如我们常说的向量$\overrightarrow{a}$=(1,3)，实际上是向量$\overrightarrow{a}$在x轴和y轴上的投影长度分别为1和3。而这里的x轴和y轴我们可以看作是一组正交的基向量(1,0)（代表x轴），（0,1）(代表y轴),其中$\overrightarrow{a}$在基向量(1,0)上的投影长度为1，在基向量(0,1)上的投影长度为3。我们通常用向量的内积来表示投影长度，因此向量$\overrightarrow{a}$可以表示为

$$\overrightarrow{a} = 1*(1,0)+3*(0,1)$$

假如我们换一组基向量$(\frac{\sqrt{2}}{2},-\frac{\sqrt{2}}{2}),(\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2})$,此时向量在这组基向量下的投影分别为$(1,3)\cdot(\frac{\sqrt{2}}{2},-\frac{\sqrt{2}}{2})=-\sqrt{2},(1,3)\cdot(\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2})=2\sqrt{2}$,因此在新的基向量下，向量$\overrightarrow{a}=(-\sqrt{2},2\sqrt{2})$,在计算新坐标的时候，我们只需要计算出向量与基向量下的内积即可。合并写可以表示为：

$$\begin{pmatrix} \frac{\sqrt{2}}{2},-\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2} \end{pmatrix}\begin{pmatrix} 1 \\ 3 \end{pmatrix}=\begin{pmatrix} -\sqrt{2} \\ 2\sqrt{2} \end{pmatrix}$$

然后我们引出一般化表示，假如基向量为$p=(p_1,p_2,...,p_n)^T$,原本我们有m个n维向量$a=(a_1,a_2,...,a_m)^T$, a在基向量p的表示下的坐标为

$$a_p=\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}\begin{pmatrix} a_1 & a2 & \dots a_m \end{pmatrix}=\begin{pmatrix} p_1a_1 & p_1a_2 & \dots & p_1a_m \\ p_2a_1 & p_2a_2 & \dots & p_2a_m \\ \vdots & \vdots & \ddots & \vdots \\ p_na_1 & p_na_2 & \dots & p_na_m \end{pmatrix}$$

其实基向量的个数和原本的向量维数不一定需要相等，当不相等时，相当于做升降维变换，a中的向量变换后的维数取决于基向量的个数。

3.1.2 PCA的算法步骤及简单原理概述

对于一个待降维的矩阵X_0

- 首先对矩阵进行去中心化得到矩阵X，即减去均值M, X = $X_0 - M$。
- 然后求出矩阵X的前k大特征值对应的特征向量组成特征矩阵P，作为新的基
- Y = XP投影到新的空间
- 然后对Y进行重构，X =YP^T,$\widetilde{X}$ = X+M

其中X, X_0为mxn的矩阵，m表示样本数，n表示维数；M为1xn的矩阵；P为nxk的矩阵；Y为mxk的矩阵；P^T为kxn的矩阵；$\widetilde{X}$为mxn维矩阵

关于为什么选取前k个最大的特征值对应的特征向量作为基向量，这个是因为较大的特征值能够反映的信息量较大，也就是对应的协方差更大，而且这k个特征值对应的特征向量相互正交，刚好满足基向量的要求。关于这一块的证明，网上有很多资料，故不再赘述。

3.2算法的实现

3.2.0 变量及函数说明

```
SIZE:图片的大小
MAX:图片的采样点为8，最大像素为255
pca(data, kk): pca降维函数
generate_data(m): 生成3数据，并将生成的数据以矩阵形式返回
photo_pca(k): 人脸图片的pca降维
read_photos(): 读取图片，生成样本数据
recreate_photos(mean, vt, data, data_eig, k, SIZE): 将降维后的数据重构成人脸
train_data(m): 生成3维数据，降维为2维，并可视化
```

3.2.1 生成三维数据

```
利用 numpy.random.multivariate_normal(mean,cov,data_size)生成三组数组，其中两组的协方差为100，200，另一组协方差为0.5
```

3.2.2 三维数据将为二维，并进行可视化

调用pca函数，完成降维。提取降维后的两维数据，将数据表格化。根据降维时的两个特征向量，求出投影面的法向量，进而给出一个投影面，然后将降维后的两维数据代入投影面上求出投影点。绘制原数据散点，投影面和数据点在这个面上的投影点。其中，绘制3D图调用Axes3D库。

3.2.3 PCA实现

首先根据输入的数据，求出样本点的个数和样本的维数。然后遍历样本求出样本每一维的均值，再将样本点的每一维减去均值。使用numpy的svd库对零均值化后的样本点求奇异值和特征向量。返回降维后的样本集，均值和特征矩阵的逆（即重构矩阵）。

```
u, s, v = np.linalg.svd(data_aver)  # 其中v是特征向量矩阵（由于矩阵是样本点数x样本维数，如果颠倒一下后u是特征向量矩阵）
```

3.2.4 数据重构

将降维后的矩阵乘以特征向量矩阵的逆，可以得到重构后的矩阵，矩阵的维数和最开始的样本集一样，但是矩阵的数据会有所不同。接下来通过一个例子来说明这个不同。

假设样本集为$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$,特征向量矩阵为$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$,那么降维后的数据集为$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix}$,重构后的矩阵为$\begin{pmatrix} 1 & 2 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 0 \end{pmatrix}$。根据这个例子，数据集中第三维的数据由于特征向量的原因，完全丢失了。当然在实际例子中可能没有这么极端，但是这种数据丢失的现象也是存在的。

重构后的图片与原图片对比，根据下面的公式计算信噪比（PSNR）：

$$MSE = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(data[i][j] - data_eig[i][j])^2$$

$$PNSR = 20\log_{10}\frac{MAX}{\sqrt{MSE}}$$

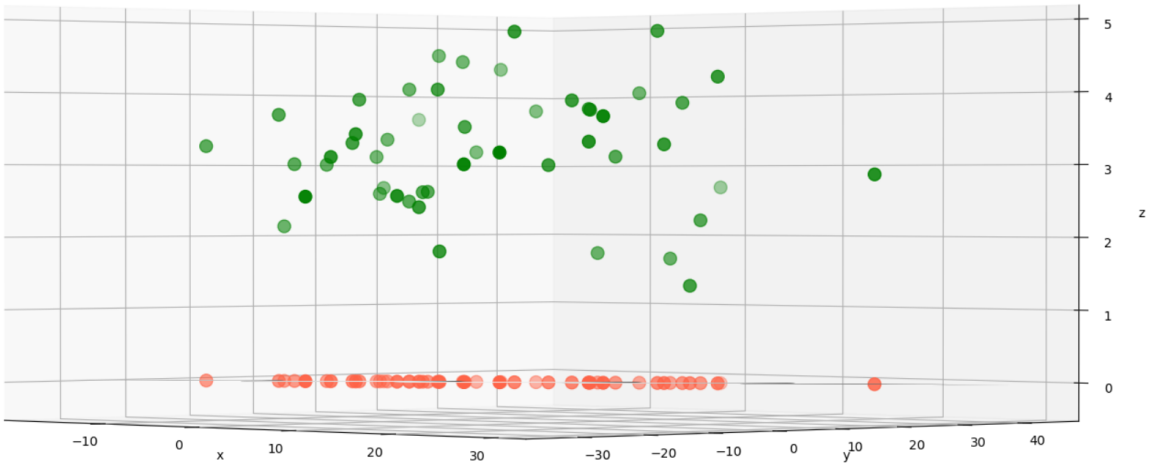
3.2.5 读取jpg格式的图片

使用Pillow内的库Image读取jpg格式的图片，由于jpg格式的图片是采用RGB格式编码，所以有三个通道。在这里我直接将读入的图片转化为了灰度图，不再讨论三个通道的分别处理，关于图片格式的处理我就没在细究，后续会在视听觉信号处理的实验中更多地去了解。然后对灰度图进行矩阵重构，变成一个一维向量，然后赋值给样本的一维。

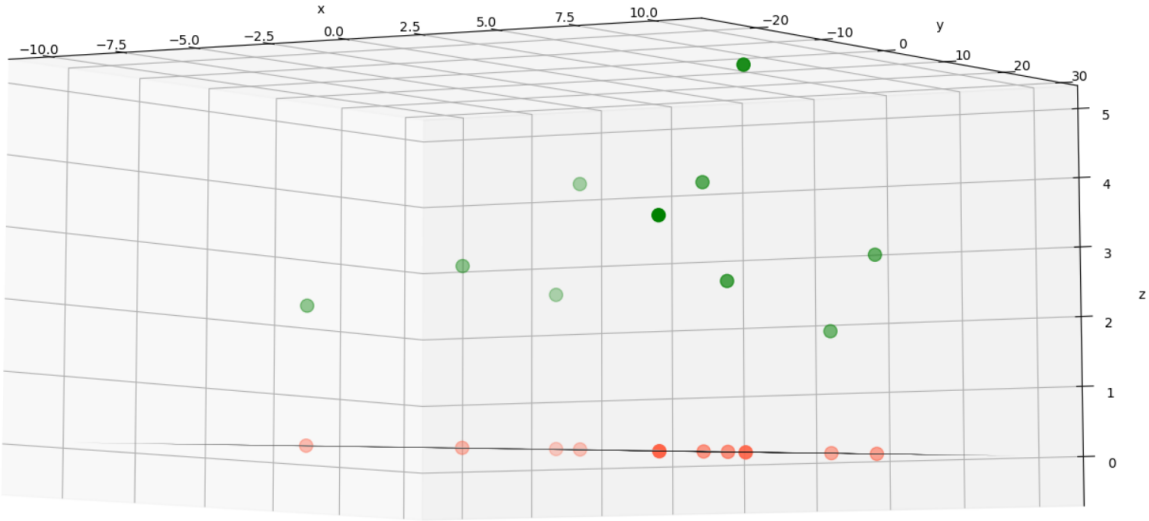
四、实验结果及分析

4.1 生成三维数据并降为2维

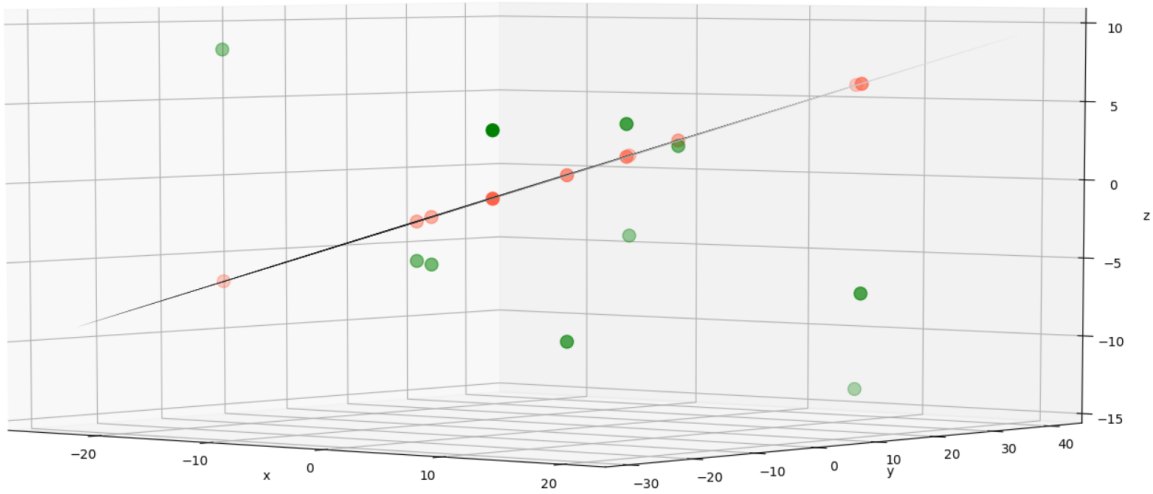
数据量为50，由于z的方差比较小，所以投影面几乎是平行于xoy平面的，从这个图中可以清晰地看到点投影到平面上



将数据量再减少到10，可以将每个点与投影点对应起来

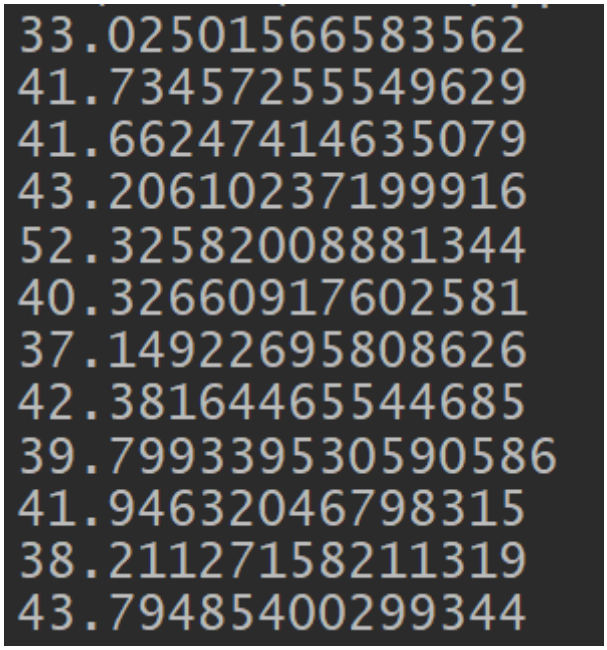


当把三个协方差都调为200时，投影面大约与xoy平面成45度角



4.2人脸数据集的降维与重构

当150x150的图片降维为100x100后再重构，信噪比如下图所示



其中图像的细节也几乎没有损失，对比图像降为100x100和10x10，可以看出100个主成分已经足以表示图片的大部分特征，但是一些图片的细节在一些特征值比较小的成分里，所以即使取的维数较高，依然不能保留细节特征

k=100, PNSR=41.73457255549629

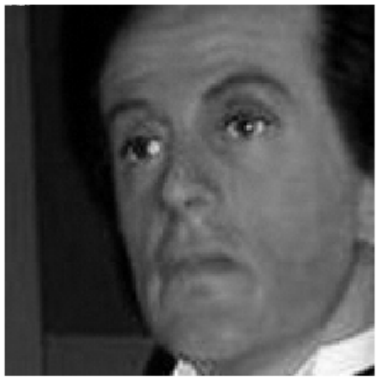
k=10, PNSR=39.19873212932201

k=100, PNSR=41.66247414635079

k=10, PNSR=39.12376774225403

k=100, PNSR=43.20610237199916

k=10, PNSR=40.94958719510552

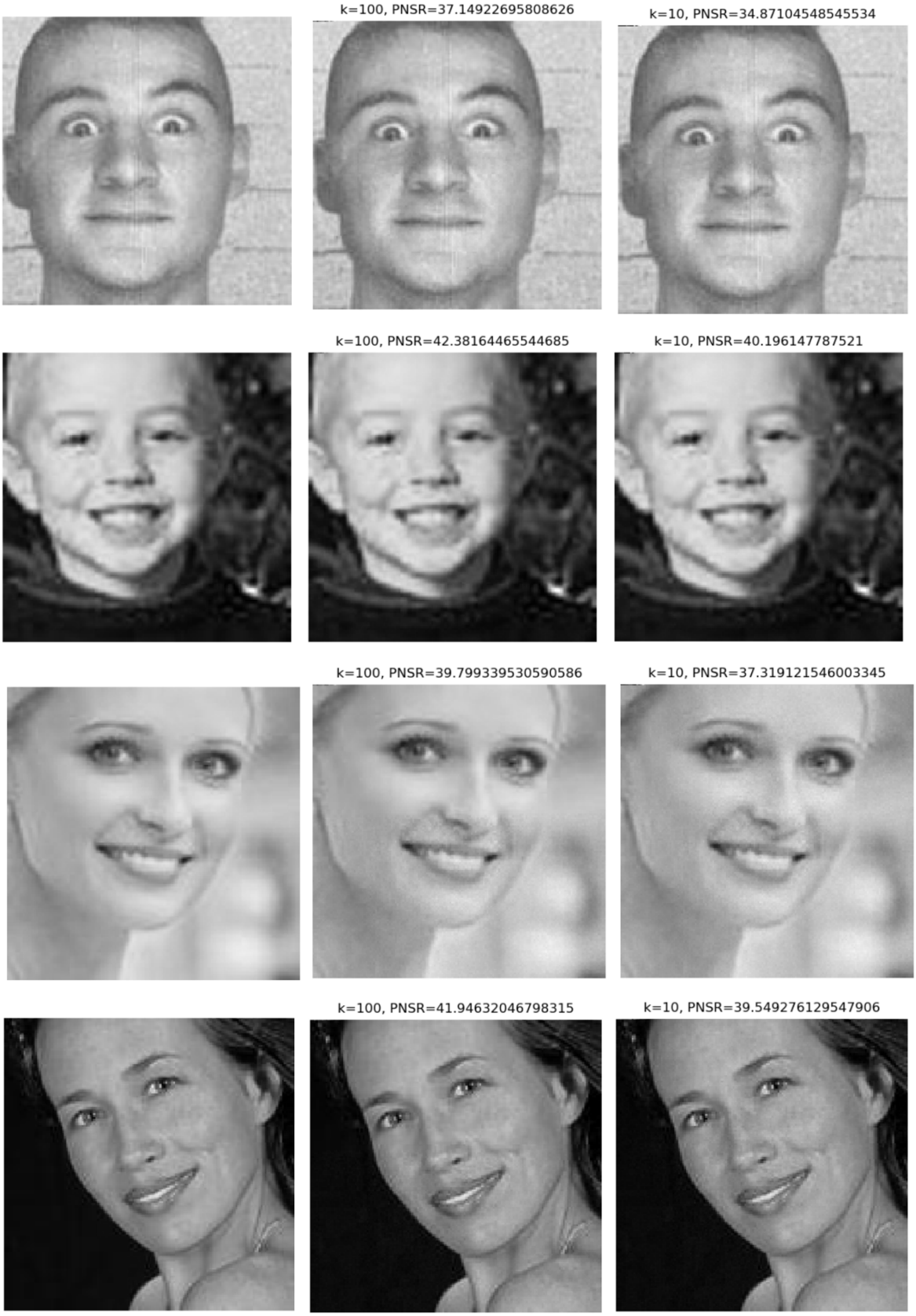


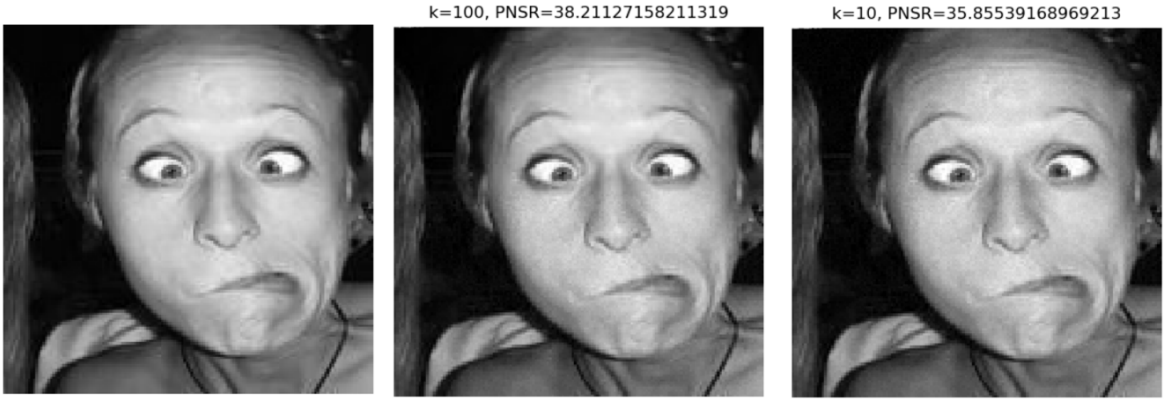
k=100, PNSR=52.32582008881344

k=10, PNSR=49.690004566601324

k=100, PNSR=40.32660917602581

k=10, PNSR=37.71387203505272


k=100, PNSR=37.14922695808626
k=10, PNSR=34.87104548545534
k=100, PNSR=42.38164465544685
k=10, PNSR=40.196147787521
k=100, PNSR=39.799339530590586
k=10, PNSR=37.319121546003345
k=100, PNSR=41.94632046798315
k=10, PNSR=39.549276129547906



五、结论

1. PCA可以用于数据的降维，而且往往不易丢失主要信息。但是PCA是对数据敏感的，如果我们的关键信息是在特征值较小的维度，很有可能会把关键信息弄丢。
2. PCA用于图片的降维可以极大地缓解存储压力，尤其是在如今像素越来越高的情况下。使用PCA降维我们只需要存储三个比较小的矩阵，能够较大地节省存储空间。

六、参考文献

https://zhuanlan.zhihu.com/p/21580949

https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca

附录：源代码（含注释）

```
import numpy as npfrom PIL
import Imageimport os
import mathimport matplotlib.pyplot as plt
from mpl_toolkits.mplot3d
import Axes3D
SIZE = 150  # 图片的大小
MAX = 255  # 图片的采样点为8，最大像素为255
def pca(data, kk):    """pca降维函数"""
    # input:  data  矩阵 待降维数据集(mxn)
    #         kk     int  降维后的维数(kk<n)
    # output: data_rel  矩阵  降维后的矩阵(mxkk)
    #         mean      列表  均值，用于重构
    #         vt        矩阵  降维时右边乘的矩阵，用于重构
    """1.零均值化"""    m = len(data)  # 样本数量
    n = len(data[0, :])  # 样本维数
    mean = []  # 每一维的均值
```

```
    for i in range(n):
        mean.append(sum(data[:, i] / m))
        data_aver = data - mean  # 对数据集的每一维数据进行零均值化

    """2.找到k个主成分特征向量，降维"""
    # svd特征值分解，其中     # u为mxm的矩阵，
    # s^2相当于特征值分解的特征值（已经按从大到小的顺序排列），
    # d是对应的特征向量矩阵
    u, s, v = np.linalg.svd(data_aver)
    data_rel = np.dot(data_aver, v[:, :kk])  # 降维
    vt = v[:, :kk].T  # 重构时需要右乘的矩阵
    return mean, vt, data_rel
def generate_data(m):
    """生成3数据，并将生成的数据以矩阵形式返回"""
    # input:  m     int 生成的数据规模
    # output: data  矩阵 生成的数据，是一个mx3的矩阵
    data = np.zeros((m, 3))  # 初始化数据集
    mean1 = [1]  # 第一维的均值
    mean2 = [2]  # 第二维的均值
    mean3 = [3]  # 第三维的均值
    cov1 = [[200]]  # 第一维的方差
    cov2 = [[200]]  # 第二维的方差
    cov3 = [[200]]  # 第三维的方差，远小于一二维的方差
    data1 = np.random.multivariate_normal(mean1, cov1, m)  # 利用multivariate_normal生成三个高斯数据集
    data2 = np.random.multivariate_normal(mean2, cov2, m)
    data3 = np.random.multivariate_normal(mean3, cov3, m)
    for i in range(m):        # 将每个维的数据对应到data中
        data[i][0] = data1[i]
        data[i][1] = data2[i]
        data[i][2] = data3[i]
        return data
def photo_pca(k):
    """人脸图片的pca降维"""
    # input: k int  k < SIZE
    data = read_photos()    kk = k * k
    mean, vt, data_eig = pca(data, kk)  # 均值，右乘矩阵，降维后的矩阵
    recreate_photos(mean, vt, data, data_eig, SIZE, k)
def read_photos():
    """读取图片，生成样本数据"""
    # output: data 矩阵 每一行代表一张图片
    data = []
    for filename in os.listdir(r"./test"):  # 遍历文件夹中的所有图片
        pic = Image.open(r"./test/" + filename).convert('L')
        pic = pic.resize((SIZE, SIZE))  # 调整图片大小
        pic = np.array(pic).reshape((1, -1))[0].astype('float')  # 将每张图片转化为一维向量，并且将数据类型由uint8转变为double          data.append(pic)    # 缩小后的图片
        for i in range(len(data)):
            plt.imshow(Image.fromarray(data[i].reshape(SIZE, SIZE)))
            plt.axis('off')
            plt.show()
            return np.array(data)
def recreate_photos(mean, vt, data, data_eig, SIZE, k):
    """将降维后的数据重构成人脸"""
    data_eig = np.dot(data_eig, vt) + mean  # 重构后的图片矩阵
    m = len(data)
    n = len(data[0])
    for i in range(m):        # 计算信噪比
        mse = 0
        for j in range(n):
            mse += (data[i][j] - data_eig[i][j]) ** 2 / n  # 计算第i张图的信噪比
```

```
            pnsr = 20 * math.log(MAX / math.sqrt(mse), 10)
            print(pnsr)
            pic = data_eig[i].reshape((SIZE, SIZE))  # 提取第i张图片的向量
            plt.imshow(Image.fromarray(pic))  # 画出降维后的图
            plt.title("k=" + str(k) + ", PNSR=" + str(pnsr))
            plt.axis('off')
            plt.show()
def train_data(m):
    """ 生成3维数据，降维为2维，并可视化"""

    data = generate_data(m)  # 生成3维数据
    *_, vt, data_eig = pca(data, 2)  # 将3维数据将为2维数据
    # 绘制图像
    fig = plt.figure()
    ax = Axes3D(fig)
    # 数据录入
    x = np.array(np.array(data[:, 0]))
    y = np.array(np.array(data[:, 1]))
    z = np.array(np.array(data[:, 2]))
    # 绘制降维前的散点图
    ax.scatter(x, y, z, marker='o', c='green', s=100)
    # 绘制降维的曲面
    f = np.cross(vt[0], vt[1])  # 根据基向量求投影面的法向量
    x_table, y_table = np.meshgrid(x, y)  # 数据表格化
    z_surf = -(f[0] * x_table + f[1] * y_table) / f[2]  # 根据法向量求z
    ax.plot_surface(x_table, y_table, z_surf, cmap='gray', shade=False, zorder=1)  # 绘
制曲面
    # 绘制降维后的散点
    z_after = []
    for i in range(len(data)):
        z_after.append(-(f[0] * data[i][0] + f[1] * data[i][1]) / f[2])
    ax.scatter(x, y, z_after, marker='o', c='tomato', s=100, zorder=2)
    # 设置三个坐标轴信息
    ax.set_xlabel('x', color='black')  # 设置坐标轴
    ax.set_ylabel('y', color='black')
    ax.set_zlabel('z', color='black')
    plt.draw()
    plt.show()
if __name__ == '__main__':
    # train_data(10)  # 生成100个维数为3的数据，旋转并降维
    photo_pca(4)  # 读取图片并降维
```