哈尔滨工业大学计算学部

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：1190201215
姓名： 冯开来

一、实验目的

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）

二、实验要求及实验环境

2.1 实验要求

(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度，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）。

2.2 实验环境

Windows10; python3.9;PyCharm 2021.2.2

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 主要算法

核心算法就是 PCA（主成分分析）降维算法。其作用是从高维数据中提取出最重要的几维特征，根据这些特征在进行重构，重构后的图像有别于原图但又可以清晰的辨认出原图。

PCA 有两种比较常见的方法，一个是最大方差，另一个是最小重构代价。两者其实本质是相同的。最大方差形式通俗描述就是将高维空间向某一超平面投影，投影得到的点的方差最大，从而最能代表原数据。如一个椭球体，投影为正面椭圆的方差最大，而不是侧面投影的的圆的方差最大。

因此可以推导如下：

一组数据$X = \{x_1, x_2, \dots, x_n\}$, X 是 D×N 的矩阵

μ_i是一个投影后的基向量，当然有一组基向量，我们拿其中一个分析

容易得到数据均值：

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

数据方差：

$$S = \frac{1}{N}\sum_{i=1}^{N} (x_i - \bar{x})^T (x_i - \bar{x})$$

投影均值：

$$\bar{\hat{x}} = \frac{1}{N}\sum_{i=1}^{N} \mu^T x_i$$

投影后的方差为：

$$\hat{S} = \frac{1}{N}\sum_{i=1}^{N}(\mu^T x_i - \mu^T \bar{x})^T(\mu^T x_i - \mu^T \bar{x})$$

$$= \mu^T \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^T(x_i - \bar{x})\,\mu$$

$$= \mu^T S \mu$$

注意让方差 S 最大，我们可以用拉格朗日乘子法：

$$L(w, \lambda) = \mu^T S \mu + \lambda(1 - \mu^T \mu)$$

然后对其求导，令极值等于 0 解得：

$$S\mu_1 = \lambda\mu_1$$

故求得最大化方差，即求最大的特征值。要将 D 维的数据降维到 P 维度，只需计算前 P 个最大的特征值，将其对应的特征向量组合成特征向量矩阵(DxP)，然后用右乘数据矩阵的转置即可实现降维压缩。

3.2 算法实现

见附录源代码。这里附上 PCA 算法的代码：

```
'''
PCA 降维
data 数量*维度 nxd
data_mean 样本均值
eigValInd 排序好的特征值 dxk
redEigVects 降维后的特征值
'''
def PCA(data, k = 2):
    rows, cols = data.shape
    data_mean = np.sum(data, 0) / rows
    data_center = data - data_mean # 中心化 nxd
    covMat = np.dot(data_center.T, data_center) # dxd
    eigVals, eigVects = np.linalg.eig(covMat) # 求协方差矩阵特征值和特征向量，特征向量按列读取dxd
    eigValInd = np.argsort(eigVals) # 特征值排序
    redEigVects = eigVects[: ,eigValInd[: -(k+1): -1]] # 序列逆向排列，取前k个特征向量dxk，
    redEigVects = np.real(redEigVects) # 如果出现复向量，对其保留实部
    data_tmp = np.dot(data_center, redEigVects) # 降维后的数据 nxk
    data_recon = np.dot(data_tmp, redEigVects.T) + data_mean # 重构后的数据 nxd
    return redEigVects, data_recon
```

首先将数据中心化处理（减去平均值），然后通过生成的中心化的数据生成协方差矩阵。求协方差矩阵的特征值和特征向量，对特征值排序取前 k 个最大的特征值对应的特征向量，即我们要找的降维后的特征向量，当然如果特征向量存在虚部的话，直接保留实部就行。

接下来就是对数据重构，直接将中心化数据乘特征向量矩阵得到投影数据值，为得到坐标，再右乘一个特征向量矩阵，最后假设均值即完成数据重构。

正常返回的应该是 DxN 的矩阵，但我这里返回的是 NxD 的矩阵，其本质没有区别。

四、实验结果与分析

4.1 人工生成数据的降维

4.1.1 生成 Swiss Roll 模型

流形学习中的经常会遇到 Swiss Roll 数据集。该数据集是把二维空间中的点映射到三维空间中，以便进行数据降维测试。该数据集正投影为漩涡状，侧投影为矩形。根据 Swiss Roll 的“厚度”变化，其特征投影面将从“薄瑞士卷”的正投影，变为“厚瑞士卷”的侧投影。

本次实验有 1000 个数据点，每个点有三个属性，分别是 x、y、z 轴的坐标。并得到 Swiss Roll 后让其绕着 x 轴需旋转一定角度（默认 45°）。具体的生成代码如下：

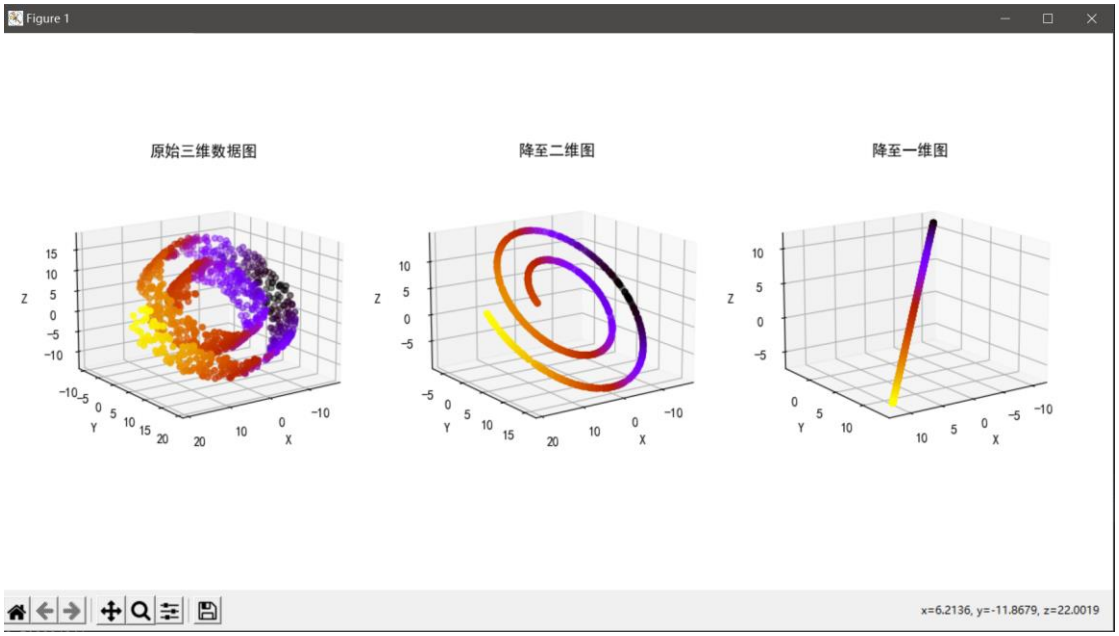
```
'''
生成瑞士卷数据 nxd
n_sample 数据点数量
noise 数据店的噪声
y_scale y方向厚度
'''
def make_swiss_roll(n_sample = 1000, noise = 0.0, y_scale = 10, degree = 45):
    t = 2 * np.pi * (1 + 2 * np.random.rand(1, n_sample)) # 定义变量
    x = t * np.cos(t) # 定义x方向数据
    y = y_scale * np.random.rand(1, n_sample)
    z = t * np.sin(t)
    data = np.concatenate((x, y, z)) # 将三维数据压缩成一个向量，即数据有三个属性
    data += noise * np.random.rand(3, n_sample) # 加入噪声
    data = rotate(data, np.pi * degree / 180, 'x') # 绕x轴旋转degree，默认45
    # show_3D(data.T)
    return data.T # 得到nxd
```

其中绕轴旋转的函数 rotate()：

```
def rotate(data, theta = 0, axis = 'x'):
    if axis == 'x':
        rotate = [[1, 0, 0], [0, np.cos(theta), -np.sin(theta)], [0, np.sin(theta), np.cos(theta)]]
        return np.dot(rotate, data)
    elif axis == 'y':
        rotate = [[np.cos(theta), 0, np.sin(theta)], [0, 1, 0], [-np.sin(theta), 0, np.cos(theta)]]
        return np.dot(rotate, data)
    elif axis == 'z':
        rotate = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0, 1]]
        return np.dot(rotate, data)
    else:
        print('ERROR asix')
        return X
```

4.1.1 PCA 降维（3D-2D-1D）

原本数据集是三维，我们先将数据降成二维，最后再降成一维。根据常识我们可以知道，二维是平面，而一维是直线，所以带入函数我们可以得到下面的展示结果：

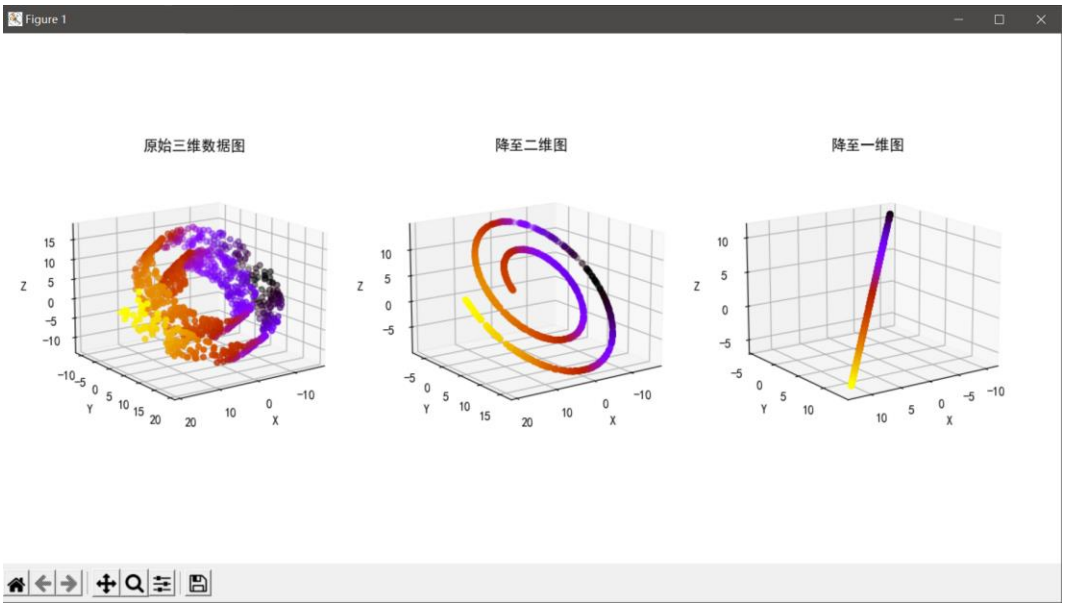


并且得到降维到二维和一维的特征向量：

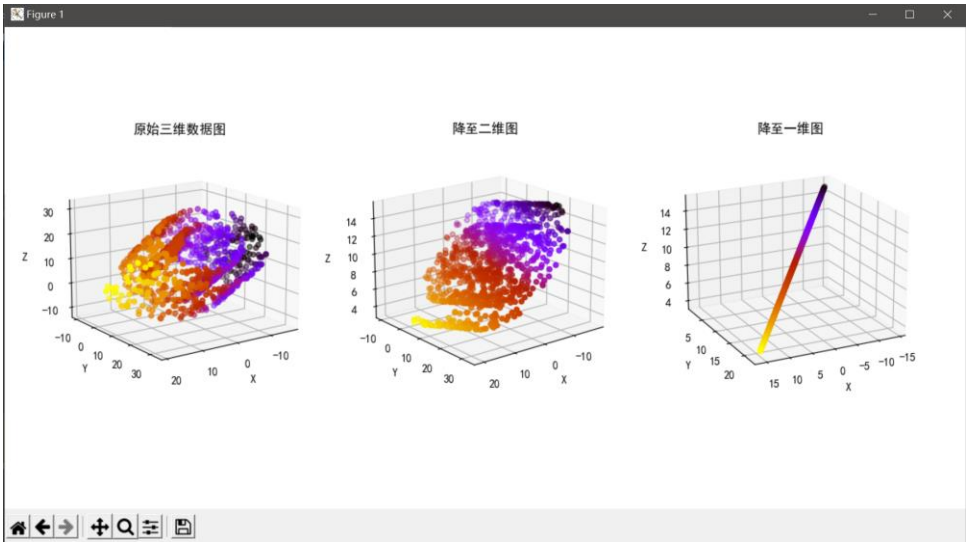
```
**********************************************************************************************
二维特征向量
[[ 0.76017053  0.64972337]
 [ 0.45889965 -0.53630751]
 [-0.45994769  0.53873351]]
一维特征向量
[[ 0.76017053]
 [ 0.45889965]
 [-0.45994769]]
**********************************************************************************************
```

4.1.3 演示结果与分析

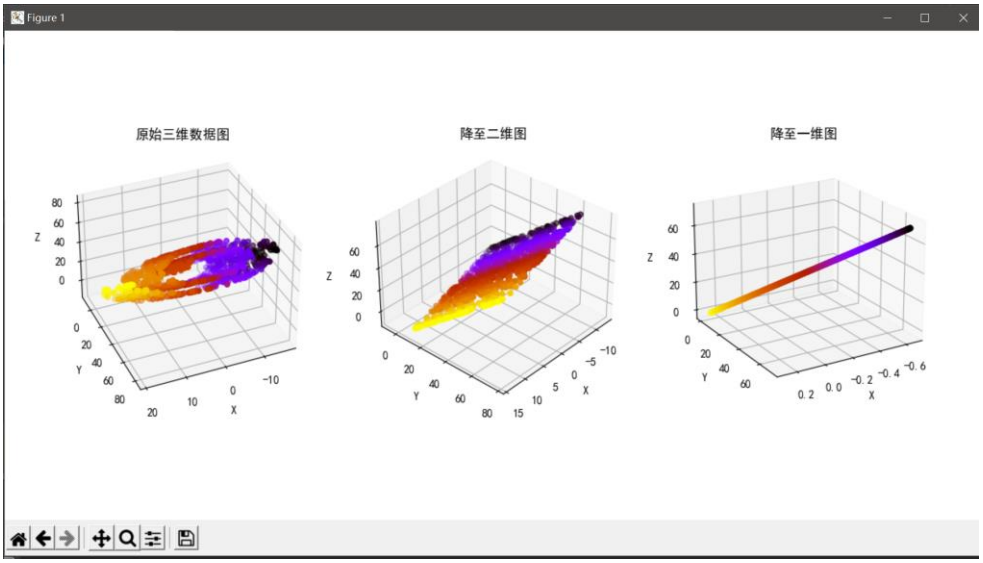
1） 样本 1000；y 方向厚度 10；绕 x 轴旋转 45°；无噪声



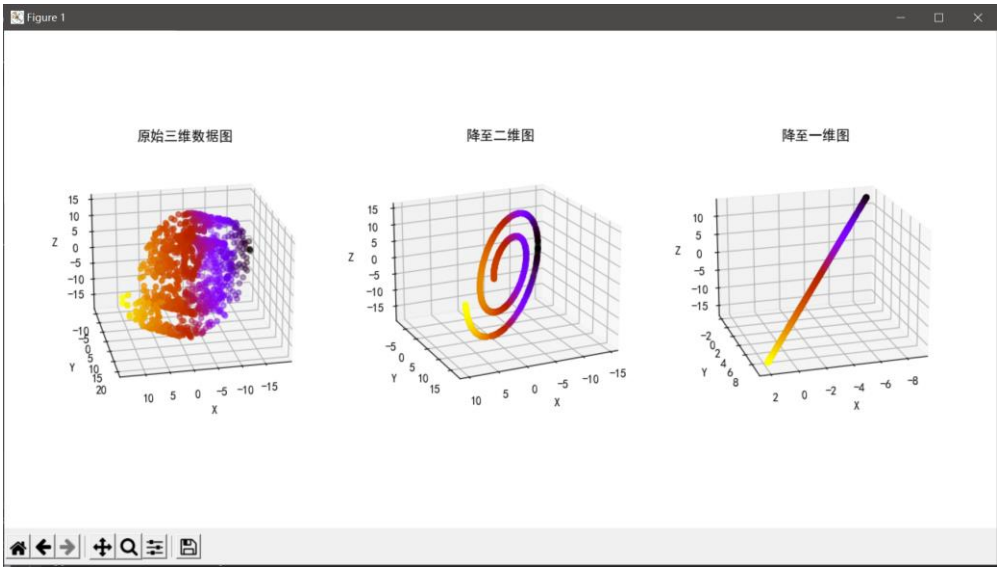
2） 样本 1000；y 方向厚度 30；绕 x 轴旋转 45°；无噪声



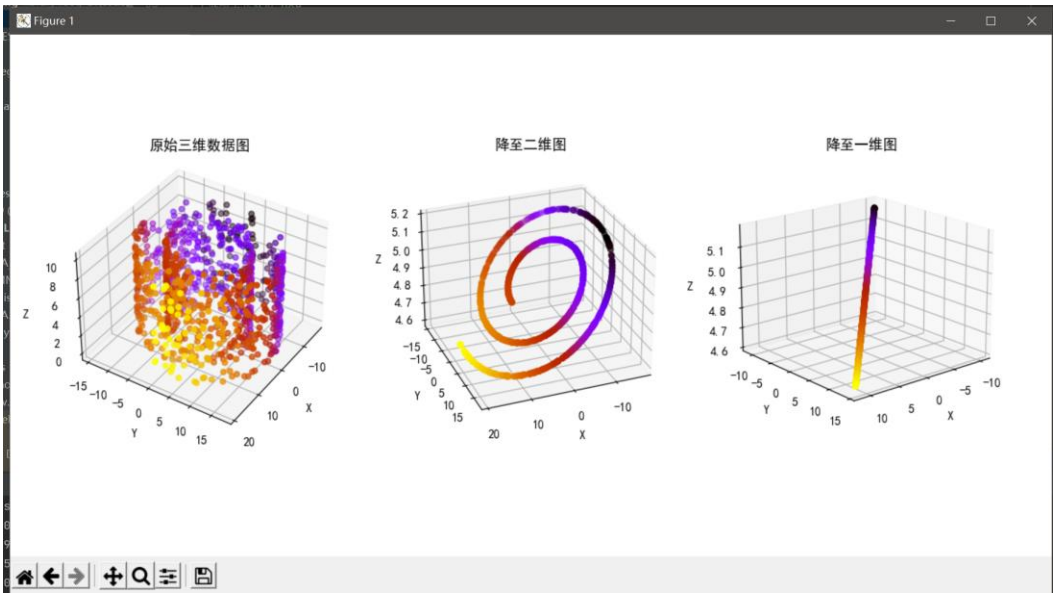
3） 样本 1000；y 方向厚度 100；绕 x 轴旋转 45°；无噪声



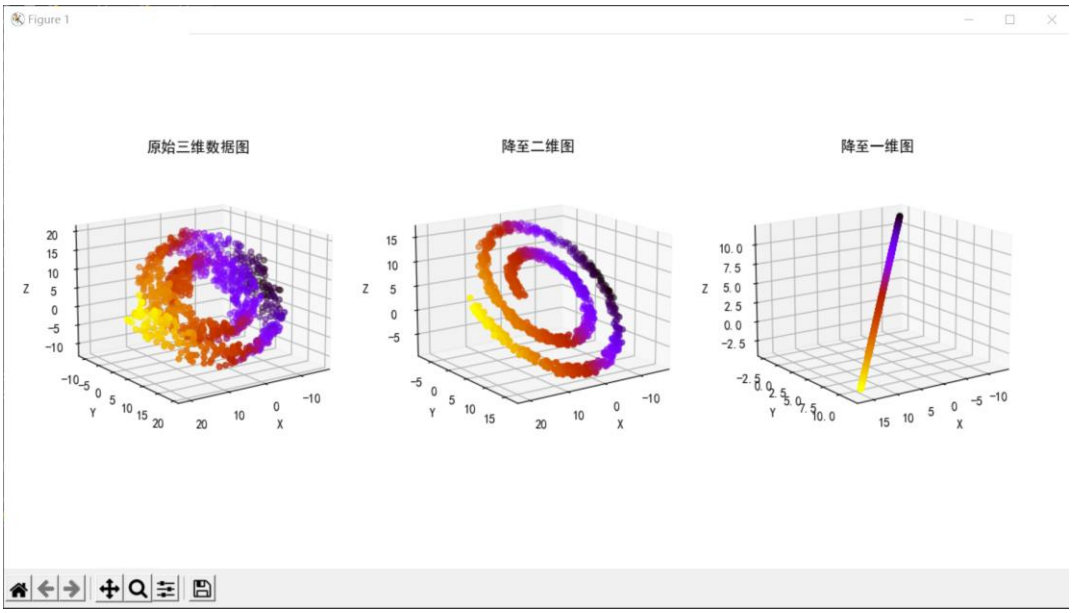
4） 样本 1000；y 方向厚度 10；绕 z 轴旋转 45°；无噪声



5） 样本 1000；y 方向厚度 10；绕 x 轴旋转 90°；无噪声



6） 样本 1000；y 方向厚度 10；绕 x 轴旋转 45°；噪声为 2



结果分析：

1. PCA 成功地将 3 维立体图降维至 2 维平面图再至 1 维直线图
2. 在改变旋转轴和旋转角度可以发现降维仍是成功的
3. 对比实验 1、2、3，可以发现在逐渐将 y 方向厚度增加的时候，二维图的平面开始不太一样，从原来漩涡状逐渐因为厚度增加而变成矩形状，这也可以看到主成分 y 分量在其中的影响，而随后的降至一维的时候，线的方向也因此改变，也逐渐平行于 y 分量（因为 y 方向主成分占比越来越大）
4. 对比实验 1、4、5、6，可以发现无论怎么旋转角度或是绕什么轴旋转，最后得到的一维主成分都是一样或是相近的，说明确实找到了最重要的主成分。

4.2 人脸数据降维

4.2.1 读取文件图片

本次实验图片来源一开始都是英雄联盟头像图片，后来再换人脸做了一遍。读取图像的方法运用了 OpenCV 库的函数，并且为了减少计算时间，我们统一把图片压缩成 40x40 的大小，然后我们将每一张图片的尺寸压缩成一个向量，就是一张图片有 1600 个属性。具体代码如下：

```
'''
生成数据，人脸图像
图片尺寸 300x300
7张图片
返回 7x90000
'''
def read_faces(file_path, size):
    file_list = os.listdir(file_path)
    data = []
    i = 1
    for file in file_list:
        path = os.path.join(file_path, file)
        with open(path) as f:
            img = cv2.imread(path, 0) # 参数0，默认灰度图打开
            img = cv2.resize(img, size) # 重新将图片压缩成size大小
            img_col = img.reshape(img.shape[0] * img.shape[1])
            data.append(img_col) # 得到数据集
    return np.array(data)
```

4.2.2 PCA 降维（三次降维）

本次实验我认为是通过几张图片找到主成分完成重构。自己在床上想了想，应该有很多用处，比如我可以通过多张一个人的不同角度和表情的图片重构出这个人重要的面部信息，或是通过多个人正面照片得到平均脸。本次实验原来的维度是 1600，首先降维至 5，再降维至 3，最后降至 1 维。

在实际计算求解特征向量的时候，特征向量会出现虚部的问题，只要保留实部就能得到较好的结果。

不过有一说一，维度越低，重构的数据越诡异…而且最后，所有的图片似乎都会融合在一起，每张一维图片都有别的图片的影子…具体代码如下：

```
'''
主函数2：
人脸数据 nx1600
PCA降维，从1600维到5维到3维到1维
输出2维和1维时候特征向量
'''
size = (40, 40)
data = read_faces('temp', size)
# data = read_faces('PCA_FACES', size)
n_sample = data.shape[0]
n_dimension = data.shape[1]
redEigVects_1, data_recon_1 = PCA(data, 5)
redEigVects_2, data_recon_2 = PCA(data, 3)
redEigVects_3, data_recon_3 = PCA(data, 1)
print('三次降维后的特征向量：')
print(redEigVects_1)
print(redEigVects_2)
print(redEigVects_3)
```

4.2.3 计算信噪比

信噪比计算公式：

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：

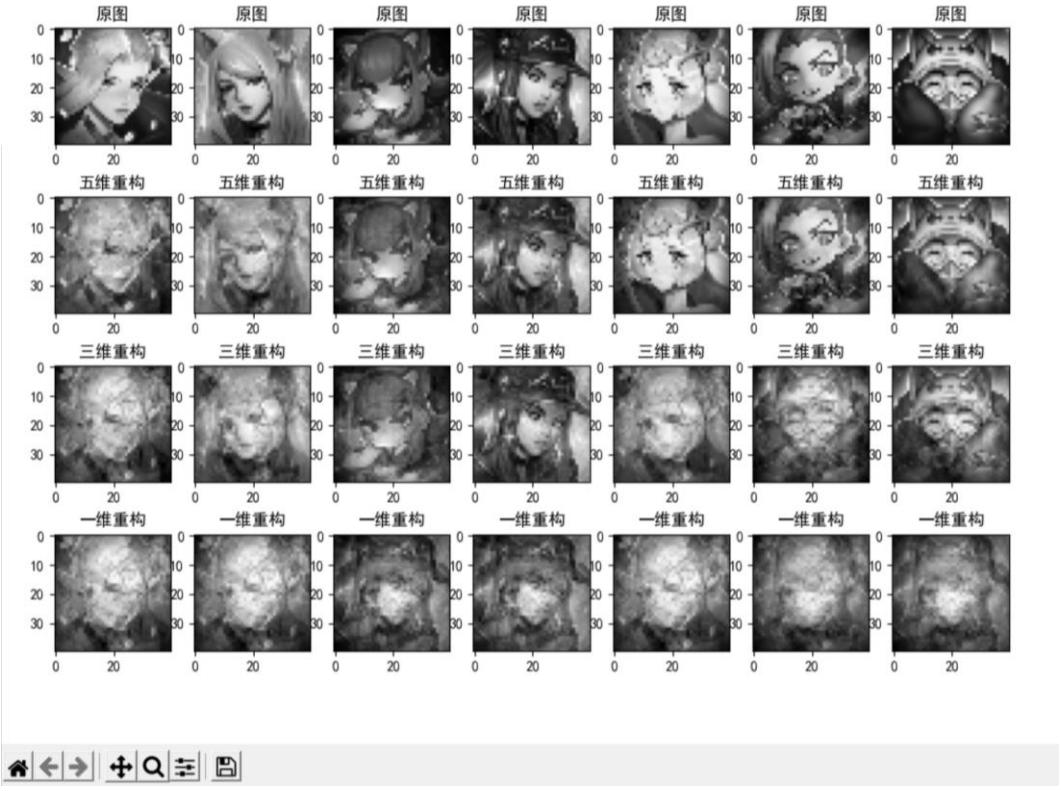
$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

具体代码如下：

```
'''
计算信噪比
'''
def psnr(img1, img2):
    mse = np.mean((img1/255. - img2/255.) ** 2)
    if mse < 1e-10:
        return 100
    PIXEL_MAX = 1
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
```

4.2.4 演示结果与分析

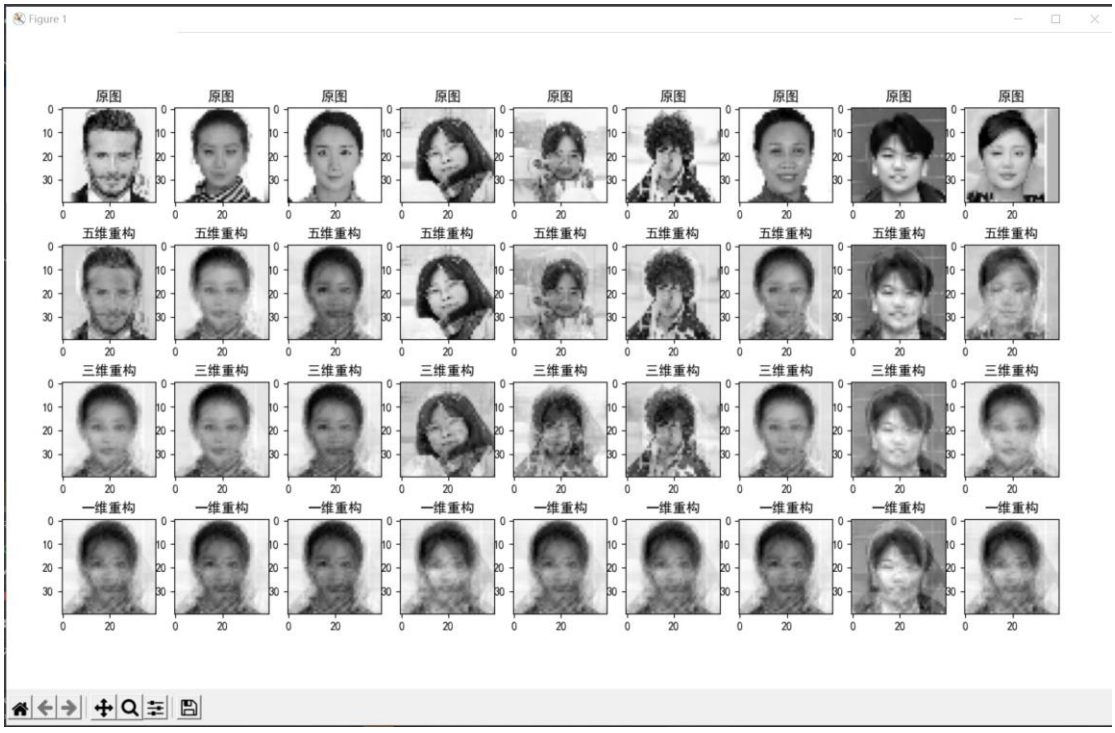
样本卡通头像；1600D-5D-3D-1D



```
--------特征向量如下--------
降至5维时：
[[-1.49937229e-02 -2.51041559e-02 -1.37134050e-02  7.92333886e-03
   2.59145888e-02]
 [-1.72668762e-02 -2.19318117e-02 -9.72669678e-03  1.10950023e-02
   2.85249894e-02]
 [-1.75604579e-02 -2.21336821e-02 -6.93063887e-03  1.42024358e-02
   3.03930632e-02]
 ...
 [ 1.30241714e-02 -3.63341276e-02 -2.18056953e-05  1.86382043e-02
   7.85046260e-03]
 [ 7.48910078e-03 -3.37630991e-02  2.12359966e-03  1.65293974e-02
   1.66152615e-02]
 [ 9.31794851e-03 -3.94694020e-02  4.66202775e-04  1.51409539e-02
   2.02294158e-02]]
降至3维时：
[[-1.49937229e-02 -2.51041559e-02 -1.37134050e-02]
 [-1.72668762e-02 -2.19318117e-02 -9.72669678e-03]
 [-1.75604579e-02 -2.21336821e-02 -6.93063887e-03]
 ...
 [ 1.30241714e-02 -3.63341276e-02 -2.18056953e-05]
 [ 7.48910078e-03 -3.37630991e-02  2.12359966e-03]
 [ 9.31794851e-03 -3.94694020e-02  4.66202775e-04]]
降至1维时：
[[-0.01499372]
 [-0.01726688]
 [-0.01756046]
 ...
 [ 0.01302417]
 [ 0.0074891 ]
 [ 0.00931795]]
```

```
---------信噪比如下---------
降至5维时：
图 0 的信噪比为： 19.393598074805798
图 1 的信噪比为： 20.017542019630575
图 2 的信噪比为： 27.595632855295804
图 3 的信噪比为： 22.34362213174869
图 4 的信噪比为： 31.967731777649767
图 5 的信噪比为： 28.29651625296833
图 6 的信噪比为： 25.13940956868632
降至3维时：
图 0 的信噪比为： 16.74775556836258
图 1 的信噪比为： 17.067311617829894
图 2 的信噪比为： 24.986740907758588
图 3 的信噪比为： 22.295621446053964
图 4 的信噪比为： 16.21121594468454
图 5 的信噪比为： 15.835685660222516
图 6 的信噪比为： 20.01499210027213
降至1维时：
图 0 的信噪比为： 16.517722491696286
图 1 的信噪比为： 14.849455087033549
图 2 的信噪比为： 14.91204411308792
图 3 的信噪比为： 15.538604599898003
图 4 的信噪比为： 15.522577074672073
图 5 的信噪比为： 14.142335570173897
图 6 的信噪比为： 14.166783159466211
```

样本人脸；1600D-5D-3D-1D



```
--------特征向量如下--------
降至5维时：
[[ 0.0282555   0.00273726  0.00041138  0.0050173   0.01146216]
 [ 0.03389631  0.00795047  0.00142006  0.00815287 -0.00144538]
 [ 0.03356954  0.00666298 -0.00100419  0.00728027  0.00198169]
 ...
 [ 0.0380599  -0.00557503  0.02080448 -0.01252881 -0.01962354]
 [ 0.02115017  0.01459823 -0.00478009 -0.00156307 -0.01025573]
 [ 0.03501474  0.01952235 -0.01358057  0.00032695 -0.00496782]]
降至3维时：
[[ 0.0282555   0.00273726  0.00041138]
 [ 0.03389631  0.00795047  0.00142006]
 [ 0.03356954  0.00666298 -0.00100419]
 ...
 [ 0.0380599  -0.00557503  0.02080448]
 [ 0.02115017  0.01459823 -0.00478009]
 [ 0.03501474  0.01952235 -0.01358057]]
降至1维时：
[[0.0282555 ]
 [0.03389631]
 [0.03356954]
 ...
 [0.0380599 ]
 [0.02115017]
 [0.03501474]]
```

```
---------信噪比如下---------
降至5维时：
图 0 的信噪比为： 21.344213753573392
图 1 的信噪比为： 18.325187122070922
图 2 的信噪比为： 20.236479947320845
图 3 的信噪比为： 35.711669350887774
图 4 的信噪比为： 26.597679660469062
图 5 的信噪比为： 27.5886845878263
图 6 的信噪比为： 19.660667401342696
图 7 的信噪比为： 27.571273752615596
图 8 的信噪比为： 19.06711173487232
降至3维时：
图 0 的信噪比为： 14.68247669241996
图 1 的信噪比为： 18.21274057574056
图 2 的信噪比为： 19.900906287213513
图 3 的信噪比为： 22.58906384860972
图 4 的信噪比为： 14.876127781384108
图 5 的信噪比为： 19.182828598884623
图 6 的信噪比为： 18.747347225514975
图 7 的信噪比为： 23.569642540174648
图 8 的信噪比为： 15.439204667238817
降至1维时：
图 0 的信噪比为： 14.152849754169438
图 1 的信噪比为： 16.112800409758627
图 2 的信噪比为： 19.238704063231097
图 3 的信噪比为： 11.805016347837276
图 4 的信噪比为： 12.710505801884677
图 5 的信噪比为： 12.183456548606708
图 6 的信噪比为： 16.33979795822405
图 7 的信噪比为： 20.017442467560723
图 8 的信噪比为： 14.516170487116234
```

结果分析：

1. 维度降低，重构图像越偏离原图，信噪比也越低（也偶尔有突然增高的）
2. 从 1600 维到 5 维，图像仍保存较好还原度，当然应该也是图片太少的原因
3. 图像出现原图偏离的维度与样本数量成正比，即样本数量越多，在同一维度下越容易产生重构的偏离
4. 个人认为重构在某种程度上是将几张图片融为一体，所以 PCA 降维可以用来通过多张图片学习出一张平均图
5. 在降维至 3 维到 1 维的时候，信噪比已经没有说明区别了，但是可以看到 3 维图像还是比 1 维图像更有辨认度
6. 该部分可以看到有明显的学习过程，如果样本越多，最后重构出的图像越相似
7. psnr 是有效衡量图像压缩信号重构的指标。

五、结论

1. PCA 模型通过降维可以适当抑制过拟合
2. PCA 训练舍弃了 d-k 个信息，这一定会导致低维空间与高维空间不同，但是这种方式提高了样本的采样密度，同时说不定还能降噪。
3. PCA 在对一组图像进行降维的时候有明显的学习过程，图片越多，它会将更多信息融合在一起，也就更容易造成与原图的偏离。但是我们将一组相近的图片进行 PCA 降维，那可以找到这一组图片的共同点，比如识别出什么是人脸，什么是汽车等。
4. PCA 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

六、参考文献

1. （CSDN）图像峰值信噪比计算方法
   https://blog.csdn.net/xrinosvip/article/details/88569111
2. （知乎）瑞士卷（Swiss Roll）模型生成方式
   https://www.zhihu.com/question/391236862

七、附录：源代码（带注释）

```
'''
(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
远小于其它维度，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）。
'''
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import math
import cv2
import os
from mpl_toolkits.mplot3d import Axes3D

'''
PCA 降维
data 数量*维度 nxd
data_mean 样本均值
eigValInd 排序好的特征值 dxk
redEigVects 降维后的特征值
'''
def PCA(data, k = 2):
    rows, cols = data.shape
    data_mean = np.sum(data, 0) / rows
    data_center = data - data_mean # 中心化 nxd
    covMat = np.dot(data_center.T, data_center) # dxd
    eigVals, eigVects = np.linalg.eig(covMat) # 求协方差矩阵特征值和特征向量，特征向量按列读取 dxd
    eigValInd = np.argsort(eigVals) # 特征值排序
    redEigVects = eigVects[: ,eigValInd[: -(k+1): -1]] # 序列逆向排列，取前 k 个特征向量 dxk，
    redEigVects = np.real(redEigVects) # 如果出现复向量，对其保留实部
    data_tmp = np.dot(data_center, redEigVects) # 降维后的数据 nxk
```

```
    data_recon = np.dot(data_tmp, redEigVects.T) + data_mean # 重构后的数据 nxd
    return redEigVects, data_recon

'''
生成瑞士卷数据 nxd
n_sample 数据点数量
noise 数据店的噪声
y_scale y 方向厚度
'''
def make_swiss_roll(n_sample = 1000, noise = 0.0, y_scale = 10, degree = 45):
    t = 2 * np.pi * (1 + 2 * np.random.rand(1, n_sample)) # 定义变量
    x = t * np.cos(t) # 定义 x 方向数据
    y = y_scale * np.random.rand(1, n_sample)
    z = t * np.sin(t)
    data = np.concatenate((x, y, z)) # 将三维数据压缩成一个向量，即数据有三个属性
    data += noise * np.random.rand(3, n_sample) # 加入噪声
    data = rotate(data, np.pi * degree / 180, 'x') # 绕 x 轴旋转 degree，默认 45
    # show_3D(data.T)
    return data.T # 得到 nxd

def rotate(data, theta = 0, axis = 'x'):
    if axis == 'x':
        rotate = [[1, 0, 0], [0, np.cos(theta), -np.sin(theta)], [0, np.sin(theta), np.cos(theta)]]
        return np.dot(rotate, data)
    elif axis == 'y':
        rotate = [[np.cos(theta), 0, np.sin(theta)], [0, 1, 0], [-np.sin(theta), 0, np.cos(theta)]]
        return np.dot(rotate, data)
    elif axis == 'z':
        rotate = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0, 1]]
        return np.dot(rotate, data)
    else:
        print('ERROR asix')
        return X

'''
生成二维高斯数据
n_sample 样本数量
data 维度 * 数目的矩阵
'''
def make_2D_gaussian(n_sample = 100):
    mean = [-3, 4]
    cov = [[1, 0], [0, 0.01]]
    data = np.random.multivariate_normal(mean, cov, n_sample).T
    return data
```

```
'''
生成数据，人脸图像
图片尺寸 300x300
7 张图片
返回 7x90000
'''
def read_faces(file_path, size):
    file_list = os.listdir(file_path)
    data = []
    i = 1
    for file in file_list:
        path = os.path.join(file_path, file)
        with open(path) as f:
            img = cv2.imread(path, 0) # 参数 0，默认灰度图打开
            img = cv2.resize(img, size) # 重新将图片压缩成 size 大小
            img_col = img.reshape(img.shape[0] * img.shape[1])
            data.append(img_col) # 得到数据集
    return np.array(data)

'''
计算信噪比
'''
def psnr(img1, img2):
    mse = np.mean((img1/255. - img2/255.) ** 2)
    if mse < 1e-10:
        return 100
    PIXEL_MAX = 1
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

'''
展示图像
'''
def show_3D(x):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.view_init(elev = 20, azim = 80)
    ax.scatter(x[:, 0], x[:, 1], x[:, 2], c=x[:, 0], cmap=plt.cm.gnuplot)
    ax.legend(loc = 'best')
    plt.show()
def show_2D(x):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.view_init(elev=20, azim=80)
```

```
    ax.scatter(x[:, 0], x[:, 1], x[:, 2], c=x[:, 0], cmap=plt.cm.gnuplot)
    ax.legend(loc = 'best')
    plt.show()
def show_1D(x):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.view_init(elev=20, azim=80)
    ax.scatter(x[:,0], x[:,1], x[:,2], c = x[:,0], cmap = plt.cm.gnuplot)
    ax.legend(loc = 'best')
    plt.show()

def show_3D_to_1D(data_1, data_2, data_3):

    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False

    fig = plt.figure(figsize=(12, 6), facecolor='w')
    # cm = mpl.colors.ListedColormap(['#FFC2CC', '#C2FFCC', '#CCC2FF'])
    # cm2 = mpl.colors.ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    plt.subplots_adjust(left=0.05, right=0.95, bottom=0.01, top=0.99)



    ax1 = fig.add_subplot(131, projection='3d')
    ax1.scatter(data_1[:, 0], data_1[:, 1], data_1[:, 2], c=data_1[:, 0], cmap=plt.cm.gnuplot)
    ax1.view_init(elev=15, azim=55)
    plt.title('原始三维数据图')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')
    plt.grid(True)

    ax2 = fig.add_subplot(132, projection='3d')
    ax2.scatter(data_2[:, 0], data_2[:, 1], data_2[:, 2], c=data_2[:, 0], cmap=plt.cm.gnuplot)
    ax2.view_init(elev=15, azim=55)
    plt.title('降至二维图')
    ax2.set_xlabel('X')
    ax2.set_ylabel('Y')
    ax2.set_zlabel('Z')
    plt.grid(True)

    ax3 = fig.add_subplot(133, projection='3d')
    ax3.scatter(data_3[:, 0], data_3[:, 1], data_3[:, 2], c=data_3[:, 0], cmap=plt.cm.gnuplot)
    ax3.view_init(elev=15, azim=55)
    plt.title('降至一维图')
```

```
        ax3.set_xlabel('X')
        ax3.set_ylabel('Y')
        ax3.set_zlabel('Z')
        plt.grid(True)

        plt.show()

'''
主函数 1：
人工生成数据 nxd
PCA 降维，从 3 维到 2 维到 1 维
输出 2 维和 1 维时候特征向量
'''
data_1 = make_swiss_roll()
redEigVects_2, data_2 = PCA(data_1)
redEigVects_3, data_3 = PCA(data_2, 1)
print('**********************************************************************
**********')
print('二维特征向量')
print(redEigVects_2)
print('一维特征向量')
print(redEigVects_3)
show_3D_to_1D(data_1, data_2, data_3)
print('**********************************************************************
**********')

'''
主函数 2：
人脸数据 nx1600
PCA 降维，从 1600 维到 5 维到 3 维到 1 维
输出 2 维和 1 维时候特征向量
'''
size = (40, 40)
# data = read_faces('temp', size)
data = read_faces('PCA_FACES', size)
n_sample = data.shape[0]
n_dimension = data.shape[1]
redEigVects_1, data_recon_1 = PCA(data, 5)
redEigVects_2, data_recon_2 = PCA(data, 3)
redEigVects_3, data_recon_3 = PCA(data, 1)
print('--------特征向量如下--------')
print("降至 5 维时：")
print(redEigVects_1)
print("降至 3 维时：")
```

```
print(redEigVects_2)
print("降至 1 维时：")
print(redEigVects_3)
print("---------信噪比如下---------")
print("降至 5 维时：")
for i in range(n_sample):
    print('图', i, '的信噪比为：', psnr(data[i], data_recon_1[i]))
print("降至 3 维时：")
for i in range(n_sample):
    print('图', i, '的信噪比为：', psnr(data[i], data_recon_2[i]))
print("降至 1 维时：")
for i in range(n_sample):
    print('图', i, '的信噪比为：', psnr(data[i], data_recon_3[i]))



'''
展示人脸图像
3 次降维图
'''
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
fig = plt.figure(figsize = (n_sample*1.5, 4*2))
plt.subplots_adjust(left=0.05, right=0.95, bottom=0.1, top=0.9)
for i in range(n_sample):
    fig.add_subplot(4, n_sample, i+1)
    plt.title('原图')
    plt.imshow(data[i].reshape(size), cmap='gray')
    fig.add_subplot(4, n_sample, i+1+n_sample)
    plt.title('五维重构')
    plt.imshow(data_recon_1[i].reshape(size), cmap='gray')
    fig.add_subplot(4, n_sample, i+1+n_sample*2)
    plt.title('三维重构')
    plt.imshow(data_recon_2[i].reshape(size), cmap='gray')
    fig.add_subplot(4, n_sample, i+1+n_sample*3)
    plt.title('一维重构')
    plt.imshow(data_recon_3[i].reshape(size), cmap='gray')
plt.show()
```