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 模型实验

学号：1190200610
姓名： 张景阳

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

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）。

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 基变换的矩阵表示

基坐标的变换可以让我们被变换的向量与目标基做内积，得到的新坐标就是在目标基下的坐标。

对于(1,1)，(2,2)，(3,3)，来说。坐标变化可以写成：

$$\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 2/\sqrt{2} & 4/\sqrt{2} & 6/\sqrt{2} \\ 0 & 0 & 0 \end{pmatrix}$$

我们也可以写成更加通用的表示形式:

$$\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \cdots & a_M \end{pmatrix} = \begin{pmatrix} p_1 a_1 & p_1 a_2 & \cdots & p_1 a_M \\ p_2 a_1 & p_2 a_2 & \cdots & p_2 a_M \\ \vdots & \vdots & \ddots & \vdots \\ p_R a_1 & p_R a_2 & \cdots & p_R a_M \end{pmatrix}$$

其中，Pi 是一个行向量，表示第 i 个基，aj 是一个列向量，表示第 j 个原始数据记录。上面的分析给矩阵相乘找到了一种物理解释：两个矩阵相乘的意义是将右边矩阵的没一列向量 ai，变换到左边矩阵中以每一行行向量为基所表示的空间中去。也就是说一个矩阵可以表示一种线性变换。

2. 最大可分性

上面我们讨论了选择不同的基可以对同样的一组数据给出不同的表示，如果基的数量小于向量本身的维数，则可以达到降维的效果。

但是我们还没回答一个最关键的问题：如果选择基才是最优的。或者说，如果我们有一组 N 维向量，现在要将其降到 K 维（K<N），那么我们应该如何选择 K 个基才能最大程度保留原有的信息?

一种直观的看法是：希望降维之后的投影值尽可能分散，因为如果重叠就会有样本消失。我们也可以从熵的角度进行理解，熵越大所含的信息越多。

3. 协方差矩阵

针对我们的优化目标，我们将从数学的角度给出。

我们对于两维的变量，其中有 m 个样本，那么我们可以按行组成矩阵 X：

$$X = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \\ b_1 & b_2 & \cdots & b_m \end{pmatrix}$$

然后：

$$\frac{1}{m}XX^{\mathsf{T}} = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix} = \begin{pmatrix} Cov(a,a) & Cov(a,b) \\ Cov(b,a) & Cov(b,b) \end{pmatrix}$$

我们可以看到这个矩阵对角线上的分别是两个变量的方差，而其他元素是 a 和 b 的协方差，二者被统一到了一个矩阵里。

推广到一般情况：设我们有 m 个 n 维数据记录，将其排列成矩阵 Xn,m，设 C=1/mXXT，则 C 是一个对称矩阵，其对角线分别对应各个变量的方差，而第 i 行 j 列和 j 行 i 列元素相同，表示 i 和 j 两个变量的协方差。

4. 矩阵对角化

根据我们的优化条件，我们需要将除对角线外的其它元素化为 0，并且在对角线上将元素按大小从上到下排列（变量方差尽可能大)，这样我们就达到了优化目的。这样说可能还不是很明晰，我们进一步看下原矩阵与基变换后矩阵协方差矩阵的关系。

设原始数据矩阵×对应的协方差矩阵为 C，而 P 是一组基按行组成的矩阵，设 Y=PX，则 Y 为 X 对 P 做基变换后的数据。设 Y 的协方差矩阵为 D，我们推导一下 D 与 C 的关系:

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

这样我们就清楚了，我们要找的 P 是能让原始协方差矩阵对角化的 P，换句话说，优化目标变成了一个矩阵 P，满足 PCPT 是一个对角矩阵，并且对角元素按从大到小依次排列，那么 P 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 维并满足上述优化条件。

至此，我们离 PCA 还有一步之遥，我们还需要完成对角化。

5. SVD（奇异值分解）

如果我们需要降维的数据不是方阵，我们是不是就不能对矩阵进行特征分解了么？这时候就要用到 SVD 了方法了。

SVD 也是对矩阵进行分解，但是和特征分解不同，SVD 并不要求分解的矩阵为方阵。SVD 的分解方法如下：

1） 首先计算出 ATA 与 AAT

$$\mathbf{A}^{\mathbf{T}}\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\mathbf{A}\mathbf{A}^{\mathbf{T}} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

2） 进而求出我们得到的两个方阵的特征值和特征向量

$$\lambda_1 = 3; v_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}; \lambda_2 = 1; v_2 = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$$

$$\lambda_1 = 3; u_1 = \begin{pmatrix} 1/\sqrt{6} \\ 2/\sqrt{6} \\ 1/\sqrt{6} \end{pmatrix}; \lambda_2 = 1; u_2 = \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{pmatrix}; \lambda_3 = 0; u_3 = \begin{pmatrix} 1/\sqrt{3} \\ -1/\sqrt{3} \\ 1/\sqrt{3} \end{pmatrix}$$

3） 利用 Av = sigma u 求奇异值：

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \sigma_1 \begin{pmatrix} 1/\sqrt{6} \\ 2/\sqrt{6} \\ 1/\sqrt{6} \end{pmatrix} \Rightarrow \sigma_1 = \sqrt{3}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \sigma_2 \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{pmatrix} \Rightarrow \sigma_2 = 1$$

注意：左奇异矩阵可以用于行数的压缩，右奇异矩阵可以用于列数即特征维度的压缩也就是 PCA 降维。

6. 总结

总结 PCA 的算法步骤：设有 m 条 n 维数据

1） 将原始数据按列组成 n 行 m 列矩阵 X

2） 将 X 的每一行进行零均值化，即减去这一行的均值

3） 求出均方差矩阵

4) 求出均方差矩阵的特征值及对应的特征向量

5) 按特征向量按对应特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 P

6) Y=PX 即为降维到 k 维后的数据

四、实验结果与分析

1. 代码分析：

生成二维和三维的数据：

```
def create_data_3(num=100):
    mean = [3, -5, 12]
    cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
    data = np.random.multivariate_normal(mean, cov, size=num).T
    return rotate_z(data, 40 * np.pi / 180)


def create_data_2(num=100):
    mean = [3, -5]
    cov = [[1, 0], [0, 0.01]]
    data = np.random.multivariate_normal(mean, cov, size=num).T
    return data
```

PCA 主体算法：

```
def pca(data, k):
    # 将数据data从D维降到k维
    d = data.shape[0]
    mean = np.mean(data, axis=1)
    zero_data = np.zeros(data.shape)
    # 零均值化
    for i in range(d):
        zero_data[i] = data[i] - mean[i]
    covM = np.dot(zero_data, zero_data.T)
    values, vectors = np.linalg.eig(covM)
    values = np.real(values)
    vectors = np.real(vectors)
    index = np.argsort(values)
    rightVectors = vectors[:, index[:-(k + 1): -1]]
    tmp_data = np.dot(rightVectors.T, zero_data)
    pca_data = np.zeros(data.shape)
    for i in range(d):
        pca_data[i] = np.dot(rightVectors[i], tmp_data) + mean[i]

    return zero_data, mean, rightVectors, pca_data
```

二维降维的可视化：

```
def visualize_2(data, pca_data, mean, vectors):
    draw = plt.subplot()
    draw.scatter(data[0], data[1], facecolor='green', label='Origin Data')
    draw.scatter(pca_data[0], pca_data[1], facecolor='r', label='PCA Data')

    x = [mean[0] - 3 * vectors[0], mean[0] + 3 * vectors[0]]
    y = [mean[1] - 3 * vectors[1], mean[1] + 3 * vectors[1]]
    draw.plot(x, y, color='black', label='eigenVector1 direction', alpha=0.5)

    draw.set_title('data vs pca_data', fontsize=16)
    draw.set_xlabel('$x$', fontdict={'size': 14, 'color': 'red'})
    draw.set_ylabel('$y$', fontdict={'size': 14, 'color': 'red'})

    plt.legend()
    plt.show()
```

三维降维的可视化：

```
def visualize_3(data, pca_data, mean, vectors):
    draw = Axes3D(plt.figure())
    draw.scatter(data[0], data[1], data[2], facecolor='green', label='Origin Data')
    draw.scatter(pca_data[0], pca_data[1], pca_data[2], facecolor='r', label='PCA Data')
    # 画出vector直线
    x = [mean[0] - 3 * vectors[0, 0], mean[0] + 3 * vectors[0, 0]]
    y = [mean[1] - 3 * vectors[1, 0], mean[1] + 3 * vectors[1, 0]]
    z = [mean[2] - 3 * vectors[2, 0], mean[2] + 3 * vectors[2, 0]]
    draw.plot(x, y, z, color='blue', label='eigenVector1 direction', alpha=1)

    x2 = [mean[0] - 3 * vectors[0, 1], mean[0] + 3 * vectors[0, 1]]
    y2 = [mean[1] - 3 * vectors[1, 1], mean[1] + 3 * vectors[1, 1]]
    z2 = [mean[2] - 3 * vectors[2, 1], mean[2] + 3 * vectors[2, 1]]
    draw.plot(x2, y2, z2, color='purple', label='eigenVector2 direction', alpha=1)

    draw.set_title('data vs pca_data', fontsize=16)
    draw.set_xlabel('$x$', fontdict={'size': 14, 'color': 'red'})
    draw.set_ylabel('$y$', fontdict={'size': 14, 'color': 'red'})
    draw.set_zlabel('$z$', fontdict={'size': 14, 'color': 'red'})

    plt.legend()
    plt.show()
```

读取照片，进行 pca 处理

```
def read_pca_face(file_path):
    face_list = read_face(file_path)
    for face in face_list:
        pca_list = []
        psnr_list = []
        for k in k_list:
            zero_data, mean, vector, pca_data = pca(face, int(k))
            pca_list.append(pca_data)
            psnr_list.append(psnr(face, pca_data))
        show_faces(face, pca_list, psnr_list)
```

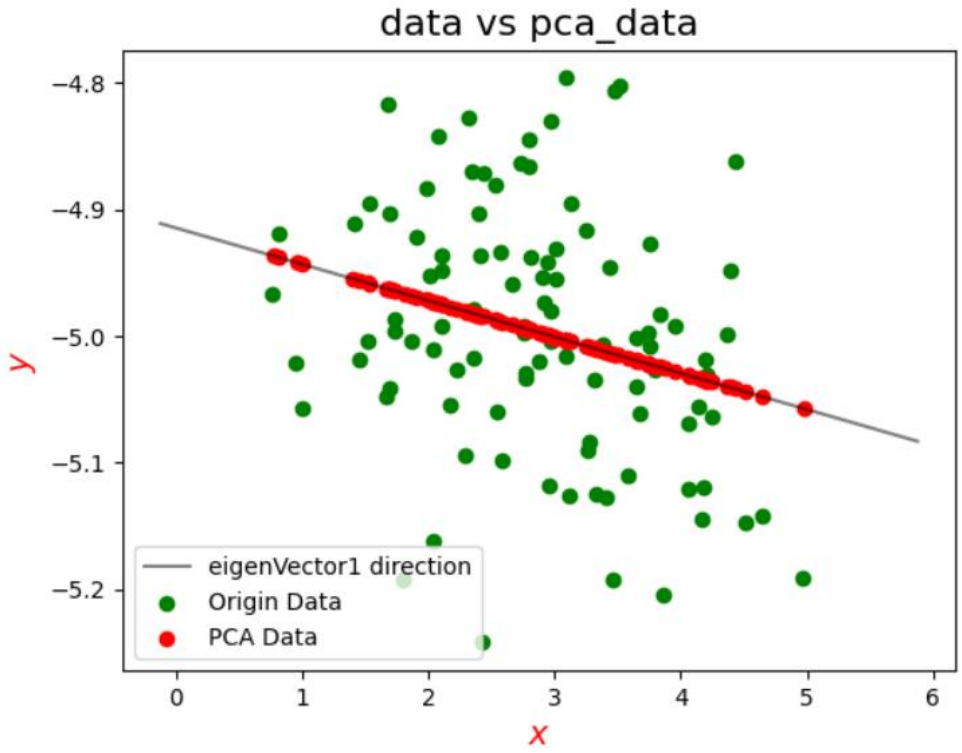
信噪比计算

```
def psnr(source, target):
    rmse = np.sqrt(np.mean((source - target) ** 2))
    return 20 * np.log10(255.0 / rmse)
```

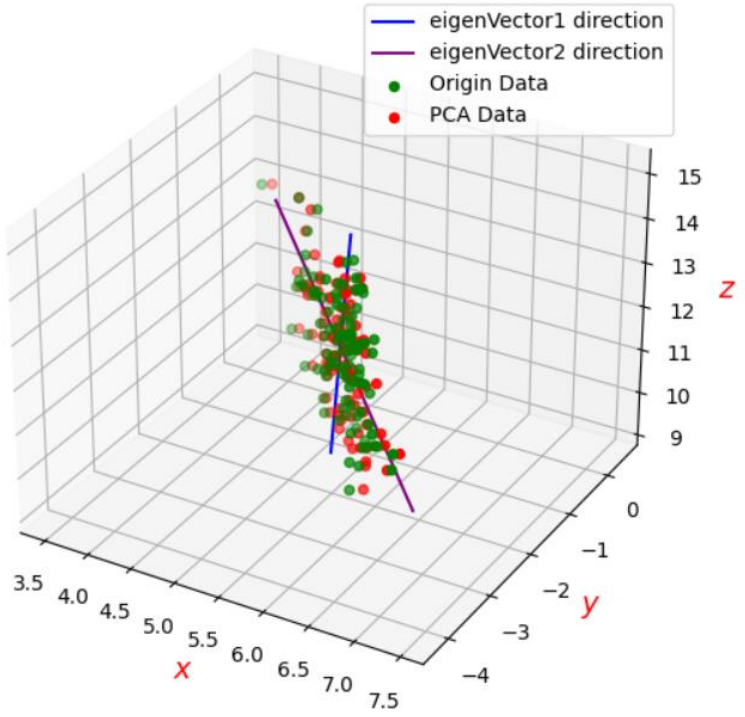
坐标轴按 z 轴旋转

```
def rotate_z(X, theta=0):
    matrix = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0, 1]]
    return np.dot(matrix, X)
```

2. 二维降为一维

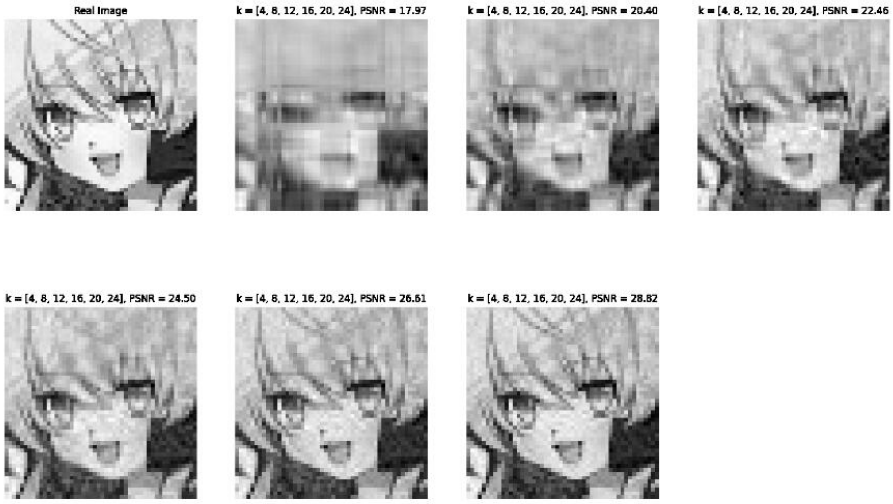


3. 三维降为二维



4. 人脸识别


Real Image
k = [4, 8, 12, 16, 20, 24], PSNR = 18.92
k = [4, 8, 12, 16, 20, 24], PSNR = 21.50
k = [4, 8, 12, 16, 20, 24], PSNR = 23.73
k = [4, 8, 12, 16, 20, 24], PSNR = 25.86
k = [4, 8, 12, 16, 20, 24], PSNR = 28.02
k = [4, 8, 12, 16, 20, 24], PSNR = 30.60
Real Image
k = [4, 8, 12, 16, 20, 24], PSNR = 19.17
k = [4, 8, 12, 16, 20, 24], PSNR = 22.08
k = [4, 8, 12, 16, 20, 24], PSNR = 24.50
k = [4, 8, 12, 16, 20, 24], PSNR = 26.49
k = [4, 8, 12, 16, 20, 24], PSNR = 28.56
k = [4, 8, 12, 16, 20, 24], PSNR = 30.83
Real Image
k = [4, 8, 12, 16, 20, 24], PSNR = 18.76
k = [4, 8, 12, 16, 20, 24], PSNR = 21.90
k = [4, 8, 12, 16, 20, 24], PSNR = 24.22
k = [4, 8, 12, 16, 20, 24], PSNR = 26.32
k = [4, 8, 12, 16, 20, 24], PSNR = 28.69
k = [4, 8, 12, 16, 20, 24], PSNR = 31.32



五、结论

1. 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。
2. 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。
3. 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
4. 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、参考文献

《机器学习》 周志华
《统计学习方法》 李航

七、附录：源代码（带注释）

```
import os

import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

from PIL import Image

k_list = [4, 8, 12, 16, 20, 24]
```

```
def create_data_3(num=100):
    mean = [3, -5, 12]
    cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
    data = np.random.multivariate_normal(mean, cov, size=num).T
    return rotate_z(data, 40 * np.pi / 180)


def create_data_2(num=100):
    mean = [3, -5]
    cov = [[1, 0], [0, 0.01]]
    data = np.random.multivariate_normal(mean, cov, size=num).T
    return data


def rotate_z(X, theta=0):
    matrix = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0, 1]]
    return np.dot(matrix, X)


def pca(data, k):
    # 将数据 data 从 D 维降到 k 维
    d = data.shape[0]
    mean = np.mean(data, axis=1)
    zero_data = np.zeros(data.shape)
    # 零均值化
    for i in range(d):
        zero_data[i] = data[i] - mean[i]
    covM = np.dot(zero_data, zero_data.T)
    values, vectors = np.linalg.eig(covM)
```

```
    values = np.real(values)
    vectors = np.real(vectors)
    index = np.argsort(values)
    rightVectors = vectors[:, index[:-(k + 1): -1]]
    tmp_data = np.dot(rightVectors.T, zero_data)
    pca_data = np.zeros(data.shape)
    for i in range(d):
        pca_data[i] = np.dot(rightVectors[i], tmp_data) + mean[i]

    return zero_data, mean, rightVectors, pca_data



def visualize_2(data, pca_data, mean, vectors):
    draw = plt.subplot()
    draw.scatter(data[0], data[1], facecolor='green', label='Origin Da
ta')
    draw.scatter(pca_data[0], pca_data[1], facecolor='r', label='PCA D
ata')

    x = [mean[0] - 3 * vectors[0], mean[0] + 3 * vectors[0]]
    y = [mean[1] - 3 * vectors[1], mean[1] + 3 * vectors[1]]
    draw.plot(x, y, color='black', label='eigenVector1 direction', alp
ha=0.5)

    draw.set_title('data vs pca_data', fontsize=16)
    draw.set_xlabel('$x$', fontdict={'size': 14, 'color': 'red'})
    draw.set_ylabel('$y$', fontdict={'size': 14, 'color': 'red'})

    plt.legend()
    plt.show()
```

```
def visualize_3(data, pca_data, mean, vectors):
    draw = Axes3D(plt.figure())
    draw.scatter(data[0], data[1], data[2], facecolor='green', label='
Origin Data')
    draw.scatter(pca_data[0], pca_data[1], pca_data[2], facecolor='r',
 label='PCA Data')
    # 画出 vector 直线
    x = [mean[0] - 3 * vectors[0, 0], mean[0] + 3 * vectors[0, 0]]
    y = [mean[1] - 3 * vectors[1, 0], mean[1] + 3 * vectors[1, 0]]
    z = [mean[2] - 3 * vectors[2, 0], mean[2] + 3 * vectors[2, 0]]
    draw.plot(x, y, z, color='blue', label='eigenVector1 direction', a
lpha=1)

    x2 = [mean[0] - 3 * vectors[0, 1], mean[0] + 3 * vectors[0, 1]]
    y2 = [mean[1] - 3 * vectors[1, 1], mean[1] + 3 * vectors[1, 1]]
    z2 = [mean[2] - 3 * vectors[2, 1], mean[2] + 3 * vectors[2, 1]]
    draw.plot(x2, y2, z2, color='purple', label='eigenVector2 directio
n', alpha=1)

    draw.set_title('data vs pca_data', fontsize=16)
    draw.set_xlabel('$x$', fontdict={'size': 14, 'color': 'red'})
    draw.set_ylabel('$y$', fontdict={'size': 14, 'color': 'red'})
    draw.set_zlabel('$z$', fontdict={'size': 14, 'color': 'red'})

    plt.legend()
    plt.show()



def read_pca_face(file_path):
    face_list = read_face(file_path)
```

```
    for face in face_list:
        pca_list = []
        psnr_list = []
        for k in k_list:
            zero_data, mean, vector, pca_data = pca(face, int(k))
            pca_list.append(pca_data)
            psnr_list.append(psnr(face, pca_data))
        show_faces(face, pca_list, psnr_list)


def read_face(path):
    face_list = os.listdir(path)
    f_list = []
    for file in face_list:
        file_path = os.path.join(path, file)
        pic = Image.open(file_path).convert('L')
        # pic.show()
        f_list.append(np.asarray(pic))
    return f_list


def psnr(source, target):
    rmse = np.sqrt(np.mean((source - target) ** 2))
    return 20 * np.log10(255.0 / rmse)


def show_faces(face, face_list, psnr_list):
    plt.figure(figsize=(20, 12), frameon=False)
    size = np.ceil((len(k_list) + 1) / 2)
    plt.subplot(2, size, 1)
```

```
    plt.title('Real Image')
    plt.imshow(face, cmap='gray')
    plt.axis('off')
    for i in range(len(k_list)):
        plt.subplot(2, size, i + 2)
        plt.title('k = ' + str(k_list) + ', PSNR = ' + '{:.2f}'.format(p
snr_list[i]))
        plt.imshow(face_list[i], cmap='gray')
        plt.axis('off')
    plt.show()



def test_3D():
    data = create_data_3()
    zero_data, mean, vector, pca_data = pca(data, 2)
    visualize_3(data, pca_data, mean, vector)



def test_2D():
    data = create_data_2()
    zero_data, mean, vector, pca_data = pca(data, 1)
    visualize_2(data, pca_data, mean, vector)



def main():
    # test_2D()
    # test_3D()
    read_pca_face("data/pca_face/data")
```

```
if __name__ == '__main__':

    main()
```