哈爾濱工業大學

Harbin Institute of Technology

机器学习实验报告

PCA 模型

课程名称：	机器学习
学院：	计算学部
专业：	计算机科学与技术
学号：	**1180300811**
姓名：	孙骁
指导老师：	刘扬
学期：	**2020** 秋季学期

2020 年 **11** 月 **10** 日

目录

一、 实验目标和测试

1.1 实验目标

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）.

1.2 实验测试

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）。

二、 实验环境

1. Anaconda 4.8.4
2. Python 3.7.4
3. PyCharm 2019.1 (Professional Edition)
4. Windows 10 2004

三、 实验原理

3.1 PCA 主成分分析法

主成分分析法 (principal component analysis, PCA) 是一种常用的无监督学习方法，这一方法利用正交变换把由线性相关变量表示的观测数据转换为由少数几个由线性无关变量表示的数据，线性无关的变量称为主成分。主成分的个数通常小于原始变量的个数，所以主成分分析属于降维方法。主成分分析法主要用于发现数据中的基本结构，即数据中变量的关系。

如果超平面可以对正交属性空间的所有样本进行恰当表达，就要具有下面两个性质：

1. 最近重构性：样本点到这个超平面的距离都足够近；
2. 最大可分性：样本点在这个超平面上的投影尽可能分开。

3.1.1 中心化

PCA 算法的第一步，需要将数据全部进行中心化，即对数据集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$，$\boldsymbol{x}_i \in \mathbb{R}^n$，对每个样本均进行中心化操作，即

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{m}\sum_{j=1}^{m}\boldsymbol{x}_j. \tag{1}$$

其中 $\mu = \frac{1}{m}\sum_{j=1}^{m}\boldsymbol{x}_j$ 为样本集 D 的中心向量。经过中心化之后的线性变换即为绕原点的坐标变换，且 $\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}} = \boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 为样本集的协方差矩阵。

经过中心化后的数据，有 $\sum_{j=1}^{m}\boldsymbol{x}_j = \boldsymbol{0}$。设使用的投影坐标系的标准正交向量基为 $\boldsymbol{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_d\}, d < n$，每个样本降维后得到的坐标是

$$\boldsymbol{z} = \{z_1, z_2, \cdots, z_d\} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}. \tag{2}$$

因此，原样本集表示为

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_m^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix}, \tag{3}$$

降维后的样本集表示为

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{z}_m^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,d} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m,1} & z_{m,2} & \cdots & z_{m,d} \end{bmatrix}. \tag{4}$$

3.1.2 最近重构性原理

得到 $\boldsymbol{z}$ 后，需要对其进行重构，重构后的样本设为

$$\hat{\boldsymbol{x}} = \boldsymbol{W}\boldsymbol{z} \tag{5}$$

将式 (2) 带入式 (5)，对于整个数据集上的所有样本与重构后的样本之间的误差为：

$$\sum_{i=1}^{m}\|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_2^2 = \sum_{i=1}^{m}\|\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i - \boldsymbol{x}_i\|_2^2 \tag{6}$$

根据定义，有：

$$\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i = \boldsymbol{W}\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i\right) = \sum_{j=1}^{d}\boldsymbol{w}_j\left(\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}_i\right) \tag{7}$$

由于 $\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}_i$ 是标量，有 $\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}_i = \left(\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}_i\right)^{\mathrm{T}} = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}_j$，所以对式 (7) 做变换，有：

$$\sum_{i=1}^{m}\|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_2^2 = \sum_{i=1}^{m}\|\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i - \boldsymbol{x}_i\|_2^2 \tag{8}$$

$$= \sum_{i=1}^{m}\|\sum_{k=1}^{d}\left(\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{w}_j\right)\boldsymbol{w}_j - \boldsymbol{x}_i\|_2^2 \tag{9}$$

$$= \sum_{i=1}^{m}\|\boldsymbol{x}_i - \sum_{k=1}^{d}\left(\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{w}_j\right)\boldsymbol{w}_j\|_2^2 \tag{10}$$

根据 $\boldsymbol{X}$ 的定义，有

$$\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}\|_2^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}\left[x_{i,j} - \left(\sum_{k=1}^{d} w_{k,j} \times \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}_k\right)\right]^2 \tag{11}$$

$$= \sum_{i=1}^{m}\|\boldsymbol{x}_i - \sum_{k=1}^{d}\left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}_k\right)\boldsymbol{x}_k\|_2^2 \tag{12}$$

因此优化目标为

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}}\sum_{i=1}^{m}\|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_2^2 \tag{13}$$

$$= \arg\min_{\boldsymbol{W}}\left[-tr\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}\right)\right] \tag{14}$$

$$= \arg\max_{\boldsymbol{W}}\left[tr\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}\right)\right] \tag{15}$$

$$= \arg\max_{\boldsymbol{W}}\left[tr\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}\right)\right] \tag{16}$$

因此得到优化条件为

$$\boldsymbol{W}^* = \arg\max_{\boldsymbol{W}}\left[tr\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}\right)\right], \tag{17}$$

约束条件为

$$\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \boldsymbol{I}_{d\times d} \tag{18}$$

3.1.3 最大可分性原理

对于原始数据样本点 $\boldsymbol{x}_i$ 在降维后在新空间的超平面上的投影为 $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i$。若使得样本点的投影尽可能分开，应该使样本点在投影后的方差最大化，即使得式 () 最大化：

$$\arg\max_{\boldsymbol{W}} = \arg\max_{\boldsymbol{W}}\sum_{i=1}^{m}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i\boldsymbol{W} \tag{19}$$

$$= \arg\max_{\boldsymbol{W}} tr\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}\right) \tag{20}$$

$$\text{s.t. } \boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \boldsymbol{I}_{d\times d} \tag{21}$$

可以看到，最近重构性原理和最大可分性原理等价，只需要求出 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 的特征值即可。

因此需要对 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 进行特征值分解，对得到的特征值进行排序，假设 *n* 个特征值分别为 $\lambda_1, \lambda_2, \cdots, \lambda_n$，则提取前 *d* 大的特征值对应的单位特征向量即可构成变换矩阵 $\boldsymbol{W}$。

四、 算法实现

4.1 PCA 主成分分析法算法

PCA 主成分分析法的算法如算法 (1) 所示.

Algorithm 1 PCA 主成分分析法

Input: $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$, 低维空间维数 d'

Output: 投影矩阵 $\boldsymbol{W} = (w_1, w_2, \cdots, w_{d'})$.

1: 对所有样本进行中心化：$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{m}\sum_{j=1}^{m}\boldsymbol{x}_j$;

2: 计算样本的协方差矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$;

3: 对协方差矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 做特征值分解;

4: 取最大的 d' 个特征值所对应的特征向量 $\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{d'}$.

五、 实验步骤及结果分析

5.1 生成数据的测试

本次实验人工生成数据采用的是 Swiss Roll 数据，Swiss Roll 数据是分布在三维空间的卷形结构，正投影为漩涡状，侧投影为矩形。根据 Swiss Roll 数据的“厚度”的增加，其特征投影面将从正投影的旋涡状，变为的侧投影的矩形。数据生成的代码见附录A。

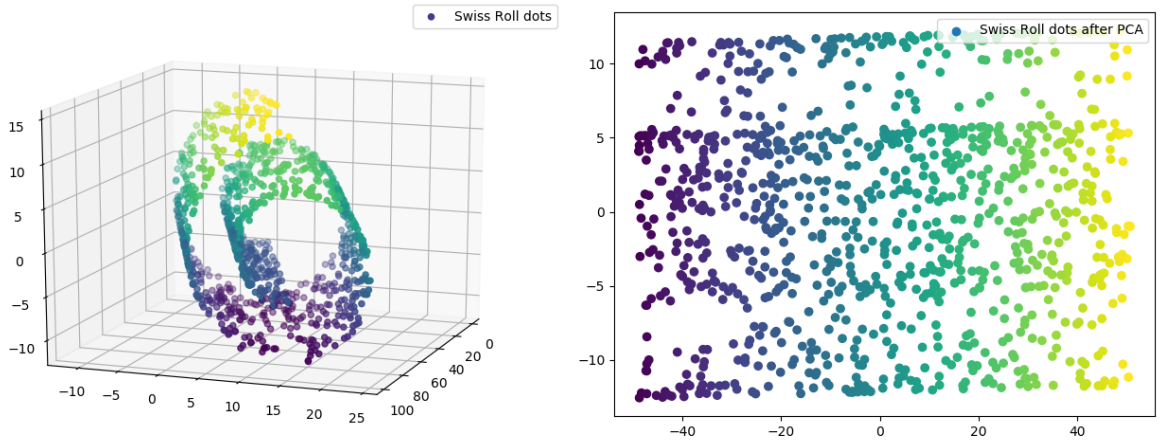
number 为每个维度上生成的数据数量，noise 为每个数据点附加的噪声值，height 为 Swiss Roll 的“厚度”。

5.2 生成数据的测试结果

当 $number = 1000, noise = 0, height = 100$ 时，生成的数据图像如图 (1a) 所示，降维的结果如图 (1b) 所示。

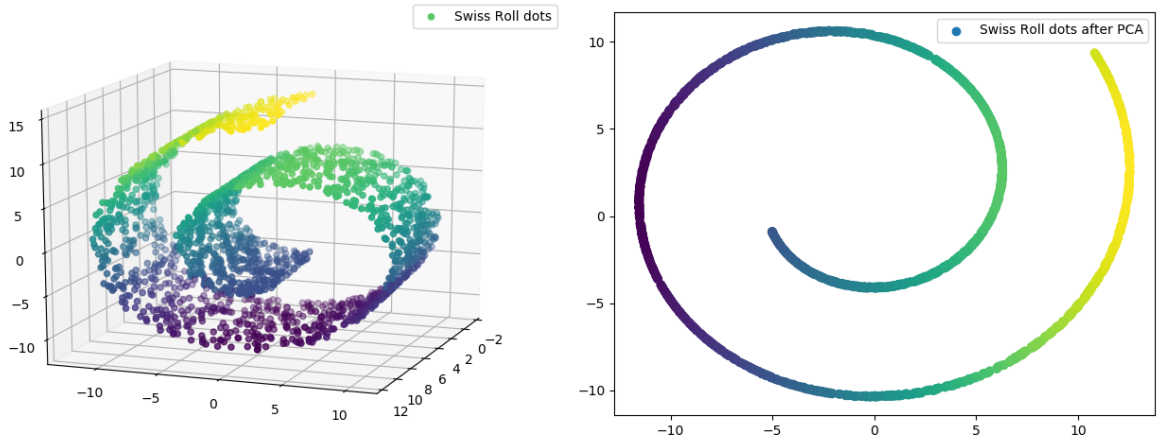
当 $number = 2000, noise = 0, height = 10$ 时，生成的数据图像如图 (2a) 所示，降维的结果如图 (2b) 所示。

当 $number = 2000, noise = 1, height = 10$ 时，生成的数据图像如图 (3a) 所示，降维的结果如图 (3b) 所示。



(a) Swiss Roll 数据　　(b) Swiss Roll 数据降维结果

图 **1**　$number = 1000, noise = 0, height = 100$ 生成的 **Swiss Roll** 数据并降维结果



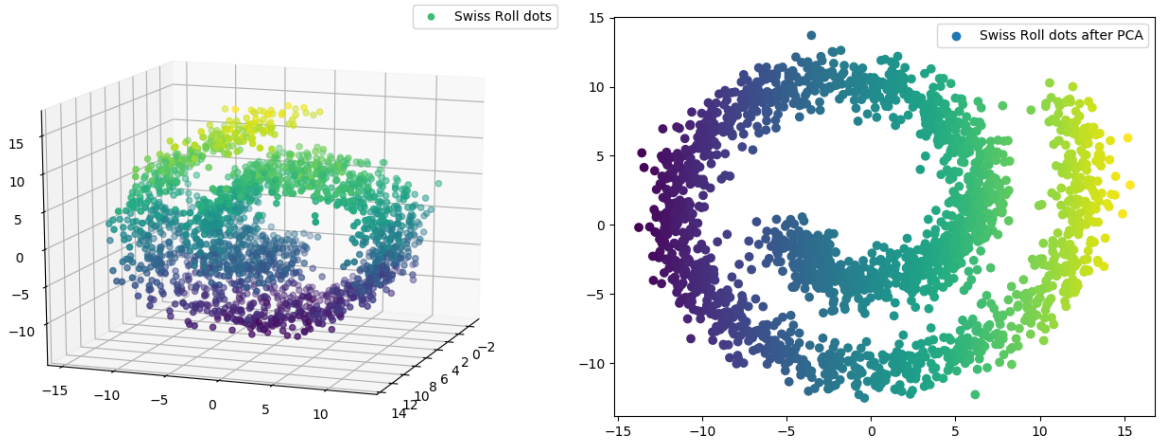
(a) Swiss Roll 数据　　(b) Swiss Roll 数据降维结果

图 **2**　$number = 2000, noise = 0, height = 10$ 生成的 **Swiss Roll** 数据并降维结果

5.3 生成数据结果分析

从上面三次实验可以看出，不同的 height 对于 Swiss Roll 数据主成分提取的影响不同。当 height=10 较小时，Swiss Roll 数据的结果较“薄”，涡旋状的正投影方差较大，所以结果为涡旋状的正投影，如图 (2b) 所示；当 height=100 较大时，Swiss Roll 数据的结果较“厚”，矩形的侧投影方差较大，所以结果为矩形的侧投影，如图 (1b) 所示。

当 height=10 时，对所有的数据点加上了 noise=1 的噪声，尽管正投影面变得不再光滑，但是可以看出依然是旋涡状，如图 (3b) 所示。



(a) Swiss Roll 数据　　(b) Swiss Roll 数据降维结果

图 **3**　$number = 2000, noise = 1, height = 10$ 生成的 **Swiss Roll** 数据并降维结果

5.4 对人脸数据的降维

人脸压缩数据来自数据集 WIDER FACE 人脸检测基准数据集，从中选择了特征较明显的 9 张图片，原图如图 (4) 所示，原图片的像素是 255*255，为了加快计算速度，使用 opencv 自带的函数，对图片进行大小重构，为 50*50，再进行降维操作，之后分别计算不同维度降维的信噪比。

对图片进行降维，降维后维度为 8，图片结果如图 (5)，计算信噪比如图 (6) 所示。

图 **4**　人脸数据原图

图 **5**　人脸数据降维维度为 **8** 的结果

```
the signal to noise ratio of the image after PCA:
The noise ratio of image 0 is 21.559582970741594
The noise ratio of image 1 is 21.863725037930845
The noise ratio of image 2 is 20.471335402900024
The noise ratio of image 3 is 25.070871312739598
The noise ratio of image 4 is 25.610500731238425
The noise ratio of image 5 is 22.22456898274831
The noise ratio of image 6 is 21.569865929566433
The noise ratio of image 7 is 24.63463885080739
The noise ratio of image 8 is 24.812022346774345
```

图 **6**　人脸数据降维维度为 **8** 的信噪比结果

对图片进行降维，降维后维度为 5，图片结果如图 (7)，计算信噪比如图 (8) 所示。

对图片进行降维，降维后维度为 3，图片结果如图 (9)，计算信噪比如图 (10) 所示。

5.5 对人脸数据降维分析

可以看出，随着降维维度的降低，人脸数据逐渐变得模糊，信噪比也越来越低，可以看到，随着降维空间的维数提高，对于源数据的信息保留的更加全面。

六、 实验结论

1. PCA 算法中舍弃了 $n-d$ 个最小的特征值对应的特征向量，一定会导致低维空间与高维空间不同，但是通过这种方式提取出了数据的主成分，对数据做了一定程度的

图 **7**　人脸数据降维维度为 **5** 的结果

```
The noise ratio of image 0 is 19.455859874677255
The noise ratio of image 1 is 19.903310110003346
The noise ratio of image 2 is 18.579591459331546
The noise ratio of image 3 is 22.894982602347756
The noise ratio of image 4 is 23.32918139660145
The noise ratio of image 5 is 20.400823972604147
The noise ratio of image 6 is 19.650715717787172
The noise ratio of image 7 is 22.25795636904955
The noise ratio of image 8 is 22.481083987806866
```

图 **8**　人脸数据降维维度为 **5** 的信噪比结果

压缩，压缩比例随降维维数的降低而提高；

2. PCA 不仅将数据压缩到低维，并且将降维之后的各维特征相互独立；
3. 通过对各维度向量计算向量均值，通过向量减法将新样本进行中心化来减小向量在不同维度上分布不均匀对降维想过的影响.

参考文献

[1] 李航, 统计学习方法 (2019.3).

[2] 周志华, 机器学习 (2016.1).

[3] WIDER FACE: A Face Detection Benchmark. (2017.3) [Data set].

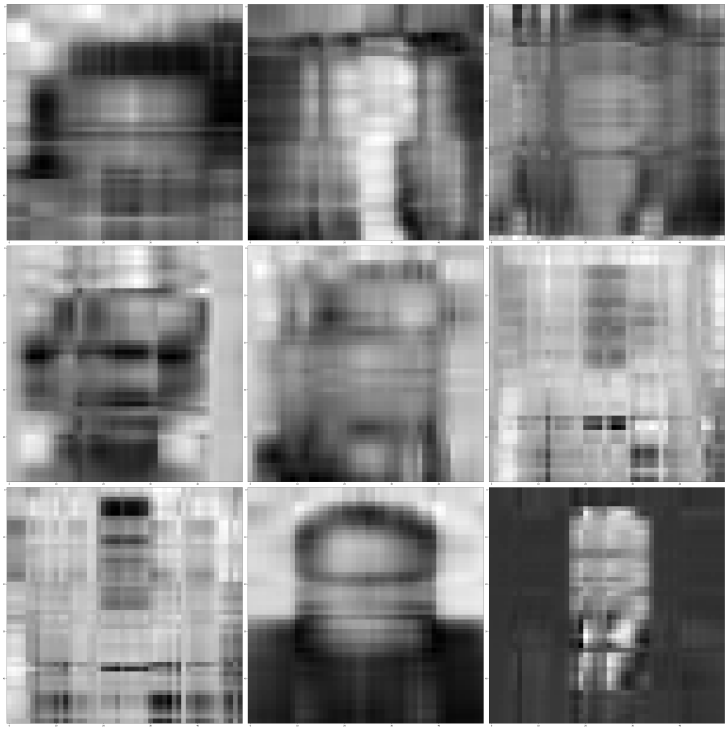


图 **9**　人脸数据降维维度为 **3** 的结果

```
the signal to noise ratio of the image after PCA:
The noise ratio of image 0 is 17.158725978176758
The noise ratio of image 1 is 18.037754653001222
The noise ratio of image 2 is 17.07622723179384
The noise ratio of image 3 is 21.237755627052803
The noise ratio of image 4 is 21.230839444721177
The noise ratio of image 5 is 18.82946934275777
The noise ratio of image 6 is 18.423813326959888
The noise ratio of image 7 is 20.263268253465412
The noise ratio of image 8 is 20.49492307126975
```

图 **10**　人脸数据降维维度为 **3** 的信噪比结果

附录 A　生成 Swiss Roll 数据–generate_data.py

```python
import numpy as np


def rotation_transformation(data, theta=0, axis='x'):
    if axis == 'x':
        rotation_matrix = [[1, 0, 0], [0, np.cos(theta), -np.sin(theta)], [0,
            np.sin(theta), np.cos(theta)]]
    elif axis == 'y':
        rotation_matrix = [[np.cos(theta), 0, np.sin(theta)], [0, 1, 0],
            [-np.sin(theta), 0, np.cos(theta)]]
    elif axis == 'z':
        rotation_matrix = [[np.cos(theta), -np.sin(theta), 0], [np.sin(theta),
```

```
            np.cos(theta), 0], [0, 0, 1]]
    else:
        print("wrong input")
        return data
    return np.dot(rotation_matrix, data)


def generate_data(number, noise, height):
    tt1 = (3 * np.pi / 2) * (1 + 2 * np.random.rand(1, number))
    x = tt1 * np.cos(tt1)
    y = height * np.random.rand(1, number)
    z = tt1 * np.sin(tt1)
    X = np.concatenate((x, y, z))
    X += noise * np.random.randn(3, number)
    X = rotation_transformation(X, 30, 'z')
    return X.T
```

附录 B　PCA 主成分分析法算法–PCA.py

```
import numpy as np


class PCA(object):

    def __init__(self, data, k):
        self.data = data
        self.k = k
        self.rows, self.columns = self.data.shape

    def pca(self):
        data_mean = np.sum(self.data, axis=0) / self.rows
        central_data = self.data - data_mean
        cov_matrix = central_data.T.dot(central_data)
        eigen_values, eigen_vector = np.linalg.eig(cov_matrix)
        eigen_value_sort = np.argsort(eigen_values)
        eigen_result = eigen_vector[:, eigen_value_sort[:-(self.k + 1):-1]]
        return central_data, eigen_result, data_mean
```

附录 C　主程序–lab4_main.py

```
from src.PCA import *
from src.generate_data import *
```

```python
from src.generate_picture import *
from src.dimensionality_reduction_image import *


def test_pca(data):
    generate_3_dimension_picture(data)
    central_data, eig_vector, data_mean = PCA(data, 2).pca()
    pca_data = np.dot(central_data, eig_vector)
    generate_2_dimension_picture(pca_data)


def test_image_data_set():
    data = read_image_data()
    image_number, image_feature = data[0].shape
    print(data.shape)
    central_data = []
    eig_vector = []
    data_mean = []
    pca_data = []
    rebuild_data = []
    for i in range(len(data)):
        central_data_i, eig_vector_i, data_mean_i = PCA(data[i], 8).pca()
        central_data.append(central_data_i)
        eig_vector.append(eig_vector_i)
        data_mean.append(data_mean_i)
        print(eig_vector[i])
        eig_vector_i = np.real(eig_vector_i)
        pca_data.append(np.dot(central_data_i, eig_vector_i))
        # print(pca_data)
        rebuild_data.append(np.dot(pca_data[i], eig_vector[i].T) + data_mean[i])
    plt.figure(figsize=(50, 50))
    for i in range(len(data)):
        plt.subplot(3, 3, i + 1)
        plt.imshow(rebuild_data[i], cmap=plt.cm.gray)
    plt.show()

    print("the signal to noise ratio of the image after PCA:")
    for i in range(len(data)):
        ratio = calculate_noise_ratio(data[i], rebuild_data[i])
        print('The noise ratio of image ' + str(i) + ' is ' + str(ratio))


def test_single_picture():
    data = read_image_data()
    image_number, image_feature = data.shape
    print(data.shape)
    central_data, eig_vector, data_mean = PCA(data, 20).pca()
```

```
    print(eig_vector)
    eig_vector = np.real(eig_vector)
    pca_data = np.dot(central_data, eig_vector)
    rebuild_data = np.dot(pca_data, eig_vector.T) + data_mean
    plt.figure(figsize=(50, 50))
    plt.imshow(rebuild_data)
    plt.show()


def main():
    data_1 = generate_data(1000, 0, 100)
    # print(np.shape(data_1))
    data_2 = generate_data(2000, 0, 10)
    data_3 = generate_data(2000, 1, 10)

    test_pca(data_1)
    test_pca(data_2)
    test_pca(data_3)

    test_image_data_set()


if __name__ == '__main__':
    main()
```

附录 D　对图片降维并计算信噪比–dimensionality_reduction_image.py

```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv
import os


def read_image_data():
    file_dir_path = '../data/'
    file_list = os.listdir(file_dir_path)
    image_data = []
    plt.figure(figsize=(50, 50))
    i = 1
    for file in file_list:
        file_path = os.path.join(file_dir_path, file)
        print("open figure " + file_path)
        plt.subplot(3, 3, i)
```

```
        with open(file_path) as f:
            img = cv.imread(file_path, cv.IMREAD_GRAYSCALE)
            print(img.shape)
            img = cv.resize(img, (50, 50), interpolation=cv.INTER_NEAREST)
            print(img.shape)
            # cv.imshow("sdg", img)
            # cv.waitKey(0)
            # cv.destroyAllWindows()
            # height, width = img.shape
            # img_temp = img.reshape(height * width)
            # data.append(img_temp)
            data = np.asarray(img)
            image_data.append(data)
            plt.imshow(img, cmap=plt.cm.gray)
            # cv.imshow("sg", img)
            # cv.waitKey(0)
            # cv.destroyAllWindows()
        i += 1
    plt.show()
    return np.asarray(image_data)


def calculate_noise_ratio(img_1, img_2):
    noise = np.mean(np.square(img_1 / 255. - img_2 / 255.))
    if noise < 1e-10:
        return 100
    return 20 * np.log10(1 / np.sqrt(noise))
```