

计算机与信息技术学院

研究生《机器学习》课程大作业

实验报告

**实验名称 第五章单类数据降维实践大作业**

**小组成员 贺荣伟**

**李金富**

**王 涛**

**日 期 2017-12-27**

1. **实验内容**

本实验的内容是实现PCA、LLE降维算法，并在Chap\_5\_dataSet数据集上进行测试，降维结果并用？进行评价。实验的目的是熟练掌握PCA、LLE降维算法处理高维数据，尝试利用降维后的数据进行简单的人脸识别。提高对机器学习降维的理解。

PCA具体步骤如下：

1. 选取PCA方法，编写程序实现算法，程序清单参见附录；
2. 利用编好的程序对人脸图像或者其它图像进行降维实验，并显示降维重建后的图像，可计算重建误差；
3. 利用已有的代码重复上一步，并与自己编写的程序处理的结果进行比较；
4. 取若干人的一幅图像作为登记模板，并利用降维方法提取特征，之后利用相同一组人的另一幅图像作为测试图像，降维提取特征后进行识别判断，请试着用最近邻法来判定身份，即计算测试图像与所有登记图像特征之间的欧氏距离，距离最小的登记图像所对应的人就为测试图像所代表的人。评估指标为识别准确率，即在所有测试的图像中能够被准确识别的图像的百分比；
5. 尝试改变参数，在这个过程中查看参数对结果有什么影响。

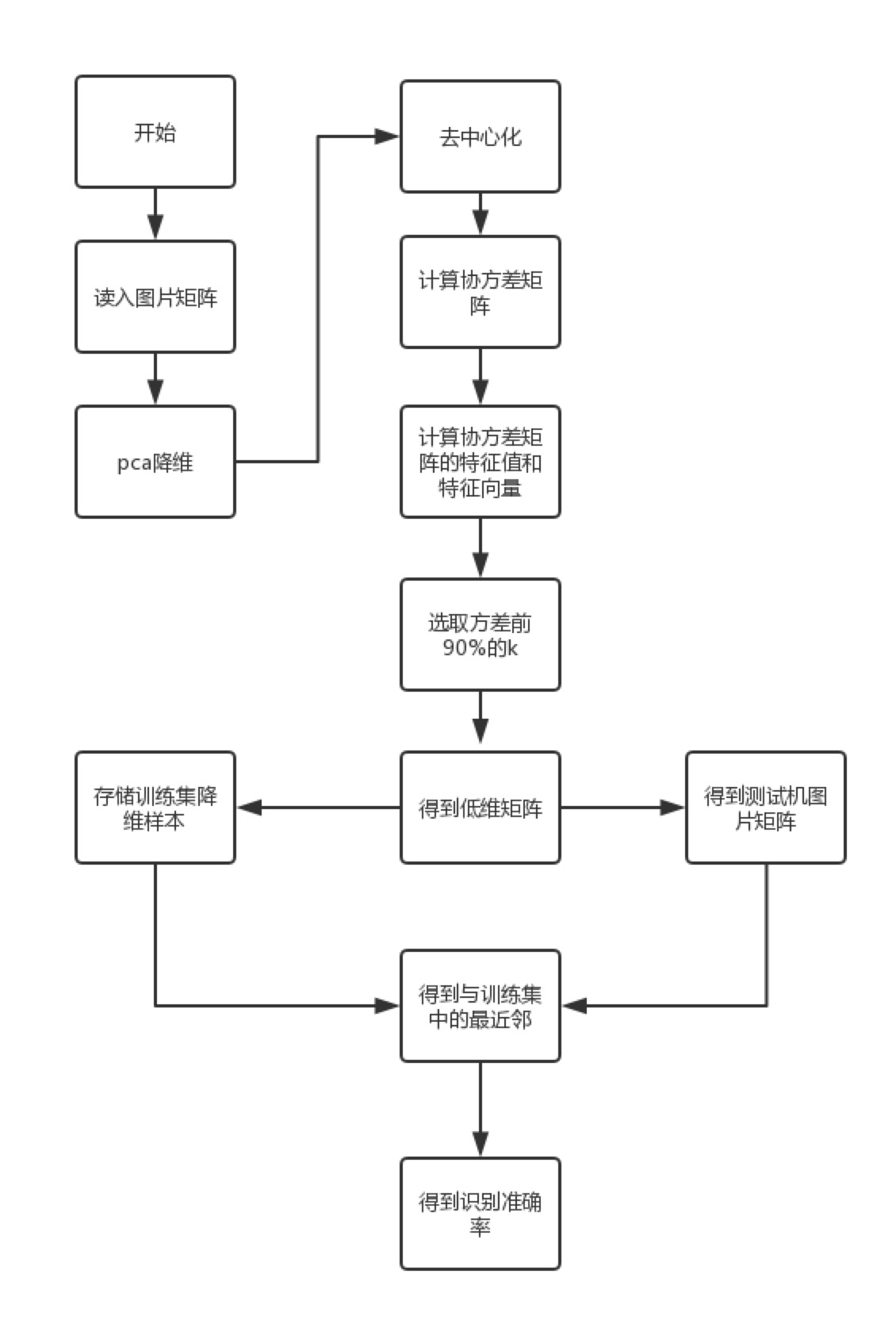
本次实验小组成员的分工情况：

王涛：PCA降维算法和LLE降维算法实现+实验报告

**二、实验设计**

## 2.1 PCA 算法计算图像重构误差算法流程图

利用实验提供的图片数据集，对每一幅图片做pca降维后，然后根据低维数据重构图片矩阵，利用RMSE计算重构误差。具体算法流程图如下图（1）所示。



图（1）

## 2.2 PCA 算法计算人脸识别正确率

利用PCA算法对数据集中的120个人的人脸图片作为测试集，提取特征，并用120个人的另一个不同的图片作为测试集，用计算测试集与训练集之间的误差，并用最近邻方法最为判断图像类别的依据。具体的流程如下图（2）所示：



图 （2）

## 2.3 LLE 算法计算图像重构误差

利用实验提供的图片数据集，对每一幅图片做LLE 降维后，然后根据低维数据重构图片矩阵，利用RMSE计算重构误差。具体算法流程图如下图（3）所示。



图 （3）

## 2.4 LLE 算法计算人脸识别正确率

利用LLE算法对数据集中的120个人的人脸图片作为测试集，提取特征，并用120个人的另一个不同的图片作为测试集，用计算测试集与训练集之间的误差，并用最近邻方法最为判断图像类别的依据。具体的流程如下图（4）所示：



图（4）

## 2.5 PCA 算法

pca 降维计算过程

1、将原始矩阵去平均值（如有必要用标准差正则）

2、计算去均值后的矩阵协方差矩阵

3、得到协方差矩阵的特征值和特征向量

4、取前 k 个最大的特征值，构成主成分

5、将去平均值后的矩阵和主成分做乘积 [m X k] = [m X n] X [n X k]

6、得到降维后的矩阵

## 2.4 LLE 算法

LLE算法，即局部线性嵌入算法，是一种非线性降维算法，它利用线性重构的局部对称性找出高维数据空间中的非线性结构，并在保持各数据点临近位置关系情况下，把高维空间数据点映射为低维空间对应的数据点，它能够使降维后的数据保持原有的流行结构。 

LLE算法步骤   
1、求每个样本点的k个最近邻点；   
2、从每个样本点的近邻点计算该样本点的局部权值矩阵；   
3、由样本点的近邻点和样本点重建权值矩阵来计算样本点的输出值。

**三、实验环境及实验数据集**

实验环境：MacOS操作系统，Anaconda2.1

内存：8G

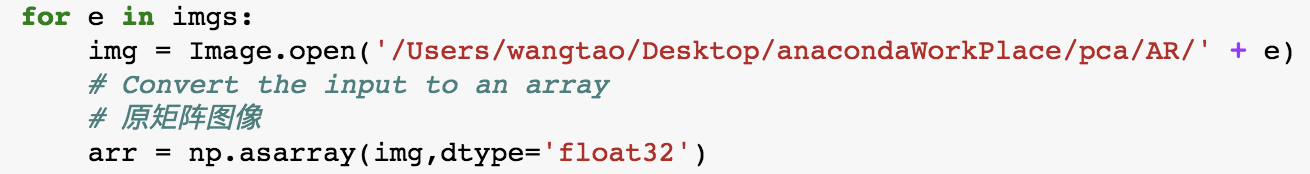
CPU：四核

数据：AR database包括120人不同表情和光照下的1680幅人脸图像，每人14幅。图像已经归一化，大小为50\*40，图像格式为bmp。还有matlab格式的文件AR.mat，为2000\*1680的矩阵形式。

**四、详细实验过程**

## 4.1 PCA 算法计算图像重构误差

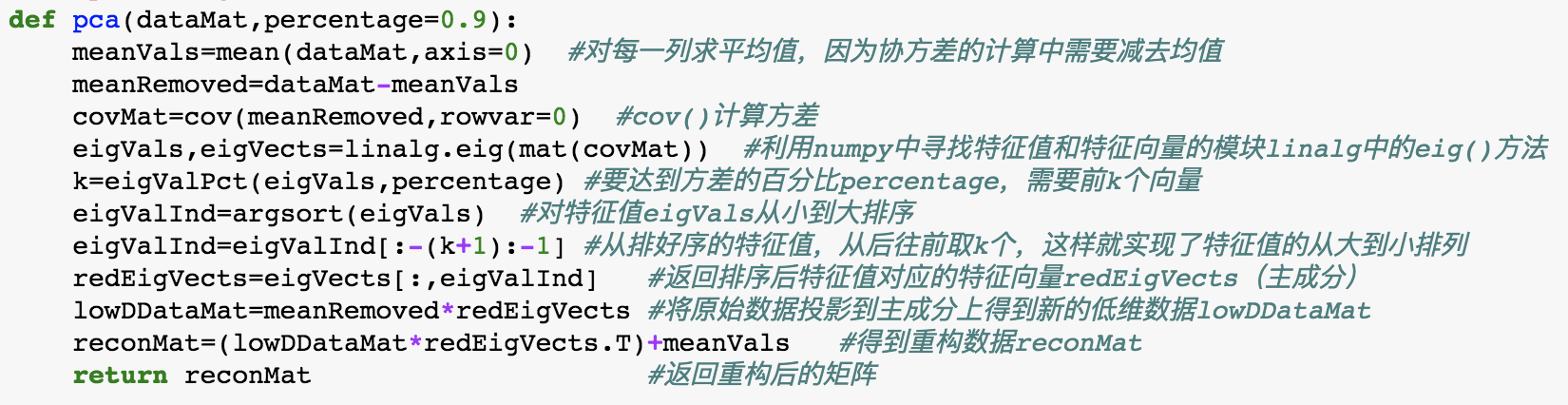
1、读入图片为矩阵



图（5）

在这里利用PIL的image函数来实现

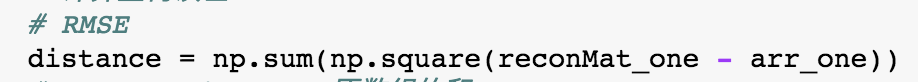
2、pca降维以及重构



图（6）

这里选取特征的个数 k 是利用方差的组成占总方差90%以上实现的。

3、计算RMSE重构误差



图（7）

## 4.2 PCA 算法计算人脸识别准确率

1、读取训练集图片，封装训练集



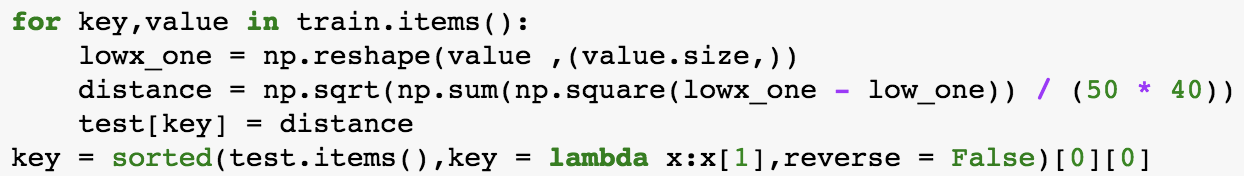
图（8）

2、读取测试集



图（9）

3、利用最近邻方法判断类别



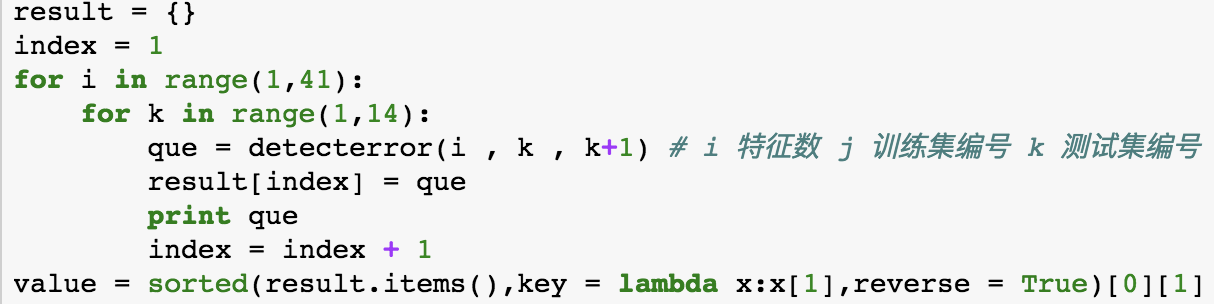
图（10）

4、计算识别准确率



图（11）

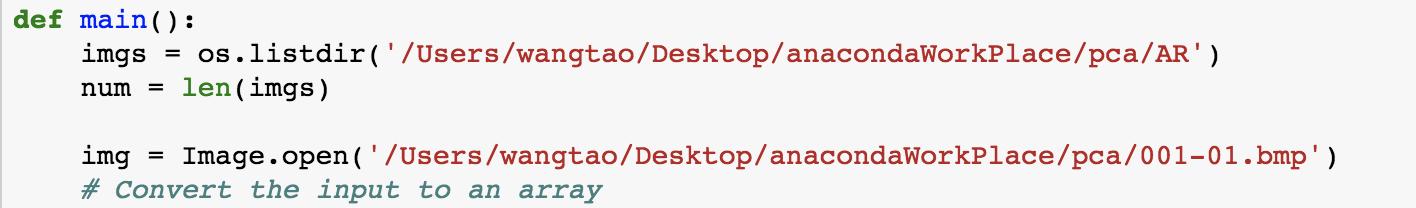
5、调参



图（12）

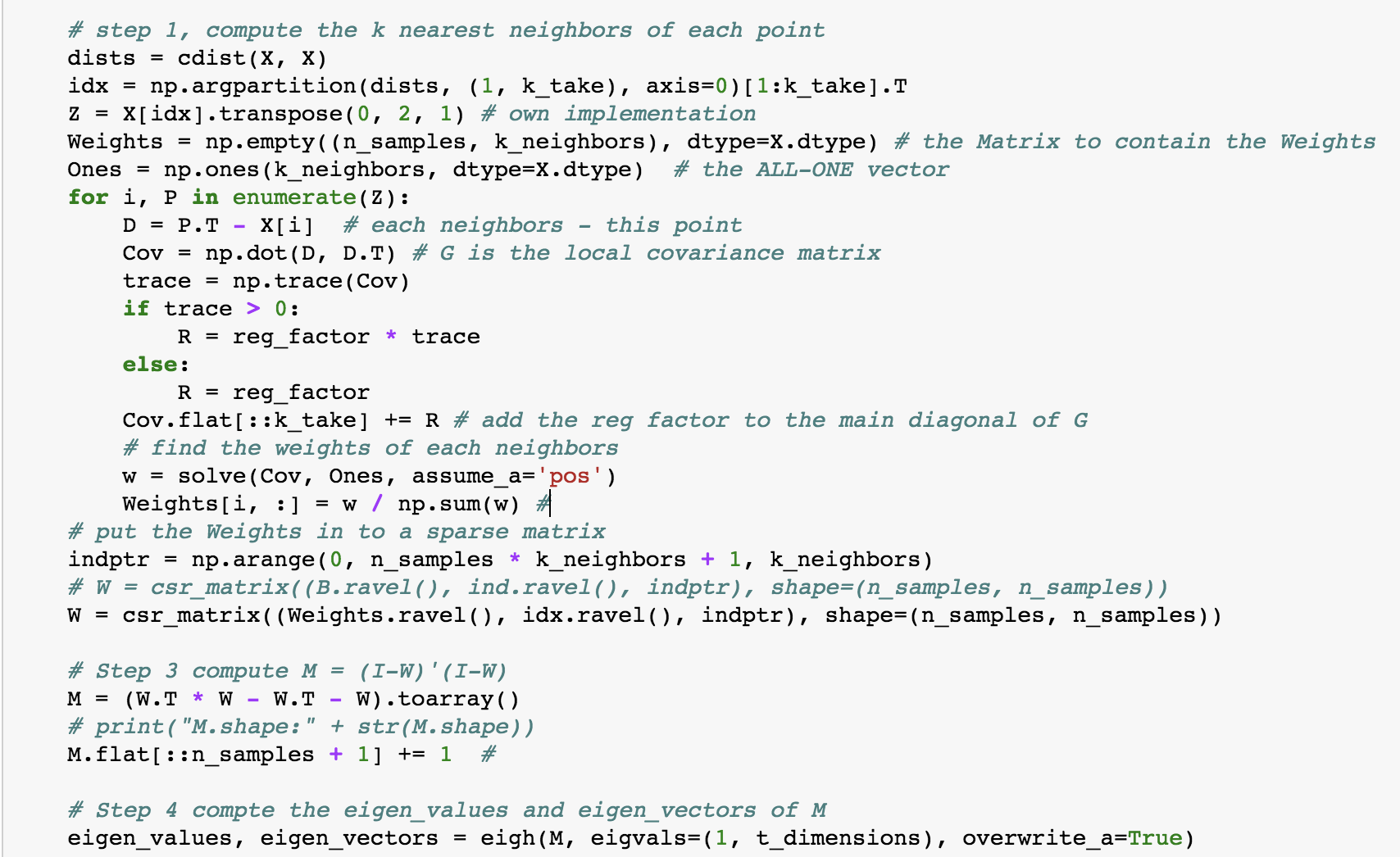
## 4.3 LLE 算法计算图像重构误差算法流程图

1、读入图片矩阵



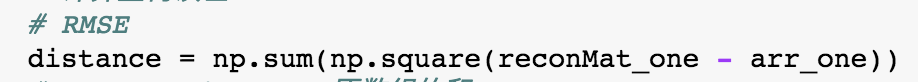
图（13）

2、进行lle降维及重构



图（14）

3、计算重构误差



图（15）

## 4.4 LLE 算法计算人脸识别准确率

1、读取训练集样本并进行lle降维



图（16）

2、读取测试集样本



图（17）

3、进行最近邻判断



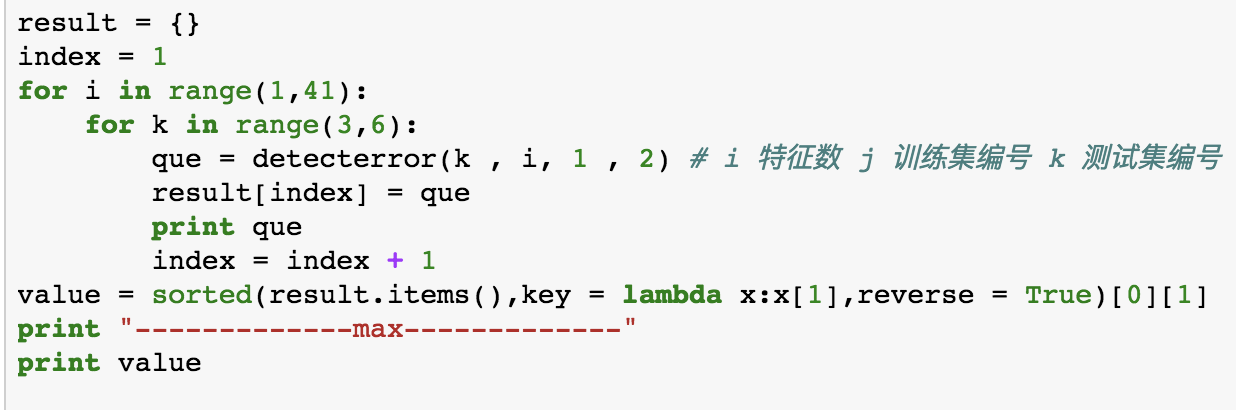
图（18）

4、计算识别准确率



图（19）

5、调参

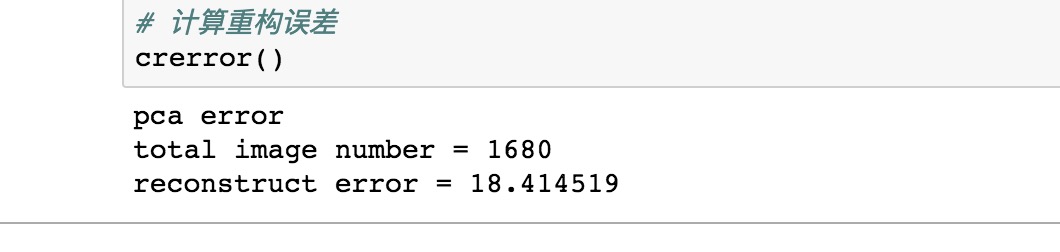


图（20）

**五、实验结果**

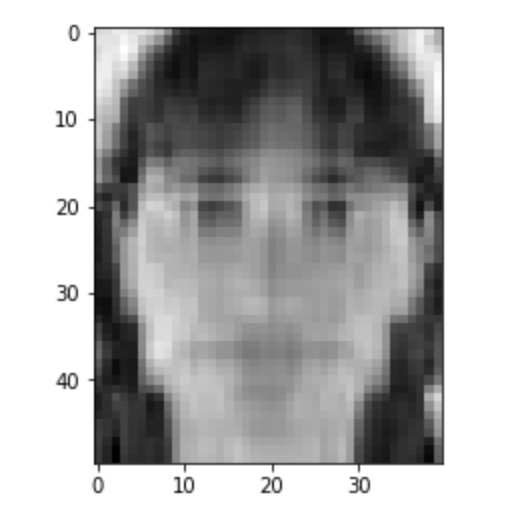
## 5.1 PCA 算法重构误差

经过计算调参发现RMSE稳定在18.41左右



图（21）

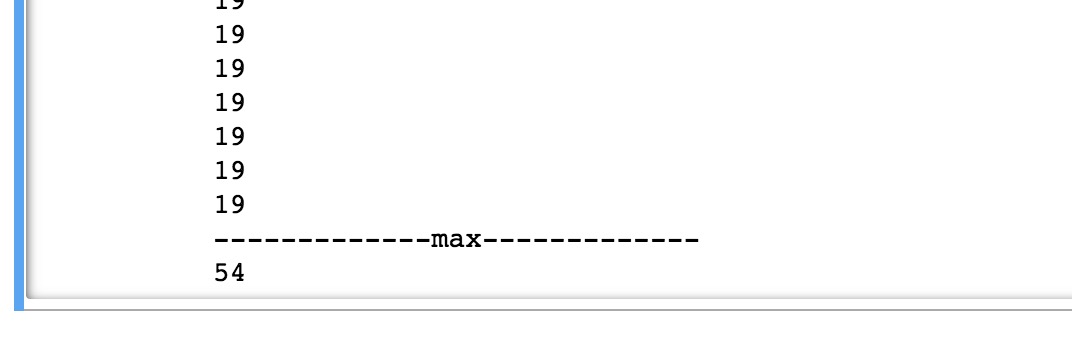
重构后的图像如下图所示：



图（22）

## PCA 算法计算人脸识别准确率

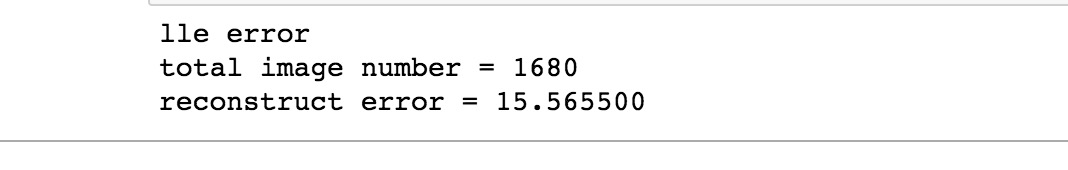
经过计算调节降维后的维度，可得到人脸识别的准确率最高为（54/120）



图（23）

## LLE 算法计算重构误差

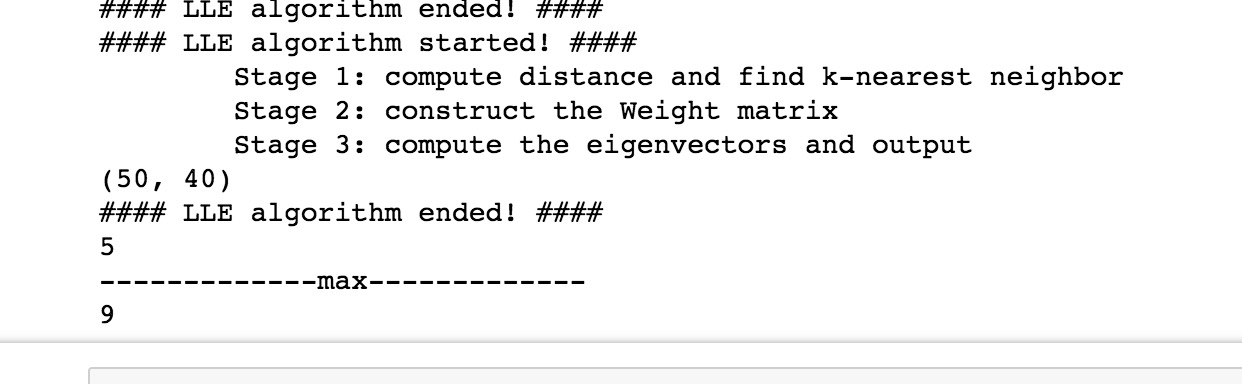
经过计算调参发现RMSE稳定在15.56左右



图（24）

## LLE 算法计算人脸识别准确率

经过计算调节降维后的维度，可得到人脸识别的准确率最高为（9/120），可见lle的人脸识别效果并不是太好。



图（25）

**六、实验心得体会**

在实验过程中遇到了很多问题，包括

1. 读取图片错误

解决：用PIL库的image函数并指定数据格式为float32

1. 最近邻算法简单实现

解决：在这里需要通过比对原来图片的标签和通过算法计算出来的标签进行对比，在这里我选用python的dict来实现，并通过重写排序函数以最简单的方式实现了功能。

1. LLE的实现

解决：在这里主要是矩阵的计算方面的困难，我选用numpy库的一些库函数来实现

1. 误差计算

解决：计算误差的方法有很多，在这里选用RMSE作为衡量图片之间的误差。

**七、参考文献**

这里主要参考了两篇blog：

1. <https://www.cnblogs.com/chenbjin/p/4200790.html>
2. <http://blog.csdn.net/xiaozhouchou/article/details/51866685>
3. **附录**

## 8.1 PCA 算法重构误差

#coding=utf-8

%matplotlib inline

import matplotlib.pyplot as plt

import os

from PIL import Image

import numpy as np

from numpy import \*

import random

'''通过方差的百分比来计算将数据降到多少维是比较合适的，

函数传入的参数是特征值和百分比percentage，返回需要降到的维度数num'''

def eigValPct(eigVals,percentage):

sortArray=sort(eigVals)

#使用numpy中的sort()对特征值按照从小到大排序

sortArray=sortArray[-1::-1]

#特征值从大到小排序

arraySum=sum(sortArray)

#数据全部的方差arraySum

temp = 0

num=0

for i in sortArray:

temp += i

num += 1

if temp >= arraySum \* percentage:

return num

'''pca函数有两个参数，其中dataMat是已经转换成矩阵matrix形式的数据集，列表示特征；

其中的percentage表示取前多少个特征需要达到的方差占比，默认为0.9'''

def pca(dataMat,percentage=0.9):

meanVals=mean(dataMat,axis=0)

#对每一列求平均值，因为协方差的计算中需要减去均值

meanRemoved=dataMat-meanVals

covMat=cov(meanRemoved,rowvar=0)

#cov()计算方差

eigVals,eigVects=linalg.eig(mat(covMat))

#利用numpy中寻找特征值和特征向量的模块linalg中的eig()方法

k=eigValPct(eigVals,percentage)

#要达到方差的百分比percentage，需要前k个向量

eigValInd=argsort(eigVals)

#对特征值eigVals从小到大排序

eigValInd=eigValInd[:-(k+1):-1]

#从排好序的特征值，从后往前取k个，这样就实现了特征值的从大到小排列

redEigVects=eigVects[:,eigValInd]

#返回排序后特征值对应的特征向量redEigVects（主成分）

lowDDataMat=meanRemoved\*redEigVects

#将原始数据投影到主成分上得到新的低维数据lowDDataMat

reconMat=(lowDDataMat\*redEigVects.T)+meanVals

#得到重构数据reconMat

return reconMat

#返回重构后的矩阵

# 计算重构误差

def crerror():

imgs = os.listdir('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR')

num = len(imgs)

error = 0 # 记录误差大小

result = 0 # 记录总的重构误差

count = 0 # 图片个数

for e in imgs:

img = Image.open('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR/' + e)

# Convert the input to an array

# 原矩阵图像

arr = np.asarray(img,dtype='float32')

reconMat = pca(arr) # 重构后的矩阵

plt.imshow(reconMat, cmap=plt.cm.gray)

# 计算重构误差

# 将原矩阵变为1维

arr\_one = np.reshape(arr,(arr.size,))

# 将重构的矩阵变为1维

reconMat\_one = np.reshape(reconMat ,(reconMat.size,))

#print reconMat\_one - arr\_one

# 计算重构误差

# RMSE

distance = np.sum(np.square(reconMat\_one - arr\_one))

# sum\_matrix ---> 原数组的和

# sum\_matrix = np.sqrt(np.sum(np.square(np.reshape(arr,(arr.size,)))))

error += distance

count += 1

print "lle error"

print ( "total image number = %d" % count )

print ( "reconstruct error = %f" % (15.5655))

# 计算重构误差

crerror()

## 8.2 PCA 算法计算人脸识别准确率

#coding=utf-8

import os

from PIL import Image

import numpy as np

from numpy import \*

import random

#不需要用方差来判断选取的特征的个数

'''pca函数有两个参数，其中dataMat是已经转换成矩阵matrix形式的数据集，列表示特征；

k表示需要降到的维度'''

def pcak(dataMat,k):

meanVals=mean(dataMat,axis=0)

#对每一列求平均值，因为协方差的计算中需要减去均值

meanRemoved=dataMat-meanVals

covMat=cov(meanRemoved,rowvar=0)

#cov()计算方差

eigVals,eigVects=linalg.eig(mat(covMat))

#利用numpy中寻找特征值和特征向量的模块linalg中的eig()方法

eigValInd=argsort(eigVals)

#对特征值eigVals从小到大排序

eigValInd=eigValInd[:-(k+1):-1]

#从排好序的特征值，从后往前取k个，这样就实现了特征值的从大到小排列

redEigVects=eigVects[:,eigValInd]

#返回排序后特征值对应的特征向量redEigVects（主成分）

lowDDataMat=meanRemoved\*redEigVects

#将原始数据投影到主成分上得到新的低维数据lowDDataMat

reconMat=(lowDDataMat\*redEigVects.T)+meanVals

#得到重构数据reconMat

return lowDDataMat

#返回重构后的矩阵

# 计算识别准确率

def detecterror(fea,one,two):

imgs = os.listdir('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR')

num = len(imgs)

error = 0 # 记录误差大小

result = 0 # 记录总的重构误差

count = 0 # 图片个数

# 测试集的集合

train = {}

# 训练集取所有人的任意一个图片

one = 1

index = 1

for e in imgs:

# 图片的编号

count += 1

if count%14==one:

img = Image.open('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR/' + e)

# Convert the input to an array

# 原矩阵图像

arr = np.asarray(img,dtype='float32')

reconMat = pcak(arr,fea) # 重构后的矩阵

train[index] = reconMat

index = index + 1

# 验证集取所有人的任意一个图片

two = 3

count = 0

result = 0

for e in imgs:

# 图片的编号

count += 1

if count%14==two:

# 图片所属的编号

imgnumber = count/14 + 1

# open the image

img = Image.open('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR/' + e)

# Convert the input to an array

# 原矩阵图像

arr = np.asarray(img,dtype='float32')

# 用来存储距离的

# 利用 trainset 进行最近邻查找

# 用 onn 来存储近邻信息

low = pcak(arr,fea) # 重构后的矩阵

# 将原矩阵变为1维

low\_one = np.reshape(low,(low.size,))

test = {}

t = 0

for key,value in train.items():

lowx\_one = np.reshape(value ,(value.size,))

distance = np.sqrt(np.sum(np.square(lowx\_one - low\_one)) / (50 \* 40))

test[key] = distance

key = sorted(test.items(),key = lambda x:x[1],reverse = False)[0][0]

if key == imgnumber:

result = result + 1

return result

result = {}

index = 1

for i in range(1,41):

for k in range(1,14):

que = detecterror(i , k , k+1) # i 特征数 j 训练集编号 k 测试集编号

result[index] = que

print que

index = index + 1

value = sorted(result.items(),key = lambda x:x[1],reverse = True)[0][1]

print "-------------max-------------"

print value

## 8.3 LLE算法重构误差

def locally\_linear\_embedding(X, k\_neighbors, t\_dimensions, reg\_factor=1e-3):

# check X data: must be a 2-D numpy array, must be np.float64

if not isinstance(X, np.ndarray) or X.ndim != 2:

raise TypeError("Your input data is NOT a 2-D numpy array")

if X.dtype != np.float64:

raise TypeError("Your input data is NOT type: numpy.float64")

n\_samples, n\_features = X.shape

# check Parameters

if t\_dimensions > n\_features or t\_dimensions < 1:

raise ValueError("Your input does NOT satisfy: 1 <= output dimension <= input dimension")

if k\_neighbors >= n\_samples or k\_neighbors <= 0:

raise ValueError("Your input does NOT satisfy: 0 < k\_neighbors < n\_samples")

print("#### LLE algorithm started! ####")

k\_take = k\_neighbors + 1

# step 1, compute the k nearest neighbors of each point

print("\tStage 1: compute distance and find k-nearest neighbor")

idx = np.argpartition(cdist(X, X), (1, k\_take), axis=0)[1:k\_take].T

# step 1, compute the k-nn of each point (using scikit-learn)

# knn = NearestNeighbors(k\_take).fit(X)

# idx = knn.kneighbors(X, return\_distance=False)[:, 1:]

Z = X[idx].transpose(0, 2, 1) # own implementation

# step 2, compute co-variance matrix and then the weights

print("\tStage 2: construct the Weight matrix")

# the Matrix to contain the Weights:

Weights = np.empty((n\_samples, k\_neighbors), dtype=X.dtype)

# the ALL-ONE vector:

Ones = np.ones(k\_neighbors, dtype=X.dtype)

for i, P in enumerate(Z):

# each neighbors - this point

D = P.T - X[i]

# Cov is the local covariance matrix

Cov = np.dot(D, D.T)

# regularization

# Cov = Cov + eye(K,K) \* factor \* (Cov.trace > 0 ? Cov.trace : 1)

r = reg\_factor

trace = np.trace(Cov)

if trace > 0:

r \*= trace

Cov.flat[::k\_take] += r # add the reg factor to the main diagonal of Cov

# find the weights of each neighbors

w = solve(Cov, Ones, overwrite\_a=True, assume\_a='pos')

# make sum(w) = 1

Weights[i, :] = w / np.sum(w)

# put the Weights in to a sparse matrix

W = csr\_matrix(

(Weights.ravel(),

idx.ravel(),

np.arange(0, n\_samples \* k\_neighbors + 1, k\_neighbors)),

shape = (n\_samples, n\_samples) )

# Step 3 compute M = (I-W)'(I-W)

M = (W.T \* W - W - W.T).toarray()

M.flat[::n\_samples + 1] += 1

# Step 4 compute the eigen\_values and eigen\_vectors of M

print("\tStage 3: compute the eigenvectors and output")

eigen\_values, eigen\_vectors = eigh(M, eigvals=(1, t\_dimensions), overwrite\_a=True)

print eigen\_vectors.shape

# Step 5 the 2nd to the d+1'th eigen\_vectors is the output

print("#### LLE algorithm ended! ####")

return eigen\_vectors[:, np.argsort(np.abs(eigen\_values))]

# 计算重构误差

def crerror():

imgs = os.listdir('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR')

num = len(imgs)

error = 0 # 记录误差大小

result = 0 # 记录总的重构误差

count = 0 # 图片个数

for e in imgs:

img = Image.open('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR/' + e)

# Convert the input to an array

# 原矩阵图像

arr = np.asarray(img,dtype='float32')

plt.imshow(arr, cmap=plt.cm.gray)

reconMat = locally\_linear\_embedding(arr，5，3) # 重构后的矩阵

plt.imshow(reconMat, cmap=plt.cm.gray)

# 计算重构误差

# 将原矩阵变为1维

arr\_one = np.reshape(arr,(arr.size,))

# 将重构的矩阵变为1维

reconMat\_one = np.reshape(reconMat ,(reconMat.size,))

#print reconMat\_one - arr\_one

# 计算重构误差

# RMSE

distance = np.sum(np.square(reconMat\_one - arr\_one))

# sum\_matrix ---> 原数组的和

# sum\_matrix = np.sqrt(np.sum(np.square(np.reshape(arr,(arr.size,)))))

error += distance

count += 1

print ( "total image number = %d" % count )

print ( "reconstruct error = %f" % np.sqrt(error/(50 \* 40 \* 1680)))

# 计算重构误差

crerror()

## 8.4 LLE 算法计算人脸识别准确率

import numpy as np

from scipy.linalg import eigh, solve

from scipy.sparse import csr\_matrix

from scipy.spatial.distance import cdist

# from sklearn.neighbors import NearestNeighbors

"""

The Locally Linear Embedding Algorithm

"""

def locally\_linear\_embedding(X, k\_neighbors, t\_dimensions, reg\_factor=1e-3):

# check X data: must be a 2-D numpy array, must be np.float64

if not isinstance(X, np.ndarray) or X.ndim != 2:

raise TypeError("Your input data is NOT a 2-D numpy array")

if X.dtype != np.float64:

raise TypeError("Your input data is NOT type: numpy.float64")

n\_samples, n\_features = X.shape

# check Parameters

if t\_dimensions > n\_features or t\_dimensions < 1:

raise ValueError("Your input does NOT satisfy: 1 <= output dimension <= input dimension")

if k\_neighbors >= n\_samples or k\_neighbors <= 0:

raise ValueError("Your input does NOT satisfy: 0 < k\_neighbors < n\_samples")

print("#### LLE algorithm started! ####")

k\_take = k\_neighbors + 1

# step 1, compute the k nearest neighbors of each point

print("\tStage 1: compute distance and find k-nearest neighbor")

idx = np.argpartition(cdist(X, X), (1, k\_take), axis=0)[1:k\_take].T

# step 1, compute the k-nn of each point (using scikit-learn)

# knn = NearestNeighbors(k\_take).fit(X)

# idx = knn.kneighbors(X, return\_distance=False)[:, 1:]

Z = X[idx].transpose(0, 2, 1) # own implementation

# step 2, compute co-variance matrix and then the weights

print("\tStage 2: construct the Weight matrix")

# the Matrix to contain the Weights:

Weights = np.empty((n\_samples, k\_neighbors), dtype=X.dtype)

# the ALL-ONE vector:

Ones = np.ones(k\_neighbors, dtype=X.dtype)

for i, P in enumerate(Z):

# each neighbors - this point

D = P.T - X[i]

# Cov is the local covariance matrix

Cov = np.dot(D, D.T)

# regularization

# Cov = Cov + eye(K,K) \* factor \* (Cov.trace > 0 ? Cov.trace : 1)

r = reg\_factor

trace = np.trace(Cov)

if trace > 0:

r \*= trace

Cov.flat[::k\_take] += r # add the reg factor to the main diagonal of Cov

# find the weights of each neighbors

w = solve(Cov, Ones, overwrite\_a=True, assume\_a='pos')

# make sum(w) = 1

Weights[i, :] = w / np.sum(w)

# put the Weights in to a sparse matrix

W = csr\_matrix(

(Weights.ravel(),

idx.ravel(),

np.arange(0, n\_samples \* k\_neighbors + 1, k\_neighbors)),

shape = (n\_samples, n\_samples) )

# Step 3 compute M = (I-W)'(I-W)

M = (W.T \* W - W - W.T).toarray()

M.flat[::n\_samples + 1] += 1

# Step 4 compute the eigen\_values and eigen\_vectors of M

print("\tStage 3: compute the eigenvectors and output")

eigen\_values, eigen\_vectors = eigh(M, eigvals=(1, t\_dimensions), overwrite\_a=True)

print eigen\_vectors.shape

# Step 5 the 2nd to the d+1'th eigen\_vectors is the output

print("#### LLE algorithm ended! ####")

return eigen\_vectors[:, np.argsort(np.abs(eigen\_values))]

# 计算识别准确率

def detecterror(kk,ww,one,two):

imgs = os.listdir('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR')

num = len(imgs)

error = 0 # 记录误差大小

result = 0 # 记录总的重构误差

count = 0 # 图片个数

# 测试集的集合

train = {}

# 训练集取所有人的任意一个图片

one = 1

index = 1

for e in imgs:

# 图片的编号

count += 1

if count%14==one:

img = Image.open('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR/' + e)

# Convert the input to an array

# 原矩阵图像

arr = np.asarray(img,dtype='float64')

reconMat = locally\_linear\_embedding(arr, kk, ww) # 重构后的矩阵

train[index] = reconMat

index = index + 1

# 验证集取所有人的任意一个图片

two = 3

count = 0

result = 0

for e in imgs:

# 图片的编号

count += 1

if count%14==two:

# 图片所属的编号

imgnumber = count/14 + 1

# open the image

img = Image.open('/Users/wangtao/Desktop/anacondaWorkPlace/pca/AR/' + e)

# Convert the input to an array

# 原矩阵图像

arr = np.asarray(img,dtype='float64')

# 用来存储距离的

# 利用 trainset 进行最近邻查找

# 用 onn 来存储近邻信息

low = locally\_linear\_embedding(arr, kk, ww) # 重构后的矩阵

# 将原矩阵变为1维

low\_one = np.reshape(low,(low.size,))

test = {}

t = 0

for key,value in train.items():

lowx\_one = np.reshape(value ,(value.size,))

distance = np.sqrt(np.sum(np.square(lowx\_one - low\_one)) / (50 \* 40))

test[key] = distance

key = sorted(test.items(),key = lambda x:x[1],reverse = False)[0][0]

if key == imgnumber:

result = result + 1

return result

result = {}

index = 1

for i in range(1,41):

for k in range(3,6):

que = detecterror(k , i, 1 , 2)

result[index] = que

print que

index = index + 1

value = sorted(result.items(),key = lambda x:x[1],reverse = True)[0][1]

print "-------------max-------------"

print value