模式识别 作业一

PCA及白化变换实验

使用语言

Python

任务(a)(b)(c)

实现代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 中文显示
```

```python
plt.rcParams["font.sans-serif"] = ["Hiragino Sans GB"] #解决中文字符乱码的问题
plt.rcParams["axes.unicode_minus"] = False #正常显示负号

# 创建图
plt.figure()

#--------任务(a)-------
# 导入生成的随机数据
data = pd.read_excel('data.xlsx', index_col = 0)
x1 = data['x']
y1 = data['y']

# 画图
plt.scatter(x1,y1,color = 'c',marker = 'p', label = '原始数据')
plt.xlabel("x",fontsize = 12)
plt.ylabel("y",fontsize = 12)

#--------任务(b)-------
# 构造np.array类型的数据矩阵A
A = np.array(data)

# 对每一个属性的样本求均值
MEAN = np.mean(A, axis=0)  # 沿轴0调用mean函数

# 去中心化
X = np.subtract(A, MEAN)

# 计算协方差矩阵
COV = np.cov(X.T)

# 计算特征值和特征向量 W:特征值 V:特征向量
W, V = np.linalg.eig(COV)
# 这里求出的W并非按照大小进行排序后的结果 此处进行优化 以保证与api求得结果相似
# 对特征值按照大小降序排序 此处返回值是特征值对应的下标
sorted_index = np.argsort(-W) # 此处将参数设定为[-][参数名称]以表明是降序
tW = W[sorted_index[::1]] # 按sorted_index中的顺序依次取W中元素 存储在tW中
W = tW
tV = V[:, sorted_index[::1]] # 按sorted_index中的顺序依次取V中元素 存储在tV中
V = tV

# 计算主成分贡献率以及累计贡献率
```

```
sum_lambda = np.sum(W)  # 特征值的和
f = np.divide(W, sum_lambda)  # 每个特征值的贡献率（特征值 / 总和）
# 要求保留两个维度 此处不计算前几个贡献率的和>0.9
# 前两大特征值对应的特征向量为:
e1 = V.T[0]
e2 = V.T[1]

# 计算主成分值（已去中心化）X是去中心化后的结果
z1 = np.dot(X, e1)
z2 = np.dot(X, e2)

# 输出降维后的结果（已去中心化）
RES = np.array([z1, z2])
RES = RES.T # 转制一遍之后是最终结果

# 画图
RES_df = pd.DataFrame(RES)
# RES_df.to_excel('my_RES.xlsx')
RES_df.columns = ['x', 'y']
x2 = RES_df['x']
y2 = RES_df['y']

# 画图
plt.scatter(x2,y2,color = 'r',marker = 'p', label = 'PCA变换')
plt.xlabel("x",fontsize = 12)
plt.ylabel("y",fontsize = 12)

#--------任务(c)-------
# 创建特征值构成的对角矩阵D 求D的-1/2次方
new_W = W ** (-1 / 2)
D = np.diag(new_W)

# V、D相乘 作为白化处理中前面要乘的矩阵
white_V = np.dot(V, D)
e1 = white_V.T[0]
e2 = white_V.T[1]

# 计算主成分值（已去中心化）X是去中心化后的结果
z1 = np.dot(X, e1)
z2 = np.dot(X, e2)

# 输出降维后的结果（已去中心化）
```

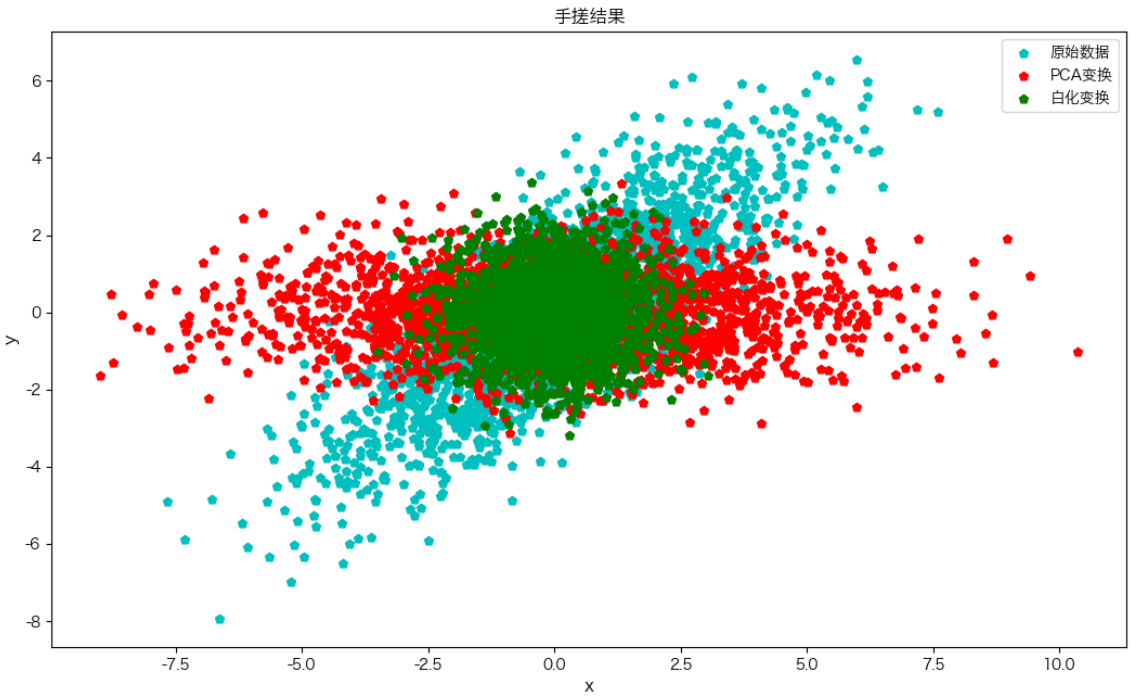
```
RES_white = np.array([z1, z2])
RES_white = RES_white.T # 转制一遍之后是最终结果

# 画图
RES_df_white = pd.DataFrame(RES_white)
# RES_df_white.to_excel('my_white_RES.xlsx')
RES_df_white.columns = ['x', 'y']
x3 = RES_df_white['x']
y3 = RES_df_white['y']

# 画图
plt.scatter(x3,y3,color = 'g',marker = 'p', label = '白化变换')
plt.xlabel("x",fontsize = 12)
plt.ylabel("y",fontsize = 12)

# 最终展示
plt.legend() # 图例
plt.title('手搓结果')
plt.show()
```

图像导出



> 如图为本人手写的代码运行的结果

为确保结果正确，以sklearn中内置的 `PCA` api进行验证

api使用代码如下

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# 中文显示
plt.rcParams["font.sans-serif"] = ["Hiragino Sans GB"] #解决中文字符乱码的问题
plt.rcParams["axes.unicode_minus"] = False #正常显示负号

#--------原始数据-------
# 读取原始数据
data = pd.read_excel('data.xlsx', index_col = 0)
x = data['x']
y = data['y']

# 画图
plt.scatter(x,y,color = 'c',marker = 'p', label = '原始数据')
plt.xlabel("x",fontsize = 12)
plt.ylabel("y",fontsize = 12)

#--------PCA-------
A = np.array(data)
pca = PCA(n_components = 2) # 保留两个维度
pca.fit(A)
RES = pca.transform(A)

RES_df = pd.DataFrame(RES)
# RES_df.to_excel('api_RES.xlsx')
RES_df.columns = ['x', 'y']
x1 = RES_df['x']
y1 = RES_df['y']

# 画图
plt.scatter(x1,y1,color = 'r',marker = 'p', label = 'PCA变换')
plt.xlabel("x",fontsize = 12)
plt.ylabel("y",fontsize = 12)

```

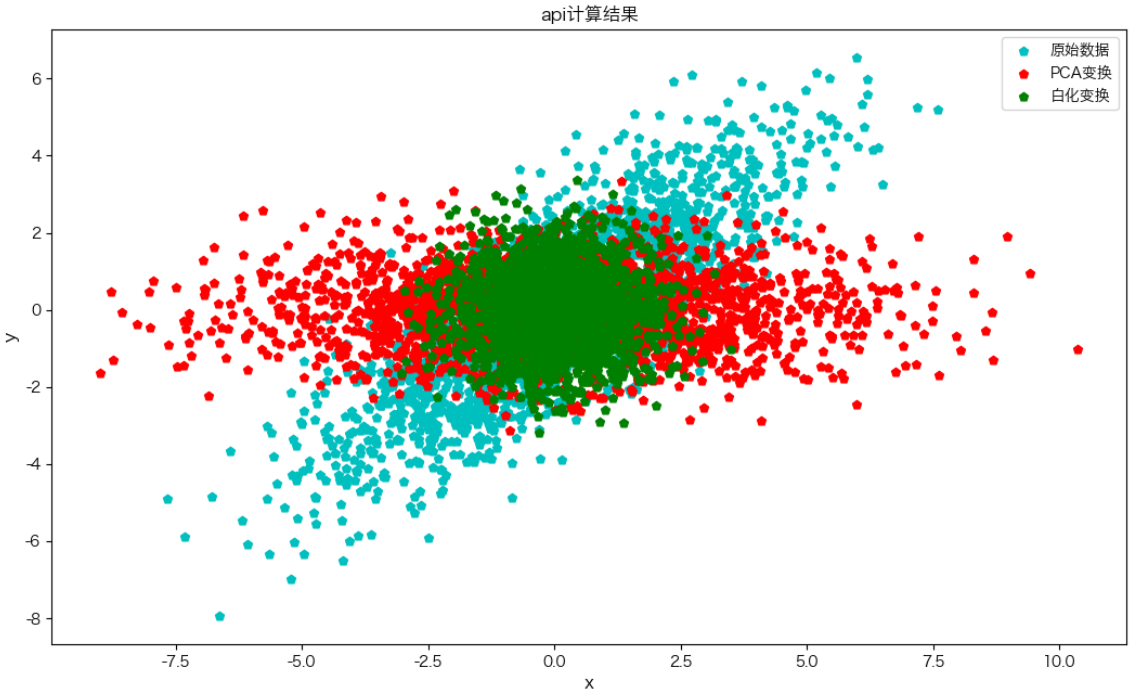
```python
#--------白化-------
pca = PCA(n_components = 2, whiten = True)
pca.fit(A)
RES = pca.transform(A)

RES_df = pd.DataFrame(RES)
# RES_df.to_excel('api_white_RES.xlsx')
RES_df.columns = ['x', 'y']
x2 = RES_df['x']
y2 = RES_df['y']

# 画图
plt.scatter(x2,y2,color = 'g',marker = 'p', label = '白化变换')
plt.xlabel("x",fontsize = 12)
plt.ylabel("y",fontsize = 12)

# 最终展示
plt.legend()
plt.title('api计算结果')
plt.show()
```

用api运行导出的图像如下



> 如图为api运行结果图像

不难看出，手写代码与api运行结果类似，结果正确

任务(d)

假设我们对所有维度应用PCA，但并未除去任何主成分，此时，我们就将数据集转化到了一个新的，与原始坐标系不同的坐标空间。但在这个新的坐标系下，数据是重新定向和重新缩放的。

这是种旋转的形式，因为我们改变了数据的方向和尺度，却并未改变其维度数。数据中同一方向的所有向量都会一起旋转，并且会根据新的主成分轴进行缩放。另外，旋转是一种保持角度和长度不变的线性变换，而这正是PCA所做的——关于角度，每对新旧基向量之间的角度都是直角；关于长度，新基向量的长度（或标准差）对应了在该主成分上的方差。

要注意的是，PCA的这种"旋转"功能，是建立在假设数据遵循线性模型的基础上的。如果你的数据具有复杂的非线性成分，那么PCA的效果可能就会降低。

这一操作将原始数据旋转至方差最大的方向，可以减少数据集的复杂性，并且可以更好地用于其他机器学习任务。

人脸识别实验

使用语言

C++

任务(a)

ORL人脸数据集共包含40个不同人的400张图像，是在1992年4月至1994年4月期间由英国剑桥的Olivetti研究实验室创建。

此数据集下包含40个目录，每个目录下有10张图像，每个目录表示一个不同的人。所有的图像是以PGM格式存储，灰度图，图像大小宽度为92，高度为112。对每一个目录下的图像，这些图像是在不同的时间、不同的光照、不同的面部表情(睁眼/闭眼，微笑/不微笑)和面部细节(戴眼镜/不戴眼镜)环境下采集的。所有的图像是在较暗的均匀背景下拍摄的，拍摄的是正脸(有些带有略微的侧偏)。

任务(b)

1. 什么是OpenCV?

   OpenCV (Open Source Computer Vision Library) 是一套开源的计算机视觉和机器学习软件库，包含超过2500种优化算法，可用于检测和识别面部，识别物体，对图像进行分类等各种复杂的执行任务。

2. 安装OpenCV

   可以使用Python pip包管理器安装Opencv：`pip install opencv-python`。具体安装过程可能会根据操作系统和环境有所不同。

3. 基本操作

   - 读取、显示和保存图像

```python
import cv2

# 读取图像
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)

# 显示图像
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# 保存图像
cv2.imwrite('new_image.jpg', img)
```

   - 视频处理

```python
import cv2

# 创建一个 VideoCapture 对象
cap = cv2.VideoCapture(0)

while True:
    # 逐帧捕获
    ret, frame = cap.read()

    # 显示结果帧
    cv2.imshow('frame', frame)
```

```
12
13          # 按'q'退出循环
14          if cv2.waitKey(1) & 0xFF == ord('q'):
15              break
16
17      # 释放捕获
18      cap.release()
19      cv2.destroyAllWindows()
```

4. 图像处理
   - 图像变换

     OpenCV提供了一系列图像变换的方法，如缩放、翻转、旋转等。
   - 颜色空间转换

     在OpenCV中，可以进行颜色空间的转换，如RGB到灰度（grayscale）、RGB到HSV等。
   - 图像阈值

     阈值是一种简单且效果良好的图像分割方法。主要的思想是把图像分割成两个部分，即背景和前景。
   - 滤波

     滤波是一种常用于图像处理的方法，用于去噪、锐化、模糊等。
5. 特征检测和描述
   - 边缘检测

     边缘检测是计算机视觉中最常用的技术之一，应用于图像分割和数据提取等任务。Canny边缘检测器是一种广泛使用的边缘检测算法。
   - 角点检测

     角点检测是检测图像中的角点，角点是图像中局部特征的重要信息，可以帮助完成一些任务如匹配跟踪等。
   - 描述子

     描述子是用于表达图像中局部特征的向量，如SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features)等。
6. 目标检测与跟踪

   使用特征匹配，我们可以完成像物体识别和跟踪这样的任务。使用训练好的分类器，我们也可以识别特定的物体，例如人脸等。
7. 机器学习在OpenCV中的应用

OpenCV提供了一些机器学习的方法与接口，如KNN、SVM、决策树等，可以用于分类、回归和聚类等任务。

任务(c)

Eigenfaces

实现代码

```
#include "opencv2/core.hpp"
#include "opencv2/face.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <fstream>
#include <sstream>
using namespace cv;
using namespace cv::face;
using namespace std;
static Mat norm_0_255(InputArray _src)
{
    Mat src = _src.getMat();
    // Create and return normalized image:
    Mat dst;
    switch (src.channels())
    {
    case 1:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC1);
        break;
    case 3:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC3);
        break;
    default:
        src.copyTo(dst);
        break;
    }
    return dst;
}
static void read_csv(const string &filename, vector<Mat> &images,
vector<int> &labels, char separator = ';')
{
```

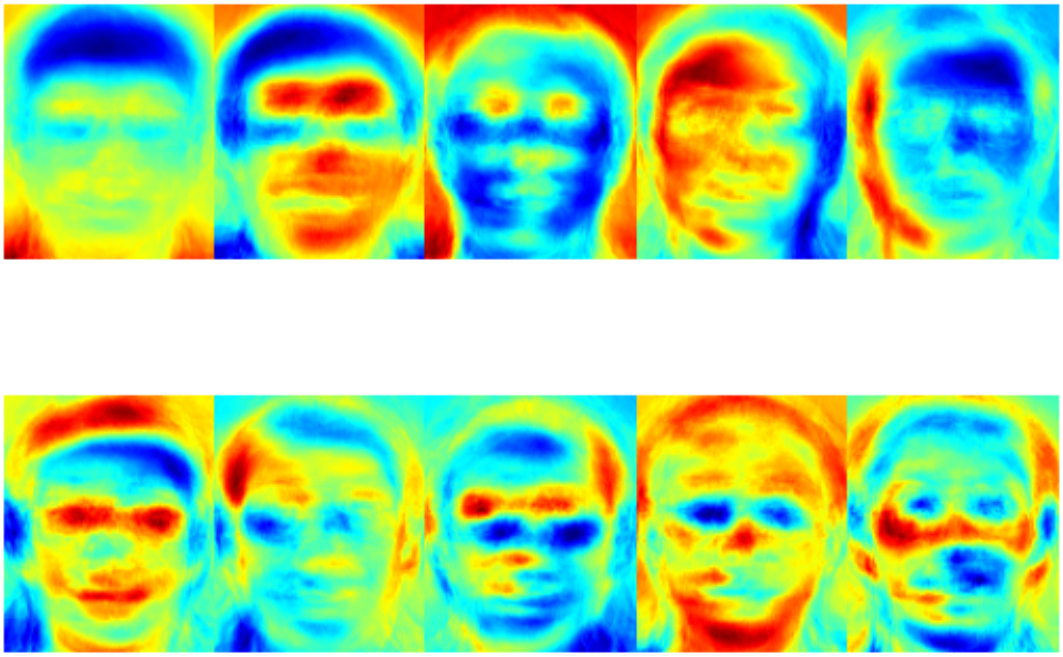
```
32      std::ifstream file(filename.c_str(), ifstream::in);
33      if (!file)
34      {
35          string error_message = "No valid input file was given, please
    check the given filename.";
36          CV_Error(Error::StsBadArg, error_message);
37      }
38      string line, path, classlabel;
39      while (getline(file, line))
40      {
41          stringstream liness(line);
42          getline(liness, path, separator);
43          getline(liness, classlabel);
44          if (!path.empty() && !classlabel.empty())
45          {
46              images.push_back(imread(path, 0));
47              labels.push_back(atoi(classlabel.c_str()));
48          }
49      }
50  }
51  int main(int argc, const char *argv[])
52  {
53      // // Check for valid command line arguments, print usage
54      // // if no arguments were given.
55      // if (argc < 2)
56      // {
57      //     cout << "usage: " << argv[0] << " <csv.ext> <output_folder> "
    << endl;
58      //     exit(1);
59      // }
60      string output_folder = "./OUTPUT";
61      // if (argc == 3)
62      // {
63      //     output_folder = string(argv[2]);
64      // }
65
66      // Get the path to your CSV.
67      string fn_csv = "./in.csv";
68
69      // These vectors hold the images and corresponding labels.
70      vector<Mat> images;
71      vector<int> labels;
```

```
72
73      // Read in the data. This can fail if no valid
74      // input filename is given.
75      try
76      {
77          read_csv(fn_csv, images, labels);
78      }
79      catch (const cv::Exception &e)
80      {
81          cerr << "Error opening file \"" << fn_csv << "\". Reason: " <<
    e.msg << endl;
82          // nothing more we can do
83          exit(1);
84      }
85      // Quit if there are not enough images for this demo.
86      if (images.size() <= 1)
87      {
88          string error_message = "This demo needs at least 2 images to
    work. Please add more images to your data set!";
89          CV_Error(Error::StsError, error_message);
90      }
91
92      // Get the height from the first image. We'll need this
93      // later in code to reshape the images to their original
94      // size:
95      int height = images[0].rows;
96
97      // The following lines simply get the last images from
98      // your dataset and remove it from the vector. This is
99      // done, so that the training data (which we learn the
100     // cv::BasicFaceRecognizer on) and the test data we test
101     // the model with, do not overlap.
102     Mat testSample = images[images.size() - 1];
103     int testLabel = labels[labels.size() - 1];
104     images.pop_back();
105     labels.pop_back();
106
107     // The following lines create an Eigenfaces model for
108     // face recognition and train it with the images and
109     // labels read from the given CSV file.
110     // This here is a full PCA, if you just want to keep
111     // 10 principal components (read Eigenfaces), then call
```

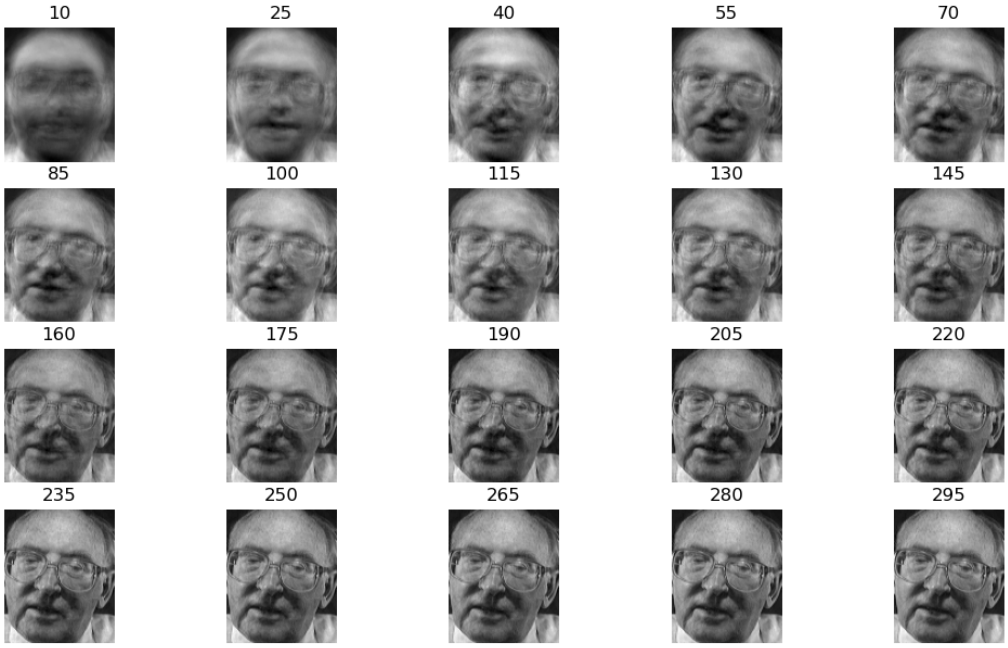
```
112     // the factory method like this:
113     //
114     //      EigenFaceRecognizer::create(10);
115     //
116     // If you want to create a FaceRecognizer with a
117     // confidence threshold (e.g. 123.0), call it with:
118     //
119     //      EigenFaceRecognizer::create(10, 123.0);
120     //
121     // If you want to use _all_ Eigenfaces and have a threshold,
122     // then call the method like this:
123     //
124     //      EigenFaceRecognizer::create(0, 123.0);
125     //
126     Ptr<EigenFaceRecognizer> model = EigenFaceRecognizer::create();
127     model->train(images, labels);
128     // The following line predicts the label of a given
129     // test image:
130     int predictedLabel = model->predict(testSample);
131     //
132     // To get the confidence of a prediction call the model with:
133     //
134     //      int predictedLabel = -1;
135     //      double confidence = 0.0;
136     //      model->predict(testSample, predictedLabel, confidence);
137     //
138     string result_message = format("Predicted class = %d / Actual class =
    %d.", predictedLabel, testLabel);
139     cout << result_message << endl;
140     // Here is how to get the eigenvalues of this Eigenfaces model:
141     Mat eigenvalues = model->getEigenValues();
142     // And we can do the same to display the Eigenvectors (read
    Eigenfaces):
143     Mat W = model->getEigenVectors();
144     // Get the sample mean from the training data
145     Mat mean = model->getMean();
146     // Display or save:
147     imshow("mean", norm_0_255(mean.reshape(1, images[0].rows)));
148     imwrite(format("%s/mean.png", output_folder.c_str()),
    norm_0_255(mean.reshape(1, images[0].rows)));
149     // Display or save the Eigenfaces:
150     for (int i = 0; i < min(10, W.cols); i++)
```

```
151     {
152         string msg = format("Eigenvalue #%d = %.5f", i,
    eigenvalues.at<double>(i));
153         cout << msg << endl;
154         // get eigenvector #i
155         Mat ev = W.col(i).clone();
156         // Reshape to original size & normalize to [0...255] for imshow.
157         Mat grayscale = norm_0_255(ev.reshape(1, height));
158         // Show the image & apply a Jet colormap for better sensing.
159         Mat cgrayscale;
160         applyColorMap(grayscale, cgrayscale, COLORMAP_JET);
161         // Display or save:
162         imshow(format("eigenface_%d", i), cgrayscale);
163         imwrite(format("%s/eigenface_%d.png", output_folder.c_str(), i),
    norm_0_255(cgrayscale));
164     }
165     // Display or save the image reconstruction at some predefined steps:
166     for (int num_components = min(W.cols, 10); num_components <
    min(W.cols, 300); num_components += 15)
167     {
168         // slice the eigenvectors from the model
169         Mat evs = Mat(W, Range::all(), Range(0, num_components));
170         Mat projection = LDA::subspaceProject(evs, mean,
    images[0].reshape(1, 1));
171         Mat reconstruction = LDA::subspaceReconstruct(evs, mean,
    projection);
172         // Normalize the result:
173         reconstruction = norm_0_255(reconstruction.reshape(1,
    images[0].rows));
174         // Display or save:
175         imshow(format("eigenface_reconstruction_%d", num_components),
    reconstruction);
176         imwrite(format("%s/eigenface_reconstruction_%d.png",
    output_folder.c_str(), num_components), reconstruction);
177     }
178     // Display if we are not writing to an output folder:
179     waitKey(0);
180     return 0;
181 }
```

导出图像



> 如图为Eigenface模型生成的灰度图



> 如图为根据Eigenface模型重构的图像 从左到右 从上到下所选取的主成分逐渐增加 可见图片逐渐接近原图

Fisherfaces

实现代码

```
#include "opencv2/core.hpp"
#include "opencv2/face.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <fstream>
#include <sstream>
using namespace cv;
using namespace cv::face;
using namespace std;
static Mat norm_0_255(InputArray _src)
{
    Mat src = _src.getMat();
    // Create and return normalized image:
    Mat dst;
```

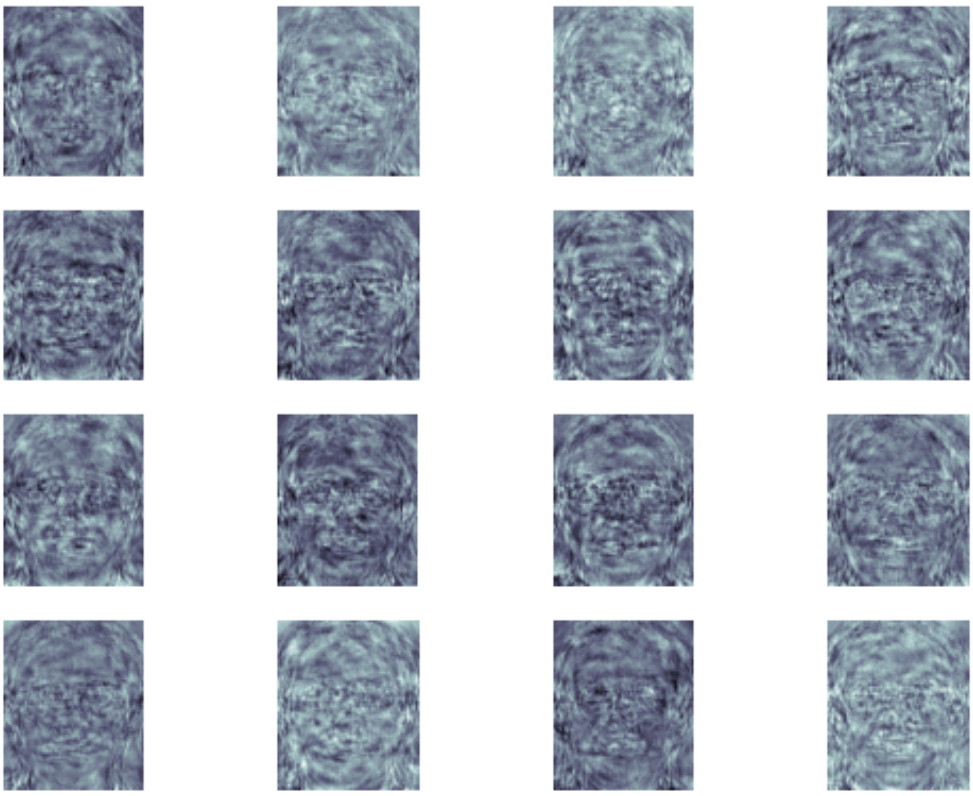
```
    switch (src.channels())
    {
    case 1:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC1);
        break;
    case 3:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC3);
        break;
    default:
        src.copyTo(dst);
        break;
    }
    return dst;
}
static void read_csv(const string &filename, vector<Mat> &images,
vector<int> &labels, char separator = ';')
{
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file)
    {
        string error_message = "No valid input file was given, please
check the given filename.";
        CV_Error(Error::StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line))
    {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if (!path.empty() && !classlabel.empty())
        {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}
int main(int argc, const char *argv[])
{
    string output_folder = "./OUTPUT";
    // Get the path to your CSV.
    string fn_csv = "./in.csv";
```

```
    // These vectors hold the images and corresponding labels.
    vector<Mat> images;
    vector<int> labels;
    // Read in the data. This can fail if no valid
    // input filename is given.
    try
    {
        read_csv(fn_csv, images, labels);
    }
    catch (const cv::Exception &e)
    {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg << endl;
        // nothing more we can do
        exit(1);
    }
    // Quit if there are not enough images for this demo.
    if (images.size() <= 1)
    {
        string error_message = "This demo needs at least 2 images to work. Please add more images to your data set!";
        CV_Error(Error::StsError, error_message);
    }
    // Get the height from the first image. We'll need this
    // later in code to reshape the images to their original
    // size:
    int height = images[0].rows;
    // The following lines simply get the last images from
    // your dataset and remove it from the vector. This is
    // done, so that the training data (which we learn the
    // cv::BasicFaceRecognizer on) and the test data we test
    // the model with, do not overlap.
    Mat testSample = images[images.size() - 1];
    int testLabel = labels[labels.size() - 1];
    images.pop_back();
    labels.pop_back();
    // The following lines create an Fisherfaces model for
    // face recognition and train it with the images and
    // labels read from the given CSV file.
    // If you just want to keep 10 Fisherfaces, then call
    // the factory method like this:
    //
```

```
96      //      FisherFaceRecognizer::create(10);
97      //
98      // However it is not useful to discard Fisherfaces! Please
99      // always try to use _all_ available Fisherfaces for
100     // classification.
101     //
102     // If you want to create a FaceRecognizer with a
103     // confidence threshold (e.g. 123.0) and use _all_
104     // Fisherfaces, then call it with:
105     //
106     //      FisherFaceRecognizer::create(0, 123.0);
107     //
108     Ptr<FisherFaceRecognizer> model = FisherFaceRecognizer::create();
109     model->train(images, labels);
110     // The following line predicts the label of a given
111     // test image:
112     int predictedLabel = model->predict(testSample);
113     //
114     // To get the confidence of a prediction call the model with:
115     //
116     //      int predictedLabel = -1;
117     //      double confidence = 0.0;
118     //      model->predict(testSample, predictedLabel, confidence);
119     //
120     string result_message = format("Predicted class = %d / Actual class =
    %d.", predictedLabel, testLabel);
121     cout << result_message << endl;
122     // Here is how to get the eigenvalues of this Eigenfaces model:
123     Mat eigenvalues = model->getEigenValues();
124     // And we can do the same to display the Eigenvectors (read
    Eigenfaces):
125     Mat W = model->getEigenVectors();
126     // Get the sample mean from the training data
127     Mat mean = model->getMean();
128     // Display or save:
129     imshow("mean", norm_0_255(mean.reshape(1, images[0].rows)));
130     imwrite(format("%s/mean.png", output_folder.c_str()),
    norm_0_255(mean.reshape(1, images[0].rows)));
131     // Display or save the first, at most 16 Fisherfaces:
132     for (int i = 0; i < min(16, W.cols); i++)
133     {
```

```
        string msg = format("Eigenvalue #%d = %.5f", i, eigenvalues.at<double>(i));
        cout << msg << endl;
        // get eigenvector #i
        Mat ev = W.col(i).clone();
        // Reshape to original size & normalize to [0...255] for imshow.
        Mat grayscale = norm_0_255(ev.reshape(1, height));
        // Show the image & apply a Bone colormap for better sensing.
        Mat cgrayscale;
        applyColorMap(grayscale, cgrayscale, COLORMAP_BONE);
        // Display or save:
        imshow(format("fisherface_%d", i), cgrayscale);
        imwrite(format("%s/fisherface_%d.png", output_folder.c_str(), i), norm_0_255(cgrayscale));
    }
    // Display or save the image reconstruction at some predefined steps:
    for (int num_component = 0; num_component < min(16, W.cols); num_component++)
    {
        // Slice the Fisherface from the model:
        Mat ev = W.col(num_component);
        Mat projection = LDA::subspaceProject(ev, mean, images[0].reshape(1, 1));
        Mat reconstruction = LDA::subspaceReconstruct(ev, mean, projection);
        // Normalize the result:
        reconstruction = norm_0_255(reconstruction.reshape(1, images[0].rows));
        // Display or save:
        imshow(format("fisherface_reconstruction_%d", num_component), reconstruction);
        imwrite(format("%s/fisherface_reconstruction_%d.png", output_folder.c_str(), num_component), reconstruction);
    }
    // Display if we are not writing to an output folder:
    waitKey(0);
    return 0;
}
```

导出图像



> 如图为Fisherface模型生成的灰度图

> 如图为根据Fisherface模型重构的图像

结果分析

- Eigenfaces不仅编码了面部特征，还编码了图像的光源，因而重构图可以显示得相当清晰。但PCA方法丢失了大量的类间判别信息，使分类变得困难。
- Fisherfaces最大化了类与类之间的分散比，而不是最大化总体分散，但不像Eigenfaces方法一样明显地捕获光照，而是通过面部特征来区分不同的人，性能在很大程度上依赖于数据。由于只识别了区分主题的特征，不能使原始图像得到很好的重建。

任务(d)

从Eigenface重构的图像来看，当选取eigenfaces的数量达到220时，重构的图片已经相当接近原图