实验四：人脸 PCA 识别

学号：2018329621262　　姓名：刘恒玮

一、实验目的

理解 PCA 降维原理，使用 PCA 解决实际问题。

二、实验环境

PC 机，Python，numpy 库，sklearn 库。

三、实验内容

给定人脸数据，利用 PCA 技术对人脸进行降维，并对降维后数据使用 SVM 或 MLP 进行分类。

四、实验原理（算法）

PCA 降维：

降维就是一种对高维度特征数据预处理方法。降维是将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。降维具有如下一些优点：使得数据集更易使用、降低算法的计算开销、去除噪声、使得结果容易理解。PCA(Principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。PCA 的工作就是从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。其中，第一个新坐标轴选择是原始数据中方差最大的方向，第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第 1,2 个轴正交的平面中方差最大的。依次类推，可以得到 n 个这样的坐标轴。通过这种方式获得的新的坐标轴，我们发现，大部分方差都包含在前面 k 个坐标轴中，后面的坐标轴所含的方差几乎为 0。于是，我们可以忽略余下的坐标轴，只保留前面 k 个含有绝大部分方差的坐标轴。事实上，这相当于只保留包含绝大部分方差的维度特征，而忽略包含方差几乎为 0 的特征维度，实现对数据特征的降维处理。

总结一下 PCA 的算法步骤：设有 m 条 n 维数据。将原始数据按列组成 n 行 m 列矩阵 X；将 X 的每一行（代表一个属性字段）进行零均值化，即减去这一行的均值；求出协方差矩阵；求出协方差矩阵的特征值及对应的特征向量；将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，即为降维到 k 维后的数据。

特征脸 EigenFace：

特征脸 EigenFace 的思想是把人脸从像素空间变换到另一个空间，在另一个空间中做相似性的计算。EigenFace 选择的空间变换方法是 PCA，也就是主成分分析。它广泛的被用于预处理中以消去样本特征维度之间的相关性。EigenFace 方法利用 PCA 得到人脸分布的主要成分，具体实现是对训练集中所有人脸图像的协方差矩阵进行本征值分解，得对对应的本征向量，这些本征向量（特征向量）就是“特征脸”。每个特征向量或者特征脸相当于捕捉或者描述人脸之间的一种变化或者特性，这就意味着每个人脸都可以表示为这些特征脸的线性组合。

五、实验步骤（分析过程）

1. 导入实验过程中所需要的相关库。

[1]:
```
import matplotlib.pyplot as plt
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn import metrics
from  PIL import Image
from sklearn.neural_network import MLPClassifier
```

2. 将所有图片的路径都存到一个 list 中，便于之后的数据读取和处理。

[2]:
```
def listdir(path, list_name):  # 传入存储的list
    for file in os.listdir(path):
        file_path = os.path.join(path, file)
        if os.path.isdir(file_path):
            listdir(file_path, list_name)
        else:
            list_name.append(file_path)
path='D:\Desktop\Study\大三上\机器学习\实验四 人脸PCA识别\第四次实验数据准备'
list_name = []
listdir(path,list_name)
```

[3]:
```
print(len(list_name))
list_name
```

```
246
```

[3]:
```
['D:\\Desktop\\Study\\大三上\\机器学习\\实验四 人脸PCA识别\\第四次实验数据准备\\2018327100030_杨栎\\01-2018327100030_yangyue_0.jpg',
 'D:\\Desktop\\Study\\大三上\\机器学习\\实验四 人脸PCA识别\\第四次实验数据准备\\2018327100030_杨栎\\02-2018327100030_yangyue_1.jpg',
 'D:\\Desktop\\Study\\大三上\\机器学习\\实验四 人脸PCA识别\\第四次实验数据准备\\2018327100030_杨栎\\03-2018327100030_yangyue_2.jpg',
 'D:\\Desktop\\Study\\大三上\\机器学习\\实验四 人脸PCA识别\\第四次实验数据准备\\2018327100146_潘俊安\\01-2018327100146_pja_0.jpg',
 'D:\\Desktop\\Study\\大三上\\机器学习\\实验四 人脸PCA识别\\第四次实验数据准备\\2018327100146_潘俊安\\02-2018327100146_pja_1.jpg',
```

发现总共是 246 张图，即每个人三张，之后选取前两张作为训练集，第三张作为验证集。

3. 将数据集中的每一张图片转为 128*128 的灰度图，并将图像矩阵从 128*128 转为 1*16384，即把每一张人脸转为 16384 个特征值存到 txt 的一行中，并在行末加上对应的姓名作为 label，处理好之后分别将训练集和验证集的数据分别保存到 train_data.txt 和 test_data.txt 中。

[9]:
```
for name in list_name:
    x = name.split("_")
    if(str(x[-1][0])!='2'):
        img_train = np.array(Image.open(name).resize((128,128)).convert('L'))
        h, w = img_train.shape
        img_train = np.reshape(img_train,(1,h*w))
        h, w = img_train.shape
        img_train = img_train.tolist()
        with open('train_data.txt','a') as f:
            for i in range(len(img_train[0])):
                f.write(str(img_train[0][i])+' ')
            f.write(str(x[-2])+'\n')
    else:
        img_test = np.array(Image.open(name).resize((128,128)).convert('L'))
        h, w = img_test.shape
        img_test = np.reshape(img_test,(1,h*w))
        h, w = img_test.shape
        img_test = img_test.tolist()
        with open('test_data.txt','a') as f:
            for i in range(len(img_test[0])):
                f.write(str(img_test[0][i])+' ')
            f.write(str(x[-2])+'\n')
```

4. 从保存的 txt 中读取训练集和验证集，要注意的是要将特征值转换为 int 型。

```
[10]: train_data = np.loadtxt('train_data.txt',dtype = str)
      test_data = np.loadtxt('test_data.txt',dtype = str)
      train_data_x = train_data[:, :-1].astype('int64')
      train_data_y = train_data[:, -1:]

      test_data_x = test_data[:, :-1].astype('int64')
      test_data_y = test_data[:, -1:]
```

5. 对读取进来的训练集和验证集进行 pca 降维，并用 SVM 和 MLP 两种方法进行分类，分类后利用 sklearn 中的 classification_report 函数显示每个类的精确度，召回率，F1 值等信息。

```
[11]: pca = PCA(n_components = 20)
      train_x = pca.fit_transform(train_data_x)
      svm_clf = SVC()
      mlp_clf = MLPClassifier()
      svm_clf.fit(train_x, train_data_y)
      mlp_clf.fit(train_x, train_data_y)
      test_x = pca.transform(test_data_x)
      svm_y_predict = svm_clf.predict(test_x)
      mlp_y_predict = mlp_clf.predict(test_x)
      print(classification_report(test_data_y, svm_y_predict))
      print(classification_report(test_data_y, mlp_y_predict))
```

SVM 分类的各项指标：

```
              precision    recall  f1-score   support

    accuracy                       0.56        82
   macro avg       0.46      0.56      0.49        82
weighted avg       0.46      0.56      0.49        82
```

MLP 分类的各项指标：

```
              precision    recall  f1-score   support

    accuracy                       0.37        82
   macro avg       0.30      0.37      0.32        82
weighted avg       0.30      0.37      0.32        82
```

发现用 SVM 分类的 F1 值要高于利用 MLP 分类的结果，但数值还是不太理想，

决定重新利用 PCA 算法对数据进行降维，在 1-100 维中选取一个准确率最为理想的维度。

6. 重新利用 PCA 算法对数据进行降维，在 1-100 维中选取一个准确率最为理想的维度，并输出对于两种分类算法验证集最高的准确度。

[13]:
```
svc_scrore = []
mlp_scrore = []
for i in range(1,101):
    pca = PCA(n_components = i)
    train_x = pca.fit_transform(train_data_x)
    svm_clf = SVC()
    mlp_clf = MLPClassifier()
    svm_clf.fit(train_x, train_data_y)
    mlp_clf.fit(train_x, train_data_y)
    test_x = pca.transform(test_data_x)
    svm_y_predict = svm_clf.predict(test_x)
    mlp_y_predict = mlp_clf.predict(test_x)
    svc_scrore.append(metrics.accuracy_score(test_data_y, svm_y_predict))
    mlp_scrore.append(metrics.accuracy_score(test_data_y, mlp_y_predict))
```

...

[15]:
```
print("SVM验证集的准确率：",max(svc_scrore))
print("MLP验证集的准确率：",max(mlp_scrore))
```

```
SVM验证集的准确率：  0.573170731707317
MLP验证集的准确率：  0.4146341463414634
```

7. 发现实验中的准确率还是比较低，看了数据集中的人脸图像之后，发现有些同学的三张人脸图像不一致，甚至相差很大，决定将这些不好的数据集去除，重新计算验证集的准确率。

[24]:
```
print("SVM验证集的准确率：",max(svc_scrore))
print("MLP验证集的准确率：",max(mlp_scrore))
```

```
SVM验证集的准确率：  0.6153846153846154
MLP验证集的准确率：  0.44871794871794873
```

去除错误数据后准确率提高了三四个点，但准确率仍然不是很理想。

六．实验结果（训练集准确度，验证集准确度等指标）

SVM 和 MLP 分类训练集和验证集准确度：

```
SVM训练集的准确率： 0.9871794871794872 SVM验证集的准确率： 0.6153846153846154 验证集准确率最高时数据的维数 20
MLP训练集的准确率： 0.9871794871794872 MLP验证集的准确率： 0.46153846153846156 验证集准确率最高时数据的维数 15
```

七．实验心得（碰到的问题，如何解决，总结使用 PCA 与未使用 PCA 的优缺点）

使用 PCA 的优点：

通过降维可以发现更便于人理解的特征，加快对样本有价值信息的处理速度，此外还可以应用于可视化和去噪。使得数据集更易使用，降低了算法的计算开销，可以去除噪声，使得结果容易理解，并且完全无参数限制。

使用 PCA 的缺点：

主成分解释其含义往往具有一定的模糊性，不如原始样本完整，若在掌握了数据的一些特征，却无法通过参数化等方法对处理过程进行干预，可能会得不到预期的效果，效率也不高，特征值分解有一些局限性，比如变换的矩阵必须是方阵，在非高斯分布情况下，PCA 方法得出的主元可能并不是最优的。

实验心得体会：

PCA 本质上是将方差最大的方向作为主要特征，并且在各个正交方向上将数据“离相关”，也就是让它们在不同正交方向上没有相关性。PCA 也存在一些限制，例如它可以很好的解除线性相关，但是对于高阶相关性就没有办法了，对于存在高阶相关性的数据，可以考虑 Kernel PCA，通过 Kernel 函数将非线性相关转为线性相关。另外，PCA 假设数据各主特征是分布在正交方向上，如果在非正交方向上存在几个方差较大的方向，PCA 的效果就大打折扣了。