# 《计算机视觉》实验报告

姓名：严昕宇 学号：20121802

## 实验 4

## 一. 任务 1

### a) 核心代码：

# 实验4 人脸识别

## 用PCA+KNN算法实现人脸识别

(1)数据集自选；

(2)详细说明实验参数，包括参与训练和测试的图片数量，分类个数，降维维度，KNN参数等，总结各参数对准确率的影响，不断提高准确率。

[1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
# PCA算法
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
# KNN算法
from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier
from sklearn.metrics import classification_report, f1_score, recall_score
```

### 1. 导入数据，此处使用ORL人脸数据集

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(实验所给的ORL数据集中已转换为BMP格式)，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)。

[2]:
```
# 传入图片路径和需要保存的文件名
# 由于图片在不同目录下，因此读入处理时，需要特殊处理
PhotoPath = './ORL/s{}/{}.bmp'
SavedFile = 'data.txt'
```

### 2. 处理数据，将图片数据转化为二维矩阵

[3]:
```
# 标签列添加到矩阵的最后一列
LabelList = []
# 将每一行的特征向量进行堆叠,最后得到(400,10305)大小的二维特征矩阵
StackMatrix = np.array([[0]])
```

```
for i in range(1, 41):
    # 加入每张图片的标签
    LabelList.append(i)
    ClassMatrix = np.array(LabelList, ndmin=2)
    for j in range(1, 11):
        Path = PhotoPath.format(i, j)
        x = Image.open(Path)
        # 转换为narray的结构，并转为二维矩阵
        data = np.reshape(np.asarray(x), (1, -1))
        # 得到的维度是(1, 10304)，取消print注释即可观察到
        # print(data.shape)
        OneDimData = np.column_stack((data, ClassMatrix))
        # 第一次不合并
        if i == 1 and j == 1:
            StackMatrix = OneDimData
            continue
        StackMatrix = np.row_stack((StackMatrix, OneDimData))
    LabelList.pop()
np.savetxt(SavedFile, StackMatrix)
```

### 3. 加载并读入数据

[4]:
```
# 读入处理后的图片二维矩阵文件
TrainData = np.loadtxt(SavedFile)
FeatureData = TrainData[:, :10304]  # 取出特征数据
LabelData = TrainData[:, -1]  # 取出标签数据
```

### 4. 划分数据集

在sklearn.model_selection，有train_test_split函数用于将样本数据切分为训练集和测试集。

其中有如下参数：

train_data：所要划分的样本特征集

train_target：所要划分的样本结果

test_size：样本占比，如果是整数的话就是样本的数量

random_state：是随机数的种子。

在保持输入数据不变的情况下，如果 random_state 等于某个固定的值， 如42，将得到同样的数据划分；如果 random_state 等于另外某个值，将得到另外一份不同的数据划分；如果 random_state = None (默认值) ，会随机选择一个种子，这样每次都会得到不同的数据划分。

[5]:
```
x_data, y_data = FeatureData, LabelData
# 训练/测试集37开
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3,
                                                    shuffle=True, random_state=None)
```

### 5. 利用PCA将特征降至n_components维

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。

[6]:
```
pca = PCA(n_components=64)
# 训练集上进行PCA降维
x_train_pca = pca.fit_transform(x_train)
# 对测试数据进行PCA降维，保证转换矩阵相同
x_test_pca = pca.transform(x_test)
```

### 6. 训练模型，返回准确率和模型

此处使用的k-最近邻算法，也称为 KNN 或 k-NN，是一种非参数、有监督的学习分类器，它使用邻近度对单个数据点的分组进行分类或预测。 虽然它可以用于回归或分类问题，但它通常用作分类算法，假设可以在彼此附近找到相似点。

[7]:
```
Model = KNeighborsClassifier(n_neighbors=1, weights='distance')
Model.fit(x_train_pca, y_train)
```

[7]:
```
KNeighborsClassifier(n_neighbors=1, weights='distance')
```

### 7. 为了研究参数对准确率的影响，将PCA的n_components从1到50遍历，KNN的n_neighbors从1到50遍历，以得到准确率最高值。

值得注意的是，使用KNeighborsClassifier会出现FutureWarning的警告。因此，在使用KNeighborsClassifie时，需要声明weights的取值方式。

其中：weights='uniform' 相当于直接取众数，类似简单平均；weights='distance' 类似加权平均，可以简单地理解为以距离的倒数作为权重的"加权众数"。由于KNN假设距离近的样本更相似，距离远的样本不相似，因此这种方式看上去似乎更合理

[8]:
```
KList = []
KAccuracyList = []
MaxK = 0
MaxAccuracy = 0
MaxDim = 0
MaxDimAccuracy = 0

# 遍历PCA降维维数
for dim in range(1, 100):
    pca = PCA(n_components=dim)
    # 训练集上进行PCA降维
    x_train_pca = pca.fit_transform(x_train)
    # 对测试数据进行PCA降维，保证转换矩阵相同
    x_test_pca = pca.transform(x_test)

    # 遍历KNN的K值
    for k in range(1, 50):
        Model = KNeighborsClassifier(n_neighbors=k, weights='distance')
        Model.fit(x_train_pca, y_train)
        Score = Model.score(x_test_pca, y_test)
        if Score > MaxAccuracy:
            MaxAccuracy = Score
            MaxK = k

    KAccuracyList.append(MaxAccuracy * 100)
    KList.append(MaxK)
    if MaxAccuracy > MaxDimAccuracy:
        MaxDimAccuracy = MaxAccuracy
        MaxDim = dim
```

[9]:
```
print('结论：当PCA降维维数Dim取', MaxDim, ', k取', KList[MaxDim],
      '时，达到最高的准确率：{:.2f}%'.format(MaxDimAccuracy * 100))
```

```
结论：当PCA降维维数Dim取 32 , k取 1 时，达到最高的准确率：95.83%
```

### 8. 由于数据集划分的随机性，在众多尝试中，最佳记录为：当PCA降维到39维，K取1时，达到最高的准确率99.17%

## 9. 利用最佳参数，得到模型的评估指标：F1-Score与召回率

[10]:
```
pca = PCA(n_components=MaxDim)
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)
Model = KNeighborsClassifier(n_neighbors=KList[MaxDim], weights='distance')
Model.fit(x_train_pca, y_train)

y_predict = Model.predict(x_test_pca)
print(classification_report(y_test, y_predict))

# 打印出F1-Score和ReCall评估参数
print('ReCall: %.4f' % recall_score(y_test, y_predict, average='micro'))
print('F1-Score: %.4f' % f1_score(y_test, y_predict, average='micro'))
```

```
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00         4
         2.0       1.00      1.00      1.00         1
         3.0       1.00      1.00      1.00         2
         4.0       1.00      1.00      1.00         3
         5.0       0.60      1.00      0.75         3
         6.0       1.00      1.00      1.00         4
         7.0       1.00      1.00      1.00         6
         8.0       1.00      1.00      1.00         1
         9.0       1.00      1.00      1.00         3
        10.0       1.00      1.00      1.00         2
        11.0       1.00      1.00      1.00         4
        12.0       1.00      1.00      1.00         5
        13.0       1.00      1.00      1.00         4
        14.0       1.00      1.00      1.00         4
        15.0       1.00      1.00      1.00         3
        16.0       1.00      0.75      0.86         4
        17.0       1.00      1.00      1.00         4
        18.0       1.00      1.00      1.00         1
        19.0       1.00      1.00      1.00         2
        20.0       1.00      1.00      1.00         2
        21.0       0.67      1.00      0.80         2
        22.0       1.00      1.00      1.00         3
        23.0       1.00      1.00      1.00         2
        24.0       1.00      1.00      1.00         4
        25.0       1.00      1.00      1.00         1
        26.0       1.00      1.00      1.00         4
        27.0       0.67      1.00      0.80         2
        28.0       1.00      1.00      1.00         2
        29.0       1.00      1.00      1.00         1
        30.0       1.00      1.00      1.00         4
        31.0       1.00      0.75      0.86         4
        32.0       1.00      1.00      1.00         3
        33.0       1.00      1.00      1.00         2
        34.0       1.00      1.00      1.00         6
        35.0       1.00      0.75      0.86         4
        36.0       1.00      1.00      1.00         2
        37.0       1.00      1.00      1.00         3
        38.0       1.00      1.00      1.00         2
        39.0       1.00      1.00      1.00         2
        40.0       0.75      0.60      0.67         5

    accuracy                           0.96       120
   macro avg       0.97      0.97      0.96       120
weighted avg       0.97      0.96      0.96       120

ReCall: 0.9583
F1-Score: 0.9583
```

### 10. 混淆矩阵 Confusion Matrix

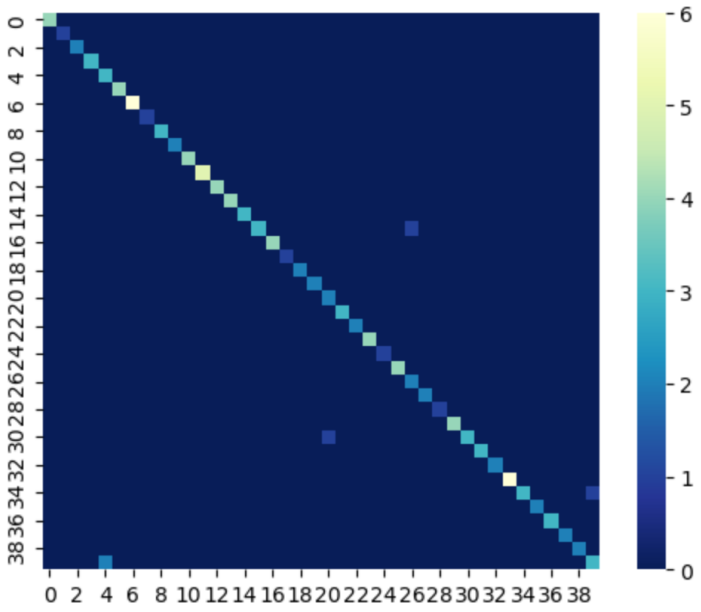
混淆矩阵是机器学习中总结分类模型预测结果的情形分析表，以矩阵形式将数据集中的记录按照真实的类别与分类模型预测的类别判断两个标准进行汇总。

此处使用的Seaborn，是基于Python且非常受欢迎的图形可视化库，在Matplotlib 的基础上，其进行了更高级的封装，使得作图更加方便快捷。

[11]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, y_predict)
sns.heatmap(cm, cmap="YlGnBu_r", fmt="d", square=True)
```

[11]: <AxesSubplot:>




**b) 实验小结**

本项目采用了 PCA 数据降维和 KNN 分类模型，最终证明了模型效果较为良好。

在实验过程中，需要注意以下几点内容：

1) 训练集大小和测试集大小对预测率并没有特别确定的关系，在实验中，通常选用 55 开或 37 开。但在其它条件不变的情况下，当噪声比较小，训练集的标签（分类数）足够多且测试集的标签与训练集的标签交集较大时，模型预测率是相对较高的。

2) 其它条件不变的情况下，当 PCA 降维维度能保证原始数据损失量最小且最能保证模型性能的情况下，模型预测性能最优。而且，在使用 PCA 的过程中，应该使用训练数据得到的转换矩阵，对测试数据进行转换，而不是重新进行 PCA。

3) 其它条件不变的情况下，对于 KNN 模型，在本实验中 n_neighbors 为 1 时，模型预测性能最优。