机器学习人脸识别实验报告:

特征脸方法

李梓铉

学号 3118103163

指导老师: 李慧斌教授

2018 年 12 月 30 日

目录

1 实验背景 **2**

2 实验设计 **4**

3 实验结果 **5**

4 结论与讨论 **7**

5 附录 **8**

1 实验背景

人脸识别技术是基于人的脸部特征，对输入的人脸图像或者视频流. 首先判断其是否存在人脸, 如果存在人脸，则进一步的给出每个脸的位置、大小和各个主要面部器官的位置信息。并依据这些信息，进一步提取每个人脸中所蕴涵的身份特征，并将其与已知的人脸进行对比，从而识别每个人脸的身份。

目前主要的人脸识别的方法主要分为五类：

第一类，基于几何特征的方法。人脸由眼睛、鼻子、嘴巴、下巴等部件构成，正因为这些部件的形状、大小和结构上的各种差异才使得世界上每个人脸千差万别，因此对这些部件的形状和结构关系的几何描述，可以做为人脸识别的重要特征。几何特征最早是用于人脸侧面轮廓的描述与识别，首先根据侧面轮廓曲线确定若干显著点，并由这些显著点导出一组用于识别的特征度量如距离、角度等。Jia 等由正面灰度图中线附近的积分投影模拟侧面轮廓图是一种很有新意的方法。

第二类方法就是局部特征分析方法（Local Face Analysis），主元子空间的表示是紧凑的，特征维数大大降低，但它是非局部化的，其核函数的支集扩展在整个坐标空间中，同时它是非拓扑的，某个轴投影后临近的点与原图像空间中点的临近性没有任何关系，而局部性和拓扑性对模式分析和分割是理想的特性，似乎这更符合神经信息处理的机制，因此寻找具有这种特性的表达十分重要。基于这种考虑，Atick 提出基于局部特征的人脸特征提取与识别方法。

第三类方法是特征脸方法（Eigenface 或 PCA）。该方法是 90 年代初期由 Turk 和 Pentland 提出的目前最流行的算法之一，具有简单有效的特点, 也称为基于主成分分析 (principal component analysis, 简称 PCA) 的人脸识别方法。特征子脸技术的基本思想是：从统计的观点，寻找人脸图像分布的基本元素，即人脸图像样本集协方差矩阵的特征向量，以此近似地表征人脸图像。这些特征向量称为特征脸 (Eigenface)。

第四类方法为人工神经网络方法（Neural Networks）。是一种非线性动力学系统，具有良好的自组织、自适应能力。目前神经网络方法在人脸识别中的研究方兴未艾。Valentin 提出一种方法，首先提取人脸的 50 个主元，然后用自相关神经网络将它映射到 5 维空间中，再用一个普通的多层感知器进行判别，对一些简单的测试图像效果较好；Intrator 等提出了一种混合型神经网络来进行人脸识别，其中非监督神经网络用于特征提取，而监督神经网络用于分类。Lee 等将人脸的特点用六条规则描述，然后根据这六条规则进行五官的定位，将五官之间的几何距离输入模糊神经网络进行识别，效果较一般的基于欧氏距离的方法有较大改善，Laurence 等采用卷积神经网络方法进行人脸识别，由于卷积神经网络中集成了相邻像素之间的相关性知识，从而在一定程度上获得了对图像平移、旋转和局部变形的不变性，因此得到非常理想的识别结果，Lin 等提出了基于概率决策的神经网络方法 (PDBNN), 其主要思想是采用虚拟 (正反例) 样本进行强化和反强化学习，从而得到较为理想的概率估计结果，并采用模块化的网络结构 (OCON) 加快网络的学习。这种方法在人脸检测、人脸定位和人脸识别的各个步骤上都得到了较好的应用。

在本文中，将使用较为简单的特征脸方法，先用主成分分析（PCA）对人脸数据集进行降维，得到数个人脸特征向量。对于任意一个人脸样本，将样本数据向特征向量投影，得到的投影系数作为人脸的

特征表示。使用支持向量机（SVM）对这些不同的投影系数向量分类，来进行人脸识别。主要包含以下部分

- 实验设计
- 实验结果
- 结论与讨论
- 附录

使用 python 语言进行实验，相关的编译环境如下:

- 操作系统 Windows 10
- python3.6 编译器
- scikit-learn 0.20.1 （机器学习 python 库）
- numpy （python 数组处理库）
- matplotlib（python 可视化库）

2 实验设计

本文使用了英国剑桥大学的 AT&T 人脸数据：AT&T 数据集下载. 该数据集大小不到 5M，有 40 类样本，每类中包含同一个人的 10 张图像（112*92）。

实验的主要过程可以总结如下：

1. 读入数据集
2. 每一幅图像拉成一列，组成数据集合（112*92,400），并保存每一列数据对应的人脸标号，以及原图的高度和宽度。
3. 进行数据集的划分，训练集与测试集（本文使用四分之三的数据用于训练，四分之一数据用于测试）。
4. PCA 获得特征脸。
5. 使用交叉检验（cross-validation）选择 SVM 的参数，训练 SVM 分类器
6. 获得分类结果

降维后的低维空间的维数由使用者决定，可以通过选不同的维数通过交叉验证得到较好的维数 n，显然在 PCA 过程中，较小的特征值对应的特征向量对应的信息被舍弃了，这是降维的必要结果，因为较小的特征值对应的信息往往是无用的噪声信息，舍弃后可以使样本采样密度增大，一定程度上起到去噪的效果。显然 n 会影响分类的精确程度。本文中 n 为 30.

3 实验结果

前 11 个特征脸可视化结果如下：

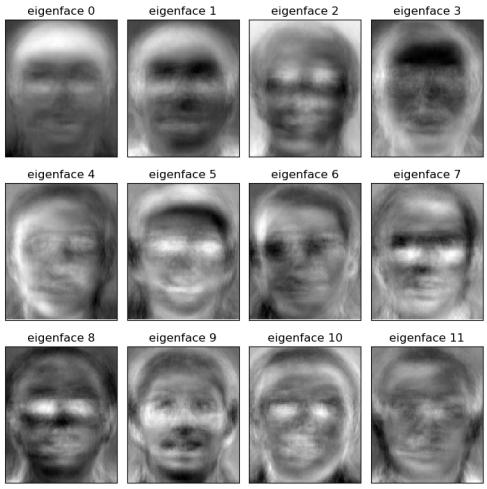

图 1: 前 11 个特征脸

部分分类结果如下：

图 2: 识别精确度

可以看出，该方法对于 2D 图像的人脸识别看可以达到较高的精度，图中仅有第三行第三列的人脸识别错误。

4 结论与讨论

实验的目的是通过使用代码，了解如何使用机器学习方法实现人脸识别，熟悉相关的编程环境以及调用方式。底层的算法实现并不是本实验的要求。

本文通过使用 Sci-kit learn python 库实现了人脸识别，尽管没有从最底层的代码进行实现，但是经过实验实现了三种降维算法的应用。通过实际的实验，对于特征脸的意义通过可视化有了更加直观的认识。对于算法原理也有了更深刻的认识。如果之后有时间，会花更多时间，从底层代码完成多种机器学习算法，而不是单纯调库。

报告写作过程中参考了周志华《机器学习》, 以及《Hands on machine learning with sci-kit learn》。代码 demo 部分来源于 scikit-learn 网站。

最后感谢李教授这学期的教学，只是有些可惜课时太少，很希望未来机器学习这门课可以调整成全周的课程，顺颂时祺。

5 附录

实验使用的源代码如下.

```
from __future__ import print_function

from time import time
import logging
import matplotlib.pyplot as plt
import cv2

from numpy import *
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC

PICTURE_PATH = "D:\\github_rep\\ML_rdtree\\face\\att_faces"


def get_Image():
    for i in range(1, 41):
        for j in range(1, 11):
            path = PICTURE_PATH + "\\s" + str(i) + "\\" + str(j) + ".pgm"
            img = cv2.imread(path)
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            h, w = img_gray.shape
            img_col = img_gray.reshape(h * w)
            all_data_set.append(img_col)
            all_data_label.append(i)
    return h, w


all_data_set = []
all_data_label = []
h, w = get_Image()

X = array(all_data_set)
y = array(all_data_label)
n_samples, n_features = X.shape
n_classes = len(unique(y))
target_names = []
for i in range(1, 41):
    names = "person" + str(i)
    target_names.append(names)

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)

# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

n_comp = 40
print("Extracting the top %d eigenfaces from %d faces"
      % (n_comp, X_train.shape[0]))

t0 = time()
pca = PCA(n_components=n_comp, svd_solver='randomized',
          whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((n_comp, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
```

```
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))

print("Fitting the classifier to the training set")
t0 = time()
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)

print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=target_names))
print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))


def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())


# plot the result of the prediction on a portion of the test set

def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i] - 1]
    true_name = target_names[y_test[i] - 1]
    return 'predicted: %s\ntrue:      %s' % (pred_name, true_name)


prediction_titles = [title(y_pred, y_test, target_names, i)
                     for i in range(y_pred.shape[0])]

plot_gallery(X_test, prediction_titles, h, w)

# plot the gallery of the most significative eigenfaces

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)

plt.show()
```