山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

学号：	姓名：	班级：
实验题目：Experiment 7: PCA in Face Recognition		
实验学时：4	实验日期： 2022/11/30	

实验目的：

1. 实现实验指导书中 PCA 的相关内容；
2. 学习使用 MATLAB、Python 等工具进行实验;
3. 通过 PCA 对面部图像进行特征提取，并通过多分类 SVM 实现面部识别。

硬件环境：

Inter（R） Core（TM） i7-8750H

RAM：16.0 GB

软件环境：

Visual Studio Code

版本: 1.67.2 (user setup)

OS: Windows_NT x64 10.0.19044

Python 3.9.7

numpy 1.20.3

matplotlib 3.4.3

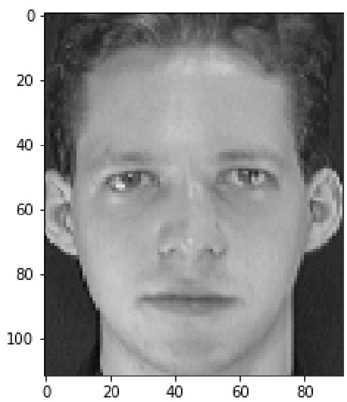
实验步骤与内容：

1. 与上一实验类似，首先使用 skimage.io.imread 对图像数据进行读取：

```
class_num, image_num = 40, 10
orl_face = []
for i in range(1, class_num+1):
    temp = []
    for j in range(1, image_num+1):
        temp.append(imread('./orl_faces/s{}/{}.pgm'.format(i, j)))
    orl_face.append(temp)
```

通过 imshow 显示其中一个图像：

```
imshow(orl_face[0][0])
```



2. 由于每个图像的数据为二维数组，要进行降维，首先需要将其展平为一维数组：

```
train_data, train_label = [], []
test_data, test_label = [], []

for i in range(len(orl_face)):
    num = random.randint(5, 7)
    train = random.sample(orl_face[i], num)
    train_data = train_data + train
    train_label = train_label + [i+1 for _ in range(num)]

    temp = []
    for j in range(len(orl_face[i])):
        flag = True
        for k in range(len(train)):
            if (orl_face[i][j] == train[k]).all():
                flag = False
                break
        if flag:
            temp.append(orl_face[i][j])

    test_data = test_data + temp
    test_label = test_label + [i+1 for _ in range(10-num)]
```

得到训练集、测试集及其标签：

```
train_data = np.array(train_data)
train_data = train_data.reshape(train_data.shape[0], -1).transpose()
train_label = np.array(train_label)
test_data = np.array(test_data)
test_data = test_data.reshape(test_data.shape[0], -1).transpose()
test_label = np.array(test_label)
```

3. 对于训练数据和测试数据，首先减去均值，做中心化：

Center the data (subtract the mean $\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$ from each data point)

```
train_data = train_data - train_data.mean(axis=1).reshape(-1, 1)
test_data = test_data - test_data.mean(axis=1).reshape(-1, 1)
```

4. 计算训练集的协方差矩阵：

$$S = \frac{1}{N}\sum_{i=1}^{N} x^{(i)} x^{(i)T} = \frac{1}{N} X X^T$$

```
N = train_data.shape[1]
S = np.dot(train_data, train_data.T) / N
```

5. 使用 numpy.linalg.eig 方法求解协方差矩阵的特征值及对应的特征向量：

```
W0, V0 = np.linalg.eig(S)
```

由于求取结果中含有复数，使用 numpy.real 方法去除虚部，保留实部：

```
W, V = np.real(W0), np.real(V0)
# W, V = W0, V0
```

将特征向量按照对应特征值从大到小的顺序进行排序，得到对应的二维矩阵 V：

```
index = (-W).argsort()
W = W[index]
V = V[:, index]
```

6. 构建之前实验中实现的 SVM：

计算海森矩阵：

```
def get_H(x, y):
    m, n = x.shape
    P = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            P[i, j] = np.dot(x[i], x[j]) * y[i] * y[j]
    return P
```

通过使用 solve_qp 库调用 cvxopt，求解对偶问题：

```
def solve_dual(x, y):
    m, n = x.shape
    P = get_H(x, y)
    q = np.ones(m) * (-1)
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = None

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    w = get_w(alpha, x, y)
    b = get_b(alpha, w, x, y)

    return w, b
```

求出原问题的 ω 及 b：

```
def get_w(alpha, x, y):
    w = np.zeros(x.shape)
    w[:, 0] = x[:, 0] * alpha * y
    w[:, 1] = x[:, 1] * alpha * y
    return np.sum(w, axis=0)
```

```
def get_b(alpha, w, x, y):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0:
            index.append(i)
    index = np.array(index)

    x = x[index]
    y = y[index]

    return (y - np.dot(x, w)).sum() / len(index)
```

通过 pred 方法使用 ω 及 b 进行预测：

```
def pred(w, b, x):
    return np.dot(x, w) + b
```

7. 本题中要使用 SVM 实现多分类问题，此处使用一对一的分类方法，即：对于每两组分别构建一个 SVM 用于分类，在预测时，也同样对于每两组之间进行依次预测，每次的预测结果对应的组别其计数加一，最终拥有最大计数的组别即为该多分类问题的最终预测结果：

```
def multi_classify_SVM(V, k):
    U = V[:, :k]

    train_Z = np.dot(U.T, train_data)
    test_Z = np.dot(U.T, test_data)

    w_b_list = []
    index_list = []
    for i in range(1, class_num+1):
        for j in range(i+1, class_num+1):
            index_list.append([i-1, j-1])
            x, y = get_x_y(train_Z, train_label, i, j)
            w, b = solve_dual(x, y)
            w_b_list.append([w, b])

    pred_res = []
    for i in range(test_label.shape[0]):
        temp = np.zeros(class_num)
        for idx in range(len(w_b_list)):
            res = pred(w_b_list[idx][0], w_b_list[idx][1], test_Z.transpose()[i])
            if res >= 0:
                temp[index_list[idx][0]] += 1
            else:
                temp[index_list[idx][1]] += 1
        pred_res.append(temp.argmax()+1)
        # print(temp.argmax()+1, test_label[i])
    pred_res = np.array(pred_res)

    return (pred_res == test_label).sum() / test_label.shape[0]
```

其中，V 为之前计算得到的特征向量矩阵，k 为要取的维度数量，程序最终会输出计算结果。

8. 测试：

```
multi_classify_SVM(V, 10)
```

0.8233532934131736

```
multi_classify_SVM(V, 20)
```

0.9514970059880239

```
multi_classify_SVM(V, 30)
```

1.0

可见，本实验中构建的多分类 SVM 成功完成了多分类任务，达到了实验目的。

结论分析与体会：

1. 在实验前，需要充分理解使用 matlab、python 等工具，才能更好地进行实验，实现实验中的各个步骤。
2. 在实验中，需要理解掌握 PCA 的实现原理，掌握其深层含义，结合实验指导书，才能更好地完成实验；
3. 本次实验通过使用 PCA 对面部特征进行提取，在降低了数据维度的基础上使用多分类 SVM 进行分类，从而实现面部识别的任务，通过 PCA 算法的应用，降低了计算所需的数据量，极大地提高了运算速度，并仍然取得了不错的预测效果。

附录：程序源代码

```
# %%
import numpy as np
import random
import matplotlib.pyplot as plt
from qpsolvers import solve_qp
from skimage.io import imread, imshow

# %%
class_num, image_num = 40, 10
orl_face = []
for i in range(1, class_num+1):
    temp = []
    for j in range(1, image_num+1):
        temp.append(imread('./orl_faces/s{}/{}.pgm'.format(i, j)))
    orl_face.append(temp)

# %%
imshow(orl_face[0][0])

# %%
train_data, train_label = [], []
test_data, test_label = [], []

for i in range(len(orl_face)):
    num = random.randint(5, 7)
    train = random.sample(orl_face[i], num)
    train_data = train_data + train
    train_label = train_label + [i+1 for _ in range(num)]

    temp = []
    for j in range(len(orl_face[i])):
```

```python
        flag = True
        for k in range(len(train)):
            if (orl_face[i][j] == train[k]).all():
                flag = False
                break
        if flag:
            temp.append(orl_face[i][j])

    test_data = test_data + temp
    test_label = test_label + [i+1 for _ in range(10-num)]

# %% [markdown]
# 对数据集进行处理，$x^{(i)}$为列向量：

# %%
train_data = np.array(train_data)
train_data = train_data.reshape(train_data.shape[0], -1).transpose()
train_label = np.array(train_label)
test_data = np.array(test_data)
test_data = test_data.reshape(test_data.shape[0], -1).transpose()
test_label = np.array(test_label)

# %% [markdown]
# Center the data (subtract the mean
$\mu=\frac{1}{N}\sum^N_{i=1}x^{(i)}$ from each data point)

# %%
train_data = train_data - train_data.mean(axis=1).reshape(-1, 1)
test_data = test_data - test_data.mean(axis=1).reshape(-1, 1)

# %% [markdown]
# Compute the covariance matrix:
# $$
# S=\frac{1}{N}\sum^N_{i=1}x^{(i)}x^{(i)^T}=\frac{1}{N}XX^T
# $$

# %%
N = train_data.shape[1]
S = np.dot(train_data, train_data.T) / N

# %% [markdown]
# The eigendecomposition of the covariance matrix S

# %%
```

```python
W0, V0 = np.linalg.eig(S)

# %%
W, V = np.real(W0), np.real(V0)
# W, V = W0, V0

# %%
index = (-W).argsort()
W = W[index]
V = V[:, index]

# %%
def get_H(x, y):
    m, n = x.shape
    P = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            P[i, j] = np.dot(x[i], x[j]) * y[i] * y[j]
    return P

# %%
def get_w(alpha, x, y):
    w = np.zeros(x.shape)
    w[:, 0] = x[:, 0] * alpha * y
    w[:, 1] = x[:, 1] * alpha * y
    return np.sum(w, axis=0)

# %%
def get_b(alpha, w, x, y):
    m, n = x.shape
    index = []
    for i in range(m):
        if alpha[i] > 0:
            index.append(i)
    index = np.array(index)

    x = x[index]
    y = y[index]

    return (y - np.dot(x, w)).sum() / len(index)

# %%
def solve_dual(x, y):
    m, n = x.shape
```

```python
    P = get_H(x, y)
    q = np.ones(m) * (-1)
    G, h = None, None
    A = y.astype('float')
    b = np.zeros(1)
    lb = np.zeros(m)
    ub = None

    alpha = solve_qp(P, q, G, h, A, b, lb, ub, solver='cvxopt')
    # print(alpha)

    w = get_w(alpha, x, y)
    b = get_b(alpha, w, x, y)

    return w, b

# %%
def pred(w, b, x):
    return np.dot(x, w) + b

# %%
def get_x_y(Z, label, label_1, label_2):
    index_1 = np.where(label==label_1)[0]
    index_2 = np.where(label==label_2)[0]

    y_1 = [1 for _ in range(index_1.shape[0])]
    y_2 = [-1 for _ in range(index_2.shape[0])]
    y = np.array(y_1 + y_2)

    index = np.concatenate((index_1, index_2)).flatten()
    x = Z[:, index].transpose()

    return x, y

# %%
def multi_classify_SVM(V, k):
    U = V[:, :k]

    train_Z = np.dot(U.T, train_data)
    test_Z = np.dot(U.T, test_data)

    w_b_list = []
    index_list = []
    for i in range(1, class_num+1):
```

```
        for j in range(i+1, class_num+1):
            index_list.append([i-1, j-1])
            x, y = get_x_y(train_Z, train_label, i, j)
            w, b = solve_dual(x, y)
            w_b_list.append([w, b])

    pred_res = []
    for i in range(test_label.shape[0]):
        temp = np.zeros(class_num)
        for idx in range(len(w_b_list)):
            res = pred(w_b_list[idx][0], w_b_list[idx][1],
test_Z.transpose()[i])
            if res >= 0:
                temp[index_list[idx][0]] += 1
            else:
                temp[index_list[idx][1]] += 1
        pred_res.append(temp.argmax()+1)
        # print(temp.argmax()+1, test_label[i])
    pred_res = np.array(pred_res)

    return (pred_res == test_label).sum() / test_label.shape[0]

# %%
multi_classify_SVM(V, 10)

# %%
multi_classify_SVM(V, 20)

# %%
multi_classify_SVM(V, 30)
```