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：201705130113	姓名：黄瑞哲	班级：计科 17.3
实验题目：PCA 与人脸检测		
实验学时：4	实验日期：2019.12.20	

实验目的：

1. 了解 PCA 原理，能够自己实现 PCA
2. 使用 PCA 与 SVM 完成人脸检测

硬件环境：

Intel Core i5-8300H @ 2.3GHz

软件环境：

Windows10 Pro 1903

Python 3.7

Visual Studio Code

实验步骤与内容：

一、 PCA 算法

步骤一： 计算样本均值

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$$

步骤二： 去中心化

步骤三： 计算协方差矩阵

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

步骤四： 求解协方差矩阵的特征值与特征向量

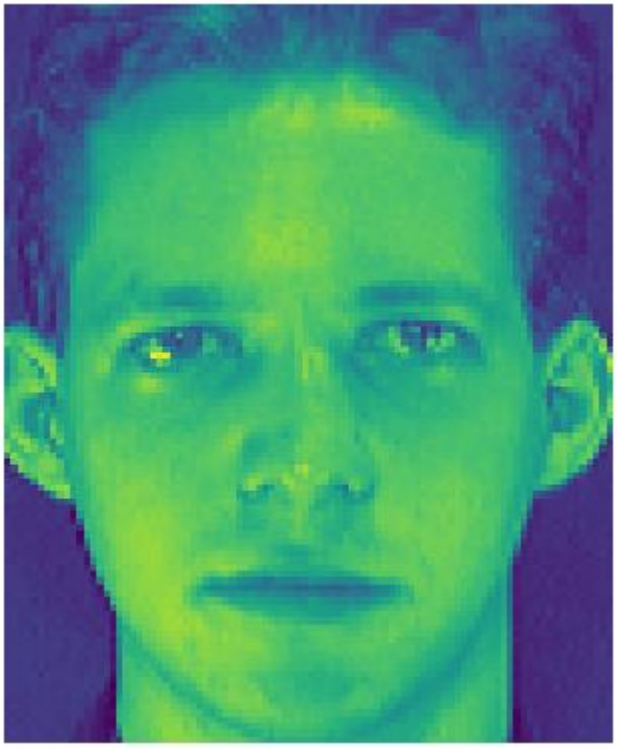
步骤五： 选取前 k 大的特征值对应的特征向量构成矩阵，完成映射

$$Z = U^T X$$

其中 Z 是 k×n 的矩阵，U 是 D×k 的矩阵

二、 人脸检测

数据中提供了 40 个人的人脸数据，每人有 10 张图片。



这种图片是单通道的灰度图片，像素为 112×92，现在的任务是每个人选择 5 张照片作为训练数据集，剩下的 5 张作为测试数据集。使用 PCA 和 SVM 完成这个任务！

为什么使用 PCA：我们利用的是每张图片像素的颜色信息，但是这个图片一共有 112*92=10304 个像素点，也就是说每个数据的 feature 共有 10304 个，维度特别高，直接进行计算是不现实的，不仅浪费内存，还浪费时间。但是考虑到这个图片的像素点中有些像素是不重要的，对整个分类器的决策没有影响，那么我们可以使用 PCA 来完成维度的压缩，仅仅选取一部分 feature 作为主要的特征。

如何使用 SVM：我们知道，对于 SVM 来说，他只能完成二分类任务，但是在这里我们的测试需要完成 40 类的分类任务，需要从 40 个类群中识别测试数据属于哪一类，直接应用 SVM 是不可能的。那么我们可以考虑两两类群之间建立一个 SVM 来完成两个类群之间的分类任务，这样一共有 40*39/2=780 个 SVM 分类器，对于每种分类器我都进行训练，最终在测试的时候枚举所有的分类器，统计下测试数据在每个分类器的得分，根据这个得分来决策该测试数据属于哪一类。

```
svm_all = []
for i in range(1, 41):
    for j in range(i + 1, 41):
        X1 = X_train_pca[y_train == i]
        X2 = X_train_pca[y_train == j]
        y1 = np.array([1 for k in range(len(X1))])
        y2 = np.array([-1 for k in range(len(X2))])
        svm = SVM()
        svm.fit(np.vstack((X1, X2)), np.hstack((y1, y2)))
        svm_all.append(svm)
```

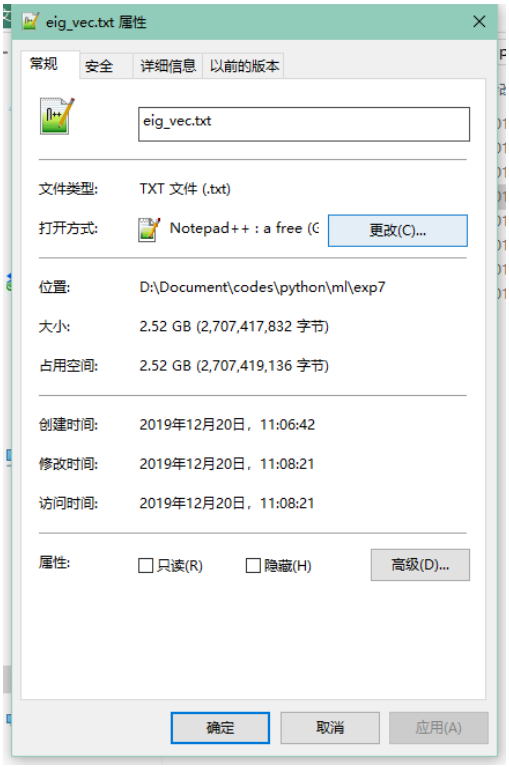
运行程序，通过调整 k 来观察分类的正确率。

```
5:5
k=1    33/200 =16.50%
k=2    88/200 =44.00%
k=5    146/200=73.00%
k=10   178/200=89.00%
k=50   185/200=92.50%
k=100  180/200=90.00%
k=200  180/200=90.00%
k=400  177/200=88.50%
```

可以发现当 k 取 10 的时候，分类的准确率已经达到了 89%，也就是说大部分的 feature 对分类的准确性提升没有特别突出的贡献，甚至还会干扰分类的决策。

结论分析与体会：

在这次实验中，我遇到了两个问题：
第一个是协方差矩阵过大，为 10304×10304 的对称阵，对其求解特征值和特征向量时需要的时间特别大。这样我只好把求解完成得到的特征值和特征向量保存下来，之后再次运行的时候直接从文件中读取，避免了重复计算。但是这样还有一个问题，就是保存的文件特别大，读取时候的效率也不是很高，这个问题目前还不知道怎么解决。



第二个问题是最开始求解特征值和特征向量的时候，发现求出来的解是复数。在网上搜索了一下 numpy 中求解特征值时是将矩阵作为复数来迭代的，最后返回结果的时候可能没有完全收敛，虚部的系数接近于零。这样面对复数的时候，不适合使用 SVM，我只好保留了它

的实部，用实部进行 PCA 的映射。sklearn 这个库中的 PCA 方法好像使用了奇异值代替了特征值，避免了出现复数的情况，具体原理以及如何实现的还不太了解。

附录：程序源代码

```
# PCA.py
import numpy as np


class PCA:
    def __init__(self, n_components):
        self.n_components_ = n_components
        self.U_ = None

    def fit_save(self, X, eig_val_path, eig_vec_path):
        m, n = np.shape(X)
        X = X - np.mean(X, axis=0)
        S = np.dot(X.T, X) / m  # (n, n)
        eig_val, eig_vec = np.linalg.eig(S)  # vec (n, 1)
        eig_val = eig_val.real
        eig_vec = eig_vec.real
        np.savetxt(eig_val_path, eig_val)
        np.savetxt(eig_vec_path, eig_vec)
        eig_pairs = [(eig_val[i], eig_vec[:, i]) for i in range(n)]
        eig_pairs.sort(key=lambda pair: -pair[0])
        self.U_ = np.array([pair[1] for pair in eig_pairs[:self.n_components_]])

    def fit_load(self, eig_val_path, eig_vec_path):
        eig_val = np.loadtxt(eig_val_path)
        eig_vec = np.loadtxt(eig_vec_path)
        eig_pairs = [(eig_val[i], eig_vec[:, i]) for i in range(len(eig_val))]
        eig_pairs.sort(key=lambda pair: -pair[0])
        self.U_ = np.array([pair[1] for pair in eig_pairs[:self.n_components_]])

    def fit_transform_save(self, X, eig_val_path, eig_vec_path):
        self.fit_save(X, eig_val_path, eig_vec_path)
        return self.transform(X)

    def fit_transform_load(self, X, eig_val_path, eig_vec_path):
```

```
        self.fit_load(eig_val_path, eig_vec_path)
        return self.transform(X)

    def transform(self, X):
        assert self.U_ is not None
        return np.dot(X - np.mean(X, axis=0), self.U_.T)

# SVM.py
import numpy as np
import cvxopt


class SVM:
    @staticmethod
    def linear_kernel(x1, x2):
        return np.dot(x1, x2)

    @staticmethod
    def RBF_kernel(x1, x2, gamma):
        return np.exp(-gamma*np.dot(x1-x2, x1-x2))

    def __init__(self, kernel=None, C=None, **kargcs):
        self.kernel = SVM.linear_kernel if kernel is None else kernel
        self.C = C if C is None else float(C)
        self.kargcs = kargcs

    def fit(self, X, y):
        m, n = X.shape
        print(np.shape(X), np.shape(y))
        print("overall %d training datas with %d dimensions" % (m, n))

        K = np.zeros((m, m))
        for i in range(m):
            for j in range(m):
                K[i, j] = self.kernel(X[i], X[j], **self.kargcs)

        P = cvxopt.matrix(np.outer(y, y) * K)
        # P = cvxopt.matrix(y.T*y*K)
        q = cvxopt.matrix(np.ones(m) * -1)
        # sigma(a*y)=0
        A = cvxopt.matrix(y, (1, m), 'd')
        b = cvxopt.matrix(0.0)

        if self.C is None or self.C == 0:  # hard-margin
            # a >= 0
```

```
            G = cvxopt.matrix(np.eye(m) * -1)
            h = cvxopt.matrix(np.zeros(m))
        else:   # soft-margin
            # a >= 0 && a <= c
            p1 = np.eye(m) * -1
            p2 = np.eye(m)
            G = cvxopt.matrix(np.vstack((p1, p2)))
            p1 = np.zeros(m)
            p2 = np.ones(m) * self.C
            h = cvxopt.matrix(np.hstack((p1, p2)))

        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        a = np.ravel(solution['x'])
        # 非0的a对应支持向量
        sv = a > 0
        # 支持向量对应下标
        self.support_ = np.arange(len(a))[sv]
        self.a = a[sv]
        self.support_vectors_ = X[sv]
        self.support_vectors_y = y[sv]
        print("%d support vectors out of %d points." % (len(self.a), m))

        if self.kernel == SVM.linear_kernel:    # 线性可分
            self.w = np.zeros(n)
            for i in range(len(self.a)):
                self.w    +=    self.a[i]    *    self.support_vectors_y[i]    *
self.support_vectors_[i]
            self.b = 0
            for i in range(len(self.a)):
                self.b    +=    self.support_vectors_y[i]    -    np.dot(self.w,
self.support_vectors_[i])
            self.b /= len(self.a)
        else:   # 非线性
            self.w = None
            self.b = 0
            for i in range(len(self.a)):
                self.b += self.support_vectors_y[i]
                self.b    -=    np.sum(self.a    *    self.support_vectors_y    *
K[self.support_[i], sv])
            self.b /= len(self.a)

    def decision_function(self, X):
        if self.w is not None:  # 线性可分
            return np.dot(X, self.w) + self.b
```

```
        y_decision = np.zeros(len(X))
        for j in range(len(X)):
            for i in range(len(self.a)):
                y_decision[j] += self.a[i] * self.support_vectors_y[i] *
self.kernel(X[j], self.support_vectors_[i], **self.kargcs)
        return y_decision + self.b

    def predict(self, X):
        return np.sign(self.decision_function(X))

# exp7.py
#  F. Samaria and A. Harter
#   "Parameterisation of a stochastic model for human face identification"
#   2nd IEEE Workshop on Applications of Computer Vision
#   December 1994, Sarasota (Florida).

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image
from PCA import PCA
from SVM import SVM
# from sklearn.decomposition import PCA
# from sklearn.svm import SVC


def load_data(file):
    X_train = []
    y_train = []
    X_test = []
    y_test = []
    for i in range(1, 41):
        for j in range(1, 11):
            face = matplotlib.image.imread(file + "/s%d/%d.pgm" % (i, j))
            m, n = np.shape(face)
            face = np.array(face, dtype=np.float64) / 255
            face = np.reshape(face, m * n)
            if j <= 5:
                X_train.append(face)
                y_train.append(i)
            else:
                X_test.append(face)
                y_test.append(i)
    return np.array(X_train), np.array(y_train), np.array(X_test),
np.array(y_test)
```

```
if __name__ == "__main__":
    X_train, y_train, X_test, y_test = load_data("exp7/orl_faces")

    pca = PCA(n_components=5)
    #     pca.fit_save(np.vstack((X_train,     X_test)),     "exp7/eig_val.txt",
"exp7/eig_vec.txt")
    pca.fit_load("exp7/eig_val.txt", "exp7/eig_vec.txt")
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    # pca.fit(np.vstack((X_train, X_test)))
    # X_train_pca = pca.transform(X_train)
    # X_test_pca = pca.transform(X_test)

    svm_all = []
    for i in range(1, 41):
        for j in range(i + 1, 41):
            X1 = X_train_pca[y_train == i]
            X2 = X_train_pca[y_train == j]
            y1 = np.array([1 for k in range(len(X1))])
            y2 = np.array([-1 for k in range(len(X2))])
            svm = SVM()
            svm.fit(np.vstack((X1, X2)), np.hstack((y1, y2)))
            svm_all.append(svm)

    correct = 0
    for k in range(len(X_test_pca)):
        score = [0 for i in range(41)]
        cnt = 0
        for i in range(1, 41):
            for j in range(i + 1, 41):
                if svm_all[cnt].predict(X_test_pca[k]) == 1:
                    score[i] += 1
                else:
                    score[j] += 1
                cnt += 1
        max_score = max(score)
        each = []
        for i in range(1, 41):
            if score[i] == max_score:
                each.append(i)
                if i == y_test[k]:
                    correct += 1
```

```
        print(each)
    print("%d out of %d predictions correct(%.2f%%)" % (correct, len(X_test_pca),
correct / len(X_test_pca) * 100))
```