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：   1. PCA算法   步骤一： 计算样本均值    步骤二： 去中心化  步骤三： 计算协方差矩阵    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    其中Z是k×n的矩阵，U是D×k的矩阵   1. 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。    运行程序，通过调整k来观察分类的正确率。      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))