**ECE 661 Homework 11**

**Weichen Xu**

**xu1363@purdue.edu**

**Principle Component Analysis (PCA)**  
The PCA method can be used for dimensionality reduction to get data of lower dimension by only using principal components of data. In this task, we are given a training data set including 30 classes of different face, each class has 21 images, 630 images in total. And another testing data set is given for testing the performance, which has also 30 classes, for each class having different 21 images.

First the images are converted to grayscale image of size 128\*128, then images are flattened to create vectors of size 16384\*1. Each vector is then normalized as .

Next, we can obtain the global mean of N vectors as following

The covariance matrix can be obtained as following

Where is expressed as following

Here directly implementing decomposition of matrix will result in instable numerical solution and high computation complexity. To solve this, we can use a trick to avoid directly calculating decomposition of such a huge matrix. Instead we will obtain the eigen decomposition of the matrix , with obtained eigen vectors . To further obtain the eigen vectors of  using the following equation

Then we normalize the to get the

The eigen vectors of will be arranged based on its corresponding eigen values in the descending order. For the given number of subspace dimension , we can build following subspace matrix

Then all flattened image vectors for training and testing datasets will be projected to the -dimensional subspace feature vectors

**Linear Discriminant Analysis (LDA)**

The LDA method tends to model the difference between classes with the object to maximize the Fischer Discriminant Function

First, we define the global mean and class mean of each single class as

Where is the number of training images in the th class, is the total number of training images.

The class difference matrix can be expressed as

Where is the total number of classes.

Thus, the between-class scatter can be expressed as Instead of directly implementing eigen decomposition on , we will use similar trick as in PCA i.e. obtain the eigen vectors of , then calculate the eigen vectors of by

Furthermore, the eigen vector is normalized. Then we can build the matrix by retaining only by retaining only largest eigen vectors

Then the matrix can be obtained as

Where is the diagonal matrix containing largest eigen values.

The within-class scatter is defined as , where matrix is expressed by

Then , similarly as previous operation we can obtain the normalized eigen vectors of . Then the smallest eigen vectors is , we will get projection by

The lower p-dimensional features will be obtained by

**Implementations**

Step 1: Obtain the lower p-dimensional representations of training images using PCA or LDA

Step 2: Train the KNN classifier with p-dimensional representations of training images and predict p-dimensional representations of testing images and obtain the overall accuracy.

Step 3: Iterate the number of subspace dimensions from 1 to 20, observe the change of accuracy for both PCA and LDA.

**Experiment Result**

A picture containing shape

Description automatically generated

Figure 1: Accuracy of PCA and LDA with increasing dimensions

From the above figure, we can see initially when the number of dimensions is smaller than 5, the accuracy of PCA is slightly higher than accuracy of LDA. While the LDA reaches 100% accuracy earlier when dimensions are 11, while the PCA reaches 100% when dimensions are 12.

**Code**

*#!/usr/bin/env python  
# coding: utf-8*import cv2  
import os  
import numpy as np  
from sklearn.neighbors import KNeighborsClassifier  
import matplotlib.pyplot as plt  
  
  
  
  
def PCA(dir\_train, dir\_test, K=20):  
 *# get train data's features using PCA* files = os.listdir(dir\_train)  
 N = len(files)  
 vec\_train = []  
 for i in range(N):  
 img = cv2.imread(dir\_train + files[i])  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img\_vec = gray.flatten()  
 img\_vec = img\_vec / np.linalg.norm(img\_vec)  
 vec\_train.append(img\_vec)  
   
 vec\_train = np.array(vec\_train).transpose()  
 mean\_train = np.mean(vec\_train, axis=1)  
 vec\_train = vec\_train - mean\_train.reshape(vec\_train.shape[0], -1)  
   
 *# get train data's labels, test data has same labels* labels = []  
 for i in range(30):  
 for j in range(21):  
 labels.append(i+1)  
 labels = np.array(labels)  
   
 *# get test data's features* files = os.listdir(dir\_test)  
 N = len(files)  
 vec\_test = []  
 for i in range(N):  
 img = cv2.imread(dir\_test + files[i])  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img\_vec = gray.flatten()  
 img\_vec = img\_vec / np.linalg.norm(img\_vec)  
 vec\_test.append(img\_vec)  
   
 vec\_test = np.array(vec\_test).transpose()  
 mean\_test = np.mean(vec\_test, axis=1)  
 vec\_test = vec\_test - mean\_test.reshape(vec\_test.shape[0], -1)  
   
 *# tricky way to get eigenvectors* d, u = np.linalg.eig(np.dot(vec\_train.transpose(), vec\_train))  
 idx\_sorted = np.argsort(d)[::-1] *# biggest eigen value first* U = u[:, idx\_sorted]  
 w = np.dot(vec\_train, U)  
 w = w / np.linalg.norm(w, axis=0)  
   
 accuracys = []  
 *# iterate with different p value* for p in range(K):  
 result = np.zeros((len(labels), 1))  
 *# get subspace of p+1 dimensions* sub = w[:, :p+1]  
 features\_train = np.dot(sub.transpose(), vec\_train)  
 features\_test = np.dot(sub.transpose(), vec\_test)  
 KNN = KNeighborsClassifier(n\_neighbors=1)  
 KNN.fit(features\_train.transpose(), labels)  
 preds = KNN.predict(features\_test.transpose())  
 result[preds == labels] = 1  
 accuracys.append(np.sum(result)/ result.shape[0])  
   
 return accuracys  
  
  
  
  
def LDA(dir\_train, dir\_test, K=20):  
 *# get train data's features using LDA* files = os.listdir(dir\_train)  
 N = len(files)  
 N\_class = 30  
 vec\_train = []  
 for i in range(N):  
 img = cv2.imread(dir\_train + files[i])  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img\_vec = gray.flatten()  
 img\_vec = img\_vec / np.linalg.norm(img\_vec)  
 vec\_train.append(img\_vec)  
   
 vec\_train = np.array(vec\_train).transpose()  
 mean\_train = np.mean(vec\_train, axis=1)  
   
 *# get train data's labels* labels = []  
 for i in range(30):  
 for j in range(21):  
 labels.append(i+1)  
 labels = np.array(labels)  
   
 *# get test data's features* files = os.listdir(dir\_test)  
 N = len(files)  
 vec\_test = []  
 for i in range(N):  
 img = cv2.imread(dir\_test + files[i])  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img\_vec = gray.flatten()  
 img\_vec = img\_vec / np.linalg.norm(img\_vec)  
 vec\_test.append(img\_vec)  
   
 vec\_test = np.array(vec\_test).transpose()  
 mean\_test = np.mean(vec\_test, axis=1)  
   
 mean\_class = np.zeros((vec\_test.shape[0], N\_class))  
 X = np.zeros(vec\_train.shape)  
 for i in range(N\_class):  
 mean\_class[:, i] = np.mean(vec\_train[:, i\*20:(i+1)\*20], axis=1)  
 X[:, i\*20:(i+1)\*20] = vec\_train[:, i\*20:(i+1)\*20]   
 \- mean\_class[:, i].reshape(X.shape[0] ,-1)  
   
 M = mean\_class - mean\_train.reshape(vec\_test.shape[0], -1)  
 *# eigen decomposition of between class scatter SB* d, u = np.linalg.eig(np.dot(M.transpose(), M))  
 idx\_sorted = np.argsort(d)[::-1] *# biggest eigen value first* D = d[idx\_sorted]  
 U = u[:, idx\_sorted]  
 v = np.dot(M, U)  
 V = v / np.linalg.norm(v, axis=0)  
 DB = np.zeros((N\_class, N\_class))  
 for i in range(N\_class):  
 DB[i,i] = D[i]\*\*(-0.5)  
 Z = np.dot(V, DB)  
 temp = np.dot(Z.transpose(), X)  
 *# eigen decomposition of ZSWZ* d, u = np.linalg.eig(np.dot(temp, temp.transpose()))  
 idx\_sorted = np.argsort(d) *# smallest eigen value first* U = u[:, idx\_sorted]  
   
 accuracys = []  
 *# iterate with different p value* for p in range(K):  
 result = np.zeros((len(labels), 1))  
 *# get subspace of p+1 dimensions* Wp = U[:, :p+1]  
 sub = np.dot(Z, Wp)   
 sub = sub / np.linalg.norm(sub, axis=0)  
 features\_train = np.dot(sub.transpose(),   
 \vec\_train - mean\_train.reshape(vec\_train.shape[0], -1))  
 features\_test = np.dot(sub.transpose(),   
 \vec\_test - mean\_test.reshape(vec\_test.shape[0], -1))  
 KNN = KNeighborsClassifier(n\_neighbors=1)  
 KNN.fit(features\_train.transpose(), labels)  
 preds = KNN.predict(features\_test.transpose())  
 result[preds == labels] = 1  
 accuracys.append(np.sum(result)/ result.shape[0])  
   
 return accuracys  
  
  
  
  
dir\_train = **'/home/xu1363/Documents/ECE 661/hw11/task1/train/'**dir\_test = **'/home/xu1363/Documents/ECE 661/hw11/task1/test/'**accuracys\_PCA = PCA(dir\_train, dir\_test, K=20)  
accuracys\_LDA = LDA(dir\_train, dir\_test, K=20)  
  
plt.plot(np.arange(1, 20+1), accuracys\_PCA, **'b'**, label=**'PCA'**)  
plt.plot(np.arange(1, 20+1), accuracys\_LDA, **'r'**, label=**'LDA'**)  
  
for i in range(20):  
 plt.plot(i+1, accuracys\_PCA[i], **'b\*'**)  
for i in range(20):  
 plt.plot(i+1, accuracys\_LDA[i], **'r+'**)  
   
plt.legend()  
plt.xlabel(**'Number of dimensions'**)  
plt.ylabel(**'Accuracy'**)  
plt.title(**'Accuracy of PCA and LDA with increasing dimension (1-20)'**)  
plt.xlim(0,20)  
plt.ylim(0,1.1)  
plt.savefig(**'/home/xu1363/Documents/ECE 661/hw11/accuracy\_task1.png'**)