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 x_i.

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

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i$$

The covariance matrix can be obtained as following

$$C = XX^T$$

Where X is expressed as following

$$X = [\, x_1 - m \mid x_2 - m \mid x_3 - m \mid \cdots \mid x_{N-1} - m \mid x_N - m \,]$$

Here directly implementing decomposition of C 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 X^TX, with obtained eigen vectors v_i. To further obtain the eigen vectors w_i of XX^T using the following equation

$$u_i = Xv_i$$

Then we normalize the u_i to get the w_i

$$w_i = \frac{u_i}{\|u_i\|}$$

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

$$W_p = [\, w_1 \mid w_2 \mid w_3 \mid \cdots \mid w_{p-1} \mid w_p \,]$$

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

$$y_i = W_p^T(\, x_i - m)$$

Linear Discriminant Analysis (LDA)

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

$$J(w_i) = \frac{w_i^T S_B w_i}{w_i^T S_W w_i}$$

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

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$m_i = \frac{1}{k_i}\sum_{i=1}^{k_i} x_i$$

Where k_i is the number of training images in the ith class, N is the total number of training images.

The class difference matrix M can be expressed as

$$M = [\, m_1 - m \mid m_2 - m \mid m_3 - m \mid \cdots \mid m_{C-1} - m \mid x_C - m \,]$$

Where C is the total number of classes.

Thus, the between-class scatter can be expressed as $S_B = MM^T$. Instead of directly implementing eigen decomposition on S_B, we will use similar trick as in PCA i.e. obtain the eigen vectors u_i of $M^T M$, then calculate the eigen vectors v_i of MM^T by

$$v_i = Mu_i$$

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

$$Y = [\, u_1 \mid u_2 \mid u_3 \mid \cdots \mid u_{p-1} \mid u_p \,]$$

Then the Z matrix can be obtained as

$$Z = YD_B^{-1/2}$$

Where D_B is the diagonal matrix containing p largest eigen values.

The within-class scatter S_W is defined as $S_W = KK^T$, where K matrix is expressed by

$$K = [\, {x_1}^1 - m_1 \mid {x_1}^2 - m_1 \mid \cdots \mid {x_1}^{k_1} - m_1 \mid \cdots \mid {x_C}^{k_C} - m_C \,]$$

Then $Z^T S_W Z = Z^T KK^T Z = Z^T K(Z^T K)^T$, similarly as previous operation we can obtain the normalized eigen vectors U of $Z^T S_W Z$. Then the smallest p eigen vectors is U_p, we will get projection W_p by

$$W_p = ZU_p$$

The lower p-dimensional features will be obtained by

$$y_i = W_p^T(\, x_i - m)$$

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 p from 1 to 20, observe the change of accuracy for both PCA and LDA.

Experiment Result

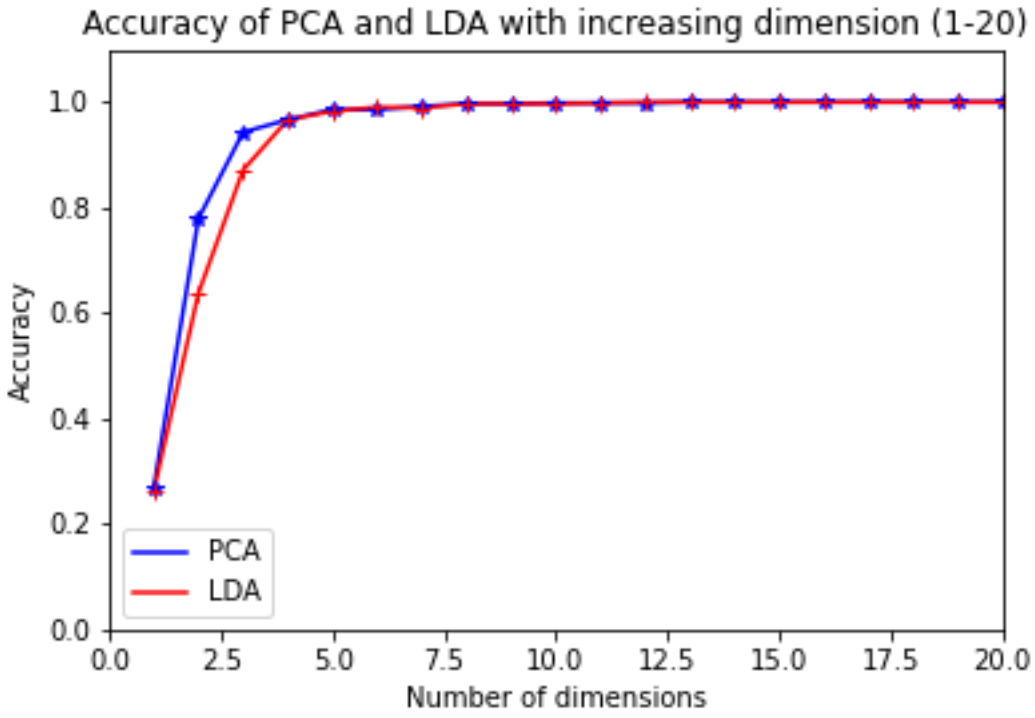

Figure 1: Accuracy of PCA and LDA with increasing dimensions

From the above figure, we can see initially when the number of dimensions p 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')
```