

Report

Project :

**Face recognition using principal component analysis**

Group members

Abdul qayyum(60809)

Syed wali(61030)

## Image and Face Recognition

In computer, pictures are represented as a matrix of pixels, with each pixel a particular color coded in some numerical values. It is natural to ask if computer can read the picture and understand what it is, and if so, whether we can describe the logic using matrix mathematics. To be less ambitious, people try to limit the scope of this problem to identifying human faces. An early attempt for face recognition is to consider the matrix as a high dimensional detail and we infer a lower dimension information vector from it, then try to recognize the person in lower dimension. It was necessary in the old time because the computer was not powerful and the amount of memory is very limited. However, by exploring how to **compress** image to a much smaller size, we developed a skill to compare if two images are portraying the same human face even if the pictures are not identical.

In 1987, a paper by Sirovich and Kirby considered the idea that all pictures of human face to be a weighted sum of a few “key pictures”. Sirovich and Kirby called these key pictures the “eigenpictures”, as they are the eigenvectors of the covariance matrix of the mean-subtracted pictures of human faces. In the paper they indeed provided the algorithm of principal component analysis of the face picture dataset in its matrix form. And the weights used in the weighted sum indeed correspond to the projection of the face picture into each eigenpicture.

In 1991, a paper by Turk and Pentland coined the term “eigenface”. They built on top of the idea of Sirovich and Kirby and use the weights and eigenpictures as characteristic features to recognize faces. The paper by Turk and Pentland laid out a memory-efficient way to compute the eigenpictures. It also proposed an algorithm on how the face recognition system can operate, including how to update the system to include new faces and how to combine it with a video capture system. The same paper also pointed out that the concept of eigenface can help reconstruction of partially obstructed picture.

**Face recongnition applicable areas**

Face recognition has many applicable areas. Moreover, it can be categorized into face identification,

face classification, or sex determination. The most useful applications contain crowd surveillance,

video content indexing, personal identification (ex. driver’s licence), mug shots matching,

entrance security, etc.

**Advantages:**

* Easy to implement and computationally less expensive.
* No knowledge (such as facial feature) of the image required (except id).

**Limitations :**

* Proper centered face is required for training/testing.
* The algorithm is sensitive to lightining, shadows and also scale of face in the image .
* Front view of the face is required for this algorithm to work properly.

## Principal component analysis

## PCA is a statistical approach used for reducing the number of variables in face recognition. In PCA, every image in the training set is represented as a linear combination of weighted eigenvectors called eigenfaces. ... A number of experiments were done to evaluate the performance of the face recognition system.

## Main idea of using principal component analysis

## The main idea of using PCA for face recognition is to express the large 1-D vector of pixels constructed from 2-D facial image into the compact principal components of the feature space. ... Eigenspace is calculated by identifying the eigenvectors of the covariance matrix derived from a set of facial images(vectors).

## Jobs of principal component analysis

The jobs which PCA can do are prediction, redundancy removal, feature extraction, data compression,

etc. Because PCA is a classical technique which can do something in the linear domain,

applications having linear models are suitable, such as signal processing, image processing, system

## and control theory, communications, etc.

## Eigenface

## Eigenfaces is a method that is useful for face recognition and detection by determining the variance of faces in a collection of face images and use those variances to encode and decode a face in a machine learning way without the full information reducing computation and space complexity.

## 

## Code of the project

## """Training set is the one on which we train and fit our model basically to fit the parameters whereas test data is used only to assess performance of

## model. Training data's output is available to model whereas testing data is the unseen data for which predictions have to be made."""

## import numpy as np

## import pandas as pd

## import matplotlib.pyplot as plt

## from time import time

## from sklearn.model\_selection import train\_test\_split #to get a set of training and testing data individually

## #Train selects samples at random. It's a good strategy. Suppose 80% of the data in the dataset can be used for training while the rest for testing purpose

## from sklearn.metrics import classification\_report #

## from sklearn.decomposition import PCA

## from sklearn.svm import SVC

## ##Helper functions. Use when needed.

## def show\_orignal\_images(pixels):

## #Displaying Orignal Images

## fig, axes = plt.subplots(6, 10, figsize=(11, 7),

## subplot\_kw={'xticks':[], 'yticks':[]})

## for i, ax in enumerate(axes.flat):

## ax.imshow(np.array(pixels)[i].reshape(64, 64), cmap='gray')

## plt.show()

## def show\_eigenfaces(pca):

## #Displaying Eigenfaces

## fig, axes = plt.subplots(3, 8, figsize=(9, 4),

## subplot\_kw={'xticks':[], 'yticks':[]})

## for i, ax in enumerate(axes.flat):

## ax.imshow(pca.components\_[i].reshape(64, 64), cmap='gray')

## ax.set\_title("PC " + str(i+1))

## plt.show()

## ## Step 1: Read dataset and visualize it

## df = pd.read\_csv("face\_data.csv")

## targets = df["target"]

## pixels = df.drop(["target"],axis=1)

## print(np.array(pixels).shape)

## show\_orignal\_images(pixels)

## ## Step 2: Split Dataset into training and testing

## x\_train, x\_test, y\_train, y\_test = train\_test\_split(pixels, targets)

## ## Step 3: Perform PCA.

## pca = PCA(n\_components=150).fit(x\_train)

## plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

## plt.xlabel('number of components')

## plt.ylabel('cumulative explained variance');

## plt.show()

## show\_eigenfaces(pca)

## ## Step 4: Project Training data to PCA

## print("Projecting the input data on the eigenfaces orthonormal basis")

## Xtrain\_pca = pca.transform(x\_train)

## ##############

## ## Step 5: Initialize Classifer and fit training data

## clf = SVC(kernel='rbf',C=1000,gamma=0.001)

## clf = clf.fit(Xtrain\_pca, y\_train)

## ## Step 6: Perform testing and get classification report

## print("Predicting people's names on the test set")

## t0 = time()

## Xtest\_pca = pca.transform(x\_test)

## y\_pred = clf.predict(Xtest\_pca)

## print("done in %0.3fs" % (time() - t0))

## print(classification\_report(y\_test, y\_pred))

## Output Of Project

## 