



# *COSC579: Computer Vision: Eigenfaces*

Jeremy Bolton, PhD  
Assistant Teaching Professor

Special Thanks to A. Gates

# *Outline*

- I. Exemplar Classification Problem/Solution: Eigenfaces
  - I. Motivation
  - II. PCA
    - I. Dimensionality Problem
  - III. Sirovich and Kirby
  - IV. Turk and Pentland
  - V. Classification
  - VI. Python Implementation and Example

# *facial recognition*

- Ability for a computer application to identify or verify an individual given an image of that person.
- Sometimes combined with other biometrics, such as finger prints and eye rental scans.
- Face recognition algorithms often use a dimensionality **reduction** techniques
  - **EG PCA (Principle Component Analysis).**

# Why feature reduction?

input: dataset of  $N$  face images

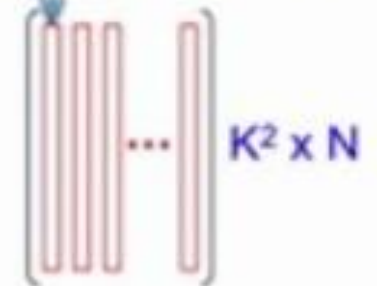


face:  $K \times K$  bitmap of pixels



"unfold" each bitmap to  $K^2$ -dimensional vector

arrange in a matrix  
each face = column



# *Key ideas for facial recognition with pca*

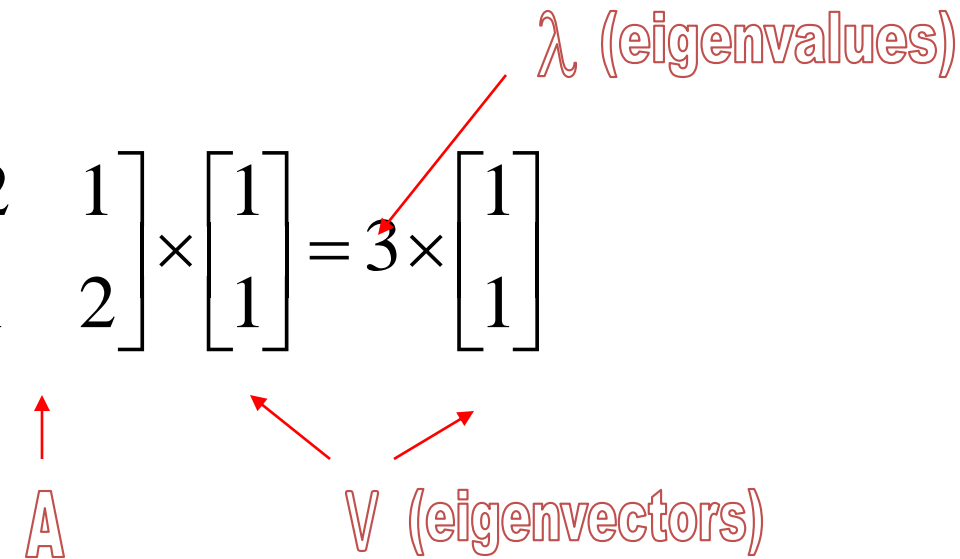
- Images of faces (assuming some preprocessing constants such as the size and position of the faces) have a similar configuration overall.
  - humans all have 2 eyes, 1 nose in the center, a chin, lips, etc..
- Therefore, face images will **NOT be randomly distributed** within the **image space**.
  - The image space, for say a 100 by 100 pixel image is a 10,000 dimensional space.
- Instead, images of faces can be described in a relatively low-D subspace (or manifold).

# *Eigenvalues and Eigenvectors - Definition*

- If  $v$  is a nonzero vector and  $\lambda$  is a number such that  $\mathbf{Av} = \lambda\mathbf{v}$ , then  $v$  is said to be an *eigenvector* of  $A$  with *eigenvalue*  $\lambda$ .

Example

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3 \times \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

  
A                      V (eigenvectors)                      λ (eigenvalues)

# *Eigenface: Preprocessing Procedure*

- 1) Get the data.
  - 2) Transform each face into a column vector and place all into a face matrix.
  - 3) Calculate the mean vector from all the faces and render it so you can see it.
  - 4) Normalize the face matrix by subtracting the mean from each column face in the matrix. (Call this A)
  - 5) Get the reduced version of the covariance matrix (Turk & Pentland):  $C' = A^T A$
  - 6) Find the eigenvalues and eigenvectors of  $C'$
  - 7) Enable an option to choose the  $k$  highest eigenvalues and corresponding eigenvectors. Create a matrix of the  $k$  highest eigenvectors.
  - 8) Transform the  $k$  highest eigenvectors **back into the original space by multiplying each of them by A. These are called the eigenfaces – they are  $(r*c)$  by 1**
- Recall that A is size  $(r*c)$  by  $N$ , where  $N$  is the number of faces in the dataset.

# *Eigenfaces*

- **PCA – Principle Component Analysis** – (AKA Karhunen Loeve Expansion) is to determine the vectors which best account for the distribution of the face images (Turk and Pentland 1991).
- - These vectors (eigenfaces) will then define a **subspace or basis** of face images – a **face space**.
- - Each eigenvector (eigenface) will be  $R \times C$  by 1, where  $R$  are the num of row pixels and  $C$  are the num of column pixels.
- - Because each eigenvector of the covariance matrix of the original faces space is the same size as a face in the dataset, **it will have face-like qualities** (minus the mean).
- - This is why the eigenvectors in this case are referred to as eigenfaces.



# *eigenface method*

RE: Sirovish and Kirby (1987) and Turk and Pentland (1991)

- 1) Create a low-dimensional representation of face images using PCA.
- 2) Specifically, use a collection of face images (properly preprocessed) to create a **basis** for a set of features.
  - - The **basis** images are called **eigenfaces**.
- 3) Then, the reduced **basis** of **eigenfaces** can be **linearly combined** to re-create and match to any image in the original (large) dataset of face images (often called the **training set**).
- Example: Suppose the training set of face images has N images. Then PCA will form a very reduced basis of Y ( $Y \ll N$ ) images with which any of the original N images can be constructed.
- Each face is a **proportion** of faces from the basis:
- **SomeFace1 = mean + .12\*BF1 + .45\*BF2 + ...+ .05BFn (where BF is basis face)**
- **Interestingly, it does not require many eigenfaces to approximate most faces.**

# *reducing size and computation*

- 1) Creating a **basis** of eigenfaces from which all faces in the original dataset (or even a variation of a face from the original dataset) can be constructed – **reduces the search space size.**
- 2) **To create the basis, the eigenvectors and eigenvalues** from the covariance matrix of the original set must be calculated.
  - - This is often **highly time and space intensive** and in some cases intractable (depending on the size of the dataset, the size of each image in pixels, and the nature of the computer being used).
- 3) **Turk and Pentland (1991)** created a method for calculating the eigenvectors without the use of such space and time.
  - - Their method used matrices sized by images, not by pixels.

# *Creating Eigenfaces*

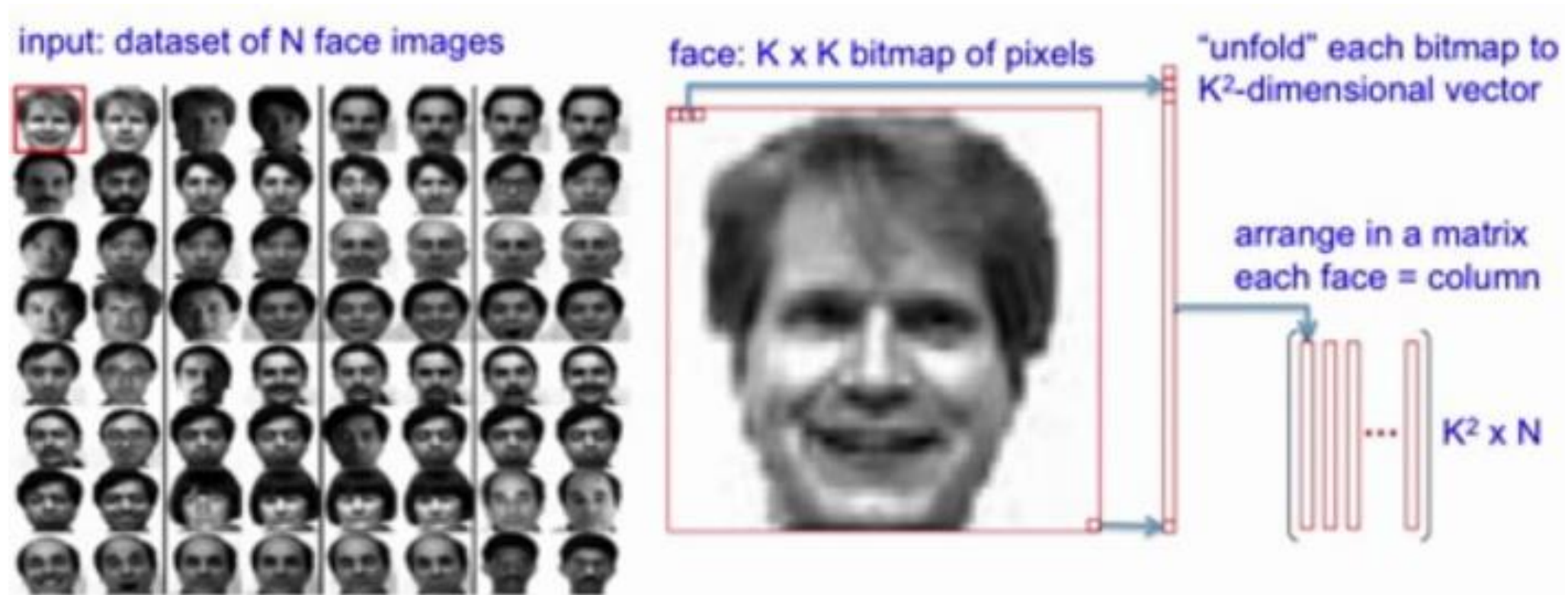
- Recall that the **eigenfaces are the eigenvectors** (from the largest eigenvalues) due to PCA on a covariance matrix.
- **Steps:**
  - 1) Create/prepare a dataset of face images. This is the **training set**.
    - Note: Face training sets are always taken under the same lighting, are normalized to that the eyes and mouth align, and are resampled to a pixel resolution of  $r \times c$  (all images are the same exact size).
  - 2) Each image can be represented as  $r \times c$  by 1 vector.
  - 3) One matrix will contain all the images - Each column of the matrix is one linearize (vectorized) image from the original faces dataset.

# *Eigenface Model*

- 1) The set of eigenfaces (also called eigenimages and eigenpictures) will form a **basis** for any search image.
- 2) The basis (or set of eigenfaces) can be linearly combined to reconstruct an estimate of any face from the original dataset.
- Example:
- $\text{Face}_{23} = .14 * \text{Eigenface}_1 + .56 * \text{Eigenface}_2 + .30 * \text{Eigenface}_3 + \text{mean}$
- NOTE: You must add back the mean that was subtracted initially.



*turn the entire faces dataset into one matrix of column vectors. Each column in the matrix is a face.*



Note: The size of each face does not have to be k by k, it can be a rectangle

# *the next steps*

- Assuming a proper dataset of faces, the dataset can be turned into **one large matrix**.
- Example:
  - 1) Suppose you have a dataset of 100 faces
  - 2) Suppose each face is 25 pixels by 30 pixels,
  - 3) Then, each facevector will be  $(25 \times 30)$  rows by 1 column
  - 4) The matrix of all faces will be:  $(25 \times 30)$  rows by 100 columns. Each column is a face.
  - 5) Given a face matrix, we need **to normalize the matrix** by **subtracting the mean**.
    - - Subtracting the mean will force each column (face) in the matrix to retain only individualized/unique features.
    - - Find the mean of all columns in the face matrix and then subtract the mean from all columns.

*example:*  
*my facematrix before subtracting the mean*

For this, I used 150 face images.

Each face image was 112 by 92 pixels.

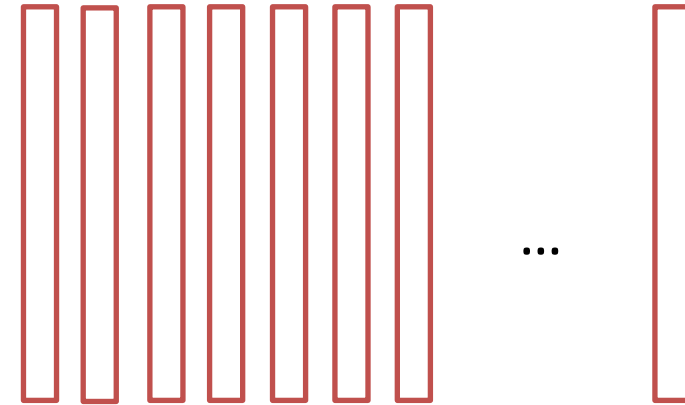
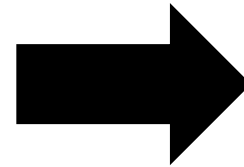
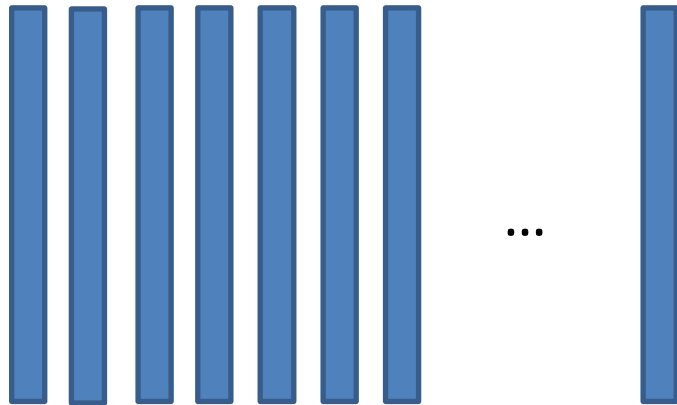
There are 256 (or  $2^8$ ) shades of grey in 8-bit grey scale.

```
The Face Matrix is
[[ 48  60  39 ...,  45 120 119]
 [ 49  60  44 ...,  38 119 121]
 [ 45  62  53 ...,  40 121 123]
 ...,
 [ 47  32  29 ..., 135  74  73]
 [ 46  34  26 ...,  24  74  74]
 [ 46  34  29 ...,  36  80  76]]
The shape of the Face Matrix is
(10304, 150)
```



# *normalizing the matrix*

**NOTE:** It does not matter whether you have each image as a column or as a row – BUT you must remember which you did.



**Calculate the mean** column using all columns.  
Subtract the mean from EACH column to create  
the normalized matrix.

Normalized face matrix

Subtract the mean

# *Average Image and Difference Images*

- The **average** of the face dataset is defined by

$$\mu = (1/m) \sum_{i=1}^m x_i$$

Note that  $\mu$  is a column vector that describes one face.

- Each face differs from the average by vector

$$r_i = x_i - \mu$$



This is the mean face from the dataset of 400 faces that I used.

# *The Covariance Matrix*

- A covariance matrix is constructed as

$$C = AA^T$$

where  $A$  is the normalized face matrix.

Recall that an image can be any size. The images that I used were 112 by 92.

(112 \* 92 = 10304) Therefore,  $C = AA^T$  is  $10304 * 10304 = 106,172,416$

- Finding eigenvectors for any an  $(r*c) \times (c*r)$  matrix is (or can be) intractable.

# *Using Alt to Cov Matrix*

- (Turk and Pentland 1991)
- The matrix  $A^T A$  of size  $c \times c$ , and find eigenvectors of this small matrix.
- Why?
  - For example, if you have 150 faces, this is  $150 \times 150 = 22,500$  (not 106,172,416)

# *why turk and pentland proposed method is correct?*

- The eigenvectors  $v_i$  of  $A^T A$  are based on:

$$A^T A v_i = u_i v_i$$

Then by pre-multiplying both sides by  $A$  (*the normalized face matrix*), we have the eigenvectors of the **original space**.

$$\mathbf{A} A^T A v_i = \mathbf{A} u_i v_i \quad \rightarrow \quad (\mathbf{A} A^T) (A v_i) = u_i (\mathbf{A} v_i)$$

- Therefore,  $\mathbf{A} v_1$  is an eigenvector (eigenface) for our face dataset and  $u_1$  is the eigenvalue.
- This matters because the Eigenvectors generated using Turk's  $A^T A$  will have to be projected back into our original space. We can do this by multiplying by  $A$ .

# *Learning the Model*

- 1) Get the data.
- 2) Transform each face into a column vector and place all into a face matrix.
- 3) Calculate the mean vector from all the faces and **render it so you can see it. It should look like a face!**
- 4) Normalize the face matrix by subtracting the mean from each column (face) in the matrix. (Call this normalized matrix, A)
- 5) Get the reduced version of the covariance matrix:  $C' = A^T A$
- 6) Find the eigenvalues and eigenvectors of  $C'$

# *Learning the Model*

- 7) Enable an option to choose the  $k$  highest eigenvalues and corresponding eigenvectors.
- 8) Transform the  $k$  highest eigenvectors back into the original space by multiplying each of them by  $A$ . ***These are called the eigenfaces and are the PCA reduced space.***

Recall that  $A$  is size  $(r \times c)$  by the number of faces in the dataset. (In our case  $A$  is shape:  $(10304, N)$ ),

$N$  is number of faces in the original dataset.

The eigenvectors of  $C' = A^T A$  will be the size of data set by 1. (In our case, is  $N$  by 1)

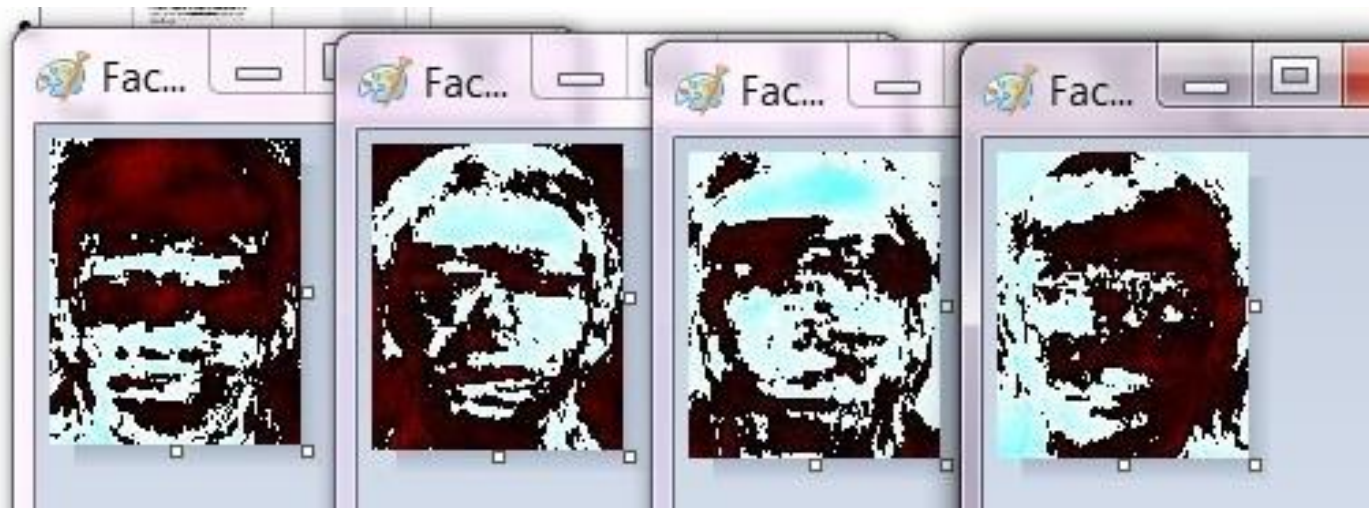
So,  $A * \text{eigenvector}$  will be  $(r \times c)$  by 1 which is exactly what we need. RE:  
 $(10304, N) * (N, 1) = (10304, 1)$

# *Learning the Model*

9) Render (view the image of) the k top eigenfaces just created, **AND save each to its own jpg file.**

10) **View the eigenfaces yourself** - they should look a bit like creepy faces. If they do not, you are doing something wrong

11) Now that you have the eigenfaces, its time to test your face-recognition ability.





# *Using the model*

- 1) Choose any face in the dataset and **remove it from the dataset**.
- 2) **Replace it with a copy of some other face** in the dataset to maintain a consistent dataset size and naming scheme, etc. Yes – there are other ways to do this
- 3) Run the program to generate  $k = 30$  eigenfaces.
- 4) Add code that will compare your test face with all other faces using the PCA reduced space.
  - Read in the test face and convert it to a vector.
  - Subtract the mean from the test face.
  - Project the normalized test face into the eigenfaces matrix reduced space.
  - Project each image in the face dataset (minus the mean) into the eigenfaces reduced space.
  - Use Euclidean distance to compare each projected dataset face to the test face. Choose the min.

## NOTES:

- Try this first by leaving the test face in the dataset. In this case, if your code works, you will get an exact match.

# *Using the model*

- 1) To test a face, you will have to read the face in, vectorize it, and subtract the mean from it.
- 2) Next, multiply the test face (minus the mean) by the eigenface matrix to project it into eigenface space (reduced space).
- 3) Recall that your test face is  $(10304, 1)$ . The eigenface matrix is  $(10304 \text{ by } k)$  because it contains  $k$  eigenfaces.  
Transpose the eigenface matrix and then multiply it by the test face vector. The resulting size and shape will be  $(k \text{ by } 1)$ .
- 4) Do this same exact process with all of the faces in the database (normalized).
- 5) Find the **least Euclidean distance** between your projected test face and all other projected faces in the dataset.

Recall that our faces are all  $112 \text{ by } 92 = 10304$ . This number will be different for different size face images.

# *Example Exercise:*

## *Using python and coding eigenfaces*

- **Step 1: Get the data**
- <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>
- - Look at the preview
- <http://www.cl.cam.ac.uk/research/dtg/attarchive/facesataglance.html>

The files are in PGM format, and can conveniently be viewed on UNIX (TM) systems using the 'xv' program. The size of each image is 112 pixels (rows) by 92 pixels (columns), with 256 grey levels per pixel. The images are organized in 40 directories (one for each subject), which have names of the form sX, where X indicates the subject number (between 1 and 40). In each of these directories, there are ten different images of that subject, which have names of the form Y.pgm, where Y is the image number for that subject (between 1 and 10).

# *Python Instructions: how to convert an image to a vector*

- `##Install PIL`
- `##On PC from cmd`
- `## anaconda search -t conda PIL`
- `## find a good option`
- `## I used:`
- `## conda install -c conda-forge pillow=4.0.0`
- `## RE`
- `#https://pypi.python.org/pypi/Pillow/3.3.1`
- `#Interesting Reference`
- `#http://www.pythonware.com/media/data/pil-handbook.pdf`
- **`import numpy as np`**
- **`from PIL import Image`**

# *Python:* *Vectorize the image*

```
fullpath=p+"\Face1.pgm" #Here p is the path to the image
img1=Image.open(fullpath).convert('L')
imagearray1 = np.array(img1)
original_shape=imagearray1.shape
flat1 = imagearray1.ravel()
facevector1 = np.matrix(flat1)
facematrix=facevector1
shape = flat1.shape
print(shape)
```

# *Python: Create Face Matrix*

```
for i in range(n-1):
    fullpath=p+"\Face"+str(i+2)+".pgm"
    print(fullpath)
    img=Image.open(fullpath).convert('L')
    imagearray = np.array(img)
    # make a 1-dimensional view of imagearray
    flat = imagearray.ravel()
    # convert it to a matrix
    facevector = np.matrix(flat)
    facematrix=np.r_[facematrix,facevector]
    print(facematrix, facematrix.shape)
    file_counter=file_counter+1
```

```
The facematrix is
[[48 49 45 ..., 47 46 46]
 [60 60 62 ..., 32 34 34]
 [39 44 53 ..., 29 26 29]
 [63 53 35 ..., 41 10 24]
 [64 76 80 ..., 35 37 39]]
The shape is
(5, 10304)
```

*transpose so that each column is an image*

```
facematrix_t=np.transpose(facematrix)
print(facematrix_t)
print(facematrix_t.shape)
```

```
[[48 60 39 63 64]
 [49 60 44 53 76]
 [45 62 53 35 80]
 ...,
 [47 32 29 41 35]
 [46 34 26 10 37]
 [46 34 29 24 39]]
The shape is
(10304, 5)
```

# *Python:*

## *How to transform a vector back to an image*

- `face_example =`  
`np.asarray(facematrix_t[:,0]).reshape(original_shape)`
- `# make a PIL image and save it to jpg`
- `face_example_img = Image.fromarray(face_example, 'L')`
- `face_example_img.show()`
- `face_example_img.save("FaceExampleOutput.jpg")`



- -----  
-----
- `#### Recall that original_shape is from here:`
- `#### fullpath=p+"\Face1.pgm"`
- `#### img1=Image.open(fullpath).convert('L')`
- `#### imagearray1 = np.array(img1)`
- `#### original_shape=imagearray1.shape`



# *Implementation Details:*

## *issues with rendering images*

- 1) There are many options for managing and rendering (viewing) images.
- 2) Our dataset contains .pgm images.
- 3) One option (that I think would have been easier) would be to first convert all images to jpg. However, I did not choose this option.
- 4) If you read in .pgm images, you can best do so with the “L” option. You can also render them with “L”.
- 5) However – to render images as .jpg successfully, such as the mean face vector, etc. I had to use RGBA.
- 6) You are welcome to work with the images in whatever way you wish – BUT – you must render them as .jpg or another format that is easy to view on most computers.

# *what happened so far?*

- 1) So far, we took 5 images from a large database (all .pgm).
- 2) We converted each of the 5 images into a vector.
- 3) We then placed all the 5 vectors into one matrix.
- 4) We transposed the matrix so that each column is an image.
- 5) We then took one of the columns and reversed the process to get an image back.

The overall goal here is to place a set of images into a matrix.

The next steps will include performing PCA on the matrix to get the eigenfaces.

Finally, we will use the eigenfaces for prediction.

## *perform PCA on the facematrix – step 1*

- 1) After creating a matrix of all the faces in the dataset – each column is a face:
- 2) Find the mean of the columns.
- 3) Subtract the mean from all columns.

!! Important !! This process can be done with a matrix such that each ROW is an image OR with a matrix such that each COLUMN is an image. The KEY is that you know and then code it correctly.

# *Python: eigenfaces*

1) Get the reduced covariance matrix based on Turk and Pentland:

```
Norm_Face_Matrix_t=np.transpose(Norm_Face_Matrix)  
CovMatrix=np.matmul(Norm_Face_Matrix_t, Norm_Face_Matrix)  
print("The covar matrix is\n",CovMatrix)
```

2) Get eigenvalues and eigenvectors

```
evals,evecs=np.linalg.eig(CovMatrix)  
print("The eigenvalues are\n",evals)  
print("The eigenvectors are\n",evecs)
```

# *Python*

- 1) Sort the eigenvalues
- 2) CHoose the top k eigenvectors based on the top k eigenvalues – these are called the eigenfaces.
- 3) Convert them back to original image space.
- 4) Render (view) and save into files all the eigenfaces. These should look like creepy faces.
- 5) Create an eigenface matrix.
- 6) Use the eigenface matrix to compare a test face with the dataset to identify the face.

NOTE: You may need to make some calculations on paper and/or to assure that your rows and columns are in the right place.

# *test face and prediction*

Test Face on Left



# Face Space: general example

- The eigenvectors of covariance matrix are

$$u_i = Av_i$$

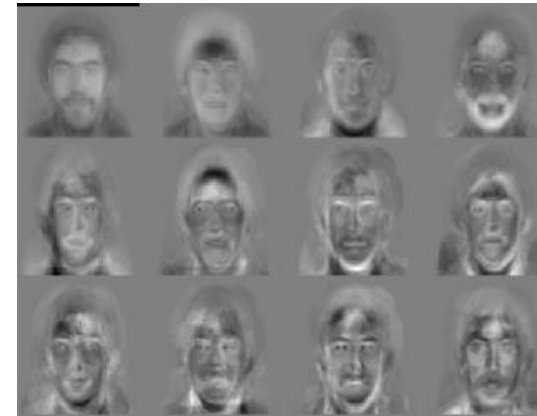
$$\begin{array}{ccc}
 \begin{bmatrix} -0.2621 \\ -0.2621 \\ -0.6527 \\ -0.1137 \\ -0.5589 \\ 0.6015 \\ -0.4895 \\ 0 \\ 0.6379 \end{bmatrix} &
 \begin{bmatrix} 0.3256 \\ 0.3256 \\ -3.3773 \\ 0.9922 \\ -1.0076 \\ -0.5080 \\ 2.3100 \\ 0 \\ -1.6434 \end{bmatrix} &
 \begin{bmatrix} -1.3511 \\ -1.3511 \\ 2.2735 \\ 1.0014 \\ -6.0561 \\ -5.4206 \\ 0.6517 \\ 0 \\ 1.7008 \end{bmatrix} \\
 u_1 & u_2 & u_3
 \end{array}$$

Face Space

↓

$$U = \begin{bmatrix} -0.2621 & 0.3256 & -1.3511 \\ -0.2621 & 0.3256 & -1.3511 \\ -0.6527 & -3.3773 & 2.2735 \\ -0.1137 & 0.9922 & 1.0014 \\ -0.5589 & -1.0076 & -6.0561 \\ 0.6015 & -0.5080 & -5.4206 \\ -0.4895 & 2.3100 & 0.6517 \\ 0 & 0 & 0 \\ 0.6379 & -1.6434 & 1.7008 \end{bmatrix}$$

$u_1$        $u_2$        $u_3$



- $u_i$  resemble facial images which look ghostly, hence called **Eigenfaces**

## *A few notes*

- Test  $k$  for a few values. The value of  $k$  (the number of eigenfaces) should be less than half of the size of the dataset. (Mine worked pretty well at  $k = 30$ ).
- Test your code at all steps using reduced dataset sizes, etc.
- Code this one step at a time and make sure that you can transform face images to vectors and vectors back to face images.
- I used numpy for all the linear alg and math.

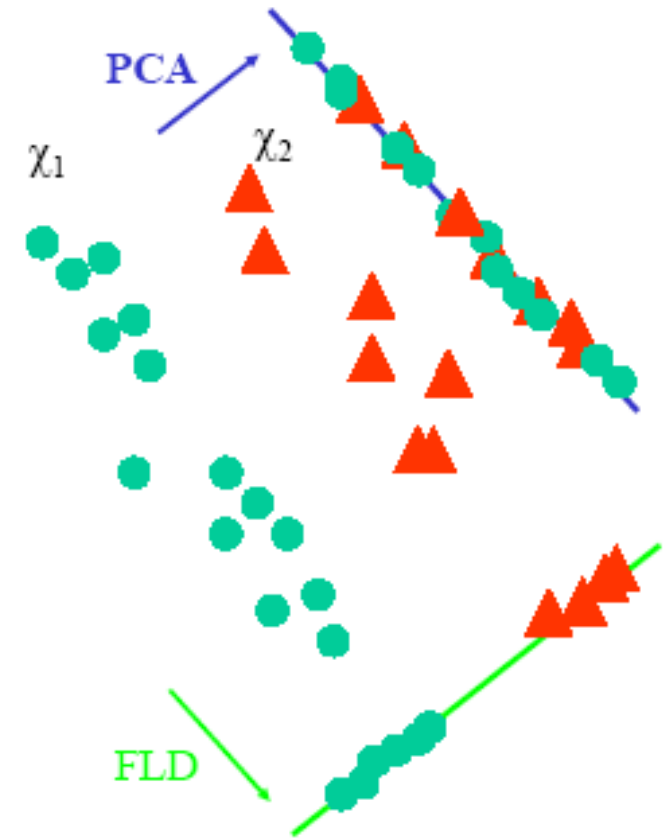


## *Other dim reduction methods*

- 1) LDA
- 2) ICA
- 3) Random Forest
- 4) Clustering
- 5) Regression
- 6) Binning

# Linear Discriminant Analysis

- PCA does not use class information
  - PCA projections are optimal for reconstruction from a low dimensional basis, they may not be optimal from a discrimination standpoint.
- LDA is an enhancement to PCA
  - Constructs a discriminant subspace that minimizes the scatter between images of same class and maximizes the scatter between different class images





## *Appendix*

Jeremy Bolton, PhD  
Assistant Teaching Professor

Special Thanks to A. Gates