Eigenfaces: Recovering Humans from Ghosts

Nev Acar · Aug 21, 2018 · 7 min read

Figure #1: Casper the Friendly Ghost (source: https://vignette.wikia.nocookie.net/christmasspecials/images/e/e8/Casper.jpg/revision/latest?cb=20121014033319)

Having more by having less… Easy! This machine learning method will help you store more information in smaller sizes with only some linear algebra fundamentals. By that, we will recover human faces from some ghost shaped messes.

You must have been told, independent vectors create spaces in abstract ways. Now, imagine for real. All the spaces are formed by the vectors and so are the things inside those spaces. Some of these vectors contribute to some of the things more than the others. The others may be negligible besides them and maybe we don't need to know about those because we basically don't care and don't want to carry their problems with us. This is exactly what I will show you to do about eigenfaces in this post.

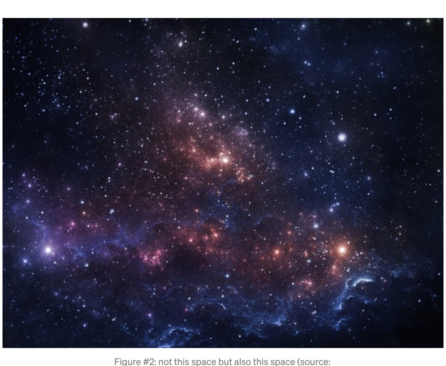

Figure #2: not this space but also this space (source: https://img.purch.com/w/660/aHR0cDovL3d3dy5saXZlc2NpZW5jZS5jb20vaW1hZ2VzL2kvMDAwLzA5Ny85NTcvb3JpZ2luYWwvc3BhY2UuanBn)

Our principle topic is PCA. It is a widely used algorithm in variety of subjects. Using it on faces makes it more human interpretable so it is one of the most popular applications. 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. Even though it was firstly used in 1991 by Turk and Pentland and has limitations compared to today's tech, its basics are still highly beneficial for new practices.

Figure #3: LFW (source: http://vis-www.cs.umass.edu/lfw/Six_Face_Panels_sm.jpg)

Before moving onto implementation, I'd like to give you a little introduction about basic concepts mentioned. Also, you can find full script and dataset (a small fraction of LFW_cropped) I used here.

Eigenvectors and Eigenvalues

We all know how eigenvectors and eigenvalues exactly work. Maybe not exactly but we've heard of them, like, many times. Here is an explanation for the minority like me who keep forgetting.

> *Let's have a square **A** matrix. Almost all vectors change direction when they are multiplied by **A** as we expect. However, there are certain vectors **x** which have the exact same direction as **Ax**. Soo… Guess what? Those are our eigenvectors. Special ones… Even if they are in the same direction, **Ax** and **x** still might be two different vectors by their magnitudes. Here is a familiar equality. $Ax=\lambda x$. What makes those two equal is λ, our magnitude equalizer scalar, eigenvalue. This will tell us how our eigenvector changed after the multiplication*

From above, you can tell why they are useful in practice. For a deeper understanding and probably appreciation go to MIT's Eigenvalues and Eigenvectors Chapter.

Principal Component Analysis (PCA)

The main goal of PCA is dimensionality reduction. It has many applications in visualisation, feature extraction, data compression, etc. The idea behind it is to linearly project original data onto a lower dimensional subspace offering the principal components (eigenvectors) maximum variance of the projected data and/or minimum distortion error from the projection. Eventually, both lead to the same result which is the best reconstruction formula. As a side note, this subspace is called principal subspace.

For calculations of maximizing variance and minimizing distortion error, you can go to ANU's lecture notes about PCA. Also, another video on youtube called Principal Components Analysis -Georgia Tech-Machine Learning helped me understand the concepts much better.

Implementation

Ok! Everyone ready for implementation. First, we'll import the required libraries. I assume you all know what they are for. I'll focus on the explanation part of algorithms. If you have any other questions about the code, leave a response below.

```
import matplotlib.pyplot as plt
import numpy as np
import os
```

import.py hosted with ❤ by GitHub · view raw

Then, we need a function to visualise the images from our dataset in a good-looking way. Here it is.

```
"""It helps visualising the portraits from the dataset."""
def plot_portraits(images, titles, h, w, n_row, n_col):
    plt.figure(figsize=(2.2 * n_col, 2.2 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.20)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i])
        plt.xticks(())
        plt.yticks(())
```

image_visualisation.py hosted with ❤ by GitHub · view raw

Let's put it on our dataset lfwcrop_grey.zip. Here, I only take first 1000 of the faces because of the computational complexity but it really doesn't matter. If you want to go full house then go for it. I also transformed images into numpy array (shaped 1000, 64, 64) for processing.

```
dir='lfwcrop_grey/faces'
celebrity_photos=os.listdir(dir)[1:1001]
celebrity_images=[dir+'/' + photo for photo in celebrity_photos]
images=np.array([plt.imread(image) for image in celebrity_images], dtype=np.float64)
celebrity_names=[name[:name.find('0')-1].replace("_", " ") for name in celebrity_photo
n_samples, h, w = images.shape
plot_portraits(images, celebrity_names, h, w, n_row=4, n_col=4)
```

portrait_visualisation.py hosted with ❤ by GitHub · view raw

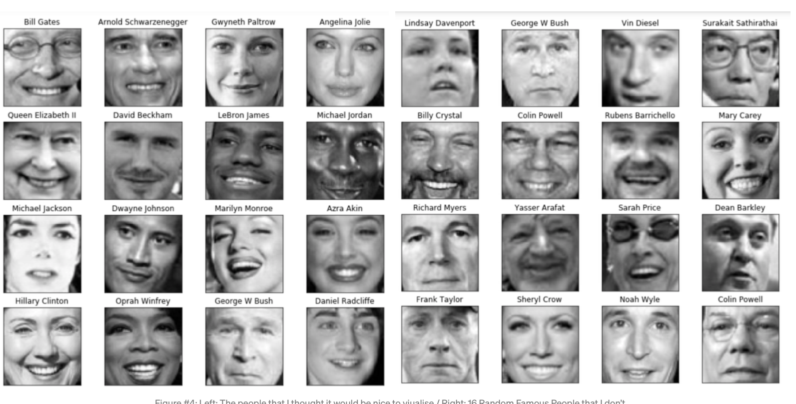

Figure #4: Left: The people that I thought it would be nice to viualise / Right: 16 Random Famous People that I don't actually care from the dataset of 1000

The table on the right is the one I will actually use. I basically visualised first 16 images from the dataset of 1000. The one on the left was a dataset that I collected from the original one when I had no idea what I was doing. Yet, I noticed it is better to have more data to explain the concepts more clearly.

Going on, the next step is to define PCA. Its arguments are X which is vectors of images and n_pc which is the number of principal components. As a little reminder, principal components define an orthonormal basis that can extract the maximum variance in the original data. In our case, its shape is (1000, 4096) since we needed to transform the images into vectors for PCA. Then, we find the mean and subtract it from our data to center it around the origin. After that, we need to perform Singular Value Decomposition on the centered data to find those principle components called eigenfaces.

$$X = USV^T$$

S matrix has the singular values along the diagonal while U and V are singular vectors

(SVD is difficult to get and much more complex than that. For a helpful illustration, go to that MIT OpenCourseWare video.)

```
def pca(X, n_pc):
    n_samples, n_features = X.shape
    mean = np.mean(X, axis=0)
    centered_data = X-mean
    U, S, V = np.linalg.svd(centered_data)
    components = V[:n_pc]
    projected = U[:,:n_pc]*S[:n_pc]

    return projected, components, mean, centered_data
```

pca.py hosted with ❤ by GitHub · view raw

Figure #5: mean face

By the way, here is our mean face of all 1000 faces in the dataset. We are trying to find the features that makes individuals different than the others. I hope subtracting the mean face from everyone and maximising variance make more sense now. Look at that. Carefully! It's a bit scary, isn't it?

The next steps are to manipulate our images to become vectors, call PCA function and manipulate our eigenfaces to become images again.

```
n_components = 50
X = images.reshape(n_samples, h*w)
P, C, M, Y= pca(X, n_pc=n_components)
eigenfaces = C.reshape((n_components, h, w))
eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_portraits(eigenfaces, eigenface_titles, h, w, 4, 4)
```

eigenfaces.py hosted with ❤ by GitHub · view raw

Figure #6: Bunch of ghost shaped images. Look at them in the eyes.

First parameter named n_components determines how many eigenfaces we want to create. Those are our principal components from which we will recover our data that is human face. Changing it won't change this table unless it is less than 16. However, I can guarantee there are only 50 images that we can trust rather than 1000 as it was the case at the beginning. (eigenfaces.shape = (50, 64, 64))

```
def reconstruction(Y, C, M, h, w, image_index):
    n_samples, n_features = Y.shape
    weights = np.dot(Y, C.T)
    centered_vector=np.dot(weights[image_index, :], C)
    recovered_image=(M+centered_vector).reshape(h, w)
    return recovered_image
recovered_images=[reconstruction(Y, C, M, h, w, i) for i in range(len(images))]
plot_portraits(recovered_images, celebrity_names, h, w, n_row=4, n_col=4)
```

reconstruction.py hosted with ❤ by GitHub · view raw

For reconstruction, we will use those eigenfaces. Each face is a weighted combination of those eigenfaces as you might say. To find weights, we should dot multiply our centered data and eigenfaces. Then, we should weight eigenfaces which gives us the centered face and add it up to the mean face again. After all, we are ready to take a look at the recovered faces compared to original ones.

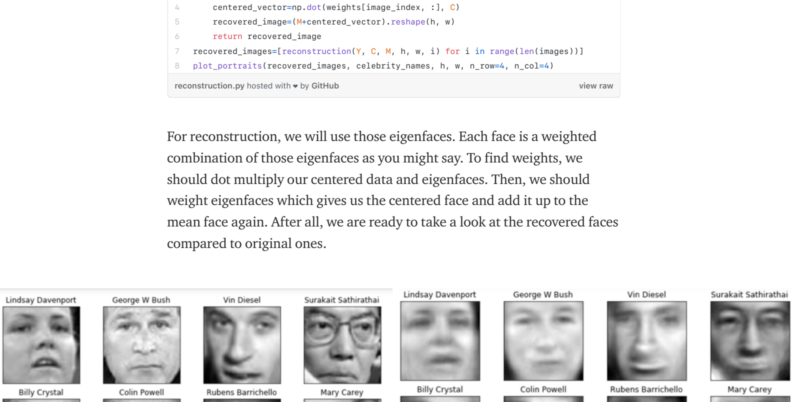

Figure #7: n_components=50

Wow! Not looking good! But, don't forget we came from 1000 x 64 x 64 to 50 x 64 x 64. In this way, we get to keep 20 times less data than before. Still, I won't fool anybody. Those are not good enough. It is clear we still have some important principal components we took out. Let's see what we can do. I'll try to use the first 100 components.

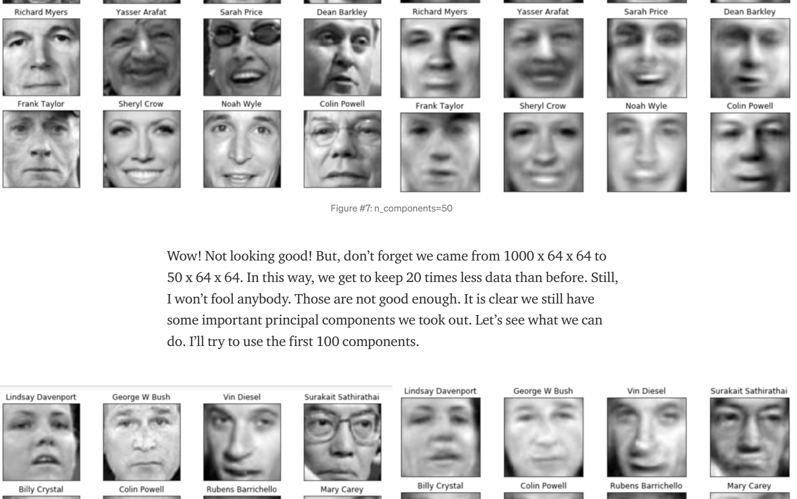

Figure #8: n_components=100

Getting better but still, not good enough. 250…

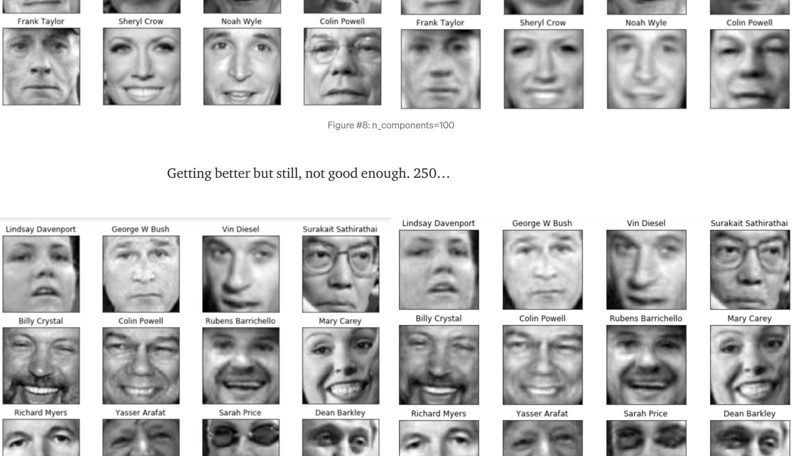

Figure #9: n_components=250

This one is totally satisfying for me. With only quarter of what we have we recovered totally distinguishable faces. It's totally pointless but you can actually use 1000 eigenface components and get the exact original results. Just to show you that, I'll run some computation and space complexity.

Still running… Hold on… It's coming.

The results have come. Pretty sweet!!!

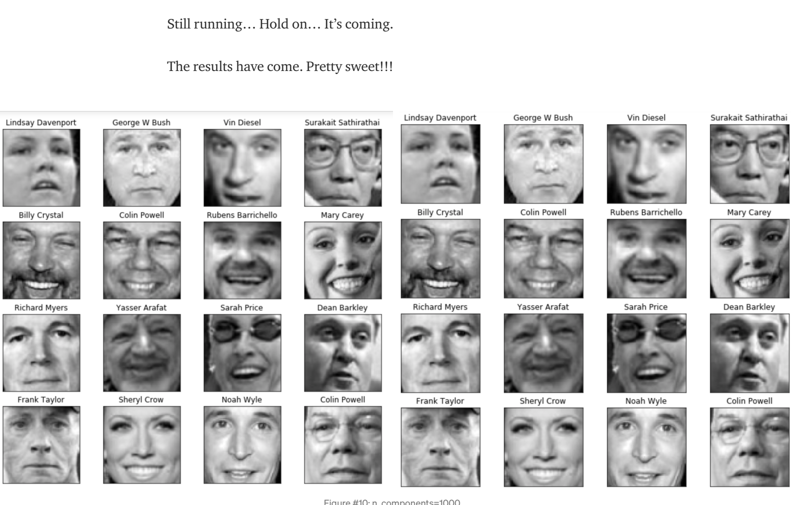

Figure #10: n_components=1000

This post actually showed you having good enough information with very small size of data and this is one of our most important expectations from machine learning. I hope you see something beyond this application and start thinking about new innovations. I stated the references as I used or above.

Once again, hope you enjoyed my post. Don't hesitate to ask your questions about it or give any type of feedback.

Cheers,

Nev

Sign up for The Variable
By Towards Data Science
Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.
Get this newsletter
Emails will be sent to ebollt@clarkson.edu. Not you?

285 · 3

Machine Learning · Data Science · Data Visualization · Programming · Artificial Intelligence

More from Towards Data Science
Follow
Your home for data science. A Medium publication sharing concepts, ideas and codes.
Read more from Towards Data Science