INFR10086 Machine Learning (MLG) Semester 1, 2022/23

# Tutorial 4: Principal Component Analysis (PCA)

In this tutorial, we will implement the PCA algorithm on a dataset of human faces. This is a common task in computer vision to demonstrate PCA, and its result is often called an "eigenface."

*Heads-up: this tutorial does require you to do coding!*

## 1 PCA, Eigenvectors, and the Covariance Matrix

PCA is a dimensionality reduction algorithm - it projects high-dimensional data onto a lower-dimensional space. This is usually done to improve computation speed, but it can also be used to get a better understanding of the data. This is because the axes of the lower-dimensional space point toward the dimensions of highest variability within the data. In other words, the first principal component points in the direction that maximises the variance of the data; the second is orthogonal to the first one and points in the remaining direction maximises the variance; etc.

It can be shown that the principal components in PCA are in fact the eigenvectors of the covariance matrix of the data. In this tutorial, we will be performing PCA on images of faces by getting the covariance matrix and using the Numpy function `numpy.linalg.eig` to get its eigenvectors.

## 2 Eigenfaces

We will be using the Yale face database. Download the code from the course homepage (primarily to get the dataset), and run the below code that shows a single one of the images.

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

img_1 = plt.imread("./yale_face_database/data/subject01.centerlight")
print(img_1.shape)
plt.imshow(img_1, cmap="gray")
plt.show()
```

Let's now load the entire dataset, and look at more of the images in the meantime.

```
data_path = './yale_face_database/data'

# X is the data matrix - each row is an image, and each column is a "feature" (i.e. pixel)
X = []

fig = plt.figure(figsize=(16, 6))
i=0
for _, _, filenames in os.walk(data_path):
    for filename in filenames:

        # Read a single image
        img = np.array(plt.imread(os.path.join(data_path, filename)))

        # Show every 5th image
        if i%5==0:
            ax = fig.add_subplot(4, 10, int(i/5) + 1, xticks=[], yticks=[])
            ax.imshow(img, cmap="gray")
        i+=1

        # Take all rows of the matrix "img" of shape (243, 320)
        # and put them into a single row of shape (243*320)
        img_row = img.flatten()

        # Add to data matrix
        X.append(img_row)

plt.savefig("./raw_faces.png")
X = np.array(X)
print(X.shape)  # The data matrix X is of shape (n, d)
```

**Discussion.** Look at the raw face images from running the above code (also shown in Figure 1). What do you think varies the most in them? What would you expect the principal components to be?

**Implementation.** Compute the "mean face" of the full dataset and visualise it. You could also look at the mean faces of specific persons in the dataset (each person has 11 images which are all together).

Figure 1: A grid showing every fifth image in the dataset.

## 2.1 Getting the Covariance Matrix

Now that we have the data matrix $X$, we need to compute its covariance matrix $\Sigma$. There is a common estimate for the covariance matrix that is $\Sigma = X^\top X - \mu^\top \mu$. However, that can be simplified by making the data be zero-mean.

**Implementation.** Compute a new data matrix `X_zero_mean` that is the original data matrix `X` but with its mean for each dimension (pixel) subtracted from it.

**Implementation.** Run the code "`print(np.mean(X_zero_mean, axis=0))`" to check if your new data matrix is indeed zero-mean.

**Discussion.** When running the above code to check, not every dimension's mean will likely be exactly 0 - why? Is this a problem?

**Implementation.** Now compute the simplified estimate of the covariance: $\Sigma = X^\top X$. What happens, and why?

Fortunately, since in this case we ultimately only want the eigenvectors of the covariance matrix rather than the matrix itself, we can avoid directly computing the covariance matrix. The eigenvalue decomposition of $\Sigma$ is given by:

$$\Sigma \mathbf{v}_i = X^\top X \mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{1}$$

Since $X^\top X$ is too big, let's instead look at the matrix $XX^\top$, which is much smaller - only $(165, 165)$ for our data. Its eigenvalue decomposition is:

$$XX^\top \mathbf{u}_i = \lambda'_i \mathbf{u}_i \tag{2}$$

Note that this has different eigenvectors and eigenvalues from the previous decomposition. However, we can multiply both sides of the equation by $X^\top$ from the left:

$$X^\top X X^\top \mathbf{u}_i = \lambda_i' X^\top \mathbf{u}_i \tag{3}$$

If we compare to equation 1, we can see that if $\mathbf{u}_i$ is an eigenvector of $XX^\top$, then $X^\top \mathbf{u}_i = \mathbf{v}_i$ is an eigenvector of $X^\top X = \Sigma$. In other words, we can find the eigenvectors of the smaller $XX^\top$, we can find the eigenvectors of the much bigger covariance matrix $\Sigma$.

**Discussion.** Comparing equations 1 and 3, they're not exactly the same - one has $\lambda_i$ and the other has $\lambda_i'$. Why is the above result still valid? Is $\lambda_i = \lambda_i'$?

## 2.2 The Eigenfaces / Principal Components

**Implementation.** Compute $XX^\top$, and using `np.linalg.eig()`, obtain its eigendecomposition, i.e. find its eigenvectors and eigenvalues. Print their values.

**Discussion.** Some of the eigenvalues and eigenvector elements are complex - they have non-zero imaginary parts. Why does that happen? What problems can that cause down the line? How would you solve this? Would simply using `np.real()` to remove imaginary values be good enough?

**Implementation.** Reuse the earlier plotting code to show the first 30 principal components. To do this, you need to project the eigenvectors of $XX^\top$ onto the eigenvectors of $\Sigma$.

**Discussion.** Observe the "eigenfaces" you obtained (the result is also shown in Figure 2). Is it what you expected? Do you think the first several of them properly capture the directions of highest variability in the data? Can you name what some of them roughly correspond to?

**Bonus Question.** If you think it might be interesting, you could compute the eigenfaces on a smaller part of the dataset, e.g. for specific persons only, or for a subset of the persons.

A further interesting question to ask is how much of the variance of the data do each of the principal components capture. This can be seen by looking at the eigenvalues - taking a component's eigenvalue and dividing it by the sum of all eigenvalues give the proportion of the overall variance that that component covers.

**Implementation.** Compute what percent of the overall variance is covered by the first component.

**Implementation.** Compute what percent of the overall variance is covered by the first 10 components.

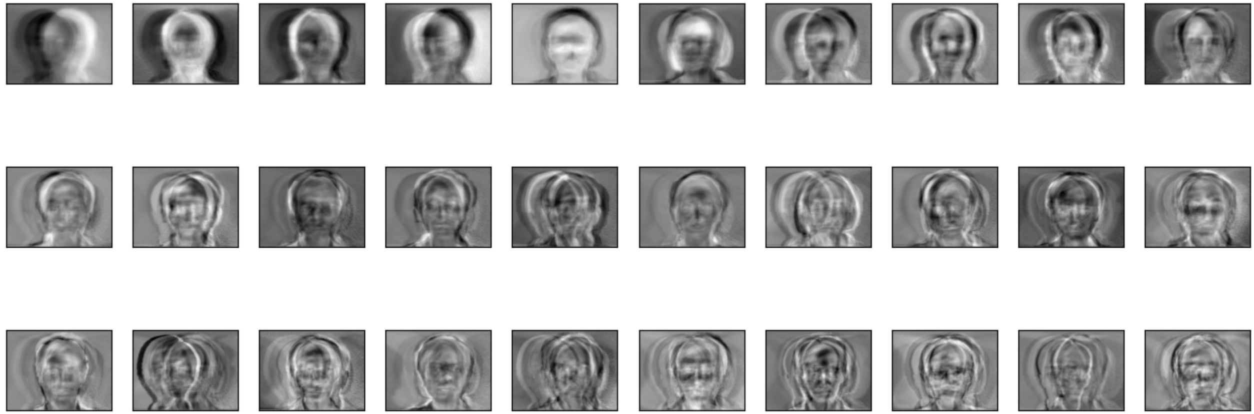

Figure 2: Eigenfaces. The largest principal component is at the top-left, and it goes to the right and then down.