Matrix Reconstruction using PCA

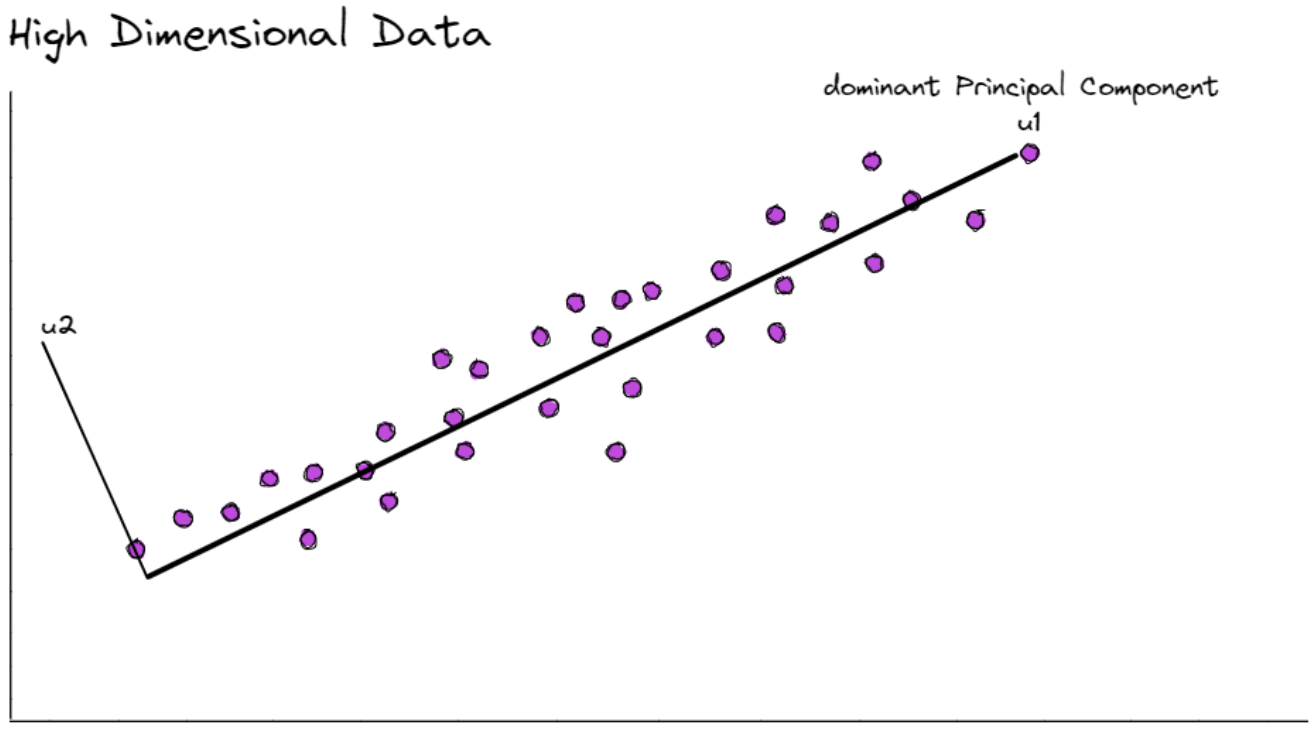
A Naive Approach

~ Shantanu Mane

1. Introduction

1.1. What is PCA?

PCA also known as Principal Component Analysis is a statistical technique used to reduce the dimensionality of a data using linear transformations of the eigen vectors of the covariance matrix of the data. It is a very popular technique used in data science and machine learning. It is used to reduce the dimensionality of the data by projecting it onto a lower dimensional subspace while retaining most of the information. It is also used to visualize high dimensional data.



1.2 Description of the Image

The above image contains data that is represented as the linear combination of `[1, 0]` and `[0, 1]`, which represent the x and y axes respectively. Therefore, x and y axes $\in \mathbb{R}^2$ can be considered as the initial basis of the data as all the data points can be represented as the linear combination of these two basis vectors.

The Principal Component of this data allows it to be represented in $\mathbb{R}^1$, from two components to a single component. BAM!!

1.3 Why PCA?

We represent data using fewer dimensions because :

1. It reduces the computational complexity of the algorithm.
2. Data is easier to visualize in lower dimensions.
3. Data has high variance along new dimensions.
4. Dimensions are **orthogonal** and **linearly independant** to each other.

Mathematical Description

Let $p_1, p_2, p_3 ... p_n$ be a set of orthonormal vectors who are also linearly independant and P be a matrix whose columns are the vectors $p_1, p_2, p_3 ... p_n$. Now, let us see why P is called the **Principal Component Matrix**...?

$$ P = \begin{bmatrix} p_1 & p_2 & p_3 & ... & p_n \end{bmatrix} $$

and,

$$ p_n = \begin{bmatrix} p_{n1} \\ p_{n2} \\ p_{n3} \\ ... \\ p_{nn} \end{bmatrix} $$

Therefore, P is a $n \times n$ matrix. We can also represent P as

$$ P = \begin{bmatrix} p_1 & p_2 & p_3 & ... & p_n \end{bmatrix} = \begin{bmatrix} p_{11} & p_{21} & p_{31} & ... & p_{n1} \\ p_{12} & p_{22} & p_{32} & ... & p_{n2} \\ p_{13} & p_{23} & p_{33} & ... & p_{n3} \\ ... & ... & ... & ... & ... \\ p_{1n} & p_{2n} & p_{3n} & ... & p_{nn} \end{bmatrix}_{n\times n} $$

and, let $x_1, x_2, x_3 ... x_m$ be a set of m data points and X be the matrix such that each column of X is a data point x_i.

$$ X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_m \end{bmatrix} $$

and,

$$ x_i = \begin{bmatrix} x_{i1} & x_{i2} & x_{i3} & ... & x_{in} \end{bmatrix} $$

Therefore, X is a $n \times m$ matrix. We can also represent X as $$ X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_m \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1m} \\ x_{21} & x_{22} & x_{23} & ... & x_{2m} \\ x_{31} & x_{32} & x_{33} & ... & x_{3m} \\ ... & ... & ... & ... & ... \\ x_{n1} & x_{n2} & x_{n3} & ... & x_{nm} \end{bmatrix}_{n\times m} $$

Now, we can represent the data points x_i as the linear combination of the basis vectors p_j as

$$ x_i = \sum_{j=1}^{n} \alpha_{ij} p_j $$

where, α_{ij} is the coefficient of the basis vector p_j in the data point x_i, and since p_j is orthonormal, α_{ij} is the projection of x_i onto p_j. Therefore, we can represent α_{ij} as $p_j x_i^T$.

A Little Example

Let us consider a basis in $\mathbb{R}^2$ as $p_1 = [1, 0]^T$ and $p_2 = [0, 1]^T$. Therefore, matrix P can be represented as

$$ P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} $$

and let us consider a few data points in $\mathbb{R}^2$ as $x_1 = [1, 2]$, $x_2 = [3, 4]$, $x_3 = [5, 6]$... $x_m = [x, y]$ Therefore, matrix X can be represented as

$$ X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ ... & ... \\ x & y \end{bmatrix} $$

Now, we can represent the data points x_i as the linear combination of the basis vectors p_j as

$$ x_i = \alpha_{i1} p_1 + \alpha_{i2} p_2

$$

and,

$$ \alpha_{i1} = p_1 x_i^T = [1, 0]^T \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix} $$

Using PCA for Image Compression

Let us consider m images of size $n \times n$ and let us represent each image as a vector of size $n^2 \times 1$. Therefore, we can represent the matrix X as

$$

x_i = \begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1n} \\ x_{21} & x_{22} & x_{23} & ... & x_{2n} \\ x_{31} & x_{32} & x_{33} & ... & x_{3n} \\ ... & ... & ... & ... & ... \\ x_{n1} & x_{n2} & x_{n3} & ... & x_{nn} \end{bmatrix}_{n \times n} $$

we can flatten this image to a vector of size $n^2 \times 1$ as

$$ x_i = \begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1n} & ... & ... & ... & ... & ... & x_{n1} & x_{n2} & x_{n3} & ... & x_{nn} \end{bmatrix}_{n^2 \times 1} $$

$$ X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_m \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1n^2} \\ x_{21} & x_{22} & x_{23} & ... & x_{2n^2} \\ x_{31} & x_{32} & x_{33} & ... & x_{3n^2} \\ ... & ... & ... & ... & ... \\ x_{m1} & x_{m2} & x_{m3} & ... & x_{mn^2} \end{bmatrix}_{n^2 \times m} $$

now, we can find a basis P such that the data points x_i can be represented as the linear combination of the basis vectors p_j as

$$ x_i = \sum_{j=1}^{n^2} \alpha_{ij} p_j $$

Practical Example

Consider we are given a large set of images (m images). Each image is of size 100×100 pixels. Therefore, each image can be represented as a vector of size 10000×1. Now, we can represent the matrix X as

$$ X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_m \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1n^2} \\
x_{21} & x_{22} & x_{23} & ... & x_{2n^2} \\ x_{31} & x_{32} & x_{33} & ... & x_{3n^2} \\ ... & ... & ... & ... & ... \\ x_{m1} & x_{m2} & x_{m3} & ... & x_{mn^2} \end{bmatrix}_{m \times 100^2}, n = 100 $$

to make it more understandable, we can also represent X as

$$ X = \begin{bmatrix} & ... & image_1 & ... & \\ & ... & image_2 & ...& \\ & ... & image_3 & ...& \\ & ... & ... & ...& \\ & ... & image_m & ... &\\ \end{bmatrix}_{m \times 100^2} $$

We now need to find a basis P such that the data points x_i can be represented as the linear combination of the basis vectors p_j. We can find this basis by using PCA. We can find the covariance matrix of X as

$$ \begin{align} \text{Cov}(X) &= \frac{1}{m} X^T X \\ &= \frac{1}{m} \begin{bmatrix}

& ... & image_1 & ... & \\ & ... & image_2 & ...& \\ & ... & image_3 & ...& \\ & ... & ... & ...& \\ & ... & image_m & ... &\\ \end{bmatrix}*{100^2 \times m} \begin{bmatrix} & ... & image_1 & ... & \\ & ... & image_2 & ...& \\ & ... & image_3 & ...& \\ & ... & ... & ...& \\ & ... & image_m & ... &\\ \end{bmatrix}*{m \times 100^2} \\ &= \frac{1}{m} \begin{bmatrix} & ... & image_1^T & ... & \\ & ... & image_2^T & ...& \\ & ... & image_3^T & ...& \\ & ... & ... & ...& \\ & ... & image_m^T & ... &\\ \end{bmatrix}*{100^2 \times 100^2} \\ &= \frac{1}{m} \begin{bmatrix} & ... & \sum*{i=1}^{m} image_1^T image_1 & ... & \\

& ... & \sum_{i=1}^{m} image_2^T image_2 & ... & \\ & ... & \sum_{i=1}^{m} image_3^T image_3 & ... & \\ & ... & ... & ... & \\ & ... & \sum_{i=1}^{m} image_m^T image_m & ... & \\ \end{bmatrix}_{100^2 \times 100^2} \end{align} $$

herein, we can also get rid of the covariance coefficient $\frac{1}{m}$ as it does not affect the eigenvectors of the covariance matrix. Therefore, we can find the eigenvectors of the resultant X^TX matrix.

> **Note:** The eigenvectors of the covariance matrix are the same as the eigenvectors of the correlation matrix. shape of X^TX is $100^2 \times 100^2$

What will be the shape of P?

Yes, the shape of P will be $100^2 \times 100^2$. But for image compression, we will only use the top k eigenvectors.

Solving a Toy Problem

Let there be a database that has 4 images only. Each image is of size 2×2 pixels. Therefore, each image can be represented as a vector of size 4×1.

$$ image_1 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}_{2 \times 2}

image_2 = \begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix}_{2 \times 2}

\\ $$

$$ \\ image_3 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}_{2 \times 2}

image_4 = \begin{bmatrix} 2 & 1 \\ 0 & 0 \end{bmatrix}_{2 \times 2} $$

Now, we can represent the matrix X as

$$ X = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 2 & 1 & 3 \\ 1 & 0 & 0 & 2 \\ 2 & 1 & 0 & 0 \end{bmatrix}_{4 \times 4} $$

where each row represents an image. We can find the covariance matrix of X as

$$ \begin{align} &= X^T X \\ &= \begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 2 & 1 & 3 \\ 1 & 0 & 0 & 2 \\ 2 & 1 & 0 & 0 \end{bmatrix}*{4 \times 4} \begin{bmatrix} 1 & 0 & 1 & 2 \ 1 & 2 & 0 & 1 \ 0 & 1 & 0 & 0 \ 2 & 3 & 2 & 0 \end{bmatrix}*{4 \times 4} \\ &= \begin{bmatrix} 6 & 3 & 0 & 4 \\ 3 & 6 & 2 & 8 \\ 0 & 2 & 1 & 3 \\ 4 & 8 & 3 & 17 \end{bmatrix}_{4 \times 4} \end{align} $$

Now let us find the eigen vectors of this X matrix, which will be

$$ \text{Eigenvectors of } X^TX = \begin{bmatrix} 0.27301976 & 0.92726137 & 0.20892004 & 0.14832053 \\ 0.45339458 & 0.07678322 & -0.35563715 & -0.81367064 \\ 0.15355822 & -0.19732821 & 0.90997894 & -0.3307866 \\ 0.83445398 & -0.30879163 & -0.04257885 & 0.45444618 \end{bmatrix}_{4 \times 4} $$

$$ \text{Eigenvalues of } X^TX = \begin{bmatrix} 23.20754468 \\ 4.91636072 \\ 0.07798872 \\ 1.79810588 \end{bmatrix}_{4 \times 1} $$

Here, each column represents an eigenvector. We can see that the first eigenvector is the most dominant one. The second eigenvector is the second most dominant one and so on. We can also see that the first eigenvector is the most dominant one as it has the highest eigenvalue. The second eigenvector is the second most dominant one as it has the second-highest eigenvalue and so on.

We can also term these eigenvectors as `eigen_images` as any image can be represented as a linear combination of these `eigen_images`. One of the `eigen_images` can be represented as

$$ EigenImage_1 = \begin{bmatrix} 0.27301976 & 0.92726137 \\ 0.20892004 & 0.14832053 \\ \end{bmatrix}_{2 \times 2} $$

Now, let's reconstruct the first image using all the eigen images. We can do this by using α and multiplying it with the `eigen_images`. The α can be calculated as

$$ image_1 = \alpha_1EigenImage_1 + \alpha_2EigenImage_2 + \alpha_3EigenImage_3 + \alpha_4EigenImage_4 $$

and as we previously saw, the α can be calculated as

$$ \alpha = EigenImage_1 \times image_1^T $$

Therefore, we can replace the α in the above equation as

$$ image_1 = EigenImage_1 \times image_1^T \times EigenImage_1 \\ + EigenImage_2 \times image_1^T \times EigenImage_2 \\ + EigenImage_3 \times image_1^T \times EigenImage_3 \\ + EigenImage_4 \times image_1^T \times EigenImage_4 $$

We can also reduce the number of eigen images to be used for reconstruction. For example, if we want to use only the first 2 eigen images, then we can replace the above equation as

$$ image_1 = EigenImage_1 \times image_1^T \times EigenImage_1 \\ + EigenImage_2 \times image_1^T \times EigenImage_2 $$

we will validate this formula by writing a code for it. Till then we will highlight the forecoming equation.

$$ image_1 = \begin{bmatrix} 0.27301976 & 0.92726137 \\ 0.20892004 & 0.14832053 \\ \end{bmatrix}*{2 \times 2} \times \begin{bmatrix} 1 & 1 \ 0 & 2 \end{bmatrix}*{2 \times 2}^T \times \begin{bmatrix} 0.27301976 & 0.92726137 \\ 0.20892004 & 0.14832053 \\ \end{bmatrix}_{2 \times 2}\\

-

\begin{bmatrix} 0.45339458 & 0.07678322 \\ -0.35563715 & -0.81367064 \\ \end{bmatrix}*{2 \times 2} \times \begin{bmatrix} 1 & 1 \ 0 & 2 \end{bmatrix}*{2 \times 2}^T \times \begin{bmatrix} 0.45339458 & 0.07678322 \\ -0.35563715 & -0.81367064\\ \end{bmatrix}_{2 \times 2}\\

-

\begin{bmatrix} 0.15355822 & -0.19732821 \\ 0.90997894 & -0.3307866 \\ \end{bmatrix}*{2 \times 2} \times \begin{bmatrix} 1 & 1 \ 0 & 2 \end{bmatrix}*{2 \times 2}^T \times \begin{bmatrix} 0.15355822 & -0.19732821 \\ 0.90997894 & -0.3307866 \\ \end{bmatrix}_{2 \times 2}\\

-

\begin{bmatrix} 0.83445398 & -0.30879163 \\ -0.04257885 & 0.45444618 \\ \end{bmatrix}*{2 \times 2} \times \begin{bmatrix} 1 & 1 \ 0 & 2 \end{bmatrix}*{2 \times 2}^T \times \begin{bmatrix} 0.83445398 & -0.30879163 \\ -0.04257885 & 0.45444618 \\ \end{bmatrix}_{2 \times 2}\\ $$

PS: Much to your surprise, the above equation does give the same result as the original image.

Code [Python]

First we will create our X matrix. Let us define an array X whose rows are our four images. We will use the images from the previous section.

```
X = np.array([[1, 1, 0, 2], [0, 2, 1, 3], [1, 0, 0, 2], [2, 1, 0, 0]])
```

Now let's calculate our X^TX matrix.

```
XTX = X.T.dot(X)
```

```
>>> XTX
array([[ 6  3  0  4]
 [ 3  6  2  8]
 [ 0  2  1  3]
 [ 4  8  3 17]])
```

Now let's find the eigen vectors of this X matrix.

```
eVa, eVe = np.linalg.eig(XTX)
```

```
>>> eVa
[23.20754468  4.91636072  0.07798872  1.79810588]
>>> eVe
array([[ 0.27301976  0.92726137  0.20892004  0.14832053]
 [ 0.45339458  0.07678322 -0.35563715 -0.81367064]
 [ 0.15355822 -0.19732821  0.90997894 -0.3307866 ]
 [ 0.83445398 -0.30879163 -0.04257885  0.45444618]])
```

Now let's try and see how our reshaped eigen vector look like.

```
eVe[0].reshape(2, 2)
```

```
>>> array([[0.27301976, 0.92726137],
 [0.20892004, 0.14832053]])
```

Now, finally let's reconstruct our image using all the eigen vectors.

```
image = np.zeros((2, 2))
for i in eVe:
    image += i.reshape(2,2).dot(X[0].reshape(2,2).T).dot(i.reshape(2, 2))

print(image)
```

```
>>> [[1.0000000e+00  1.0000000e+00]
 [-8.8817842e-16  2.0000000e+00]]
```

as you can see our result is very close to the original image. Now let's try and reconstruct our image using only the first two eigen vectors.

```
image = np.zeros((2, 2))
for i in eVe[:2]:
    image += i.reshape(2,2).dot(X[0].reshape(2,2).T).dot(i.reshape(2, 2))

print(image)
```

```
>>> [[0.90091324 1.30379438]
 [0.20809319 1.60958995]]
```

we see a similar image, but the error is very high. This is because we are using only two eigen vectors. Let's try and use three eigen vectors.

```
image = np.zeros((2, 2))
for i in eVe[:3]:
    image += i.reshape(2,2).dot(X[0].reshape(2,2).T).dot(i.reshape(2, 2))

print(image)
```

```
>>>[[ 0.53506298  1.44297848]
 [-0.30498474  1.71413851]]
```

> The sudden rise in loss from 2 eigen vectore to 3 eigen vectors despite more eigen vectors suggest that the third eigen vector might not be more dominant than the second and the fourth eigen vector.