

Dimensionality Reduction with PCA

Posted Sep 25, 2022

By [Davide](#)

9 min read

[Evangelista](#)

Dimensionality Reduction

While working with data, it is common to have access to very high-dimensional unstructured informations (e.g. images, sounds, ...). To work with them, it is necessary to find a way to project them into a low-dimensional space where data which is semantically similar is close. This approach is called **dimensionality reduction**.

For example, assume our data can be stored in an $d \times N$ array, $X = [x^1 x^2 \dots x^N] \in \mathbb{R}^{d \times N}$ where each datapoint $x^j \in \mathbb{R}^d$. The idea of dimensionality reduction techniques in ML is to find a projector operator $P : \mathbb{R}^d \rightarrow \mathbb{R}^k$, with $k \ll d$, such that in the projected space $P(x)$, images semantically similar are close together. If the points in a projected space forms isolated populations such that *inside* of each population the points are close, while the distance *between* populations is large, we call them **clusters**. A clusering algorithm is an algorithm which is able to find clusters from high-dimensional data.

Principal Component Analysis (PCA)

Principal Componenti Analysis (PCA) is probably the simplest yet effective technique to perform dimensionality reduction and clustering. It is an unsupervised algorithm, thus it does not require any label.

The idea is the following: consider a dataset $X \in \mathbb{R}^{d \times N}$ of high-dimensional data and assume we want to project it into a low-dimensional space \mathbb{R}^k . Define

$$Z = [z^1 z^2 \dots z^N] \in \mathbb{R}^{k \times N}$$

the projected version of X . We want to find a matrix $P \in \mathbb{R}^{k \times d}$ such that $Z = PX$, with the constraint that in the projected space we want to keep as much information as possible from the original data X .

You already studied that, when you want to project a matrix by keeping informations, a good idea is to use the Singular Value Decomposition (SVD) of it and, in particular, the Truncated SVD (TSVD). Let $X \in \mathbb{R}^{d \times N}$, then

$$X = U \Sigma V^T$$

is the SVD of X , where $U \in \mathbb{R}^{d \times d}$, $V \in \mathbb{R}^{N \times N}$ are orthogonal matrices ($U^T U = U U^T = I$ and $V V^T = V^T V = I$), while $\Sigma \in \mathbb{R}^{d \times N}$ is a diagonal matrix whose diagonal elements σ_i are the singular values of X , in decreasing order ($\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_d$). Since the singular values represents the *quantity of informations* contained in the corresponding singular vectors, keeping the first k singular values and vectors can be the solution to our projection problem. Indeed, given $k < d$, we define the Truncated SVD of X as

$$X_k = U_k \Sigma_k V_k^T$$

where $U_k \in \mathbb{R}^{d \times k}$, $\Sigma_k \in \mathbb{R}^{k \times k}$, and $V_k \in \mathbb{R}^{k \times N}$.

The PCA use this idea and defines the projection matrix as $P = U_k$, and consequently,

$$Z = U_k^T X$$

is the projected space. Here, the columns of U_k are called **feature vectors**, while the columns of Z are the **principal components** of X .

Implementation of PCA (Pseudocode)

Centroid: Given a set $X = [x^1 x^2 \dots x^N]$ its *centroid* is defined to be $c(X) = \frac{1}{N} \sum_{i=1}^N x^i$

Thus, the implementation of PCA is as follows:

- Consider the dataset X ;
- Compute the centered version of X as $X_c = X - c(X)$, where the subtraction between matrix and vector is executed column-by-column;
- Compute the SVD of X_c , $X_c = U\Sigma V^T$;
- Given $k < n$, compute the Truncated SVD of X_c : $X_{c,k} = U_k \Sigma_k V_k^T$;
- Compute the projected dataset $Z_k = U_k^T X_c$;

Python example

For the following example, I'm assuming you downloaded the MNIST dataset, either from Virtuale or from Kaggle ([click here](#) do download it), renamed it `data.csv` and placed it in the same folder of the Python file on which you are working.

The first step is to load the dataset into memory, this can be done with Pandas as we saw into the [introductory post](#).

🔗 Plaintext



```
import numpy as np
import pandas as pd

# Load data into memory
data = pd.read_csv('data.csv')
```

After that, it is important to inspect the data, i.e. to look at its structure and understand how it is distributed. This can be done either by reading at the documentation of the website where the data has been downloaded or by using the `pandas` method `.head()`.

🔗 Plaintext



```
# Inspect the data
print(f"Shape of the data: {data.shape}")
print("")
print(data.head())
```

which prints out all the columns of `data` and the first 5 rows of each column. With this command, we realize that our dataset is a 42000×785 frame, where the columns from the second to the last are the pixels of an image representing an handwritten digit, while the first column is the *target*, i.e. the integer describing the represented digit.

Following the [introductory post on Machine Learning](#), first of all we convert the dataframe into a matrix with `numpy`, and then we split the input matrix X and the corresponding target vector Y . Finally, we note that X is an $N \times d$ matrix, where $N = 42000$ and $d = 784$. Since in our notations the shape of X **must** be $d \times N$, we have to transpose it.

🔗 Plaintext



```
# Convert data into a matrix
data = np.array(data)

# Split data into a matrix X and a vector Y where:
#
# X is dimension (42000, 784)
# Y is dimension (42000, )
# Y is the first column of data, while X is the rest
X = data[:, 1:]
X = X.T

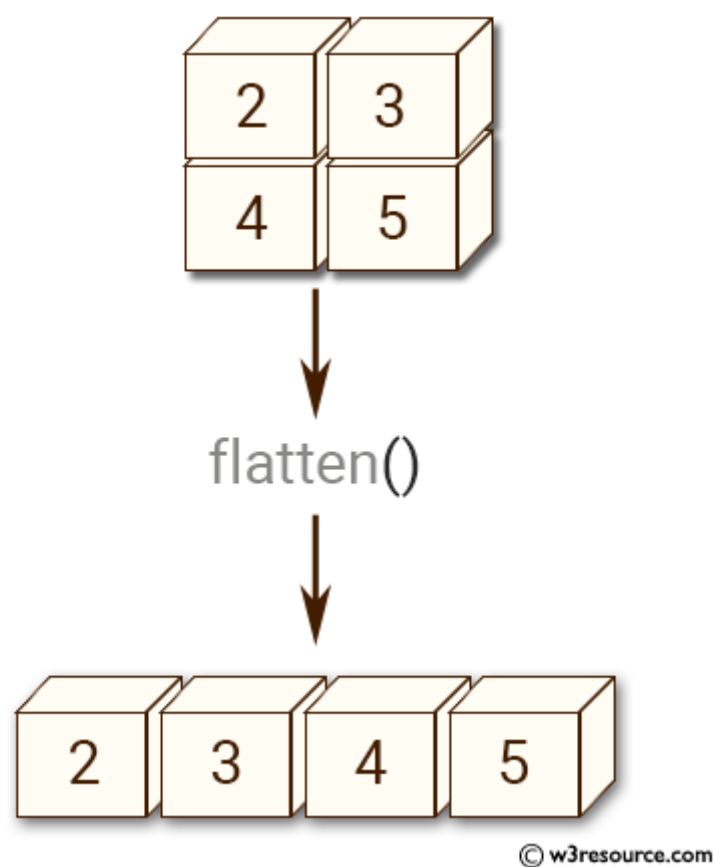
Y = data[:, 0]

print(X.shape, Y.shape)

d, N = X.shape
```

Visualizing the digits

We already said that X is a dataset of images representing handwritten digits. We can clearly visualize some of them. In the documentation, we can read that each datapoint is a 28×28 grey-scale image, which has been flattened. Flattening is the operation of taking a 2-dimensional array (a matrix) and converting it to a 1-dimensional array, by concatenating the rows of it. This can be implemented in `numpy` with the function `a.flatten()`, where `a` is a 2-dimensional numpy array.



Since we know that the dimension of each image was 28×28 before flattening, we can invert this procedure by **reshaping** them. After that, we can simply visualize it with the function `plt.imshow()` from `matplotlib`, by setting the `cmap` to `'gray'` since the images are grey-scale.

```
import matplotlib.pyplot as plt

def visualize(X, idx):
    # Visualize the image of index 'idx' from the dataset 'X'

    # Load an image in memory
    img = X[:, idx]

    # Reshape it
    img = np.reshape(img, (28, 28))

    # Visualize
    plt.imshow(img, cmap='gray')
    plt.show()

# Visualize image number 10 and the corresponding digit.
idx = 10
visualize(X, idx)
print(f"The associated digit is: {Y[idx]}")
```

Splitting the dataset

Before implementing the algorithm performing PCA, you are required to split the dataset into a training set and a test set, following the [introductory post on Machine Learning](#). Remember that to correctly split X , it has to be random.

🔗 Plaintext



```
def split_data(X, Y, Ntrain):
    d, N = X.shape

    idx = np.arange(N)
    np.random.shuffle(idx)

    train_idx = idx[:Ntrain]
    test_idx = idx[Ntrain:]

    Xtrain = X[:, train_idx]
    Ytrain = Y[train_idx]

    Xtest = X[:, test_idx]
    Ytest = Y[test_idx]

    return (Xtrain, Ytrain), (Xtest, Ytest)

# Test it
(Xtrain, Ytrain), (Xtest, Ytest) = split_data(X, Y, 30_000)

print(Xtrain.shape, Xtest.shape)
```

Now the dataset (X, Y) is divided into the train and test components, and you can implement the PCA algorithm on X_{train} . Remember to not access X_{test} during training.

A note on SVD

In Python, the (reduced) SVD decomposition of a matrix A can be implemented as

🔗 Plaintext



```
# SVD of a matrix A
U, s, VT = np.linalg.svd(A, full_matrices=False)
```

where $U \in \mathbb{R}^{d \times k}$, $VT \in \mathbb{N} \times \mathbb{K}$, with $k = \text{rank}(A)$, $s \in \mathbb{R}^k$ is a vector containing the k non-zero singular values of A , $\sigma_1, \dots, \sigma_k$, in decreasing order.

Visualizing clusters in Python

When $k = 2$, it is possible to visualize clusters in Python. In particular, we want to plot the datapoints, with the color of the corresponding class, to check how well the clustering algorithm performed in 2-dimensions. This can be done by the `matplotlib` function `plt.scatter`. In particular, if $Z = [z^1 z^2 \dots z^N] \in \mathbb{R}^{2 \times N}$ is the projected dataset and $Y \in \mathbb{R}^N$ is the vector of the corresponding classes, then

Plaintext



```
# Visualize the clusters
plt.scatter(z[0, :], z[1, :], c=Y)
plt.show()
```

will do the job.

Working example

Here I report a working example on how to visualize clusters on `matplotlib` with the function `plt.scatter()`. In the example, we generate two clusters, composed by normally distributed datapoints in 2-dimensions, from two different gaussian distributions. The first class is labelled as 0, while the second one is labelled as 1. The datapoints are colored into the scatterplot with a different color for each class.

Plaintext



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

# Create two cloud of points in 2-dimensional space
# and plot it, divided by color
x1 = np.random.normal(0, 1, (2, 1000))
x2 = np.random.normal(3, 0.1, (2, 100))

y1 = np.zeros((1000, ))
y2 = np.ones((100, ))

# Visualize them (coloring by class)

# Join together x1 x2, y1 y2
X = np.concatenate((x1, x2), axis=1)
Y = np.concatenate((y1, y2))

# Visualize
plt.scatter(X[0, :], X[1, :], c=Y)
plt.show()
```

This is the result:

This post is licensed under [CC BY 4.0](#) by the author.

Share:    

Further Reading

[Sep 27, 2022](#)

[Classification with clustering algorithms](#)

[What is classification? Consider, as an example, the MNIST dataset we used to te...](#)

[Sep 26, 2022](#)

[Clustering with Linear Discriminant Analysis \(LDA\).](#)

[Why PCA is not sufficient? Since PCA is an unsupervised learning technique, the lack...](#)

[Sep 24, 2022](#)

[A \(very short\) introduction to Machine Learning](#)

[Definition: Machine Learning \(ML\) is the set of all the techniques and algorithms able t...](#)

OLDER

[A \(very short\) introduction to Machine Learning](#)

NEWER


[Clustering with Linear Discriminant Analysis \(LDA\)](#)

0 Comments - powered by [utteranc.es](#)

Write

Preview

Sign in to comment

 Styling with Markdown is supported

Sign in with GitHub

