Principal Component Analysis

Implemente from scratch and validate with sklearn framework

Introduction :

“Excess of EveryThing is Bad”

The above line is specially in machine learning. When the data becomes too much in its dimension then it becomes a problem for pattern learning. Too much information is bad on two things : compute and execution time and quality of the model fit. When the dimension of the data is too high we need to find a way to reduce it. But that reduction has to be done in such a way that we maintain the original pattern of the data. The algorithm that we are going to discuss in this article does the similar job. The algorithm is quite famous and widely used in varieties of tasks. Its name is Principal Component Analysis aks PCA.

The main purposes of a principal component analysis are the analysis of data to identify patterns and finding patterns to reduce the dimensions of the dataset with minimal loss of information.

PCA is used to transform a high-dimensional dataset into a smaller-dimensional subspace ;into a new coordinate system.In the new coordinate system, the first axis corresponds to the first principal component, which is the component that explains the greatest amount of the variance in the data.

In simple words, principal component analysis is a method of extracting important variables known as principal components from a large set of variables available in a data set. It captures as much information as possible from the original high dimensional data. It represents the original data in terms of its principal components in a new dimension space.

What are Principal components :

Principal components are the underlying structure in the data. They are the directions where there is the most variance, the directions where the data is most spread out. There are multiple principle components of a data each representing different variance of the data.They are arranged in a chronological order of variance. The first PC will capture the most variance i.e. the most information about the data followed by second,third and so on.

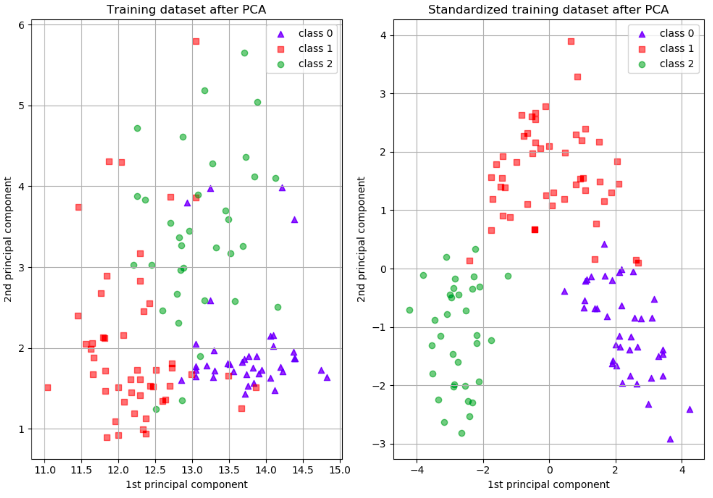
Mathematical Explanation :

Mathematically, the principal components are the eigenvectors of the symmetric correlation or covariance matrix of the original dataset. This means the matrix should be numeric and have standardized data. Eigenvectors of real symmetric matrices are orthogonal.The principal components (eigenvectors) correspond to the direction (in the original n-dimensional space) with the greatest variance in the data.

Each eigenvector has a corresponding eigenvalue. An eigenvalue is a scalar. Recall that an eigenvector corresponds to a direction. The corresponding eigenvalue is a number that indicates how much variance there is in the data along that eigenvector (or principal component). A larger eigenvalue means that that principal component explains a large amount of the variance in the data. A principal component with a very small eigenvalue does not do a good job of explaining the variance in the data.

Tips before doing PCA :

When performing PCA, it is typically a good idea to standardize the data first. Because PCA seeks to identify the principal components with the highest variance, if the data are not properly standardize, attributes with large values and large variances (in absolute terms) will end up dominating the first principal component when they should not. standardize the data gets each attribute onto more or less the same scale, so that each attribute has an opportunity to contribute to the principal component analysis.



When should you use PCA?

It is often helpful to use a dimensionality-reduction technique such as PCA prior to performing machine learning because:

* Reducing the dimensionality of the dataset reduces the size of the space on which k-nearest-neighbors (kNN) must calculate distance, which improve the performance of kNN.
* If your learning algorithm is too slow because the input dimension is too high, then using PCA to speed it up can be a reasonable choice. This is probably the most common application of PCA.
* Reducing the dimensionality of the dataset reduces the number of degrees of freedom of the hypothesis, which reduces the risk of overfitting.
* Reducing the dimensionality via PCA can simplify the dataset, facilitating description, visualization, and insight.
* Visualising the data in lower dimension is much intuitive than a higher dimension. PCA finds an important application in cases where the data of higher dimension needs a good visual representation.

Let’s try doing PCA on a random generated dataset. We will implement things from scratch. Then we will also use the implementation from sklearn.decomposition module.

Summarizing the PCA approach Listed below are the 6 general steps for performing a principal component analysis, which we will investigate in the following sections.

1. Take the entire dataset
2. Normalize columns of A so that each feature has zero mean
3. Compute sample covariance matrix Σ=AT x A/(m−1)
4. Perform eigen-decomposition of Σ using np.linalg.eig(Sigma)
5. Compress by ordering k eigenvectors according to largest eigenvalues and compute Axk
6. Reconstruct from compressed version by computing Axk x k.T

**Import the necessary libraries**

|  |
| --- |
| %matplotlib inline  import numpy as np  import matplotlib.pyplot as plt  from scipy import stats  # use seaborn plotting style defaults  import seaborn as sns; sns.set() |

**Take the entire dataset**

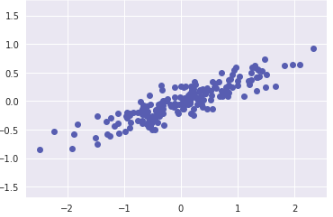
We will generate a random dataset on the fly.

|  |
| --- |
| np.random.seed(1)  A0 = (np.random.random(size=(2, 2)) @ np.random.normal(size=(2, 200))).T  print(A0.shape)  (200, 2) |

We have got 200 rows of 2-D vectors stored in a matrix.

Let’s visualise the generated data :

|  |
| --- |
| plt.plot(A0[:, 0], A0[:, 1], 'o')  plt.axis('equal'); |



Normalize columns of A0 so that each feature has zero mean

|  |
| --- |
| mu = np.mean(A0,axis=0)  A = A0 - mu  print(np.mean(A,axis=0))  [-2.44249065e-17 -1.11022302e-18] |

Does A have zero mean across rows? Yes they are pretty close to zero(notice the e-17/18 at the end).

Compute sample covariance matrix Σ=AT x A/(m−1)

|  |
| --- |
| # 2. Compute sample covariance matrix $\Sigma = {A^TA}/{(m-1)}$  m,n = A.shape  Sigma = (A.T @ A)/(m-1)  print("---")  print("Sigma:")  print(Sigma)  Sigma: [[0.68217761 0.23093475]  [0.23093475 0.09883179]] |

Perform eigen-decomposition of Σ using np.linalg.eig(Sigma)

Decompose the covariance matrix into eigen vectors and eigen values.

|  |
| --- |
| l,X = np.linalg.eig(Sigma)  print("---")  print("Evalues:")  print(l)  print("---")  print("Evectors:")  print(X)  --- Evalues: [0.7625315 0.0184779] --- Evectors: [[ 0.94446029 -0.32862557]  [ 0.32862557 0.94446029]] |

Compress by ordering k eigenvectors according to largest eigenvalues and compute Axk

|  |
| --- |
| # Compress by ordering $k$ evectors according to largest evalues and compute $AX\_k$  print("---")  print("Compressed - 2D to 1D:")  Acomp = A @ X[:,:1] # first 2 evectors  print(Acomp[:5,:]) # first 5 observations  --- Compressed - 2D to 1D: [[-0.67676923]  [ 1.07121393]  [-0.72791236]  [-2.30964136]  [-0.63005232]] |

We have successfully compressed the 2-D dataset into a 1-D data.

Reconstruct from compressed version

We can reconstruct the data back by using inverse transformation mathematically represented by Axk x k.T

|  |
| --- |
| # 5. Reconstruct from compressed version by computing $A X\_k X\_k^T$  print("---")  print("Reconstructed version - 1D to 2D:")  Arec = A @ X[:,:1] @ X[:,:1].T # first 2 evectors  print(Arec[:5,:]+mu) # first 5 obs, adding mu to compare to original  Reconstructed version - 1D to 2D: [[-0.60566999 -0.22648439]  [ 1.0452307 0.34794757]  [-0.65397264 -0.24329133]  [-2.14785286 -0.76308793]  [-0.56154772 -0.21113202]] |

**Validate the implementation with PCA from sklearn.decomposition**

|  |
| --- |
| from sklearn.decomposition import PCA  pca = PCA(n\_components=1) # two components  pca.fit(A0) # run PCA, putting in raw version for fun  print("Principal components:")  print(pca.components\_)  print("---")  print("Compressed - 4D to 2D:")  print(pca.transform(A0)[:5,:]) # first 5 obs  print("---")  print("Reconstructed - 2D to 4D:")  print(pca.inverse\_transform(pca.transform(A0))[:5,:]) # first 5 obs  Principal components: [[-0.94446029 -0.32862557]] --- Compressed - 2D to 1D: [[ 0.67676923]  [-1.07121393]  [ 0.72791236]  [ 2.30964136]  [ 0.63005232]] --- Reconstructed - 1D to 2D: [[-0.60566999 -0.22648439]  [ 1.0452307 0.34794757]  [-0.65397264 -0.24329133]  [-2.14785286 -0.76308793]  [-0.56154772 -0.21113202]] |

We can see the same set of compressed vectors and decompressed vectors.

Applications of PCA :

* Compression
* Visualisation
* Speeding up Machine Learning Algorithms
* Reducing Noise from the data

We will cover the applications in upcoming blogs.