**Principal Component Analysis (PCA) with Python Examples — Tutorial**

**An in-depth tutorial on principal component analysis (PCA) with mathematics and Python coding examples**

Last updated, January 8, 2021

**Author(s):** Saniya Parveez, [Roberto Iriondo](https://mktg.best/vguzs)

**This tutorial’s code is available on** [**Github**](https://github.com/towardsai/tutorials/tree/master/principal_component_analysis) **and its full implementation as well on** [**Google Colab**](https://colab.research.google.com/drive/12wfYg3mHOvVw2TZk3SPsC4naMJhcMkA2?usp=sharing)**.**

**Table of Contents**

1. [Introduction](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#f641)
2. [Curse of Dimensionality](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#82ac)
3. [Dimensionality Reduction](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#2000)
4. [Correlation and its Measurement](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#08f8)
5. [Feature Selection](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#7295)
6. [Feature Extraction](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#85da)
7. [Linear Feature Extraction](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#7524)
8. [Principal Component Analysis (PCA)](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#5e9a)
9. [Math behind PCA](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#993b)
10. [How does PCA work?](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#3628)
11. [Applications of PCA](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#8885)
12. [Implementation of PCA with Python](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#5d0a)
13. [Conclusion](https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa#a31f)

📚 Check out our [convolutional neural networks](https://towardsai.net/p/deep-learning/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f) tutorial with Python. 📚

**Introduction**

When implementing [**machine learning algorithms**](https://towardsai.net/p/machine-learning/machine-learning-algorithms-for-beginners-with-python-code-examples-ml-19c6afd60daa), the inclusion of more features might lead to worsening performance issues. Increasing the number of features will not always improve classification accuracy, which is also known as the curse of dimensionality. Hence, we apply dimensionality reduction to improve classification accuracy by selecting the optimal set of lower dimensionality features.

Principal component analysis (PCA) is essential for data science, [**machine learning**](https://mld.ai/mldcmu), data visualization, statistics, and other quantitative fields.

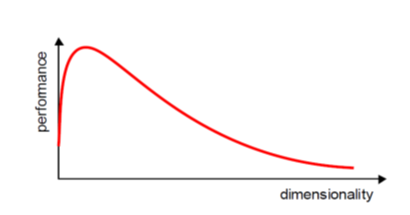


Figure 1: Curse of dimensionality.

There are two techniques to make dimensionality reduction:

* Feature Selection
* Feature Extraction

It is essential to know about vector, matrix, and transpose matrix, eigenvalues, eigenvectors, and others to understand the concept of dimensionality reduction.

**Curse of Dimensionality**

Dimensionality in a dataset becomes a severe impediment to achieve a reasonable efficiency for most algorithms. Increasing the number of features does not always improve accuracy. When data does not have enough features, the model is likely to underfit, and when data has too many features, it is likely to overfit. Hence it is called the curse of dimensionality. The curse of dimensionality is an astonishing paradox for data scientists, based on the exploding amount of n-dimensional spaces — as the number of dimensions, n, increases.

**Sparseness**

The sparseness of data is the property of being scanty or scattered. It lacks denseness, and its high percentage of the variable’s cells do not contain actual data. Fundamentally full of “empty” or “N/A” values.

Points in an n-dimensional space frequently become sparse as the number of dimensions grows. The distance between points will extend to grow as the number of dimensions increases.

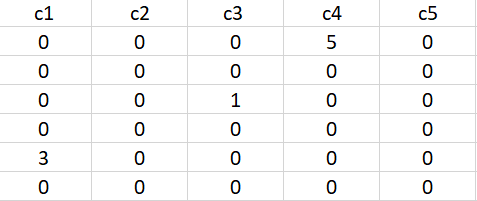


Figure 2: Data sparseness.

**Implications of the Curse of Dimensionality**

There are few implications of the curse of dimensionality:

* Optimization problems will be infeasible as the number of features increases.
* Due to the absolute scale of inherent points in an n-dimensional space, as n maintains to grow, the possibility of recognizing a particular point (or even a nearby point) proceeds to fall.

**Dimensionality Reduction**

Dimensionality reduction eliminates some features of the dataset and creates a restricted set of features that contains all of the information needed to predict the target variables more efficiently and accurately.

Reducing the number of features normally also reduces the output variability and complexity of the learning process. The covariance matrix is an important step in the dimensionality reduction process. It is a critical process to check the correlation between different features.

**Correlation and its Measurement**

There is a concept of correlation in machine learning that is called multicollinearity. Multicollinearity exists when one or more independent variables highly correlate with each other. Multicollinearity makes variables highly correlated to one another, which makes the variables’ coefficients highly unstable [[8](https://online.stat.psu.edu/stat501/book/export/html/981)].

The coefficient is a significant part of regression, and if this is unstable, then there will be a poor outcome of the regression result. Multicollinearity is confirmed by using Variance Inflation Factors (VIF). Therefore, if multicollinearity is suspected, it can be checked using the variance inflation factor (VIF).

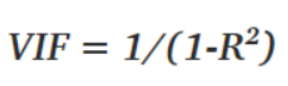


Figure 3: VIF equation.

Rules from VIF:

* A VIF of 1 would indicate complete independence from any other variable.
* A VIF between 5 and 10 indicates a very high level of collinearity [[4](https://online.stat.psu.edu/stat462/node/180/)].
* The closer we get to 1, the more ideal the scenario for predictive modeling.
* Each independent variable regresses against each independent variable, and we calculate the VIF.

Heatmap also plays a crucial role in understanding the correlation between variables.

The type of relationship between any two quantities varies over a period of time.

Correlation varies from **-1** to **+1**

To be precise,

* Values that are close to +1 indicate a positive correlation.
* Values close to -1 indicate a negative correlation.
* Values close to 0 indicate no correlation at all.

Below is the heatmap to show how we will correlate which features are highly dependent on the target feature and consider them.

**The Covariance Matrix and Heatmap**

The covariance matrix is the first step in dimensionality reduction because it gives an idea of the number of features that strongly relate, and it is usually the first step in dimensionality reduction because it gives an idea of the number of strongly related features so that those features can be discarded.

It also gives the detail of all independent features. It provides an idea of the correlation between all the different pairs of features.

**Identification of features in Iris dataset that are strongly correlated**

Import all the required packages:

import numpy as np  
import pandas as pd  
from sklearn import datasets   
import matplotlib.pyplot as plt

Load Iris dataset:

iris = datasets.load\_iris()  
iris.data

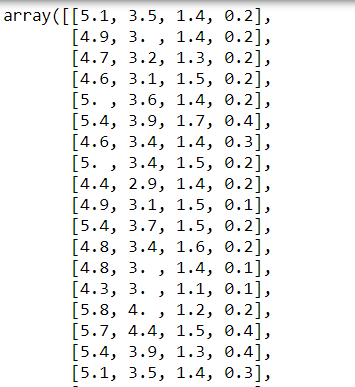


Figure 4: Iris dataset.

List all features:

iris.feature\_names

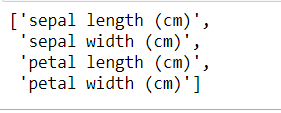


Figure 5: Features of the Iris dataset.

Create a covariance matrix:

cov\_data = np.corrcoef(iris.data.T)cov\_data

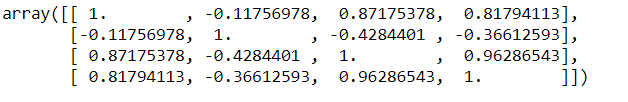


Figure 6: Covariance matrix of the Iris dataset.

Plot the covariance matrix to identify the correlation between features using a heatmap:

img = plt.matshow(cov\_data, cmap=plt.cm.rainbow)  
plt.colorbar(img, ticks = [-1, 0, 1], fraction=0.045)for x in range(cov\_data.shape[0]):  
 for y in range(cov\_data.shape[1]):  
 plt.text(x, y, "%0.2f" % cov\_data[x,y], size=12, color='black', ha="center", va="center")  
   
plt.show()

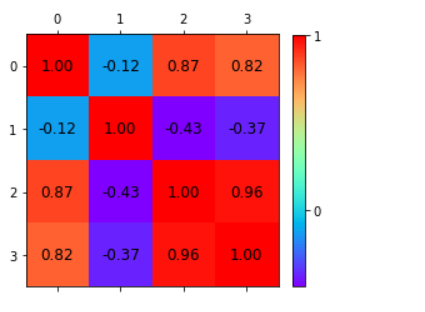


Figure 7: Heatmap of the correlation matrix.

A correlation from the representation of the heatmap:

* Among the **first** and the **third** features.
* Between the **first** and the **fourth** features.
* Between the **third** and the **fourth** features.

Independent features:

* The **second** feature is almost independent of the others.

Here the correlation matrix and its pictorial representation have given the idea about the potential number of features reduction. Therefore, two features can be kept, and other features can be reduced apart from those two features.

There are two ways of dimensionality reduction:

* Feature Selection
* Feature Extraction

Dimensionality Reduction can ignore the components of lesser significance.

**Feature Selection**

In feature selection, usually, a subset of original features is selected.

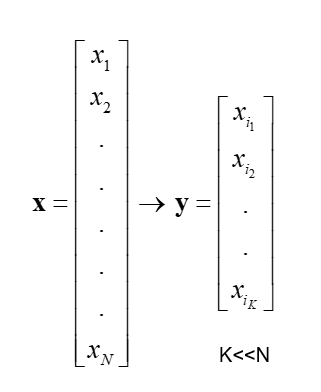


Figure 8: Feature selection.

**Feature Extraction**

In feature extraction, a set of new features are found. That is found through some mapping from the existing features. Moreover, mapping can be either linear or non-linear.

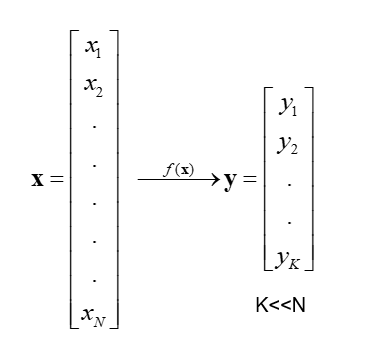


Figure 9: Feature Extraction.

**Linear Feature Extraction**

Linear feature extraction is straightforward to compute and analytically traceable.

Widespread linear feature extraction methods:

* **Principal Component Analysis (PCA)**: It seeks a projection that preserves as much information as possible in the data.
* **Linear Discriminant Analysis (LDA)**:- It seeks a projection that best discriminates the data.

**Principal Component Analysis (PCA)**

Principal Component Analysis (PCA) is an exploratory approach to reduce the data set's dimensionality to 2D or 3D, used in exploratory data analysis for making predictive models. Principal Component Analysis is a linear transformation of data set that defines a new coordinate rule such that:

* The highest variance by any projection of the data set appears to laze on the first axis.
* The second biggest variance on the second axis, and so on.

We can use principal component analysis (PCA) for the following purposes:

* To reduce the number of dimensions in the dataset.
* To find patterns in the high-dimensional dataset
* To visualize the data of high dimensionality
* To ignore noise
* To improve classification
* To gets a compact description
* To captures as much of the original variance in the data as possible

In summary, we can define **principal component analysis (PCA)** as the transformation of any high number of variables into a smaller number of uncorrelated variables called principal components (PCs), developed to capture as much of the data’s variance as possible.

PCA was invented in 1901 by Karl Pearson and Harold Hotelling as an analog of the Principal axis theorem [[1](https://zenodo.org/record/1430636#.X6HvO4hKguU)] [[2](https://pdfs.semanticscholar.org/e0be/f0bd8e07de281230ae5df28daabb4047e8f0.pdf)] [[3](https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.325.1383&rep=rep1&type=pdf)].

Mathematically the main objective of PCA is to:

* Find an orthonormal basis for the data.
* Sort dimensions in the order of importance.
* Discard the low significance dimensions.
* Focus on uncorrelated and Gaussian components.

**Steps involved in PCA**

* Standardize the PCA.
* Calculate the covariance matrix.
* Find the eigenvalues and eigenvectors for the covariance matrix.
* Plot the vectors on the scaled data.

**Example of a problem where PCA is required**

There are 100 students in a class with *m* different features like grade, age, height, weight, hair color, and others.

Most of the features may not be relevant that describe the student. Therefore, it is vital to find the critical features that characterize a student.

Some analysis based on the observation of different features of a student:

* Every student has a vector of data that defines him the length of m. e.g. (height, weight, hair\_color, grade,….) or (181, 68, black, 99, ….).
* Each column is one student vector. So, n = 100.
* It creates an *m\*n* matrix.
* Each student lies in an m-dimensional vector space.

**Features to Ignore**

* Collinear features or linearly dependent features. e.g., leg size and height.
* Noisy features that are constant. e.g., the thickness of hair
* Constant features. e.g., Number of teeth.

**Features to Keep**

* Non-collinear features or low covariance.
* Features that change a lot, high variance. e.g., grade.

**Math Behind PCA**

It is essential to understand the mathematics involved before kickstarting PCA. Eigenvalues and eigenvector play important roles in PCA.

**Eigenvectors and eigenvalues**

The eigenvectors and eigenvalues ​​of a covariance matrix (or correlation) describe the source of the PCA. Eigenvectors (main components) determine the direction of the new attribute space, and eigenvalues ​​determine its magnitude.

The PCA’s main objective is to reduce the data’s dimensionality by projecting it into a smaller subspace, where the eigenvectors form the axes. However, the eigenvectors define only the new axes’ directions because they all have a size of **1**. Consequently, to decide which eigenvector(s), we can discard without losing much information in the subspace construction and checking the corresponding eigenvalues. The eigenvectors with the highest values ​​are the ones that include more information about the distribution of our data.

**Covariance Matrix**

The classic PCA approach calculates the covariance matrix, where each element represents the covariance between two attributes. The covariance between two attributes is calculated as shown below:

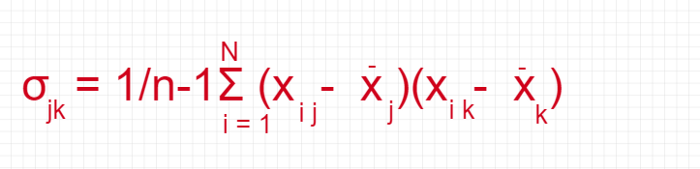


Figure 10: The equation to calculate the covariance between two attributes.

Create a matrix:

import pandas as pd  
import numpy as npmatrix = np.array([[0, 3, 4], [1, 2, 4], [3, 4, 5]])   
matrix

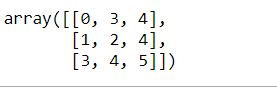


Figure 11: Matrix.

Convert matrix to covariance matrix:

np.cov(matrix)

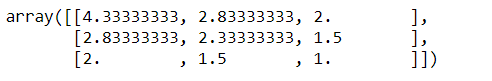


Figure 12: Covariance matrix.

An exciting feature of the covariance matrix is ​​that the sum of the matrix’s main diagonal is equal to the eigenvalues’ sum.

**Correlation Matrix**

Another way to calculate eigenvalues ​​and eigenvectors is by using the correlation matrix. Although the matrices are different, they will result in the same eigenvalues ​​and eigenvectors (shown later) since the covariance matrix's normalization gives the correlation matrix.



Figure 13: Equation of the correlation matrix.

Create a matrix:

matrix\_a = np.array([[0.1, .32, .2, 0.4, 0.8],   
 [.23, .18, .56, .61, .12],   
 [.9, .3, .6, .5, .3],   
 [.34, .75, .91, .19, .21]])

Convert to correlation matrix:

np.corrcoef(matrix\_a.T)

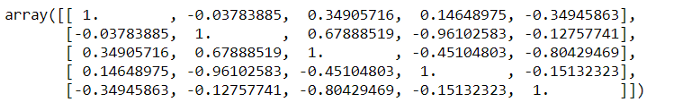


Figure 14: Correlation matrix.

**How does PCA work?**

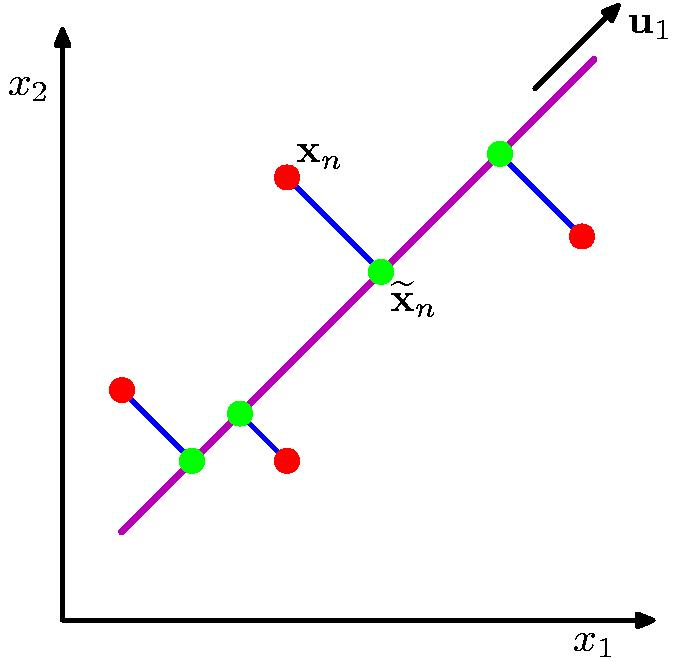


Figure 15: Working with PCA [[5](https://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/tutorial7.pdf)].

The orthogonal projection of data from high dimensions to lower dimensions such that (from figure 15):

* Maximizes the variance of the projected line (purple)
* Minimizes the MSE between the data points and projections (blue)

**Applications of PCA**

These are the typical applications of PCA:

* Data Visualization.
* Data Compression.
* Noise Reduction.
* Data Classification.
* Image Compression.
* Face Recognition.

**Implementation of PCA with Python**

Implementation of principal component analysis (PCA) on the Iris dataset with Python:

Load Iris dataset:

import pandas as pd  
import numpy as np  
from sklearn.datasets import load\_iris  
from sklearn.preprocessing import StandardScaleriris = load\_iris()  
df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)df['class'] = iris.target  
df

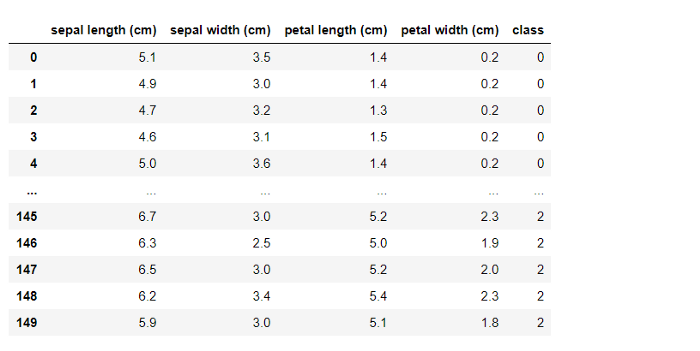


Figure 16: Iris dataset.

Get the value of x and y:

x = df.drop(labels='class', axis=1).values  
y = df['class'].values

Implementation of PCA with a covariance Matrix:

class convers\_pca():  
 def \_\_init\_\_(self, no\_of\_components):  
 self.no\_of\_components = no\_of\_components  
 self.eigen\_values = None  
 self.eigen\_vectors = None  
   
 def transform(self, x):  
 return np.dot(x - self.mean, self.projection\_matrix.T)  
   
 def inverse\_transform(self, x):  
 return np.dot(x, self.projection\_matrix) + self.mean  
   
 def fit(self, x):  
 self.no\_of\_components = x.shape[1] if self.no\_of\_components is None else self.no\_of\_components  
 self.mean = np.mean(x, axis=0)  
   
 cov\_matrix = np.cov(x - self.mean, rowvar=False)  
   
 self.eigen\_values, self.eigen\_vectors = np.linalg.eig(cov\_matrix)  
 self.eigen\_vectors = self.eigen\_vectors.T  
   
 self.sorted\_components = np.argsort(self.eigen\_values)[::-1]  
   
 self.projection\_matrix = self.eigen\_vectors[self.sorted\_components[:self.no\_of\_components]]self.explained\_variance = self.eigen\_values[self.sorted\_components]  
 self.explained\_variance\_ratio = self.explained\_variance / self.eigen\_values.sum()

Standardization of x:

std = StandardScaler()  
transformed = StandardScaler().fit\_transform(x)

PCA with two components:

pca = convers\_pca(no\_of\_components=2)  
pca.fit(transformed)

Check eigenvectors:

cov\_pca.eigen\_vectors

Check eigenvalues:

cov\_pca.eigen\_values

Check sorted component:

cov\_pca.sorted\_components

Plot PCA with several components = 2:

x\_std = pca.transform(transformed)plt.figure()  
plt.scatter(x\_std[:, 0], x\_std[:, 1], c=y)

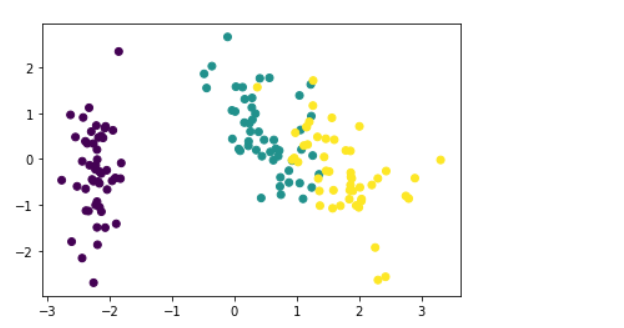


Figure 17: PCA visualization.

**Conclusion**

Massive datasets are increasingly widespread in all sorts of disciplines. To interpret such data sets, we need to decrease dimensionality to preserve the most highly related data.

PCA solves the issue of eigenvectors/values. We apply PCA to remove colinearity during the training phase of linear regression and neural networks. PCA resembles a low-dimensional representation of the observations that explains a good fraction of the variance [[6](http://noiselab.ucsd.edu/ECE285/lecture10.pdf)].

We can use PCA to reduce the number of variables, avoid multicollinearity, or have too many predictors relative to the number of observations. PCA is a linear combination of the *p* features, and taking these linear combinations of the measurements is essential to reduce the number of plots necessary for visual analysis while retaining most of the information present in the data [[7](https://uc-r.github.io/pca)].

Feature reduction is an essential preprocessing step in machine learning. Therefore, PCA is an essential step of preprocessing and very useful for compression and noise removal in the data. It reduces the dimensionality of a dataset by finding a new set of variables smaller than the original set of variables.