

# ML: Principal component analysis

-

[Exercise 8 \(explained variance\)](#)

## ML: Principal component analysis

### Principal component analysis

Principal component analysis is an unsupervised learning method that tries to detect the directions in which the vector formed data varies most. It first finds the direction of highest variance, and then proceeds to discover directions of highest variance that are orthogonal to those direction already found. So, for  $n$  dimensional data, it returns, by default,  $n$  orthogonal directions and the corresponding variances. These directions are called *principal axes*, and if we project a data point to these axes, we get the *principal components* of each axis.

To use another terminology, the set of principal axes forms a base for the vector space where the data points reside, and the principal components are the coordinates of the data points in this new coordinate system. The PCA class in the scikit-learn library has a `transform` method, which transforms data to this new coordinate system.

Let's look at an example where the data is from multi-variate Gaussian distribution.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sklearn.decomposition import PCA
%matplotlib inline
```

First we sample data from this distribution, and then we plot it.

```
rng=np.random.RandomState(0)
X=rng.randn(2,400)
scale=np.array([[1,0], [0,0.4]]) # Standard deviations are 1 and 0.4
rotate=np.array([[1,-1], [1,1]]) / math.sqrt(2)
transform = np.dot(rotate, scale)
X=np.dot(transform, X)
#X=np.dot(scale, X)
#X=np.dot(rotate, X)
X=X.T
plt.axis('equal')
plt.scatter(X[:,0], X[:,1]);
```



Let's first apply the PCA to the data to obtain the principal axes and their variances.

```
from sklearn.decomposition import PCA
def arrow(v1, v2, ax):
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate("", v2, v1, arrowprops=arrowprops)
pca=PCA(2)
pca.fit(X)
print("Principal axes:", pca.components_)
print("Explained variance:", pca.explained_variance_)
print("Mean:", pca.mean_)
```

```
Principal axes: [[-0.73072907 -0.68266758]
 [-0.68266758  0.73072907]]
Explained variance: [ 0.97980663  0.16031015]
Mean: [ 0.01333067 -0.05370929]
```

Then we draw vectors whose directions reflect those of the principal axes, and whose lengths are the corresponding variances. Then we plot the data in this new coordinate system.

```
Z=pca.transform(X)
fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].axis('equal')
axes[0].scatter(X[:,0], X[:,1])
axes[1].axis('equal')
axes[1].set_xlim(-3,3)
axes[1].scatter(Z[:,0], Z[:,1])
for l, v in zip(pca.explained_variance_, pca.components_):
    arrow([0,0], v*1*3, axes[0])
for l, v in zip([1.0,0.16], [np.array([1.0,0.0]),np.array([0.0,1.0])]):
    arrow([0,0], v*1*3, axes[1])
axes[0].set_title("Original")
axes[1].set_title("Transformed");
```



You may have noticed that we gave the PCA constructor a parameter with value 2. This parameter gives the number of principal axes we want. If the parameter value is  $n$ , then the algorithm returns only  $n$  components with the highest variances and drops those components with lower variance. So, this algorithm can be used as a dimensionality reduction technique. The components with low variance are assumed not to contain any important information.

Let's use PCA to project the above data to one dimension.

```
pca=PCA(n_components=1)
pca.fit(X)
Z=pca.transform(X)
print(pca.components_)
plt.axis('equal')
plt.scatter(Z[:,0],np.zeros(400), marker="d", alpha=0.1);
```

```
[[ -0.73072907 -0.68266758]]
```



The dimensionality reduction can be used, for example, to project high-dimensional data in two or three dimensions to allow visualization of data. Additionally, dimensionality reduction can be used as a preprocessing method to obtain only the important features from the data. These important features can then be used, for example, for regression or classification.

## Example of feature extraction

```
from sklearn.datasets import load_diabetes
X, y = load_diabetes(True)
print(X.shape)
print(y.shape)
```

```
(442, 10)
(442,)
```

```
pca=PCA(2)
pca.fit(X)
print(pca.explained_variance_)
Z=pca.transform(X)
plt.axis('equal')
plt.scatter(Z[:,0], Z[:,1]);
```

```
[ 0.0091252 0.00338394]
```



```

rng=np.random.RandomState(0)
X=rng.randn(3,400)
p=rng.rand(10,3) # Random projection into 10d
X=np.dot(p, X)
print(X)

```

```

[[ 0.4891489  0.05331736  0.22206019 ..., -0.38084297 -0.71682684
  0.07960158]
 [ 1.43327136  0.08236987  0.83741626 ..., -0.56888509 -1.0661402
  0.30652977]
 [ 0.39117808 -0.22450206  0.56194345 ..., -0.00355792  0.28652033
  0.00938993]
 ...,
 [-0.35164664 -0.99714767  0.79151041 ..., -1.11453672 -0.63849744
 -0.72193644]
 [ 1.34095136 -0.32676544  1.33260878 ..., -0.2610145 -0.01341865
  0.16976249]
 [ 1.34454622  0.55682337  0.22536013 ..., -0.3262513 -1.25934636
  0.52731505]]

```

```

pca=PCA()
pca.fit(X)
v=pca.explained_variance_
print(v)
plt.plot(np.arange(1,11), np.cumsum(v));

```

```

[ 5.84366626e+01  3.69031722e+01  1.56915171e+01  2.39492996e-29
 6.40732397e-30  1.94630708e-30  1.45347633e-30  4.28030379e-31
 2.60778979e-31  1.54023377e-31]

```



## Exercise 8 (explained variance)

This exercise can give two points at maximum!

Part 1.

Write function `explained_variance` which reads the tab separated file "data.tsv". The data contains 10 features. Then fit PCA to the data. The function should return two lists (or 1D arrays). The first list should contain the variances of all the features. The second list should consist of the explained variances returned by the PCA.

In the main function print these values in the following form:

The variances are: `?.??? ?.??? ...`

The explained variances after PCA are: `?.??? ?.??? ...`

Print the values with three decimal precision and separate the values by a

space.

Part 2.

Plot the cumulative explained variances. The y-axis should be the cumulative sum, and the x-axis the number of terms in the cumulative sum.

## Summary (week 6)

- We got to know another supervised learning method, namely, naive Bayes classification
- We saw examples of naive Bayes classification where either Gaussian or multinomial distribution was used to model the features of samples belonging to a class
- We saw how to use cross validation to assess prediction abilities of a model. This allows us to be sure that the model is not overfitting.
- In the clustering section we saw examples of using k-means, DBSCAN, and hierarchical clustering methods. They have different approaches to clustering, and each have different strengths.
- Clustering is based on the notion of distance between the points in the data.
- Principal component analysis is another example of unsupervised learning
- It can reduce the dimensionality of a data by throwing away those dimensions where the variability is low.

You have reached the end of this section!

Remember to check your points from the ball on the bottom-right corner of the material!

### In this part:

1. ML: Naive Bayes classification

---

2. ML: Clustering

---

3. ML: Principal component analysis

This course is created by the Agile Education Research -research group [of the University of Helsinki](#).

[Credits and about the material.](#)



HELSINGIN YLIOPISTO  
HELSINGFORS UNIVERSITET  
UNIVERSITY OF HELSINKI



MASSIIVISET AVOIMET VERKKOKURSSIT  
MASSIVE OPEN ONLINE COURSES · MOOC.FI