CHAPTER

EIGHT

DIMENSION REDUCTION AND FEATURE EXTRACTION

Introduction

In machine learning and statistics, dimensionality reduction or dimension reduction is the process of reducing the number of features under consideration, and can be divided into feature selection (not addressed here) and feature extraction.

Feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction.

The input matrix $\mathbf{X}$, of dimension $N \times P$, is

$$\begin{bmatrix} x_{11} & \dots & x_{1P} \\ \vdots & \mathbf{X} & \vdots \\ x_{N1} & \dots & x_{NP} \end{bmatrix}$$

where the rows represent the samples and columns represent the variables.

The goal is to learn a transformation that extracts a few relevant features. This is generally done by exploiting the covariance $\mathbf{\Sigma_{XX}}$ between the input features.

Singular value decomposition and matrix factorization

Matrix factorization principles

Decompose the data matrix $\mathbf{X}_{N \times P}$ into a product of a mixing matrix $\mathbf{U}_{N \times K}$ and a dictionary matrix $\mathbf{V}_{P \times K}$.

$$\mathbf{X} = \mathbf{U}\mathbf{V}^T,$$

If we consider only a subset of components $K < rank(\mathbf{X}) < \min(P, N-1)$, $\mathbf{X}$ is approximated by a matrix $\hat{\mathbf{X}}$:

$$\mathbf{X} \approx \hat{\mathbf{X}} = \mathbf{U}\mathbf{V}^T,$$

Each line of $\mathbf{x_i}$ is a linear combination (mixing $\mathbf{u_i}$) of dictionary items $\mathbf{V}$.

N P-dimensional data points lie in a space whose dimension is less than $N - 1$ (2 dots lie on a line, 3 on a plane, etc.).

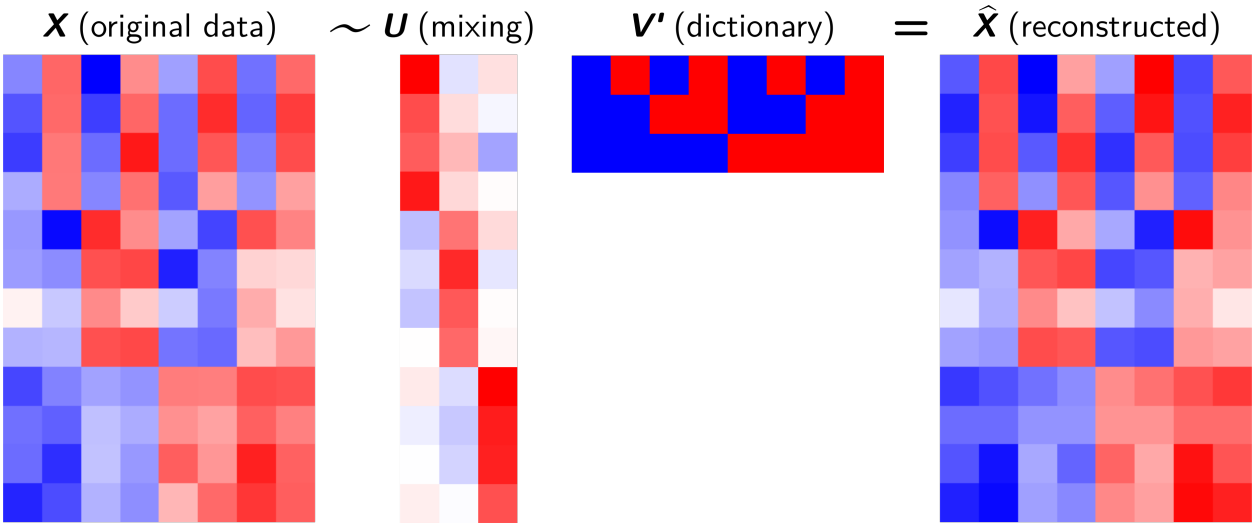


Fig. 8.1: Matrix factorization

Singular value decomposition (SVD) principles

Singular-value decomposition (SVD) factorises the data matrix $\mathbf{X}_{N \times P}$ into a product:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

where

$$\begin{bmatrix} x_{11} & & x_{1P} \\ & \mathbf{X} & \\ x_{N1} & & x_{NP} \end{bmatrix} = \begin{bmatrix} u_{11} & & u_{1K} \\ & \mathbf{U} & \\ u_{N1} & & u_{NK} \end{bmatrix} \begin{bmatrix} d_1 & & 0 \\ & \mathbf{D} & \\ 0 & & d_K \end{bmatrix} \begin{bmatrix} v_{11} & & v_{1P} \\ & \mathbf{V}^T & \\ v_{K1} & & v_{KP} \end{bmatrix}.$$

U: **right-singular**

- $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_K]$ is a $P \times K$ orthogonal matrix.
- It is a **dictionary** of patterns to be combined (according to the mixing coefficients) to reconstruct the original samples.
- $\mathbf{V}$ perfoms the initial **rotations** (**projection**) along the $K = \min(N, P)$ **principal component directions**, also called **loadings**.
- Each $\mathbf{v}_j$ performs the linear combination of the variables that has maximum sample variance, subject to being uncorrelated with the previous $\mathbf{v}_{j-1}$.

D: **singular values**

- $\mathbf{D}$ is a $K \times K$ diagonal matrix made of the singular values of $\mathbf{X}$ with $d_1 \geq d_2 \geq \cdots \geq d_K \geq 0$.
- $\mathbf{D}$ scale the projection along the coordinate axes by $d_1, d_2, \cdots, d_K$.
- Singular values are the square roots of the eigenvalues of $\mathbf{X}^T\mathbf{X}$.

V: **left-singular vectors**

- $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_K]$ is an $N \times K$ orthogonal matrix.
- Each row $\mathbf{v_i}$ provides the **mixing coefficients** of dictionary items to reconstruct the sample $\mathbf{x_i}$

- It may be understood as the coordinates on the new orthogonal basis (obtained after the initial rotation) called **principal components** in the PCA.

SVD for variables transformation

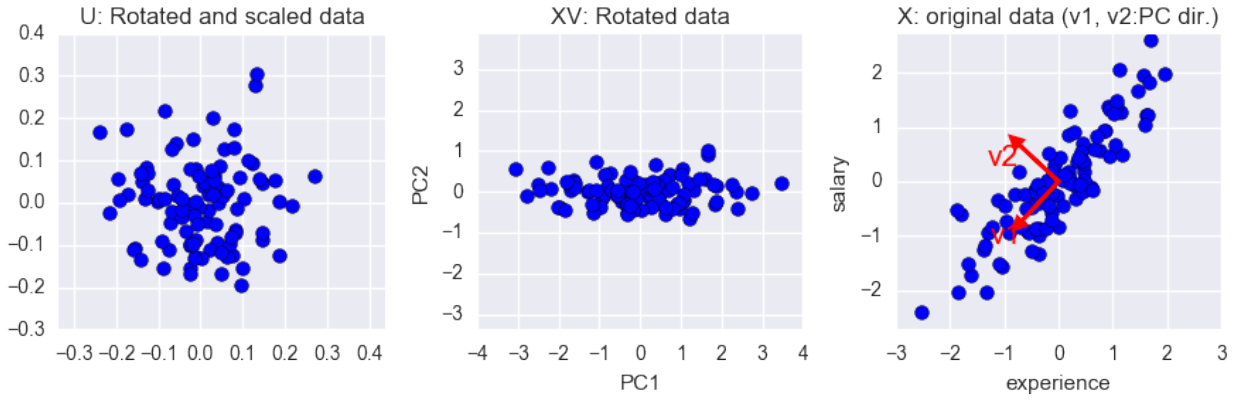
$\mathbf{V}$ transforms correlated variables ($\mathbf{X}$) into a set of uncorrelated ones ($\mathbf{UD}$) that better expose the various relationships among the original data items.

$$\mathbf{X} = \mathbf{UDV}^T, \tag{8.1}$$
$$\mathbf{XV} = \mathbf{UDV}^T\mathbf{V}, \tag{8.2}$$
$$\mathbf{XV} = \mathbf{UDI}, \tag{8.3}$$
$$\mathbf{XV} = \mathbf{UD} \tag{8.4}$$

At the same time, SVD is a method for identifying and ordering the dimensions along which data points exhibit the most variation.

```
import numpy as np
import scipy
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

np.random.seed(42)

# dataset
n_samples = 100
experience = np.random.normal(size=n_samples)
salary = 1500 + experience + np.random.normal(size=n_samples, scale=.5)
X = np.column_stack([experience, salary])

# PCA using SVD
X -= X.mean(axis=0)  # Centering is required
U, s, Vh = scipy.linalg.svd(X, full_matrices=False)
# U : Unitary matrix having left singular vectors as columns.
#     Of shape (n_samples,n_samples) or (n_samples,n_comps), depending on
#     full_matrices.
#
# s : The singular values, sorted in non-increasing order. Of shape (n_comps,),
#     with n_comps = min(n_samples, n_features).
#
# Vh: Unitary matrix having right singular vectors as rows.
#     Of shape (n_features, n_features) or (n_comps, n_features) depending
# on full_matrices.

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.scatter(U[:, 0], U[:, 1], s=50)
plt.axis('equal')
plt.title("U: Rotated and scaled data")

plt.subplot(132)

# Project data
PC = np.dot(X, Vh.T)
```

```
plt.scatter(PC[:, 0], PC[:, 1], s=50)
plt.axis('equal')
plt.title("XV: Rotated data")
plt.xlabel("PC1")
plt.ylabel("PC2")

plt.subplot(133)
plt.scatter(X[:, 0], X[:, 1], s=50)
for i in range(Vh.shape[0]):
    plt.arrow(x=0, y=0, dx=Vh[i, 0], dy=Vh[i, 1], head_width=0.2,
              head_length=0.2, linewidth=2, fc='r', ec='r')
    plt.text(Vh[i, 0], Vh[i, 1],'v%i' % (i+1), color="r", fontsize=15,
             horizontalalignment='right', verticalalignment='top')
plt.axis('equal')
plt.ylim(-4, 4)

plt.title("X: original data (v1, v2:PC dir.)")
plt.xlabel("experience")
plt.ylabel("salary")

plt.tight_layout()
```



Principal components analysis (PCA)

Sources:

- C. M. Bishop *Pattern Recognition and Machine Learning*, Springer, 2006
- Everything you did and didn't know about PCA
- Principal Component Analysis in 3 Simple Steps

Principles

- Principal components analysis is the main method used for linear dimension reduction.
- The idea of principal component analysis is to find the K **principal components directions** (called the **loadings**) $\mathbf{V}_{K \times P}$ that capture the variation in the data as much as possible.

- It converts a set of N P-dimensional observations $\mathbf{N}_{N\times P}$ of possibly correlated variables into a set of N K-dimensional samples $\mathbf{C}_{N\times K}$, where the $K < P$. The new variables are linearly uncorrelated. The columns of $\mathbf{C}_{N\times K}$ are called the **principal components**.
- The dimension reduction is obtained by using only $K < P$ components that exploit correlation (covariance) among the original variables.
- PCA is mathematically defined as an orthogonal linear transformation $\mathbf{V}_{K\times P}$ that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

$$\mathbf{C}_{N\times K} = \mathbf{X}_{N\times P}\mathbf{V}_{P\times K}$$

- PCA can be thought of as fitting a P-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipse is small, then the variance along that axis is also small, and by omitting that axis and its corresponding principal component from our representation of the dataset, we lose only a commensurately small amount of information.
- Finding the K largest axes of the ellipse will permit to project the data onto a space having dimensionality $K < P$ while maximizing the variance of the projected data.

Dataset preprocessing

Centering

Consider a data matrix, $\mathbf{X}$, with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), ie. $\mathbf{X}$ is replaced by $\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T$.

Standardizing

Optionally, standardize the columns, i.e., scale them by their standard-deviation. Without standardization, a variable with a high variance will capture most of the effect of the PCA. The principal direction will be aligned with this variable. Standardization will, however, raise noise variables to the save level as informative variables.

The covariance matrix of centered standardized data is the correlation matrix.

Eigendecomposition of the data covariance matrix

To begin with, consider the projection onto a one-dimensional space ($K = 1$). We can define the direction of this space using a P-dimensional vector $\mathbf{v}$, which for convenience (and without loss of generality) we shall choose to be a unit vector so that $\|\mathbf{v}\|_2 = 1$ (note that we are only interested in the direction defined by $\mathbf{v}$, not in the magnitude of $\mathbf{v}$ itself). PCA consists of two mains steps:

Projection in the directions that capture the greatest variance

Each P-dimensional data point $\mathbf{x}_i$ is then projected onto $\mathbf{v}$, where the coordinate (in the coordinate system of $\mathbf{v}$) is a scalar value, namely $\mathbf{x}_i^T\mathbf{v}$. I.e., we want to find the vector $\mathbf{v}$ that maximizes these coordinates along $\mathbf{v}$, which we will see corresponds to maximizing the variance of the projected data. This is equivalently expressed as

$$\mathbf{v} = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{N}\sum_i \left(\mathbf{x}_i^T\mathbf{v}\right)^2.$$

We can write this in matrix form as

$$\mathbf{v} = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{N}\|\mathbf{X}\mathbf{v}\|^2 = \frac{1}{N}\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} = \mathbf{v}^T\mathbf{S}_{\mathbf{XX}}\mathbf{v},$$

where $\mathbf{S}_{\mathbf{XX}}$ is a biased estiamte of the covariance matrix of the data, i.e.

$$\mathbf{S}_{\mathbf{XX}} = \frac{1}{N}\mathbf{X}^T\mathbf{X}.$$

We now maximize the projected variance $\mathbf{v}^T\mathbf{S}_{\mathbf{XX}}\mathbf{v}$ with respect to $\mathbf{v}$. Clearly, this has to be a constrained maximization to prevent $\|\mathbf{v}_2\| \to \infty$. The appropriate constraint comes from the normalization condition $\|\mathbf{v}\|_2 \equiv \|\mathbf{v}\|_2^2 = \mathbf{v}^T\mathbf{v} = 1$. To enforce this constraint, we introduce a Lagrange multiplier that we shall denote by λ, and then make an unconstrained maximization of

$$\mathbf{v}^T\mathbf{S}_{\mathbf{XX}}\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{v} - 1).$$

By setting the gradient with respect to $\mathbf{v}$ equal to zero, we see that this quantity has a stationary point when

$$\mathbf{S}_{\mathbf{XX}}\mathbf{v} = \lambda\mathbf{v}.$$

We note that $\mathbf{v}$ is an eigenvector of $\mathbf{S}_{\mathbf{XX}}$.

If we left-multiply the above equation by $\mathbf{v}^T$ and make use of $\mathbf{v}^T\mathbf{v} = 1$, we see that the variance is given by

$$\mathbf{v}^T\mathbf{S}_{\mathbf{XX}}\mathbf{v} = \lambda,$$

and so the variance will be at a maximum when $\mathbf{v}$ is equal to the eigenvector corresponding to the largest eigenvalue, λ. This eigenvector is known as the first principal component.

We can define additional principal components in an incremental fashion by choosing each new direction to be that which maximizes the projected variance amongst all possible directions that are orthogonal to those already considered. If we consider the general case of a K-dimensional projection space, the optimal linear projection for which the variance of the projected data is maximized is now defined by the K eigenvectors, $\mathbf{v_1}, \ldots, \mathbf{v_K}$, of the data covariance matrix $\mathbf{S}_{\mathbf{XX}}$ that corresponds to the K largest eigenvalues, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_K$.

Back to SVD

The sample covariance matrix of **centered data** $\mathbf{X}$ is given by

$$\mathbf{S}_{\mathbf{XX}} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X}.$$

We rewrite $\mathbf{X}^T\mathbf{X}$ using the SVD decomposition of $\mathbf{X}$ as

$$\begin{aligned}
\mathbf{X}^T\mathbf{X} &= (\mathbf{U}\mathbf{D}\mathbf{V}^T)^T(\mathbf{U}\mathbf{D}\mathbf{V}^T) \\
&= \mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{U}\mathbf{D}\mathbf{V}^T \\
&= \mathbf{V}\mathbf{D}^2\mathbf{V}^T \\
\mathbf{V}^T\mathbf{X}^T\mathbf{X}\mathbf{V} &= \mathbf{D}^2 \\
\frac{1}{N-1}\mathbf{V}^T\mathbf{X}^T\mathbf{X}\mathbf{V} &= \frac{1}{N-1}\mathbf{D}^2 \\
\mathbf{V}^T\mathbf{S}_{\mathbf{XX}}\mathbf{V} &= \frac{1}{N-1}\mathbf{D}^2
\end{aligned}$$

.

Considering only the k^{th} right-singular vectors $\mathbf{v}_k$ associated to the singular value d_k

$$\mathbf{v_k}^T \mathbf{S_{XX}} \mathbf{v_k} = \frac{1}{N-1} d_k^2,$$

It turns out that if you have done the singular value decomposition then you already have the Eigenvalue decomposition for $\mathbf{X}^T\mathbf{X}$. Where - The eigenvectors of $\mathbf{S_{XX}}$ are equivalent to the right singular vectors, $\mathbf{V}$, of $\mathbf{X}$. - The eigenvalues, λ_k, of $\mathbf{S_{XX}}$, i.e. the variances of the components, are equal to $\frac{1}{N-1}$ times the squared singular values, d_k.

Moreover computing PCA with SVD do not require to form the matrix $\mathbf{X}^T\mathbf{X}$, so computing the SVD is now the standard way to calculate a principal components analysis from a data matrix, unless only a handful of components are required.

PCA outputs

The SVD or the eigendecomposition of the data covariance matrix provides three main quantities:

1. **Principal component directions** or **loadings** are the **eigenvectors** of $\mathbf{X}^T\mathbf{X}$. The $\mathbf{V}_{K\times P}$ or the **right-singular vectors** of an SVD of $\mathbf{X}$ are called principal component directions of $\mathbf{X}$. They are generally computed using the SVD of $\mathbf{X}$.
2. **Principal components** is the $N \times K$ matrix $\mathbf{C}$ which is obtained by projecting $\mathbf{X}$ onto the principal components directions, i.e.

$$\mathbf{C}_{N\times K} = \mathbf{X}_{N\times P}\mathbf{V}_{P\times K}.$$

Since $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ and $\mathbf{V}$ is orthogonal ($\mathbf{V}^T\mathbf{V} = \mathbf{I}$):

$$\mathbf{C}_{N\times K} = \mathbf{U}\mathbf{D}\mathbf{V}^T_{N\times P}\mathbf{V}_{P\times K} \tag{8.5}$$
$$\mathbf{C}_{N\times K} = \mathbf{U}\mathbf{D}^T_{N\times K}\mathbf{I}_{K\times K} \tag{8.6}$$
$$\mathbf{C}_{N\times K} = \mathbf{U}\mathbf{D}^T_{N\times K} \tag{8.7}$$
$$\tag{8.8}$$

Thus $\mathbf{c}_j = \mathbf{X}\mathbf{v}_j = \mathbf{u}_j d_j$, for $j = 1, \ldots K$. Hence $\mathbf{u}_j$ is simply the projection of the row vectors of $\mathbf{X}$, i.e., the input predictor vectors, on the direction $\mathbf{v}_j$, scaled by d_j.

$$\mathbf{c}_1 = \begin{bmatrix} x_{1,1}v_{1,1} + \ldots + x_{1,P}v_{1,P} \\ x_{2,1}v_{1,1} + \ldots + x_{2,P}v_{1,P} \\ \vdots \\ x_{N,1}v_{1,1} + \ldots + x_{N,P}v_{1,P} \end{bmatrix}$$

3. The **variance** of each component is given by the eigen values $\lambda_k, k = 1, \ldots K$. It can be obtained from the singular values:

$$var(\mathbf{c}_k) = \frac{1}{N-1}(\mathbf{X}\mathbf{v}_k)^2 \tag{8.9}$$
$$= \frac{1}{N-1}(\mathbf{u}_k d_k)^2 \tag{8.10}$$
$$= \frac{1}{N-1} d_k^2 \tag{8.11}$$

Determining the number of PCs

We must choose $K^* \in [1, \ldots, K]$, the number of required components. This can be done by calculating the explained variance ratio of the K^* first components and by choosing K^* such that the **cumulative explained variance** ratio is

greater than some given threshold (e.g., $\approx 90\%$). This is expressed as

$$\text{cumulative explained variance}(\mathbf{c}_k) = \frac{\sum_j^{K^*} var(\mathbf{c}_k)}{\sum_j^{K} var(\mathbf{c}_k)}.$$

Interpretation and visualization

PCs

Plot the samples projeted on first the principal components as e.g. PC1 against PC2.

PC directions

Exploring the loadings associated with a component provides the contribution of each original variable in the component.

Remark: The loadings (PC directions) are the coefficients of multiple regression of PC on original variables:

$$\mathbf{c} = \mathbf{X}\mathbf{v} \tag{8.12}$$

$$\mathbf{X}^T\mathbf{c} = \mathbf{X}^T\mathbf{X}\mathbf{v} \tag{8.13}$$

$$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{c} = \mathbf{v} \tag{8.14}$$

Another way to evaluate the contribution of the original variables in each PC can be obtained by computing the correlation between the PCs and the original variables, i.e. columns of $\mathbf{X}$, denoted $\mathbf{x}_j$, for $j = 1, \ldots, P$. For the k^{th} PC, compute and plot the correlations with all original variables

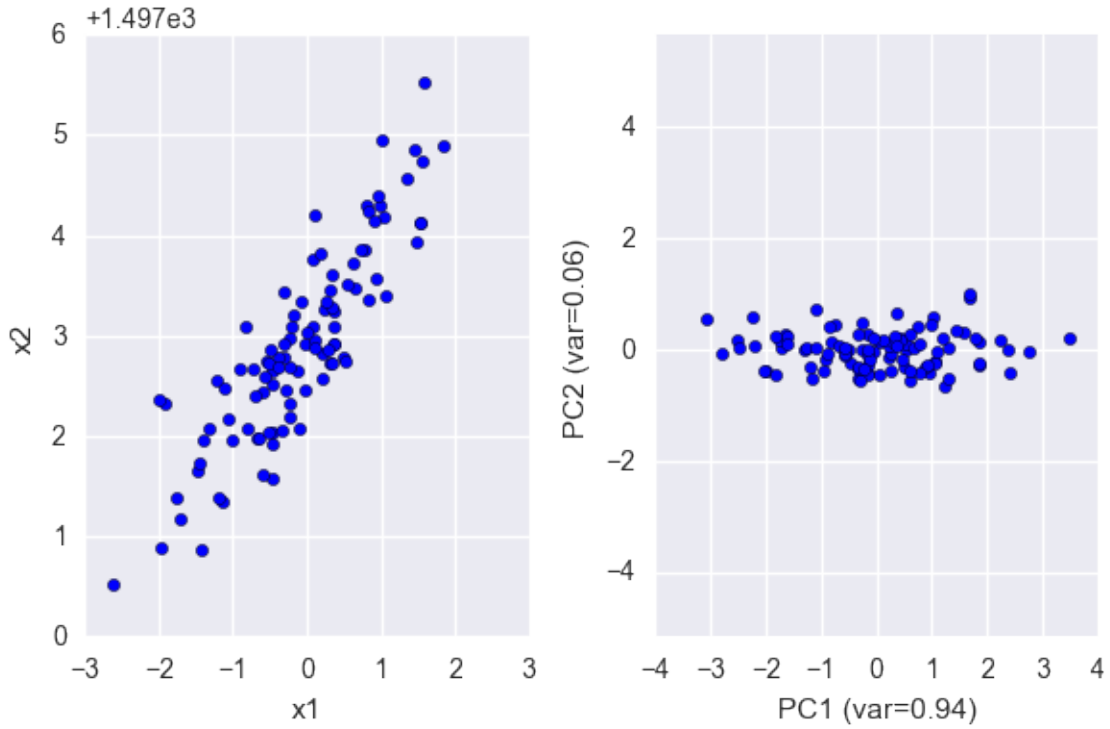
$$cor(\mathbf{c}_k, \mathbf{x}_j), j = 1 \ldots K, j = 1 \ldots K.$$

These quantities are sometimes called the *correlation loadings*.

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

np.random.seed(42)

# dataset
n_samples = 100
experience = np.random.normal(size=n_samples)
salary = 1500 + experience + np.random.normal(size=n_samples, scale=.5)
X = np.column_stack([experience, salary])

# PCA with scikit-learn
pca = PCA(n_components=2)
pca.fit(X)
print(pca.explained_variance_ratio_)

PC = pca.transform(X)

plt.subplot(121)
plt.scatter(X[:, 0], X[:, 1])
plt.xlabel("x1"); plt.ylabel("x2")

plt.subplot(122)
plt.scatter(PC[:, 0], PC[:, 1])
plt.xlabel("PC1 (var=%.2f)" % pca.explained_variance_ratio_[0])
plt.ylabel("PC2 (var=%.2f)" % pca.explained_variance_ratio_[1])
```

```
plt.axis('equal')
plt.tight_layout()
```

```
[ 0.93646607  0.06353393]
```



Multi-dimensional Scaling (MDS)

Resources:

- http://www.stat.pitt.edu/sungkyu/course/2221Fall13/lec8_mds_combined.pdf
- https://en.wikipedia.org/wiki/Multidimensional_scaling
- Hastie, Tibshirani and Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York: Springer, Second Edition.

The purpose of MDS is to find a low-dimensional projection of the data in which the pairwise distances between data points is preserved, as closely as possible (in a least-squares sense).

- Let $\mathbf{D}$ be the $(N \times N)$ pairwise distance matrix where d_{ij} is *a distance* between points i and j.
- The MDS concept can be extended to a wide variety of data types specified in terms of a similarity matrix.

Given the dissimilarity (distance) matrix $\mathbf{D}_{N \times N} = [d_{ij}]$, MDS attempts to find K-dimensional projections of the N points $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^K$, concatenated in an $\mathbf{X}_{N \times K}$ matrix, so that $d_{ij} \approx \|\mathbf{x}_i - \mathbf{x}_j\|$ are as close as possible. This can be obtained by the minimization of a loss function called the **stress function**

$$\text{stress}(\mathbf{X}) = \sum_{i \neq j} \left(d_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|\right)^2 .$$

This loss function is known as *least-squares* or *Kruskal-Shepard* scaling.

A modification of *least-squares* scaling is the *Sammon mapping*

$$\text{stress}_{\text{Sammon}}(\mathbf{X}) = \sum_{i \neq j} \frac{(d_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)^2}{d_{ij}}.$$

The Sammon mapping performs better at preserving small distances compared to the *least-squares* scaling.

Classical multidimensional scaling

Also known as *principal coordinates analysis*, PCoA.

- The distance matrix, $\mathbf{D}$, is transformed to a *similarity matrix*, $\mathbf{B}$, often using centered inner products.
- The loss function becomes

$$\text{stress}_{\text{classical}}(\mathbf{X}) = \sum_{i \neq j} \left(b_{ij} - \langle \mathbf{x}_i, \mathbf{x}_j \rangle\right)^2.$$

- The stress function in classical MDS is sometimes called *strain*.
- The solution for the classical MDS problems can be found from the eigenvectors of the similarity matrix.
- If the distances in $\mathbf{D}$ are Euclidean and double centered inner products are used, the results are equivalent to PCA.

Example

The `eurodist` datset provides the road distances (in kilometers) between 21 cities in Europe. Given this matrix of pairwise (non-Euclidean) distances $\mathbf{D} = [d_{ij}]$, MDS can be used to recover the coordinates of the cities in *some* Euclidean referential whose orientation is arbitrary.

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

# Pairwise distance between European cities
try:
    url = '../data/eurodist.csv'
    df = pd.read_csv(url)
except:
    url = 'https://raw.github.com/neurospin/pystatsml/master/data/eurodist.csv'
    df = pd.read_csv(url)

print(df.ix[:5, :5])

city = df["city"]
D = np.array(df.ix[:, 1:])  # Distance matrix

# Arbitrary choice of K=2 components
from sklearn.manifold import MDS
mds = MDS(dissimilarity='precomputed', n_components=2, random_state=40, max_
↪iter=3000, eps=1e-9)
X = mds.fit_transform(D)
```

```
        city  Athens  Barcelona  Brussels  Calais
0     Athens       0       3313      2963    3175
1  Barcelona    3313          0      1318    1326
```

```
2   Brussels     2963        1318         0      204
3     Calais     3175        1326       204        0
4  Cherbourg     3339        1294       583      460
5    Cologne     2762        1498       206      409
```

Recover coordinates of the cities in Euclidean referential whose orientation is arbitrary:

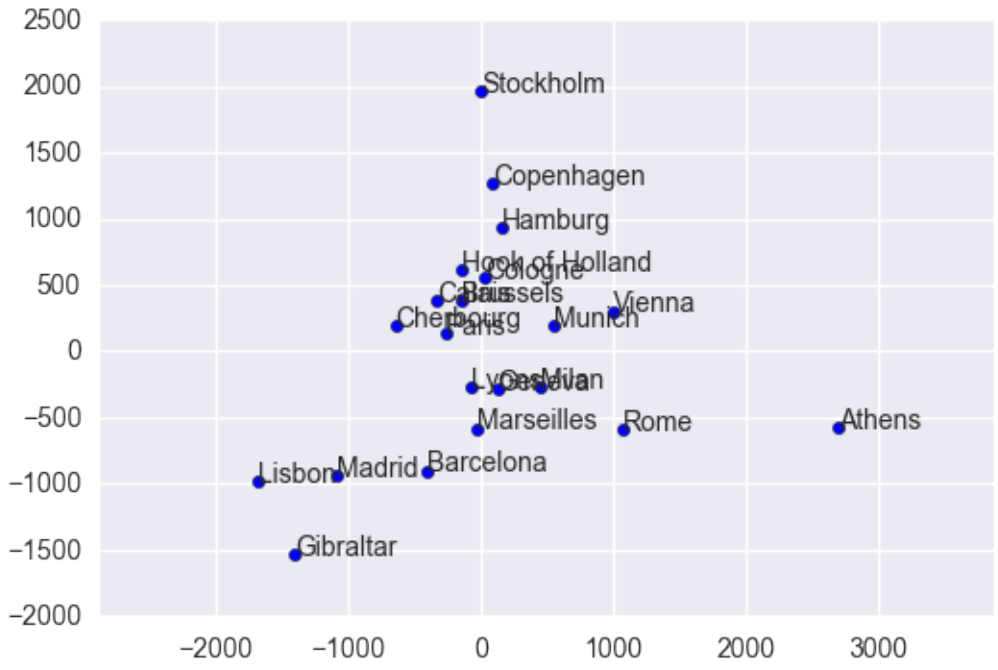
```
from sklearn import metrics
Deuclidean = metrics.pairwise.pairwise_distances(X, metric='euclidean')
print(np.round(Deuclidean[:5, :5]))
```

```
[[    0.  3116.  2994.  3181.  3428.]
 [ 3116.     0.  1317.  1289.  1128.]
 [ 2994.  1317.     0.   198.   538.]
 [ 3181.  1289.   198.     0.   358.]
 [ 3428.  1128.   538.   358.     0.]]
```

Plot the results:

```
# Plot: apply some rotation and flip
theta = 80 * np.pi / 180.
rot = np.array([[np.cos(theta), -np.sin(theta)],
                [np.sin(theta),  np.cos(theta)]])
Xr = np.dot(X, rot)
# flip x
Xr[:, 0] *= -1
plt.scatter(Xr[:, 0], Xr[:, 1])

for i in range(len(city)):
    plt.text(Xr[i, 0], Xr[i, 1], city[i])
plt.axis('equal')
```

```
(-2000.0, 3000.0, -2000.0, 2500.0)
```



Determining the number of components

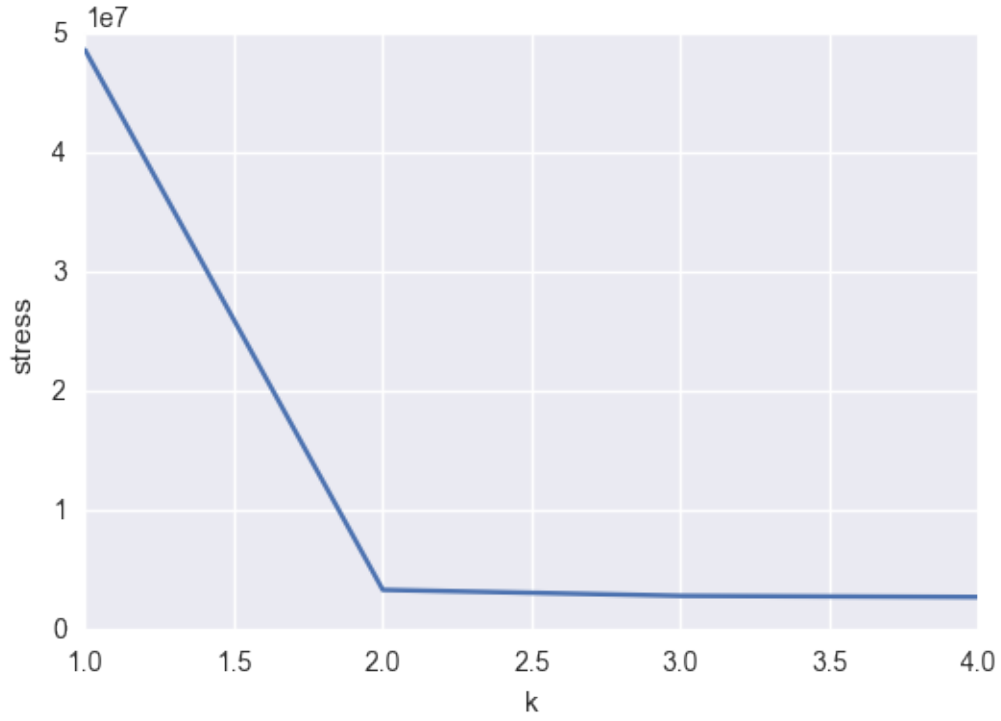
We must choose $K^* \in \{1, \ldots, K\}$ the number of required components. Plotting the values of the stress function, obtained using $k \leq N - 1$ components. In general, start with $1, \ldots K \leq 4$. Choose K^* where you can clearly distinguish an *elbow* in the stress curve.

Thus, in the plot below, we choose to retain information accounted for by the first *two* components, since this is where the *elbow* is in the stress curve.

```
k_range = range(1, min(5, D.shape[0]-1))
stress = [MDS(dissimilarity='precomputed', n_components=k,
           random_state=42, max_iter=300, eps=1e-9).fit(D).stress_ for k in k_range]

print(stress)
plt.plot(k_range, stress)
plt.xlabel("k")
plt.ylabel("stress")
```

```
[48644495.285714284, 3356497.3657523869, 2858455.4958879612, 2756310.6376280105]
```

```
<matplotlib.text.Text at 0x7fefc49a2518>
```



Nonlinear dimensionality reduction

Sources:

- Scikit-learn documentation
- Wikipedia

Nonlinear dimensionality reduction or **manifold learning** cover unsupervised methods that attempt to identify low-dimensional manifolds within the original P-dimensional space that represent high data density. Then those methods provide a mapping from the high-dimensional space to the low-dimensional embedding.

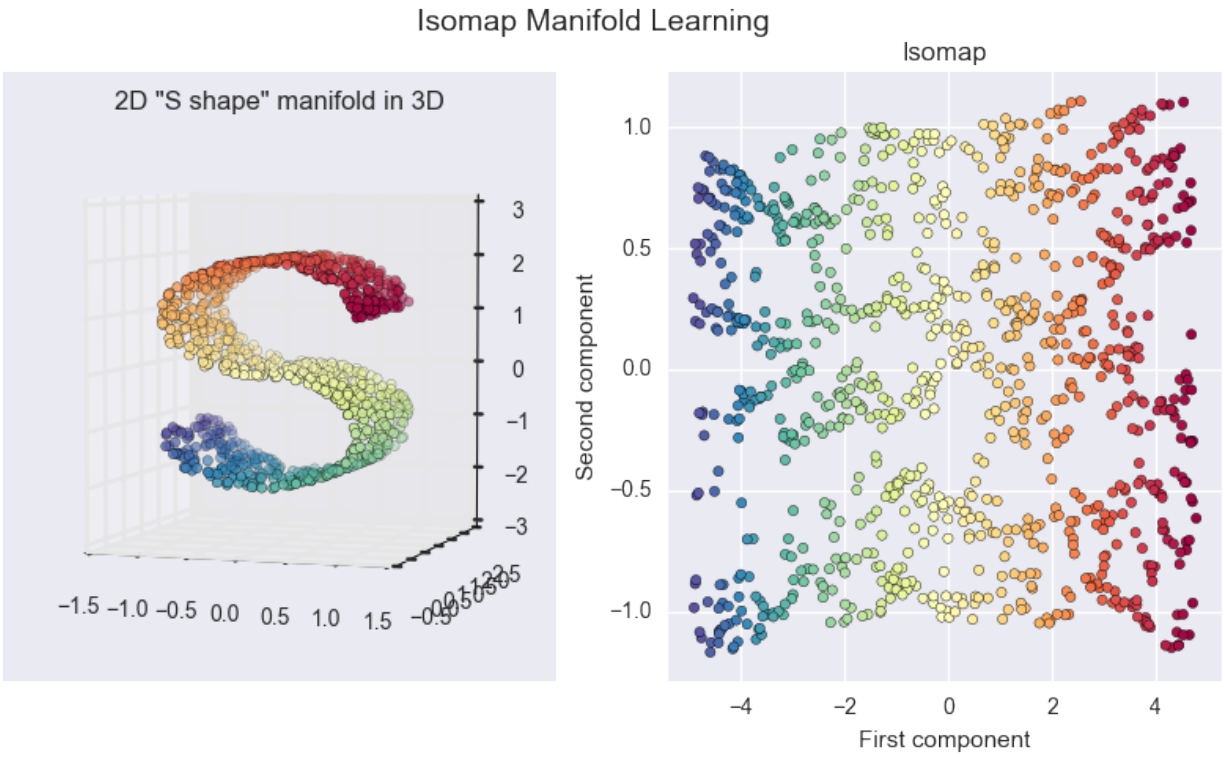
Isomap

Isomap is a nonlinear dimensionality reduction method that combines a procedure to compute the distance matrix with MDS. The distances calculation is based on geodesic distances evaluated on neighborhood graph:

1. Determine the neighbors of each point. All points in some fixed radius or K nearest neighbors.
2. Construct a neighborhood graph. Each point is connected to other if it is a K nearest neighbor. Edge length equal to Euclidean distance.
3. Compute shortest path between pairwise of points d_{ij} to build the distance matrix $\mathbf{D}$.
4. Apply MDS on $\mathbf{D}$.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import manifold, datasets

X, color = datasets.samples_generator.make_s_curve(1000, random_state=42)

fig = plt.figure(figsize=(10, 5))
plt.suptitle("Isomap Manifold Learning", fontsize=14)

ax = fig.add_subplot(121, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax.view_init(4, -72)
plt.title('2D "S shape" manifold in 3D')

Y = manifold.Isomap(n_neighbors=10, n_components=2).fit_transform(X)
ax = fig.add_subplot(122)
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("Isomap")
plt.xlabel("First component")
plt.ylabel("Second component")
plt.axis('tight')
```

```
(-5.4131242078919239,
 5.2729984345096854,
 -1.2877687637642998,
 1.2316524684384262)
```



Exercises

PCA

Write a basic PCA class

Write a class `BasicPCA` with two methods:

- `fit(X)` that estimates the data mean, principal components directions $\mathbf{V}$ and the explained variance of each component.
- `transform(X)` that projects the data onto the principal components.

Check that your `BasicPCA` gave similar results, compared to the results from `sklearn`.

Apply your Basic PCA on the `iris` dataset

The data set is available at: https://raw.github.com/neurospin/pystatsml/master/data/iris.csv

- Describe the data set. Should the dataset been standardized?
- Describe the structure of correlations among variables.
- Compute a PCA with the maximum number of components.
- Compute the cumulative explained variance ratio. Determine the number of components K by your computed values.
- Print the K principal components directions and correlations of the K principal components with the original variables. Interpret the contribution of the original variables into the PC.

- Plot the samples projected into the K first PCs.
- Color samples by their species.

MDS

Apply MDS from `sklearn` on the `iris` dataset available at:

https://raw.github.com/neurospin/pystatsml/master/data/iris.csv

- Center and scale the dataset.
- Compute Euclidean pairwise distances matrix.
- Select the number of components.
- Show that classical MDS on Euclidean pairwise distances matrix is equivalent to PCA.