Fundaments of Machine learning for and with engineering applications

Enrico Riccardi[1]

Department of Energy Resources, University of Stavanger (UiS).[1]

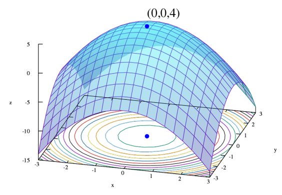
Feb 3, 2025

Optimization

Optimization is a field on its own.

Definition

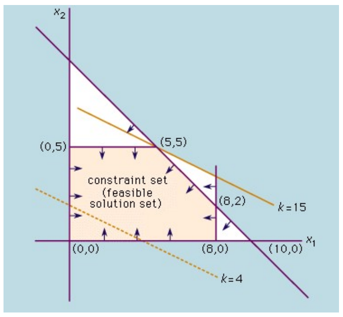
- Finding the "optimus" (latin), the best.
- Collection of mathematical principles and methods used for solving quantitative problems.



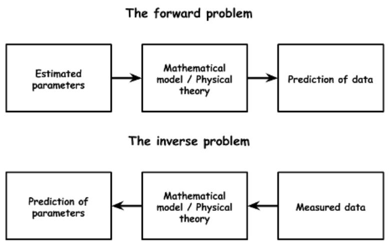
Constrains

Not only we want to find the best solution, we also need to respect the constraints.

Finding the minimum and maxima of functions subjected to constraints.



Forward and backward methods



Requirements:

1. Existence
2. Uniqueness
3. Stability
4. Efficiency

Gradient descend

Let's reconsider a linear model:

$$\hat{y}_i = \sum_{j=0}^{m} x_{ij} b_j$$

Let's get back the residuals

$$R = e^T e$$

What we do is to calculate the derivative of the residuals in respect to the coefficients b_j.

and the ideal solution is when

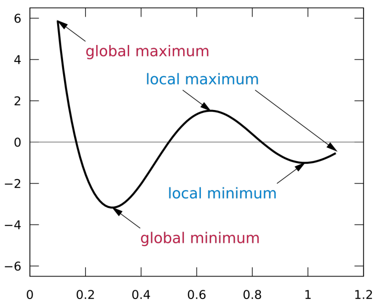
$$\frac{\partial R}{\partial b_j} = 0 \quad \forall j \in [0, n]$$

Gradient descend

ok, but what if we do not know

$$\frac{\partial R}{\partial b_j} = 0$$

We hopefully be able to numerically calculate $\frac{\partial R}{\partial b_j}$

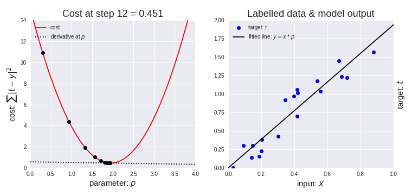


Learning rate

The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

This is not that easy... Learning rate can be:

- Constant
- Variable (a given function)
- Block-set up (sparse data)
- Adaptive (as a function of the loss function)



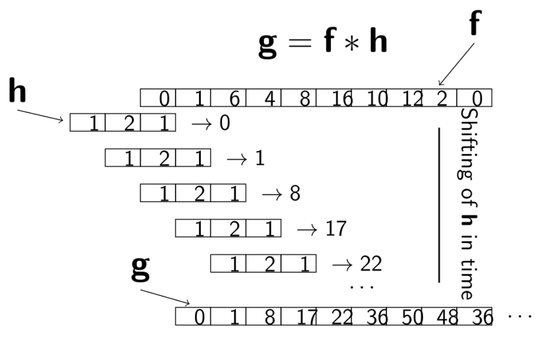
Filtering

Convolution

Convolution involves a window function that changes another function by *sliding* over it and performing local multiplications and additions. Depending on the shape of the convolution function we can perform

- Smoothings
- Deformations
- Differentiations

Convolution



Convolution

In general we can write the convolution between a function f and a convolving (deforming) function h as:

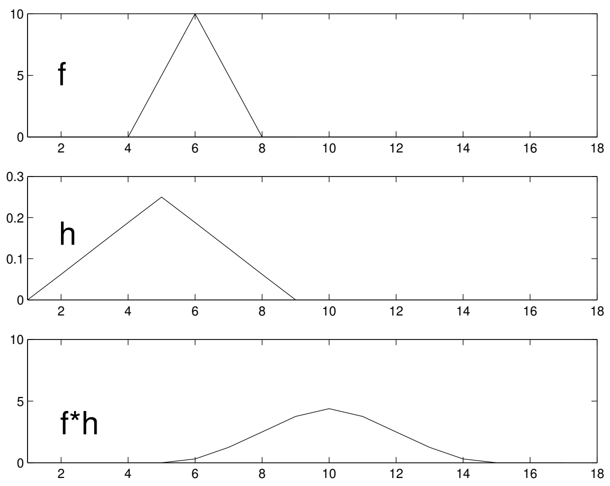
$$g(t) = \sum_{m=-\infty}^{\infty} f(m)h(m-t)$$

Often we use a more compact notation:

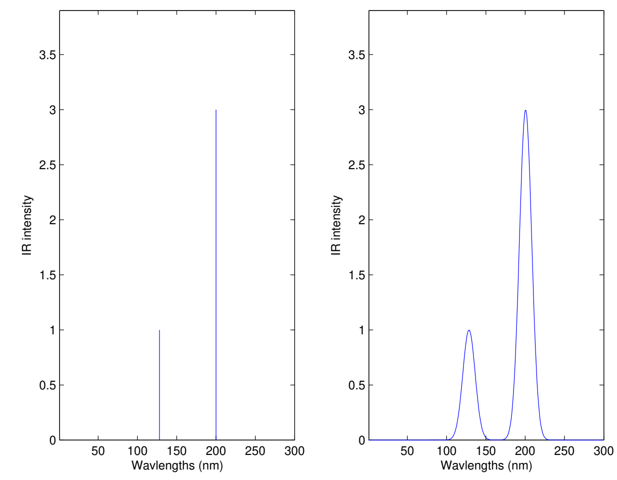
$$g(t) = f(t) * h(t) = h(t) * f(t)$$

where $*$ is the convolution operator

Convolution function



Peak broadening



Convolution operator properties

The convolution operator follows the distributive rule:

$$f_1(t) * [f_2(t) + f_3(t)] = f_1(t) * f_2(t) + f_1(t) * f_3(t)$$

It also follows the associative rule regarding order:

$$f_1(t) * [f_2(t) * f_3(t)] = [f_1(t) * f_2(t)] * f_3(t)$$

Convolution operator properties

Repeated convolutions

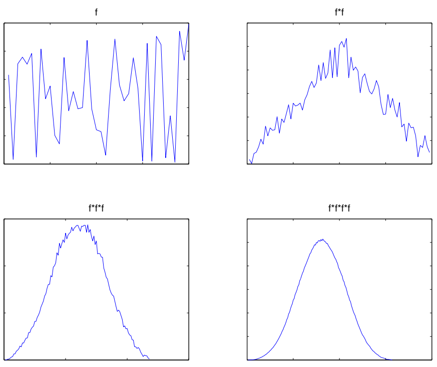
Take any function $g(t)$ and convolve it with any function $f(t)$ multiple times:

$$\begin{aligned} a_1(t) &= g(t) * f(t) \\ a_2(t) &= g(t) * a_1(t) \\ a_3(t) &= g(t) * a_2(t) \\ &\vdots \\ a_n(t) &\to \text{gaussian}(t) \end{aligned}$$

i.e. the result with always converge to a Gaussian function

Convolution properties

Any signal convolved with itself repeated many times will converge to a Gaussian function:



Mean Smooth operator

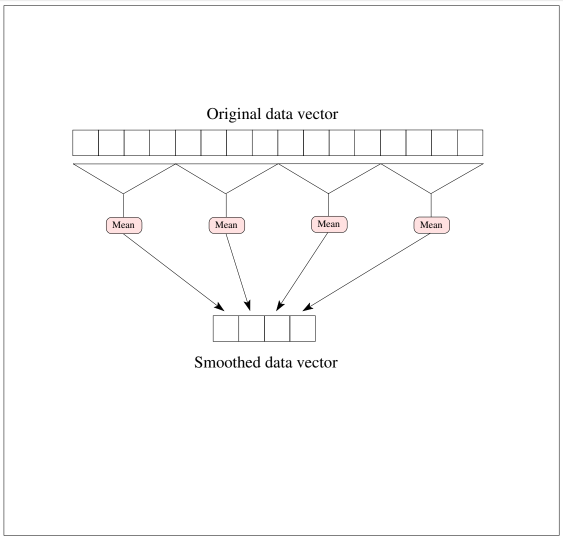
This is a very simple method which works as follows:

- Assume data vector x which contains n data points
- Start with the k first points, i.e. $[x_1, x_2, \cdots, x_k]$ and compute mean u_1 of these k points
- Take the next k points, $[x_{k+1}, x_{k+2}, \cdots, x_{2k}]$ and compute the mean u_2 of these k points
- Continue with this process until the data vector x is exhausted of points

There are two effects of this preprocessing:

- The new data vector u is of length approximately $1/k$'th of compared to the original
- Each element u_j has less noise due the cancelling effects of computing the mean

Mean Smooth operator



Running Average Smooth operator

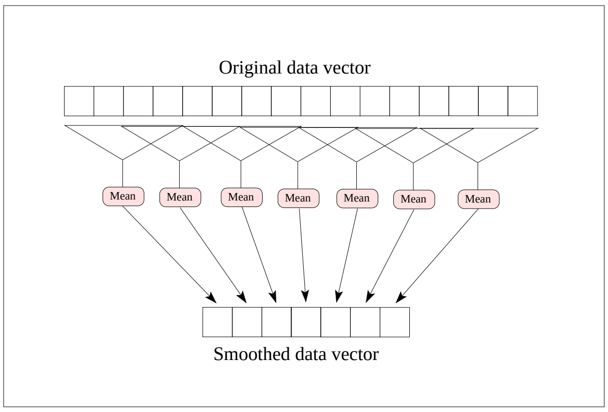
Let f be the original data profile and g the smooth version of this profile. Then we have:

$$g(i) = \sum_{j=-m}^{m} \frac{f(i+j)}{2m+1}$$

where m is the number of points in the window

Running Average Smooth operator

This is similar to the mean smoother but moves in shorter steps than the whole window length



Convolution or Moving average?

If we have a window with 3 points ($m = 1$) and we want to calculate the new value of point no. 5 in the original profile:

$$g(5) = [0 \cdot f(3) + 1 \cdot f(4) + 1 \cdot f(5) + 1 \cdot f(6) + 0 \cdot f(7)] \cdot \frac{1}{3}$$

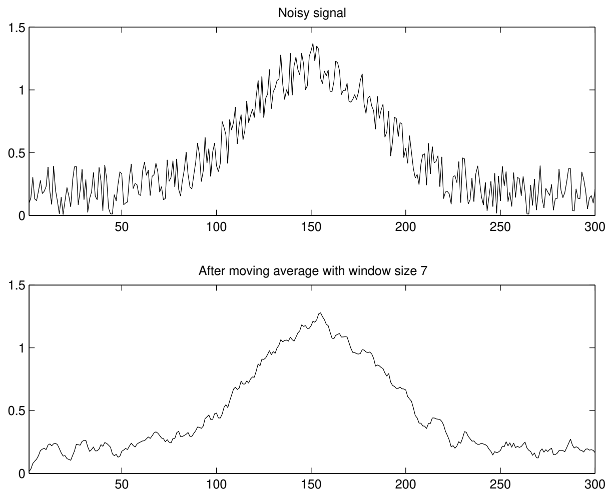
$$g(5) = \frac{1}{3}[0 \;\; 1 \;\; 1 \;\; 1 \;\; 0] \cdot [f(3) \;\; f(4) \;\; f(5) \;\; f(6) \;\; f(7)]^T$$

Convolution or Moving average?

A moving average IS the convolution between the vector **f** and a vector of ones (times a constant), i.e. :

$$\text{moving average} = \mathbf{f} * \mathbf{h} = \mathbf{f} * \frac{1}{n}[1 \;\; 1 \cdots 1 \;\; 1]$$

Moving average



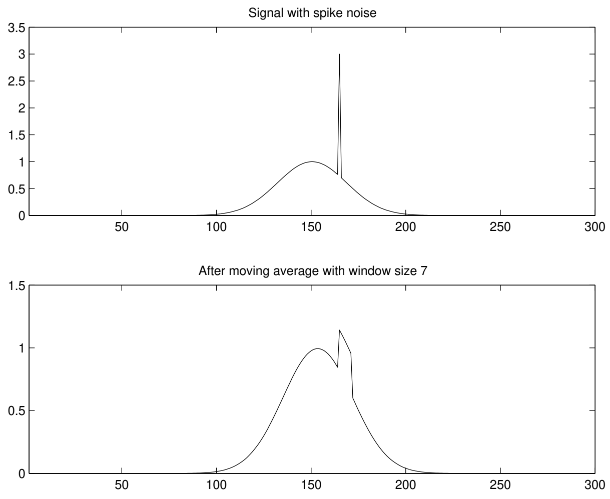
Moving average problems

- Broadening of peaks
- *Spike-noise* affects the smoothed profile

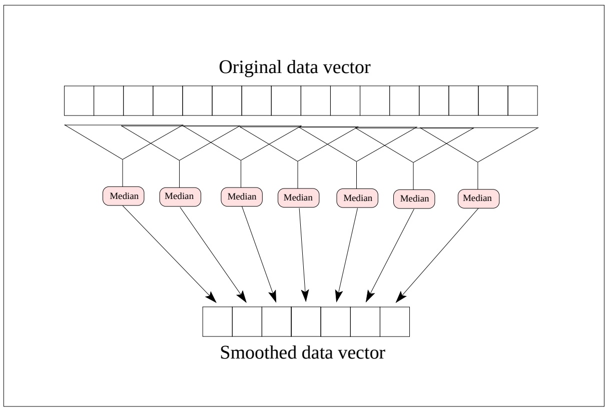
A better alternative:

Use the median instead of the mean, then we don't have the problems with *spike-noise*. However, this filter is not linear

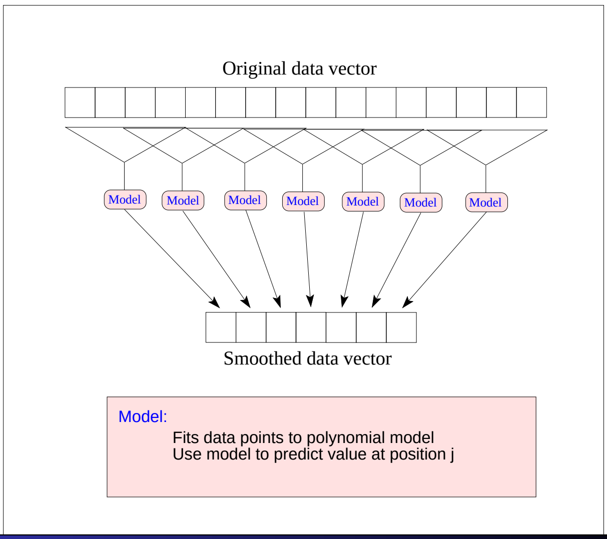
Running average example



Running mediam

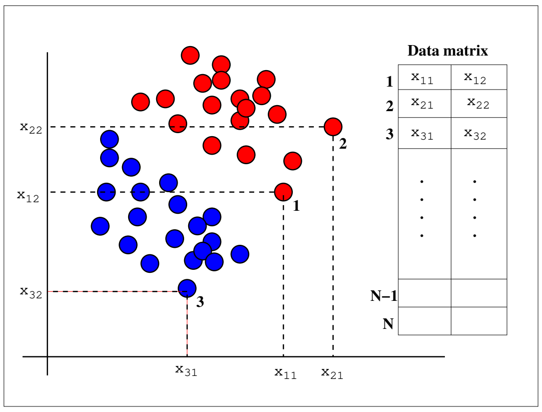


Polynomial smoothing



PCA

Central in any data analysis is the use of a *data matrix*



PCA base

Question:
How can we understand the information contained in such a data matrix?

- The geometry of the "object cloud" in N dimensions is used to understand relations

Geometrical insights
Plotting provides **geometrical insights** and observation of **hidden data structures** and **patterns**

Correlated variables

Problem
How can we use plotting if we have more than 3 variables?

Solution
1. Seek for **correlated variables** in the data matrix
2. Seek for latent variable
3. Give up (use AI)

Correlated variables

- Correlated variables contain approximately the same information.
- Several correlated variables suggests:
  - the same **phenomenon** is manifested in different way
  - an **underlying phenomenon** more fundamental exists

Let's assume the latter:

Linear combinations
Assuming the latent variables to be a **linear combinations** of the original variables, i.e.:

$$LV = a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

Latent variables

A new coordinate system

1. Latent variables are based on creating a new coordinate system based on linear combination of the original variables
2. Objects are projected from a higher dimensional

data space onto this new (lower dimensional) coordinate system

1. The new coordinate system improves interpretation and prediction.

Principal component analysis (PCA) can automatically create useful latent variables

PCA

Originally invented in 1901 by Karl Pearson and re-invented several times. The PCA method is also referred to as:

1. Singular value decomposition (numerical analysis)
2. Karhunen-Loeve expansion (electric engineering)
3. Eigenvector analysis (physical sciences)
4. Hotelling transform (image analysis/statistics)
5. Correspondence analysis (double scaled version of PCA)

Karl Pearson

PCA

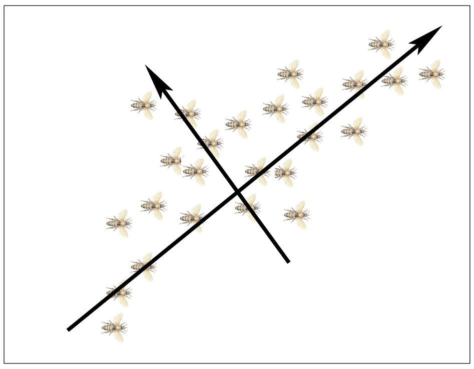
Goals of PCA:

1. Simplification.
2. Data reduction and data compression
3. Modeling
4. Outlier detection
5. Variable selection
6. Classification
7. Prediction
8. ... world peace ...

PCA

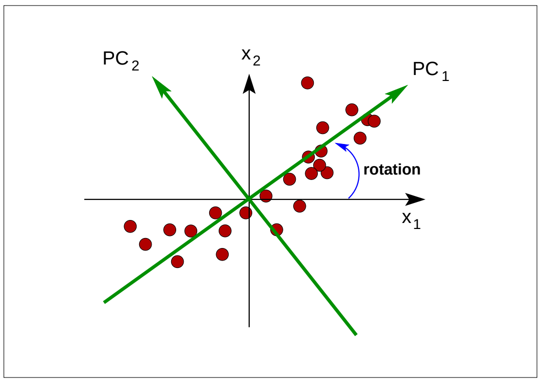
Rotation of the coordinate system

In PCA the original coordinate system is rotated such that the new latent variable axes point in the direction of max variance



PCA

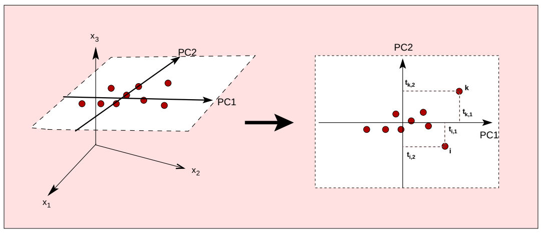
Rotation of the coordinate system



PCA

Scores are new coordinates

Scores are the coordinates of objects in the **new** coordinate system



PCA loading

Loading and direction

The loadings are the weights needed to define the **direction** of the latent variable axis in the original space

The loading weights p_j are the coefficients in the linear combination of the original variables:

$$t_i = p_1x_1 + p_2x_2 + \cdots + p_Mx_M$$

Model description

The PCA model

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}$$

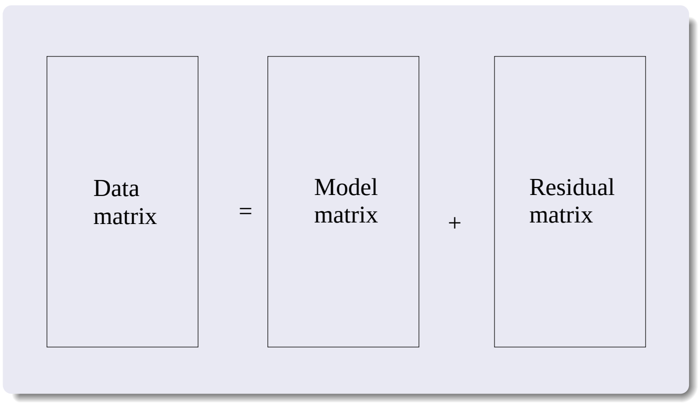
where

- $\mathbf{X}$ is the data matrix
- $\mathbf{T}$ is the scores matrix
- $\mathbf{P}$ is the loadings matrix
- $\mathbf{E}$ is the residual matrix

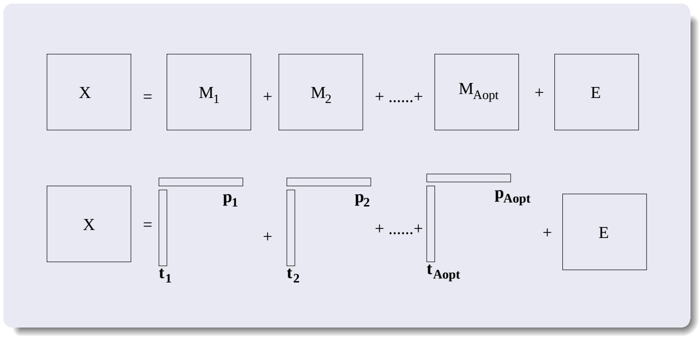
PCA is an example of a **bilinear model** where a matrix $\mathbf{Z}$ is written as a product of two others:

$\mathbf{Z} = \mathbf{A}\mathbf{B}$

Data = Model + Noise



PCA Model



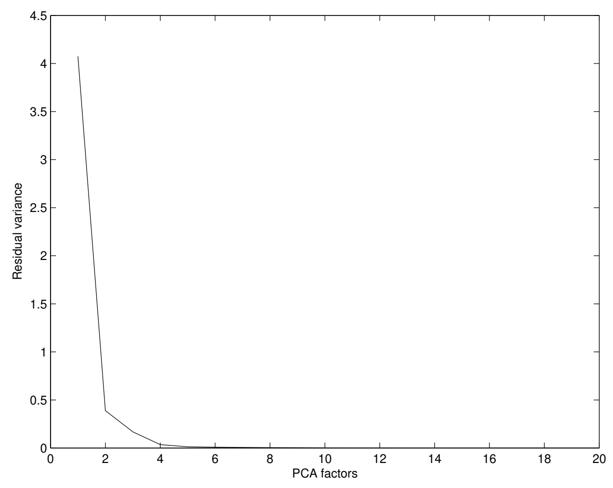
PCA sorting

The PC's are sorted according to variance

1. The new latent variables are **sorted with respect to how much variance they explain**. This means the first component explains the most, followed by no.2 etc.
2. Only the $A < A_{max}$ components are actually used
3. The remaining last K components are related to noise (or are zero).

The contribution by each PC can be seen from the **residual variance plot**

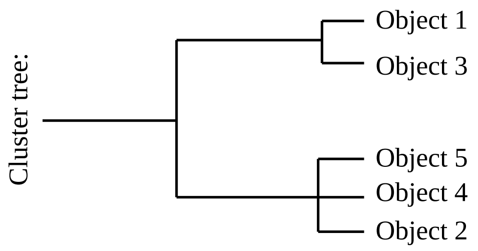
Residual variance plot



Agglomerative algorithms

Agglomerative cluster analysis "clumps" objects together according to a definition of similarity or dissimilarity. The objects are merged progressively into larger clusters until only one cluster remains which consists of all the objects in the data set

This can be summarised in a hierarchical cluster tree.



Clumping objects

One of the simpliest iterative approaches for unsupervised clustering is:

n_clusters = n_datapoints

1. WHILE no. clusters > 1
2. Find smallest distance between clusters A and B
3. Merge clusters A and B
4. Define a new cluster (AB)
5. Distance matrix between all clusters
6. ENDWHILE

Cluster distances

Cluster distances

Single linkage

$$d_{AB} = \min(f_{i_A j_B}), \forall i \in A, j \in B$$

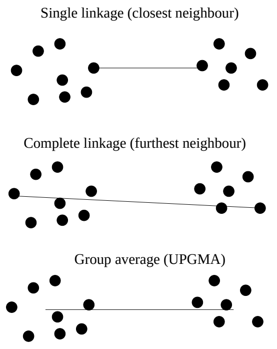
Complete linkage

$$d_{AB} = \max(f_{i_A j_B}), \forall i \in A, j \in B$$

Group average (UPGMA)

$$d_{AB} = \frac{1}{nm} \sum_{i \in A} \sum_{j \in B} f_{i_A j_B},$$

$$\forall i \in A, j \in B$$


Single linkage (closest neighbour)
Complete linkage (furthest neighbour)
Group average (UPGMA)

Ward's method

Assume we have partitioned a set of objects into K clusters. For each cluster we can compute the total within-cluster error sum of squares:

$$E_m = \sum_{i=1}^{n_m} \sum_{j=1}^{M} \left[x_{ij}^{(m)} - \bar{x}_j^{(m)}\right]^2$$

where n_m is the number of objects in cluster m. $\bar{x}_j^{(m)}$ is the j'th variable for the mean vector of cluster m. M is the total number of variables.

Ward Equation

Now we sum all these E_m's for every cluster:

$$E_{tot}^{(0)} = \sum_{m=1}^{K} E_m$$

Assume we merge two clusters A and B so we are left with $K - 1$ clusters. For the new clusters we compute $E_{tot}^{(1)}$ for the $K - 1$ clusters.

The Ward criterion

The Ward criterion for merging A and B is that we want the number:

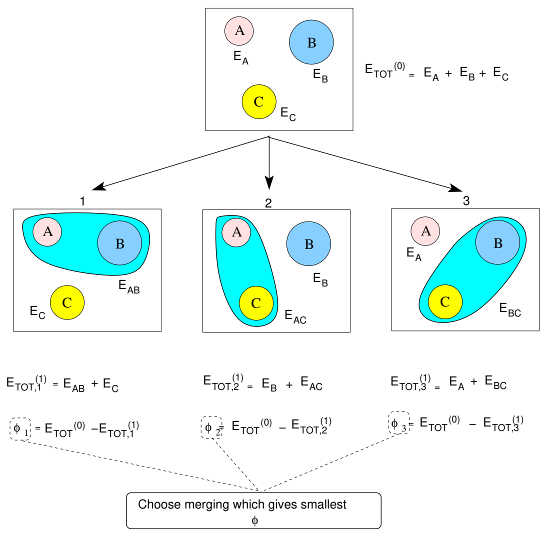
$$\phi = E_{tot}^{(1)} - E_{tot}^{(0)}$$

to be as small as possible. So in order to find this number we need to check ϕ for every possible merging of two clusters. In general there are

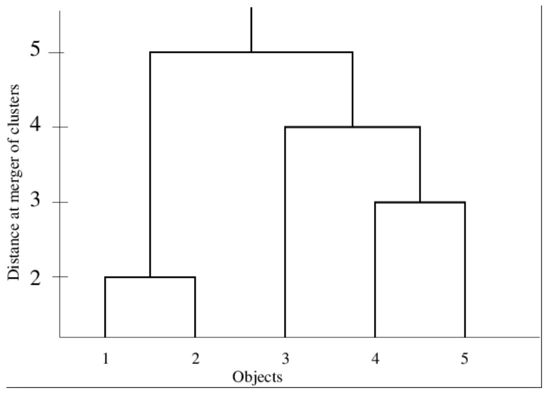
$$\frac{K(K-1)}{2}$$

number of such possible pairs of clusters. The pair A and B which gave the smallest number is chosen to be merged.

Example Ward's method


A
B
C
E_A
E_B
E_C
$E_{TOT}^{(0)} = E_A + E_B + E_C$
1
2
3
E_{AB}
E_{AC}
E_{BC}
$E_{TOT,1}^{(1)} = E_{AB} + E_C$
$E_{TOT,2}^{(1)} = E_B + E_{AC}$
$E_{TOT,3}^{(1)} = E_A + E_{BC}$
$\phi_1 = E_{TOT}^{(0)} - E_{TOT,1}^{(1)}$
$\phi_2 = E_{TOT}^{(0)} - E_{TOT,2}^{(1)}$
$\phi_3 = E_{TOT}^{(0)} - E_{TOT,3}^{(1)}$
Choose merging which gives smallest ϕ

Example dendrogram



k-means cluster analysis

With the raise of Machine Learning, one of the most popular approach is k-means.

The algorithm consists of:

1. Select the number of clusters $K \leq K_{max}$ to look for
2. Start by creating K random cluster centres $\mathbf{m}_k$
3. For each object $\mathbf{x}_j$ assign it to the cluster center it is nearest to
4. Re-compute center points $\mathbf{m}_k$ for the new clusters and re-iterate towards convergence

This procedure minimises the within-cluster variance

Optimal no of clusters

In k-means cluster analysis we assume a **true** number of clusters.

To estimate the optimal no. of clusters K^* from data we may do as follows:

1. Compute k means for $K \in [1, 2, \cdots, K_{max}]$
2. Compute the mean within cluster variance W_K for each selection of $K \in [1, K_{max}]$
3. The variances $[W_1, W_2, \cdots, W_{max}]$ generally decrease with increasing K. This will even be the case for an independent test set such that cross-validation cannot be used.

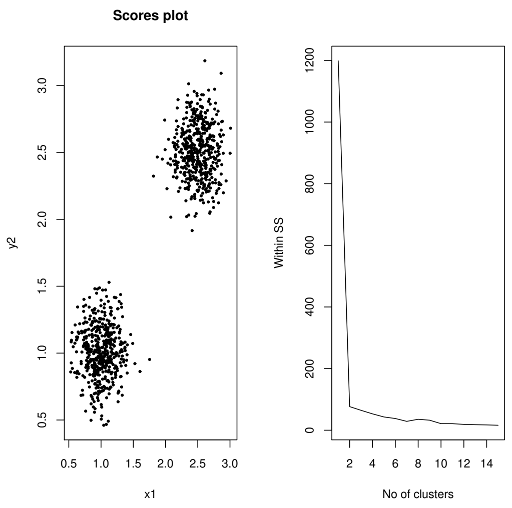
Optimal no of clusters

1. Intuitively, when $K < K^*$ we expect that an additional cluster will lower the within cluster variance: $W_{K+1} \ll W_K$.
2. When $K > K^*$ the decrease of the variance will be less evident.

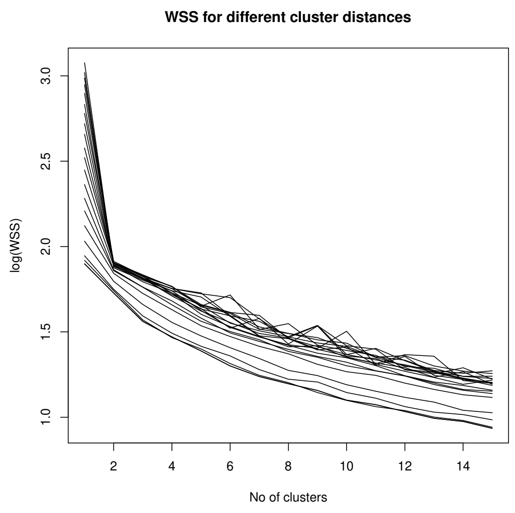
Optimal *n_clusters*

This means there will be flattening of the W_j curve. A sharp drop in the variance may be used to identify the optimal no. of clusters.

Optimal no of clusters, example



Optimal no of clusters, example



Self Organising Feature Mapping (SOFM)

A technique invented by T. Kohonen in 1982. It is used for performing non-linear unsupervised classification. The results are presented as 2D maps where the different classes are distributed as political geographical map of the Earth. The map consists of a matrix of neurons that compete for the samples.

Typical application areas are:

1. Biological taxonomy
2. Chemistry
3. Image analysis
4. Geo-plotting

SOFM algorithm

1. Initialize network by setting all weights to random numbers. However note that

$$w_j(0) \neq w_k(0), \ \forall k \neq j$$

$i \in 1, 2, \cdots, N$ where N is the number of nodes in the lattice

1. Take one sample x from the training (calibration) data set
2. What node i is most similar to $\mathbf{x}$? We look at $\|\mathbf{x} - \mathbf{w}_j\|$

for nodes $j \in [1, 2, \cdots, N]$

1. Adjust the connection weights:

$$\mathbf{w}_j(n+1) = \begin{cases} \mathbf{w}_j(n) + \eta(n)(\mathbf{x} - \mathbf{w}_j(n)) & \forall j \in \Lambda_i(n) \\ \mathbf{w}_j(n) & \text{otherwise} \end{cases}$$

$\Lambda_i(n)$ is the neighbourhood function centered around the winning node i. Both $\eta(n)$ and $\Lambda_i(n)$ vary dynamically during learning. $\Lambda_i(n)$ becomes smaller - a shrinking effect

SOFM map