Fundaments of Machine learning for and with engineering applications

Reidar B. Bratvold, Enrico Riccardi[1]

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

Feb 26, 2024

Loss function

One of the main core concept behind machine learnign is the **loss function**.

In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. [Wiki]

Loss function in Machine Learning

It quantifies the difference between the predicted outputs of a machine learning algorithm and the actual target values.

Loss function aims

It porovides a set of core quantifications/possibilities:

1. Performance measurement: it provides the metric of the prediction performances
2. Direction for improvement: it allows the identification of convergent solutions
3. Balancing bias and variance: it allows to account for artifact in the sampling phase
4. Influencing model behavior: it allows to bridge data driven methods with mathematics/physics

Loss function aims

It porovides a set of core quantifications/possibilities:

1. Performance measurement: it provides the metric of the prediction performances
2. Direction for improvement: it allows the identification of convergent solutions
3. Balancing bias and variance: it allows to account for artifact in the sampling phase
4. Influencing model behavior: it allows to bridge data driven methods with mathematics/physics

Loss function aims

It porovides a set of core quantifications/possibilities:

1. Performance measurement: it provides the metric of the prediction performances
2. Direction for improvement: it allows the identification of convergent solutions
3. Balancing bias and variance: it allows to account for artifact in the sampling phase
4. Influencing model behavior: it allows to bridge data driven methods with mathematics/physics

Loss function aims

It porovides a set of core quantifications/possibilities:

1. Performance measurement: it provides the metric of the prediction performances
2. Direction for improvement: it allows the identification of convergent solutions
3. Balancing bias and variance: it allows to account for artifact in the sampling phase
4. Influencing model behavior: it allows to bridge data driven methods with mathematics/physics

Loss function aims

It porovides a set of core quantifications/possibilities:

1. Performance measurement: it provides the metric of the prediction performances
2. Direction for improvement: it allows the identification of convergent solutions
3. Balancing bias and variance: it allows to account for artifact in the sampling phase
4. Influencing model behavior: it allows to bridge data driven methods with mathematics/physics

Most common loss function in machine learning

Most popular entries:

1. Mean Square Error (regression): Fast computations, good convergence.
2. Mean Absolute Error (regression): No focus on the outliers, poor convergence.
3. Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

Most common loss function in machine learning

Most popular entries:

1. Mean Square Error (regression): Fast computations, good convergence.
2. Mean Absolute Error (regression): No focus on the outliers, poor convergence.
3. Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

Most common loss function in machine learning

Most popular entries:

1. Mean Square Error (regression): Fast computations, good convergence.
2. Mean Absolute Error (regression): No focus on the outliers, poor convergence.
3. Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

Most common loss function in machine learning

Most popular entries:

1. Mean Square Error (regression): Fast computations, good convergence.
2. Mean Absolute Error (regression): No focus on the outliers, poor convergence.
3. Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

Losses

Mean absolute error, L1 loss

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|y_i - \bar{y}|)$$

Mean square error, L2 loss

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Log Loss for binary classification

$$Log\_Loss = -[y \ log(f(x)) + (1-y)log(1-f(x))]$$

MSE in Vanilla Python

```
def mean_squared_error(observed, predicted):
    """ Compute the mean squared error.

    Paramets:
    ---------
    observed : list
        The observed values.
    predicted : list
        The predicted values

    Output:
    ------
    mse : float
       mean squared error.

    """
    if len(observed) != len(predicted):
        raise ValueError(
            "The lengths of input lists are" +
            f"not equal {len(observed)} {len(predicted)}.")

    # Initialize the sum of squared errors
    sum_squared_error = 0
```

MSE in Vanilla Python

```python
    sum_squared_error = 0

    # Loop through all observations
    for obs, pred in zip(observed, predicted):
        # Calculate the square difference, and add it to the sum
        sum_squared_error += (obs - pred) ** 2

    # Calculate the mean squared error
    mse = sum_squared_error / len(observed)

    return mse
```

MSE in Numpy Python

```python
import numpy as np

def mean_squared_error(observed, predicted):
    observed_np = np.array(observed)
    predicted_np = np.array(predicted)
    mse = np.mean((observed_np - predicted_np) ** 2)
    return mse
```

MSE in Python

```python
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
import sklearn.metrics as sk

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]


mse_vanilla = vanilla_mse(observed, predicted)
print("Mean Squared Error, vanilla :", mse_vanilla)

mse_numpy = numpy_mse(observed, predicted)
print("Mean Squared Error, numpy :   ", mse_numpy)

sk_mse = sk.mean_squared_error(observed, predicted)
print("Mean Squared Error, sklearn :", sk_mse)

assert(mse_vanilla == mse_numpy == sk_mse)
```

MSE in benchmark

```python
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
import sklearn.metrics as sk
import timeit as it

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]

mse_vanilla = vanilla_mse(observed, predicted)
time_v = it.timeit('vanilla_mse(observed, predicted)',
                   globals=globals(), number=10) / 100
mse_numpy = numpy_mse(observed, predicted)
time_np = it.timeit('numpy_mse(observed, predicted)',
                    globals=globals(), number=10) / 100
sk_mse = sk.mean_squared_error(observed, predicted)
time_sk = it.timeit('sk.mean_squared_error(observed, predicted)',
                    globals=globals(), number=10) / 100

for mse, mse_type, time in zip([mse_vanilla, mse_numpy, sk_mse],
                               ['vanilla', 'numpy', 'sklearn'],
                               [time_v, time_np, time_sk]):
    print(f"Mean Squared Error, {mse_type}:", mse,
          f"Average execution time: {time} seconds")

assert(mse_vanilla == mse_numpy == sk_mse)
```

Coding MSE

```python
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
from sklearn.metrics import mean_squared_error as sk_mse
import timeit as it

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]

karg = {'observed': observed, 'predicted': predicted}

factory = {'mse_vanilla' : vanilla_mse,
           'mse_numpy' : numpy_mse,
           # 'mse_sk' : sk_mse
           }

for talker, worker in factory.items():
    exec_time = it.timeit('{worker(**karg)}',
                          globals=globals(), number=100) / 100
    mse = worker(**karg)
    print(f"Mean Squared Error, {talker} :", mse,
          f"Average execution time: {exec_time} seconds")
```

Coding MSE

```python
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
from sklearn.metrics import mean_squared_error as sk_mse
import timeit as it

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]

karg = {'observed': observed, 'predicted': predicted}

def sk_mse_interface(observed, predicted):
    return sk_mse(observed, predicted)

factory = {'mse_vanilla' : vanilla_mse,
           'mse_numpy' : numpy_mse,
           'mse_sk' : sk_mse_interface
           }

for talker, worker in factory.items():
    exec_time = it.timeit('{worker(**karg)}',
                          globals=globals(), number=100) / 100
    mse = worker(**karg)
    print(f"Mean Squared Error, {talker} :", mse,
          f"Average execution time: {exec_time} seconds")
```

Regularization

Conventionally, MSE is used as loss function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

We might want to force our model not not fit so well the data, i.e. we *penalize* the model.

$$L1\_loss = MSE + \lambda \sum_{i=1}^{m} b_i^2$$

$$L2\_loss = MSE + \lambda \sum_{i=1}^{m} |b_i|$$

$$Elastic\ Net = MSE + \lambda_1 \sum_{i=1}^{m} |b_i| + \lambda_2 \sum_{i=1}^{m} b_i^2$$

Regularization

λ is the regularization strenght. The larger value it has, the more the target function is "wrong". For $\lambda = 0$, no regularization is imposed.

Forcing a model to be "wrong" might sound rather counter-intuitive. Furthermore, we have introduced one more parameter that has not much meaning?!

Bias - variance trade off

$$Error = bias^2 + variance + statistical\ error$$

Regularization

λ is the regularization strenght. The larger value it has, the more the target function is "wrong". For $\lambda = 0$, no regularization is imposed.

Forcing a model to be "wrong" might sound rather counter-intuitive. Furthermore, we have introduced one more parameter that has not much meaning?!

Bias - variance trade off

$$Error = bias^2 + variance + statistical\ error$$

Regularization

λ is the regularization strenght. The larger value it has, the more the target function is "wrong". For $\lambda = 0$, no regularization is imposed.

Forcing a model to be "wrong" might sound rather counter-intuitive. Furthermore, we have introduced one more parameter that has not much meaning?!

Bias - variance trade off

$$Error = bias^2 + variance + statistical\ error$$

Bias

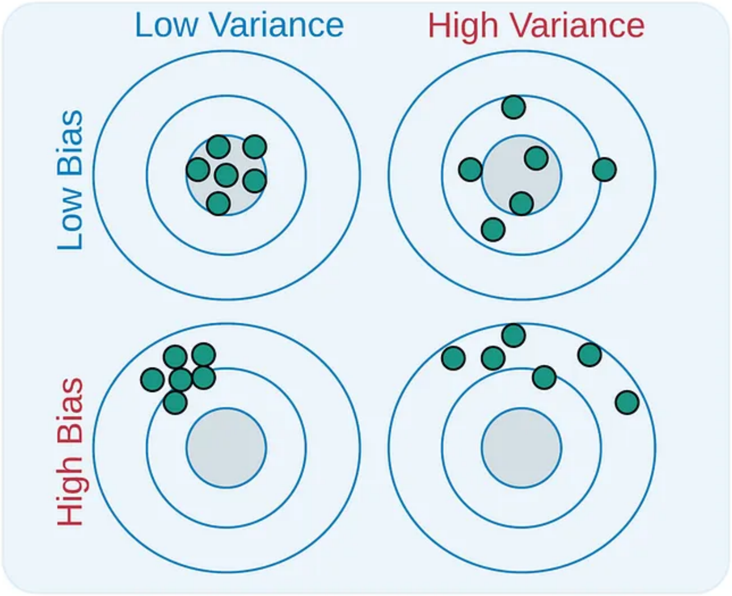
Bias is the difference of the average value of predictions (q) from the true relationship function (f):

$$bias[q(x)] = \mathbb{E}[q(x)] - f(x)$$

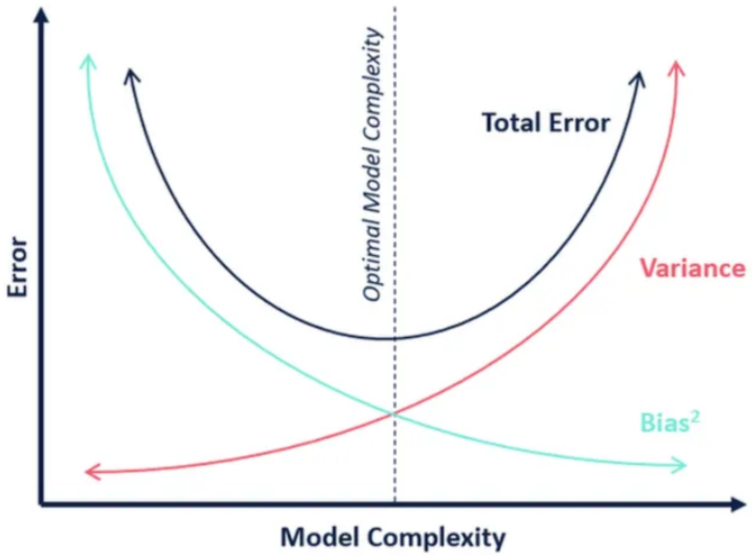
Variances is the expectation of the squared deviation of q(x) from its expected value $\mathbb{E}[q(x)]$.

$$var[q(x)] = \mathbb{E}[(q(x) - \mathbb{E}[q(x))^2]$$

Bias-Variance



Bias-Variance



Classification types

There are two main types of classification methods for analysis of a set of objects stored in matrix X:

Unsupervised classification

- Only the X data is used
- "Natural" classes/clusters/groupings in X are discovered

Supervised classification

- We know the class/group/cluster membership of every object/sample
- Class information is stored in an Y matrix

Classification types

There are two main types of classification methods for analysis of a set of objects stored in matrix X:

Unsupervised classification

- Only the X data is used
- "Natural" classes/clusters/groupings in X are discovered

Supervised classification

- We know the class/group/cluster membership of every object/sample
- Class information is stored in an Y matrix

Classification types

There are two main types of classification methods for analysis of a set of objects stored in matrix X:

Unsupervised classification

- Only the X data is used
- "Natural" classes/clusters/groupings in X are discovered

Supervised classification

- We know the class/group/cluster membership of every object/sample
- Class information is stored in an Y matrix

Classification types

Several approaches can be found in classification tasks:

Unsupervised classification

- Principal component analysis (PCA)
- Agglomerative (hierarchical) cluster analysis.
- k-means cluster analysis
- Fuzzy c-means cluster analysis
- Self organising feature maps (SOFM)

Supervised classification

- Linear discriminant analysis (LDA)
- k-nearest neighbours (kNN)
- Discriminant partial least squares
- Soft independent modeling of class analogies (SIMCA)

Classification types

Several approaches can be found in classification tasks:

Unsupervised classification

- Principal component analysis (PCA)
- Agglomerative (hierarchical) cluster analysis.
- k-means cluster analysis
- Fuzzy c-means cluster analysis
- Self organising feature maps (SOFM)

Supervised classification

- Linear discriminant analysis (LDA)
- k-nearest neighbours (kNN)
- Discriminant partial least squares
- Soft independent modeling of class analogies (SIMCA)

Classification types

Several approaches can be found in classification tasks:

Unsupervised classification

- Principal component analysis (PCA)
- Agglomerative (hierarchical) cluster analysis.
- k-means cluster analysis
- Fuzzy c-means cluster analysis
- Self organising feature maps (SOFM)

Supervised classification

- Linear discriminant analysis (LDA)
- k-nearest neighbours (kNN)
- Discriminant partial least squares
- Soft independent modeling of class analogies (SIMCA)

Limitations of unsupervised classification

- Do "natural" clusters in a data set exists and/or have any meaning?
- First we must have a definition of what is a cluster. To do this we

must define what we mean by similar or dissimilar objects.

- Objects that are close have low dissimilarity and high similarity.

A metric system is required.

Limitations of unsupervised classification

- Do "natural" clusters in a data set exists and/or have any meaning?
- First we must have a definition of what is a cluster. To do this we

must define what we mean by similar or dissimilar objects.

- Objects that are close have low dissimilarity and high similarity.

A metric system is required.

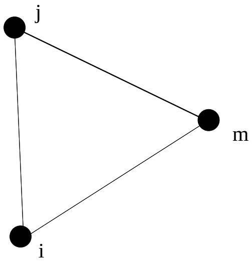
Proximity: Continuous variables

Triangle inequality

Considering a vectors in an N-dimensional space, to be a distance it must satisfy the triangle inequality:

$$d_{ij} + d_{im} \geq d_{jm}$$

If also $d_{jj} = 0$ we call it a *metric*.



Proximity: Continuous measures

Common metrics:

- Euclidean.

$$d_{ij}^{(E)} = \left[\sum_{k=1}^{N} (x_{ik} - x_{jk})^2\right]^{\frac{1}{2}}$$

- Manhattan

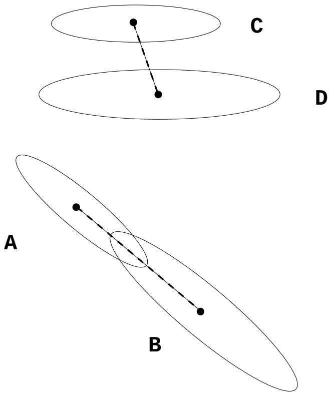
$$d_{ij}^{(M)} = \sum_{k=1}^{N} \|x_{ik} - x_{jk}\|$$

- Minkowski

$$d_{ij}^{(M(p))} = \left[\sum_{k=1}^{N} (x_{ik} - x_{jk})^p\right]^{\frac{1}{p}}$$

Mahalanobis distance

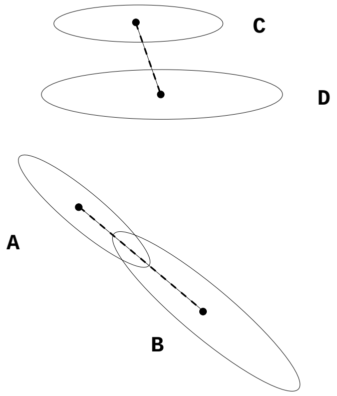
P.C. Mahalanobis invented in 1936 a distance measure which takes into consideration the covariance of a population when computing the distance between two vectors:



The Euclidean distance from C to D is shorter than A to B, but the Mahalanobis distance A-B is smaller than C-D because A and B are oriented along the same direction.

Mahalanobis distance

P.C. Mahalanobis invented in 1936 a distance measure which takes into consideration the covariance of a population when computing the distance between two vectors:



The Euclidean distance from C to D is shorter than A to B, but the Mahalanobis distance A-B is smaller than C-D because A and B are oriented along the same direction.

Proximity: Categorical variables

- Many applications consist of binary vectors, typical is "yes" and "no" answers to a lot of tests
- It is tempting to use distance between binary vectors to signify distance. However that is *by far* not optimal.

Lets look at an example:

- $\mathbf{v}_1 = [1\ 1\ 0\ 0\ 0\ 0]$
- $\mathbf{v}_2 = [0\ 0\ 1\ 1\ 0\ 0]$
- $\mathbf{v}_3 = [1\ 1\ 1\ 1\ 1\ 1]$

It makes NO SENSE to compute the Euclidean distances between these vectors

Proximity: Categorical variables

- Many applications consist of binary vectors, typical is "yes" and "no" answers to a lot of tests
- It is tempting to use distance between binary vectors to signify distance. However that is *by far* not optimal.

Lets look at an example:

- $\mathbf{v}_1 = [1\ 1\ 0\ 0\ 0\ 0]$
- $\mathbf{v}_2 = [0\ 0\ 1\ 1\ 0\ 0]$
- $\mathbf{v}_3 = [1\ 1\ 1\ 1\ 1\ 1]$

It makes NO SENSE to compute the Euclidean distances between these vectors

Proximity: Categorical variables: Binary matching

	Object B value 1	*Object B* value 0
Object A value 1	a	b
Object A value 0	c	d

Example

$\mathbf{a} = [0\ 0\ 0\ 1]$

$\mathbf{b} = [1\ 1\ 0\ 1]$

- $c = 2$: two places where A has 0 and B has 1.
- $d = 1$: one place where A and B are equal to 0.
- $a = 1$: one place where A and B are equal to 1.

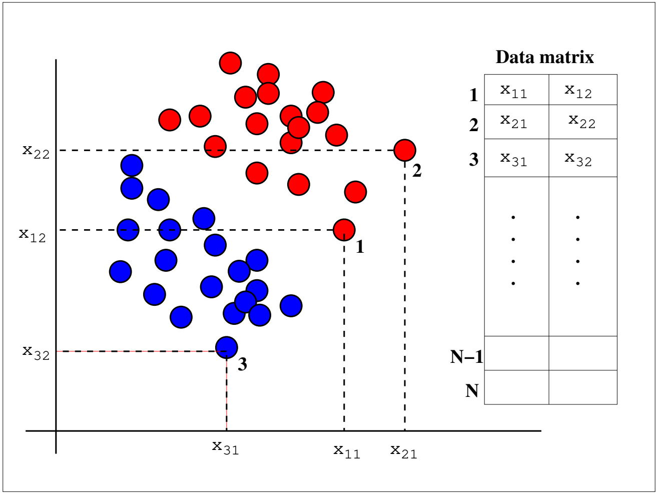
Binary proximity measures

Types of binary proximity measures

Binary proximity measure	*Formula*
Matching coefficient	$d_{AB} = \frac{a+d}{a+b+c+d}$
Jackard	$d_{AB} = \frac{a}{a+b+c}$
Rogers and Tanimoto	$d_{AB} = \frac{a+d}{a+2(b+c)}$
Sokal and Sneath	$d_{AB} = \frac{a}{a+2(b+c)}$

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**

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

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.:

$$\mathrm{LV} = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

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.:

$$\mathrm{LV} = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

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

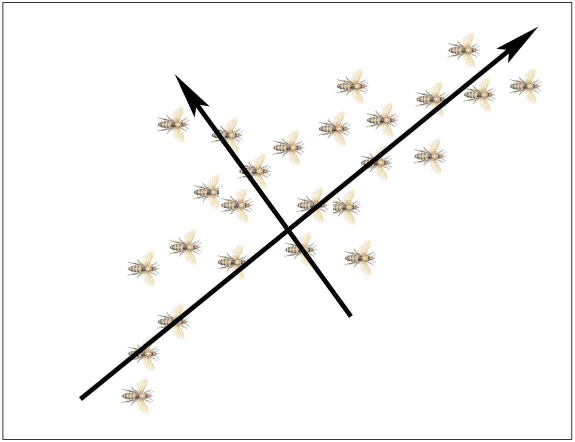
Goals of PCA:

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

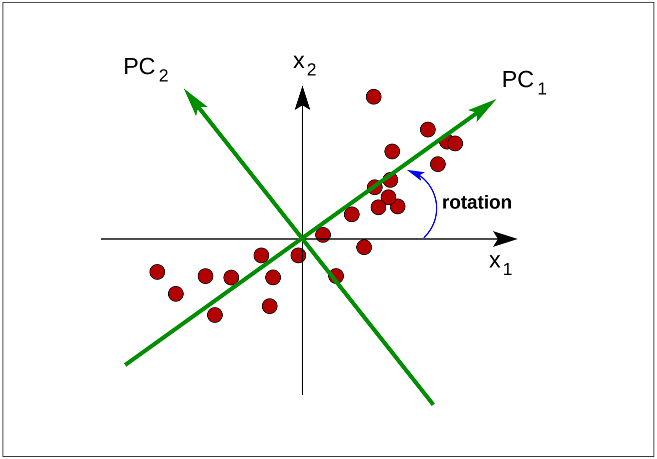
PCA

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



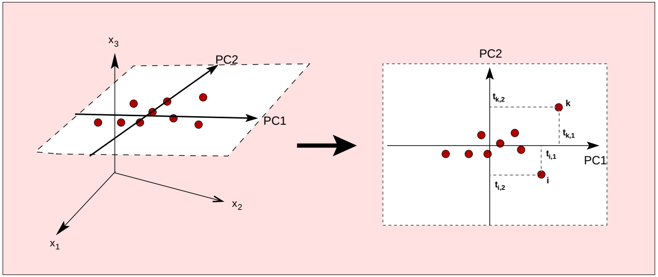
Rotation of the coordinate system



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:

$$t_i = p_1 x_1 + p_2 x_2 + \cdots + p_M x_M$$

Model description

The PCA model

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

where

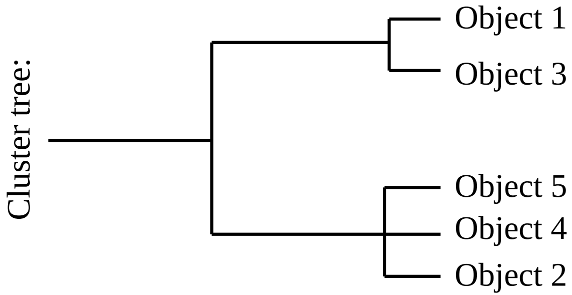
$\mathbf{X}$ is the data matrix $\mathbf{T}$ is the scores matrix $\mathbf{P}$ is the loadings matrix $\mathbf{E}$ is the residual matrix

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

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

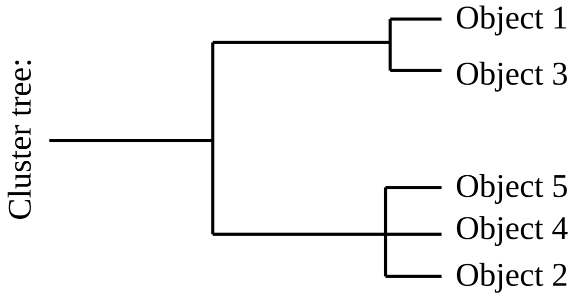
This can be summarised in a hierarchical cluster tree.



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
```

Distance between clusters

What is the distance between one cluster of objects to the next? The most common approaches are:

1. Single linkage
2. Complete linkage
3. Group average (unweighted pair group method average, UPGMA)
4. Ward's method

Cluster distances

Cluster distances

Single linkage

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

Complete linkage

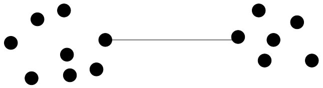
$$d_{AB} = \max(f_{i_A, j_B}), \forall i \in A, j \in B$$

Group average (UPGMA)

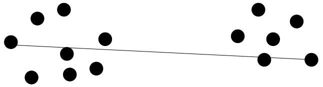
$$d_{AB} = \frac{1}{nm} \sum_{i \in A} \sum_{j \in B} f_{i_A, j_B},$$

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

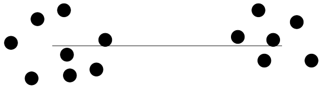
Single linkage (closest neighbour)



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:

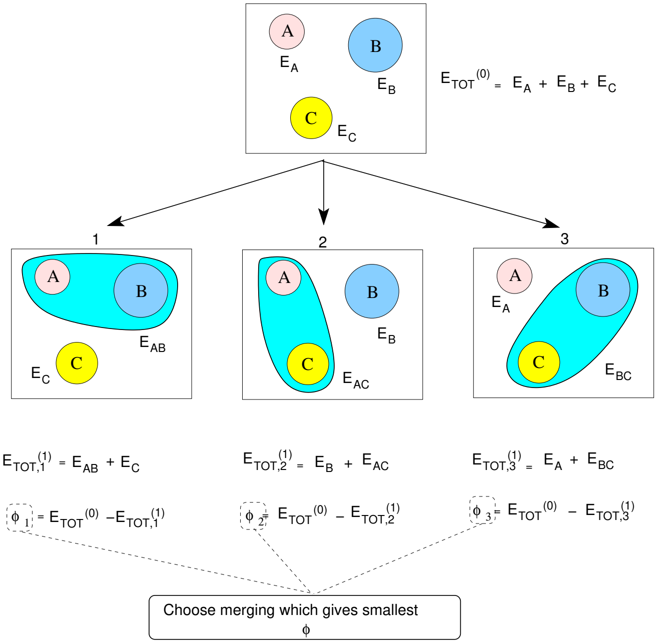
$$\phi = E_{tot}^{(1)} - E_{tot}^{(0)}$$

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

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

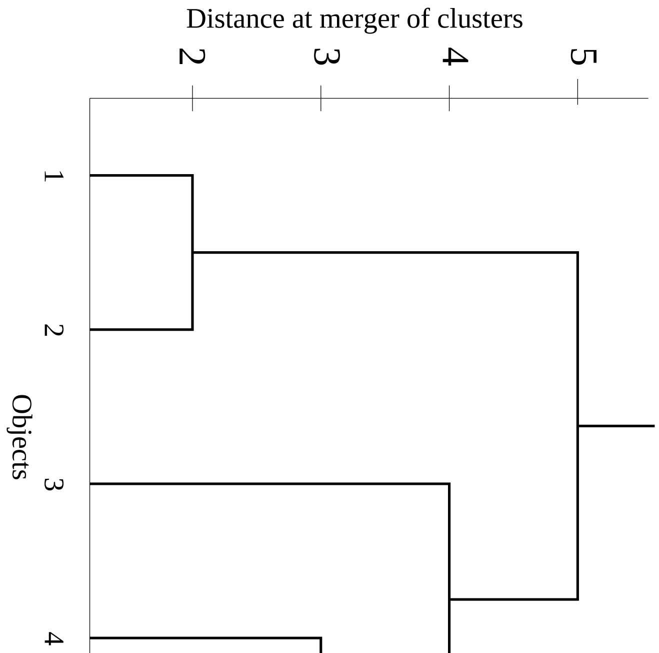
number of such possible pairs of clusters. The pair A and B which gave the smallest number is chosen to be merged.

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 centers $\mathbf{m}_k$
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.

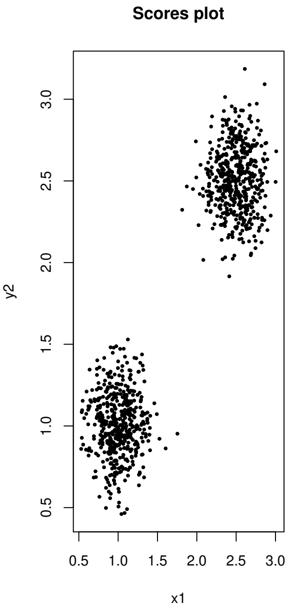
Optimal no of clusters

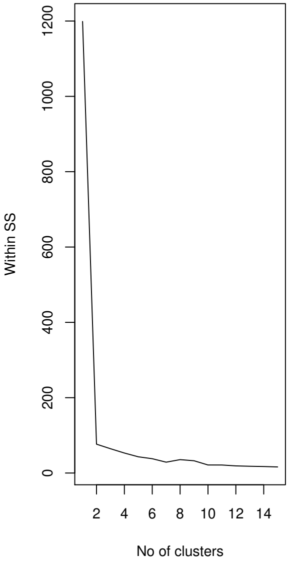
1. Intuitively, when $K < K^*$ we expect that an additional cluster will lower the within cluster variance: $W_{K+1} \ll W_K$.
2. When $K > K^*$ the decrease of the variance will be less evident.

Optimal *n_clusters*

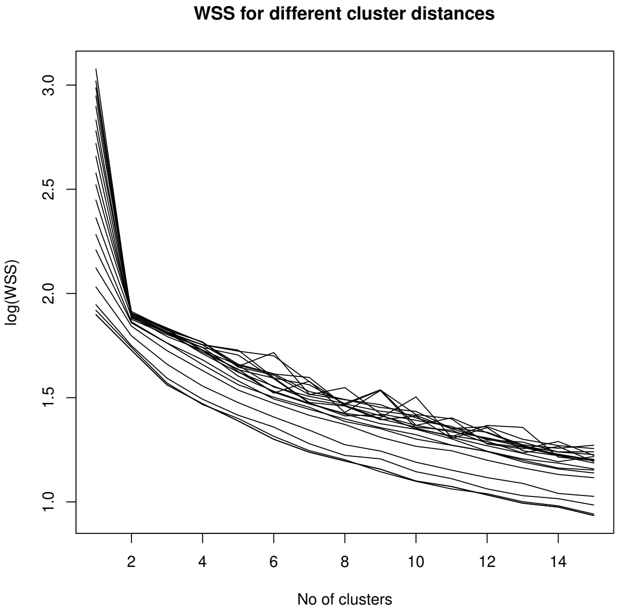
This means there will be flattening of the W_j curve. A sharp drop in the variance may be used to identify the optimal no. of clusters.

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

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

$$\mathrm{w}_j(0) \neq \mathrm{w}_k(0),\ \forall k \neq j$$

$i \in 1, 2, \cdots, N$ 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 $\|\mathbf{x} - \mathbf{w}_j\|$

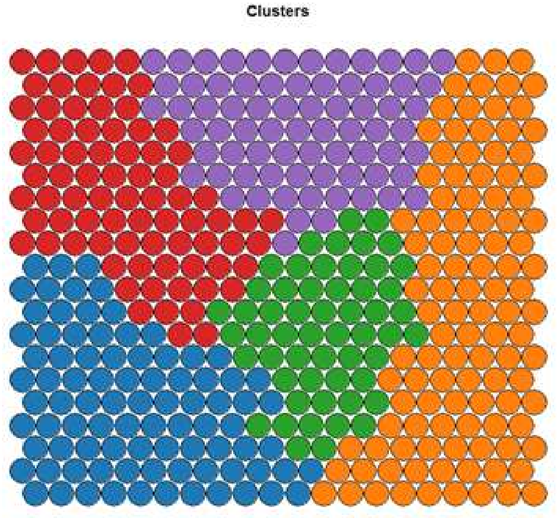
for nodes $j \in [1, 2, \cdots, N]$

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



Fisher's method

Fisher's approach is to find a plane which minimises the **within-class variance** and maxmimises the **between-class variance**. In other words: We want to find new coordinates **canonical variates** which produces tight clusters that are far from each other. These are **latent variables** which are best suited to separate the classes.

Method called Fisher's Linear Discriminant Analysis (LDA), Discriminant Function Analysis (DFA) or Canonical Discriminant Analysis (CDA)

Fisher's Method

1. Transformation performed in $K-1$ dimensions where K is the number of classes.
2. No assumptions of distributions are needed
3. New coordinates based on "importance of discrimination" and "maximum variation"
4. Hopefully objects are more separated into classes in the new coordinates (scores).
5. Fisher's method can also be used as a postprocessing step, i.e. analysis of results from e.g. sophisticated non-linear classification methods.

Fisher's method

The new variables can be seen as a linear combination of the original variables:

$$z_i = \sum_{j=1}^{M} X_{ij} r_j = \mathbf{X}\mathbf{r} \tag{1}$$

where **r** is the direction of the new DFA coordinate system. It is analogous to a principal component. What criterion should be used to compute **r**?

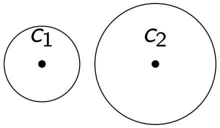
Fisher's method

To have isolated class clusters we want two things: Good separation of the mean centers and clusters with little variance (i.e. tight clusters with little variance). This corresponds to maximizing the between class variance and minimize the within class variance. This is closely linked to analysis of variance (ANOVA). To compare variance we can make use of the F ratio which we want to maximise:

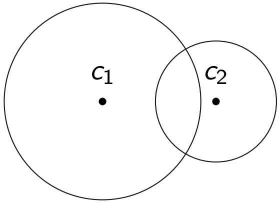
$$F_{max} = \arg\max_{\mathbf{r}} \left(\frac{\mathbf{r}^T \mathbf{B} \mathbf{r}}{\mathbf{r}^T \mathbf{W} \mathbf{r}} \right) \tag{2}$$

where $\mathbf{r}^T \mathbf{B} \mathbf{r}$ is the between-class variance and $\mathbf{r}^T \mathbf{W} \mathbf{r}$ is the within-class variance with the constraint that $\mathbf{r}_i \mathbf{W} \mathbf{r}_j = \delta_{ij}$.

Fisher's method



A



B

Fisher's method

We have that:

$$\mathbf{B} = \sum_{i=1}^{K} \frac{n_i}{N} \left(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}\right) \left(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}\right)^T \tag{3}$$

where $\bar{\mathbf{x}}_i$ is the mean vector of class i objects, $\bar{\mathbf{x}}$ is the mean vector for *all* objects and

$$\mathbf{W} = \sum_{i=1}^{K} \frac{n_i}{N} \mathbf{C}_i \tag{4}$$

where $\mathbf{C}_i$ is the covariance matrix of class i.

Fisher's method

To find the vectors which maximise the F function above we make use of **eigenvalue decomposition**. To do it we need generalised eigenvector decomposition:

$$\mathbf{B}\mathbf{R} = \mathbf{W}\mathbf{R}\Lambda \tag{5}$$

where $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_{K-1}]$. Multiplying with $\mathbf{W}^{-1}$ on both sides we get:

$$\mathbf{W}^{-1}\mathbf{B}\mathbf{R} = \mathbf{R}\Lambda. \tag{6}$$

where we assume that $\mathbf{W}^{-1}$ exists.

Fisher's method

In many cases this may not be the case, in particular for problems where we typically have many more variables than samples. The solution $\mathbf{R}$ of the above equation can also diagonalise $\mathbf{B}$:

$$\mathbf{R}^T\mathbf{B}\mathbf{R} = \Lambda. \tag{7}$$

Note that $\mathbf{W}^{-1}\mathbf{B}$ in general is not a symmetric matrix. We can convert it into one by performing Cholesky decomposition using a lower triangular matrix $\mathbf{L}$ and assuming:

$$\mathbf{W} = \mathbf{L}\mathbf{L}^T. \tag{8}$$

Fisher's method

If $\mathbf{W}$ is close to singular then the eigenvectors of $\mathbf{W}^{-1}\mathbf{B}$ cannot be computed properly and we need to handle the problem. Two popular approaches are:

1. Perform pseudo-inverse using SVD or PCA such that $\mathbf{W}^{-1}$ is replaced with $\mathbf{W}^{+}$ where $\mathbf{W}^{+} = \mathbf{V}\mathbf{S}^{+}\mathbf{U}^{T}$. PCA can be used as a preprocessing step where the scores matrix $\mathbf{T}$ is used instead of the original data matrix $\mathbf{X}$ (see below).
2. Use of perturbation where the $\mathbf{W}$ is stabilised by assuming a small perturbation matrix $\mathbf{G}$ to it.

Fisher's method

1. Problem: The new coordinates **R** do not perform classification. Actual assignment of class memberships to new objects is not performed.
2. Suggested solution: Create discriminant functions which involve the **R**.
3. The discriminant functions become:

$$g_i(\mathbf{x}) = \ln p(c_i) - \frac{1}{2}\overline{\mathrm{x}}_i^T \mathbf{R}\mathbf{R}^T \overline{\mathbf{x}}_i + \mathbf{x}^T \mathbf{R}\mathbf{R}^T \overline{\mathbf{x}}_i$$

Using PCA scores in DFA

1. Very often the number of variables is much larger than the number of

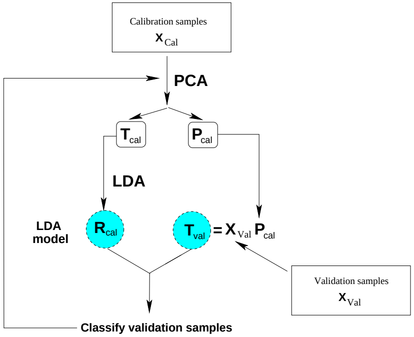
objects so we need to find a solution.

1. A popular solution is to use PCA where the scores vectors contained in the matrix **T** are used instead of the original data matrix **X**.
2. This is possible since the scores vectors are **linearly independent** or **orthogonal**.

Using PCA scores in DFA

The optimal no. of PCA factors is determined iteratively:

1. Use *A* no. of PCA factors and produce scores matrix **T**
2. Use **T** in the LDA routine
3. Assess by using independent data
4. *A* selected with lowest classification error



Calibration samples
$\mathbf{X}_{Cal}$
PCA
$\mathbf{T}_{cal}$
$\mathbf{P}_{cal}$
LDA
LDA model
$\mathbf{R}_{cal}$
$\mathbf{T}_{val} = \mathbf{X}_{Val}\,\mathbf{P}_{cal}$
Validation samples
$\mathbf{X}_{Val}$
Classify validation samples