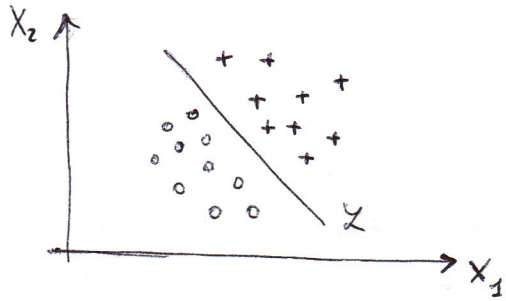
SUPPORT VECTOR MACHINES

28/04

Consider a dichotomous (2 groups) supervised classification problem.
Training set:

$$\mathbb{X} = \begin{bmatrix} \underline{x}_1^T & \ell_1 \\ \vdots & \vdots \\ \underline{x}_n^T & \ell_n \end{bmatrix} \qquad \underline{x}_i \in \mathbb{R}^p \qquad \ell_i \in \{1,2\}$$



Motivating idea:
find a hyperplane separating the two groups.
The hyperplane defines a partition of $\mathbb{R}^p$
$\{R_1, R_2\} \iff \delta$
(the partition is equivalent to a classifier)

- Given two sets $A, B \subseteq \mathbb{R}^p$, when can be they separated by an hyperplane?

Let $CH(A)$ and $CH(B)$ be the convex hulls containing A and B.
(smallest convex set)
if:

- $CH(A) \neq \emptyset$, $CH(B) \neq \emptyset$
- $CH(A) \cap CH(B) = \emptyset$
- Either $CH(A)$ or $CH(B)$ is open

(this is the **Hahn-Banach** theorem)
(geometric form)

$\Longrightarrow$ $\exists$ a separation hyperplane

Obs. condition 3. can be replaced with:

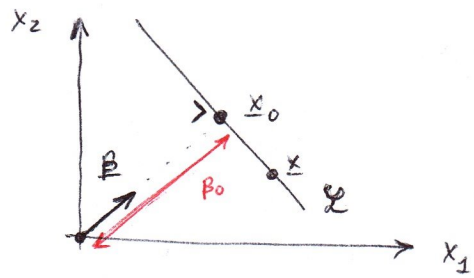
- $CH(A)$ and $CH(B)$ are closed and at least one of them is compact

This condition is satisfied if A and B have finite cardinality.

- Suppose $\exists$ hyperplane, how to characterize it?

HYPERPLANES

Let $\mathcal{L}$ be an hyperplane in $\mathbb{R}^p$: affine subspace of $\mathbb{R}^p$ of dim $= p-1$.
To identify $\mathcal{L}$ we introduce a vector $\beta \in \mathbb{R}^p$, $\beta \perp \mathcal{L}$, $\|\beta\| = 1$



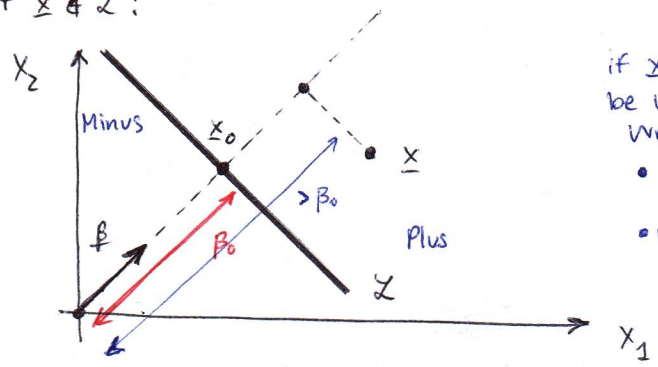
$\underline{x}_0 \in \text{span}(\beta) \cap \mathcal{L}$
$\beta_0 = \|\underline{x}_0\|$

$\underline{x} \in \mathcal{L} \iff \Pi_{\underline{x}|\beta} = \underline{x}_0$

$$\underline{x} \in \mathcal{L} \iff \underline{\beta}^T \underline{x} = \beta_0$$

$$\iff \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p = \beta_0 \qquad \text{for } \underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

what if $\underline{x} \notin \mathcal{L}$?

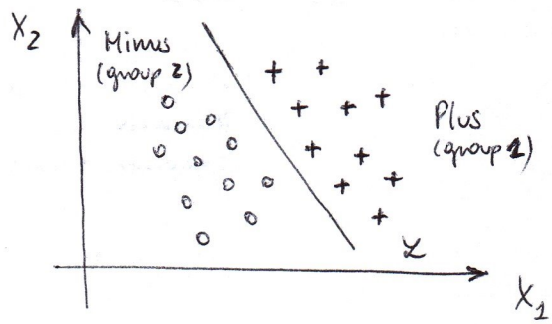


if $\underline{x} \notin \mathcal{L}$ then the projection on $\underline{\beta}$ will be larger/smaller than β_0.
We're defining two classes (a partition of $\mathbb{R}^p$):
- plus: zone (class) where points have a projection larger than β_0
- minus: " smaller than β_0

If $\underline{x} \notin \mathcal{L}$ then either:
- $\underline{\beta}^T \underline{x} > \beta_0 \iff \underline{x} \in$ Plus
- $\underline{\beta}^T \underline{x} < \beta_0 \iff \underline{x} \in$ Minimus

In fact $\underline{\beta}^T \underline{x} - \beta_0$ measures (with sign (±)) the distance between $\underline{x}$ and $\mathcal{L}$.

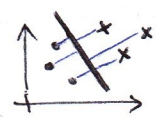
let's now go back to $\mathcal{X}$. We want to do a reparametrization.
Suppose there exists a separating hyperplane $\mathcal{L}$.



let $y_i = +1$ if $\ell_i = 1$
$y_i = -1$ if $\ell_i = 2$.

With this reparametrization:

$$y_i \left(\underline{\beta}^T \underline{x}_i - \beta_0 \right) \geq 0 \qquad \text{for } i = 1, \dots, n$$

and it is the distance between $\underline{x}_i$ and $\mathcal{L}$. ( it's the blue distance)

let $M_1 = \min \{ y_i (\underline{\beta}^T \underline{x}_i - \beta_0) : i = 1, \dots, n \}$

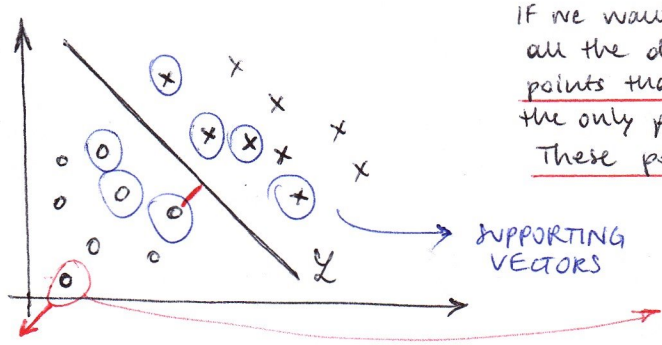
We want to find an hyperplane s.t. the smallest of these distances (:= M_1) is as small as possible.

Optimal separating hyperplane:
find $\mathcal{L}$ (find $\underline{\beta}$ and β_0) s.t. M_1 is max.

Equivalently:

$$\max_{\underline{\beta}, \beta_0} M \quad \text{s.t.} \quad \begin{cases} \|\underline{\beta}\| = 1 \\ y_i (\underline{\beta}^T \underline{x}_i - \beta_0) \geq M \end{cases} \qquad \text{for } i = 1, \dots, n$$

Note that:

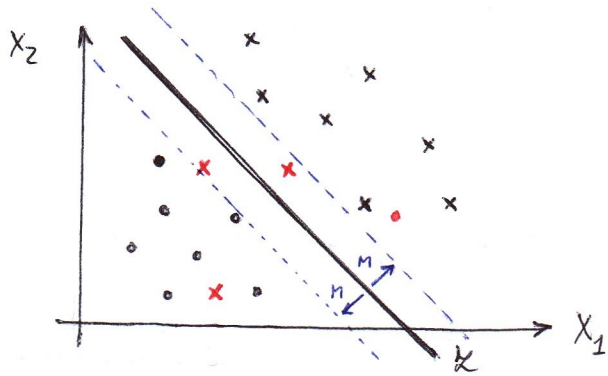


If we want to find $\mathcal{L}$ we don't have to consider all the dataset, it's enough to consider those points that are close to the boundary (those are the only points that can influence the linear space $\mathcal{L}$). These points are called **SUPPORTING VECTORS**

if we move the point along the red direction, $\mathcal{L}$ doesn't change
$\Rightarrow$ $\mathcal{L}$ is very robust w.r.t. modification of the dataset in those points wich are not supporting vectors
(**LDA** is not strong on this since the mean is not robust w.r.t. outliers)

- What if the two groups of $\mathbb{X}$ are not separable?

Approach **1.** : allow some overlapping



Can we allow for these red units that not respect the rules to stay on one side or the other of the boundary?
Yes, if we pay a **cost**.

Soft version of the optimization problem:

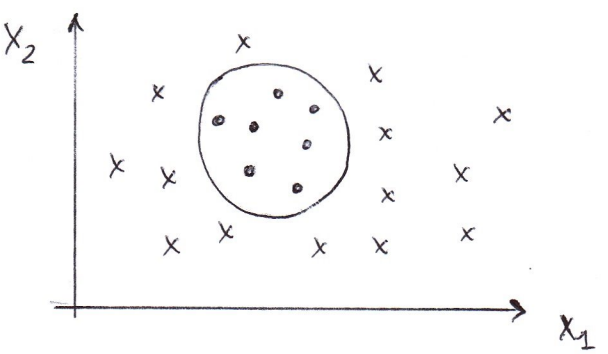
$$\max_{\beta, \beta_0} M \quad \text{s.t.} \begin{cases} \|\beta\| = 1 \\ y_i(\beta^T x_i - \beta_0) \geq M(1 - \varepsilon_i) \\ \varepsilon_i \geq 0 \\ \sum \varepsilon_i \leq C \end{cases}$$

$\longrightarrow$ if $\varepsilon_i > 1$ we allow negative M (meaning that we're on the other side of the boundary)

C: control parameter (smoothing parameter / penalization parameter) which control the overlap

$:=$ BUDGET CONSTRAINT

Approach **2.** : useful when we don't see clear linear boundaries between the two groups:



Be careful about what we mean by "linear":
we always think that a model (or something) must be linear w.r.t. the features, but these features are not necessarily those that we received: we can modify them, we can change them and obtain new features and with respect to these new features we use a linear model.

Example:

$$\underset{\text{training set}}{\mathbb{X}} = \begin{bmatrix} z_1 & z_2 & L \\ z_{11} & z_{12} & l_1 \\ z_{21} & z_{22} & l_2 \\ \vdots & & \\ z_{n1} & z_{n2} & l_n \end{bmatrix}$$

"received" data

We are free to transform the variables.

For instance:

$$X_1 = z_1, \quad X_2 = z_2, \quad X_3 = z_1^2, \quad X_4 = z_2^2, \quad X_5 = z_1 \cdot z_2,$$
$$X_6 = \sin(z_1), \ldots, \quad X_p = \log(z_2)$$

⟹ new training set

$$\mathbb{X} = \begin{bmatrix} X_1 & X_2 & X_3 & X_4 & \cdots & L \\ z_{11} & z_{12} & z_{11}^2 & z_{11}z_{22} & \cdots & l_1 \\ \vdots & \vdots & \vdots & & & \vdots \\ z_{n1} & z_{n2} & z_{n1}^2 & & \cdots & l_n \end{bmatrix}$$

$\mathcal{L}$: $\beta_1 X_1 + \ldots + \beta_p X_p = \beta_0$

$\beta_1 z_1 + \beta_2 z_2 + \ldots + \beta_3 z^2 + \beta_4 z_2^2 + \ldots = \beta_0$

⟹ use **Kernels** for exploring non-linearity

(RKHS): reproducing kernel Hilbert spaces

automatic tools for exploring non-linearity: in order to solve this optimization problem we have to compute "some" inner products. The only elements that enter in the solution are inner products between the data and the **X**'s. We have machines that produces a large class of inner products and these are called kernels.

Note:

Why do we want more features?
We're enlarging p (even if n remains the same) because we know (thanks to the curse of dimensionality) that if we enlarge p the points are going to be more far from each other and eventually we'll find a separating hyperplane.

⟹ the idea is to take a larger space to represent the same dataset (not adding informations but transforming informations)

what is the risk? **Overfitting**

Support Vector Machines

Here we approach the two-class classification problem in a direct way:

> *We try and find a plane that separates the classes in feature space.*

If we cannot, we get creative in two ways:

- We soften what we mean by "separates", and
- We enrich and enlarge the feature space so that separation is possible.

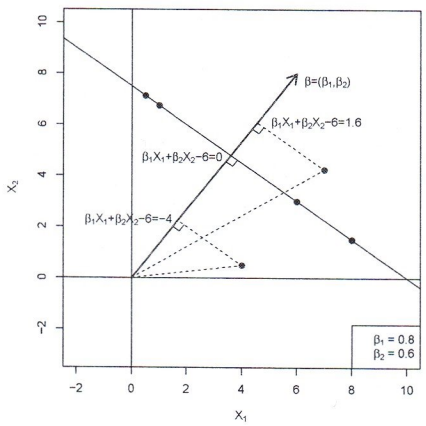
What is a Hyperplane?

- A hyperplane in p dimensions is a flat affine subspace of dimension $p-1$.
- In general the equation for a hyperplane has the form

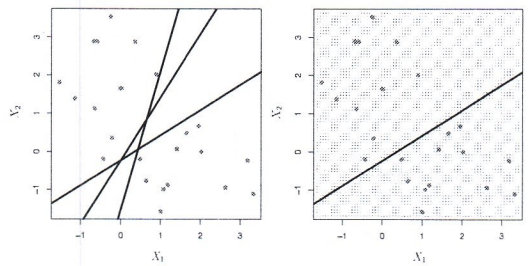
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

- In $p = 2$ dimensions a hyperplane is a line.
- If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
- The vector $\beta = (\beta_1, \beta_2, \cdots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.

Hyperplane in 2 Dimensions



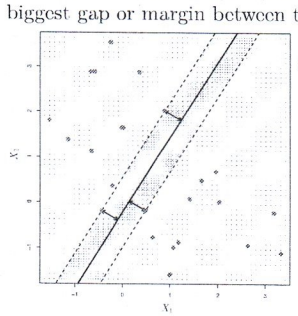
Separating Hyperplanes



- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for mauve, then if $Y_i \cdot f(X_i) > 0$ for all i, $f(X) = 0$ defines a *separating hyperplane*.

Maximal Margin Classifier

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.



Constrained optimization problem

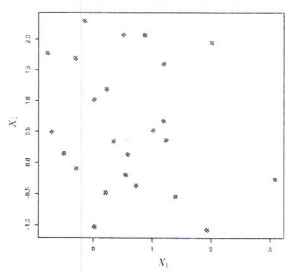
$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \ldots, N.$$

This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package `e1071` solves this problem efficiently

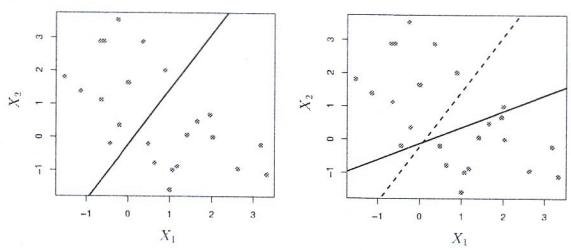
Non-separable Data



The data on the left are not separable by a linear boundary.

This is often the case, unless $N < p$.

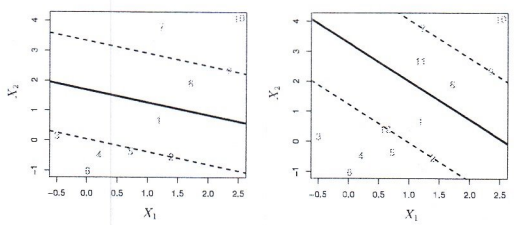
Noisy Data



Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

The *support vector classifier* maximizes a *soft* margin.

Support Vector Classifier

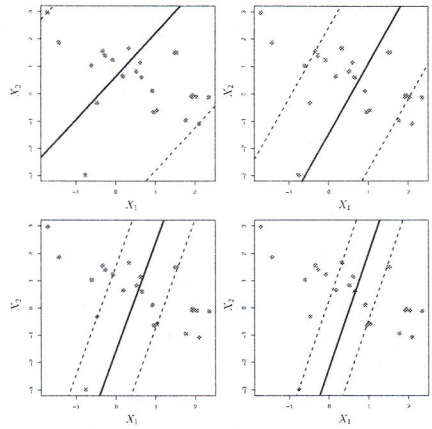


$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}}\; M \quad \text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

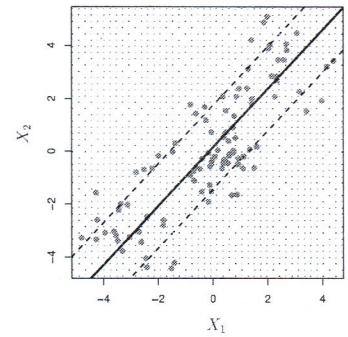
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \; \sum_{i=1}^{n} \epsilon_i \leq C,$$

C is a regularization parameter



Linear boundary can fail



Sometime a linear boundary simply won't work, no matter what value of C.

The example on the left is such a case.

What to do?

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,.... Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

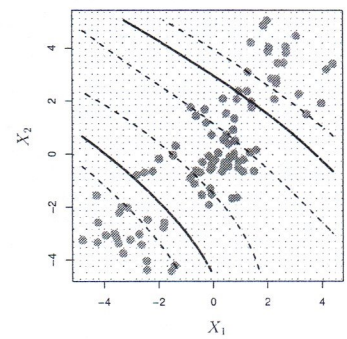
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Cubic Polynomials

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space



$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2 = 0$$

Nonlinearities and Kernels

- Polynomials (especially high-dimensional ones) get wild rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of *kernels*.
- Before we discuss these, we must understand the role of *inner products* in support-vector classifiers.

Inner products and support vectors

$$\langle x_i, x_{i'}\rangle = \sum_{j=1}^{p} x_{ij}x_{i'j} \quad \text{— } \textit{inner product between vectors}$$

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i\rangle \quad \text{— } \textit{n parameters}$$

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'}\rangle$ between all pairs of training observations.

It turns out that most of the $\hat{\alpha}_i$ can be zero:

$$f(x) = \beta_0 + \sum_{i\in\mathcal{S}} \hat{\alpha}_i \langle x, x_i\rangle$$

$\mathcal{S}$ is the *support set* of indices i such that $\hat{\alpha}_i > 0$. [see slide 8]

Kernels and Support Vector Machines

- If we can compute inner-products between observations, we can fit a SV classifier. Can be quite abstract!
- Some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij}x_{i'j}\right)^d$$

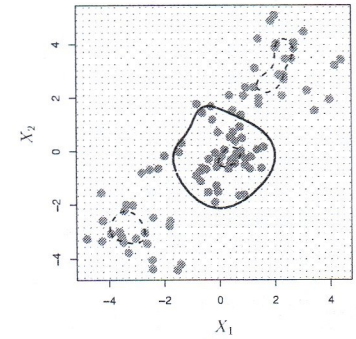
computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!
Try it for $p = 2$ and $d = 2$.

- The solution has the form

$$f(x) = \beta_0 + \sum_{i\in\mathcal{S}} \hat{\alpha}_i K(x, x_i).$$

Radial Kernel

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

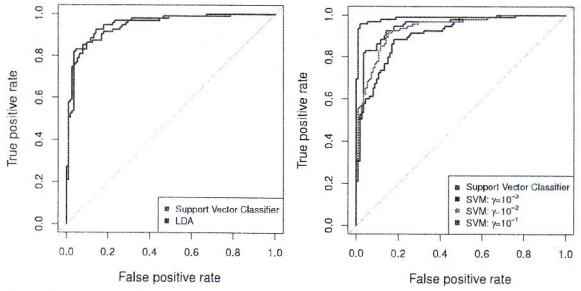


$$f(x) = \beta_0 + \sum_{i\in\mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

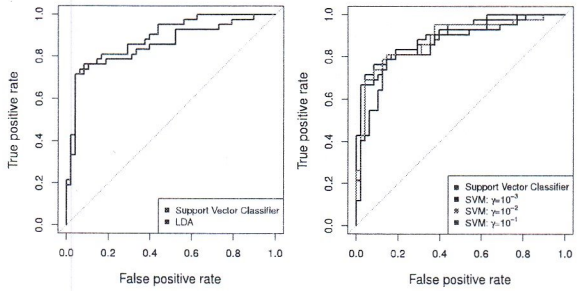
Controls variance by squashing down most dimensions severely

Example: Heart Data



ROC curve is obtained by changing the threshold 0 to threshold t in $\hat{f}(X) > t$, and recording *false positive* and *true positive* rates as t varies. Here we see ROC curves on training data.

Example continued: Heart Test Data



SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

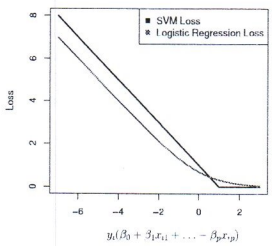
- **OVA** One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.
- **OVO** One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.

Support Vector versus Logistic Regression?

With $f(X) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$ can rephrase support-vector classifier optimization as

$$\underset{\beta_0, \beta_1, \ldots, \beta_p}{\text{minimize}} \left\{ \sum_{i=1}^{n} \max\left[0, 1 - y_i f(x_i)\right] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$



This has the form *loss plus penalty.*
The loss is known as the *hinge loss.*
Very similar to "loss" in logistic regression (negative log-likelihood).

Which to use: SVM or Logistic Regression

- When classes are (nearly) separable, SVM does better than LR. So does LDA.
- When not, LR (with ridge penalty) and SVM very similar.
- If you wish to estimate probabilities, LR is the choice.
- For nonlinear boundaries, kernel SVMs are popular. Can use kernels with LR and LDA as well, but computations are more expensive.

UNSUPERVISED CLASSIFICATION (CLUSTER ANALYSIS)

30/04

Training set:

$$\mathbb{X} = \begin{bmatrix} \underline{x}^T \\ \underline{x}_1^T \\ \underline{x}_2^T \\ \vdots \\ \underline{x}_n^T \end{bmatrix} \quad \begin{matrix} L \\ \ell_1 \\ \ell_2 \\ \vdots \\ \ell_n \end{matrix}$$

L is hidden

$\underline{x}_i \in \mathbb{R}^p$

We believe that these units belong to different groups but the labels are hidden.

Goal: "estimate" the labels $\hat{\ell}_1, \dots, \hat{\ell}_n$ and also the number of groups (g is also unknown) $\hat{g}$.

There are two approaches:

- parametric, often based on ML → EM algorithm (expectation and minimisation (EM)) — it requires knowledge of the problem (we'll skip this)
- non parametric: based on dissimilarities

Basic idea: (for both cases)
units belonging to the same group (/ cluster) are more similar than units belonging to different groups.

How to capture dissimilarities? (= what does it mean "similar"?)

$d : \mathbb{R}^p \times \mathbb{R}^p \longrightarrow [0, \infty)$ — **dissimilarity function** s.t.
(features space × features sp.)

→ tries to capture the dissimilarity between two units as expressed by vectors of features

Properties:

1. $\forall \underline{x} \in \mathbb{R}^p : d(\underline{x}, \underline{x}) = 0$: the dissimilarity of an element with itself is zero *

1! (stronger) $\forall \underline{x}, \underline{y} \in \mathbb{R}^p : d(\underline{x}, \underline{y}) = 0 \iff \underline{x} = \underline{y}$ **

Example: (counterexample for $\Leftarrow$)
$f, g \in L^2$:
$d(f, g) = \int |f - g|^2 \, d\underline{x}$
$d(f, g) = 0 \not\Longrightarrow f = g$

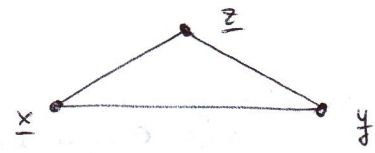
* note that we check the dissimilarities based on the features: two humans have 0 dissimilarity if the feature is 1 and is "breathing"
meanwhile
** means that the two humans are the same human!

2. $\forall \underline{x}, \underline{y} \in \mathbb{R}^p : d(\underline{x}, \underline{y}) = d(\underline{y}, \underline{x})$

3. $\forall \underline{x}, \underline{y}, \underline{z} \in \mathbb{R}^p : d(\underline{x}, \underline{y}) \leq d(\underline{x}, \underline{z}) + d(\underline{z}, \underline{y})$

"**Triangle inequality**":

(obv. in euclidean geometry, not so obvious in other cases)



If d satisfies 1!, 2., 3. : **metric / distance**
If d satisfies 1., 2., 3. : **pseudo-metric**

4. $\forall \underline{x}, \underline{y} \in \mathbb{R}^p : \; d(\underline{x}, \underline{y}) \leq \max\{ d(\underline{x}, \underline{z}), d(\underline{y}, \underline{z}) \}$

(stronger than 3.)

(4. $\Rightarrow$ 3.)

If d satisfies 1.', 2., 4. : **ultra-metric**

* it's all about finding the right meter (which is the standard deviation) with which we measure

Examples: (for quantitative variables)

1. $d(\underline{x}, \underline{y}) = \sqrt{(\underline{x}-\underline{y})^T(\underline{x}-\underline{y})} = \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}$:= **Euclidean metric**

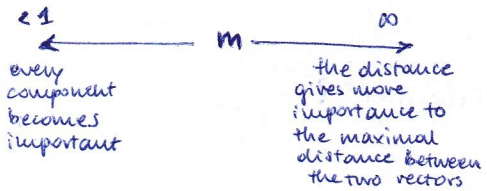
Often applied to standardized vectors (in cluster analysis) : we standardize w.r.t. standard deviation and then we apply the Euclidean distance *

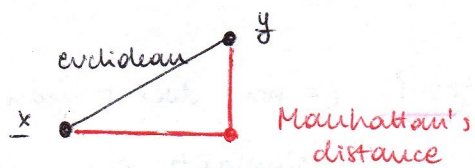
2. $d_{\Sigma^{-1}}(\underline{x}, \underline{y}) = \sqrt{(\underline{x}-\underline{y})^T \Sigma^{-1} (\underline{x}-\underline{y})}$:= **Mahalanobis distance**

Problem which occurs also with standardization: assumes same covariance structure in each cluster ↔ ! it's a strong assumption. It's better to standardize and apply the Euclidean distance (which is the same as Mahalanobis but we take only the diagonal of the covariance being the same in every group)

3. $d(\underline{x}, \underline{y}) = \left(\sum_{i=1}^{p} |x_i - y_i|^m \right)^{1/m}$:= **ℓ^m distances** (Minkowski's distances)

- $m = 2 \Rightarrow$ Euclidean distance
- $m = 1 \Rightarrow$ Manhattan distance





note that: when we try to find the point that minimizes the euclidean distances between all the points we find the baricenter, aka the MEAN, if we minimize the Manhattan distance we find the MEDIAN.

4. $\underline{x}, \underline{y} \in (\mathbb{R}^p)^+$ (non-negative components)

$$d(\underline{x}, \underline{y}) = \sum_{i=1}^{p} \frac{|x_i - y_i|}{x_i + y_i}$$:= **Canberra distance**

if we want to see the distance between my income in January and in February (two similar incomes) we can simply use ℓ^1 distance, but if Berlusconi does the same the difference in his salary may not be meaningful as mine: if Berlusconi loses 10 € it's not like me losing 10 € ⇒ Canberra's distance makes the things relative

Dissimilarities for categorical variables

$\underline{x} = \{0,1\}^p \longrightarrow \underline{x} = [\,0\;1\;1\;0\;0\;0\;1\;1\,]^T$

Obs. if $x \in \{$blue, red, brown$\}$ we can represent x in 0,1 - repr. :

$x \longrightarrow (x_1, x_2)$:

$$x_1 = \begin{cases} 1 & \text{if } x = \text{blue} \\ 0 & \text{if } x \neq \text{blue} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } x = \text{red} \\ 0 & \text{if } x \neq \text{red} \end{cases}$$

(1,0) blue
(0,1) red
(0,0) brown
(1,1) impossible

We can always code a categorical variable through **0** and **1**.

If $\underline{x}, \underline{y} \in \{0,1\}^p$ and we apply the Euclidean distance:

$$d_{Euclidean}(\underline{x}, \underline{y}) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2} = \sqrt{\#\text{ discordances}}$$

since $x_i - y_i = \begin{cases} 0 & x_i = y_i = \begin{cases} 1 \\ 0 \end{cases} \\ 1 & \text{they're discordant} \end{cases}$

$$d^2_{Euclidean}(\underline{x}, \underline{y}) = \#\text{ discordances}$$

Idea for extension:

$\underline{x}$ \ $\underline{y}$	1	0
1	a	b
0	c	d

(p)

$a = \#(x_i = 1 \wedge y_i = 1)$

number of times that the i-th component of $\underline{x}$ and the i-th component of $\underline{y}$ are 1

$\underline{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ $\underline{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $\Rightarrow$ $a = 1$, $b = 1$, $c = 0$, $d = 1$

$$d^2_{Euclidean}(\underline{x}, \underline{y}) = c + b = \#\text{ discordances}$$

Other possibilities:

$$d(\underline{x}, \underline{y}) = \frac{c+b}{p} \quad \text{(discordances rate)}$$

$$d(\underline{x}, \underline{y}) = 1 - \frac{a}{p}$$

⋮

It's not really a research for distances, it's looking for dissimilarities

(Reference: Johnson-Wichern p. 674 (cluster analysis))

Mixed cases: $\underline{x}^T = [\underline{x}^{Q\,T}, \underline{x}^{C\,T}]$

$\underline{y} = [\underline{y}^{Q\,T}, \underline{y}^{C\,T}]$

Q = quantitative
C = categorical

$$d(\underline{x}, \underline{y}) = \lambda\, d^2_Q(\underline{x}^q, \underline{y}^q) + (1-\lambda)\, d^2_C(\underline{x}^c, \underline{y}^c) \qquad \lambda \in (0,1)$$

Obs. $\mathbb{X} = \begin{bmatrix} \underline{x}_1^T \\ \vdots \\ \underline{x}_n^T \end{bmatrix}$ $\longrightarrow$ cluster the units

$\mathbb{X} = [y_1, \dots, y_p]$ $\longrightarrow$ cluster the variables *

* to see if there are variables that are more or less related together and they talk about the same thing
$\Rightarrow$ one obvious measure to measure the similarities between variables is the correlation
$\Rightarrow$ dissimilarities := 1 − correlation

$$d^2(x_i, x_j) = d^2(\text{variable}_i, \text{variable}_j) = 2(1 - \text{Corr}(X_i, X_j))$$

Dissimilarity matrix (this will be the input of the clustering algorithm)

$$\mathbb{X} = \begin{bmatrix} \underline{x}_1^T \\ \vdots \\ \underline{x}_n^T \end{bmatrix} \qquad \underline{x}_i \in \mathbb{R}^p$$

Compute $d_{ij} = d(\underline{x}_i, \underline{x}_j)$ $\quad i, j = 1, \dots, n$

(whatever is the d chosen)

$\longrightarrow D = [d_{ij}]$ $n \times n$

$= \begin{bmatrix} 0 & \cdots & d_{ij} \\ \vdots & \ddots & \vdots \\ d_{ji} & \cdots & 0 \end{bmatrix}$

this can be coded in a triangular matrix (that's how dissimilarity matrices are represented)

$d_{ij} = d_{ji}$ because of 2.

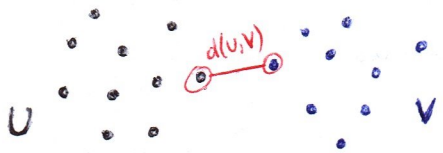
$d_{ii} = 0$ because of 1.

Dissimilarities between clusters (= distance between sets / clusters)

U, V are two finite sets of points in $\mathbb{R}^p$: $d(U,V) = ?$

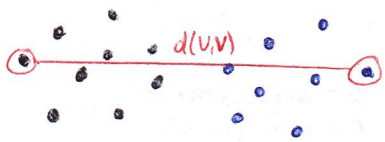
- **Single** linkage:

$$d(U,V) = \min \{ d(x,y) : x \in U, y \in V \} \qquad (d_{sl}(U,V))$$



- **Complete** linkage:

$$d(U,V) = \max \{ d(x,y) : x \in U, y \in V \} \qquad (d_{cl}(U,V))$$



- **Average** linkage:

$$d(U,V) = \frac{1}{\#U \cdot \#V} \sum_{\substack{x \in U \\ y \in V}} d(x,y) \qquad (d_{ave}(U,V))$$

we take all the possible distances between points belonging to the two different sets and then we do the average

---

Hierarchical Agglomerative Clustering Algorithm

← we now have all the ingredients for the clustering algorithm

Set d and the linkage.

- Initialization:
  each unit is a cluster
- Until convergence, repeat:
  1. merge the two clusters which are less dissimilar
  2. compute the new distance matrix

- **HIERARCHICAL**: it's an iterative algorithm and at each iteration we move to a hierarchy of clusters within our training set
- **AGGLOMERATIVE**: we move from one step to the other by merging together clusters (until at the end we'll have 1 cluster)

Example: $n = 5$ $\quad D =$

0					1
2	0				2
6	5	0			3
10	9	4	0		4
9	8	5	3	0	5
1	2	3	4	5	

distance between unit 2 and unit 1 is 2

notice that: we don't really need the units, it doesn't matter for clustering, we need the distances / dissimilarities between the units (most of the algorithms want a dissimilarity matrix instead of the data as the input)

We use single linkage:

- Iteration 1:

  merge 1 & 2 $\Longrightarrow$ $\{1,2\}$

  $d(1,2) = 2$

  D =

0				$\{1,2\}$
5	0			3
9	4	0		4
8	5	3	0	5
$\{1,2\}$	3	4	5	

What is the difference, for instance $d(\{1,2\},3) = ?$
We're using single linkage:
$d(1,3) = 6$, $d(2,3) = 5$ $\Rightarrow$ $d(\{1,2\},3) = \min\{\cdot,\cdot\} = 5$

- Iteration 2:

  merge 5 & 4 $\Longrightarrow$ $\{4,5\}$

  $d(4,5) = 3$

  D =

0			$\{1,2\}$
5	0		3
8	4	0	$\{4,5\}$
$\{1,2\}$	3	$\{4,5\}$	

- Iteration 3:

  merge 3 & $\{4,5\}$ $\Longrightarrow$ $\{3,4,5\}$

  $d(3,\{4,5\}) = 4$

  D =

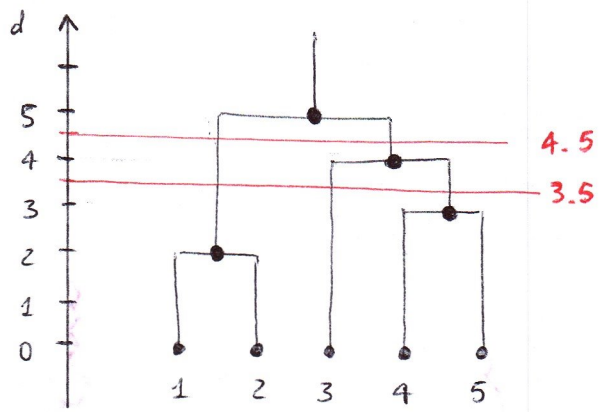
0		$\{1,2\}$
5	0	$\{3,4,5\}$
$\{1,2\}$	$\{3,4,5\}$	

- Iteration 4:

  merge $\{1,2\}$ & $\{3,4,5\}$ $\Longrightarrow$ $\{1,2,3,4,5\}$

  $d(\{1,2\},\{3,4,5\}) = 5$

Graphical representation: **DENDROGRAM**



So we have a visual representation of the clustering structure of the dataset. We have to decide at what distance we want to look at the dataset. If we look at the data by distance 3.5 then we have 3 clusters. If we look by distance 4.5 then we have 2 clusters.

Note that: if we had inverted the order of the units we would have to cross the lines making the representation. This representation is very useful if we can write down the units in an order s.t. we don't have to cross horizontally the lines that continue up. The computer explores all the possible permutations of the units in order to find the right representation.

There is an other representation of this Dendrogram.
We have to compute a new distance matrix based on the dendrogram:

$C =$

0					1
2	0				2
5	5	0			3
5	5	4	0		4
5	5	4	3	0	5
1	2	3	4	5	

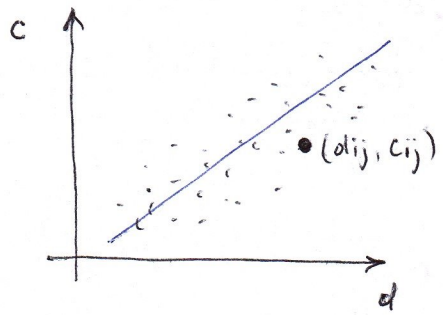
every cell is the level at wich every unit has been merged in the same cluster (the distance at wich the two units has been merged)

COPHENETIC DISTANCE
(ultra-metric)
(we're transforming the original **d** into an ultrametric by using the dendrogram)

If we take $(\underline{x}_i, \underline{x}_j)$ we can associate two distances:

$$(\underline{x}_i, \underline{x}_j) \longrightarrow (d_{ij}, c_{ij}) \qquad i,j = 1, \dots, n$$

so:



We can compute the correlation: —

$$\left| \text{Corr}\{(d_{ij}, c_{ij}) : i,j = 1, \dots, n\} \right| = \left| \text{corr}(C, D) \right| \downarrow = |CPCC|$$

COPHENETIC CORRELATION COEFFICIENT

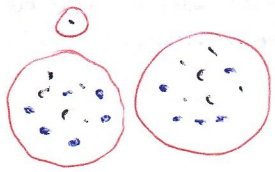
(the closer to **1** → the better the clustering structure as represented by the dendrogram is representing the true structure that is in **D**)

Obs.

- try all the linkages if we can
- JITTER the data (= add some noise and see what happens)
- single linkage often produces a "chain effect"

We create a cluster that is like a chain which goes across the two real clusters

- complete linkage / average linkage generate clusters that are elliptical:



WARD'S METHOD FOR HIERARCHICAL CLUSTERING

(Instead of defining a linkage for joining up the clusters we follow a different criterium)

Let $\mathcal{T} = \{\underline{x}_i : i = 1, \dots, n\} \subseteq \mathbb{R}^p$

$\left(\text{e.g. } X = \begin{bmatrix} \underline{x}_1^T \\ \vdots \\ \underline{x}_n^T \end{bmatrix} \quad \underline{x}_i \in \mathbb{R}^p \right)$

Suppose we split data into k groups
Let $C_1, \dots, C_k$ be clusters:

$$ESS_j = \sum_{\underline{x}_i \in C_j} (\underline{x}_i - \bar{\underline{x}}_j)^T (\underline{x}_i - \bar{\underline{x}}_j) = \sum_{\underline{x}_i \in C_j} \|\underline{x}_i - \bar{\underline{x}}_j\|^2$$

; basically in each cluster we compute the variability within the cluster (= covariance matrix / total variability)

$$\bar{\underline{x}}_j = \frac{1}{\#C_j} \sum_{\underline{x}_i \in C_j} \underline{x}_i$$ = baricenter of the cluster

and

$$ESS = ESS_1 + \dots + ESS_k$$

Ward's method proceeds iteratively bottom-up. (agglomerative method)

We start with all the units belonging to a separate cluster. Every unit is a cluster $\Rightarrow$ $ESS = 0$ ($k = n$, $ESS_j = 0 \ \forall j$). Then we merge together the two units for which the increse in ESS will be minimum. And we procede like that at each iteration.

At each iteration merge the two clusters which generates the minimum increase in ESS. It tends to create ellipsoidal clusters but it doesn't have problems with the chain effect.

NON-HIERARCHICLE METHOD FOR CLUSTERING: K-MEANS

(k-medoids, ..)
(variation of K-means)

$\mathcal{T} = \{\underline{x}_i : i = 1, \dots, n\}$ training set, $\mathcal{T} \subseteq \mathbb{R}^p$

$d : \mathbb{R}^p \times \mathbb{R}^p \longrightarrow [0, \infty)$ dissimilarity
(most often d will be a metric) (not necessary)

Given $k \geq 1$, finding K clusters mean identifing k subsets of the training set: $C_1, \dots, C_k \subseteq \mathcal{T}$ such that:

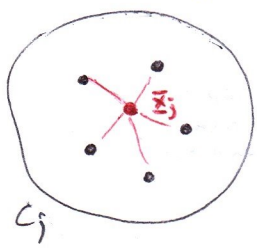
- $C_i \cap C_j = \phi \quad \forall i \neq j$
- $C_1 \cup \dots \cup C_k = \mathcal{T}$

In other words: clustering means to find a partition of the training set in K subsets
(**Note**: check for **FUZZY** CLUSTERING, where it's not like that (sovrapposition and points left out are allowed))

Def. **Centroids**: given $C_j \subseteq \mathcal{T}$ (given a cluster) let

$$\bar{\underline{x}}_j = \underset{\underline{x} \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{\underline{x}_i \in C_j} d^2(\underline{x}_i, \underline{x})$$

= point in $\mathbb{R}^p$ which minimizes (Σ) all the distances squared from the point in the cluster



Obs. if d is the Euclidean distance then the centroid is the baricenter

$$\left(\bar{\underline{x}}_j = \frac{1}{\#C_j} \sum_{\underline{x}_i \in C_j} \underline{x}_i \right)$$

Optimal clustering:

find $C_1, \dots, C_k$ s.t. $\sum_{j=1}^{k} \left[\sum_{x_i \in C_j} d^2(x_i - \bar{x}_j) \right]$ is minimum.

We want to find the clusters so that if we sum the total variability within that cluster and we sum across the clusters, that quantity is minimal.

K-means algorithm for solving the optimization problem:

- Initialization step:
  - assign at random the units of Z among k subsets $C_1, \dots, C_k$ (then go to step 1.)
  - assign at random k centroids $\bar{x}_1, \dots, \bar{x}_k$ (then go to step 2.)

  it depends on a random assignment and the final result strongly depend on it

- Iterate until convergence:
  1. For $j = 1, \dots, k$ compute the centroids of $C_j \Longrightarrow \bar{x}_1, \dots, \bar{x}_k$
  2. For $x_i \in Z$: assign x_i to cluster C_j if :
     $$d^2(x_i, \bar{x}_j) = \min \{ d^2(x_i, \bar{x}_1), \dots, d^2(x_i, \bar{x}_k) \}$$

- End: if centroids at step 1 are the same as those computed in the previous iteration.

The difficulty is to prove convergence.
For d = Euclidean the convergence has been proved (Hartigan & McQueen ('70))

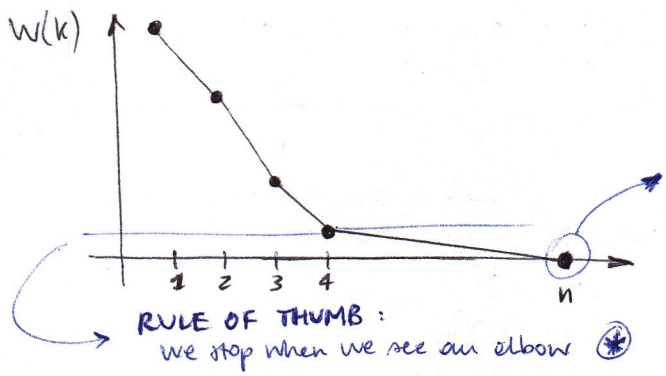
Obs.

1. Strong dependence on the initialization ($\Rightarrow$ try many initializations)
2. The difficult step for a general d is step 1.
   A possible (and easy) way out is:
   $$\bar{x}_j = \arg\min_{x \in Z} \sum_{x_i \in C_j} d^2(x_i, x) \qquad j = 1, \dots, k$$
   $\Longrightarrow$ $\bar{x}_j$ is called **medoid** (k-medoid)
   (k-medoid: name of the algorithm (instead of k-means))

   we minimize w.r.t. the training set, not w.r.t. the entire space $\mathbb{R}^p$.

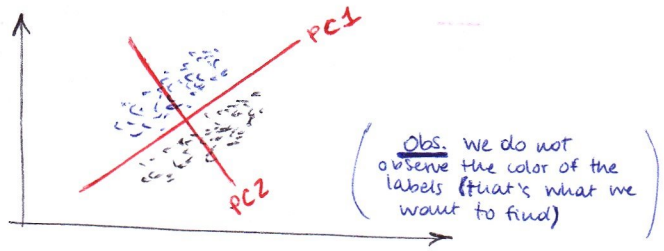
3. To choose k:
   $$W(k) = \sum_{j=1}^{k} \left[\sum_{x_i \in C_j} d^2(x_i, \bar{x}_j) \right]$$

(*) increasing k by one unit marginally does not payoff in terms of decrease of $W(k)$



if we take $k = n$ then $W(k) = 0$ (if every point is a cluster the distance from one point to itself is always 0 $\Rightarrow W(n) = 0$)

4. PCA and clustering



(Obs. We do not observe the color of the labels (that's what we want to find))

In this case, to discriminate the two clusters it would be optimal to take the second principal component and not the first.

Usually the clustering structure is not identified by the first principal component but by the last one.

GRAPHICAL REPRESENTATIONS

≡ reducing dimensionality in such a way that in the reduced space the clustering will be more obvious to see

- use Fisher's scores after LDA
- **Multidimensional Scaling (MDS)** (it's also a tool for clustering)

$\mathbb{R}^p, d, \mathcal{I} = \{x_i : i = 1, \dots, n\} \subseteq \mathbb{R}^p$

$\mathbb{R}^q, d_{Euclidean}, q < p$

(we want to change the representation space (we want also to reduce the dimension of the problem: $q < p$))

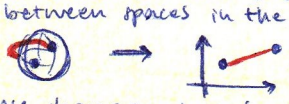
$\mathcal{I} = \{x_i : i = 1, \dots, n\} \longrightarrow \mathcal{I}' = \{y_i : i = 1, \dots, n\}$

with $y_1, \dots, y_n$ s.t.

$$d(x_i, x_j) \approx d_{Euclidean}(y_i, y_j)$$

↑ needs to be specified

A good example: suppose we want to describe with the Euclidean distance the distance between spaces in the world:



We change points ($x_i \to y_i$) and the distance ($\cap \to —$)

$D = [d_{ij}] \longrightarrow \Delta = [\delta_{ij}]$ s.t. $d_{ij} \approx \delta_{ij}$

(original) distance matrix — Euclidean dist.

Obs. the solution is not unique:
the Euclidean distances are invariant w.r.t. rigid transformations.

Obs. If $d = d_{Euclidean}$ (the initial d)
$\Longrightarrow$ the solution is given by the space spanned by the first q principal components

(That's because we're trying to find a linear subspace $\mathbb{R}^q$ which is closest to the original training set. (But in that case we use **PCA** instead of **MDS**, it's more accurate))

Objective function for approximating D with Δ:

- classical MDS:

$$\sum_{i \neq j} (d_{ij} - \delta_{ij})^2$$ = they're close in quadratic sense

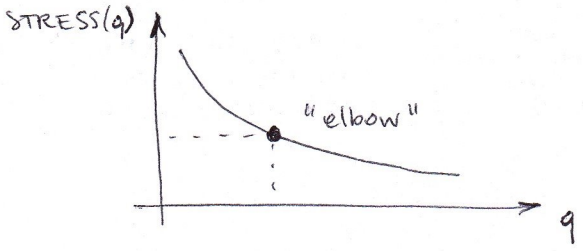
- More recently Kruskaal & Co.:

$$STRESS = \frac{\sum_{i \neq j} (\theta(d_{ij}) - \delta_{ij})^2}{\sum_{i \neq j} \delta_{ij}^2}$$

$\theta : \mathbb{R} \to \mathbb{R}$ monotone ↑

Minimize STRESS both w.r.t. $\hat{z}' = \{ y_i : i = 1, \ldots, n \}$ and θ.

Choose q:



Note: we can use multidimensional scaling BEFORE clustering (just considering it as a transformation to the data)