Introduction à l'Extraction de Connaissances

Chapitre IV

Part 4

Classification par les méthodes linéaire, logistique, RNs et Perceptron, à base de noyau SVM

IBL

Alexandre Saidi
Ecole Centrale de Lyon
Département Mathématiques-Informatique
UMR LIRIS - CNRS

Novembre 2017

Introduction

Ce qu'on a étudie :

• Les méthodes produisant des arbres de décision ou des règles (ID3, C4.5, PRISM, A Priori, ...) sont bien adaptées aux attributs nominaux.

• Pour les attributs numériques :

L'extension aux numériques de ces méthodes est possible :

➛ Par discrétisation, l'ajout de tests numériques dans l'arbre/règles, etc.

• Des méthodes diverses (cf. CART numérique avec choix selon min variance) construisent des Arbres de Régression.

➛ Voir aussi REPTree (cf. BE2) : *Regression Tree with Reduce-Error Pruning*

Ce chapitre :

• Lorsque tous les attributs sont numériques, on peut utiliser les méthodes de régression : ➛ *Régression* et ses variantes

Modèles Linéaires : Régression Linéaire

La Régression linéaire :

- La classe = une combinaison linéaire d'attributs avec des pondérations

$$\boxed{y = w_0 + w_1\, a_1 + w_2\, a_2 + \ldots + w_k\, a_k}$$

avec y = classe, a_i les valeurs d'attributs et w_i les pondérations.

- Les pondérations (w_i) sont calculées sur l'ensemble d'apprentissage .
- Formulation pour la première instance dont le vecteur d'attributs est $a^{(1)}$:

$$w_0\, a_0^{(1)} + w_1\, a_1^{(1)} + w_2\, a_2^{(1)} + \ldots + w_k\, a_k^{(1)} = \sum_{j=0}^{k} w_j\, a_j^{(1)}$$

➙ $\mathbf{a_0^{(i)}}$: attribut (*extra*) dont la **valeur est toujours = 1**
(à ne pas confondre avec w_0).

➙ $\sum_{j=0}^{k} w_j\, a_j^{(1)}$ = la **prédiction** de la valeur de la classe de la 1^e instance

Modèles Linéaires : Régression Linéaire (suite)

La **Rég. lin.** apprend les (k+1) pondérations w_j en **minimisant** la somme des carrés des différences entre les classes connue ($y^{(i)}$) et prédite ($\sum w_j\, a_j^{(i)}$).

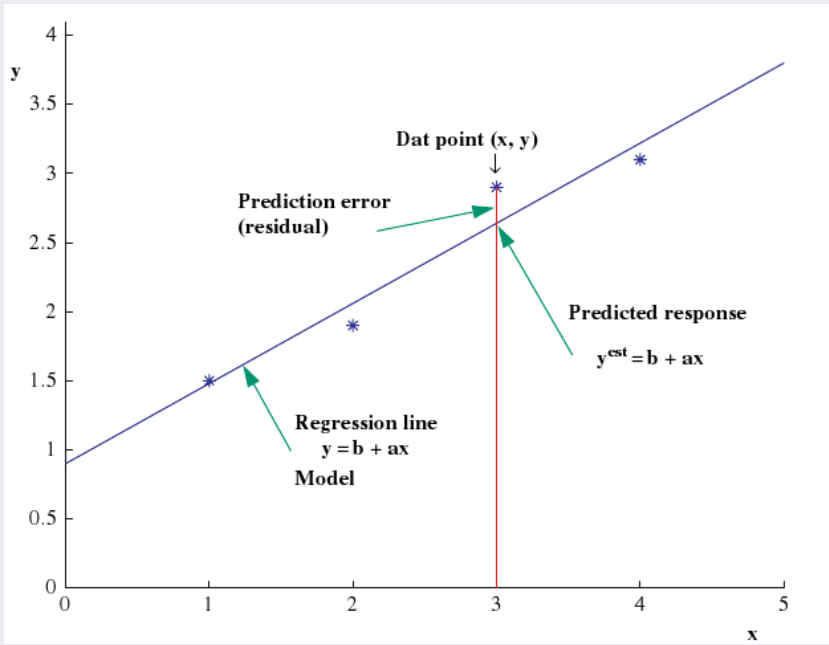
→ Avec n instances d'apprentissage, la somme des carrés des différences :

$$\sum_{i=1}^{n} \left(y^{(i)} - \sum_{j=0}^{k} w_j\, a_j^{(i)} \right)^2$$

$y^{(i)}$ = la classe connue,

$\sum w_j\, a_j^{(i)}$ = prédiction

→ La valeur entre parenthèses = l'erreur pour la classe de la ième instance.



Modèles Linéaires : Régression Linéaire (suite)

Rappel (erreur) : $$\sum_{i=1}^{n} \left(y^{(i)} - \sum_{j=0}^{k} w_j \, a_j^{(i)} \right)^2$$

- Cette **somme** doit être minimisée en choisissant les *bons* coefficients w_j.
- L'erreur minimale est un indice de performance du modèle.
- Les coefficients sont calculés par des opérations classiques sur les matrices (voir plus loin),
- Davantage de *justesse* s'il y a plus d'instances que d'attributs
- La minimisation de l'erreur absolue est plus difficile (v. chap svt)
- La régression linéaire (ci-dessus) est une base pour la prédiction linéaire.

Un exemple :

la prédiction des performances des ordinateurs avec calcul / prédiction d'une sortie numérique (la sortie n'est pas une classe). ../..

Modèles Linéaires : Régression Linéaire (suite)

- Rappel : prédiction des performances relatives des PC selon certains attributs.

	Main Memory (Kb)				Channels		
	Cycle Time (ns)	*Min*	*Max*	Cache (KB)	*Min*	*Max*	Performance
	MYCT	*MMIN*	*MMAX*	*CACH*	*CHMIN*	*CHMAX*	*PRP*
1	125	256	6000	256	16	128	198
2	29	8000	32,000	32	8	32	269
3	29	8000	32,000	32	8	32	220
4	29	8000	32,000	32	8	32	172
5	29	8000	16,000	32	8	16	132
...							
207	125	2000	8000	0	2	14	52
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

TABLE 1: Données Performances CPU

$$PRP = -55.9 + 0.0489MYCT + 0.0153MMIN + 0.0056MMAX + 0.6410CACH - 0.2700CHMIN + 1.480CHMAX$$

- BD : 209 configurations différentes (une ligne = une configuration)
  ➙ Tous les attributs (la cible comprise) sont **numériques**.

Régression : Ex. de calcul de l'erreur

Cas général de la régression sur un ensemble de points $(x, y), x \in \mathbb{R}^k, y \in \mathbb{R}$.

Exemple trivial : minimisation de l'erreur (et mesure des performances) pour un cas trivial de n instances avec $k = 1$:

➛ On trouve les paramètre w_0 et w_1 tels que $< y^i, x^{(i)} >$ modélise l'espace.

- Pour les différentes instances (x^i, y^i), on a (avec y^i connue) :

$$\begin{aligned} y^1 &= (w_1 x^1 + w_0) + e^1, \\ y^2 &= (w_1 x^2 + w_0) + e^2, \\ &..., \\ y^n &= (w_1 x^n + w_0) + e^n \end{aligned}$$

tel que $x^{(1)} - y^{(1)} = e^{(1)}$

avec e^i= l'erreur de l'instance x^i.

L'erreur à minimiser $Err_{w_0,w_1} = \sum_{i=1}^{n} [y^i - (w_1 x^i + w_0)]^2$

➛ valeurs optimales : $\hat{w_0}, \hat{w_1} = \underset{\hat{w_0},\hat{w_1}}{\arg\min}\ Err_{w_0,w_1}$

Régression : Ex. de calcul de l'erreur (suite)

- Un calcul simple pour la minimisation donne (par la dérivée de Err_{w_0,w_1}) :

$$\frac{\partial Err_{w_0,w_1}}{\partial w_1} = 0 \Rightarrow \frac{\partial Err_{w_0,w_1}}{\partial w_1} = \sum_{i=1}^{n} -2[y^i - (w_1 x^i + w_0)](x^i) = 0$$

$$\frac{\partial Err_{w_0,w_1}}{\partial w_0} = 0 \Rightarrow \frac{\partial Err_{w_0,w_1}}{\partial w_0} = \sum_{i=1}^{n} -2[y^i - (w_1 x^i + w_0)] = 0$$

Courbe de performances de la régression linéaire (pour l'exemple) :

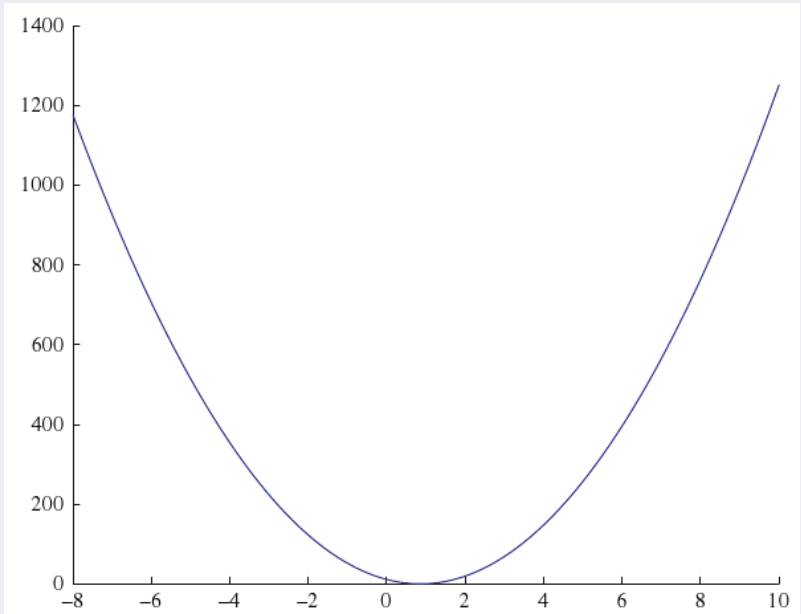
Exemple hypothétique :

$$Err_{w_0,w_1} = \sum_{i=1}^{n} [y^i - (w_1 x^i + w_0)]^2$$

La courbe en face :

- axe des X : performances de w_1
- axe des Y : Err_{w_0,w_1}

➛ Permet de repérer $\hat{w}_1 = 1$



Régression : Ex. de calcul de l'erreur (suite)

Après calculs, on obtient :

$$\hat{w_1} = \frac{n.\sum_1^n x^i y^i - \sum_1^n x^i . \sum_1^n y^i}{n.\sum_1^n (x^i)^2 - \left(\sum_1^n x^i\right)^2}$$

$$\hat{w_0} = \frac{\sum_1^n (x^i)^2 . \sum_1^n y^i - \sum_1^n x^i . \sum_1^n x^i y^i}{n.\sum_1^n (x^i)^2 - \left(\sum_1^n x^i\right)^2}$$

- Rappel : les points sont représentés par les couples (x^i, y^i) avec l'estimation de (la vrai) y^i par $x^i = w_1 a^i + w_0$
  ➛ y^i = la classe connue de la i^{eme} instance

- Ces calculs vont servir à un cas numérique ... ↝

Régression : Ex. de calcul de l'erreur (suite)

Un exemple concret :

- Soit les 4 points $(x^i, y^i), i = 1..4$:

$$(1, 1.5), (2.0, 1.9), (3.0, 2.7), (4.0, 3.1)$$

Le dénominateur des calculs : $d = 4\sum_1^4 (x^i)^2 - \left(\sum_1^4 x^i\right)^2 = 4 \times 30 - 10^2 = 20$

$$\hat{w_1} = \tfrac{1}{d}\left(4.\sum_1^4 x^i y^i - \sum_1^4 x^i \sum_1^4 y^i\right) = \frac{1}{20}(4 \times 25,8 - 10 \times 9,2) = 0,56$$

$$\hat{w_0} = \tfrac{1}{d}\left(\sum_1^4 (x^i)^2 \sum_1^4 y^i - \sum_1^4 x^i \sum_1^4 x^i y^i\right) = \frac{1}{20}(30 \times 9,2 - 10 \times 25,8) = 0,90$$

$Err^1 = y^1 - (\hat{w_0} + \hat{w_1} x^1) = 1,5 - (0,9 + 0,56 \times 1) = 0,04$
$Err^2 = -0,12 \quad Err^3 = 0,12 \quad Err^4 = -0,04$

La somme des carrées de ces erreurs donne $Err_{\hat{w_0},\hat{w_1}} = 0,0304$

➟ La droite (régression) : $y = 0.56x + 0.9$
➟ Dessiner la droite (triviale).

Remarques sur la régression linéaire

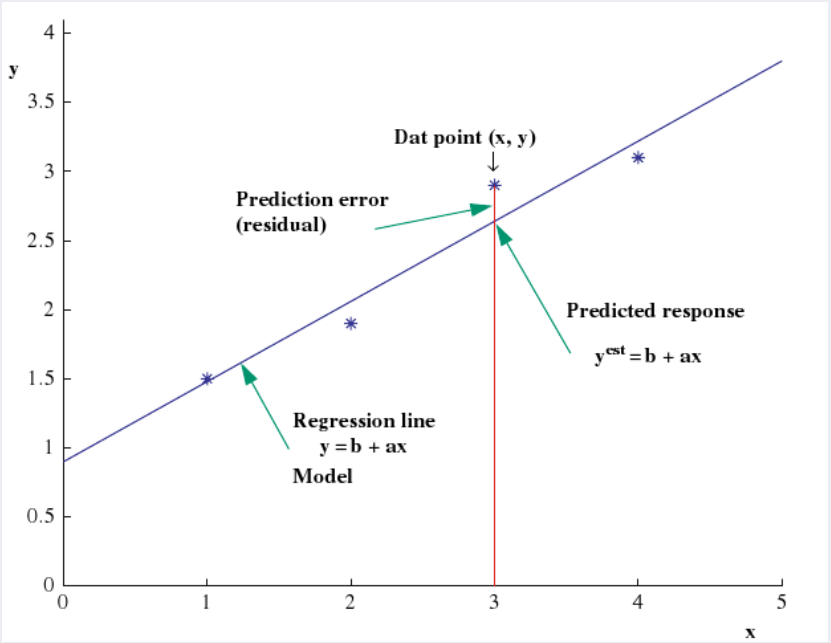
• La régression linéaire = un schéma simple et fiable pour la prédiction numérique :

→ Permet une première exploration des données (numériques).

• **Inconvénient** : la **linéarité**

➙ Trouver la *meilleure droite* au sens de la **moindre carré** des différences peut être mal adapté si les données n'ont pas une dépendance linéaire.

➙ Une droite (linéaire) ne suffit pas toujours à séparer les instances (par définition à n-dimensions).



• La *co-linéarité* des points est un autre problème (voir plus loin).

Remarques sur la régression linéaire (suite)

Interprétation par un Hyperplan (cas bi-classes, dimension=2+1)

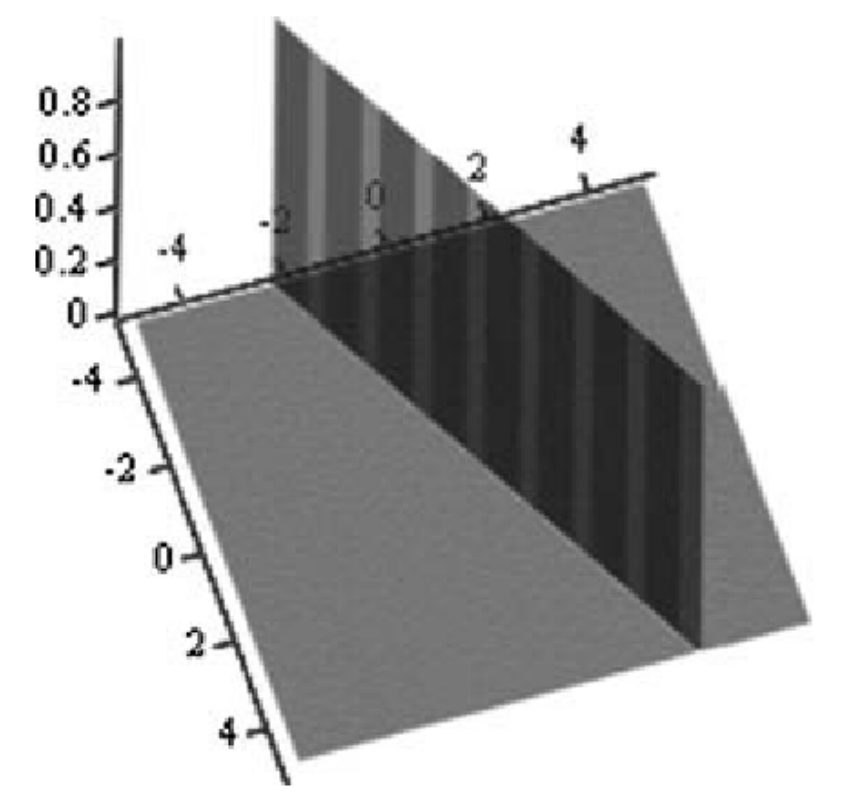


FIGURE 1: Ex. cas bi-classes : plan de séparation pour $\phi(x_1, x_2) = 0,7 + 1.3x_1 - 2.5x_2$

Divers cas et exemples

Exemple (sur un même ensemble d'apprentissage) :

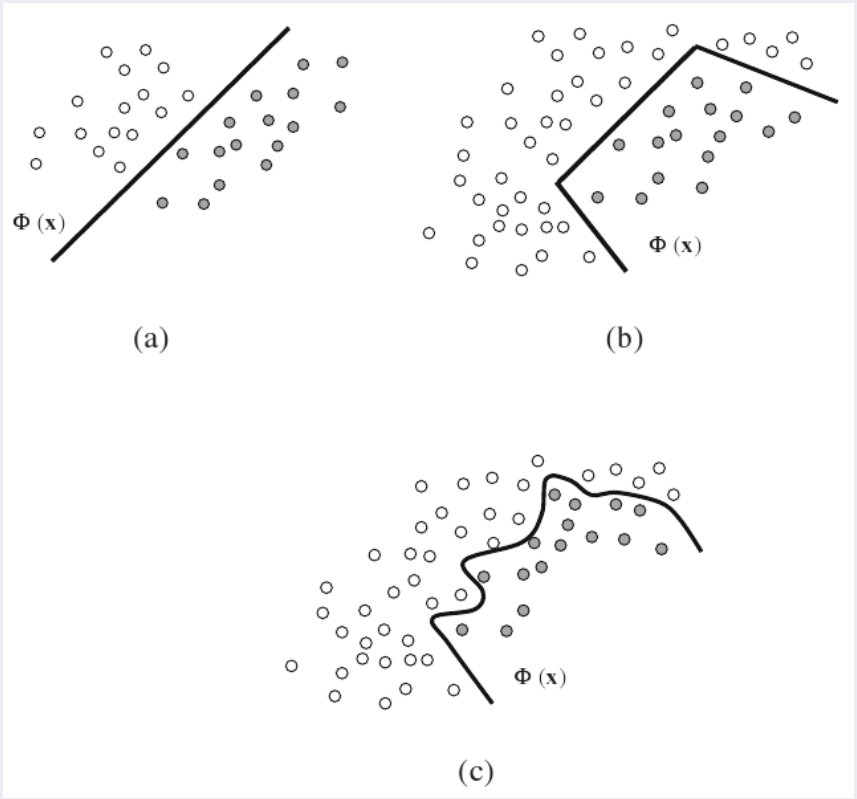


FIGURE 2: **Cas bi-classes**, (a) : classifieur linéaire (mal adapté), (b) : classifier deux à deux, (c) : classifieur non linéaire

Divers cas et exemples (suite)

Difficultés et limites des modèles de régression :

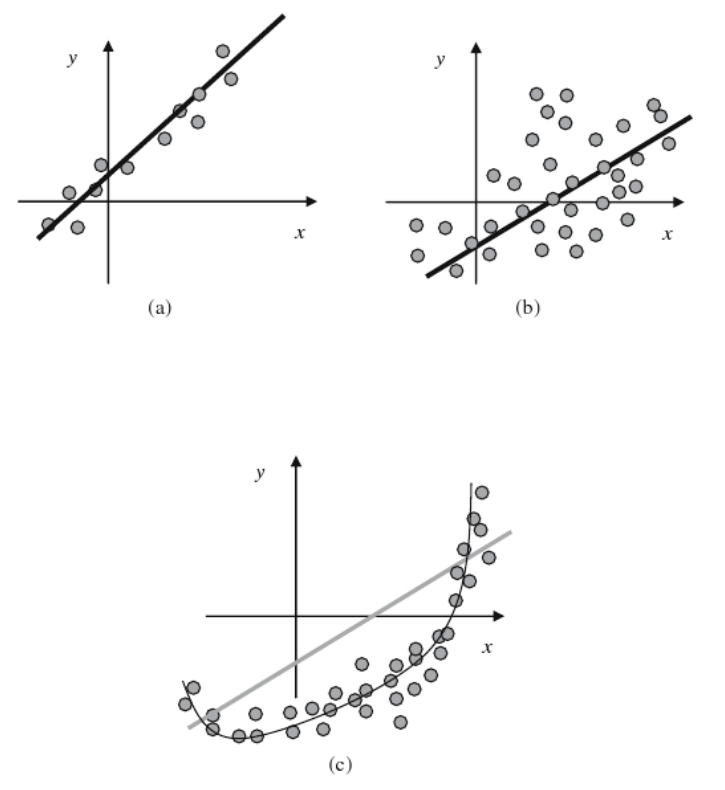


FIGURE 3: (a) : classifieur linéaire avec une bonne dispersion, (b) : linéaire avec une dispersion importante, (c) : non linéaire avec une dispersion importante

Divers cas et exemples (suite)

Aperçu de la classification non linéaire :

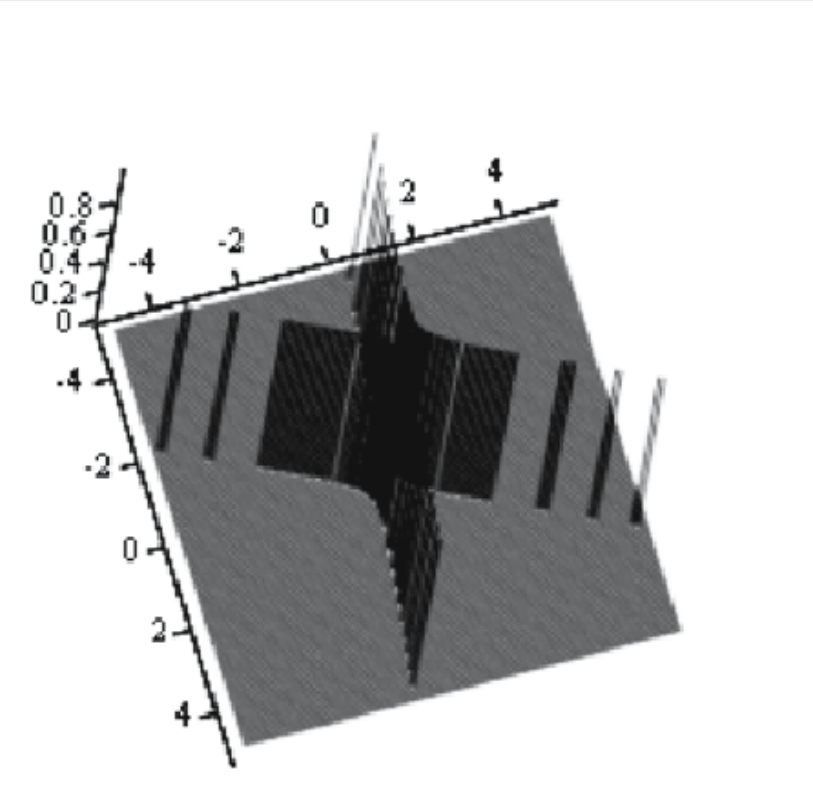


FIGURE 4: Hyperplan de séparation non linéaire produit par un classifieur quadratique pour
$\phi(x_1, x_2) = 0,1 + 0,6x_1^2 - 2x_2^2 + 4x_1x_2 + 1,5x_1 - 1,6x_2$

Divers cas et exemples (suite)

- Cas non linéaire : on se méfie des classifieurs à 0 erreur ! → sur-apprentissage.

  ➙ Détérioration des capacités de généralisation complexe :

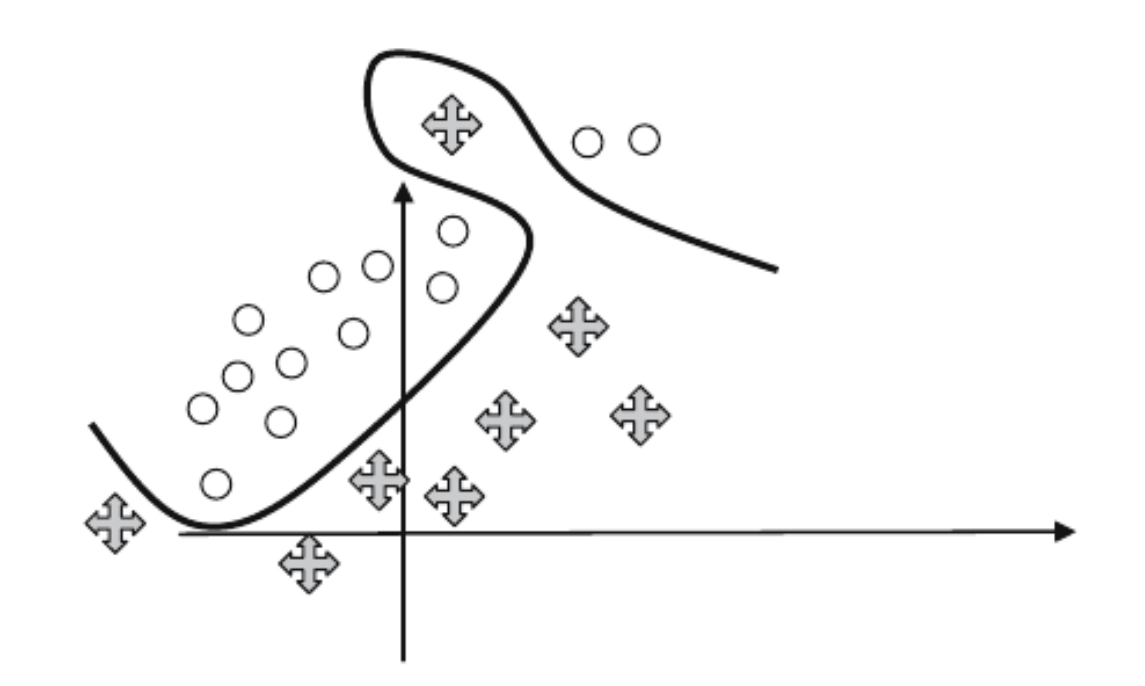


FIGURE 5: Un classifieur non linéaire à 0 erreur risque l'overfitting

Régression et la classif. linéaire multi-réponses

- Toute technique de régression peut être utilisée pour la classification

Cas spécifique bi-classes (C_1 et C_2) :

on peut définir un *hyperplan* pour 2 classes données par la procédure :

- Chaque classe reçoit un vecteur de poids calculé à partir de ses instances.
- Soit le vecteur pour la classe 1 (idem pour 2) :

$$w_0^{(1)} + w_1^{(1)} a_1 + w_2^{(1)} a_2 + ... + w_k^{(1)} a_k$$

- Une nouvelle instance est de la classe 1 plutôt que de la classe 2 si :

$$w_0^{(1)} + w_1^{(1)} a_1 + w_2^{(1)} a_2 + ... + w_k^{(1)} a_k > w_0^{(2)} + w_1^{(2)} a_1 + w_2^{(2)} a_2 + ... + w_k^{(2)} a_k$$

- En d'autres mots (rappel : $a_0 = 1$) :

  Une nouvelle instance est de la classe 1 plutôt que de la classe 2 si :

$$\boxed{(w_0^{(1)} - w_0^{(2)}) a_0 + (w_1^{(1)} - w_1^{(2)}) a_1 + (w_2^{(1)} - w_2^{(2)}) a_2 + ... + (w_k^{(1)} - w_k^{(2)}) a_k > 0}$$

➙ C'est une expression linéaire sur les valeurs des attributs.

Régression et la classif. linéaire multi-réponses (suite)

Extension à n classes : méthode générale de Régression **multi-réponses** :

☞ **En Apprentissage** : une régression linéaire pour chaque classe
(traitée une par une)

→ Mettre la sortie/**classe = 1** pour les instances dans la classe, **0** sinon
→ Résultat : une expression linéaire pour chaque classe

☞ **En Prédiction** : toutes les équations évaluées pour la nouvelle instance
➛ la classe correspondra à la plus grande valeur de l'expression linéaire

➛ une sorte de fonction *d'appartenance :*
la plus grande valeur → *appartenance=vraie* (valeur = 1)

Décision : si *appartenance =1* → l'instance appartient à telle classe, 0 sinon

➛ Voir *Addendum* plus loin pour la justification.

☞ Cette manière de faire est simple mais insuffisante
➛ En particulier pour des problèmes non linéaires ../..

Régression et la classif. linéaire multi-réponses (suite)

Insuffisance de la décision (précédente) en régression linéaire :

• Conflit pour un cas bi-classe (C_1 et C_2) avec une régression sur C_1 et une pour C_2.

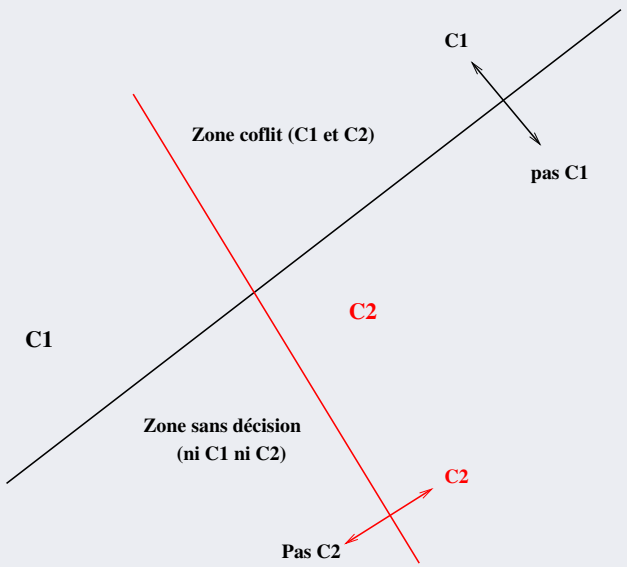


FIGURE 6: Cas bi-classes, conflit et région de non décision (pb. de séparabilité)

➙ Pondération / probabilité ... des classes prédites

Régression et la classif. linéaire multi-réponses (suite)

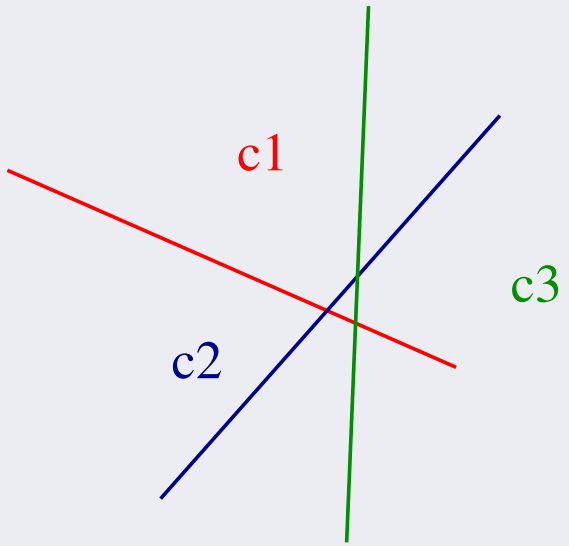
Une autre méthode pour la classification linéaire à n classes :

- Technique de la *régression par paire* :
  Une expression pour toute paire de classes.
- En apprentissage : **ne considère que** les instances des 2 classes considérées.
  - ➛ la frontière entre chaque paire de classe est un plan linéaire
  - ➛ un **hyperplan** dans l'espace des instances.

- Par cette technique, on apprendre à distinguer entre 2 classes

Inconvénient : les instances pas toujours séparée par un seul hyperplan,

Néanmoins, on peut utiliser le modèle linéaire pour construire des schémas plus complexes (e.g. les *arbres de modèles*).



Justification de la minimisation de l'erreur

$$\boxed{\mathbb{E}_y[(f(X) - Y)^2 | X = x]} \longleftarrow \textit{expression à minimiser,}$$

$f(X)$ = la prédiction, Y = le modèle connu **(0 ou 1)**, x = instance

$$\boxed{= \mathbb{E}_y\left[(f(X) - \underline{Pr}(Y = 1|X = x) + \underline{Pr}(Y = 1|X = x) - Y)^2 \ |X = x\right]}$$

$Pr(Y = 1|X = x)$ ↗ : la vraie probabilité de la classe (les 2 s'annulent)

$$\boxed{\begin{array}{l} = (f(X) - Pr(Y = 1|X = x))^2 + 2 \times (f(X) - Pr(Y = 1|X = x)) \times \\ \quad \mathbb{E}_y[Pr(Y = 1|X = x) - Y|X = x] + \mathbb{E}_y[(Pr(Y = 1|X = x) - Y)^2|X = x] \end{array}}$$

$$\boxed{\begin{array}{l} = (f(X) - Pr(Y = 1|X = x))^2 \qquad \swarrow \text{ la ligne svte = zéro (cf. p. svte. sur ☞)} \\ \quad +2 \times (f(X) - Pr(Y = 1|X = x)) \times (Pr(Y = 1|X = x) - \mathbb{E}_y[Y|X = x]) \\ \quad +\mathbb{E}_y[(Pr(Y = 1|X = x) - Y)^2|X = x] \end{array}}$$

$$\boxed{= (f(X) - Pr(Y = 1|X = x))^2 + \mathbb{E}_y[(Pr(Y = 1|X = x) - Y)^2|X = x]}$$

$(f(X) - Pr(Y = 1|X = x))^2 \longleftarrow$ à minimiser CQFD

$\mathbb{E}_y[(Pr(Y = 1|X = x) - Y)^2|X = x] \longleftarrow$ Constante $\cdots \rightsquigarrow$

Justification de la minimisation de l'erreur (suite)

A propos de la démonstration précédente :

- On a : $\mathbb{E}_y[Y|X=x]$ est constante car (voir page suivante) :
$$\mathbb{E}_y[Y|X=x]) = \sum_y y \cdot \Pr(Y=y|X=x)$$
Et puisque la variable discrète $y \in \{0,1\}$, on aura
$\mathbb{E}_y[Y|X=x]) = 1 \times \Pr(Y=1|X=x) + 0 \times \Pr(Y=0|X=x)$
$\mathbb{E}_y[Y|X=x]) = \Pr(Y=1|X=x)$ qui est une constante.

☞ A la page précédente : multiplication par zéro au milieu de l'expression.

- Il en découle que
$\mathbb{E}_y[(Pr(Y=1|X=x) - Y)^2|X=x]$ est un terme constant.

- Pour $X=x$, le terme à minimiser $\boxed{(f(X) - Pr(Y=1|X=x))^2}$
se réécrit, pour l'ensemble des instances $x^{(i)}$, en $\sum_{i=1}^{n} \left(f(x^{(i)}) - \sum_{j=0}^{k} w_j a_j^{(i)} \right)^2$

- Voir aussi en section "méthodes à base de noyau".

Justification de la minimisation de l'erreur (suite)

Rappels : l'espérance $\mathbb{E}$ est un opérateur linéaire :

- $\mathbb{E}(1) = 1$,
- $\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y)$
- $\mathbb{E}(aX) = a\mathbb{E}(X)$
- $\mathbb{E}(aX + b) = a\mathbb{E}(X) + b$
- $\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$

pour les variables aléatoires X, Y définies sur le même espace probabiliste et pour deux constantes réels a, b.

- Pour X aléatoire discrète et Y aléatoires :
  - $\mathbb{E}(X|Y) \equiv \sum_{x_i} Pr_Y(X = x_i) = \frac{\sum_{x_i} Pr(\{X=x_i\} \cap Y)}{Pr(Y)}$ $\quad$ X aléatoire discrète
  - $\mathbb{E}(X|Y) \equiv \frac{1}{Pr(Y)} \int_{X(Y)} x\, f(x)\, dx$ $\quad$ X continue de densité f
  - $\mathbb{E}_y(X|Y) \equiv \mathbb{E}(X|Y = y) \equiv \sum_x x \cdot \mathrm{P}(X = x|Y = y)$.
  - $\mathbb{E}(\mathbb{E}(X|Y)) = \mathbb{E}(X)$
  - $\mathbb{E}(XY) = \mathbb{E}(X)\mathbb{E}(Y)$ si les variables X et Y sont indépendantes.

Résumé Régression linéaire en classification

Généralisation de la méthode de régression linéaire dans la classification multi-réponses :

- En apprentissage : pour calculer les pondérations w_i, choisir une classe c_1 parmi C classes :
  - poser $w_0 + w_1\, a_1^i + \ldots w_k\, a_k^i = 1$ si la i^{eme} instance est de cette classe (i.e. $y^i = c_1$),
  - 0 sinon

  ➛ Résultat : une expression linéaire pour chaque classe c_i

- En test : ayant estimé les w_i pour chaque classe, pour une nouvelle instance x_u, calculer $w_0 + w_1\, a_1 + \ldots w_k\, a_k$ pour chaque classe, puis prendre **la plus grande** valeur.

Exemple avec 3 classes et une nouvelle instance donnée :
- Pour c_1, on obtient la valeur 1.05
- Pour c_2, , on obtient 0.95
- Pour c_3, , on obtient 0.88

➛ Résultat : c_1

Résumé Régression linéaire en classification (suite)

Généralisation de la méthode par **paires de classes** :

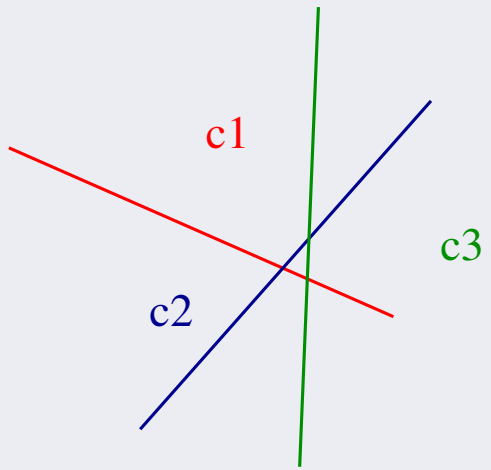
Apprentissage : choisir une paire de classes (c_1 et c_2)

- Ne considérer que les instances dont la valeur de y (classe connue) correspond à c_1 ou c_2 ; écarter les autres instances,
- On obtient une expression linéaire qui sépare les instances considérées

Test : pour connaître la classe d'une nouvelle instance, évaluer toutes ces expressions et prendre la classe la mieux représentée.

Exemple : une instance donne :
- entre c_1 et c_2 : c_1
- entre c_1 et c_3 : c_1
- entre c_2 et c_3 : c_2

➟ Résultat : c_1



Régression Logistique

- Caractéristiques du modèle linéaire général (1 pour l'appartenance, 0 sinon) :

  ➙ variables quantitatives, non bornées et (linéairement) dépendantes.

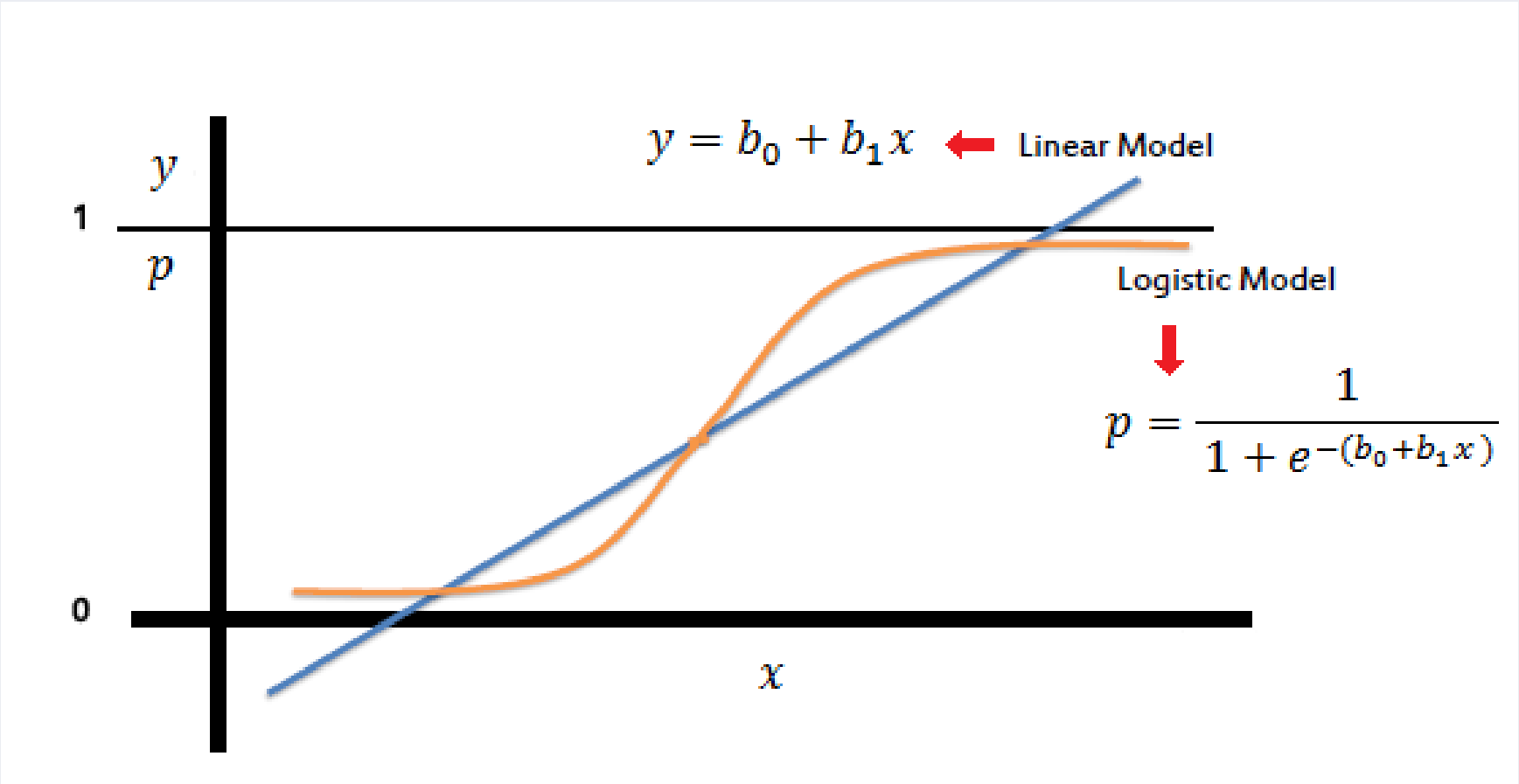
- **Inconvénients**

  1. La valeur d'appartenance calculée n'est pas une probabilité
     ➙ La valeur calculée peut être hors [0, 1]
     ➙ on prend la plus grande valeur $\rightsquigarrow$ 1
  2. La Régression suppose que les erreurs (moindre carré) sont i.i.d. et statistiquement indépendantes des observations.
  3. ... Et que ces erreurs ont une Distribution *Normale* (de moyenne idéalement nulle et avec le même écart type)

  ➙ **En général, ces conditions sont violées** dans la classification multi-réponses. .../...

Régression Logistique (suite)

Comparaison de la régression linéaire vs. Logistique :



Régression Logistique : cas bi-classses

Régression Logistique : une meilleure alternative pour la classification.

- Au lieu d'approximer 0 / 1 (risque de problème de probabilité), on transforme la variable (de la classe) → appelé *logit transformation*
  ➙ Liens avec *Bernoulli* (voir plus loin)

Cas bi-classe :

On remplace : $Pr[1|a_1, a_2, ..., a_k]$ par $ln\left(\frac{Pr[1|a_1,a_2,\ldots,a_k]}{1-Pr[1|a_1,a_2,\ldots,a_k]}\right)$

en posant $ln\left(\frac{Pr[1|a_1,a_2,\ldots,a_k]}{1-Pr[1|a_1,a_2,\ldots,a_k]}\right) = w_0 + a_1 w_1 + a_2 w_2 + ... + a_k w_k$

➙ Permet de tenir compte des valeurs dans $[-\infty, +\infty]$ (au lieu de [0, 1])

- C-à-d., au lieu de calculer $x = w_0 a_0 + w_1 a_1 + w_2 a_2 + ... + w_k a_k$ et de décider de la classe, on considère la valeur x comme l'expression :

$$ln\ \frac{P}{1-P} \qquad \text{où} \qquad P = Pr[1|a_1, ..., a_k]$$

N.B. : $P = Pr[1|a_1, ..., a_k]$ veut dire $P(Y = 1|X = x)$, avec x : instance, Y : classe.

- Voir Addendum ci-dessous pour $Pr[1|..]$ et pour $ln(...)$...

Régression Logistique : cas bi-classses (suite)

• La variable (de classe) est approximée en utilisant une fonction linéaire (comme pour la régression linéaire)

On pose $\boxed{ln\ \frac{P}{1-P} = w_0\,a_0 + w_1\,a_1 + w_2\,a_2 + ... + w_k\,a_k}$, $P = Pr[1|a_1, ..., a_k]$

➛ N.B. : $ln\ \frac{P}{1-P} = x = w_0\,a_0 + w_1\,a_1 + w_2\,a_2 + ... + w_k\,a_k$

De $ln\ \frac{P}{1-P}$, on obtient $P = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$

➛ Le modèle résultant : $\boxed{Pr[1|a_1, a_2, ..., a_k] = \frac{1}{1+e^{-w_0-w_1\,a_1-...-w_k\,a_k}}}$

• On trouve les pondérations w_i en phase d'apprentissage (cf. linéaire)

➛ Pour l'estimation de ces pondérations, on maximise la *vraisemblance*.

➛ Mieux : la Régression Logistique maximise la *log-vraisemblance*. ../..

Maximum de vraisemblance

• La **vraisemblance** d'un échantillon (x_i, y_i), x_i les valeurs observées et y_i la classe, ($y_i = 0$ *ou* 1 pour le cas bi-classes) :

$$p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} \qquad \text{où} \qquad p(x) = \frac{exp(w_0 + w_1\, a_1 + \ldots w_k\, a_k)}{1 + exp(w_0 + w_1\, a_1 + \ldots w_k\, a_k)}$$

• Dans notre cas ($x_i = x^{(i)}$) : $p(x_i) = Pr[1|a_1, ..., a_k] = \frac{e^x}{1+e^x}$ et

y_i pour $x^{(i)}$ est la classe associée au calcul de $x = w_0 + w_1\, a_1 + \ldots w_k\, a_k$

☞ en apprentissage, les expressions $x^{(i)}$ et y^i sont connues

• La vraisemblance $L(w_0, \ldots w_k) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

• Sachant que la croissance d'une fonction et de son log sont corrélées, pour maximiser la vraisemblance, on peut maximiser sa log-vraisemblance :

➙ Notation : $\ell(w_0, w_1, ..., w_k) = ln\ L(w_0, ..., w_k)$ **à maximiser** .

☞ Maximiser la log-vraisemblance = minimiser $-ln\ L(w_0, ..., w_k)$.

• Voir un exemple plus loin...

Maximum de vraisemblance (suite)

- Calcul de **l'erreur** (pour un cas logistique bi-classes) :

  On avait $\ell(w_0, w_1, ..., w_k) = ln\left(\prod_{i=1}^{n} p(x_i)^{y_i}(1-p(x_i))^{1-y_i}\right)$

  où $(x_i = x^{(i)})$: $p(x_i) = Pr[1|a_1, ..., a_k] = \frac{e^x}{1+e^x}$ et y_i pour $x^{(i)}$ est la classe associée au calcul de $x = w_0 + w_1\,a_1 + ...w_k\,a_k$ $\quad$ ($x^{(i)}$ et y^i sont connues)

  Ce qui donne le log-vraisemblance $\ell(w_0, w_1, ..., w_k) =$ (application de ln)

  $$\sum_{i=1}^{n}(1-y^{(i)})\ ln(1-Pr[1|a_1^{(i)}, a_2^{(i)}, ..., a_k^{(i)}]) + y^{(i)}\ ln(Pr[1|a_1^{(i)}, a_2^{(i)}, ..., a_k^{(i)}])$$

  $$= \sum_{i=1}^{n}\left[(1-y^{(i)})\ ln\left(1-\frac{e^{x^{(i)}}}{1+e^{x^{(i)}}}\right) + y^{(i)}\ ln\left(\frac{e^{x^{(i)}}}{1+e^{x^{(i)}}}\right)\right]$$

- Les w_i sont choisis pour maximiser *la vraisemblance* selon diff. méthodes :
  ➙ Dérivée, utilisation d'une régression linéaire (cf. ci-dessus) **pondérée**, ...

  ... ↝

Maximum de vraisemblance (suite)

Maximisation de la (log-)vraisemblance :

- Une méthode courante (avec peu d'itérations, voir plus loin) :

  On résout une séquence de régressions moindre-carré pondérées (calcul des $w_0\,a_0 + w_1\,a_1 + w_2\,a_2 + ... + w_k\,a_k$) jusqu'à obtenir une convergence de la log-vraisemblance ($ln\dfrac{P}{1-P}$) vers son **maximum**.

- Rappel du modèle logistique linéaire utilisé : trouver $\hat{P}$ telle que

$$\boxed{ln\frac{\hat{P}}{1-\hat{P}} = w_0\,a_0 + w_1\,a_1 + w_2\,a_2 + ... + w_k\,a_k = x}$$

avec $\hat{P} = Pr[1|a_1, a_2, ..., a_k] = \dfrac{e^x}{1+e^x} = \dfrac{1}{1+e^{-x}}$ = la probabilité de la classe

Et maximiser $\ell(w_0, w_1, ..., w_k) = \sum_{i=1}^{n}\left[(1-y^{(i)})\ ln\left(1-\dfrac{e^{x^{(i)}}}{1+e^{x^{(i)}}}\right) + y^{(i)}\ ln\left(\dfrac{e^{x^{(i)}}}{1+e^{x^{(i)}}}\right)\right]$

- La Régression Logistique fait **une estimation directe de la probabilité** de la classe d'une instance.

Régression Logistique : un exemple

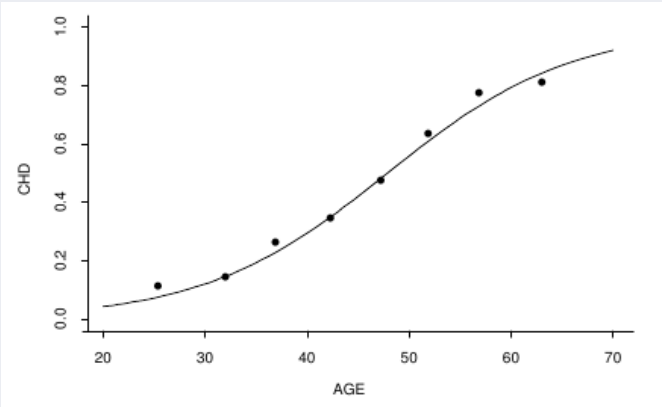
Exemple à une dimension : où AGRP=groupe d'âge, AGE (la variable explicative a_1 ci-dessous)

La classe binaire CHD = présence ou absence de maladie cardio-vasculaire.

$$\hat{p}(x) = \frac{exp(\hat{w_0}+\hat{w_1}\,a_1)}{1+exp(\hat{w_0}+\hat{w_1}\,a_1)}$$

- On trouve $\hat{w_0} = -5.31$ et $\hat{w_1} = 0.111$ et $\ell(\hat{w_0}, \hat{w_1}) = -53.677$
- La courbe s'adapte assez bien aux fréquences relatives de CHD selon AGE.

$$\hat{p} = Pr(CHD|AGE) = \frac{exp(-5.31+0.111AGE)}{1+exp(-5.31+0.111AGE)}$$



- N.B. : un test statistique T confirme que AGE est déterminant pour expliquer la probabilité de CHD=1.

ID	AGRP	AGE	CHD	ID	AGRP	AGE	CHD	ID	AGRP	AGE	CHD
1	1	20	0	35	3	38	0	68	6	51	0
2	1	23	0	36	3	39	0	69	6	52	0
3	1	24	0	37	3	39	1	70	6	52	1
4	1	25	0	38	4	40	0	71	6	53	1
5	1	25	1	39	4	40	1	72	6	53	1
6	1	26	0	40	4	41	0	73	6	54	1
7	1	26	0	41	4	41	0	74	7	55	0
8	1	28	0	42	4	42	0	75	7	55	1
9	1	28	0	43	4	42	0	76	7	55	1
10	1	29	0	44	4	42	0	77	7	56	1
11	2	30	0	45	4	42	1	78	7	56	1
12	2	30	0	46	4	43	0	79	7	56	1
13	2	30	0	47	4	43	0	80	7	57	0
14	2	30	0	48	4	43	1	81	7	57	0
15	2	30	0	49	4	44	0	82	7	57	1
16	2	30	1	50	4	44	0	83	7	57	1
17	2	32	0	51	4	44	1	84	7	57	1
18	2	32	0	52	4	44	1	85	7	57	1
19	2	33	0	53	5	45	0	86	7	58	0
20	2	33	0	54	5	45	1	87	7	58	1
21	2	34	0	55	5	46	0	88	7	58	1
22	2	34	0	56	5	46	1	89	7	59	1
23	2	34	1	57	5	47	0	90	7	59	1
24	2	34	0	58	5	47	0	91	8	60	0
25	2	34	0	59	5	47	1	92	8	60	1
26	3	35	0	60	5	48	0	93	8	61	1
27	3	35	0	61	5	48	1	94	8	62	1
28	3	36	0	62	5	48	1	95	8	62	1
29	3	36	1	63	5	49	0	96	8	63	1
30	3	36	0	64	5	49	0	97	8	64	0
31	3	37	0	65	5	49	1	98	8	64	1
32	3	37	1	66	6	50	0	99	8	65	1
33	3	37	0	67	6	50	1	100	8	69	1
34	3	38	0								

Addendum : maximisation log-vraisemblance

Pour un cas bi-classes :

- Rappel du modèle logistique linéaire utilisé : trouver $\hat{P}$ telle que :

$$\boxed{ln\frac{\hat{P}}{1-\hat{P}} = w_0\,a_0 + w_1\,a_1 + w_2\,a_2 + ... + w_k\,a_k = x}$$

avec $\hat{P} = Pr[1|a_1, a_2, ..., a_k] = \dfrac{e^x}{1+e^x} = \dfrac{1}{1+e^{-x}}$ = la probabilité de la classe

Et maximiser $\ell(w_0, w_1, ..., w_k) = \sum_{i=1}^{n}\left[(1-y^{(i)})\ ln\left(1-\dfrac{e^{x^{(i)}}}{1+e^{x^{(i)}}}\right) + y^{(i)}\ ln\left(\dfrac{e^{x^{(i)}}}{1+e^{x^{(i)}}}\right)\right]$

- Notons $P(x;w) = \hat{P} = Pr[1|a_1, a_2, ..., a_k] = \dfrac{e^x}{1+e^x}$

- Maximiser $\ell(w_0, w_1, ..., w_k) = \sum_{i=1}^{n}\left[(1-y^{(i)})\ ln\left(1-P(x^{(i)};w)\right) + y^{(i)}\ ln\left(P(x^{(i)};w)\right)\right]$

Addendum : maximisation log-vraisemblance (suite)

Maximiser $\sum_{i=1}^{n} \left[(1 - y^{(i)}) \; ln \left(1 - P(x^{(i)}; w) \right) + y^{(i)} \; ln \left(P(x^{(i)}; w) \right) \right]$

Avec $w_0 a_0 + w_1 a_1 + w_2 a_2 + ... + w_k a_k = x$ où $a_0 = 1$

- Soit les matrices w et x (k le nombre d'attributs) cas bi-classes

$$w = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ ... \\ w_k \end{pmatrix} \qquad a = \begin{pmatrix} 1 \\ a_1 \\ a_2 \\ ... \\ a_k \end{pmatrix}$$

- Si $p(x; w) = \frac{e^x}{1+e^x} = \frac{e^{wa}}{1+e^{wa}}$ pour une classe

  Alors $(1 - p(x; w)) = \frac{1}{1+e^{wa}}$ pour l'autre classe

➙ Il faut maximiser $\boxed{\ell(w) = \sum_{i=1}^{n} \left[y^{(i)} w^t a^{(i)} - ln(1 + e^{w^T a^{(i)}}) \right]}$

Addendum : maximisation log-vraisemblance (suite)

- Pour maximiser $\ell(w)$, on pose :

$$\begin{aligned}\frac{\partial \ell(w)}{\partial w_j} &= \sum_{i=1}^{n} y^{(i)} a_j^{(i)} - \sum_{i=1}^{n} \frac{a_j^{(i)} e^{w^T a^{(i)}}}{1 + e^{w^T a^{(i)}}} \\ &= \sum_{i=1}^{n} y^{(i)} a_j^{(i)} - \sum_{i=1}^{n} P(x^{(i)}; w).a_j^{(i)} \\ &= \sum_{i=1}^{n} a_j^{(i)} (y^{(i)} - P(x^{(i)}; w))\end{aligned}$$

  ➙ Pour $\forall j = 0..k$

- Plus généralement $$\frac{\partial \ell(w)}{\partial w} = \sum_{i=1}^{n} a^{(i)} (y^{(i)} - P(x^{(i)}; w))$$

  ➙ $a^{(i)}$ est un vecteur colonne et $y^{(i)}$ un scalaire.

Addendum : maximisation log-vraisemblance (suite)

• Pour résoudre les $p+1$ équations obtenues (toutes posées $= 0$), on peut utiliser par exemple la méthode *Newton-Raphson*.

• Cette méthode aura besoin de la matrice carrée de la dérivée seconde (= matrice de *Hessian*).

• Pour ce problème, la matrice de *Hessian* sera :

$$\rightarrow \frac{\partial^2 \ell(w)}{\partial w \partial w^T} = -\sum_{i=1}^{n} a^{(i)} a^{(i)^T} P(x^{(i)}; w)(1 - P(x^{(i)}; w))$$

• Selon le problème spécifique donné, la résolution Newton donnera les solutions attendues.

Régression logistique multi-classes

Généralisation au cas multi-classes : plusieurs méthodes

1- Soit comme pour le cas linéaire :
- On fait une Régression Logistique indépendante pour chaque classe
- La somme des estimations de probas ne sera pas = 1
  - ➙ nécessite une pondération (cf. Bayes naïve)

2- Soit (obtenir des probas réelles) :
- On couple des modèles individuels à chaque classe
- Par exemple $Pr[Classe = C_i|X = x_i]$ vs. $Pr[Classe \neq C_i|X = x_i]$
- Donne un problème d'*optimisation jointe*
  - ➙ il y a des méthodes efficaces pour le résoudre

3- **Une solution plus simple** : classification par paire

On peut décider pour un cas bi-classes (affecter la bonne classe),

➙ on généralise à n classes en faisant $\frac{n(n-1)}{2}$ classifications 2 à 2 ../..

Régression logistique multi-classes (suite)

Régression appareillée (par paires, 2 à 2) :

Apprentissage :

- Utiliser la Régression pour la classification :
  - Traite de 2 à n classes
  - Poser une fonction de Régression pour toute **paire de classes**, utilisant **uniquement les instances de ces 2 classes**
  - Habituellement : la sortie = **+1** pour l'une des 2 classes, **-1** pour l'autre
- Si l'on a k classes, on construit $k(k-1)/2$ classifications (appareillées)

Prédiction :

- La prédiction pour une nouvelle instance en *votant* :
  - La classe qui reçoit la majorité des votes est prédite
  - "ne sait pas" si pas de gagnant unanime (plus conservative) :
- Donne de meilleurs résultats (en erreur) mais plus complexe en calcul.

Inconvénient des données non séparables

☞ Cet inconvénient (général) existe aussi pour la Régression Logistique.

- Un exemple de cas à problème :

  Cas de deux classes quand $-w_0 - w_1\,a_1 - w_2\,a_2 - ... - w_k\,a_k = 0$
  ➛ Cette expression est une égalité linéaire entre les valeurs des attributs

- La frontière de décision est placée où la prédiction de probabilité = 0.5,
  ➛ C'est à dire que (dans $P = \frac{1}{1+e^{-x}}$) :

$$Pr[1|a_1, a_2, ..., a_k] = 1/(1 + exp(-w_0\,a_0 - w_1\,a_1 - w_2\,a_2 - ... - w_k\,a_k)) = \frac{1}{1+1} = 0.5$$

Décision impossible dans ce cas.

- La frontière dans l'espace des instances est un plan linéaire (un hyperplan)
- **On ne peut pas toujours** séparer toutes les instances par un seul hyperplan

Addendum : origines de la régression logistique

- On s'intéresse à la classification binaire (notion de succès / échec)

  ➙ Exemple : présence d'une maladie ou pas, survie ou pas, avoir un emploi ou pas, appareil fonctionne ou pas, ...

- Loi Bernoulli avec $p = P(Y = 1)$ pour le succès et $(1 - p)$ pour l'échec.
  ➙ Avec $E(Y) = p$, $\sigma^2(Y) = p(1 - p)$

- Pour le cas à une dimension (X est la variable explicative) :

  On pose $p(x) = P(Y = 1|X = x)$, $1 \geq p(x) \geq 0$
  ➙ $p(x)$ n'est donc pas linéaire !

- La fonction $p(x)$ peut être approchée par une CDF, par exemple une distribution Normale ϕ

  ➙ $p(x) = \phi(w_0 + w_1 x)$ où w_0, w_1 sont les paramètre du modèle.

- Si ϕ^{-1} est la fonction inverse (*transformation probit*) de ϕ, on aura :

$$\phi^{-1}(p(x)) = w_0 + w_1 x,$$

  C-à-d., une *relation linéaire* appelée **modèle probit**.

Addendum : origines de la régression logistique (suite)

- La forme la plus utilisée par la Rég. Log. est la distribution logistique notée :

$$p(x) = \phi(w_0 + w_1 x) = \frac{exp(w_0 + w_1 x)}{1 + exp(w_0 + w_1 x)}$$

  appelé **modèle logit** ou **logistique**.

- La transformation inverse (utilisée dans la régression *logistique*) :

$$\phi^{-1}(p(x)) = logit(p(x)) = ln\left(\frac{p(x)}{1 - p(x)}\right) = w_0 + w_1 x$$

  ➙ La fonction $logit(p(x))$ est appelée une *link function* (notée $F_L^{-1}(p(x))$).

- Le passage à la dimension k :

  ➙ $p(x_1, ..., x_k) = \frac{exp(w_0 + w_1 x_1, ..., w_k x_k)}{1 + exp(w_0 + w_1 x_1, ..., w_k x_k)}$

  ➙ Et la fonction link :

$$logit(p(x_1, ..., x_k)) = ln\left(\frac{p(x_1, ..., x_k)}{1 - p(x_1, ..., x_k)}\right) = w_0 + w_1 x_1 + ... + w_k x_k$$

Addendum : origines de la régression logistique (suite)

- Les paramètres w_i peuvent être estimés par différentes méthodes.
  - Par ex. *Maximum de vraisemblance* (sur la base des observations indépendantes, vu ci-dessus, voir aussi plus loin)
  - Elle fournit des estimations (des paramètres) sur la base d'une distribution *Normale* et une variance petite, en particulier si la taille des données n est large et le nombre des paramètres petit.
  - Des méthodes statistiques (e.g. BN) peuvent aussi être utilisées pour éliminer en amont les paramètres inutiles au modèle avec la prise en compte d'intervalles de confiance sur leur estimation (et T *test*).

Un exemple de la régression logistique

• Prédiction sur les données de défaillance sur un composant mécanique dans les véhicules Renault.

Buts :

1. Identifier et hiérarchiser l'influence des *variables explicatives* sur le *Taux de défaillance* moyenne du composant.

   Deux types de variables explicatives :
   - ➙ Caractéristique technique du véhicule ("type véhicule", "puissance moteur", etc.)
   - ➙ Les conditions d'utilisation ("pays", "type utilisateur", etc.)
2. Déterminer la corrélation entre les variables explicatives.
3. Prédiction

• Le modèle permet de de construire une méthodologie d'exploitation des **données ReX**.

Un exemple de la régression logistique (suite)

Les données ReX du problème :

- Récupérées pendant une certaine période de garanti (par les garagistes)
  - Identifiant du véhicule défaillant (Id unique, date de fabrication,..),
  - Caractéristiques techniques (puissance, type d'équipement, etc.),
  - Données de son utilisation (pays, kilométrage, etc)
  - Les données de maintenance.
- La variable cible : **Taux de défaillance.**
- Les variables explicatives (qui influencent le $Taux$) :
  - Type du véhicule : $V = \{V1, V2, ...V5\}$
  - Pays où l'incident a été enregistré : $Pays = \{P1, ...P10\}$
  - Type de boite de vitesse : $B = \{BVA(\text{automatique}), BVM(\text{manuel})\}$
  - Présence de climatisation : $C = \{CA(\text{présent}), DA(\text{absent})\}$
  - Type d'utilisateur : $U = \{S(\text{société}), P(\text{particulier})\}$
  - Puissance du moteur : $P = \{pw_1, ...pw_4\}$

Un exemple de la régression logistique (suite)

- Exemple de données (exploitables par les méthodes d'EC) :

Véhicule (V)	Puissance (P)	Clim (C)	Utilisateur (U)	Boit (B)	Pays	Nb. véhic.	Taux
V1	pw1	DA	S	BVA	P1	5857	0,022
V2	pw1	CA	P	BVM	P1	100	0
V3	pw2	CA	P	BVM	P3	750	0,015
V3	pw3	DA	S	BVA	P4	220	0,02
...	...	...	...	...	...	...	

- Régression Logistique : méthode intéressante dans le cas de classes binaires.
  ➙ Extensible au cas à n classes.

- **Prédiction** :
  Si pour chaque donnée :
  - les variables explicatives sont $X = x_1 ... x_p$
  - et la classe $Y \in \{0, 1\}$ (défaillant ou non),

  ALORS on note
  $P(x) = Pr(Y = 1|X = x)$ la proba d'être défaillant sachant X.

- Le modèle de régression logistique est estimé par la *log vraisemblance*.

$$log\left(\frac{P(x)}{1-P(x)}\right) = w_0 + w_1 x_1 + ... + w_p x_p$$

Un exemple de la régression logistique (suite)

Influence mutuelles des variables par la Régression Logistique

- Dans le cas des classes binaires, on peut calculer $Odds\ Ratio = OR$ (le rapport de cotes) qui est rapport entre les deux modalités :

$$OR(X = 1\ vs\ X = 0) = \frac{\left(\frac{P(1)}{1-P(1)}\right)}{\left(\frac{P(0)}{1-P(0)}\right)}$$

  Avec $\frac{P(1)}{1-P(1)}$ calcullées par la régression Logistique.

Par exemple :

- Pour la variable explicative X = genre $\in \{femme, homme\}$,

  ➙ $X = 1$ veux dire *genre=homme* et $X = 0$ veut dire *genre=femme*.

- On calcule, pour une **variable cible : une certaine maladie** (= la *classe*) :
  - le ratio entre $P(1)$ (être homme et être malade) et $1 - P(1)$ (homme et non malade)
  - le ratio entre $P(0)$ (femme et malade) et $1 - P(0)$ (femme non malade)
  - le ratio entre les deux est le *OddsRatio* (OR)

Un exemple de la régression logistique (suite)

Représentation Graphique

• Exemple de l'effet du *genre* (variable explicative) sur une certaine maladie (variable cible) :

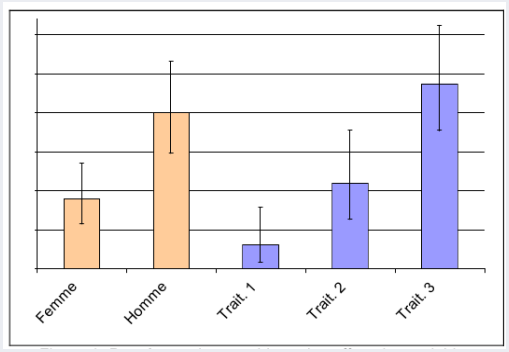


FIGURE 7: représentation graphique de **Odds Ratio**, attributs : homme/femme/trait1/...

• On constate dans cette figure : $OR(X = 1\ vs.\ X = 0) = \dfrac{\left(\frac{p(1)}{1-p(1)}\right)}{\left(\frac{p(0)}{1-p(0)}\right)} > 1$

pour les hommes ($X = 1$) et les femmes (X=0)
où $P(1)$ (être homme et être malade) et $1 - P(1)$ (homme et non malade)

Un exemple de la régression logistique (suite)

Retour à la Régression Logistique sur la BD

- Sur l'ensemble de la BD, on obtient les résultats suivants :
  ➙ 3 variables ont des effets statistiques significatifs sur le **Taux** de défaillance : "Type Véhicule", "type utilisation" et "Pays" :

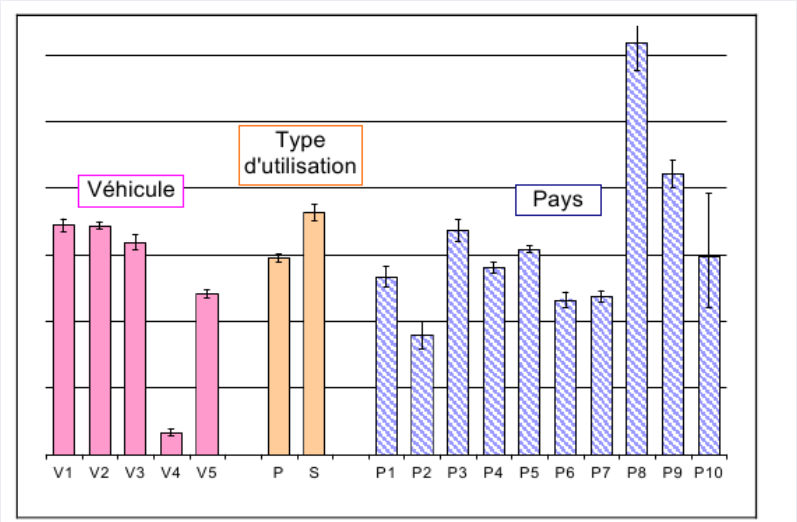


FIGURE 8: les 3 variables significatives sur la cible

➙ Pour réaliser cette figure, on a calculé l'effet de chacune des 3 variables

Un exemple de la régression logistique (suite)

• L'**interaction** entre "Véhicule" et "Type Utilisation" semble significative sur la défaillance.

➙ Elle montre l'effet du "Type Utilisation" sur la défaillance pour un "Véhicule" donné (P = "Particulier", S = "Société", V_i : différents types de "Véhicule").

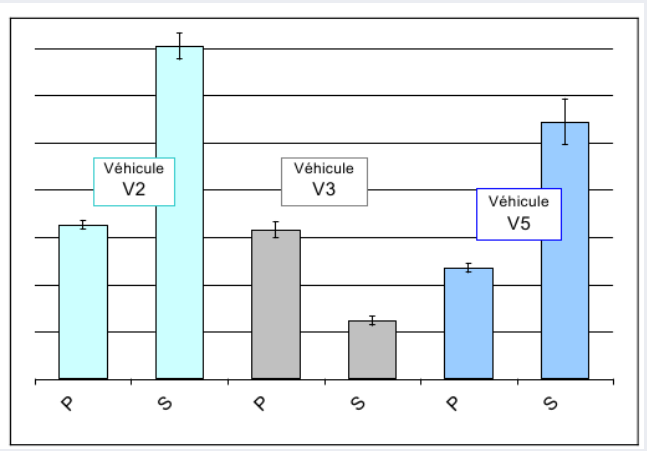
Par exemple :

- $OR(P\ vs.\ S|\, V_3) = 2.5$
- $OR(P\ vs.\ S|\, V_2) = 0.5$
- $OR(P\ vs.\ S|\, V_5) = 0.5$

On remarque (la figure) :

◦ S ("Société") est pénalisant pour la défaillance des véhicules du type V_2 et V_5

◦ Le type d'utilisation "particulier" est pénalisant pour V_3.



Un exemple de la régression logistique (suite)

• N.B. : les calculs ont également montré que le type de boite (BVA, BVM) a un effet statistique sur la défaillance.

➙ Le type BVA n'existe que sur des variantes particulières ("type véhicule" V_5, P : "Particulier" et CA : "avec climatiseur").

➙ Si on restreint les calculs à ces 3 sortes de véhicules, alors $OR(BVA\ vs.\ BVM) = 1.9$ montre que BVA est pénalisant pour la défaillance.

Avantages / Inconvénients de Régression logistique :

- ◦ Identification des variables significatives et leur hiérarchisation,
- ◦ Traitement des variables continues (selon l'outil utilisé).

➙ Par contre, l'interprétation des résultats (Odds Ratio par exemple) est quelque peu difficile (nécessité de post traitement).

➙ De même, si les variables sont fortement corrélées, il faut avoir recours aux techniques de sélection des variables (nécessité de pré traitement).

Introduction aux méthodes à base de noyau

• Idée de base : on peut séparer (classifier) un ensemble d'instances dans un espace autre que celui décrit par les données originales.

• Une méthode à base de noyau comprend 2 parties :

1. Une 'application' (*mapping*) des données vers l'**espace des caractéristiques** (*feature space*) noté *EdC* ici.
   - Ce *mapping* est appelé **fonction de noyau**.
   - Il dépend des données et du domaine des informations (Knowledge) concernant les motifs dans une BD.
2. On utilise un algorithme d'apprentissage pour découvrir des motifs dans cet espace.

   On doit considérer :
   - Les performances dans l'apprentissage de relations linéaires.
     ➛ Une complexité polynomiale (en taille de données) est souhaitable.
   - La Stabilité Statistique de l'algorithme.

Introduction aux méthodes à base de noyau (suite)

Illustration simple de la méthode :
on revisite la rég. linéaire moindre carré (c-à-d. la *régression supervisée*).

- On aura :
  - Les données d'apprentissage originales
  - Les *images* de ces données dans un espace vectoriel EdC (*feature space*).
  - Une relation linéaire sur ces images dans EdC.

☞ **Motivation** : disposer d'une méthode pour les BDs. où on ne trouve pas ou il n'y a pas de relation linéaire dans l'espace d'origine.

- Intérêt opérationnel de la méthode (voir plus loin) :
  l'exploitation des coordonnées des points dans EdC ne sont pas nécessaires mais seulement leur produit matriciel (interne) 2 à 2

  ➙ Ces produits matriciels sont calculés 2 à 2 sur les instances en entrée à partir de la BD en utilisant une fonction de noyau .

Introduction aux méthodes à base de noyau (suite)

- Illustration de la projection :

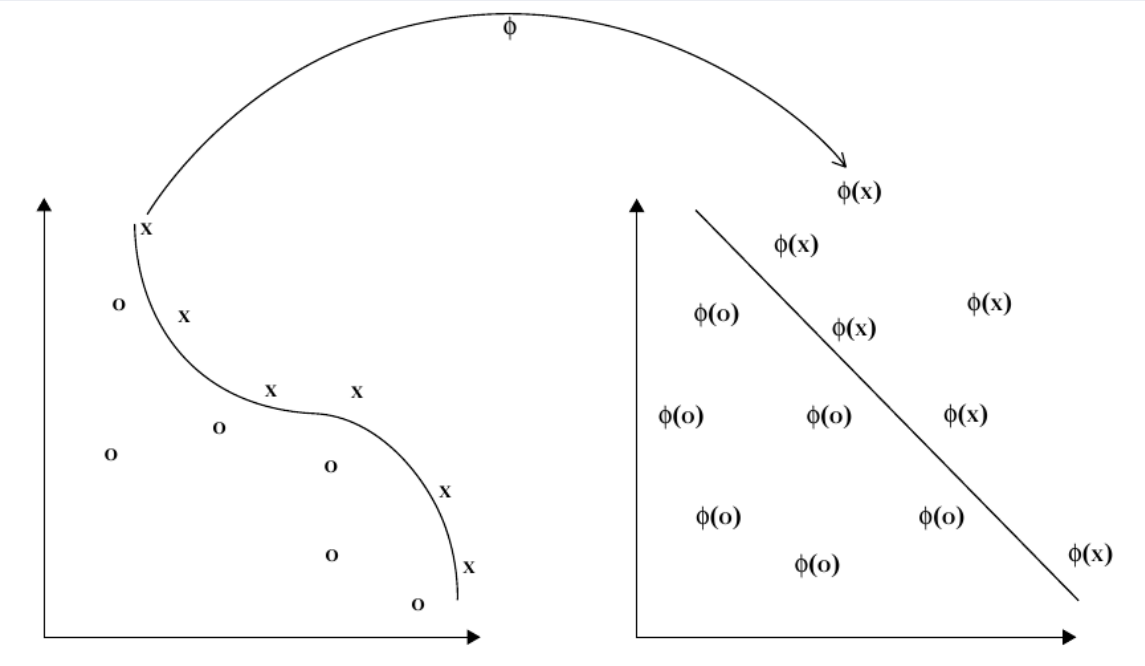


FIGURE 9: La fonction ϕ envoie les données dans un EdC où les motifs non-linéaires (d'origine) paraissent linéaires. Pour ce faire, le noyau calcule des produits matriciels dans EdC directement à partir des instances d'origine.

- Dans la section suivante : une introduction à ce cadre et ses calculs matriciels.

Régression linéaire dans EdC revisitée

Définition (et rappels)

Un **motif linéaire** *(pattern) est une fonction linéaire induite (apprise) sur un ensemble de motifs basés sur une fonction linéaire de classification.*

- Soit à rechercher une fonction linéaire homogène à valeur réelles

$$g(x) =< w, x >= w^T x = \sum_{i=1}^{n} w_i x_i$$

qui fait **la meilleure interpolation** étant donné un ensemble d'apprentissage $S = \{(x1, y1), ..., (x_\ell, y_\ell)\}$ de points x_i dans $X \subseteq \mathbb{R}^n$ avec les classes correspondantes $y_i \in Y \subseteq \mathbb{R}$.

- Le vecteur $x_i = (x_{i1}, x_{i2}, .., x_{in}) \in \mathbb{R}^n$ désigne une ième instance en entrée.

☞ $< w, x >= ||w||\ ||x||\ cos\theta \in \mathbb{R}$ désigne le produit **scalaire** (*dot /inner produc*) généralisé par le produit **vectoriel** (*cross/outer product*) noté $w \times x$ dans un espace vectoriel. Les deux ont la même interprétation géométrique.

Régression linéaire dans EdC revisitée (suite)

Remarque sur le produit scalaire et la distance :

- Le prod. scalaire correspondra à une distance (la normale) entre y et $g(x)$.

  Cela revient à considérer la perpendiculaire à la droite plutôt que les ξ qu'on utilise : la normale à la droite vaut le produit scalaire $||A||\ ||B||\ cos(\Theta)$.

Rappels prod. scalaire (dot, inner) et prod. vectoriel (cross, outer) :

- Produit scalaire : de 2 vecteurs $\overrightarrow{A} = [a_1, ..., a_n]$ et $\overrightarrow{B} = [b_1, ..., b_n]$ est noté

$$\overrightarrow{A}.\overrightarrow{B} = \sum_{i=1}^{n} a_i\, b_i = a_1\, b_1 + a_2\, b_2 + ...a_n\, b_n$$

(correspond à un point entre les 2 vecteurs)

Le Produit vectoriel **généralise le produit scalaire** pour créer des espaces vectoriels abstraits dans $\mathbb{R}$ est noté (règle du pouce) :

- $< A, B >= \overrightarrow{A} \times \overrightarrow{B} = -\overrightarrow{B} \times \overrightarrow{A} = ||\overrightarrow{A}|| \times ||\overrightarrow{B}||.sin\Theta$
- Est une fonction $V \times V \to V$ pour V un espace vectoriel.

$$\vec{A} \times \vec{B} = \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} \times \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} = \begin{pmatrix} A_y B_z - A_z B_y \\ A_z B_x - A_x B_z \\ A_x B_y - A_y B_x \end{pmatrix}$$

Point de vue génératif : critères

Rappels : $g(x) =< w, x >= w^T x = \sum_{i=1}^{n} w_i x_i$

On cherche **la meilleure** interpolation $g(.)$ sur un ensemble d'apprentissage $S = \{(x1, y1), ..., (x_\ell, y_\ell)\}$ de points $x_i \in \mathbb{R}$ avec les classes $y_i \in \mathbb{R}$.

• **g(.)** : une **fonction linéaire** des caractéristiques x qui *matchent* un label y, créant une *fonction de motif* (pattern function) $f(x, y)$ qui vérifie :

$$f((x, y)) = |y - g(x)| = |y - < w, x > | \approx 0$$

- Correspond à considérer la **distance** (*la normale*) entre y et $g(x)$.
- Dans $y = g(x)$: g est la fonction à apprendre (à inférer, à induire) ;
- g est une fonction qui, appliquée à x "devine" le label y de sorte que la valeur "devinée" $(g(x))$ soit la plus proche du vrai label y

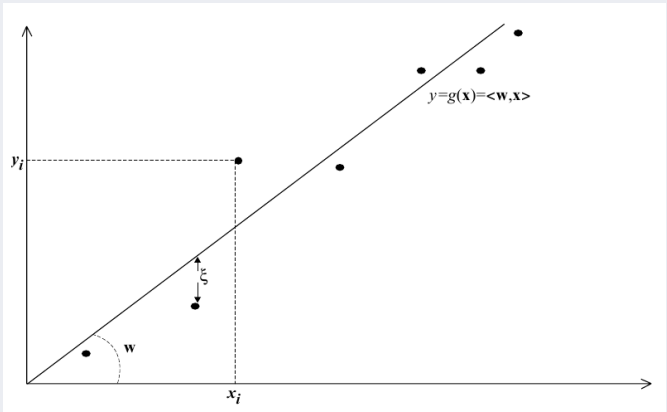
☞ Le cas idéal $g(x) = y$ est quasi impossible sauf dans des cas très triviaux.

Point de vue génératif : critères (suite)

- Le but (de l'apprentissage) est de **découvrir** $g(x)$ **qui a "généré"** y :
  ➙ les couples $(x, y) \in X \times Y$ sont connus mais on ne sait pas par quelle fonction un y_i a été associée à un x_i.

- Cette recherche est également appelée **interpolation linéaire** :
  ➙ Du point de vu géométrique, $g(x)$ correspond à un hyperplan étant donné les points à n-dimensions.

- La figure montre un exemple pour $n = 1$ (dimension de X) où y_i est le label (classe) de x_i (et le vecteur des ξ dont on minimisera la norme L_2 (i.e. $||\xi||^2$)).



- Dans le cas à une dimension, la droite $g(x)$ interpole les points.
  - D'une manière générale, on assigne un label à chaque x par la droite $g(x)$
  - En classification, $g(x)$ représente la classe prédite (le label) de x,
  - On aimerait que ce label soit le plus proche de y, le vrai label de x.

Point de vue génératif : critères (suite)

Conditions pour le point de vu *génératif* :

Lorsque les données $S = \{(x1, y1), ..., (x_\ell, y_\ell)\}, x_i \in \mathbb{R}^n$ sont générées sous forme de couples $(x, g(x))$, où $g(x) =< w, x >$ avec exactement $\ell = n$ points linéairement indépendants, il est possible de trouver les paramètres w en résolvant le système d'équations linéaires $Xw = y$

où X dénote la matrice des données (les points) dont les lignes sont des vecteurs lignes $x_1^T, ..., x_\ell^T$ et y dénote le vecteur $(y_1, ..., y_\ell)^T$.

- Le critère "exactement $\ell = n$ points" ci-dessus n'est pas toujours satisfait.

➙ 3 autres cas possibles :

❶ Si $\ell < n$ (il y a moins de points que de dimension), il y aura de multiples w possibles qui décrivent les données (il faut un critère pour en choisir un).

→ Attention au sur-apprentissage.

→ Dans ce cas, on choisira le vecteur w avec une norme minimum.

Point de vue génératif : critères (suite)

❷ Si $\ell > n$ (il y a plus de données que la dimension) avec du bruit dans le processus de génération, alors on ne s'attend pas à trouver des motifs exacts.

→ Dans ce cas, un critère d'approximation est nécessaire.

→ Dans une telle situation, on choisira le motif avec le minimum d'erreur.

❸ Et si on manipule **peu de données bruitées**, on combinera ces 2 stratégies et on trouve le vecteur w qui aura à la fois une petite norme avec un minimum d'erreur.

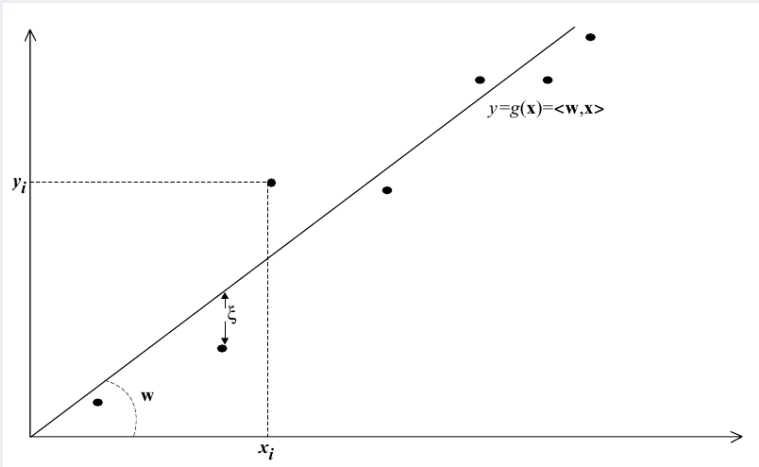
Erreur : la distance ξ dans la figure est l'erreur de la fonction linéaire sur une BD :

$$\xi = (y - g(x))$$

qui est la sortie de la *pattern function* :

$$f((x, y)) = |y - g(x)| = ||\xi||.$$

But : trouver une *pattern function* qui **minimise** toutes ces erreurs.



Minimisation de l'erreur

Rappel : l'erreur $\xi = (y - g(x))$, la fonction $f((x, y)) = |y - g(x)| = ||\xi||$

• La **somme des carrés** de ces erreurs est la mesure la plus utilisée indiquant la dispersion collective des données d'apprentissage.

• La fonction de **perte** : $\mathcal{L}(f, S)$ dénote la **perte collective** d'une fonction f sur l'ensemble d'apprentissage S.

• Dans notre cas (*la moindre carrée*) :

$$\mathcal{L}(g, S) = \mathcal{L}(w, S) = \sum_{i=1}^{\ell} (y_i - g(x_i))^2 = \sum_{i=1}^{\ell} \xi_i^2 = \mathcal{L}((x_i, y_i), g)$$

où $\mathcal{L}((x_i, y_i), g) = \xi_i^2$ est le carré d'erreur (*moindre carrée* dite **perte**) de g sur l'instance (x_i, y_i)

• Le problème d'apprentissage devient :
le choix du vecteur $w \in W$ qui minimise cette erreur. ../..

Minimisation de l'erreur (suite)

La méthode utilisée : **approximation moindre carré**.

➢ Introduite par *Gauss*, largement utilisée dans les minimisations d'erreurs.

- Le vecteur de la dispersion en sortie (suivant la notation) est donné par :

$$\xi = y - g(x) = y - Xw$$

- La fonction de **perte** (collective) sera :

$$\mathcal{L}(w, S) = \| \xi \|_2^2 = (y - Xw)^T (y - Xw) \qquad (2.1)$$

➢ La *norme L2* $\| \xi \|_2$ = la *taille* du vecteur ξ qui sera levée au carré (méthode moindre carrés).

- Rappel : pour M une matrice $m \times n$, M au carrée est : $< M, M >= M^T M$

  Ici $||\xi||_2 = |y - g(x)| = |y - Xw| \quad \rightarrow \quad \| \xi \|_2^2 = (y - Xw)^T (y - Xw)$.

  La norme *L2* d'une matrice M est : $\sqrt{\sum_{ij} M_{ij}^2}$

Minimisation de l'erreur (suite)

On peut chercher un w optimal en utilisant la dérivée de la *perte* par rapport aux paramètre w (la dérivée posée = vecteur nul pour un extrémum) :

$$\frac{\partial \mathcal{L}(w, S)}{\partial w} = -2X^T y + 2X^T Xw = 0$$

➙ d'où **l'équation** (dite *normale*) pour minimiser l'erreur :

$$X^T Xw = X^T y \qquad (2.2)$$

- Remarques naïves :
  - Le résultat $X^T Xw = X^T y$ (équation 2.2) semble empiriquement cohérent avec la forme non matricielle vue plus haut $Xw = y$.
  - **L'erreur à ne pas faire** est de ne pas passer par $\| \xi \|_2^2$ mais directement par $Xw = y$.

  → Cette égalité n'est jamais atteinte (un réel n'a jamais une valeur précise).

  → De plus, elle pose la question de l'inverse de la matrice X.

Minimisation de l'erreur (suite)

Si l'inverse de $X^T X$ existe, la solution du problème des moindre carrés peut être formulée par :

$$w = (X^T X)^{-1} X^T y$$

- Pour minimiser la *perte* au carrée de l'interpolation linéaire, on a besoin de maintenir toute la dimension en résolvant un système $n \times n$ d'équations linéaires avec un cout cubique en n (c-à-d. $O(n^3)$).

- **Prédiction** de la classe d'une nouvelle instance :

> La sortie (classe) d'une nouvelle instance peut être calculée à l'aide de la fonction de prédiction $g(x)$ pour une nouvelle instance x :
>
> $$g(x) =< \hat{w}, x >$$

➙ $\hat{w}$ (w optimal) aura été estimé par le calcul via $w = (X^T X)^{-1} X^T y$

Représentation Duale

Rappel : $\hat{w} = (X^T X)^{-1} X^T y$

- Si l'inverse de $X^T X$ existe, on peut exprimer w de la manière suivante :

$$w = (X^T X)^{-1} X^T y = X^T X (X^T X)^{-2} X^T y = X^T \alpha$$

qui transforme w en une combinaison des données d'apprentissage :

$$w = \sum_{i=1}^{\ell} \alpha_i \; x_i$$

- Si l'inverse de $X^T X$ n'existe pas
  - **Pseudo-inverse** : Si $X^T X$ est *singulière* (a des val. propres 0 → non inversible), la pseudo-inverse est utilisée pour trouver w de l'équation
    $X^T X w = X^T y$ (2.2) avec une norme $||w||$ minimale.
  - Autre possibilité (si $X^T X$ est singulière) : *Régularisation*
    Chercher un compromis entre la norme et la perte (vue + haut) :
    ➞ Approche **Régression Ridge** (Régression d'arête ou Pénalisée) ../..

Pour 18-19 : mettre ceci

• Le livre IA machine learning Python (Santanu Pattanayak-Pro Deep Learning with TensorFlow. A Mathematical Approach...pdf)
• Il n'y a pas de No page dans ce pdf (!!), mais dans tabmat, il y a "dim reduc" qui indique P79, et dans cette partie, il y a :
  ◦ Discussion sur biais variance avec un ex très simple (voir fig ci-dessous)
  ◦ sur la réduc de dim (PCA, SVD)puis vient la régularisation (voir tabmat) !
  ◦ Puis le gars dit : su régularisation avec norme l2, alors Ridge (régression).
Si régularisation avec la norme 1, alor slasso. J'ai trouvé simple et clair !

Regularization

The process of building a machine-learning model involves deriving parameters that fit the training data. If the model is simple, then the model lacks sensitivity to the variation in data and suffers from high bias. However, if the model is too complex, it tries to model for as much variation as possible and in the process models for random noise in the training data. This removes the bias produced by simple models but introduces high variance; i.e., the model is sensitive to very small changes in the input. High variance for a model is not a good thing, especially if the noise in the data is considerable. In such cases, the model in the pursuit of performing too well on the training data performs poorly on the test dataset since the model loses its capability to generalize well with the new data. This problem of models' suffering from high variance is called *overfitting*.

As we can see in Figure 1-47, we have three models fit to the data. The one parallel to the horizontal is suffering from high bias while the curvy one is suffering from high variance. The straight line in between at around 45 degrees to the horizontal has neither high variance nor high bias.

Pour 18-19 : mettre ceci (suite)

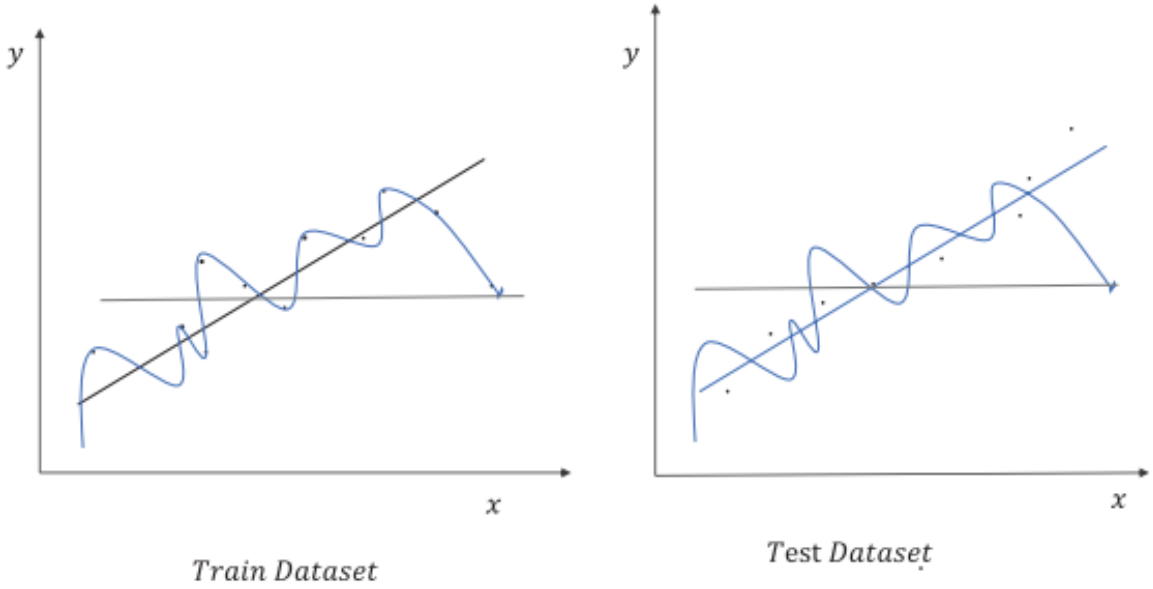


Figure 1-47. *Illustration of models with high variance and high bias*

The model with high variance does well on the train data but fails to do well on the test dataset, even when the dataset's nature hasn't changed much. The model represented in blue color may not fit the training perfectly, but it does better on the test data since the model doesn't suffer from high variance. The trick is to have a model that doesn't suffer from high bias and at the same time isn't so complex that it models for random noise as well.

Models with high variance typically have model parameters with a large magnitude since the sensitivity of the models to small changes in data is high. To overcome the problem of overfitting resulting from high model variance, a popular technique called regularization is widely used.

To put things into perspective, let's look at the linear regression cost function that we looked at earlier:

Pour 18-19 : mettre ceci (suite)

$$C(\theta) = \|X\theta - Y\|_2^2$$

$$= (X\theta - Y)^T (X\theta - Y)$$

As discussed earlier, models with high variance have model parameters with a large magnitude. We can put an extra component into the cost function $C(\theta)$ that penalizes the overall cost function in case the magnitude of the model parameter vector is high.

So, we can have a new cost function, $L(\theta) = \|X\theta - Y\|_2^2 + \lambda\|\theta\|_2^2$, where $\|\theta\|_2^2$ is the square of the l^2 norm of the model parameter vector. The optimization problem becomes

$$\theta^* = \underbrace{Arg\ Min}_{\theta} L(\theta) = \|X\theta - Y\|_2^2 + \lambda\|\theta\|_2^2$$

Taking the gradient ∇L with repect to θ and setting it to 0 gives us $\theta^* = \left(X^T X + \lambda I\right)^{-1} X^T Y$.

Pour 18-19 : mettre ceci (suite)

Now, as we can see because of the $\|\theta\|_2^2$ term in the cost function, the model parameter's magnitude can't be too large since it would penalize the overall cost function. λ determines the weight of the regularization term. A higher value for λ would result in smaller values of $\|\theta\|_2^2$, thus making the model simpler and prone to high bias or underfitting. In general, even smaller values of λ go a long way in reducing the model complexity and the model variance. λ is generally optimized using cross validation.

When the square of the l^2 norm is used as the regularization term, the optimization method is called l^2 regularization. At times, the l^1 norm of model parameter vectors is used as the regularization term, and the optimization method is termed l^1regularization. l^2 regularization applied to regression problems is called ridge regression, whereas l^1 regularization applied to such regression problems is termed lasso regression.

For l^1 regularization, the preceding regression problem becomes

$$\theta^* = \underbrace{Arg\ Min}_{\theta}\ L(\theta) = \|X\theta - Y\|_2^2 + \lambda\|\theta\|_1$$

Ridge regression is mathematically more convenient since it has a closed-form solution whereas lasso regression doesn't have a closed-form solution. However, lasso regression is much more robust to outliers in comparison to ridge regression. Lasso problems give sparse solutions, so it's good for feature selection, especially when there is moderate to high correlation among the input features.

Pour 18-19 : mettre ceci (suite)

$$\theta^* = argmin_\theta C(\theta) = \|X\theta - Y\|_2^2$$

such that $\|\theta\|_2^2 \leq b$

where b is a constant.

We can convert this constrained minimization problem to an unconstrained minimization problem by creating a new Lagrangian formulation, as seen here:

$$L(\theta,\lambda) = \|X\theta - Y\|_2^2 + \lambda\left(\|\theta\|_2^2 - b\right)$$

To minimize the Lagrangian cost function per the Karush Kuhn Tucker conditions, the following are important:

- The gradient of L with respect to θ; i.e., $\nabla_\theta L(\theta,\lambda)$ should be the zero vector, which on simplification gives

$$\theta = \left(X^T X + \lambda I\right)^{-1} X^T Y \quad (1)$$

- Also at the optimal point $\lambda\left(\|\theta\|_2^2 - b\right) = 0$ and $\lambda \geq 0$

  If we consider regularization, i.e., $\lambda > 0$, then $\|\theta\|_2^2 - b = 0$ (2)

Pour 18-19 : mettre ceci (suite)

As we can see from (1), θ obtained is a function of λ. λ should be adjusted such that the constraint from (2) is satisfied.

The solution $\theta = \left(X^T X + \lambda I\right)^{-1} X^T Y$ from (1) is the same as what we get from l^2 regularization. In machine-learning applications, the Lagrange multiplier is generally optimized through hyper parameter tuning or cross-validation since we have no knowledge of what a good value for b would be. When we take small values of λ the value of b increases and so does the norm of θ, whereas larger values of λ provide smaller b and hence a smaller norm for θ.

Coming back to regularization, any component in the cost function that penalizes the complexity of the model provides regularization. In tree-based models, as we increase the number of leaf nodes the complexity of the tree grows. We can add a term to the cost function that is based on the number of leaf nodes in the tree and it will provide regularization. A similar thing can be done for the depth of the tree.

Even stopping the model-training process early provides regularization. For instance, in gradient descent method the more iterations we run the more complex the model gets since with each iteration the gradient descent tries to reduce the cost-function value further. We can stop the model-learning process early based on some criteria, such as an increase in the cost function value for the test dataset in the iterative process. Whenever in the iterative process of training the training cost-function value decreases while the test cost-function value increases, it might be an indication of the onset of overfitting, and thus it makes sense to stop the iterative learning.

Whenever training data is less in comparison to the number of parameters the model must learn, there is a high chance of overfitting, because the model will learn too many rules for a small dataset and might fail to generalize well to the unseen data. If the dataset is adequate in comparison to the number of parameters, then the rules learned are over a good proportion of the population data, and hence the chances of model overfitting go down.

- Un (très) peu plus loin dans le mm coin :

Pour 18-19 : mettre ceci (suite)

Regularization Viewed as a Constraint Optimization Problem

Instead of adding the penalty term, we can add a constraint on the magnitude of the model parameter vector to be less than or equal to some constant value. We can then have an optimization problem as follows:

$$\theta^* = argmin_\theta\, C(\theta) = \|X\theta - Y\|_2^2$$

such that $\|\theta\|_2^2 \le b$

where b is a constant.

We can convert this constrained minimization problem to an unconstrained minimization problem by creating a new Lagrangian formulation, as seen here:

$$L(\theta,\lambda) = \|X\theta - Y\|_2^2 + \lambda\left(\|\theta\|_2^2 - b\right)$$

To minimize the Lagrangian cost function per the Karush Kuhn Tucker conditions, the following are important:

- The gradient of L with respect to θ; i.e., $\nabla_\theta L(\theta,\lambda)$ should be the zero vector, which on simplification gives

$$\theta = \left(X^T X + \lambda I\right)^{-1} X^T Y \qquad (1)$$

- Also at the optimal point $\lambda\left(\|\theta\|_2^2 - b\right) = 0$ and $\lambda \ge 0$

  If we consider regularization, i.e., $\lambda > 0$, then $\|\theta\|_2^2 - b = 0$ (2)

As we can see from (1), θ obtained is a function of λ. λ should be adjusted such that the constraint from (2) is satisfied.

The solution $\theta = \left(X^T X + \lambda I\right)^{-1} X^T Y$ from (1) is the same as what we get from l^2 regularization. In

Addendum : Régularisation et Régression Ridge

• Il y a des situations d'échec pour trouver une classe y exacte (par manque de données fiables) :

(1) Soit parce que il n'y a pas assez de données pour assurer que la matrice $X^T X$ est inversible,

(2) Soit beaucoup de bruits dans les données tels que les sorties (classes) ne sont pas fiables : mieux vaut ne pas y coller !
→ (connu sous le nom de problèmes *mal-conditionnés*).

• De plus, la rareté de données fiables peut introduire une co-linéarité entre les instances (du moins, un over/under-fitting)

• Approche ***régularisation*** : restreindre le choix des fonctions par un biais

① Une solution possible : rechercher des fonctions avec une petite norme.

② Une autre solution (+ classique) : pour le cas de moindre carré, il s'agit d'un critère d'optimisation de Régression d'Arête (**ridge regression**).

Addendum : Régularisation et Régression Ridge (suite)

Intérêts de la Régression Ridge (utilisée pour les calculs) :

➙ Utilisé dans le cas de moindre carré (avec éventuellement peu de données fiables)

- Contourne le problème de co-linéarité des variables explicatives.
- Introduit un biais sur les estimations des paramètres
  ➙ la *Moindre Carrée* (MC) classique est non biaisée.
- Ce biais réduit la variance des paramètres estimés ainsi que celle de l'Erreur MC
- Est un compromis "biais-variance" (i.e. méthode Perte+Norme).

➙ Voir aussi *Régularisation Tychonoff* : généralisation de la *régularisation*.

Addendum : Régularisation et Régression Ridge (suite)

- **Régression Ridge** correspond à résoudre le problème d'optimisation

$$min_w \mathcal{L}_\lambda(w, S) = min_w \left[\lambda \parallel w \parallel^2 + \sum_{i=1}^{\ell} (y_i - g(x_i))^2 \right] \qquad (2.3)$$

où $||w||^2 = \sum_n w_i^2$, *la dimension* $n = nb$ *attributs*,

la constante prédéfinie $\lambda \geq 0$ représente l'écart relatif entre la *norme* et la *perte* ; ce qui permet de contrôler le degré de *régularisation*.

☞ Ridge impose davantage de contraintes sur les w_i du modèle Linéaire :

➞ Au lieu de juste optimiser la somme des carrés résiduels, on a une pénalité sur les w_i via le terme de la pénalité $\lambda \, ||w||^2$ (λ constante prédéfinie).

- Rappel : $g(x)$ dépend de w, et $\sum_{i=1}^{\ell} (y_i - g(x_i))^2$ est la perte $\mathcal{L}$ vue ci-dessus.
- Le problème d'apprentissage revient à résoudre ce problème d'optimisation dans $\mathbb{R}^n$.

Addendum : Régularisation et Régression Ridge (suite)

Régression Ridge : formes primale et duale

• On utilise la dérivée de la fonction *cout* (la *perte* $\mathcal{L}_\lambda$) par rapport aux paramètres et on obtient l'équation

$$X^T X w + \lambda w = (X^T X + \lambda I_n) w = X^T y \qquad (2.4)$$

où I_n est la matrice $n \times n$ d'identité.

Rappel : $\dfrac{\partial \mathcal{L}(w, S)}{\partial w} = 0$ avait produit $X^T X w = X^T y$ (2.2)

☞ La matrice $(X^T X + \lambda I_n)$ est toujours inversible pour $\lambda > 0$

• **La solution sera (Primal) :**

$$\boxed{w = (X^T X + \lambda I_n)^{-1} X^T y} \qquad (2.5)$$

➙ La résolution de cette équation pour w implique de résoudre le système d'équations linéaires avec n inconnues et n équations.

➛ N.B. : si $\lambda = 0$, on revient à la forme simple précédente (vue en 2.2)

Addendum : Régularisation et Régression Ridge (suite)

... Avec $$w = (X^T X + \lambda I_n)^{-1} X^T y \qquad (2.5)$$

• **La fonction de prédiction** résultante est donnée par (x = nouvelle instance)

$$g(x) =< w, x >= w^T x = y^T X (X^T X + \lambda I_n)^{-1} x$$

• La complexité de ce calcul est $O(n^3)$.

• Remarques en marge et rappels sur ces calculs matriciels :

$$(A^T)^{-1} = (A^{-1})^T \ ,$$
$$(A + B)^T = A^T + B^T,$$
$$(\lambda I_n)^T = \lambda I_n,$$
$$(AB)^T = B^T A^T$$

$$(AB) = trace(A^T.B)$$

Addendum : Régularisation et Régression Ridge (suite)

La forme Duale : alternativement, on peut ré-écrire l'équation (2.4) en termes de w pour obtenir :

- $X^T X w + \lambda w = X^T y$ de l'équation (2.4)
- $\lambda^{-1} X^T X w + w = \lambda^{-1} X^T y$ on divise par $\lambda \neq 0$
- $w = \lambda^{-1} X^T (y - Xw) = X^T \alpha$ avec $\alpha = \lambda^{-1}(y - Xw)$

- Sachant que w peut être une combinaison linéaire des instances :

$$w = \sum_{i=1}^{\ell} \alpha_i \; x_i \qquad \text{avec} \qquad \alpha = \lambda^{-1}(y - Xw)$$

On a alors (Duale) :

$$\alpha = \lambda^{-1}(y - Xw)$$

- $\alpha\lambda = (y - XX^T\alpha)$ de ci-dessus où $w = X^T\alpha$
- $(XX^T + \lambda I_\ell)\alpha = y$
- $$\alpha = (G + \lambda I_\ell)^{-1} y \qquad (2.6)$$

où $G = XX^T$ ou encore $G_{ij} = < x_i, x_j >$.

Addendum : Régularisation et Régression Ridge (suite)

- La fonction de prédiction résultante sera (x nouvelle instance) :

  ➙ On avait $w = \sum_{i=1}^{\ell} \alpha_i \; x_i$

  ➙ $g(x) =< w, x >= \langle \sum_{i=1}^{\ell} \alpha_i \; x_i \; , \; x \rangle = \sum_{i=1}^{\ell} \alpha_i \; < x_i, x >$ $\qquad\qquad$ α de (2.6)

  ➙ $g(x) = y^T (G + \lambda I_\ell)^{-1} k$

  où $G = XX^T \rightarrow G_{ij} =< x_i, x_j >$
  et $g(x) =< w, x >= w^T x$
  et $k_i = < x_i, x >$.

- La matrice $G = XX^T$ est appelée la **matrice de noyau Gram** (v. +loin).
  ➙ La matrice de Gram et la matrice $(G + \lambda I_\ell)$ sont de dimension $\ell \times \ell$.

Complexité : résoudre sur α implique de résoudre ℓ équations linéaires avec ℓ inconnues, de complexité $O(\ell^3)$ (principalement le cout d'inversion de G).

Addendum : Régularisation et Régression Ridge (suite)

Résumé (Ridge Régression) :

• On a donc trouvé 2 méthodes distinctes pour résoudre une optimisation de Régression Ridge de l'équation (2.3).

(1) **primale** : équation (2.5) où on calcule le vecteur de pondérations w explicitement : $w = (X^T X + \lambda I_n)^{-1} X^T y$

(2) **duale** : l'équation (2.6) donne la solution comme une combinaison linéaire des instances d'apprentissage ($w = X^T \alpha$)

$$w = \sum_{i=1}^{\ell} \alpha_i \; x_i \qquad \text{où} \quad \alpha = (G + \lambda I_\ell)^{-1} y$$

Les paramètres α_i sont appelés les **variables duales**.

➛ $\alpha = (G + \lambda I_\ell)^{-1} y$ où $G = XX^T$ et $G_{ij} =< x_i, x_j >$.

☞ N.B. : dans le cas **duale**, les coefficients α_i sont des entiers qui représentent le nombre de fois qu'un exemple a été mal classé pendant l'apprentissage.

Addendum : Régularisation et Régression Ridge (suite)

Le point clef de l'équation (2.6) de la **solution Duale** :

• Les infos sont données à partir des instances seulement par un produit interne (matriciel) entre les paires de points d'apprentissage dans la matrice $G = XX^T$.

• De même, l'information sur une nouvelle instance x nécessaire pour la fonction de prédiction est simplement un produit matriciel entre les exemples d'apprentissage (x_i) et la nouvelle instance x.

Notes sur le choix de la méthode :

• Si la dimension n de l'EdC est plus grande que le nombre ℓ d'instances d'apprentissage, on préfère résoudre l'équation duale (2.6) plutôt que l'équation primale (2.5) qui nécessite la matrice $(X'X + \lambda I_n)$ de dimension $n \times n$.

• En termes de complexité : l'évaluation de la fonction prédictive duale est en tous cas plus coûteuse car la solution primale nécessite $O(n)$ opérations, alors que celle de la duale est $O(n\ell)$.

➙ Malgré cela, la solution duale offre d'énormes avantages (cf. données pauvres).

Addendum : Résumé Ridge

• La **Régression d'Arête ou Pénalisée** (*Ridge*) est une variante de la *Régression Linéaire Multiple* dont l'objectif est de de contourner l'obstacle de la co-linéarité entre variables explicatives.

➙ Pour estimer les paramètres du modèle, elle renonce à la seule méthode des Moindre Carrés directe et modifie la matrice $X^T X$ (par un biais) pour rendre à son déterminant une valeur appréciable $\neq 0$ (+baisse de la variance).

• De ce fait, elle introduit un biais sur les estimations des paramètres (alors que les paramètres obtenus par MC simple sont non biaisés).

➙ Ce léger inconv. est plus que compensé par la réduction de la variance des params, et conduit à la réduction de leur Erreur Quadratique Moyenne.

➙ Ainsi, les erreurs de prédiction de la Régression Ridge sont **plus faibles** que celles de la Régression classique en cas de quasi co-linéarité.

• En générale, un estimateur biaisé mais de faible variance peut être plus performant qu'un estimateur sans biais mais de forte variance.

➙La Régression Ridge = un "compromis biais-variance" (ici Perte-Norme).

Addendum : Résumé Ridge (suite)

• **Un intérêt principal** de la *Régression Ridge* Duale est que sa résolution nécessite seulement des produits vectoriels entre les instances d'apprentissage.

• La Régression Ridge permet d'identifier une relation linéaire (la fonction $g(.)$) entre y et x via w.

• Cependant, cette relation n'est pas toujours linéaire.

• Une méthode (voir la suite) est de procéder à une projection des données dans un espace où la relation est linéaire et peut être découverte par la régression Ridge.

➙ La projection aura lieu via un fonction Noyau (Kernel),
➙ L'espace dans lequel la projection a lieu est en général de dimension supérieure mais on verra plus loin que l'on n'a pas besoin de calculer explicitement ces projections :
  ➛ Elles seront calculées par des produits vectoriels.

☞ Dans cette introduction générale à la notation matricielle, nous avons utilisé une fonction de noyau *identité* ($f(x) = x$).

Addendum : Calcul de la Régression Ridge

Rappel :

$$min_w \mathcal{L}_\lambda(w, S) = min_w \left[\lambda \parallel w \parallel^2 + \sum_{i=1}^{\ell} (y_i - g(x_i))^2 \right]$$

$$\text{où } ||w||^2 = \sum_n w_i^2, \quad la\ dimension\ n = nb\ attributs,$$

- La Régression Ridge impose davantage de contraintes sur les w_i du modèle Linéaire.
  ➙ Au lieu de juste optimiser la somme des carrés résiduels, on a une pénalité sur les w_i via le terme de la pénalité $\lambda\ ||w||^2$ (λ constante prédéfinie).
- Si les w_i ont de grandes valeurs, la fonction d'optimisation est pénalisée.
  ➙ Des w_i plus larges donneront une meilleure somme des carrés des résiduels mais augmentent le second terme.
- On préfère choisir des w_i plus petits (proche de zéro) pour minimiser la pénalité.

Addendum : Calcul de la Régression Ridge (suite)

• Du point de vue d'optimisation, la pénalité est équivalente à une contrainte sur les w_i.

➟ Cette fonction est encore la somme des carrés des résiduels mais maintenant on contraint la norme $||w||$ d'être plus petite qu'une constante s.

• Il y a une correspondance entre λ et s : plus λ est grand, plus on aura les w_i proches de zéro.

➟ Dans le cas extrême de $\lambda = 0$, on fera simplement une Régression Linéaire.

➟ Et si λ approche ∞, on aura les w_i nuls : on approche la réponse par une constante.

• En Régression Ridge, le but n'est pas de contraindre la complexité du modèle, son objectif est de régler le problème numérique quand $X^T X$ est quasi une matrice *singulière*.

• Dans la Régression Ridge, la matrice $X^T X$ est augmentée d'une petite matrice scalaire.

Addendum : Calcul de la Régression Ridge (suite)

Formulation : minimiser une somme des carrés résiduels pénalisée

$$\hat{w}^{ridge} = argmin_w \left\{ \sum_{i=1}^{\ell} (y_i - w_0 - \sum_{j=1}^{n} x_{ij}\, w_j)^2 + \lambda \sum_{j=1}^{n} w_j^2 \right\}$$

n est le nbr des attributs, w_0 vient du modèle linéaire, ℓ est la taille de la BD.

Ou de manière équivalente (problème de contraintes) :

$$\hat{w}^{ridge} = argmin_w \sum_{i=1}^{\ell} \left(y_i - w_0 - \sum_{j=1}^{n} x_{ij}\, w_j \right)^2 \quad \text{s.t.} \quad \sum_{j=1}^{n} w_j^2 \leq s$$

➙ λ et s contrôlent la complexité du modèle.

- La solution à la Régression Ridge :

  somme des carrés résiduels (RSS) $= (y - Xw)^T (y - Xw) + \lambda w^T w$

  et (la forme **primale**) $\hat{w}^{ridge} = (X^T X + \lambda I)^{-1} X^T y$

- Des solutions existent lorsque $X^T X$ est singulière (**a des valeurs propres nulles**)
- Si $X^T X$ est mal conditionnée (quasi singulière), la solution est plus robuste.

Addendum : Calcul de la Régression Ridge (suite)

Calcul et Interprétation géométrique :

- Soit les entrées (instances) centrées
- La réponse finale du système devra être (voir les Notes) :

$$\hat{y} = X\hat{w}^{ridge}$$
$$= X(X^TX + \lambda I)^{-1}X^Ty \qquad \text{équation Primale}$$
$$= UD(D^2 + \lambda I)^{-1}DU^Ty \qquad \text{où } X = UD, X^T = D^TU^T \text{ et } D^T = D, UU^T = I$$
$$X^TX = D^TU^TUD = D^TD = D^2$$
$$= \sum_{j=1}^{k} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T y$$

Où les u_j sont les les composantes principales de X normalisées.

☞ **Notes importantes :**

Dans ce calcul, la *Régression Ridge* transforme (rétrécie) les coordonnées selon les bases orthonormées obtenues par l'analyse des composantes principales,

Au lieu d'utiliser la matrice X comme les variables explicatives, on utilise les variables transformées $(Xv_1, Xv_2, ..., Xv_p)$ comme prédicteurs (cf SVD),

Addendum : Calcul de la Régression Ridge (suite)

La matrice d'entrée devient $\tilde{X} = UD$ après la transformation précédente (au lieu de $X = UDV^T$) issues de l'analyse des composantes principales (ou SVD où $UU^T = I$ et $VV^T = I$)

- D'où (applicable à toute nouvelle instance en entrée) :

$$\hat{w}_j^{ridge} = \frac{d_j}{d_j^2 + \lambda} u_j^T y$$

$$Var(\hat{w}_j) = \frac{\sigma^2}{d_j^2}$$

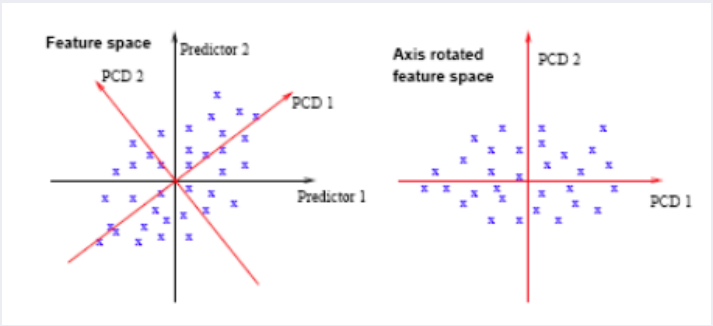
Où σ^2 est la variance de l'erreur dans le modèle linéaire.

- Le facteur de rétrécissement (suivant l'ACP) : $\frac{d_j^2}{d_j^2 + \lambda}$

➙ Plus λ est grand, plus la projection est rétrécie dans la direction des u_j.

Addendum : Calcul de la Régression Ridge (suite)

Exemple de projection suivant les axes ACP (pour $n = 2$)



N.B. : une variante de Ridge Régression est l'estimateur **Lasso**
(de manière équivalente : problème de contraintes) où :

$$\hat{w}^{lasso} = argmin_w \sum_{i=1}^{\ell} \left(y_i - w_0 - \sum_{j=1}^{n} x_{ij} w_j \right)^2 \quad \text{s.t.} \quad \sum_{j=1}^{n} |w_j| \leq s$$

➙ λ et s contrôlent la complexité du modèle.

➙ On note la contrainte $\sum_{j=1}^{n} |w_j| \leq s$ (Lasso) vs. $\sum_{j=1}^{n} w_j^2 \leq s$

- Comparaison ($n = 2$) : *Lasso* : $|w_1| + |w_2| \leq s$ avec *Ridge* $w_1^2 + w_2^2 \leq s^2$

Classification Linéaire utilisant un Perceptron

• La Régression Logistique produit des estimations de probabilités en maximisant les probabilités sur l'ensemble d'instances,

➙ Estimation des *meilleures* probabilités pour les instances.

• Mais on n'a pas toujours besoin des probabilités si l'on veut seulement définir la classe d'une instance.

➙ Des valeurs (scores) comparables suffisent (cf. régression multi-classes).

• **Idée** : trouver (simplement) les hyperplans qui séparent bien les instances

• **Hypothèses** (bi-classes) :

◦ Seulement 2 classes → un hyperplan

◦ Les instances pourraient être proprement séparées par un hyperplan :

➛ c-à-d. : elles sont *linéairement séparables*

• Sous ces conditions, il existe une méthode plus simple :

La règle d'apprentissage de Perceptron (≈ de *Neurone*)

Classification Linéaire utilisant un Perceptron (suite)

La méthode (cas bi-classe) :

- On reprend l'équation de l'hyperplan :

  $w_0\, a_0 + w_1\, a_1 + w_2\, a_2 + ... + w_k\, a_k$

  ➛ w_i : pondération, a_i : valeurs d'attribut

- Une instance est caractérisée par $a_1, a_2, ..., a_k$

  ➛ $a_0 = 1$ (toujours)

  ➛ Veut aussi dire : il n'y a pas d'autre constante dans la somme.

- Si cette $somme > 0$ (notion de *signe*), on prédit la 1ère classe, sinon la 2e.

  ➛ $classe = sign(...)$

- On veut trouver les pondérations pour pouvoir classer les instances par l'hyperplan.

- L'algorithme de recherche de l'hyperplan : ../..

Classification Linéaire utilisant un Perceptron (suite)

- $\forall i, w_i = 0$
- Tant que toutes les instances ne sont pas bien classées (*prédiction vs. modèle*)
  - $\forall I$ dans l'ensemble d'apprentissage
    - Classifier I (avec les poids actuels)
    - Si I est mal classée par le *perceptron*
    - Alors
      - Si I appartient à la 1ère classe Alors ajouter l'instance au vecteur de poids
      - Sinon le soustraire du vecteur de poids

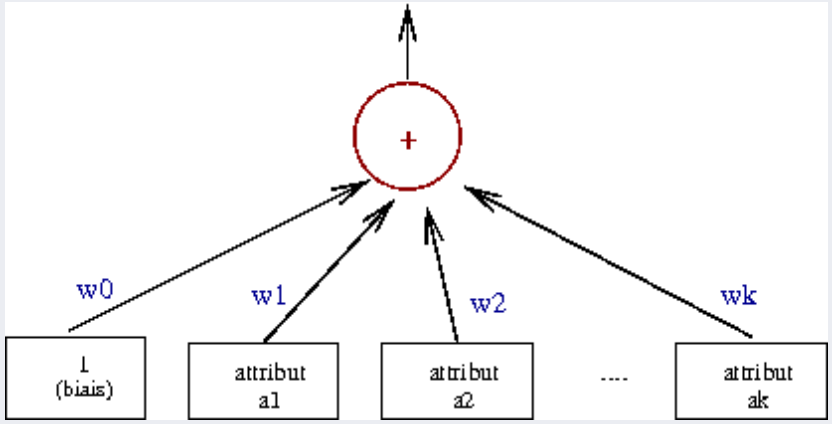


FIGURE 10: Représentation d'un réseau de neurones et Algorithme de recherche de l'hyperplan

- Cet algorithme trouve l'hyperplan s'il existe
  - ➙ c-à-d. : si les données sont linéairement séparables.
- La figure ci-dessus : *la couche d'entrée* a un noeud par attribut (+ le biais)
  - ➙ La sortie = un seul noeud qui somme les valeurs $w_i\, a_i$
  - ➙ Si cette somme > 0 ➙ 1e classe, sinon la 2e classe

Classification Linéaire utilisant un Perceptron (suite)

• Par les modifications des poids, les instances se rapprochent de l'hyperplan (le franchissent si nécessaire).

➙ On ajoute ou soustrait des valeurs d'attributs au vecteur de poids

Un exemple *d'ajout d'une instance (cf. algo précédent) :*

Soit l'instance a affectée à la 1ère classe (on ajoute a_i aux w_i) :

On a $(w_0 + a_0)a_0 + (w_1 + a_1)a_1 + (w_2 + a_2)a_2 + ... + (w_k + a_k)a_k$

➙ *Ce qui veut dire que la sortie pour a est augmentée par :*

$a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 + ... + a_k \times a_k$ *(un nbr. toujours* > 0*)*

➙ *l'hyperplan a bougé pour classer a positif (la 1e classe)*

• L'algorithme précédent converge si les instances sont séparables

• Pour la terminaison : mettre une limite au nombre d'itérations

• L'hyperplan résultant est appelé perceptron (ancêtre des Réseaux de Neurones)

Classification Linéaire avec Winnow

- **Problème** de l'algorithme précédent (voir plus loin) :
  ➙ interférence avec les modifications (des itérations) précédentes
- Une solution : on modifie la version basique précédente.
- Cas des attributs binaires (0 , 1) : la méthode **Winnow**.

```
- Tant que des instances mal classées existent
  - ∀I dans l'ensemble d'apprentissage
    + Classifier I par le vecteur des poids
    + Si I est mal classée (prédiction)
        - Si I appartient à la 1ère classe
          - Pour ∀ attribut a_i = 1
                multiplier w_i par la constante α
                (laisser w_i inchangé si a_i = 0)
        - Sinon   % I n'appartient pas à la 1ère classe
          - Pour tout a_i = 1,
                diviser w_i par α
                (laisser w_i inchangé si a_i = 0)
```

Algorithme Winnow : versions basique

Ici, la constante α est spécifiée par l'utilisateur

Classification Linéaire avec Winnow (suite)

- Les attributs a_i sont binaires (comparaison à 0/1 dans l'algorithme) :
  - ➛ Si $a_i = 0$, cet attribut ne participe pas à la décision
  - ➛ Si $a_i = 1$, on multiplie par α si a_i fait la bonne décision, sinon par $1/\alpha$

- Autre différence par rapport à l'algorithme *Perceptron* (précédent) :

  ➛ L'utilisateur définit un seuil θ de telle sorte qu'une instance sera classée en 1ère classe si :

$$w_0\, a_0 + w_1\, a_1 + w_2\, a_2 + \ldots + w_k\, a_k > \theta$$

- La constante $\alpha > 1$ et les w_i sont initialisés au départ à une certaine cste.

Inconvénient de la version basique de Winnow :

ne traite pas les poids négatifs ➛ versions balancée

Classification Linéaire avec Winnow (suite)

L'algorithme Winnow : versions balancée

➙ On maintient une vecteur de poids w^+ pour la 1e classe, w^- pour la seconde.

- Tant que des instances mal classées existent
  - $\forall I$ dans l'ensemble d'apprentissage
    - + Classifier I par le vecteur des poids
    - + Si I est mal classée (prédiction)
      - Si I appartient à la 1ère classe
        - $\forall$ attribut $a_i = 1$
          - multiplier w_i^+ par α
          - diviser w_i^- par α
          - (laisser w_i^+ / w_i^- inchangé si $a_i = 0$)
      - Sinon
        - Pour toute $a_i = 1$
          - multiplier w_i^- par α
          - diviser w_i^+ par α
          - (laisser w_i^+ / w_i^- inchangé si $a_i = 0$)

→ w^+ : vecteur de poids pour la 1e classe, w^- : pour l'autre.

→ La constante α et le seuil θ spécifiés par l'utilisateur

Classification Linéaire avec Winnow (suite)

- Dans la version **balancée** :
  - ➙ On maintient **deux vecteurs de poids** : un par classe
  - ➙ Une instance est classée en classe 1 si

$$(w_0^+ - w_0^-)a_0 + (w_1^+ - w_1^-)a_1 + (w_2^+ - w_2^-)a_2 + ... + (w_k^+ - w_k^-)a_k > \theta$$

- Winnow est très efficace sur les attributs effectivement discriminants

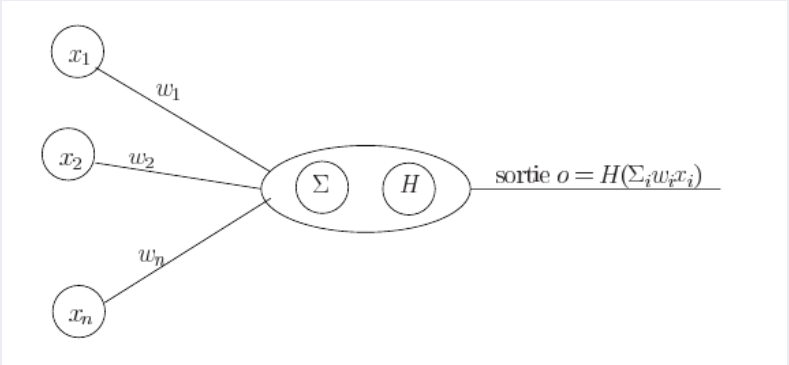
- Il est considéré comme "bon" sur une BD avec beaucoup d'attributs binaires dont la plupart n'est pas discriminant.

- Autre intérêt de Perceptron et Winnow :
  méthodes utilisables à la volée (online)
  - ➙ Quand les données d'apprentissage arrivent au fur et à mesure

N.B. : on a vu les versions basiques des *réseaux de neurones (RN)*.
➙ Suit : RNs avec *logit* (plutôt qu'une version linéaire). ../..

Réseaux de neurones

- Comme pour le neurone biologique dont la sortie est activée lorsque l'activité en entrée dépasse un certain seuil :
  - La sortie d'un neurone artificiel est activée selon une fonction H appliquée à ses entrées.
  - La sortie sera =0 (ou parfois -1) lorsque l'activité en entrée n'est pas suffisante.



- Un exemple simple pour H :

$$H(x) = 1 \ si \ x > 0$$
$$H(x) = 0 \ si \ x = 0.$$

Réseaux de neurones (suite)

• Pour pouvoir utiliser les méthodes de gradient, on préfère utiliser une fonction H dérivable (n'était pas le cas ci-dessus).

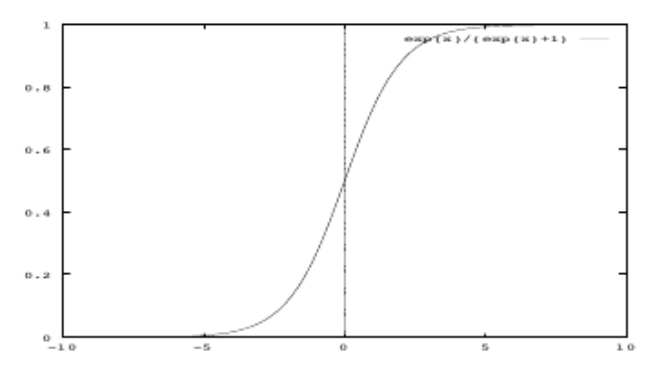
• Une fonction souvent employée est la **sigmoïde** (cf. *logit*)

$$\sigma(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$$

➙ $\sigma(x)$ à valeur dans $[0,1]$ passera de 0 à 1 lorsque l'entrée est "suffisante".

➙ Elle est **continue et dérivale**.

• L'entrée est ici une somme pondérée (vu plus haut).



☞ On préfère souvent une fonction symétrique $\in [-1,1]$:

➙ e.g. $TanH(x) = \frac{2}{1+e^{-2x}} - 1$ dont la dérivée est $1 - [TanH(x)]^2$

Réseaux de neurones (suite)

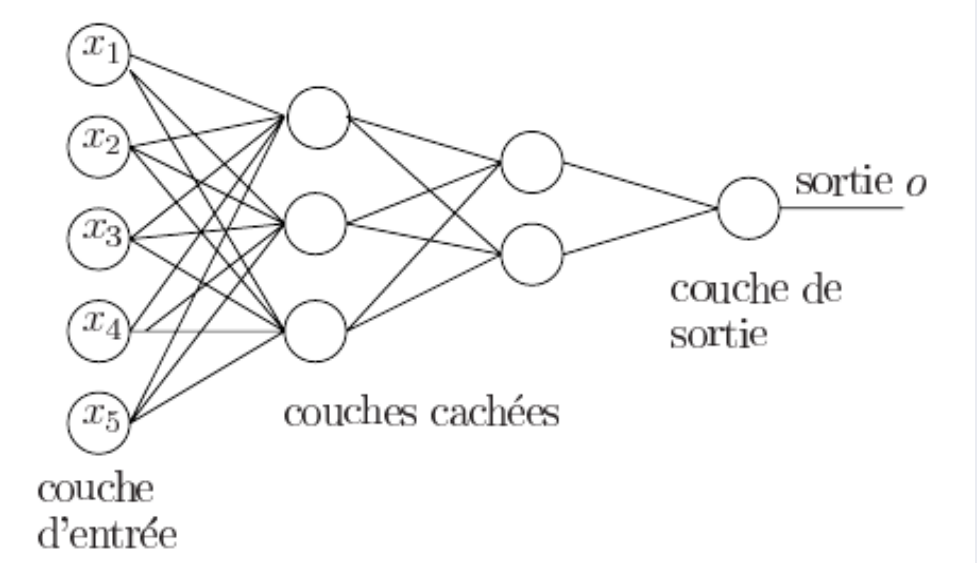
Perceptron multi couches :

• Les neurones élémentaires ont la même structure que ci-dessus.

• Dans le cas simple, il n'y a pas de retour vers une couche précédente.

• Chaque couche peut contenir de 1 à plusieurs neurones, y compris la couche de sortie.

➙ C'est le problème à traiter qui définira le nombre de neurones en entrée et en sortie.



N.B. : les RNs existent depuis longtemps mais c'est dans les années 80s que l'algorithme de *rétro-propagation du gradient* [Rumelhart] (le retour signalé ci-dessus) a permis leur développement.

Réseaux de neurones (suite)

Exemple d'apprentissage :

• On se place dans le cas de classification binaire (2 classes) avec un seul neurone en sortie à valeur dans $[0, 1]$.

• L'ensemble d'apprentissage

$$S = \{(x_1, y_1), (x_2, y_2), ..., (x_\ell, y_\ell)\} \text{ avec } y_i \in [0, 1] \text{ et } x_i \in \mathbb{R}^n$$

➙ On note $O(x_i)$ la sortie du réseau pour l'instance x_i.

• L'apprentissage doit minimiser l'erreur (MC comme ci-dessus) avec :

$$\mathbb{E}_i = 1/2 \sum_{x_i, O(x_i) \in S} (y_i - O(x_i))^2$$

L'algorithme de principe :

- Initialiser aléatoirement le vecteur de pondérations w_i (cf. *régression*)
- Répéter : présenter une instance $x_i, i = 1..\ell$
  - . calculer la sortie $O(x_i)$ et l'écart entre $O(x_i)$ et y_i
  - . Mise à jour des pondérations (voir ci-dessous)

Jusqu'à *condition d'arrêt*.

Réseaux de neurones (suite)

- A définir sur ces itérations (pour cet algorithme) :
  - Une fonction d'activation
  - Une condition d'arrêt
  - $nbInstances \in 1..\ell$: le nombre d'instances présentées en entrée avant la MAJ des pondérations ($\neq$ taille de la BD)
  - *Taux d'apprentissage* ε
  - La *tolérance* : pour l'arrêt des itérations (= erreur cible).
- Les pondérations sont initialisées de manière aléatoire
  $\rightarrow$ RN différent selon les initialisations.
- Si $nbInstances = 1$ (avant MAJ des w_i) alors MAJ pour chaque instance
  ➙ Convergence rapide
  ➙ Mais risque de minimum local
- Si $nbInstances = \ell$ (la totalité de la BD.), alors convergence plus lente.
  ➙ Ce paramètre s'apprend.

Réseaux de neurones (suite)

- Les pondérations peuvent être MAJ pour différentes valeurs de ε (noté aussi par η dans la littérature).
  - Dans la pratique, on commence par une grande valeur que l'on diminue au fur et à mesure des itérations
  - Diminution : dans la pratique, évite les oscillations et converge mieux.
  - $\varepsilon = 0.1$ *ou* 0.2 sont des exemples de grandes valeurs utilisées.

- Le paramètre *d'inertie* α pour lisser les modifications des pondérations
  - Et pour se rappeler de la modification précédente
  - Sans quoi on risque d'augmenter puis diminuer alternativement (cf. Perceptron).

- La *tolérance* : un autre paramètre de condition d'arrêt des itérations
  - Définit l'erreur cible à atteindre.
  - Est définie en fonction du mode de calcul de l'erreur.
  - Défini parfois selon le nombre d'instances bien classées (ou bien estimés = avec un écart inférieur à un certain seuil).

Réseaux de neurones (suite)

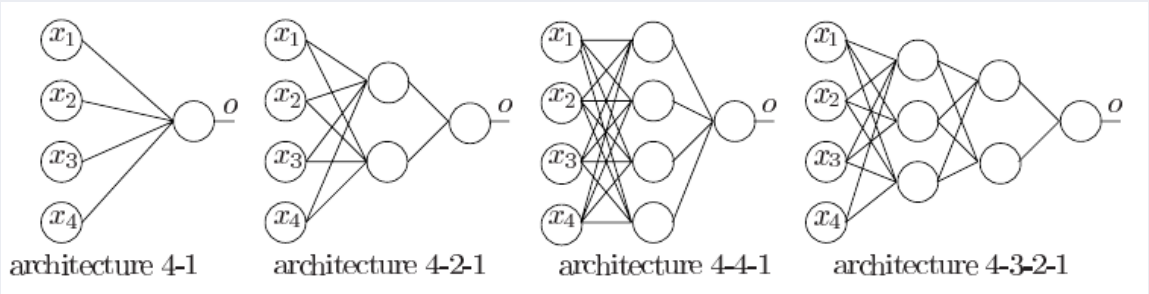
Codages divers :

- Les entrées de RN sont des réels que l'on normalise dans l'intervalle $[0, 1]$.
- Les données binaires inchangées
- Les énumérés à k valeurs : k booléens (avec un seul vrai)
  Ou codés sur $log(k)$ entrées ($log(k)$ bits pour représenter k valeurs).
  Ou sur k intervalles de réels (e.g. 5 valeurs : 0.0, 0.25, 0.5, 0.75, 1.0)
- Les sorties peuvent être normalisées (e.g. 2 bits pour 4 classes).

Réseaux de neurones (suite)

Décision de l'architecture (avec traitement par *fenêtre*)

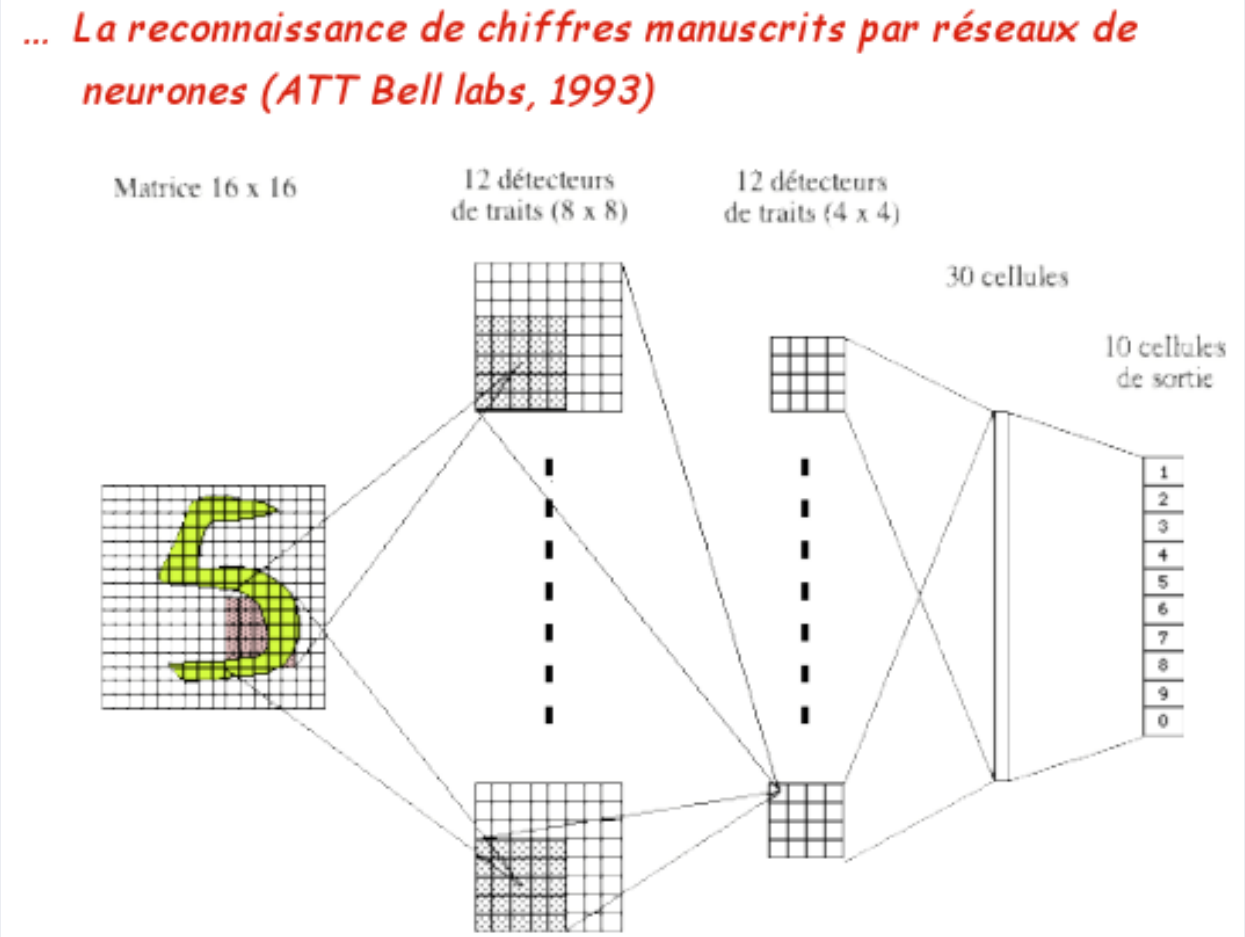
- Vient après la détermination des entrées et des sorties (et leur normalisation)
- On a recours à des couches cachées (peaufinées par Expériences / Essais)
  - ➙ Augmente la capacité d'induction mais complexifie le RN.
  - ➙ Compromis à trouver, *overfitting* (*le par-coeur*) à éviter.
- Exemple de RN pour 4 entrées et une sortie :



- Validation : si assez de données (ensemble S), S découpé en trois L, T et V :
  - ➙ **L** pour apprendre (plusieurs RNs)
  - ➙ **T** pour les tester et choisir le meilleur
  - ➙ **V** pour l'estimation des performances réels (du RN retenu).

Réseaux de neurones (suite)

Exemple d'architecture :

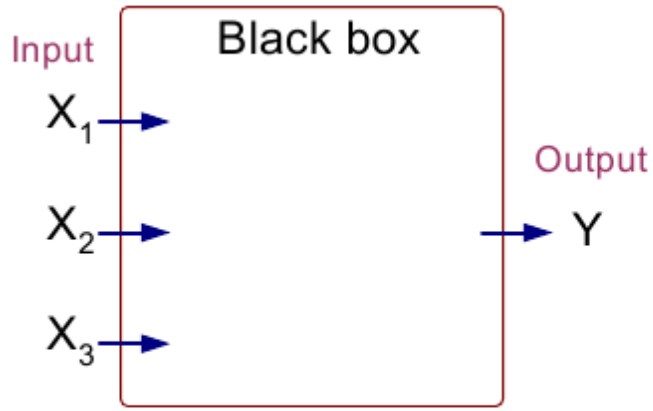


Addendum : RNs illustrés

- Des *Perceptrons* aux Réseaux de neurones (rétro propagation).
- Un exemple simple des réseaux de neurones

Addendum : RNs illustrés (suite)

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

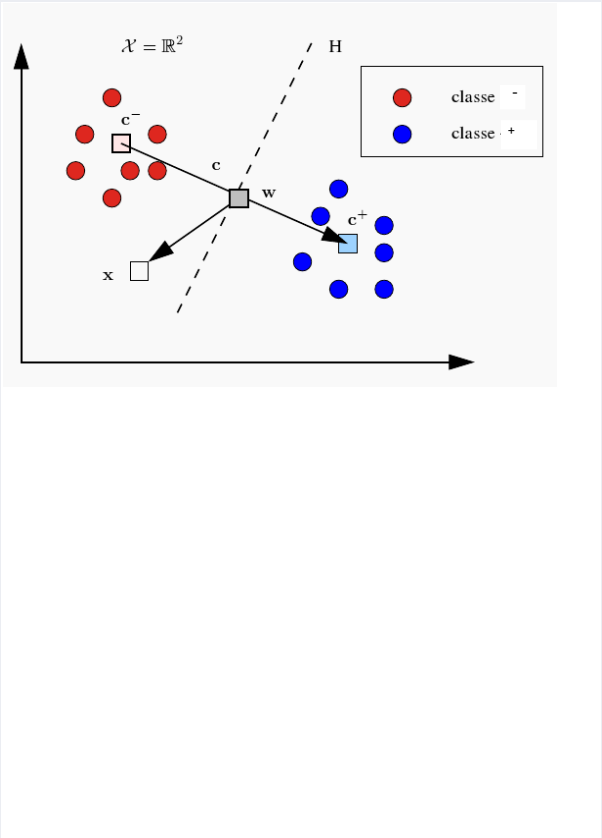


La sortie Y=1 si au moins deux des trois entrées = 1

Addendum : Généralisation de la régression

• La régression (et les RNs) s'inscrivent dans un cadre général de recherche d'une frontière (de *Rocchio*) :

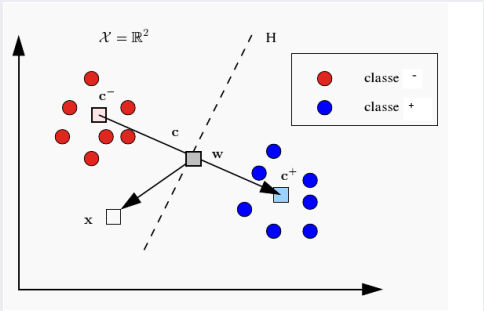
Exemple : construction d'un classifieur simple



Cas bi-classes :

- $c^+ = \frac{1}{n^+} \sum_{i:y_i=+1} x_i$ la moyenne c^+
- $c^- = \frac{1}{n^-} \sum_{i:y_i=-1} x_i$ idem c^-
- $c = \frac{1}{2}(c^+ + c^-)$ c : la moyenne des
- $w = (c^+ - c^-)$ w : écart relati

Addendum : Généralisation de la régression (suite)



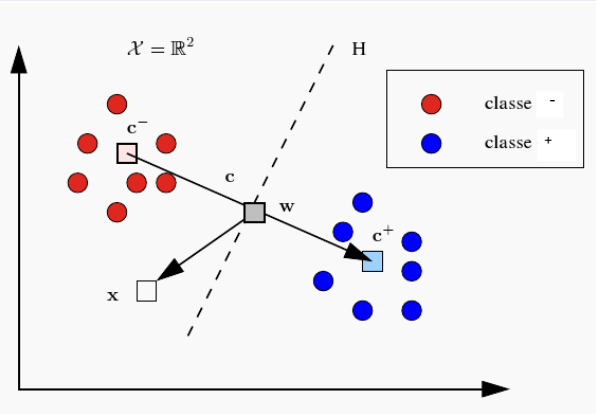
Cas bi-classes :

- $c^+ = \frac{1}{n^+} \sum_{i:y_i=+1} x_i$
- $c^- = \frac{1}{n^-} \sum_{i:y_i=-1} x_i$
- $c = \frac{1}{2}(c^+ + c^-)$
- $w = (c^+ - c^-)$

On évalue $h(x) = \langle w, x - c \rangle$ où $sign(h(x))$ donne la classe de x

$$h(x) = \langle (x - \frac{1}{2}(c^+ + c^-)), (c^+ - c^-) \rangle = \langle x, c^+ \rangle - \langle x, c^- \rangle + b$$

Addendum : Généralisation de la régression (suite)



- $c^+ = \frac{1}{n^+} \sum_{i:y_i=+1} x_i$
- $c^- = \frac{1}{n^-} \sum_{i:y_i=-1} x_i$
- $c = \frac{1}{2}(c^+ + c^-)$
- $w = (c^+ - c^-)$

Addendum : Généralisation de la régression (suite)

- On obtient **la forme générale** d'une classification en terme de $\{x_i, y_i\}$:

$$\boxed{h(x) = \sum_{i=1..n} \alpha_i \langle x_i, x \rangle + b,} \qquad b \in \mathbb{R}$$

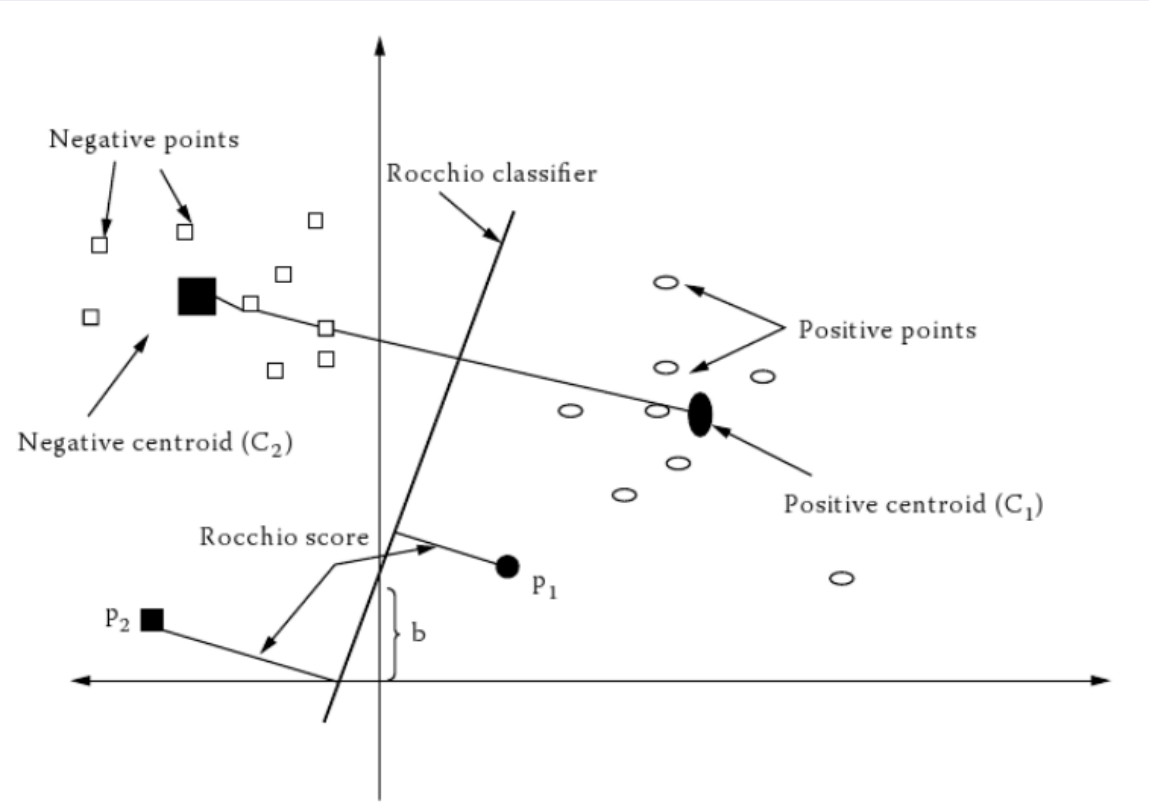
avec (ici) $\alpha_{i:y_i=+1} = 1/n^+$ et $\alpha_{i:y_i=-1} = 1/n^-$

et $b = \frac{1}{2}(\| c^- \|^2 - \| c^+ \|)$ (voir un exemple de b page suivante)

- Un exemple pour b peut être la pondération constante w_0.

Addendum : Généralisation de la régression (suite)

Une illustration de Rocchio (simplifiée) :



➙ P_1 et P_2 sont des instances à classer.

➙ On remarque l'illustration du seuil b.

Addendum : Classification et erreur

D'une manière générale :

le problème de la classification peut être définie par :

Données :

- Un ensemble d'apprentissage $S_n = \{(x_i, y_i)\}_{i=1}^n \in (\mathcal{X} \times \mathcal{Y})^n$
  $y_i \in \mathcal{Y}$ est la classe connue de l'instance x_i
- Un espace d'hypothèses $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$
- Une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ qui mesure l'écart et son coût entre $h(x)$ *et* y

Addendum : Classification et erreur (suite)

But : Trouver l'hypothèse de prédiction h^* qui minimise l'erreur réelle (**risque réel**) :

$$R(h) = \mathbb{E}[\ell(y, h(x))] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, h(x))\, dPr(x, y)$$

☞ N.B. : pour une notation homogène avec l'espace des hypothèses $\mathcal{H}$, nous utilisons la fonction $h(x)$ à la place de $g(x)$ utilisée plus haut.

Erreur théorique de la classification :
Principe du MRE (minimisation du risque empirique=erreur d'apprentissage) :

- On mesure le risque empirique $R_n(h) = \dfrac{1}{n}\sum_{i=1}^{n} \ell(y_i, h(x_i))$
- On trouve l'hypothèse $h \in \mathcal{H}$ qui minimise ce risque : $h_n^* = \underset{h\in\mathcal{H}}{\text{argmin}}\ R_n(h)$
  - ➛ MRE choisit une fonction qui montre un désaccord minimum avec les données d'apprentissage (cf. MC).

Addendum : Classification et erreur (suite)

• N.B. : la notion du risque et de sa minimisation est un élément important de l'apprentissage.

• La convergence n'est pas toujours vraie pour n'importe quelle hypothèse.

Rappel :

• Une hypothèse correspond à la frontière de décision de **Rocchio**,

• De même, aux différents éléments d'un RN (hors encodage des entrées / sorties).

• **Une remarque générale :**

Les notations / calculs par les produits vectoriels permettent une harmonisation des notations (introduite dans le cas de la régression).

Addendum : Classification et erreur (suite)

Un outil (pour borner l'erreur) :

- Un résultat appelé la "convergence uniforme" permet de borner l'erreur :
  ➙ Il a été démontré qu'avec une probabilité de $1-\delta$:

$$\forall h \in \mathcal{H}, \quad |R_n(h) - R(h)| \leq \sqrt{\frac{log|\mathcal{H}| + log\frac{2}{\delta}}{2n}}$$

où $|R_n(h) - R(h)|$ = val_abs(...) et $|\mathcal{H}|$ = nbr. d'hypothèses possibles

Exemple : avec $|\mathcal{H}|$=100000 (hypothèses), $\delta = 0.01$ et $n = 10000$ (instances) :

$$\sqrt{\frac{log|\mathcal{H}| + log\frac{2}{\delta}}{2n}} = 0,028$$

➛ C-à-d. : avec une probabilité de 99%, $R_n(h)$ ne s'écartera en moyenne pas de plus de 2,8% de l'hypothèse optimale $R(h), \forall h \in \mathcal{H}$

Méthode de Noyau revisitée

Vers les méthodes générales à base de noyau.

Chercher une solution :

Que faire si l'ensemble d'apprentissage n'est pas linéairement séparable ?

Réponses :

- Classification non linéaire (très difficile, données numériques)

ou

- On trouve une transformation dans un espace où l'ensemble devient séparable.

La transformation est appelée (réalisée par) une fonction de noyau (*kernel*).

Méthode de Noyau revisitée (suite)

Méthode noyau (kernel)

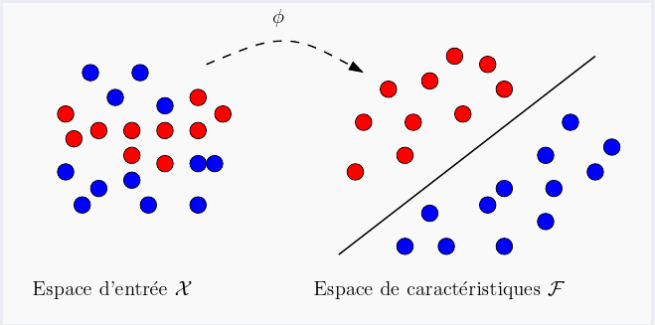
Soit un ensemble de données non linéairement séparables dans l'espace H

$$S_n : \{(x_1, y_1), ..., (x_n, y_n)\} \qquad y_i : \text{classe de } x_i$$

- Choisir une transformation non linéaire ϕ

$$\phi : \chi \to \mathcal{F}$$
$$x \to \phi(x)$$

où $\mathcal{F}$ est un espace vectoriel appelé *espace de caractéristiques* (Feature space).

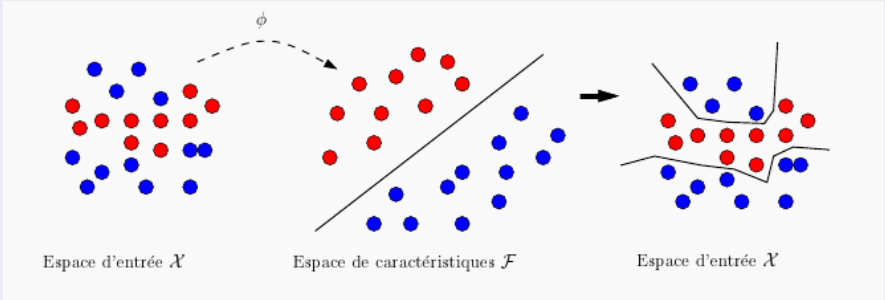


- Trouver un classifieur linéaire (i.e. un hyperplan séparateur) dans $\mathcal{F}$ pour classifier $\{(\phi(x_1), y_1), ..., (\phi(x_n), y_n)\}$

➙ Procéder à une classification linéaire dans l'espace de caractéristiques.

- Implanter ensuite ce classifieur linéaire dans H (espace des hypothèses) :

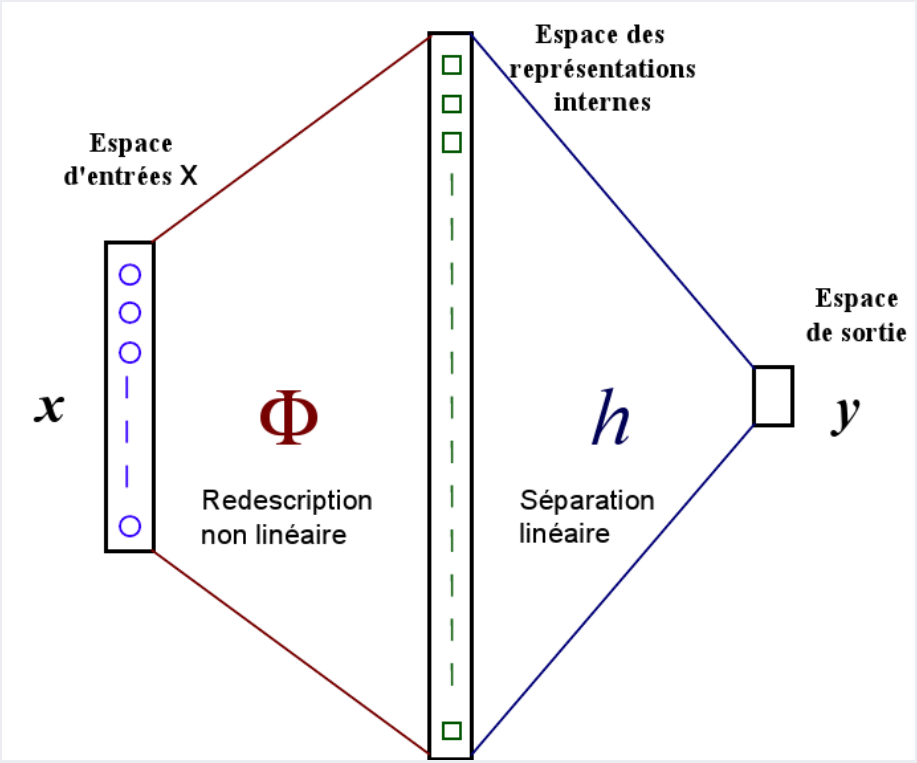
$$h(x) = \sum_{i=1,...,n} \alpha_i \langle \phi(x_i), \phi(x) \rangle + b$$



Transformation-Projection

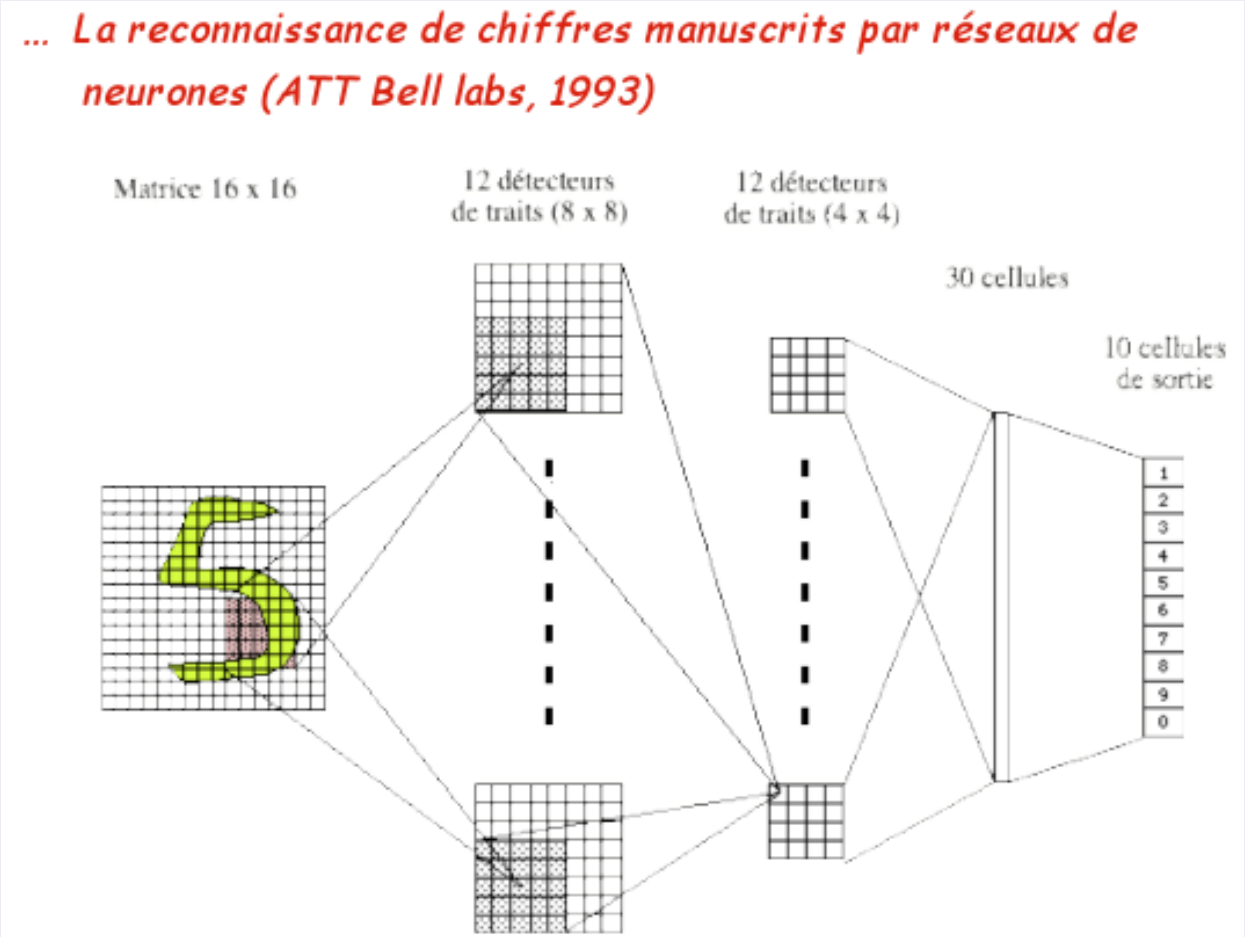
Idée générale de la projection

• Modification de la dimension (augmentation) des instances non-linéairement séparables pour trouver un espace (des caractéristiques) où les (projections des) instances sont linéairement séparable :



Transformation-Projection (suite)

Exemple (ici les couches internes d'un RN représentent la projection) :



Transformation-Projection (suite)

Remarques :

- L'astuce du noyau s'applique s'il y a une fonction k dans $\mathbb{R}$:

$$k : \chi \times \chi \to \mathbb{R} \text{ telle que } k(u, v) = \langle \phi(u), \phi(v) \rangle_{\mathcal{F}}$$

  ➙ Dans ce cas, toutes les occurrences de $\langle \phi(u), \phi(v) \rangle$ sont remplacées par $k(x, x_i)$.

  ➙ $k(.,.)$ peut être vu comme une fonction de "**similarité**".

- Le classifieur obtenu sera (p. ex.) : $f(x) = signe\left(\sum_{i=1,\ldots,n} \alpha_i.y_i.k(x_i, x) + b\right)$

N.B. : un des intérêts de l'utilisation d'un noyau est d'éviter les calculs effectifs des projections et d'utiliser le produit matriciel dans l'espace même des données (entre les instances x_i et x_j).

➙ Éventuellement (mais rarement), on peut calculer le produit sur les projections.

Transformation-Projection (suite)

L'intérêt d'un noyau : l'idée est de ne pas calculer explicitement la transformation ϕ mais de se baser directement sur le noyau.

Un exemple : une transformation linéaire $\phi(x) = Ax$.

➙ Dans ce cas, le noyau peut s'écrire (la matrice $A^T A$ est définie positive) :

$$k(x, y) =< \phi(x), \phi(y) >= (Ax)^T Ay = x^T A^T A\ y$$

Caractéristique du noyau :

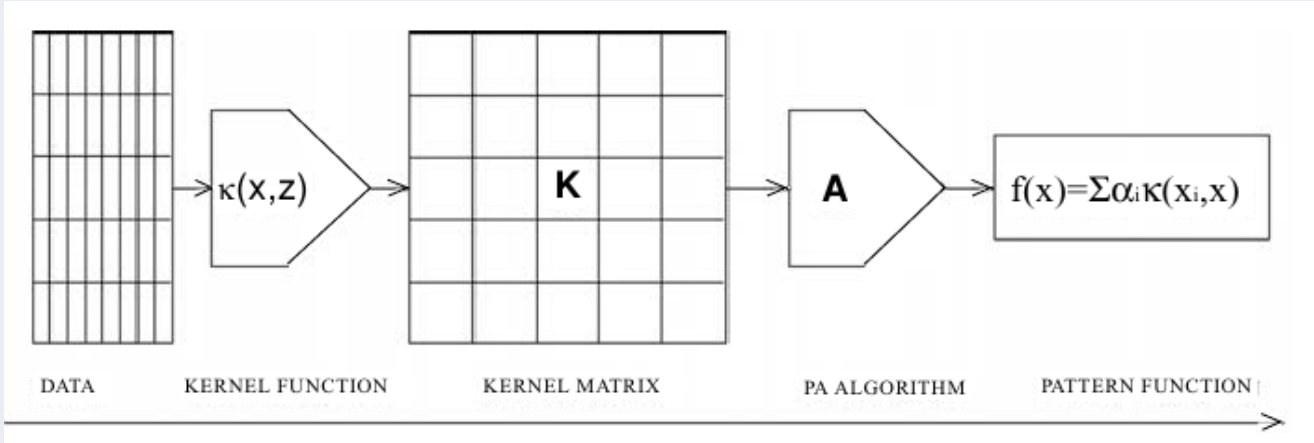
- Les noyaux doivent vérifier certaines propriétés pour être valides.
- Les conditions (pour une fonction noyau $k(x, y)$) :

Pour qu'une fonction $K(x, y)$ de deux vecteurs x, y définis sur un espace Euclidien soit un noyau, k doit définir un **produit scalaire** dans un certain espace vectoriel des caractéristiques (cf. Espace vect. Hilbert et prod. vectoriel).

➙ Autrement dit, s'assurer qu'il existe un espace $\mathcal{F}$ et une transformation $\phi : \chi \to \mathcal{F}$ tel que $k(u, v) = \langle \phi(u), \phi(v) \rangle_{\mathcal{F}}$

Matrice de Kernel

- La figure ci-dessous montre les étapes de réalisation d'un tel processus.



- Les données sont traitées par le noyau créant une matrice de noyau.

Cette matrice est à son tour traitée par l'algorithme d'apprentissage (PA : Pattern Analysis) pour créer une *fonction de motif* (*pattern function*).

- Cette fonction sera utilisée pour traiter les instance nouvelles.

Matrice de Kernel (suite)

K	1	2	⋯	ℓ
1	$\kappa(\mathbf{x}_1, \mathbf{x}_1)$	$\kappa(\mathbf{x}_1, \mathbf{x}_2)$	⋯	$\kappa(\mathbf{x}_1, \mathbf{x}_\ell)$
2	$\kappa(\mathbf{x}_2, \mathbf{x}_1)$	$\kappa(\mathbf{x}_2, \mathbf{x}_2)$	⋯	$\kappa(\mathbf{x}_2, \mathbf{x}_\ell)$
⋮	⋮	⋮	⋱	⋮
ℓ	$\kappa(\mathbf{x}_\ell, \mathbf{x}_1)$	$\kappa(\mathbf{x}_\ell, \mathbf{x}_2)$	⋯	$\kappa(\mathbf{x}_\ell, \mathbf{x}_\ell)$

Matrice de Gram :

• Pour un ensemble de vecteurs (les instances) $S = \{x_1, ..., x_\ell\}$, la matrice Gram G est une matrice $\ell \times \ell$ avec les entrées $G_{ij} =< x_i, x_j >$.

• Si on utilise une fonction k pour évaluer les produits vectoriels dans un espace de caractéristiques avec une *application* de caractéristiques ϕ, on aura :

$$G_{ij} =< \phi(x_i), \phi(x_j) >= k(x_i, x_j).$$

• Elle contient toutes les informations pour calculer la distance entre toutes paires d'instances.

• Cette matrice joue un rôle important dans la représentation *Duale* de certains algorithmes d'apprentissage.

➙ Elle est symétrique puisque $G_{ij} = Gji$ et donc $G^T = G$.

Un exemple : Perceptron

Application du principe de classification linéaire générale aux Perceptrons.

• **Rappel** : pour chaque exemple x_i mal classé, le vecteur de poids courant w est mis à jour par ajout ou soustraction de x_i.

• Le vecteur w final s'écrit en terme d'exemples d'apprentissage $\{(x_i, y_i)\}$:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i,\ \alpha_i \geq 0$$

appelé la **représentation duale** de l'hyperplan w.

☞ Remarque : dans le cas **dual** (voir section *Introduction aux Noyaux*), les coefficients α_i sont des entiers qui représentent le nombre de fois qu'un exemple a été **mal classés** pendant l'apprentissage.

• La fonction linéaire discriminante f s'exprime alors par :

$$f(x) = signe(\langle w, x\rangle + b) = signe(\langle \sum_{i=1}^{n} \alpha_i y_i x_i, x\rangle + b) = signe(\sum_{i=1}^{n} \alpha_i y_i \langle x_i, x\rangle)$$

Un exemple : Perceptron (suite)

Exemple : schéma de fonctionnement d'un noyau (cf. SVM, voir plus loin) :

- Cet exemple utilise un noyau quelconque $k(.,.)$
- Dans cet exemple, $b = w_0$

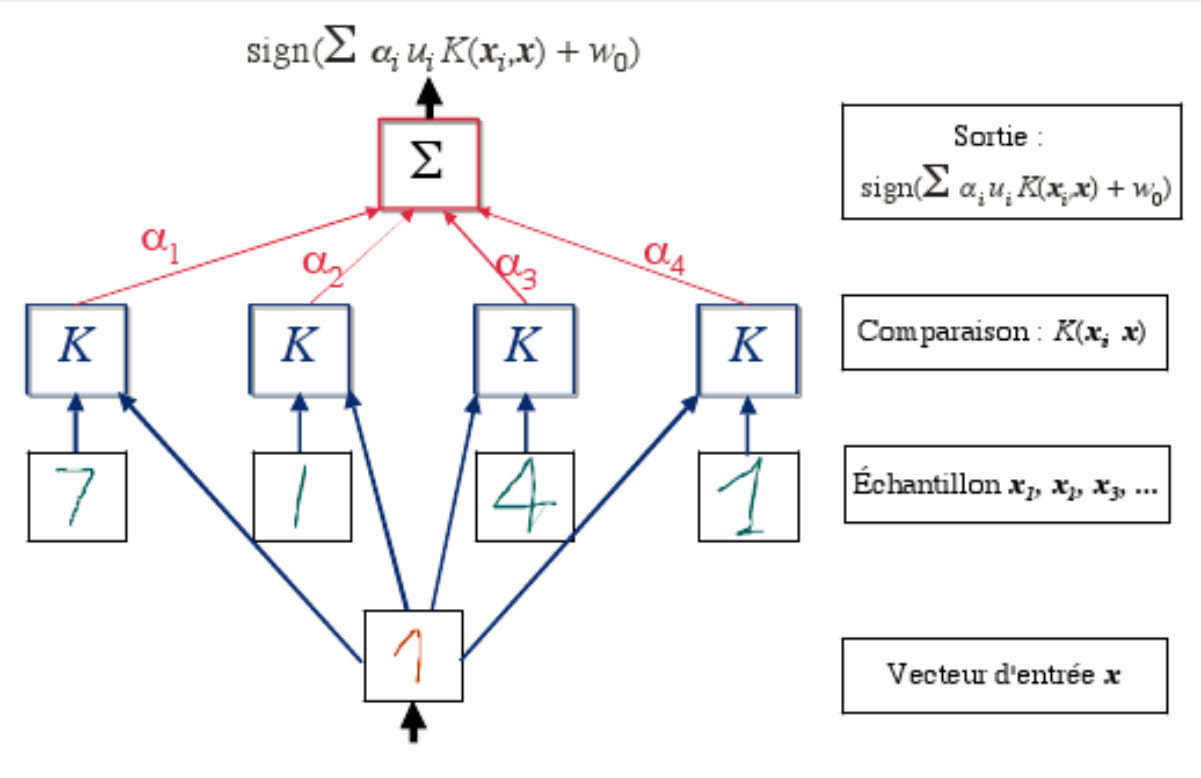


FIGURE 11: Reconnaissance de chiffre : $k(.,.)$ est le noyau, y_i (ici u_i) est la classe de x_i

Un exemple : Perceptron (suite)

- La classification fonctionnera selon l'algorithme (de Perceptron) suivant : (forme **duale**, cas bi-classes)

$\alpha = 0, b = 0$

Répéter

$erreurs = 0$

pour i=1 à n faire

si $signe(\sum_j \alpha_j y_j \langle x_j, x_i \rangle + b) \neq y_i$ **(instance mal classé)**

alors $\alpha_i = \alpha_i + 1; \quad b = b + y_i$

$erreurs = erreurs + 1$

finsi

finpour

Jusqu'à $erreurs = 0$ **(ou** $erreurs \leq \varepsilon$ **si normalisation)**

Exemples de noyaux

Un peu de théorie :

• Définition : étant donné un ensemble d'objets X, un noyau (kernel) défini positif est une fonction symétrique $k(x, x')$ telle que pour toute séquence finie de points $x_i \in X$ et $\alpha_i \in \mathbb{R}$: $\sum_{i,j} \alpha_i \alpha_j \, k(x_i, x_j) \geq 0$

➙ I.e., la matrice de de Gram $k(x_i, x_j)$ est symétrique semi-définie positive.

Théorème de Aronszajn (1950) :

k est un kernel défini positif SSI il existe un espace de *Hilbert* F et une application (mapping) $\Phi : X \mapsto F$ tels que

$$\forall (x, x') \in X^2, k(x, x') =< \Phi(x), \Phi(x') >_F = \Phi(x)^T \, \Phi(x')$$

➙ X = "espace d'entrées" (les points), Φ = "feature map"
F="espace des caractéristiques" (feature space).

Du point de vu fonctionnel, on a $f(x) = f^T \, \Phi(x)$
(reproduction de l'espace *Hilbert* du noyau)
➙ Considérer f comme une matrice (= *mapping*, application).

Exemples de noyaux (suite)

Exemples de noyaux courants (avec $x, z \in \mathbb{R}^d$) :

- **Linéaire** : $k(x,z) = x^T z = \langle x, z \rangle$
  ➙ $\Phi(x) = x$

- **Polynomial** : $k(x,z) = (\langle x, z \rangle)^d$
  ou $k(x,z) = (c + \langle x, z \rangle)^d$

  Exemple : avec $c = 1$, $k(x,z) = (1 + x^T z) = (1 + \langle x, z \rangle)^d$
  ➙ $\Phi(x) = \textit{Monôme}$
  ➛ c-à-d. expression comportant un seul terme de la forme 1 ou x^n pour le cas mono variable ou $x^a y^b$ si 2 variables x, y, voire $x_1^a x_2^b$ dans $\mathbb{R}^2$)

- **Gaussien** : $k(x,z) = e^{-\frac{\|x-z\|^2}{\sigma}}$
  ou $k(x,z) = e^{-\frac{\|x-z\|^2}{2\sigma^2}}$

- **Laplacien** : $k(x,z) = e^{-\frac{\|x-z\|}{\sigma}}$

Exemples de noyaux (suite)

- **Réseau de Neurones (RN)** : un noyau souvent utilisé

  $k(x,z) = tanh(\Theta < x,z > +v)$

  avec $tanh(u) = \dfrac{e^u - e^{-u}}{e^u + e^{-u}}$, $tanh$: tangente hyperbolique

  ➙ NB : dans un RN trivial, $u = \sum_k w_k x_k$

- **Normalized polyKernel** (cf. Weka) :

  ➙ $k(x,z) = \dfrac{\langle x,z\rangle^2}{(\langle x,x\rangle^2 \ . \ \langle z,z\rangle^2)^{1/2}}$

- **RBF** : fonction *Radiale* (cf. Weka)

  $k(x,z) = e^{-(0.01 \ * \ \langle x-z,x-z\rangle^2)}$

N.B. : Les noyaux à base *Radiale* constituent une famille dans laquelle une mesure de distance est lissée par une fonction *radiale* (*exponentielle*).
La forme générale d'un noyau RBF est $e^{-\gamma . d(x,z)}$, où γ est un paramètre d'échelle et $d(.,.)$ une métrique (distance).
Pour les noyaux Gaussiens, cette métrique est la distance Euclidienne.

Il y a d'autres métriques comme la Norme-L1, χ^2, etc.

Addendum : Construction de noyaux

- La construction d'un kernel s'appuie sur un principe :

  kernel sur X = Espace de fonctions sur X + Norme.

➙ On construit souvent un kernel à partir d'autres par des opérations algébriques (somme, produit, etc) :

Somme = concaténation d'espaces de caractéristiques

$$k_1(x,y) + k_2(x,y) = \langle \begin{pmatrix} \phi_1(x) \\ \phi_2(x) \end{pmatrix}, \begin{pmatrix} \phi_1(y) \\ \phi_2(y) \end{pmatrix} \rangle$$

Produit = produit *tenseuriel* d'espaces de caractéristiques

$$k_1(x,y).k_2(x,y) = \langle \phi_1(x)\phi_2(x)^T, \phi_1(y)\phi_2(y)^T \rangle$$

N.B. : dans le cas trivial des vecteurs, le produit *tenseuriel* de deux formes linéaires (vecteurs) représentera une forme bi-linéaire, linéaire par rapport à chacune des variables des formes de départ.

➙ Il représentera donc dans ce cas un produit de fonctions.

Exemples de noyau

Un exemple de noyau Polynomial dans $\mathbb{R}^2$: $k(x,z) = (\langle x,z\rangle)^2$

- Trouver $\phi(x)$ pour $x \in \mathbb{R}^2$ tel que $\langle x,z\rangle^2 = \langle \phi(x), \phi(z)\rangle = \phi(x)\ \phi(z)$

$$\begin{aligned} k(x,z) &= \langle x,z\rangle^2 \\ &= (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= \langle (x_1^2, \sqrt{2}x_1 x_2, x_2^2), (z_1^2, \sqrt{2} z_1 z_2, z_2^2)\rangle \end{aligned}$$

$$\phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{pmatrix}$$

➙ On a $\phi : x \in \mathbb{R}^2 \to \mathcal{H} \in \mathbb{R}^3$
tel que $(x.z)^2 = \phi(x).\phi(z)$ avec l'espace d'hypothèses $\mathcal{H}$.

- Le produit $\langle \phi(x), \phi(z)\rangle$ peut être calculé dans $\mathbb{R}^2$ l'espace d'origine au moyen du noyau $\langle x,z\rangle^2$ sans avoir à se projeter dans $\mathbb{R}^3$
  ➙ Ici, on peut calculer $\phi(x).\phi(z)$ sans devoir calculer ϕ : $k(x.z) = (x.z)^2$

- L'espace *intrinsèque* reste de dimension 2.

Exemples de noyau (suite)

Illustration :

- Soit des données définies dans un carré $[0,1] \times [-1,1] \in \mathbb{R}^2$
  - ➙ Situation typique dans les images (photos) de niveaux gris
- L'image complète de ϕ est donnée dans la figure illustrant cette projection.

- Cette image peut être dans un espace de plus grande dimension mais dont la dimension *intrinsèque* est 2 (comme pour les données).

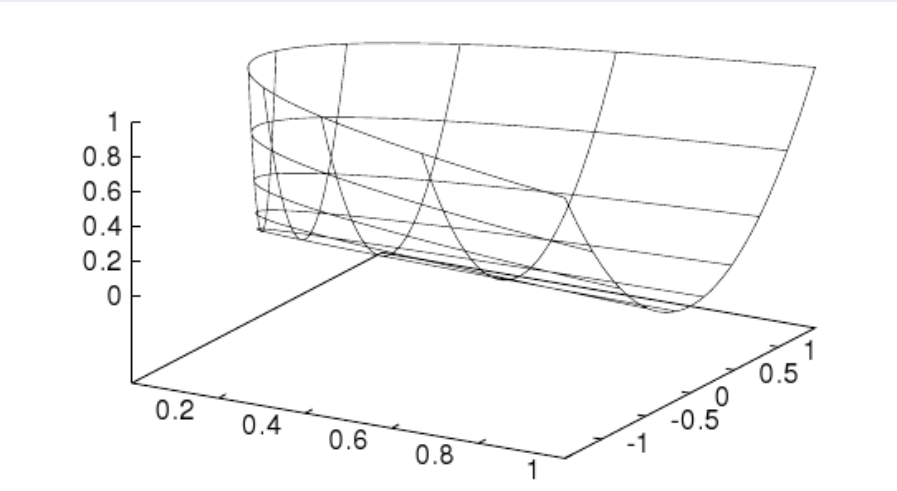


FIGURE 12: L'image dans $\mathcal{H}$ du carré $[0,1] \times [-1,1] \in \mathbb{R}^2$ par la projection ϕ

Exemples de noyau (suite)

- On note que ni ϕ ni $\mathcal{H}$ ne sont **uniques** pour un noyau donné.
  - ➙ On pourrait conserver $\mathcal{H}$ dans $\mathbb{R}^3$ mais utiliser

$$\phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} (x_1^2 - x_2^2) \\ 2x_1 x_2 \\ (x_1^2 + x_2^2) \end{pmatrix}$$

  - ➙ Ou encore $\mathcal{H}$ dans $\mathbb{R}^4$ et

$$\phi(x) = \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Notes sur $\mathcal{H}$:

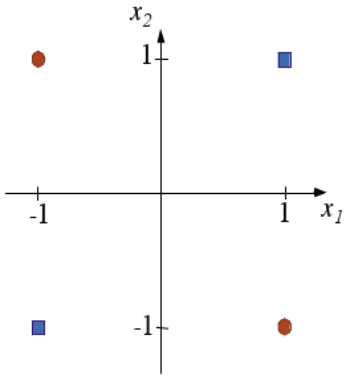
- Espace dit de *Hilbert* = une généralisation du linéaire / Euclidien qui définit un opérateur *produit* quelconque : pas seulement scalaire.
- Espace séparable : a un sous ensemble dont la fermeture est $\mathcal{H}$ lui même.
- + ...

Exemples de noyau (suite)

Un autre exemple de noyau quadratique : XOR

- Exemple de points non linéairement-séparables (la fonction XOR) :
- Fonction noyau **Polynomiale** :

$$k(x, z) = [1 + (x^T.z)]^2$$

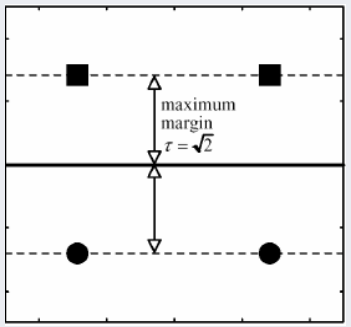


Index *i*	*x*	*u*
1	(1,1)	1
2	(1,-1)	-1
3	(-1,-1)	1
4	(-1,1)	-1

Soit : $1 + x_1^2 x_{i1}^2 + 2x_1 x_2 x_{i1} x_{i2} + x_2^2 x_{i2}^2 + 2x_1 x_{i1} + 2x_2 x_{i2}$

On aura : $w = \sum_{i=1..4} \alpha_i . y_i . \phi(x_i) = [0, 0, 0, \frac{1}{\sqrt{2}}, 0, 0]^T$

Correspond à $\mathcal{H} \in \mathbb{R}^6$ et à la projection $\phi(x) = \begin{pmatrix} 1 \\ x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \end{pmatrix}$



Complément : Opérateurs de l'EdC

• La méthode des noyaux a de multiples applications.
• Par exemple, en *clustering*, la méthode *k-means* cherche à minimiser la distance entre un ensemble de points (dans un cluster) et le centroïde (ou le médoïde) de ce ensemble.
➙ Cette distance peut être calculée par l'utilisation du produit matriciel :

$$||x - z||^2 =< x, x > + < z, z > -2 < x, z >.$$

• Pour cette méthode ainsi que d'autres à base de noyaux appliquée dans un espace de caractéristiques (*EdC*), nous pouvons définir quelques opérations ci-dessous.

• Pour un ensemble d'instances (vecteurs) $S = \{x_1, ..., x_\ell\}$, on a un noyau $k(x, z)$ et une *application* ϕ dans un EdC F qui satisfait

$$k(x, z) =< \phi(x), \phi(z) >.$$

• Soit $\phi(S) = \{\phi(x_1), ..., \phi(x_\ell)\}$ l'image de S par ϕ.
➙ $\phi(S)$ est un sous-ens. de l'espace F muni du produit matricielle $< ., . >$.

Complément : Opérateurs de l'EdC (suite)

- Soit la matrice K du noyau k contenant les évaluations du noyau entre toute paires des éléments de S : $K_{ij} = k(x_i, x_j), \quad i, j = 1..\ell$
- On travaillera donc avec l'image $\phi(x)$ à la place de l'instance x.
- On a :
  - $||\phi(x)||_2 = \sqrt{||\phi(x)||^2} = \sqrt{< \phi(x), \phi(x) >} = \sqrt{k(x,x)}$ (Norme de EdC)
  - $\hat{\phi}(x) = \dfrac{\phi(x)}{||\phi(x)||}$ (normalisation de ϕ nécessaire dans les algos, voir ci-dessous)
  - $\hat{k}(x,z) =< \hat{\phi}(x), \hat{\phi}(z) >= \left\langle \dfrac{\phi(x)}{||\phi(x)||}, \dfrac{\phi(z)}{||\phi(z)||} \right\rangle = \dfrac{< \phi(x), \phi(z) >}{||\phi(x)||\ ||\phi(z)||}$
    $= \dfrac{k(x,z)}{\sqrt{k(x,x)k(z,z)}}$ (le noyau transformé $\hat{k}$ pour 2 instances,)
  - $\left|\left|\sum_{i=1}^{\ell} \alpha_i \phi(x_i)\right|\right|^2 = \left\langle \sum_{i=1}^{\ell} \alpha_i \phi(x_i), \sum_{j=1}^{\ell} \alpha_j \phi(x_j) \right\rangle = \sum_{i=1}^{\ell} \alpha_i \sum_{j=1}^{\ell} \alpha_j \langle \phi(x_i), \phi(x_j) \rangle$
    $= \sum_{i,j=1}^{\ell} \alpha_i \alpha_j k(x_i, x_j)$ (La norme de la combinaison des images dans EdC)

Complément : Opérateurs de l'EdC (suite)

- Un cas spécial de la Norme est la longueur de la droite (distance) rejoignant deux images $\phi(x)$ et $\phi(z)$:
  $||\phi(x) - \phi(z)||^2 =< \phi(x) - \phi(z), \phi(x) - \phi(z) >$ (Distance entre 2 vecteurs dans F)
  $=< \phi(x), \phi(x) > -2 < \phi(x), \phi(z) > + < \phi(z), \phi(z) >$
  $= k(x,x) - 2k(x,z) + k(z,z)$

- $\phi_S = \frac{1}{\ell} \sum_{i=1}^{\ell} \phi(x_i)$ (Norme et distance du *centroïde* (centre de la masse) de l'ensemble $\phi(S)$)

  ➙ Il se peut que ϕ_S ne correspondent à aucun point x dont l'image sous ϕ soit dans ϕ_S. Cependant, on peut calculer sa Norme à l'aide du noyau :

  - $||\phi_S||_2^2 =< \phi_S, \phi_S > = \left\langle \frac{1}{\ell} \sum_{i=1}^{\ell} \phi(x_i), \frac{1}{\ell} \sum_{j=1}^{\ell} \phi(x_j) \right\rangle = = \frac{1}{\ell^2} \sum_{i,j=1}^{\ell} < \phi(x_i), \phi(x_j) >$

    $= \frac{1}{\ell^2} \sum_{i,j=1}^{\ell} k(x_i, x_j)$

    ➙ Le carrée de la Norme du centre des masses est la moyenne des entrées de la matrice du noyau.

    ➙ Ce qui implique que la somme obtenue est ≥ 0 et en cas d'égalité, si le centroïde est l'origine du système des coordonnées.
  - On Peut calculer la distance de l'image d'un point x du centroïde ϕ_S :
    $||\phi(x) - \phi_S||^2 =< \phi(x), \phi(x) > + < \phi_S, \phi_S > -2 < \phi(x), \phi_S >$

    $= k(x,x) + \frac{1}{\ell^2} \sum_{i,j=1}^{\ell} k(x_i, x_j) - \frac{2}{\ell} \sum_{i=1}^{\ell} k(x, x_i)$

Complément : Opérateurs de l'EdC (suite)

- De manière analogue, pour un ensemble donné de points S dans EdC, la distance de chaque point de S au centroïde ϕ_S est donné ci-dessous.
  ➙ Ce calcul a ud'intérêt par exemple dans Kmeans :

$$\frac{1}{\ell}\sum_{s=1}^{\ell}||\phi(x_s)-\phi_S||^2 = \frac{1}{\ell}\sum_{s=1}^{\ell}k(x_s,x_s)+\frac{1}{\ell^2}\sum_{i,j=1}^{\ell}k(x_i,x_j)-\frac{2}{\ell^2}\sum_{i=1,s}^{\ell}k(x_s,x_i)$$

$$= \frac{1}{\ell}\sum_{s=1}^{\ell}k(x_s,x_s)-\frac{1}{\ell^2}\sum_{i,j=1}^{\ell}k(x_i,x_j)$$

  ➙ La moyenne des carrées des distances des points à leur centroïde est la moyenne de la diagonale de la matrice du noyau (la *trace* de la matrice du noyau de l'ensemble S divisée par sa taille) moins la moyenne de toutes les entrées.
- Et pour terminer : le centroide ϕ_S d'un ensemble de points $\phi(S)$ doit résoudre le problème d'optimisation suivant :

$$argmin_{\mu}\frac{1}{\ell}\sum_{s=1}^{\ell}||\phi(x_s)-\mu||^2 \qquad \text{Où } \mu=\phi_S \text{ ci-dessus.}$$

N.B. : le code MatLab de normalisation de la matrice du noyau k est assez simple :

```
% Soit K la matrice du noyau.
% D sera la matrice diagonale contenant l'inverse de la Norme
    D = diag(1.0/sqrt(diag(K)));
    K = D * K * D;          % k contiendra le résultat
```

SVM : introduction

SVM : **Support Vector Machine** (*Séparateur à Vastes Marges*)

- Méthode d'apprentissage **supervisé**
- Classification binaire
- Le SVM est un *classifieur linéaire à marge maximale* dans un espace à noyau.

  Elle est basée sur une minimisation du *risque empirique* régularisé (erreur) sur un espace fonctionnel (de *Hilbert*) et avec une *fonction de perte* linéaire par morceaux (cumulée).

- Le **Perceptron** est un cas simple de SVM.

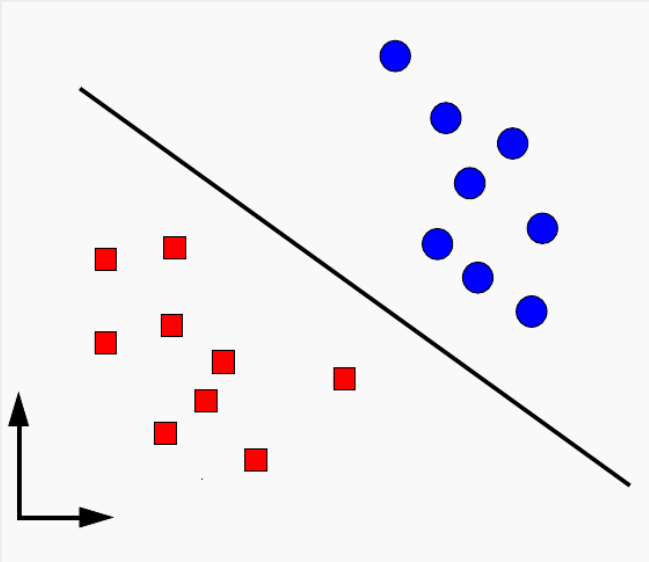
SVM : introduction (suite)

Définitions (cas binaire) :

- Un ensemble d'exemples étiquetés $\{(x_1, y_1), ..., (x_n, y_n)\}$ est dit **linéairement séparable**, si il existe w, b tels que :

$$\forall i,\ \gamma_i = y_i(\langle w, x_i \rangle + b) > 0$$

- Ce qui signifie qu'il existe un hyperplan tel que :
  - tous les exemples d'apprentissage positifs sont dans un demi-plan et
  - tous les négatifs dans l'autre.



- On cherche h sous forme d'une fonction linéaire : $h(x) = w.x + b$
  - ➙ N.B. : b est le biais $= w_0$ du schéma linéaire.

SVM : introduction (suite)

- Il peut exister une infinité de plans de séparation linéaires (selon w) :

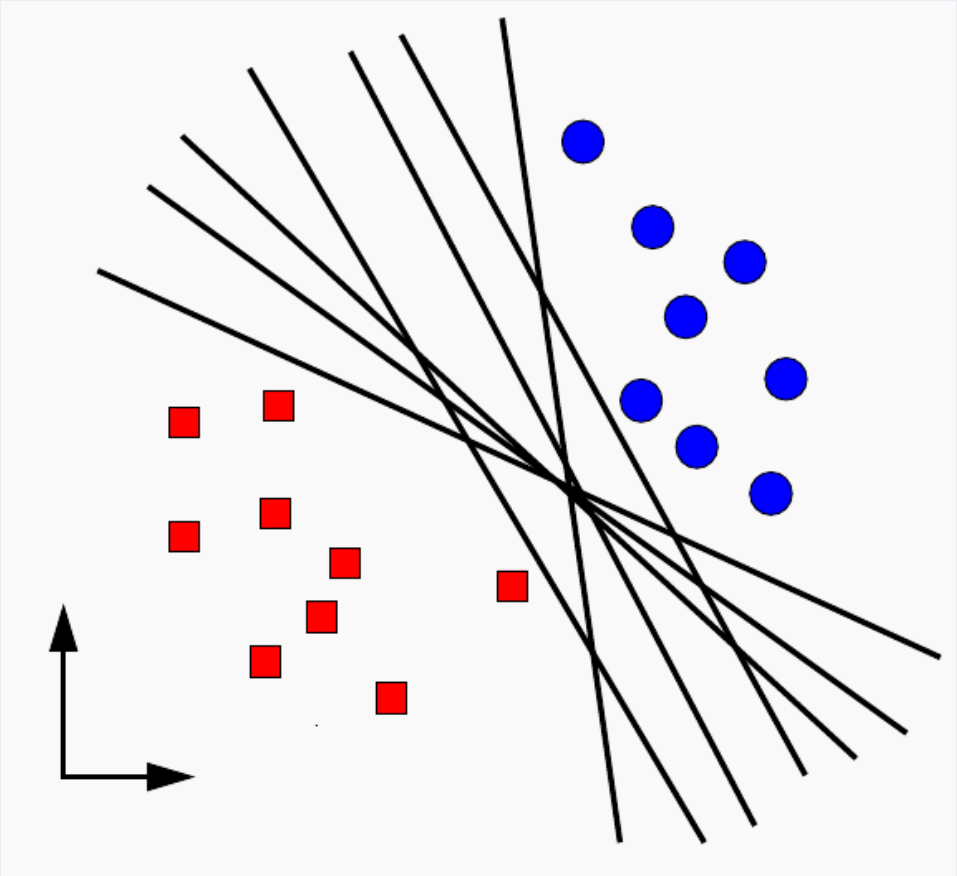


FIGURE 13: Le problème du choix du plan linéaire

SVM : introduction (suite)

- On choisira celui qui a la plus grande marge
- Exemple de 2 plans de séparation :

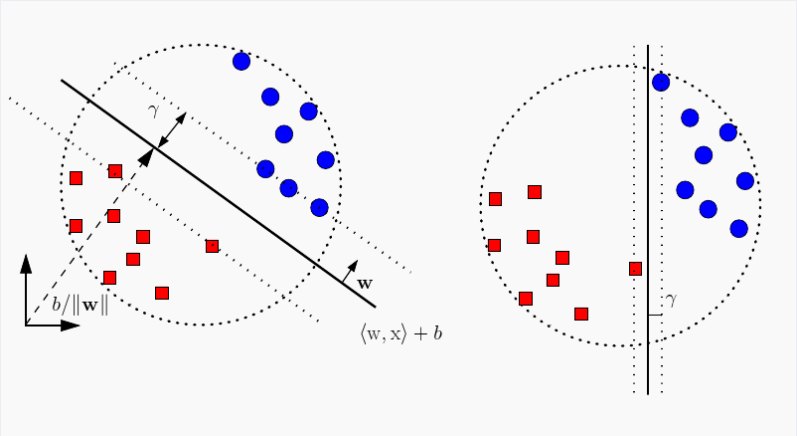


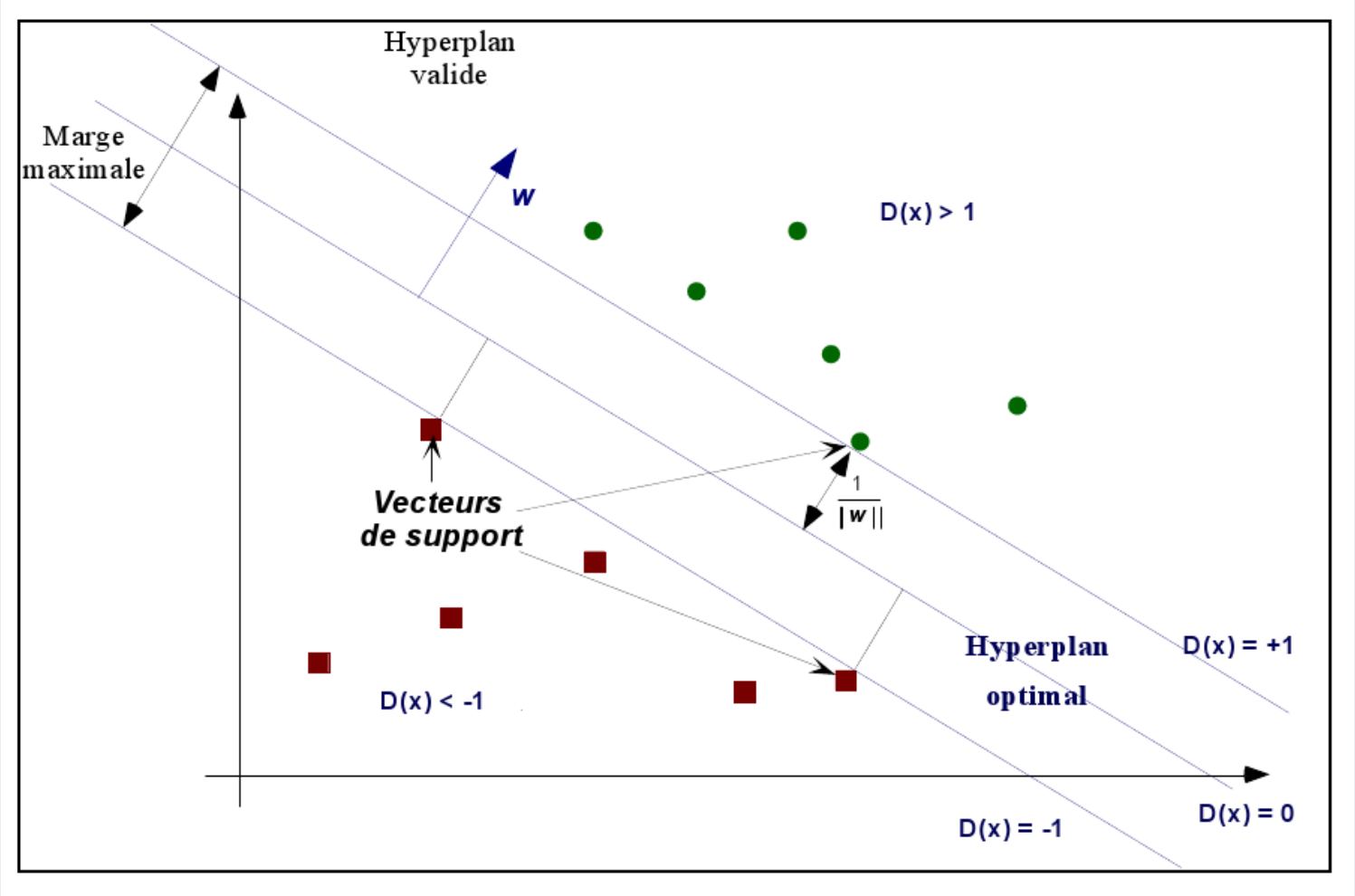
FIGURE 14: La marge à gauche est plus grande que celle de droite

- La fonction linéaire h recherchée : $h(x) =< w, x > +b = w.x + b$.
  - La surface de séparation correspondant à la fonction linéaire h est l'hyperplan $w.x + b = 0$
  - Elle est valide si $\forall i,\ y_i h(x_i) \geq 0$

➛ On se pose la question : **marge par rapport à quoi** ?

SVM : calcul de la marge optimale

Optimisation de la marge (remarquer les *vecteurs de support* d'où SVM) :

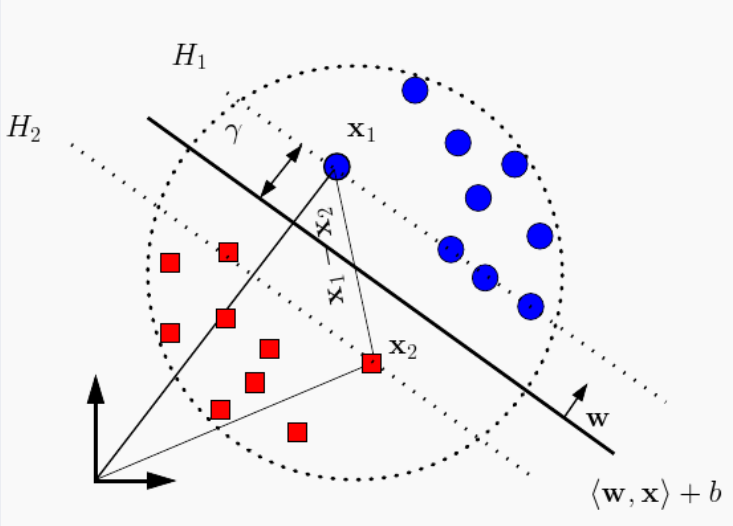


SVM : calcul de la marge optimale (suite)

• Notons x_{+1} un exemple d'un côté du plan et x_{-1} (e.g. x_2 rouge) de l'autre.

• Choisir x_{+1} et x_{-1} correspondants aux vecteurs se trouvant sur les frontières définissant les marges,

• Constatons : l'hyperplan doit se trouver à **mi-distance** entre x_{+1} et x_{-1}.



• La marge sera simplement **la moitié** de la distance perpendiculaire entre x_{+1} et x_{-1} projetée sur la normale au plan H (cf. *Rocchio*) :

$$\gamma = \frac{1}{2} \frac{\langle w, (x_{+1} - x_{-1}) \rangle}{\| w \|}$$

☞ Rappel : d'une manière générale, la distance de tout point x_i (de l'espace) à l'hyperplan d'équation $ax - by + c = 0$ (notée $< w, x_i >= w^T x_i + w_0 = 0$) est :

$$\frac{|<w,x_i>|}{\|w\|} = \frac{|w^T x_i + w_0|}{\|w\|}.$$

SVM : calcul de la marge optimale (suite)

A noter : pour tout $c \neq 0$, on a :

$$\{x : \langle w, x\rangle + b = 0\} = \{x : \langle cw, x\rangle + cb = 0\}$$

➙ (cw, cb) défini le même hyperplan que (w, b).

• L'hyperplan est en forme dite **canonique** relativement à l'ensemble de points $X = \{x_1, ..., x_n\}$ si : $min_{x_i \in X} |\langle w, x_i\rangle + b| = 1$. $|.|$: val abs.

• Dans notre cas, on a : $\langle w, x_{+1}\rangle + b = +1$ $\langle w, x_{-1}\rangle + b = -1$

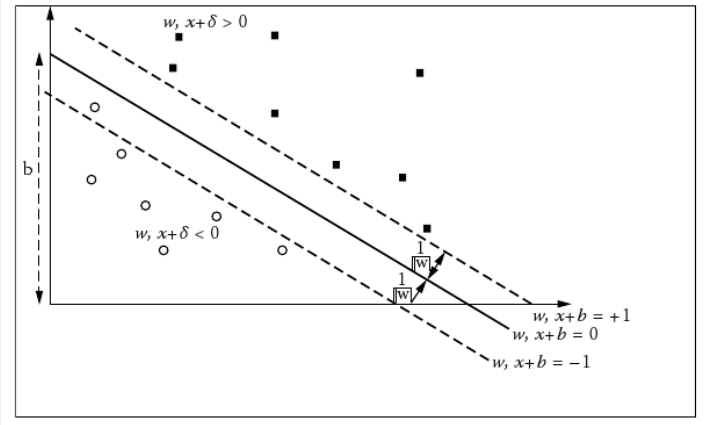
Et donc : $2 = (\langle w, x_{+1}\rangle + b) - (\langle w, x_{-1}\rangle + b)$

➙ $\langle w, (x_{+1} - x_{-1})\rangle = 2$

• Nous cherchons un hyperplan canonique.

On avait $\gamma = \frac{1}{2} \frac{\langle w, (x_{+1} - x_{-1})\rangle}{\|w\|}$

➙ La marge à maximiser $\gamma = \dfrac{1}{\| w \|}$



SVM : calcul de la marge optimale (suite)

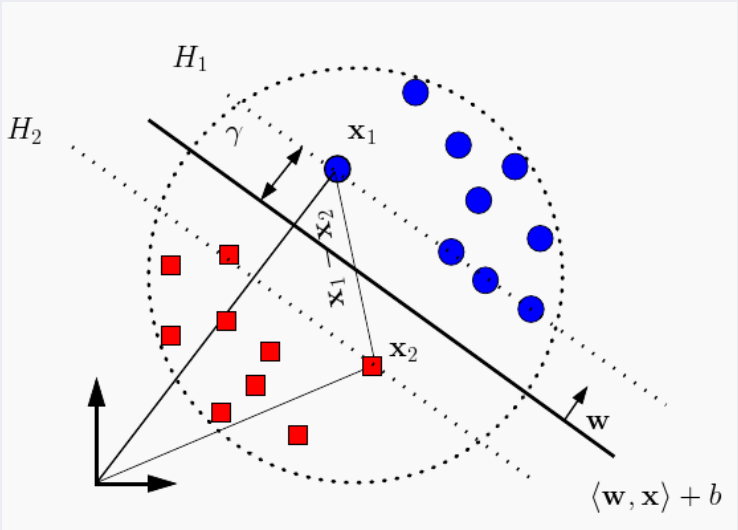
- Maximiser la marge $\gamma = \frac{1}{\|w\|} \equiv$ minimiser $\| w \|$
- La marge maximum visera $\gamma = 1$ (pour les vecteurs de support), $\gamma \geq 1$ (pour les autres) sous la contrainte que TOUS les exemples soient bien classifiés.
- La résolution de ce problème relève de la **programmation quadratique**.
  ➙ Pour ce faire, on fait figurer un terme quadratique.
- Astuce : minimiser $\| w \|$ revient à minimiser $\frac{1}{2} \| w \|^2$:

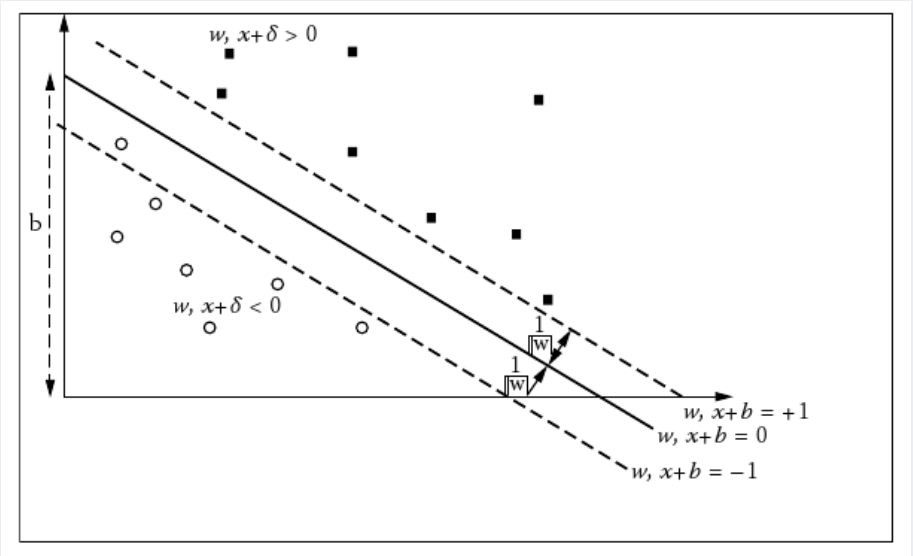
$$(P) = \begin{cases} min. & \frac{1}{2} \| w \|^2 \\ s.t. & y_i[\langle w, x_i \rangle + b] \geq 1 \end{cases}$$

☞ Rappel : $\| w \|^2 = \langle w, w \rangle = x_w^2 + y_w^2$ où $w = (x_w, y_w)$
- Le carré de $\| w \|^2$ est utilisé pour éviter la racine carrée de $\| w \|$.
- Minimiser $\| w \|$ (ou $\| w \|^2$) revient à minimiser $\frac{\|w\|^2}{2}$ (mêmes solutions),
- **Un autre intérêt** d'introduire $1/2 \| w \|^2$ est de simplifier la dérivée de la forme Primale (v. suite)

SVM : calcul de la marge optimale (suite)

Rappel :





→

$$(P) = \begin{cases} min. & \frac{1}{2} w^T.w \\ s.t. & y_i(w.x_i + b) \geq 1 \end{cases}$$

SVM : calcul de la marge optimale (suite)

Interprétation Géométrique :

- La marge **fonctionnelle** pour un x_i (ici, $w_0 = b$) :
$$\gamma_i = h(x_i).y_i = (w^T x_i + w_0).y_i$$

- La marge **géométrique** :
$$\gamma_i^{(g)} = \frac{1}{\| w \|}(w^T x_i + w_0).y_i = \frac{1}{\| w \|}\gamma_i$$

- Le problème d'optimisation de la marge revient à (de manière équivalente) :
  - Maximiser la marge géométrique
  - Maximiser la marge fonctionnelle sous la contrainte $\| w \| = 1$
  - Minimiser $\| w \|$ sous la contrainte $\gamma_i \geq 1$

SVM : calcul de la marge optimale (suite)

- **Présentation géométrique à deux classes :**

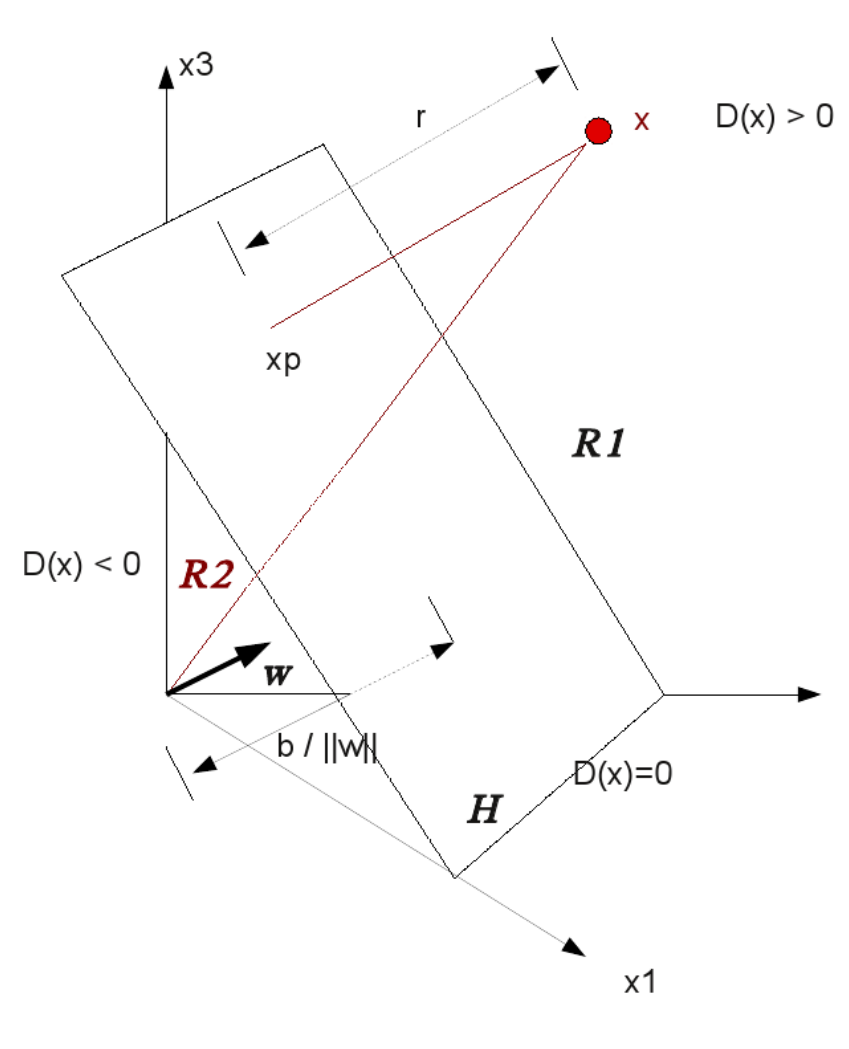
$r =$ distance algébrique de x au plan H (ici $b = w_0$) :

$$x = x_p + r.\frac{w}{\| w \|}$$

$$w^T x + w_0 = r.\frac{w.w^T}{||w||} = r.\frac{||w||^2}{||w||}$$

$$w^T x + w_0 = r \| w \| = D(x)$$

$$r = \frac{D(x)}{\| w \|}$$



SVM : calcul de la marge optimale (suite)

La formulation du problème :

Le problème d'optimisation peut être formulé par :

- Soit un ensemble d'exemples étiquetés $\{(x_1, y_1), ..., (x_n, y_n)\}$ linéairement séparables,

- Minimiser $\| w \|^2 = w^T w$. sous la contrainte
$$\gamma_i = (w^T x_i + w_0).y_i \geq 1, \ i = 1, ...n$$

➙ Le résultat : l'hyperplan $w^T x_i + w_0 = 0$ avec une marge géométrique $\frac{1}{\| w \|}$ **maximale**

☞ Pour résoudre le système Primal (P), nous avons recours à la technique de *Lagrange*. ../..

Complément : le Lagrangien et l'optimisation*

Quelques éléments utiles pour comprendre la résolution (SVM).

- Soit note problème d'optimisation

$$P = \begin{cases} min & f(x) \\ s.t. & g_i(x) \leq 0, i = 1, ..., k \\ & h_i(x) = 0, i = 1, ..., m \end{cases}$$

- La fonction $\mathcal{L} : \mathbb{R}^{n+k+m} \to \mathbb{R}$ définie par

$$\mathcal{L}(x, \alpha, \beta) = f(x) + \sum_{i=1}^{k} \alpha_i g_i(x) + \sum_{i=1}^{m} \beta_i h_i(x)$$

est appelée **Lagrangien** ou fonction Lagrangienne et les $\alpha_i \in \mathbb{R}^k$ et $\beta_i \in \mathbb{R}^m$ sont appelés les *multiplicateurs* de Lagrange.

Complément : le Lagrangien et l'optimisation* (suite)

Un exemple (d'optimisation Primale) :
Trouver la boite avec le plus grand volume étant donné une certaine surface c.

- On note les cotés de la boite u, v, w
- Le problème d'optimisation :
  Minimiser $-uvw$ (ou maximiser le volume uvw)
  s.t. $wu + uv + vw = c$
- Le **Lagrangien** : $\mathcal{L} = -uvw + \alpha(uv + wu + vw - c)$ à minimiser
  ➙ Conditions nécessaires :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= -uv + \alpha(u+v) = 0\,; \\
\frac{\partial \mathcal{L}}{\partial u} &= -wv + \alpha(w+v) = 0\,; \\
\frac{\partial \mathcal{L}}{\partial v} &= -uw + \alpha(u+w) = 0\,;
\end{aligned}$$

- La résolution de ces équations donnent $\alpha v(w-u) = \alpha w(u-v) = 0$
  ➙ Ce qui donne comme résultat : $w = v = u = (\frac{c}{3})^{\frac{1}{2}}$

Complément : le Lagrangien et l'optimisation* (suite)

La fonction lagrangienne duale de P (cf. Prog. Math.) :

- Est un Lagrangien où il faut maximiser un objectif (P minimise).
- Pour le problème Primale d'optimisation $(P) = \begin{cases} min\ f(x) \\ s.t. \quad g_i(x) \leq 0, i = 1, ..., k \\ \qquad\ h_i(x) = 0, i = 1, ..., m \end{cases}$

  et sa fonction de Lagrange $\mathcal{L}(x, \alpha, \beta) = f(x) + \sum_{i=1}^{k} \alpha_i g_i(x) + \sum_{i=1}^{m} \beta_i h_i(x)$

- La fonction $\Theta : \mathbb{R}^{k+m} \to \mathbb{R}$ définie par $\Theta(\alpha, \beta) = \inf_{x \in X} \mathcal{L}(x, \alpha, \beta)$ est la fonction **duale** de (P).

  ➙ Les paramètres α et β sont les variables *duales* et les x variables *Primales*.
- Le problème dual de Lagrange correspondant à (P) est donné par

$$(D) \begin{cases} max \quad \Theta(\alpha, \beta) \\ s.t. \quad \alpha_i \geq 0, \ i = 1, ..., k \end{cases}$$

où $\Theta(\alpha, \beta) = \inf_{x \in X} \mathcal{L}(x, \alpha, \beta)$

(α, β obtenus via les extrema de P en dérivant f, voir résolution SVM)

Complément : le Lagrangien et l'optimisation* (suite)

$$(P) = \begin{cases} min\ f(x) \\ s.t. \quad g_i(x) \leq 0, i = 1, ..., k \\ \qquad h_i(x) = 0, i = 1, ..., m \end{cases}$$

$$(D) = \begin{cases} max \quad \Theta(\alpha, \beta) \\ s.t. \quad \alpha_i \geq 0, \ i = 1, ..., k \end{cases}$$

où $\Theta(\alpha, \beta) = \inf_{x \in X} \mathcal{L}(x, \alpha, \beta)$

• La valeur de la fonction objectif (la solution optimale pour Primale et Duale) est appelée **la valeur du problème**.

• La différence entre les valeurs des problèmes Primal et Dual s'appelle *le saut (ou gap) de dualité.*

Théorème de la dualité faible (lien entre Primal et Dual) :

• Soit $x \in X$ une solution admissible de (P) et (α, β) une solution admissible de (D) le dual de (P).

➙ Alors $\Theta(\alpha, \beta) \leq f(x)$

➙ C-à-d. la valeur optimale de (P) est plus grande ou égale à celle de (D).

➙ Si le saut est nul (cas de *h et g sont affines* de la forme $g(x) = Ax + c$), on a le choix entre la formulation Primale et Duale.

SVM : résolution*

On décide de résoudre (P) via les multiplicateurs de Lagrange :

$$\begin{cases} min. & \frac{1}{2} \parallel w \parallel^2 \\ s.t. & y_i[\langle w, x_i \rangle + b] \geq 1 \end{cases}$$

• Pour satisfaire les contraintes dans ce problème de minimisation (par un Lagrangien primal L_P), on y associe le multiplieur de Lagrange α : $\alpha_i \geq 0, \forall i$:

$$\begin{aligned} L_P = \mathcal{L}(w, b, \alpha) &= \frac{1}{2} \parallel w \parallel^2 - \alpha_i[y_i(\langle w, x_i \rangle + b) - 1] \; \forall i \qquad \text{d'où} \\ &= \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i[y_i(\langle w, x_i \rangle + b) - 1] \\ &= \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i y_i(\langle w, x_i \rangle + b) + \sum_{i=1}^{n} \alpha_i \end{aligned}$$

☞ Il faut minimiser le Lagrangien $\mathcal{L}(w, b, \alpha)$.

➙ Pour cela, on dérive $\mathcal{L}(w, b, \alpha)$ par rapport aux variables (primales) w et b et on obtient le Lagrangien dual L_D.

SVM : résolution* (suite)

Remarque (sur un problème) : on avait (page précédente)

$$L_P = \mathcal{L}(w, b, \alpha) = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i [y_i(\langle w, x_i \rangle + b) - 1]$$

à minimiser sur (w, b, α), c'est à dire :

$$\min_{w,b,\alpha} \left\{ \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i [y_i(\langle w, x_i \rangle + b) - 1] \right\}$$

➙ Or, si on trouve un hyperplan séparant les points (avec tous les $y_i(\langle w, x_i \rangle + b) - 1 \geq 0$), on peut décider $\alpha_i \to \infty$ pour obtenir le minimum recherché, pour tous les points et pas seulement pour les vecteurs supports.

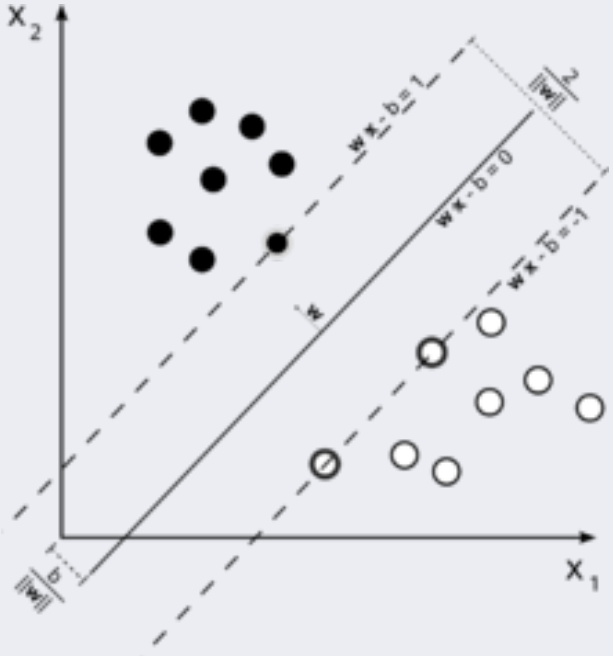
• Pour éviter ce problème, on reformule la contrainte précédente en :

$$\min_{w,b} \max_{\alpha} \left\{ \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i [y_i(\langle w, x_i \rangle + b) - 1] \right\}$$

SVM : résolution* (suite)

➙ De cette manière, les points avec $y_i(\langle w, x_i \rangle + b) - 1 > 0$ ne nous intéresseront plus (auront leur $\alpha_i = 0$, voir plus loin).

☞ Seuls quelques points donneront les vecteurs supports, les autres seront sans intérêt.



Addendum : Résolution Primale

La remarque précédente justifie une dérivation par rapport à w et b :

• Les conditions à l'optimum exigent que les dérivées par rapport à w et b soient nulles (pas de dérivation p/r à α_i) :

$$\frac{\partial \mathcal{L}(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\partial \mathcal{L}(w,b,\alpha)}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Conclusions Primal :

- On aura seuls quelques $\alpha_i > 0$ dont x_i donneront les *vecteurs supports* placés sur la marge maximisée et qui satisfont $y_i(\langle w, x_i \rangle + b) - 1 = 0$.
- Ces points satisferont $\langle w, x_i \rangle + b = \frac{1}{y_i} = y_i$ (puisque $y = 1$ ou $y = -1$)
  ➙ et donc $b = (w.x_i) - y_i$
- En pratique, on préfère calculer $b_i = (w.x_i) - y_i$ pour chacun des Nb_{SV} vecteurs supports puis utiliser $b = \frac{1}{Nb\_SV} \sum_1^{Nb\_SV} b_i$.

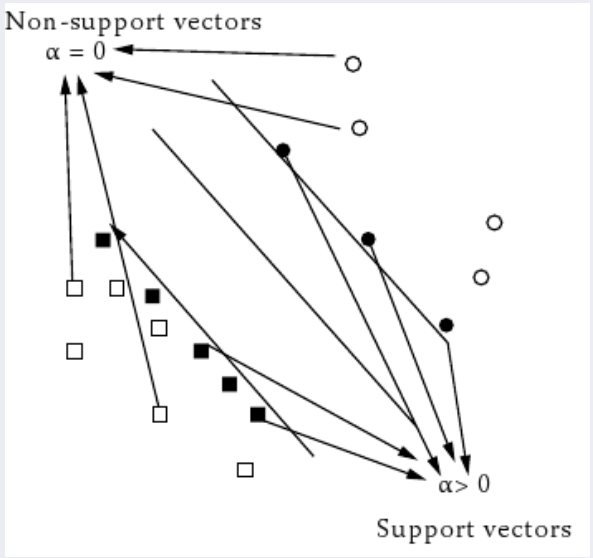
Addendum : Résolution Primale (suite)

• Issues des conclusions précédentes, une série de conditions (appelées conditions de *Karush-Kuhn-Tucker* = *KKT*) pour la SVM à marge maximale impliquent (α_i^* est l'optimum de α) :

$\alpha_i^* > 0 \Rightarrow x_i$ est un Vecteur de support
$\alpha_i^* = 0 \Rightarrow$ Pas un Vecteur support

• Rappel : α_i : coefficients de Lagrange

• Figure en face :
- les SV (les formes pleines, avec $\alpha > 0$) et
- les non SV (les autres, avec $\alpha = 0$)



Addendum : Passage à la forme Duale

Rappel forme Primale et les optima : :

$$L_P = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i y_i (\langle w, x_i \rangle + b) + \sum_{i=1}^{n} \alpha_i \quad \alpha_i \geq 0, \forall i$$

Dans laquelle (optimal) $\quad w^* = \sum_{i=1}^{n} \alpha_i y_i x_i \quad$ et $\quad \sum_{i=1}^{n} \alpha_i y_i = 0$

Forme Duale : une meilleure technique d'optimisation.

➙ Simplifie les calculs.

• Pour obtenir le problème dual de cette optimisation, il faut substituer les conditions stationnaires w^* et b^* dans $\mathcal{L}$, puis de maximiser le résultat (dual).

• La contrainte sur les variables duales (α_i) est $\alpha_i \geq 0$.

Addendum : Passage à la forme Duale (suite)

En substituant les résultats de $\frac{\partial \mathcal{L}(w,b,\alpha)}{\partial w} = 0$ et $\frac{\partial \mathcal{L}(w,b,\alpha)}{\partial b} = 0$ dans $L_P = \mathcal{L}(w, b, \alpha)$, on obtient le problème **Dual à maximiser**.

- On avait : $$L_P = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i y_i (\langle w, x_i \rangle + b) + \sum_{i=1}^{n} \alpha_i$$
- On réorganise : $$L_P = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i (w^T x_i + b) + \frac{1}{2} w^T w$$

- On substitue $w = \sum_{i=1}^{n} \alpha_i y_i x_i$:

$$L_D = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \qquad \textbf{à maximiser}$$

$$s.t. \qquad \alpha_i \geq 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \forall i \qquad \text{☞ pour tous les } i$$

Addendum : Passage à la forme Duale (suite)

Ce qui conduit à la formulation duale :

$$(D) \begin{cases} max_\alpha \quad \Theta_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ s.t. \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \qquad \alpha_i \geq 0, \ i = 1, ..., n \quad \text{☞ pour tous les i} \end{cases}$$

• On remarque que le problème dual est indépendant de la dimension de x (nbr. d'attributs) mais est dépendant du nombre d'observations (n).

> ☞ *C'est une propriété très intéressante*, spécialement lorsque X est de grande dimension et que le nombre d'observations reste petit.

• On a (issue de Lp optimale) $w = \sum_{i=1}^{n} \alpha_i y_i x_i$

• D'une point de vue de noyau (kernel), on dira que la formulation Duale montre que le noyau **linéaire** utilisé ici est $k(x_i, x_j) = \langle x_i, x_j \rangle = x_i \cdot x_j$

Addendum : Passage à la forme Duale (suite)

Remarques :

• Parfois, pour simplifier les calculs, on impose que l'hyperplan calculé passe par l'origine du repère.

➙ On l'appellera un hyperplan *non-biaisé* (vs. *biaisé* = cas général).

☞ Pour imposer cette contrainte supplémentaire, on pose $b = 0$ dans la forme Primale

➙ La contrainte correspondant à la forme Duale implique que l'on supprime de celle-ci la contrainte $\sum_{i=1}^{n} \alpha_i y_i = 0$.

Addendum : Calculs effectifs de SVM

Calculs effectifs de w et de b :

• Le problème quadratique d'optimisation convexe (Lagrangien) peut être soumis à par exemple un solveur quadratique (QP solver) qui renvoie α (les coefficients de Lagrange) permettant de calculer w.
• Il nous reste à calculer b (w_0).

• Tout exemple satisfaisant L_D sera un Vecteur de Support x_{sv} de la forme

$$y_{sv}(x_{sv}.w + b) = 1$$

On avait, lors du calcul de $\frac{\partial \mathcal{L}}{\partial w}$: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$

➙ On obtient : $y_{sv}(\sum_{m \in SV} \alpha_m y_m x_m.x_{sv} + b) = 1$

où SV est l'ensemble des Vecteurs de Support contenant les x_i pour lesquels $\alpha_i > 0$.

• On vait également $x_i.w + b \geq +1 \quad pour\ y_i = +1$ (instances positives)
et $x_i.w + b \leq -1 \quad pour\ y_i = -1$ (instances négatives)

Addendum : Calculs effectifs de SVM (suite)

Rappel : $y_{sv}(\sum_{m \in SV} \alpha_m y_m x_m . x_{sv} + b) = 1$

- Ce qui permet de multiplier l'équation ci-dessus par y_{sv} sachant que $y_{sv}^2 = 1$:
  - ➙ $y_{sv}^2(\sum_{m \in SV} \alpha_m y_m x_m . x_{sv} + b) = y_{sv}$ (multiplication)
  - ➙ $b = y_{sv} - \sum_{m \in SV} \alpha_m y_m x_m . x_{sv}$
- Sachant que pour différents x_{sv}, on obtient différents b, on utilise la moyenne des bs ainsi calculés sur l'ensemble SV de taille N_{sv} des vecteurs de support :

$$b = \frac{1}{N_{sv}} \sum_{m \in SV} (y_{sv} - \sum_{m \in SV} \alpha_m y_m x_m . x_{sv})$$

- Ainsi, on aura b et w et donc l'hyperplan optimal et notre SVM.

N.B. : pour simplifier, b peut également être exprimé par :

$$b^* = -\frac{1}{2}(\max_{y_i=-1} \mathrm{w}^* . \mathrm{x_i} + \max_{y_i=+1} \mathrm{w}^* . \mathrm{x_i})$$

Addendum : résumé des calculs

La fonction de décision :

- Vecteur de poids optimal $w^* = \sum_{i:\alpha_i^* \geq 0}^{n} \alpha_i^* x_i y_i$
- Marge optimale $\gamma^* = \frac{1}{\| w^* \|} = \left(\sum_{i=1}^{n} \alpha_i^* \right)^{-1/2}$
- Le biais optimal : $b^* = \frac{1}{2} \left[\min_{y_i=+1} (\langle w^*, x_i \rangle) + \min_{y_i=-1} (\langle w^*, x_i \rangle) \right]$

  ➙ Voir aussi le calcul de b ci-dessus.

- Seuls les α_i correspondant aux **points les plus proches** sont non-nuls.

  ➙ Ce sont les vecteurs supports avec $\alpha_i > 0$.

- Fonction de décision (dont le signe départage les instances) :

$$f(x, \alpha^*, b^*) = \sum_{i \in SV} \underbrace{y_i \alpha_i \langle x_i, x \rangle}_{\langle w^*, x \rangle} + b^* \qquad \text{SV : } \textit{support vector}$$

Addendum : résumé des calculs (suite)

Modus Operandi (calculs) :

- Créer H où $H_{ij} = y_i y_j x_i.x_j$ (pour simplifier les calculs)
- Trouver (à l'aide d'un QP-solver) α tel que :

  maximiser $\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\alpha^T H \alpha$ (venant de L_D)

  sous les contraintes $\alpha_i > 0 \; \forall i$ et $\sum_{i=1}^{n} \alpha_i y_i = 0$

- Calculer $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
- Déterminer l'ensemble SV en trouvant les x_i pour lesquels $\alpha_i > 0$
- Calculer $b = \frac{1}{N_{sv}} \sum_{m \in SV} (y_{sv} - \sum_{m \in SV} \alpha_m y_m x_m.x_{sv})$

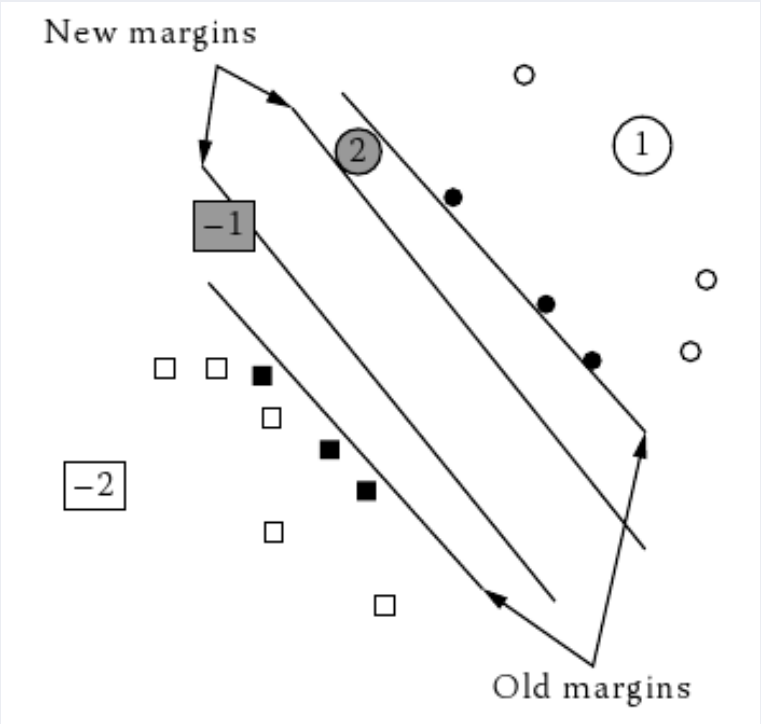
Décision :

- Toute instance x' est classifié par l'évaluation de $y' = signe(w.x' + b)$

Addendum : résumé des calculs (suite)

Remarque : modification de la marge en fonction des nouvelles instances :

➙ Figure : l'ajout de nouvelles instances (-1 et 2 grisées) changent la marge mais l'ajout de 1 ou -2 (non grisées) ne la modifient pas.



Addendum : Remarques et Conclusions sur SVM

• But de SVM : trouver l'hyperplan optimal, qui correspond à la plus grande marge

• Résoudre (facilement) en utilisant la formulation duale (et un QP-solver)

• La solution est creuse (*sparse*) :
le nombre de vecteurs support peut-être très petit en comparaison de la taille de l'ensemble d'apprentissage

• **Seuls les vecteurs supports sont importants** pour la prédiction de futurs exemples.

➙ Tous les autres exemples peuvent être oubliés !

• Classification par SVM :

- Linéairement séparable :
  ➙ **Lagrangien** ou **noyau** (pour faciliter les calculs)
- Non linéairement séparable :
  ➙ Noyau non linéaire
  ➙ Voir aussi SVM non linéaire (non traité dans ce document).

Addendum : Remarques et Conclusions sur SVM (suite)

Raisons des bonnes performances de SVM :

• L'espace des caractéristiques est souvent de très grande dimension.

➙ Pourtant, on n'a pas le problème du "fléau de la dimensionnalité"

→ Alors qu'un classifieur dans un espace de grande dimension a beaucoup de paramètres et est très difficile à estimer.

• Il est démontré (Vapnik) que le problème fondamental ne réside pas dans le nombre de paramètres à estimer, mais plutôt dans la capacité d'un classifieur.

• Typiquement un classifieur avec un grand nombre de paramètres est très flexible, mais il y également quelques exceptions :

Par exemple : soit $x_i = 10^i$ avec $i \in \{1, ..., n\}$.

➙ Le classifieur $y = signe(sin(\alpha(x)))$ peut classifier tous les x_i correctement pour toutes les combinaisons d'étiquetage possibles

➙ Ce classifieur à un-paramètre est très flexible !

Addendum : Remarques et Conclusions sur SVM (suite)

• L'argument de Vapnik est que la capacité d'un classifieur n'est pas caractérisée pas son nombre de paramètres, mais uniquement par sa capacité de discrimination.

➙ Il l'a formalisé par la *dimension de Vapnik=VC* d'un classifieur

La minimisation de $\| w \|^2$ sujette à la condition que la marge géométrique $= 1$ a pour effet de réduire la dimension de VC du classifieur dans l'espace de caractéristiques (EdC).

• La SVM réalise une minimisation du risque structurel :

➙ le risque empirique (erreur d'apprentissage) est minimisé.

• La fonction de perte est analogue à celle de la (*régression Ridge*).

➙ Le terme $\frac{1}{2} \| w \|^2$ réduit les paramètres vers 0 pour éviter le sur-apprentissage.

Addendum : Remarques et Conclusions sur SVM (suite)

SVM avec noyau non linéaire (voir plus loin) : choix du noyau

• Probablement la partie la plus délicate de l'utilisation des SVM

• La fonction noyau doit **maximiser** la similarité parmi les instances d'une même classe et **accentuer** les différences entre classes

• Un grand choix de noyaux a été proposé (Noyau de *Fisher*, Noyau pour chaînes, ...) pour différents types de données

• En pratique un noyau polynomial de degré faible ou un noyau RBF (*Radial Basis Function* : exponentiel) avec une largeur de bande raisonnable est un **bon choix initial** pour commencer.

Addendum : Remarques et Conclusions sur SVM (suite)

A propos d'erreur :

- Principe : laisser de coté une instance qui ne devient pas un vecteur support :
  ➙ il ne changera pas la solution.

- L'erreur d'un SVM dépend du nombre de SV (Support Vector) :
  Soit $nbVS$ représentant le nombre de vecteurs supports obtenus par apprentissage sur n exemples i.i.d. avec $\sim P(x, y)$, et $\mathbb{E}$ l'espérance.

  Alors :

$$\mathbb{E}[P(erreur\ de\ test)] \leq \frac{\mathbb{E}[nbVS]}{n}$$

  où $P(erreur\ de\ test)$ est la valeur espérée du risque, avec l'espérance prise sur l'apprentissage de la SVM sur des ensemble de taille $n-1$.

Addendum : Remarques et Conclusions sur SVM (suite)

Quelques conclusions :

• L'introduction des SVMs a permis de réunir de manière productive des méthodes et des personnes venant de l'informatique, des statistiques et de l'optimisation.

• Une grande force de SVM est sa modularité qui permet de séparer clairement les tâches.

• Bien que cousine de méthodes statistiques plus anciennes, la SVM a permis d'introduire un point de vue radicalement nouveau sur ces méthodes,

➞ En faisant ressortir l'intérêt pratique d'utilisation de systématique de noyaux pour des données très diverses.

➞ L'utilisation généralisée de noyaux pour le traitement de données apparaît finalement comme un principe et un acquis plus fondamental que celui de l'algorithme SVM lui-même.

SVM multi-classes

Rappel : SVM s'applique bien au cas binaire (2 classes).

- Il peut être appliqué, comme les autres méthodes, aux cas multi-classes (multi-réponses) :
- k classes
- Stratégies :
  - Un contre tous
  - Un contre un (par paire)
  - DAG (Graphe orienté)
  - Fonctions objectifs multi-classes

SVM multi-classes (suite)

SVM : **Un contre Tous**

- Apprendre k SVMs $f^1, ..., f^k$ pour séparer chaque classe du reste.
- Fonction de décision :

$$f_{\alpha^j, b^j}(x) = \underset{j}{\text{argmax}}\ g^{(j)}(x) \qquad \text{où} \qquad g^{(j)}(x) = \sum_{i=1}^{n} \alpha_i^j y_i k(x, x_i) + b^j$$

- Pour les points pour lesquels il n'y a pas de fonction de décision $g^{(j)}(x)$ qui l'emporte, on décide :

  ➙ le rejet ou
  ➙ le choix aléatoire de la classe ...

☞ Il y a d'autres méthodes : ../..

SVM multi-classes (suite)

SVM : **Un contre Un** (par paires)

- Apprendre $k(k-1)/2$ SVMs binaires pour chaque paire possible de classes
- Assigner à l'exemple la classe avec le plus de votes
- Pour k classes, on créé $k(k-1)/2$ SVMs binaires

  Exemple : Si 4 classes $\rightarrow$ 6 SVMs binaires

$y_i = +1$	$y_i = -1$	Fonction de décision
classe1	classe2	$f^{12} = \langle w^{12}, x \rangle + b^{12}$
classe1	classe3	$f^{13} = \langle w^{13}, x \rangle + b^{13}$
classe1	classe4	$f^{14} = \langle w^{14}, x \rangle + b^{14}$
....	...	...

Exemple : ../..

SVM multi-classes (suite)

Exemple :

- Soit le tableau des confrontation un-contre-un pour un exemple x_{new} :

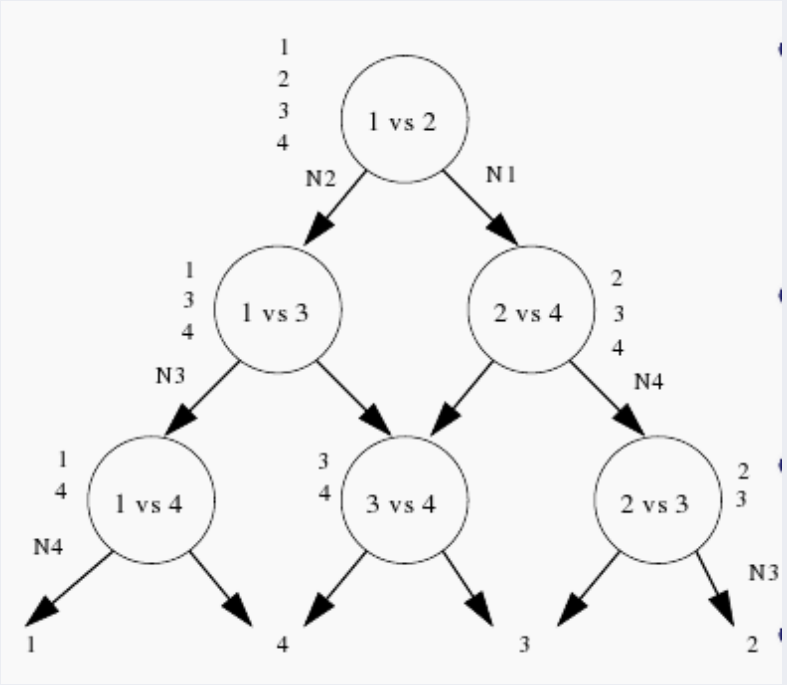
  c_1 vs $c_2 \rightarrow c_1$ l'emporte
  c_1 vs $c_3 \rightarrow c_1$ l'emporte
  c_1 vs $c_4 \rightarrow c_1$ l'emporte
  c_2 vs $c_3 \rightarrow c_2$ l'emporte
  c_2 vs $c_4 \rightarrow c_4$ l'emporte
  c_3 vs $c_4 \rightarrow c_3$ l'emporte

- x_{new} aura 3 votes dans c_1, un dans c, c_3, c_4
  ➙ Sa classe sera c_1.

SVM multi-classes (suite)

SVM : **DAG**

- Semblable à la stratégie 1-contre-1, mais la décision est réalisée au moyen d'un arbre.
- Le graphe possède $k(k-1)/2$ noeuds internes et k feuilles.
- Chaque noeud représente une SVM binaire sur les classe i et j
- Les feuilles indiquent la classe prédite.
- N_i sur un arc : "Non i"



- Fonction de décision (dont le signe départage les instances) :

$$f(x, \alpha^*, b^*) = \sum_{i \in SV} \underbrace{y_i \alpha_i \langle x_i, x \rangle}_{\langle w^*, x \rangle} + b^*$$

SV : support vector

- L'exemple précédent emprunte la descente à gauche.

SVM multi-classes (suite)

Fonctions objectifs multi-classe simplifiée :

- Rappel de la section "Résumé SVM" :

  Fonction de décision (dont le signe départage les instances) :

$$f(x_{new}, \alpha^*, b^*) = \sum_{i \in SV} \underbrace{y_i \alpha_i \langle x_i, x_{new} \rangle}_{\langle w^*, x_{new} \rangle} + b^* \qquad \text{SV : support vector}$$

- On peut donc procéder au calcul :

$$f(x) = w.x_{new} + b = \sum_{i}^{N} \underbrace{y_i \alpha_i \langle x_i, x_{new} \rangle}_{\langle w^*, x_{new} \rangle} + b^*$$

  où N représente le nombre d'instances d'apprentissage, x_{new} la nouvelle instance , y_i la classe (1 ou -1) ;

- Puis de calculer la classes de la nouvelle instance par :

$$prediction(x_{new}) = \underset{1 \leq c \leq M}{\text{argmax}}\ f_c(x)$$

  où M est le nombre total des classes.

SVM multi-classes (suite)

Fonctions objectifs multi-classe (méthode globale) :

• Similaire à la méthode **Un-Contre-Tous**

• Construction de k SVM binaires où la fonction de décision m (spéciale pour la classe m) sépare les exemples d'apprentissage de cette classe (m) des autres.

$$(D) \begin{cases} \min\limits_{w,b,\xi} \dfrac{1}{2}\sum\limits_{r=1}^{k} \parallel w_r \parallel^2 + C/m \sum\limits_{i=1}^{m}\sum\limits_{r\neq y_i} \xi_i^r \\ s.t. \quad \langle w_{y_i}\phi(x_i)\rangle + b_y \geq \langle w_r\phi(x_i)\rangle + b_r + -\xi_i^r \\ \qquad \xi_i^r \geq 0 \end{cases}$$

Où $r \in \{1, ..., k\} \setminus y_i$ et $y_i \in \{1, ..., k\}$ (classe de x_i)

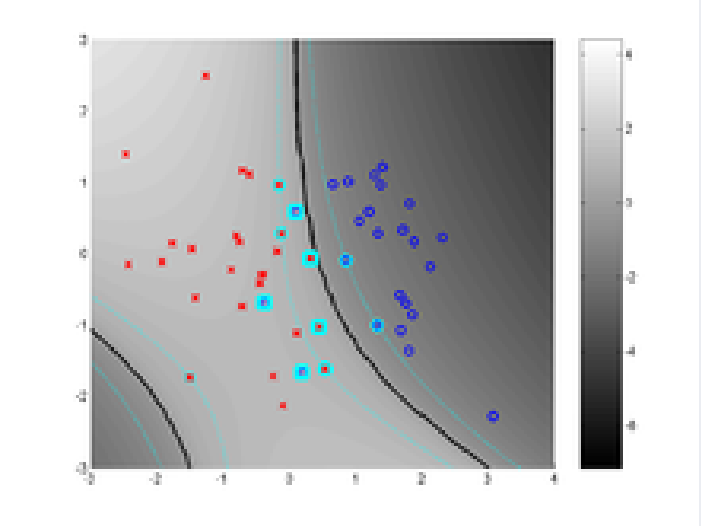
$$y_i = \arg\max_{i=1...k} (w^i)^T \phi(x) + b^i$$

SVM non linéaire

- Nous avons étudié le cas de noyaux linéaires ci-dessus
- Pour certains problèmes, il est besoin de kernel plus sophistiqué.

- Exemple d'une classification avec une SVM non linéaire.
- La frontière en noir sépare l'espace d'entrée,

  Les vecteurs supports sont les cercles bleus clairs.



SVM non linéaire (suite)

• Cette variante de SVM est appelée *SVM souple*.

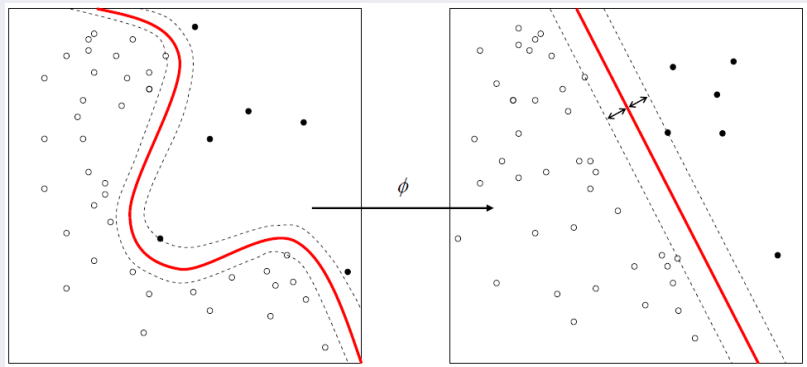
• L'idée (appelée *Kernel Trick*) est d'appliquer un noyau non linéaire à l'hyperplan de marge maximisée.

• La méthode reste similaire à ci-dessus (Duale) excepté que le noyau linéaire (chaque produit scalaire) utilisé ci-dessus est remplacé par une fonction de noyau non linéaire.

➙ Par conséquent, l'hyperplan à marge maximum est projeté dans l'espace des caractéristiques (EdC).

• La projection peut donc être non linéaire et l'EdC dans lequel se place l'hyperplan trouvé de dimension supérieure.

➙ Dans ce cas, l'hyperplan correspondant dans l'espace d'origine aura été non linéaire.



SVM non linéaire (suite)

• Les kernels utilisés sont les mêmes que ci-dessus (polynomiale, Gaussienne, RBF, Tanh, &c.).

• Rappelons que le kernel transformera par $\phi(.)$ via $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$.

➙ w sera également projeté dans l'EdC et devient $w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$

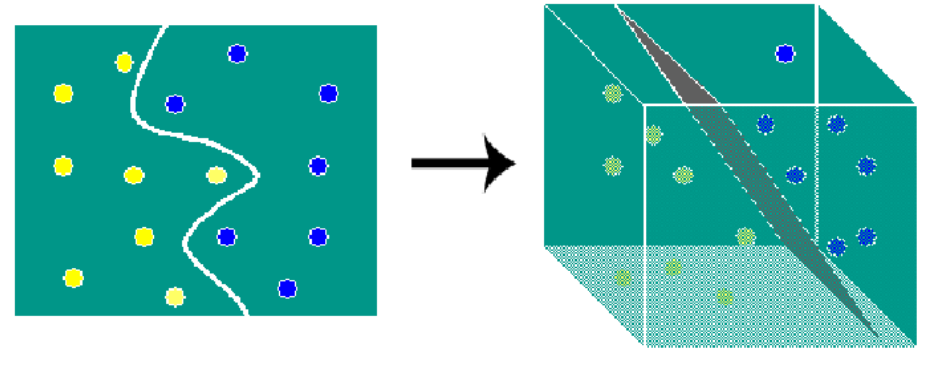
➙ De même, un produit tel que $w \cdot x$ deviendra

$$w \cdot \phi(x) = \sum_{i=1}^{n} \alpha_i y_i \ k(x_i, x)$$

sans garantir que $\quad \exists w^* | w \cdot \phi(x) = k(w^*, x)$

SVM non linéaire : exemples

- On utilise un noyau pour trouver une projection dans un espace séparable.
  - ➙ La projection a lieu dans l'espace des caractéristiques (attribut des données).



  - ➙ On constate que la projection (re-description) peut conduire à un espace de séparation *linéaire* mais de plus grande dimension .

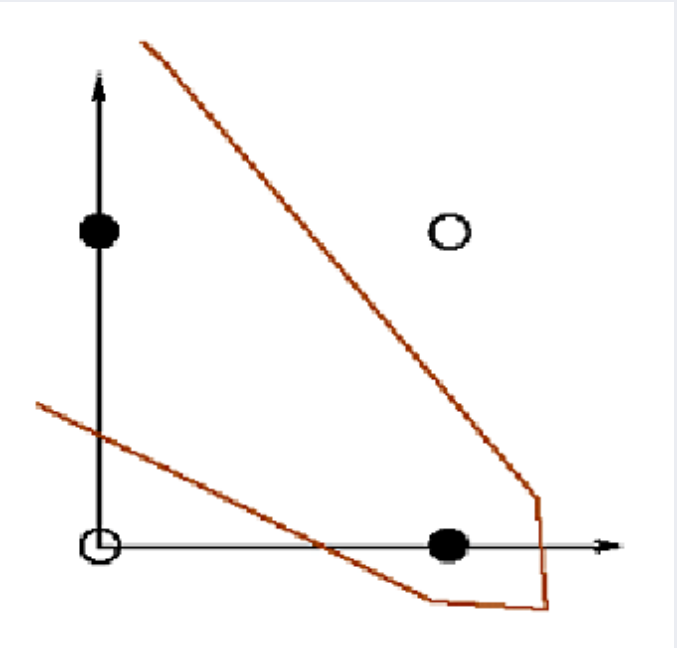
☞ **On peut utiliser** les noyaux même dans les cas linéaires pour simplifier la classification
  - ➙ La projection a souvent lieu dans $\mathbb{R}$ (facilité des calculs).

SVM non linéaire : exemples (suite)

Exemple : on reprend le problème XOR

- Les coordonnées des points :
  (0,0) ; (0,1) ; (1,0) ; (1,1)
- Le cas de XOR n'est pas linéairement séparable.

  ➙ Si on place les points dans un plan à deux dimension, on obtient l'image →

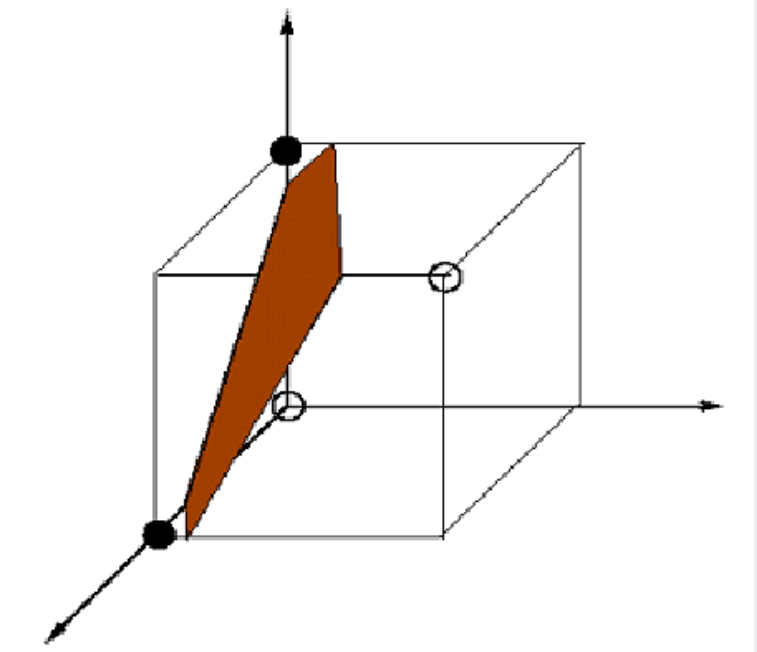


- On prend un noyau polynomiale : $(x, y) \rightarrow (x, y, x.y)$
  ➙ Fait passer de $\mathbb{R}^2$ à $\mathbb{R}^3$

SVM non linéaire : exemples (suite)

• On obtient un problème en 3 dimensions linéairement séparable :

$(0,0) \rightarrow (0,0,0)$
$(0,1) \rightarrow (0,1,0)$
$(1,0) \rightarrow (1,0,0)$
$(1,1) \rightarrow (1,1,1)$



• On avait vu une solution dans $\mathbb{R}^6$

Bilan des méthodes supervisées

Méthodes d'apprentissage supervisé

- Rappel des objectifs de l'apprentissage supervisé :

  ☞ A partir d'un échantillon d'apprentissage $S = \{(x_i, y_i)\}$ avec y_i la classe de l'instance x_i,

  ☞ Chercher une loi de dépendance sous-jacente telle que

  ➙ une fonction h (hypothèse) la plus proche possible de la fonction cible f (la vraie telle que $y_i = f(x_i)$)

  ➙ Ou une loi (distribution) de probabilités $Pr(x_i, y_i)$

  ☞ Afin de prédire l'avenir !

...

Bilan des méthodes supervisées (suite)

La recherche d'une loi de dépendance sous-jacente telle qu'une fonction h (hypothèse) la plus proche possible de la fonction cible f

Revient à :

rechercher des régularités sous-jacentes sur un ensemble d'apprentissage d

- sous forme d'une fonction :
  - Si f est une fonction continue : Régression, estimation de densité
  - Si f est une fonction discrète : classification
  - Si f est une fonction binaire (booléenne) : apprentissage de concepts
- sous forme d'un nuage de points (e.g. mixture de gaussiennes)
- sous forme d'un modèle complexe (e.g. réseau Bayésien)

➙ Afin de résumer, détecter des régularités, comprendre, ...

IBL

• IBL : on n'a pas un espace d'hypothèse (à vérifier, apprendre, induire, ... cf. SVM) et on raisonne par cas ⇒ Stockage des instances

• Fonction de **distance** pour déterminer la classe d'une nouvelle instance
  ◦ La fonction de distance *définit ce qui est appris*
  ◦ Une fonction de distance simple si les attributs numériques.

Distance : il y ait beaucoup de choix :
  ◦ La plupart des schémas *IBL* utilise une **distance Euclidienne**.

• La distance entre les instances avec des valeurs de k attributs :
  1^e instance $a^{(1)} : a_1^{(1)}, a_2^{(1)}, \ldots a_k^{(1)}$ 2^e instance $a^{(2)} : a_1^{(2)}, a_2^{(2)}, \ldots a_k^{(2)}$

  ➙ La **distance Euclidienne** :

$$\sqrt{\left(a_1^{(1)} - a_1^{(2)}\right)^2 + \left(a_2^{(1)} - a_2^{(2)}\right)^2 + \ldots + \left(a_k^{(1)} - a_k^{(2)}\right)^2}$$

☞ Quand on compare les distances, la racine carré est inutile

IBL (suite)

- Une alternative à la distance Euclidienne
  - ➛ La métrique **"Manhattan"** ou "city-block" :
  - ➛ Simple addition des différences (pas au carré)

$$\boxed{\left(a_1^{(1)} - a_1^{(2)}\right) + \left(a_2^{(1)} - a_2^{(2)}\right) + \ldots + \left(a_k^{(1)} - a_k^{(2)}\right)}$$

  - ➛ Augmente l'influence des grandes différences au détriment des petites.
- En générale, la distance Euclidienne représente **un bon compromis.**

IBL (suite)

- **Choix de la métrique** : qu'est-ce ?
  - ➛ Le sens de la distance qui sépare 2 instances :
    - ➛ Que veut dire le double de cette distance, etc. ?

- Différents attributs sont mesurés à différentes échelles.
  - ➛ Si distance Euclidienne → l'effet de certains attributs peut être complètement inhibé par d'autres qui ont une échelle de mesure plus large.
  - ➛ Solution : **normaliser** tous les attributs (ramenés entre 0 et 1) en calculant :

$$a_i = \frac{v_i - \ min\ v}{max\ v - \ min\ v}$$

avec v_i et la valeur de l'attribut i,
et min et max de l'ensemble des instances.

IBL (suite)

Distance (suite) : autres cas

- Cas d'attributs **nominaux** :
  - ➡ la distance = 1 si deux valeurs différentes
  - ➡ la distance = 0 si deux valeurs identiques
    - ➛ Pas de mise à échelle nécessaire (on manipule des 0 et des 1)
  - ➡ Si **valeurs manquantes** → attribuer le maximum de la distance (après normalisation) :
  - ➛ Ou une valeur très différente de toute autre valeur d'attribut
    - → distance = **1** si l'une (ou les 2) manquantes (cf. 2 val. $\neq$)
    - → distance = **0** si les 2 sont non manquantes et identiques.
- Cas d'attributs numériques (après normalisation) :
  - ➡ La différence entre 2 valeurs manquantes = 1.
  - ➡ Si une des valeurs est manquante,
    - → la différence = l'autre valeur V (normalisée)
    - → ou $= (1 - V)$, quelque soit la valeur.
    - → i.e. si val. manquantes, la différence est aussi large que possible.

Plus Proches Voisins (NN)

Plus Proche(s) Voisin(s) :

Apprentissage à base d'instances avec le **1-NN** (*Un-plus proche voisin*) :
- ➛ (cf. distance ci-dessus) : calcul simple et juste mais lente.

- On doit examiner **tout l'ensemble d'apprentissage** pour une prédiction
- Tous les attributs sont supposés d'égale importance (comme pour Bayes)
- Une solution : sélection d'attribut prépondérant ou affectation de poids
- Pour les données erronées (bruitées) :
  - ➛ Enlever les instances "parasitées"
    - ➛ = choix de "bons" exemples (difficile !)
  - ➛ Utiliser un vote majoritaire sur les K-plus proches voisins (e.g. K=5)
- **K-plus proches voisins** utilisée depuis 1950s ... (en Stat et en PR)

Un exemple : la cueillette de champignons

• Un groupe d'amis se propose d'aller ramasser des champignons le week-end prochain.
Mais seules de bonnes conditions météorologiques dans la journée qui précède favorisent leur croissance.

• Ils décident analyser les cueillettes des années précédentes pour savoir si le ramassage permettra ou non une fricassée au dîner.

➛ Ils disposent de l'agenda suivant :

		Temps	Humidité	Vent	**Bonne cueillette** (classe)
E1	WE 1, année-1	nuageux	haute	oui	oui
E2	WE 2, année-1	nuageux	basse	non	non
E3	WE 3, année-1	nuageux	basse	oui	non
E4	WE 1, année-2	soleil	haute	oui	oui
E5	WE 2, année-2	pluvieux	basse	oui	non
E6	WE 3, année-2	nuageux	haute	non	oui
E7	WE 4, année-2	soleil	basse	non	non

TABLE 2: Annales champignons

On sait : le week-end prochain sera plutôt **ensoleillé, humide et sans vent**.

Un exemple : la cueillette de champignons (suite)

Déroulement :

➙ Les trois colonnes (caractéristiques météorologiques) sont des attributs, la dernière colonne, la **classe**.

		Temps	Humidité	Vent	**Bonne cueillette** (classe)
E1	WE 1, année-1	nuageux	haute	oui	oui
E2	WE 2, année-1	nuageux	basse	non	non
E3	WE 3, année-1	nuageux	basse	oui	non
E4	WE 1, année-2	soleil	haute	oui	oui
E5	WE 2, année-2	pluvieux	basse	oui	non
E6	WE 3, année-2	nuageux	haute	non	oui
E7	WE 4, année-2	soleil	basse	non	non

Etape 1 : Choisir une distance sur chacun des attributs.

• Ici, c'est un cas de données catégorielles.

➙ Pour chacun des attributs a, on choisit :

$$\begin{cases} d(a,a) = 0 \\ d(a,a') = 1 \qquad a \neq a' \end{cases}$$

Un exemple : la cueillette de champignons (suite)

Etape 2 :

On utilise la distance *Euclidienne* pour calculer les voisins de notre dimanche.

Soit le prochain WE représenté par l'événement

E8 : "Soleil, haute, non".

Distance entre $E8$ et la BD :

$d(E8, E1) = \sqrt{1+0+1} = \sqrt{2}$
$d(E8, E2) = \sqrt{1+1+0} = \sqrt{2}$
$d(E8, E3) = \sqrt{1+1+1} = \sqrt{3}$
$d(E8, E4) = \sqrt{0+0+1} = 1$
$d(E8, E5) = \sqrt{1+1+1} = \sqrt{3}$
$d(E8, E6) = \sqrt{1+0+0} = 1$
$d(E8, E7) = \sqrt{0+1+0} = 1$ ↗

Les voisins sont, dans l'ordre de proximité à $E8$:

$(E4 - oui, E6 - oui, E7 - non)$	$d = 1$
$(E1 - oui, E2 - non)$	$d = \sqrt{2}$
$(E3 - non, E5 - non)$	$d = \sqrt{3}$

Un exemple : la cueillette de champignons (suite)

Etape 3 : k = nbr. de voisins :

Ira-t-on cueillir des champignons si k = 3, puis k = 4, k = 5 ?

- Si k = 3, on retient E4, E6 et E7, donc la cueillette sera bonne.
- Si k = 4, on retient E4, E6, E7 et E1
  - ➙ Pas de problème, la cueillette sera bonne.
  - ➙ Mais on aurait pu retenir E2 plutôt que E1 (même distance à E8).
  - ➙ Si l'on retient E4, E6, E7 et E2,
    - ➛ pas de conclusion (car deux oui et deux non).
- Si k = 5, on retient E4, E6, E7, E1 et E2, donc la cueillette sera bonne.

Remarques : on prendre en général un nombre de voisins dont *le modulo du nombre de classes* (ici 2) n'est pas égal à zéro.

☞ La val. k et l'ordre de choix des voisins influent sur la détermination de la classe.

N.B. : une fois un voisin jugé "proche", la valeur de la distance est oubliée et on n'utilise que le nombre de oui/non.

Un exemple : la cueillette de champignons (suite)

Etape 4 : pondérer le poids de chacun des voisins par la distance ?

➤ On doit cette fois pondérer la classe de chacun des k-voisins par sa distance à l'événement E8.

	E1	E2	E3	E4	E5	E6	E7
Distance à E8	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	1	$\sqrt{3}$	1	1
Classe (connue)	oui	non	non	oui	non	oui	non

- Pour k = 3, la classe de E8 est alors :
  $1/1 * classe(E4) + 1/1 * classe(E6) + 1/1 * classe(E7)$
  ➤ = (2 * oui) + (1 * non)
  ➤ donc la cueillette sera bonne.

N.B. : à l'étape 3, on a simplement compté le nbr. de oui et de non ; un oui/non par voisin, quelque fut la distance.

Un exemple : la cueillette de champignons (suite)

- Pour k = 4, la classe de E8 est alors soit :

  $1/1 * classe(E4) + 1/1 * classe(E6) + 1/1 * classe(E7) + 1/\sqrt{2} * classe(E1)$

  ➛ = $((2\sqrt{2}+1)/\sqrt{2}$ * oui) + (1 * non) ➛ la cueillette sera **bonne**.

Soit : $1/1 * classe(E4) + 1/1 * classe(E6) + 1/1 * classe(E7) + 1/\sqrt{2} * classe(E2)$

  ➛ = (2 * oui) + $(\sqrt{2}+1)/\sqrt{2}$ * non) ➛ la cueillette sera **bonne**.

- Pour k = 5, la classe de E8 est alors :

  $1/1 * classe(E4) + 1/1 * classe(E6) + 1/1 * classe(E7)$
  $+1/\sqrt{2} * classe(E1) + 1/\sqrt{2} * classe(E2)$

  ➛ = $((2\sqrt{2}+1)/\sqrt{2}$ * oui) + $(\sqrt{2}+1)/\sqrt{2}$ * non)

  ➛ donc la cueillette sera **bonne**.

Addendum : Recherche efficace des KNN

- IBL est une méthode simple mais lente
  - ➡ Il faut comparer l'instance inconnue à <u>toutes</u> les instances
- **Une solution** : représenter les instances comme un arbre binaire

  → **KD-tree**

  - ➢ K : nombre d'attributs, D : division
  - ➢ On divise l'espace (des entrées) avec un hyperplan et ainsi récursivement
  - ➢ Les divisions sont parallèles à un des deux axes du plan (verticale ou horizontale, dans le cas à 2 dimensions)
  - ➢ Il faut un espace à k dimensions (k = le nombre d'attributs)

Addendum : Recherche efficace des KNN (suite)

Un exemple avec k=2

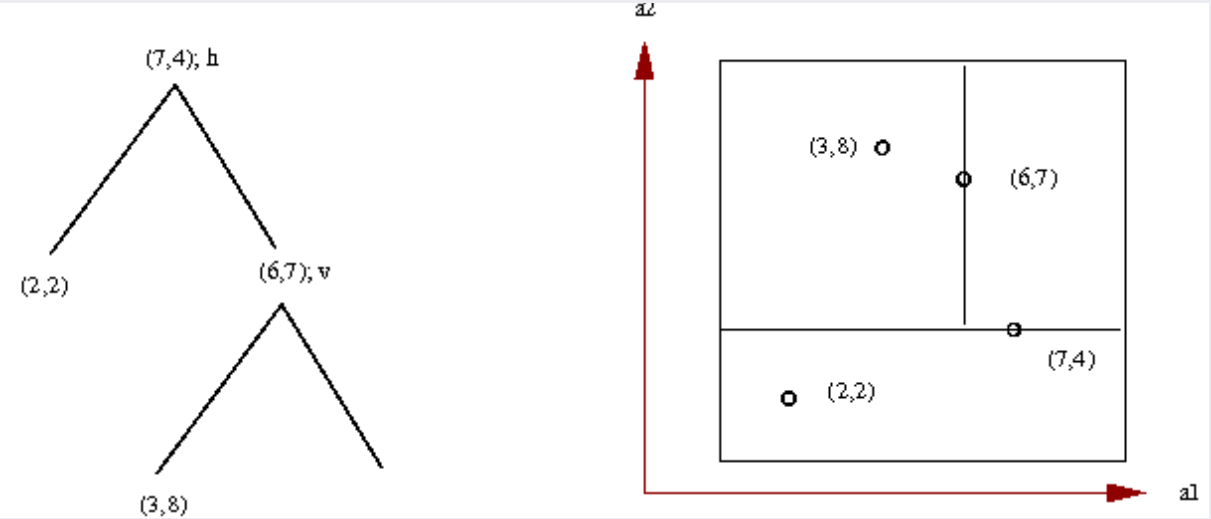


FIGURE 15: La double représentation d'un KD-tree (h=horizontal, v=vertical)

- La figure contient 4 instances à 2 attributs sous forme (a_1, a_2)
  - ➛ (a_1, a_2) donne les valeurs sur les axes X et Y du repère.
- Les hyperplans ne sont pas des frontières
  - ➛ Les décisions seront prises selon les plus-proches-voisins
- Chaque région peut contenir de 0 à plusieurs instances

Addendum : Recherche efficace des KNN (suite)

- **Les questions soulevées** :

  Recherche efficace du plus proche voisin, **Création** du KD-Tree, **Ajout** de nouvelles instances ?

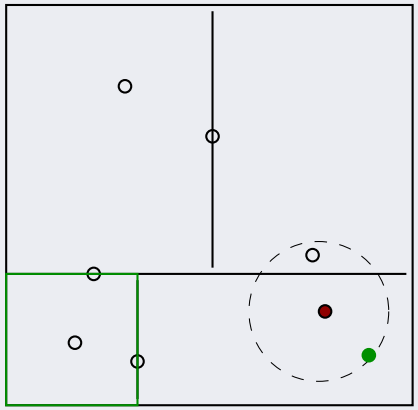


FIGURE 16: Exemple de KD-tree et la recherche du plus proche voisin de l'instance rouge

- On recherche l'instance rouge.
- L'instance verte est la première approximation trouvée (dans sa région)
- On cherche ensuite le PPV dans le cercle du centre rouge et de rayon vert
  - ➛ Le rond.
  - ➛ La recherche est guidée et restreinte à certaines régions (cf. ABOH)

Addendum : Recherche efficace des KNN (suite)

Structure de données (et recherche) plus efficace

- La notion de (hyper)rectangle pose problèmes (cf. instances dans les "coins")
- Une meilleure structure : des cercles au lieu de rectangles.
  - ➛ Des hyper-shpères pour k dimensions (à la place d'hyper-rectangles)
  - ➛ Chaque cercle est appelé **ball** (défini par au moins 2 instances)

- Le principe de la recherche est le même que pour un KD-Tree

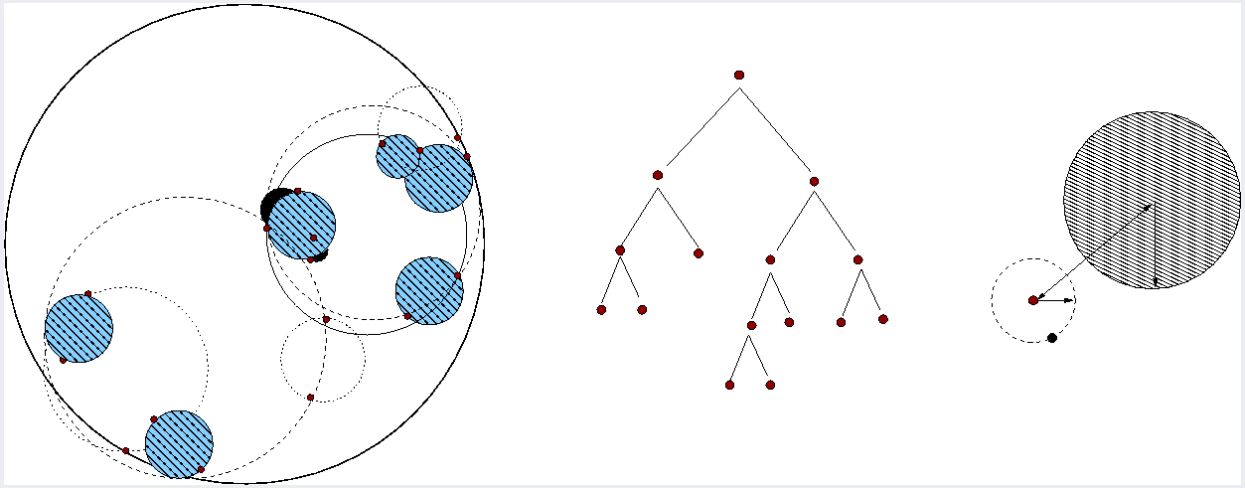


FIGURE 17: Le **Ball-Tree** présenté par une hypersphère, le **KD-Tree** correspondant et un exemple de **Recherche**

Remarques sur la méthode IBL

- **Intérêt principal** : ajout des instances au fur et à mesure
  - ➛ Techniques de rééquilibrage (si $hauteur > 2\ log\ N$)
- Dans les exemples ci-dessus, les attributs ont la même importance
- Problèmes des données bruitées
  - ➛ Extension à K-PPV (au lieu de 1-PPV des exemples)
  - ➛ Souvent $K = 5$, **Vote** à la majorité !
  - ➛ Autre solution : choix des instances à retenir (v. Chap 6)
- Si $lim_{N \to \infty} \frac{K}{N} = 0$ (K voisins), la probabilité de l'erreur tend vers le minimum théorique
- KD-Tree est une ancienne technique,
  - ➛ *Ball (ou Metric)-Tree* assez récente, capable de gérer de grandes dimensions
- Variante : regroupements en régions selon des partitions de valeurs d'attributs

Remarques sur les méthodes

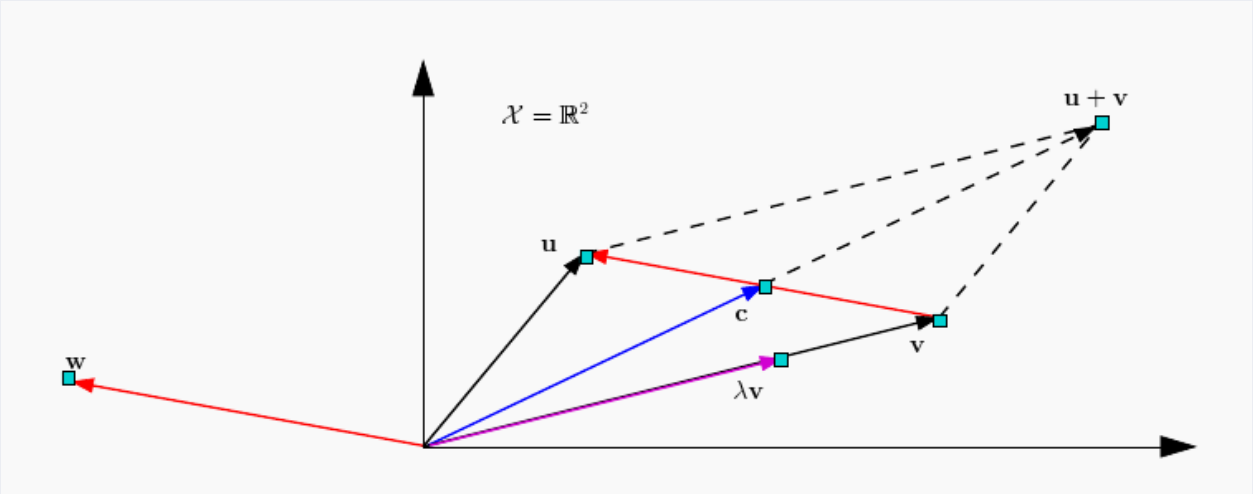
- La méthode **1R** : pour montrer les structures simples (peu utilisée en pratique !)
- La méthode de **Bayes** (philosophe du 18e) :
  - "Résoudre un problème en tenant compte des chances"
  - La difficulté : estimation des probabilités *a priori* (e.g *Pr[H]* dans "météo")
  - Amélioration → *intervalles de confiance* (cf. chap. svt.)
  - Une extension de la méthode Bayesienne :
  - les *Réseaux Bayesiens* (prise en compte des dépendances)
- Arbres de décision (**ID3, C4.5**) : notion d'entropie :
  - traduction directe en règles
- **PRISM** et règles de classification (covering)
- **A PRIORI** : l'algorithme pour les règles d'association

Remarques sur les méthodes (suite)

- **Méthodes linéaires**
  - Une limitation : par ex., ne peuvent pas apprendre XOr (Minsky-Apt 1969)
    → Mais leur combinaison le peut ! (cf. Réseaux de Neurones)
  - Notion de "seuil linéaire" (linear threshold unit)
    → C'est un test binaire pour savoir si l'expression linéaire est plus grande (ou petite) que zéro (comme dans la classification "par paire", Régression multi-réponses).
  - La notion de "linear machine" = un ensembles de fonctions linéaires
  - Les classifieurs multi-réponses utilisés dans le schéma **stacking** :
    → combiner les sorties de plusieurs systèmes d'apprentissage (v. chap 7)
- SVM : une des meilleurs méthodes. Applicable au cas multi-classes.
- La méthode du **PPV** dans l'apprentissage à base d'instances :
  Combinée avec l'élagage d'instances bruitées (parasitées)
  + la pondération d'attributs
  → donne des résultats intéressants (voir chapitre 6)

Addendum : rappels sur les vecteurs

A propos de l'erreur (l'écart)



u, v, w, c sont des vecteurs

$w = u - v$

$c = 1/2(u + v)$

Ici $0 < \lambda < 1$

Addendum : rappels sur les vecteurs (suite)

Produit scalaire $\langle ., . \rangle : \chi \times \chi \to \mathbb{R}$: (dit également dot-product : $v.u$)

Si $u = [u_1, u_2, ..., u_n]$ et $v = [v_1, v_2, ..., v_n]$

➡ $u.v = \sum_i u_i v_i$

- symétrique : $\langle u, v \rangle = \langle v, u \rangle$
- bi-linéaire : $\langle \lambda u_1 + \gamma u_2, v \rangle = \lambda \langle u_1, v \rangle + \gamma \langle u_2, v \rangle$
- positif : $\langle u, u \rangle \geq 0$
- défini : $\langle u, u \rangle = 0 \Rightarrow u = 0$

Un produit scalaire

- Donne une structure à χ
- Peut être vu comme une "similarité"
- défini une norme $\| . \|$ sur χ : $\sqrt{\langle u, u \rangle}$

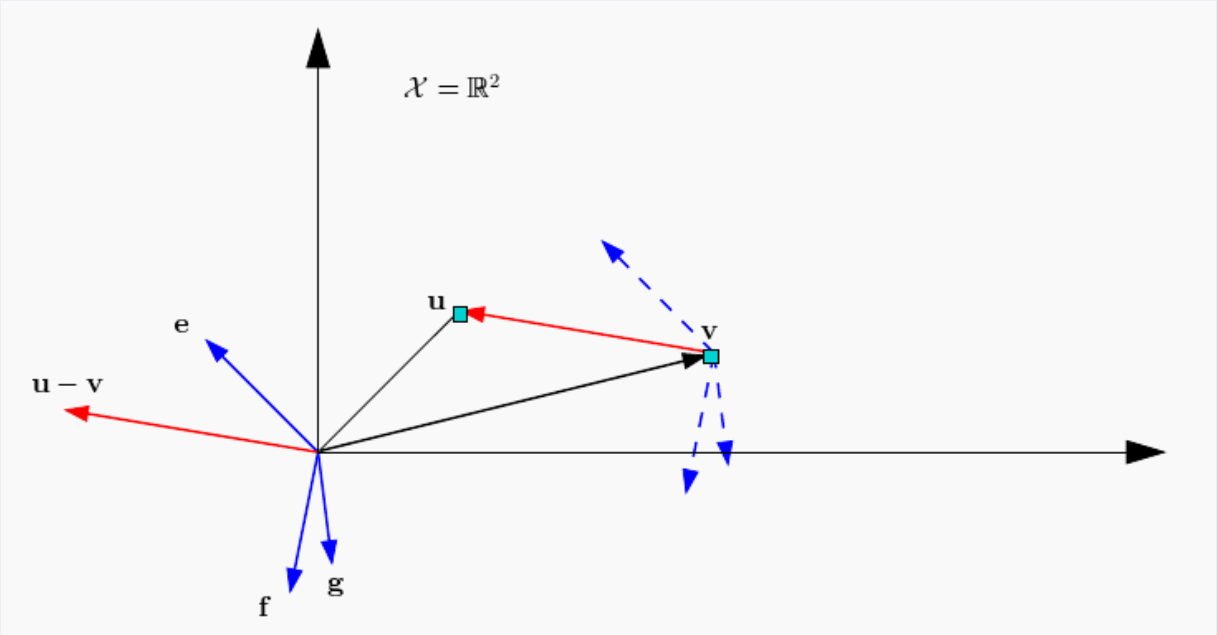
Addendum : rappels sur les vecteurs (suite)

Rappels sur la norme $\| . \|$:

Soit A et B deux points de l'espace, $\| \vec{AB} \|$ = la distance de A à B.

- La norme d'un vecteur est un réel positif.
- $\| \vec{0} \| = 0$
- Un vecteur unitaire est un vecteur de norme 1.
- Pour tout réel k et tout vecteur $\vec{u}$: $\| \vec{ku} \| = |k| . \| \vec{u} \|$
- Pour tous vecteurs $\vec{u}, \vec{v}$, $\| \vec{uv} \| \leq \| \vec{u} \| + \| \vec{v} \|$ (inégalité triangulaire).
- Dans un repère orthonormé du plan $(O, \vec{i}, \vec{j})$, si $\vec{u}(x, y)$, alors $\| \vec{u} \| = \sqrt{u.u} = \sqrt{x^2 + y^2}$
- De même dans l'espace 3D...

Addendum : rappels sur les vecteurs (suite)



- $\langle u - v, e \rangle > 0$: $u - v$ et e pointent dans la même direction
- $\langle u - v, f \rangle = 0$: $u - v$ et f sont orthogonaux
- $\langle u - v, g \rangle < 0$: $u - v$ et g pointent dans des directions opposées

Addendum : compléments sur les matrices

• En multiplication de matrices, il y a deux formes de multiplication qui sont (*produit scalaire* notée $A.B$ et) *produit vectoriel* (noté $< A, B >$) et *outer-product* (*produit tensoriel* noté $A \otimes B$), voir les figure ci-dessous.

• Par définition, le *produit vectoriel* et le *produit scalaire* peuvent être exprimé via une multiplication matricielle : $A.B = A^T B$ (on exprime dot en fonction de mult-matrices)

➙De même, $< A, B >= A^T.B = \sum A_i B_i$ (pour A et B deux vecteurs).

• *produit vectoriel* ($< A, B >= A^T.B$) tout comme *produit scalaire* produisent un scalaire.

➙ Il s'agit de réaliser le produit entre deux vecteurs $1 \times n$ et $n \times 1$ donnant la matrice (le scalaire) 1×1.

• Par contre, *outer product* $A \otimes B = A.B^T$ produit une matrice.

➙ Il s'agit de réaliser le produit entre deux vecteurs $m \times 1$ et $1 \times n$ donnant la matrice $m \times n$.

☞ Le *produit vectoriel* est **la trace** du *outer product* ($< A, B >= tr(A^T.B)$)

Addendum : compléments sur les matrices (suite)

• NB : l'important est de savoir si on a besoin d'un scalaire ou d'une matrice, le reste (inner, outer, dot) sont des formes de multiplications de matrice.

• N.B. : Quand on écrit $< A, B >$, on suppose que le nombre ligne/col correspondent pour que la multiplication puisse avoir lieu.

• La multiplication matricielle est une généralisation de *inner* et *outer* : il suffit de considérer p.ex. chaque ligne de la matrice et de la renommer pour voir la matrice comme un vecteur et donc tomber dans le cas inner/outer.

De même, on peut voir les cols de la matrice comme un tout et donc idem...

produit vectoriel $< A.B >= A^T\ B$

$$A \cdot B = A^T B = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \end{bmatrix}.$$

produit tensoriel $A \otimes B = A\ B^T$

$$AB^T = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n \end{bmatrix}.$$

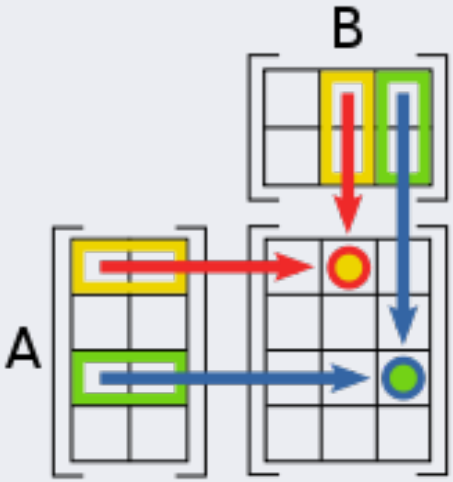
Addendum : compléments sur les matrices (suite)

Multiplication de matrices (généralisation de inner/outer)

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix}$$

$$A_i = \begin{bmatrix} a_{i,1} & a_{i,2} & \cdots & a_{i,n} \end{bmatrix}$$

$$AB = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} \begin{bmatrix} B_1 & B_2 & \dots & B_p \end{bmatrix} = \begin{bmatrix} (A_1 \cdot B_1) & (A_1 \cdot B_2) & \dots & (A_1 \cdot B_p) \\ (A_2 \cdot B_1) & (A_2 \cdot B_2) & \dots & (A_2 \cdot B_p) \\ \vdots & \vdots & \ddots & \vdots \\ (A_m \cdot B_1) & (A_m \cdot B_2) & \dots & (A_m \cdot B_p) \end{bmatrix}.$$



$$c_{12} = \sum_{r=1}^{2} a_{1r} b_{r2} = a_{11} b_{12} + a_{12} b_{22}$$

$$c_{33} = \sum_{r=1}^{2} a_{3r} b_{r3} = a_{31} b_{13} + a_{32} b_{23}$$

Addendum : compléments sur les matrices (suite)

Trace :

• En Algèbre linéaire, *la trace d'une matrice* carrée A est la somme des éléments de la diagonale principale de A :

➙ $\text{tr}(A) = a_{11} + a_{22} + \cdots + a_{nn} = \sum_{i=1}^{n} a_{ii}$

• On a :

- $\text{tr}(X^{\text{t}}\, Y) = \text{tr}(XY^{\text{t}}) = \text{tr}(\, Y^{\text{t}} X) = \text{tr}(\, YX^{\text{t}}) = \sum_{i,j} X_{i,j}\, Y_{i,j}$.
- $\text{tr}(AB) = \text{tr}(BA)$.
- $\text{tr}(ABCD) = \text{tr}(BCDA) = \text{tr}(CDAB) = \text{tr}(DABC)$ (perm. cyclique)
  - ➙ Mais $\text{tr}(ABC) \neq \text{tr}(ACB)$. (non cyclique)
- $\text{tr}(X \otimes Y) = \text{tr}(X)\text{tr}(\, Y)$.
- $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$,
- $\text{tr}(cA) = c \cdot \text{tr}(A)$,
- $\text{tr}(AB) = \text{tr}(BA)$
- Invariant de similarité :
  $\text{tr}(P^{-1}AP) = \text{tr}(P^{-1}(AP)) = \text{tr}((AP)P^{-1}) = \text{tr}(A(PP^{-1})) = \text{tr}(A)$.
- La trace d'une matrice de projection est la dimension de l'espace cible :
  - ➙ Si $P_X = X\left(X^T X\right)^{-1} X^T$, alors $\text{tr}\left(P_X\right) = \text{rank}\left(X\right)$.

Addendum : compléments sur les matrices (suite)

- **Produit interne Frobenious (inner product)** :
  - ➙ Le *Frobenious inner product* est un produit interne sur les éléments des matrices (comme si on avait des vecteurs à la place de matrices) noté :

$$A : B = \sum_i \sum_j A_{ij}\ B_{ij} = trace(A^T B) = trace(A B^T)$$

  - ➙ De fait, on a $||A||_F = \sqrt{\sum_i^m \sum_j^n a_{ij}^2}$

- **Dérivée** de matrices (utilisée dans cette partie) :
  - ➙ Formule générale : $\frac{\partial}{\partial s}(x - As)^T M(x - As) = -2A^T\ M\ (x - As)$

    avec dans notre cas la dérivée de $(y - Xw)^T(y - Xw)$ sur w
  - ➙ dans notre cas $M = Id.$
  - ➙ On trouve bien pour notre cas le résultat :

$$-2X^T(y - Xw) = -2X^T y + 2X^T Xw$$

Addendum : compléments sur les matrices (suite)

A propos des matrices inverses :

Certaines des propriétés des matrices inverses sont aussi vérifiées par les matrices pseudo-inverses qui peuvent être définies pour n'importe quelle matrice, même pour celles qui ne sont pas carrées (donc *singulières*).

Au cas où la matrice X n'est pas carrée, il est possible d'inverser grâce à une pré multiplication par le groupe de matrices $(X^T X)^{-1} X^T$, ou une post-multiplication par $X^T (X X^T)^{-1}$, car :

$$(X^T X)^{-1} X^T X = I, \qquad \text{et} \qquad X X^T (X X^T)^{-1} = I$$

- Ces opérations s'appuient sur la définition de l'inverse :
  ➞ *A est inversible* est équivalent à
  - *le système homogène* $AX = 0$ *a pour unique solution* $X = 0$,
  - *la matrice A est inversible à gauche*, c'est-à-dire qu'il existe une matrice B carrée d'ordre n telle que $BA = I_n$,
  - *la matrice A est inversible à droite*, c'est-à-dire qu'il existe une matrice B carrée d'ordre n telle que $AB = I_n$.

Rappel : la matrice unité : des 1 sur la diagonale et 0 ailleurs.

Addendum : Rappels Sur l'espace vectoriel

Remarques sur le produit vectoriel (*inner product*) et le produit scalaire (*dot product*)

• D'une manière générale, un espace d'*inner product* (espace PréHilbertien) est un espace vectoriel muni du produit vectoriel :

$$< . , . > : V \times V \to \mathbb{R} \text{ (ou } \mathbb{C}) \qquad \text{On s'intéresse ici à } \mathbb{R}$$

Du produit vectoriel au produit scalaire :

• Un espace PréHilbertien est un espace vectoriel muni d'une structure qui associe un scalaire à une paire de vecteurs (le produit vectoriel des 2 vecteurs $< ., . >$).

• Le produit vectoriel permet de définir la longueur d'un vecteur ($||V||$) et l'angle θ entre 2 vecteurs.

➡ Ce qui permet de définir l'orthogonalité de 2 vecteurs lorsque $\theta = 0$.

Addendum : Rappels Sur l'espace vectoriel (suite)

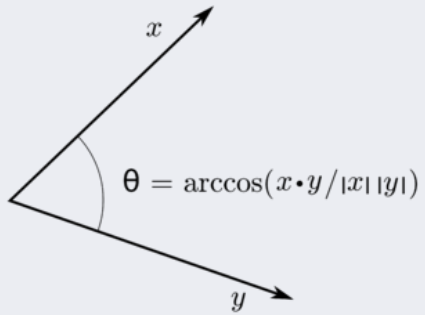
• La *Norme* $||a||$ d'un vecteur a dans un tel espace de *produits vectoriels* est défini par : $||a|| = \sqrt{< a, a >}$

➙ généralise la longueur et l'angle θ entre 2 vecteurs a et b :

$$cos\theta = \frac{< a, b >}{||a||\ ||b||} \quad \Rightarrow \quad < a, b >= ||a||\ ||b||\ cos\theta$$

➙ En particulier, 2 vecteurs a et b sont considérés **orthogonaux** si leur *produit vectoriel* est nul ➙ $< a, b >= 0$ (car $cos(90) = 0$)

➙ L'interprétation géométrique de l'angle entre 2 vecteurs utilisant un produit vectoriel :



Addendum : Rappels Sur l'espace vectoriel (suite)

• Un espace PréHilbertien généralise l'espace **Euclidien** dans lequel le produit vectoriel (de l'espace PréHilbertien) est appelé le produit scalaire.

• Produit scalaire : de 2 vecteurs $a = [a_1, ..., a_n]$ et $b = [b_1, ..., b_n]$

est $$a.b = \sum_{i=1}^{n} a_i\, b_i = a_1\, b_1 + a_2\, b_2 + ...a_n\, b_n$$

➙ $< A, B >= ||A||\ ||B||\ cos\theta \in \mathbb{R}$ désigne un produit vectoriel (*inner product*) qui généralise, dans un espace vectoriel Euclidien, le produit scalaire (*dot product*) noté $A.B$

➙ Les deux produits ont la même interprétation géométrique.

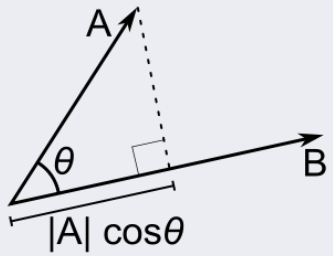
Interprétation Géométrique :

• $A_B = ||A||cos\theta$ est la *projection* scalaire de A sur B

• Sachant $A.B = ||A||\ ||B||\ cos\theta$ alors $A_B = \dfrac{A.B}{||B||}$

• $a.b = ||a||||b||cos\theta$

• $a.a = ||a||^2$ $\theta = 0$



Addendum : Rappels Sur l'espace vectoriel (suite)

• Les propriétés (axiomes) sur le produit vectoriel de l'espace vectoriel PréHilbertien sont :

$< x, y >=< y, x >$ *symétrie conjuguée* = symétrie dans $\mathbb{R}$
$< \alpha x, y >= \alpha < x, y >$ la linéarité ($\alpha \in \mathbb{R}$)
$< x + y, z >=< x, z > + < y, z >$ la linéarité
$< x, x > \geq 0$ (et $< 0, 0 >= 0$)

- IBL : Plus Proches Voisins (NN)
- Un exemple : la cueillette de champignons
- Addendum : Recherche efficace des voisins les plus proches (NN)
- Remarques sur la méthode IBL

11 Remarques générales
- Remarques sur les méthodes

12 Addendum : rappels sur les vecteurs / matrices
- Addendum : rappels sur les vecteurs
- Addendum : compléments sur les matrices
- Addendum : Rappels Sur l'espace vectoriel

Table des matières

Table des matières

Table des matières (suite)

Table des matières (suite)