

Chapitre 5: Méthode non linéaire en apprentissage supervisé: Support Vector Machine (SVM)

Séparateurs à Vaste Marge

Mahdi LOUATI

2 Ingénierie des Donnés et Systèmes Décisionnels
Ecole Nationale d'Electronique et des Télécommunications de Sfax

09 Novembre 2022



Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

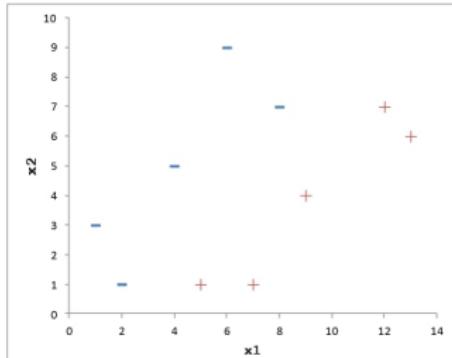
Discrimination binaire

Apprentissage supervisé: $Y = f(x_1, \dots, x_p; \beta)$ dans un cadre binaire i.e.,

$$Y \in \{+, -\} \text{ ou } Y \in \{0, 1\},$$

avec $\beta = (\beta_1, \dots, \beta_p)$ ainsi que β_0 sont les $p + 1$ paramètres à estimer.

x1	x2	y
1	3	-1
2	1	-1
4	5	-1
6	9	-1
8	7	-1
5	1	1
7	1	1
9	4	1
12	7	1
13	6	1



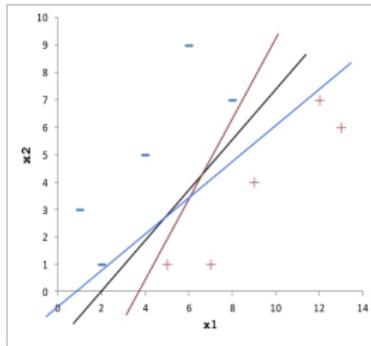
Objectif

Trouver une séparation linéaire permettant de distinguer les "+" des "-". Le classifieur se présente sous forme de combinaison linéaire des variables.

$$f(x) = x^t \beta + \beta_0 = \beta_1 x_1 + \beta_2 x_2 \dots + \beta_p x_p + \beta_0.$$

Recherche de la solution "optimale"

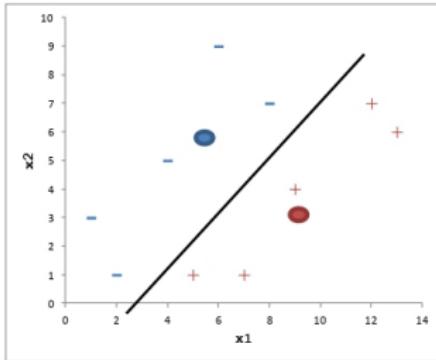
Une fois la "forme" de la fonction de séparation choisie, il faut choisir une solution parmi l'infinité de solutions possibles.



Deux questions clés toujours en "machine learning":

- ① Choisir la forme de la séparation ("representation bias" ou "hypothesis bias") \Rightarrow **choix de famille de fonction**.
- ② Privilégier une solution parmi l'ensemble des solutions possibles ("preference bias") \Rightarrow **définir souvent un critère à optimiser**.

Exemple: Analyse discriminante linéaire



La droite de séparation est à mi-chemin entre les deux barycentres conditionnels au sens de la distance de Mahalanobis.

Distance de Mahalanobis

Mesure de distance mathématique introduite en 1936, basée sur la corrélation entre des variables. Elle est définie comme étant la mesure de dissimilarité entre deux vecteurs aléatoires \vec{x} et \vec{y} de même distribution avec une matrice de covariance Σ

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^t \Sigma^{-1} (\vec{x} - \vec{y})}.$$

Distance de Mahalanobis

C'est une mesure de distance mathématique introduite par Prasanta Chandra Mahalanobis en 1936, basée sur la corrélation entre des variables. En pratique, la distance de Mahalanobis est définie comme étant la mesure de dissimilarité entre deux vecteurs aléatoires \vec{x} et \vec{y} de même distribution avec une matrice de covariance Σ

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^t \Sigma^{-1} (\vec{x} - \vec{y})}.$$

Remarques

- Si $\Sigma = I_r \implies$ distance Euclidienne.
- Si Σ est diagonale, \implies distance Euclidienne normalisée:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^p \frac{(x_i - y_i)^2}{\sigma_i^2}}.$$

où σ_i est l'écart type de x_i sur la série de données.

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Proposition

Soit $x_0 \in \mathbb{R}^d$ et H_f un hyperplan

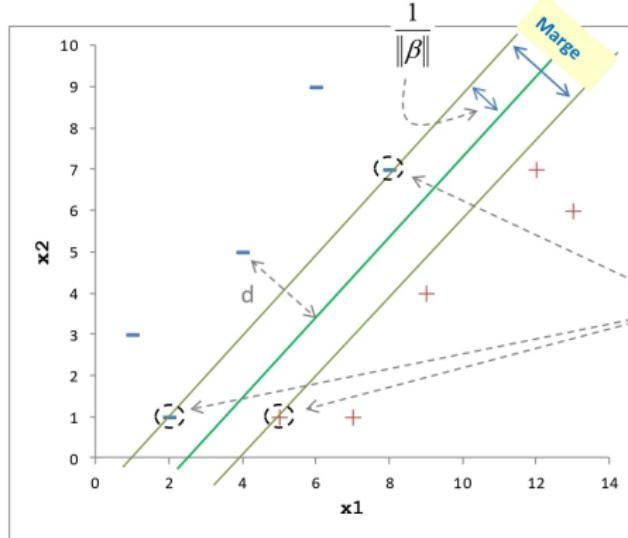
$$H_f = \{x \in \mathbb{R}^d; f(x) = x^t w + b = 0\}.$$

La distance de x_0 à H_f définie par $\inf_{y \in H_f} \|x_0 - y\|$ est donnée par

$$d(x_0, H_f) = \frac{|f(x_0)|}{\|w\|} = \frac{|x_0^t w + b|}{\|w\|}.$$

Principe de la maximisation de la marge

L'hyperplan optimal doit maximiser la distance entre la frontière de séparation et les points de chaque classe qui lui sont les plus proches.



- Distance d'un point quelconque

x avec la frontière (cf. [projection orthogonale](#))

$$d = \frac{|x^T \beta + \beta_0|}{\|\beta\|}$$

- La marge maximale est égale à

$$\delta = \frac{2}{\|\beta\|}$$

- Les points où « s'appuient » les droites « marges » sont les « vecteurs de support ». Si on les retire de l'échantillon, la solution est modifiée.

- Plusieurs zones sont définies dans l'espace de représentation

$f(x) = 0$, on est sur la **frontière**

$f(x) > 0$, on classe « + »

$f(x) < 0$, on classe « - »

$f(x) = +1$ ou -1 , on est sur les droites délimitant des vecteurs de support

Maximisation de la marge (Formulation mathématique)

Maximiser la marge revient à minimiser la norme du vecteur β .

$$\max \frac{2}{\|\beta\|} \iff \min \|\beta\|.$$

$$\min_{\beta, \beta_0} \|\beta\|$$

Sous contrainte :

$$y_i \times f(x_i) \geq 1, i = 1, \dots, n$$

- Norme :

$$\|\beta\| = \sqrt{\beta_1^2 + \dots + \beta_p^2}$$

- Les contraintes indiquent que tous les points sont du bon côté, au pire ils sont sur la droite définissant les vecteurs de support.

Remarque : on retrouve souvent aussi l'écriture

$$\min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2$$

- On a un problème d'optimisation convexe (fonction objectif quadratique, contraintes linéaires). Un optimum global existe.

- Mais il n'y a pas de solution littérale. Il faut passer par des programmes d'optimisation numérique.

Maximisation de la marge (exemple sous EXCEL)

On utilise le **SOLVEUR** pour résoudre le problème d'optimisation.

	beta.1 0.667	beta.2 -0.667	beta.0 -1.667	
n°	x1	x2	y	f(x)
1	1	3	-1	-3
2	2	1	-1	-1
3	4	5	-1	-2.333333333
4	6	9	-1	-3.666666667
5	8	7	-1	-1
6	5	1	1	1
7	7	1	1	2.333333333
8	9	4	1	1.666666667
9	12	7	1	1.666666667
10	13	6	1	3

(p + 1) cellules variables

Contraintes saturées : 3 points supports à l'issue de l'optimisation (n°2, 5 et 6)

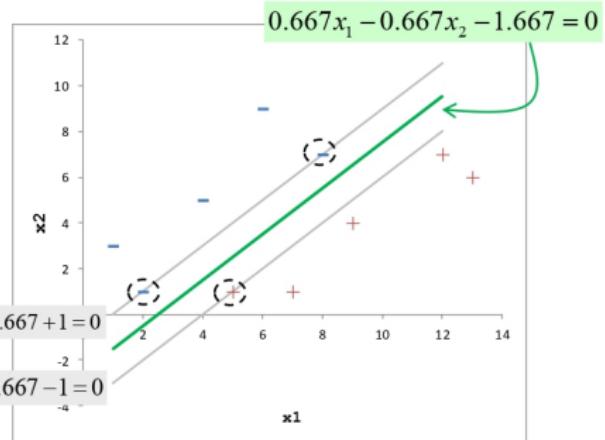
Norme.Beta **0.943**

n = 10 contraintes

Cellule cible à définir $\|\beta\|$

$$x^T \beta + \beta_0 \pm 1 = 0$$

$$\left. \begin{array}{l} 0.667x_1 - 0.667x_2 - 1.667 + 1 = 0 \\ 0.667x_1 - 0.667x_2 - 1.667 - 1 = 0 \end{array} \right\}$$



Première formulation

Règle de classement pour un individu i^* basé sur les coefficients estimés $\hat{\beta}_j$.

$$f(x_{i^*}) \begin{cases} \geq 0 \Rightarrow \hat{y}_{i^*} = 1 \\ < 0 \Rightarrow \hat{y}_{i^*} = -1 \end{cases}$$

beta.1	beta.2	beta.0			
0.667	-0.667	-1.667			
x1	x2	y	f(x)	prediction	
1	3	-1	-3	-1	
2	1	-1	-1	-1	
4	5	-1	-2.3333	-1	
6	9	-1	-3.6667	-1	
8	7	-1	-1	-1	
5	1	1	1	1	
7	1	1	2.33333	1	
9	4	1	1.66667	1	
12	7	1	1.66667	1	
13	6	1	3	1	

Inconvénients de cette première écriture

- ① Les algorithmes d'optimisation numérique (prog. quadratique) ne sont pas opérationnels dès que le nombre de prédicteurs p est grand ($>$ quelques centaines). Ce qui arrive souvent dans le traitement des données complexes (text mining, image,...) "peu d'exemples, beaucoup de descripteurs".
- ② Cette écriture ne met pas en évidence la possibilité d'utiliser des fonctions "noyau" qui permettent d'aller au-delà du cadre des classificateurs linéaires.

Problème primal et problème dual

Les problèmes d'optimisation linéaire sont des problèmes d'optimisation dans lesquels la fonction objectif et les contraintes sont toutes linéaires.

Dans le problème primal, la fonction objectif est une combinaison linéaire de n variables. Il y a m contraintes, qui chacune place une majoration sur une combinaison linéaire des n variables. Le but est de maximiser la valeur de la fonction objectif soumise aux contraintes. Une solution sera alors un vecteur de n valeurs qui atteint le maximum possible pour la fonction objectif.

Dans le problème dual, la fonction objectif est une combinaison linéaire des m valeurs qui sont limites sur les m contraintes pour le problème primal. Il y a donc n contraintes duales, chacune plaçant une minoration sur une combinaison linéaire des m variables duales.

Principe de Lagrange: Dualité de Lagrange

Soit un problème d'optimisation non linéaire dans sa forme standard

minimiser $f(x)$

avec $g_i(x) \leq 0, i \in \{1, \dots, m\}$

$h_j(x) = 0, j \in \{1, \dots, p\}$

dans le domaine $\mathcal{D} \subset \mathbb{R}^n$ non vide, le Lagrangien $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$

$$\mathcal{L}(x, \lambda, \nu) = f(x) + \sum_{i=1}^m \lambda_i g_i(x) + \sum_{j=1}^p \nu_j h_j(x).$$

Les vecteurs $\lambda \in \mathbb{R}^m$ et $\nu \in \mathbb{R}^p$ sont appelés variables duales ou vecteurs multiplicateurs de Lagrange associés au problème.

La fonction duale de Lagrange $g : \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$ est définie par

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} \mathcal{L}(x, \lambda, \nu) = \inf_{x \in \mathcal{D}} \left(f(x) + \sum_{i=1}^m \lambda_i g_i(x) + \sum_{j=1}^p \nu_j h_j(x) \right).$$

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)**
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Expression duale (Passage au Lagrangien)

Un problème d'optimisation possède une forme duale si on évolue dans un espace convexe. Ce qui est le cas. On passe alors par le Lagrangien.

$$\min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2$$

Le problème initial...

$$s.c. \quad y_i \times (x_i^T \beta + \beta_0) \geq 1, \forall i = 1, \dots, n$$

...devient sous sa forme
duale

$$L_P(\beta, \beta_0, \alpha) = \frac{1}{2} \|\beta\|^2 - \sum_{i=1}^n \alpha_i [y_i \times (x_i^T \beta + \beta_0) - 1]$$

Où α_i sont les [multiplicateurs de Lagrange](#)

En annulant des dérivées partielles premières, nous avons les relations

$$\left[\begin{array}{l} \frac{\partial L}{\partial \beta} = \beta - \sum_{i=1}^n \alpha_i y_i x_i = 0 \\ \frac{\partial L}{\partial \beta_0} = \sum_{i=1}^n \alpha_i y_i = 0 \\ \frac{\partial L}{\partial \alpha_i} = -y_i \times (x_i^T \beta + \beta_0) + 1 \leq 0, \forall i \end{array} \right.$$

Il est possible d'obtenir les coefficients de l'hyperplan à partir des multiplicateurs de Lagrange

La solution doit satisfaire aux [conditions \(complémentaires\) de Karush-Kuhn-Tucker \(KKT\)](#) $\alpha_i [y_i \times (x_i^T \beta + \beta_0) - 1] = 0, \forall i$

Expression duale (Optimisation)

En introduisant les informations issues de l'annulation des dérivées partielles du Lagrangien, on obtient une optimisation ne dépendant que des multiplicateurs.

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{i'=1}^n \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle$$

Sous
contrainte :

$$\alpha_i \geq 0, \forall i$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

- $\langle x_i, x_{i'} \rangle$ est le produit scalaire entre les observations i et i'

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^p x_{ij} \times x_{i'j}$$

- $\alpha_i > 0$ vont définir les points importants c.à-d. les points supports

- Forcément, il y aura des points supports d'étiquettes différentes sinon cette condition ne peut pas être respectée.

Exemple numérique (Traitements sous Excel)

Toujours avec **SOLVEUR**, essayons de résoudre le problème d'optimisation.

Cellules variables α_i

Seuls les points supports présentent une « pondération » α_i non-nulle (n°2, 5 et 6)

La matrice $\langle x_i, x_i \rangle$ est appelée **matrice de Gram**

$$\alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle$$

n°	x1	x2	y	alpha	y*alpha
1	1	3	-1	0	0
2	2	1	-1	0.333333	-0.333333
3	4	5	-1	0	0
4	6	9	-1	0	0
5	8	7	-1	0.111111	-0.111111
6	5	1	1	0.444444	0.444444
7	7	1	1	0	0
8	9	4	1	0	0
9	12	7	1	0	0
10	13	6	1	0	0

n°	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0.6	0	0	0.9	-1.6	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0.9	0	0	1.4	-2.3	0	0	0	0
6	0	-1.6	0	0	-2.3	5.1	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

$$\sum_{i=1}^n \alpha_i \quad \text{LD} \quad 0.444444$$

Fonction objectif $L_D(\alpha)$

$$\begin{aligned} \text{Somme} & 0.889 \\ \text{Racine} & 0.943 \end{aligned}$$

$$\sum_{i=1}^n \sum_{i'=1}^n \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle = 0.889$$

$$\|\beta\| = \sqrt{\sum_{i=1}^n \sum_{i'=1}^n \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle} = 0.943$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

N'oublions pas que la marge est égale à $\delta = \frac{2}{\|\beta\|}$

Pts supports aux coef. de l'hyperplan (Obtention des β à partir des α).

Puisque les expressions primales et duales sont deux facettes du même problème, on doit pouvoir passer de l'un à l'autre.

A partir de la dérivée partielle du Lagrangien par rapport à β

$$\frac{\partial L}{\partial \beta} = \beta - \sum_{i=1}^n \alpha_i y_i x_i = 0 \Rightarrow \beta = \sum_{i=1}^n \alpha_i y_i x_i$$

Seuls les points supports participent au calcul des coefficients, puisque ce sont les seuls pour lesquels ($\alpha_i > 0$)

A partir des conditions de KKT, calculées sur n'importe lequel des points supports (il faut que α_i soit non-nul), on peut obtenir β_0

$$\alpha_i [y_i \times (x_i^T \beta + \beta_0) - 1] = 0$$

$$\Rightarrow \beta_0 = \frac{1 - y_i x_i^T \beta}{y_i}$$

Remarque

Comme $y_i \in \{-1, 1\}$, alors on peut aussi écrire

$$\beta_0 = y_i - x_i^T \beta.$$

Des points supports aux coefficients de l'hyperplan (Exemple numérique)

Pour β_1 , seule la variable X_1 prend part aux calculs

$$\begin{aligned}\beta_1 &= 0.333 \times (-1) \times 2 \\ &\quad + 0.111 \times (-1) \times 8 \\ &\quad + 0.444 \times (1) \times 5 \\ &= 0.6667\end{aligned}$$

Vecteurs de support

n°	x1	x2	y	alpha
2	2	1	-1	0.333
5	8	7	-1	0.111
6	5	1	1	0.444

beta.1	0.6667
beta.2	-0.6667

beta.0	-1.6667
	-1.6667
	-1.6667

$$\begin{aligned}\beta_0 &= \frac{1 - y_i x_i^T \beta}{y_i} \\ &= \frac{1 - (-1) \times [0.667 \times 2 + (-0.667) \times 1]}{(-1)} \\ &= -1.6667\end{aligned}$$

Le résultat est le même quel que soit le point support utilisé.

Si on choisit le point support n°2

Classement d'un individu supplémentaire (Utilisation des points supports)

Utilisation des points supports pour le classement des individus. Cette formulation sera fondamentale pour l'utilisation des fonctions "noyau".

$$f(x) = x^T \beta + \beta_0$$

La fonction de classement peut s'écrire en fonction des coefficients β ou des multiplicateurs α

$$= \sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + \beta_0$$

S est l'ensemble des points supports. Ce sont les seuls à avoir un poids α_i non nul.

Seuls les points supports participent au classement !

Remarques

- On a une sorte d'algorithme des plus proches voisins où seuls les points supports participent au classement (lesquels étant pondérés α_i)
- La constante β_0 peut être obtenue à partir des conditions de KKT appliquées sur un des points supports

$$\beta_0 = \frac{1 - y_i x_i^T \beta}{y_i}$$

Classement d'un individu supplémentaire (Exemple numérique)

Pour le classement de l'individu n°1

$$\begin{aligned}
 f(x) &= \sum_{r \in S} \alpha_r y_r \langle x_r, x \rangle + \beta_0 \\
 &= 0.333 \times (-1) \times (2 \times 1 + 1 \times 3) + 0.111 \times (-1) \times (8 \times 1 + 7 \times 3) + 0.444 \times (1) \times (5 \times 1 + 1 \times 3) + (-1.667) \\
 &= -3.0
 \end{aligned}$$

n°	x1	x2	y	f(x)	prediction
1	1	3	-1	-3.000	-1
2	2	1	-1	-1.000	-1
3	4	5	-1	-2.333	-1
4	6	9	-1	-3.667	-1
5	8	7	-1	-1.000	-1
6	5	1	1	1.000	1
7	7	1	1	2.333	1
8	9	4	1	1.667	1
9	12	7	1	1.667	1
10	13	6	1	3.000	1

Vecteurs de support

Utilisation des 3 points supports.

n°	x1	x2	y	alpha
2	2	1	-1	0.333
5	8	7	-1	0.111
6	5	1	1	0.444

Beta.0	-1.667
--------	--------

Remarques

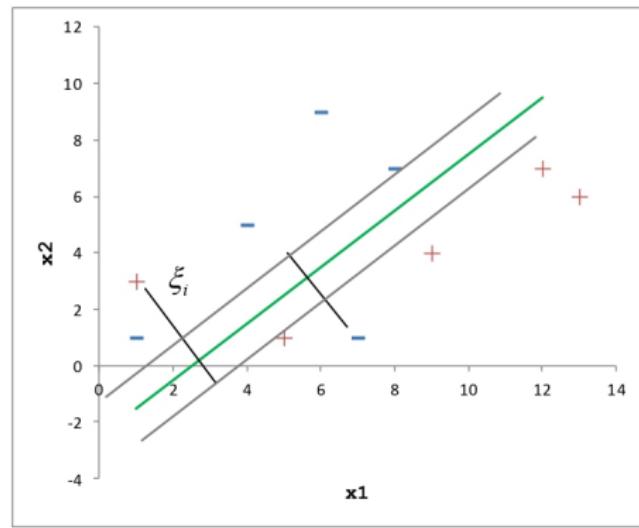
- Cette écriture est complètement cohérente avec la formulation primale.
- Elle met mieux en lumière le rôle des points supports avec les pondérations α_i .
- Elle met en lumière également le rôle fondamental du produit scalaire $\langle x_i, x_j \rangle$ dans les calculs, ce sera très important par la suite (cf. fonctions "noyau").
- Dans les très grandes dimensionnalités (p très élevé, n relativement faible traitement d'images, text mining), cette écriture rend les calculs plus simples pour les techniques d'optimisation.

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 **Soft margin**
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Variables d'écart - Slack variables (Utilisation des variables de relaxation ξ_i)

La séparation parfaite est une vue de l'esprit. En pratique, il arrive que des individus soient du mauvais côté de la frontière.



- ξ est un vecteur de taille n
- $\xi_i \geq 0$ matérialise l'erreur de classement pour chaque observation
- $\xi_i = 0$, elle est nulle lorsque l'observation est du bon côté de la droite « marge » associée à sa classe
- $\xi_i < 1$, le point est du bon côté de la frontière, mais déborde de la droite « marge » associée à sa classe
- $\xi_i > 1$, l'individu est mal classé

Reformulation de l'optimisation (Paramètre de coût - cost parameter)

Il faut pénaliser les erreurs, plus ou moins fortement selon que l'on veuille plus ou moins "coller" aux données d'apprentissage (régularisation).

Formulation
primale

$$\min_{\beta, \beta_0, \xi_i} \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^n \xi_i$$

s.c.

$$y_i \times (x_i^T \beta + \beta_0) \geq 1 - \xi_i, \forall i = 1, \dots, n$$

$$\xi_i \geq 0, \forall i$$

La tolérance aux erreurs est plus ou moins accentuée avec le paramètre **C** ("cost" parameter)

→ **C** trop élevé, danger de sur-apprentissage

→ **C** trop faible, sous-apprentissage

Formulation
duale

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{i'=1}^n \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle$$

s.c.

$$\sum_{i=1}^n \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \forall i$$

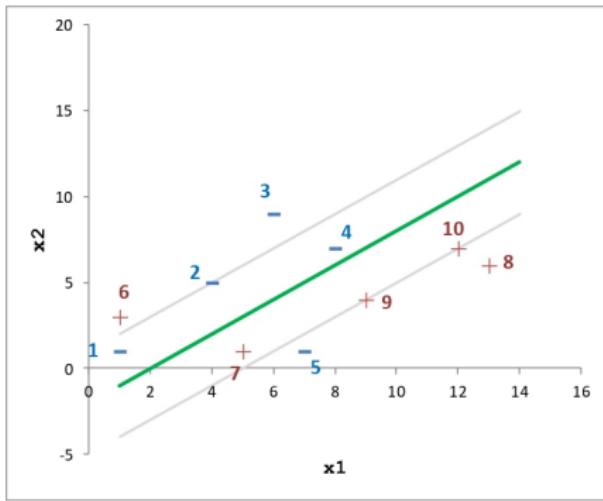
Soft-margin (Exemple)

Formulation primaire

- Minimisation de la fonction objectif en fonction des β
- et ξ
- C est un paramètre que l'on a fixé à $C = 5$



Le choix de **C** constituera un enjeu important en pratique



n°	β		y	ξ_i	1- ξ_i	$y^*\beta(x)$
	beta.1 0.333	beta.2 -0.333				
1	1	1	-1	0.333	0.667	0.667
2	4	5	-1	0	1	1
3	6	9	-1	0	1	1.667
4	8	7	-1	0.667	0.333	0.333
5	7	1	-1	2.333	-1.333	-1.333
6	1	3	1	2.333	-1.333	-1.333
7	5	1	1	0.333	0.667	0.667
8	13	6	1	0	1	1.667
9	9	4	1	0	1	1
10	12	7	1	0	1	1

C **5**

Fonc.Obj **30.1111**

- $y_i(x_i^T\beta + \beta_0) = 1 - \xi_i$: contrainte saturée → point support (fond jaune dans le tableau) c.-à-d. on retire le point, la solution serait différente (8 points supports ici)
- $\xi_i = 0$: Le point est du bon côté de sa droite marge
- $\xi_i \geq 1$: Le point est du mauvais côté de la frontière (on observe 2 individus mal classés)
- $0 < \xi_i < 1$: le point est du bon côté de la frontière, mais déborde de la droite « marge » associée à sa classe

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Changement de représentation (Transformation de variables)

En réalisant les transformations de variables adéquates, on peut rendre linéairement séparable un problème qui ne l'est pas dans l'espace initial.

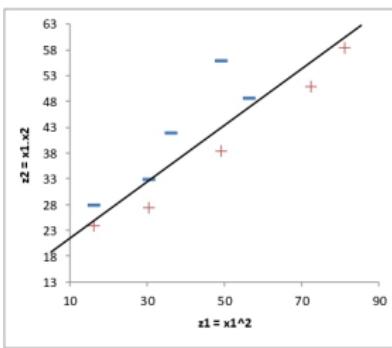
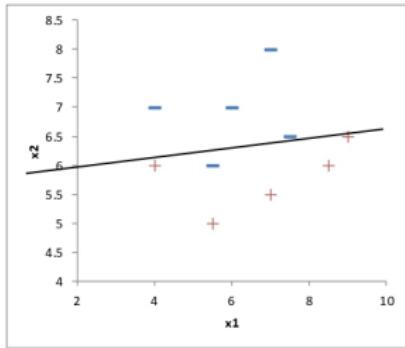
n°	x1	x2	y
1	4	7	-1
2	7	8	-1
3	5.5	6	-1
4	6	7	-1
5	7.5	6.5	-1
6	5.5	5	1
7	4	6	1
8	7	5.5	1
9	8.5	6	1
10	9	6.5	1



n°	z1	z2	y
1	16	28	-1
2	49	56	-1
3	30.25	33	-1
4	36	42	-1
5	56.25	48.75	-1
6	30.25	27.5	1
7	16	24	1
8	49	38.5	1
9	72.25	51	1
10	81	58.5	1

$$z_1 = x_1^2$$

$$z_2 = x_1 x_2$$



Mais démultiplier
« physiquement » les
variables intermédiaires
dans la base est coûteux,
sans être sûr d'aboutir à la
bonne transformation.

Les fonctions "noyau" (Appliquées aux produits scalaires)

Le produit scalaire entre les vecteurs individus tient une place importante dans les calculs (formulation duale). Or elles peuvent tirer profit des fonctions "noyau".

Soit une fonction de transformation $\phi(x)$ des variables initiales

$$\text{Ex. } x = (x_1, x_2) \rightarrow \phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



Avec la formulation duale, pour optimiser le Lagrangien, nous devons calculer la matrice des produits scalaire $\langle \phi(x_i), \phi(x_{i'}) \rangle$ pour chaque couple d'individus (i, i')

On doit manipuler 3 variables au lieu de 2, les calculs sont plus coûteux, sans compter le stockage des variables supplémentaires.

On peut trouver une fonction $K(\cdot)$, dite fonction noyau, tel que



La principale conséquence est que l'on calcule simplement le produit scalaire $\langle x_i, x_{i'} \rangle$, et on ne transforme que ce résultat avec la fonction noyau.

$$K(x_i, x_{i'}) = \langle \phi(x_i), \phi(x_{i'}) \rangle$$

On ne manipule que les 2 variables initiales pour les calculs. Mais on se projette bien dans un espace à 3 dimensions !

Noyau polynomial (Exemple)

Produit scalaire entre deux individus (vecteurs) u et v dont voici les valeurs

$$\begin{aligned} u &= (4,7) \\ v &= (2,5) \end{aligned}$$

$$\langle u, v \rangle = 4 \times 2 + 7 \times 5 = 43$$

Transformation (1)

$$\phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\left. \begin{aligned} \phi(u) &= (16, 39.6, 49) \\ \phi(v) &= (4, 14.1, 25) \end{aligned} \right\} \Rightarrow \langle \phi(u), \phi(v) \rangle = 1849$$

Fonction noyau correspondante (1)

$$K_1(u, v) = (\langle u, v \rangle)^2 = 43^2 = 1849$$

Les résultats sont équivalents. Avec $K(\cdot)$, on se projette dans un espace plus grand sans avoir à former explicitement les variables.

Transformation (2)

$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\left. \begin{aligned} \phi(u) &= (1, 5.7, 9.9, 16, 49, 39.6) \\ \phi(v) &= (1, 2.8, 7.1, 4, 25, 14.1) \end{aligned} \right\} \Rightarrow \langle \phi(u), \phi(v) \rangle = 1936$$

Fonction noyau correspondante (2)

$$K_2(u, v) = (1 + \langle u, v \rangle)^2 = (1 + 43)^2 = 1936$$

On se positionne dans un espace à 5 dimensions dans cette configuration.

Formulation duale (Utilisation des fonctions "noyau")

Formulation duale
avec « soft margin »

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{i'=1}^n \alpha_i \alpha_{i'} y_i y_{i'} K(x_i, x_{i'})$$

s.c.

$$\sum_{i=1}^n \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \forall i$$



Il n'est plus possible d'obtenir une fonction de classement explicite, il faut absolument passer par les points supports pour classer les individus c.-à-d. il faut les stocker (ainsi que leurs poids) pour le déploiement

$$f(x) = \sum_{i' \in S} \alpha_{i'} y_{i'} K(x_{i'}, x) + \beta_0$$



β_0 peut être obtenu via la réalisation de la condition de Karush-Kuhn-Tucker (KKT), mais en utilisant la fonction noyau toujours

Quelques fonctions noyau- Les plus utilisées dans les logiciels (ex. Package Scikit-learn pour Python - SVC)

Le paramétrage est le principal enjeu lorsqu'on souhaite les mettre en oeuvre sur nos données. Sans oublier le "cost parameter" **C**.

Noyau
polynomial

$$K(u, v) = (\text{coef0} + \langle u, v \rangle)^{\text{degree}}$$

`coef0 = 0` et `degree = 1`, nous avons le « noyau linéaire »

Noyau RBF (radial
basis function)

$$K(u, v) = \exp(-\gamma \times \|u - v\|^2)$$

Si non spécifié, les outils
choisissent par défaut (`p` :
nombre de variables)

$$\gamma = \frac{1}{p}$$

Noyau sigmoid

$$K(u, v) = \tanh(\gamma \times \langle u, v \rangle + \text{coef0})$$

Plan

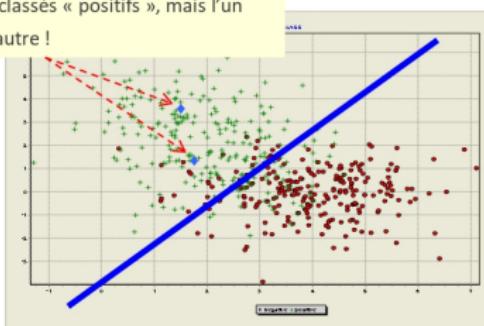
- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Probabilités d'appartenance aux classes

L'output de la fonction $f(x)$ permet de classer les individus

$$f(x) \begin{cases} \geq 0 \Rightarrow \hat{y} = 1 \\ < 0 \Rightarrow \hat{y} = -1 \end{cases}$$

Les deux points sont classés « positifs », mais l'un est plus positif que l'autre !



Mais nous avons besoin d'une indication sur le degré de crédibilité de la réponse

$|f(x)|$ est déjà une indication. Elle permet de classer les individus selon leur degré de positivité (ex. scoring)

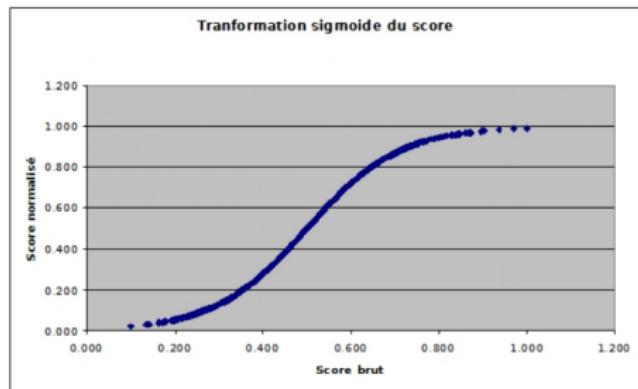
Mais...

Mais dans de nombreux domaines, on a besoin d'une probabilité d'appartenance (ex. interprétation, utilisation d'une matrice de coûts, comparaison d'outputs avec d'autres méthodes, etc.)

Méthode de Platt (Estimation par le maximum de vraisemblance)

On utilise une fonction sigmoïde simple permet de « mapper » $f(x)$ dans l'intervalle $[0, 1]$

$$P(Y = 1/x) = \frac{1}{1 + \exp[-f(x)]}$$



On peut aller plus loin en utilisant une expression paramétrée et estimer les coefficients par maximum de vraisemblance

$$P(Y = 1/x) = \frac{1}{1 + \exp[-(a \times f(x) + b)]}$$



Un programme de régression logistique saura très bien estimer les valeurs de "a" et "b"

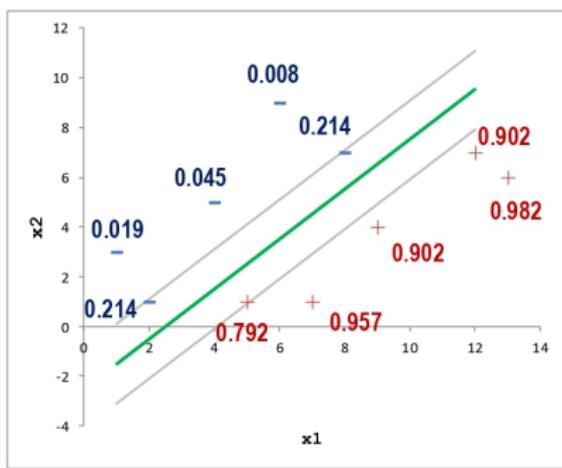
Méthode de Platt (Exemple)

$$P(Y=1/x) = \frac{1}{1 + \exp[-(1.32 \times f(x) + 0.02)]}$$

	beta.1	beta.2	beta.0		
	0.667	-0.667	-1.667	'	
n°	x1	x2	y	f(x)	P(y=1/x)
1	1	3	-1	-3.000	0.019
2	2	1	-1	-1.000	0.214
3	4	5	-1	-2.333	0.045
4	6	9	-1	-3.667	0.008
5	8	7	-1	-1.000	0.214
6	5	1	1	1.000	0.792
7	7	1	1	2.333	0.957
8	9	4	1	1.667	0.902
9	12	7	1	1.667	0.902
10	13	6	1	3.000	0.982

a	1.32
b	0.02

Les probabilités sont cohérentes avec la position du point et son degré d'éloignement par rapport à la frontière.



Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Méthodes externes (Filtrage et wrapper)

Techniques de sélection qui ne sont pas en relation directe avec la méthode d'apprentissage.

Méthode de filtrage

La sélection se fait en amont, avant et indépendamment de la méthode d'apprentissage utilisée par la suite. Souvent basée sur la notion de « corrélation » au sens large.

Avantages : Rapidité, généricité.

Inconvénients : Non relié aux caractéristiques de la méthode, rien de dit que les variables ainsi sélectionnées seront les meilleures.

Méthode « wrapper »

Utilise le modèle comme une boîte noire. Recherche (ex. forward, backward) du meilleur sous-ensemble de variables optimisant un critère de performances (ex. taux d'erreur en validation croisée).

Avantage : relié à un critère de performance.

Inconvénients : Lourdeur des calculs ; danger de sur-apprentissage ; non connecté avec les caractéristiques intrinsèques de la méthode (ex. max de la marge pour les SVM).

Méthode interne (intégrée, embedded) Critère de maximisation de la marge

Deux axes clés

- Mesurer la contribution d'une variable.
- Monter un algorithme autour de ce critère.

Mesurer la contribution d'une

variable " x_j " dans le modèle,
sans avoir à relancer

explicitement l'apprentissage
sans " x_j "

La variable x_j est désactivée en mettant à
zéro sa valeur

$$\Delta^{(j)} \|\beta\|^2 = \sum_{i,i' \in S} \alpha_i \alpha_{i'} y_i y_{i'} \left(K(x_i, x_{i'}) - K(x_i^{(j)}, x_{i'}^{(j)}) \right)$$

→ Pour un noyau linéaire, cela équivaut à tester la nullité du coefficient β_j

Stratégie de recherche
backward du « meilleur »
sous-ensemble de variables

1. Calculer δ_0 , la marge initiale avec l'ensemble des variables
2. Trouver j^* tel que $\Delta^{(j^*)} \|\beta\|^2$ est minimum, la mettre de côté
3. Relancer l'apprentissage sans x_{j^*} , calculer la nouvelle marge δ
4. Si $\frac{\delta_0 - \delta}{\delta} < \varepsilon$ alors retirer x_{j^*} de la base, faire $\delta_0 = \delta$ et retour en 2. Sinon arrêt de l'algorithme.

2 informations
essentielles :

- Le retrait d'une variable réduit toujours la marge. La question, est-ce que la réduction est significative ? Auquel cas il faut conserver la variable.
- ε est un paramètre de l'algorithme (ε élevé → moins de variables)

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes**
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

SVM Multiclasses (Approche "one-against-rest")

L'approche SVM est formellement définie pour les problèmes à deux classes, **comment l'étendre (simplement) à des problèmes à K classes avec $Y \in \{y_1, y_2, \dots, y_K\}$?**



- Construire tour à tour K modèles discriminants avec la modalité y_k contre les autres ($Y' \in \{y_k=+1, y_{(k)}=-1\}$). Nous obtenons K fonctions de décision $f_k(x)$
- En classement, prédire la classe présentant le score le plus élevé c.-à-d.

$$\hat{y} = \arg \max_k f_k(x)$$

On retrouve le schéma bayésien avec la maximisation de la probabilité a posteriori



- **Caractéristiques** : K apprentissages à effectuer sur les données
- **Avantages** : simplicité
- **Inconvénients** : on peut introduire artificiellement un déséquilibre des classes dans la construction des modèles individuels, si les scores sont mal calibrés, les comparaisons des output des fonctions de décision ne sont pas équitables

SVM Multiclasses (Approche "one vs. one" (pairwise))

- Construire tour à tour $K(K-1)/2$ modèles discriminant chaque paire de classes
- $(Y' \in \{y_k=+1, y_j=-1\})$. Nous obtenons $K(K-1)/2$ fonctions $f_{k,j}(x)$
- En classement, prédire la classe en utilisant un système de vote c.-à-d. présentant le plus grand nombre de victoires

$D_k(x)$ va fournir le "#votes" de la modalité y_k
 Sachant que $f_{j,k}(x) = -f_{k,j}(x)$

La règle d'affectation s'appuie sur le
 $\#vote \max.$

$$D_k(x) = \sum_{j \neq k, j=1}^K \text{sign}[f_{k,j}(x)]$$

$$\hat{y} = \arg \max_k D_k(x)$$

- **Remarque :** en cas d'ex-aequo (2 classes ou plus ont le même nombre de votes), on procède à la somme des scores $f_{k,j}(x)$ et on prend le max
- **Caractéristiques :** $K(K-1)/2$ apprentissages à effectuer, mais avec des ensembles de données de taille réduite
- **Avantages :** moins de problème de déséquilibre des classes, les scores sont mieux calibrés
- **Inconvénients :** temps de calcul quand K augmente

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Python scikit-learn SVC

```
#importation des données
import pandas
dtrain = pandas.read_table("ionosphere-train.txt",sep="\t",header=0,decimal=".")
print(dtrain.shape)
y_app = dtrain.as_matrix()[:,32]
X_app = dtrain.as_matrix()[:,0:32]
#importation de la classe de calcul
from sklearn.svm import SVC
svm = SVC() #instanciation de l'objet
#affichage des paramètres (paramètres par défaut ici c.-à-d. noyau 'rbf')
#pas de standardisation (scale) des données apparemment
print(svm)

#apprentissage - construction du modèle prédictif
svm.fit(X_app,y_app)
#importation des données test
dtest = pandas.read_table("ionosphere-test.txt",sep="\t",header=0,decimal=".")
print(dtest.shape)
y_test = dtest.as_matrix()
X_test = dtest.as_matrix()[:,0:32]
#prédition sur l'échantillon test
y_pred = svm.predict(X_test)
#évaluation : taux d'erreur = 0.07
from sklearn import metrics
err = 1.0 - metrics.accuracy_score(y_test,y_pred)
print(err)
```

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,
decision_function_shape='ovo', random_state=None)
```

[\[source\]](#)

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

Python scikit-learn GridSearchCV

Scikit-learn propose un mécanisme de recherche des paramètres optimaux en validation croisée. L'échantillon test n'est pas mis à contribution.

```
#classe grille de recherche
from sklearn.grid_search import GridSearchCV

#paramètres à tester - jouer sur les noyaux et le 'cost parameter'
parametres = {"kernel":['linear','poly','rbf','sigmoid'],"C":[0.1,0.5,1.0,2.0,10.0]}

#classifieur à utiliser
svmc = SVC()

#instanciation de la recherche
grille = GridSearchCV(estimator=svmc,param_grid=parametres,scoring="accuracy")

#lancer l'exploration
resultats = grille.fit(X_app,y_app)

#meilleur paramétrage : {'kernel' : 'rbf', 'C' : 10.0}
print(resultats.best_params_)

#prédiction avec le ''meilleur'' modèle identifié
ypredc = resultats.predict(X_test)

#performances du ''meilleur'' modèle - taux d'erreur = 0.045 (!)
err_best = 1.0 - metrics.accuracy_score(y_test,ypredc)
```

R e1071 svm () de LIBSVM

```
#importer les données
dtrain <- read.table("ionosphere-train.txt",header=T,sep="\t")
dtest <- read.table("ionosphere-test.txt",header=T,sep="\t")

#package "e1071"
library(e1071)

#apprentissage
#standardisation automatique des données, cf. paramètre scale
m1 <- svm(class ~ ., data = dtrain)

#affichage
print(m1) <--
```

#prediction

```
y1 <- predict(m1,newdata=dtest)

#matrice de confusion - taux d'erreur = 0.04 <-
mc1 <- table(dtest$class,y1)
err1 <- 1 - sum(diag(mc1))/sum(mc1)
print(err1)
```

Description

svm is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

call:
svm(formula = class ~ ., data = dtrain)

Parameters:
SVM-Type: C-classification
SVM-Kernel: radial
cost: 1
gamma: 0.03125

Number of Support Vectors: 77

Par rapport à scikit-learn, la standardisation des données ([conseillée dans la plupart des cas](#)) joue pleinement

R e1071 tune()

« e1071 » propose également (comme scikit-learn) un outil pour la recherche automatique des meilleurs paramètres

```
#grille de recherche des meilleurs paramètres en validation croisée « cross »
set.seed(1000) #pour obtenir à chaque lancement le même résultat
obj <- tune(svm, class ~ ., data = dtrain, ranges =
            list(kernel=c('linear','polynomial','radial', 'sigmoid'), cost =
                 c(0.1,0.5,1.0,2.0,10)), tunecontrol = tune.control(sampling="cross"))
```

```
#affichage
print(obj) <-----
```

#modélisation avec les nouveaux paramètres

```
m2 <- svm(class ~ ., data = dtrain, kernel='radial', cost = 2)
```

```
#affichage
print(m2) <-----
```

#prédiction

```
y2 <- predict(m2,newdata=dtest)
```

```
#matrice de confusion - taux d'erreur = 0.035
mc2 <- table(dtest$class,y2)
err2 <- 1 - sum(diag(mc2))/sum(mc2)
print(err2)
```

Parameter tuning of 'svm':
 - sampling method: 10-fold cross validation
 - best parameters:
 kernel cost
 radial 2
 - best performance: 0.06666667

Parameters:
 SVM-Type: C-classification
 SVM-Kernel: radial
 cost: 2
 gamma: 0.03125
 Number of Support Vectors: 62

Plan

- 1 Discrimination linéaire
- 2 Maximisation de la marge (Première formulation)
- 3 Maximisation de la marge (Expression duale du pb d'optimisation)
- 4 Soft margin
- 5 Discrimination non linéaire (Astuce du noyau - kernel trick)
- 6 Normalisation des scores (Probabilités d'appartenance aux classes)
- 7 Sélection de variables (Recherche des descripteurs "pertinents")
- 8 Problèmes multi-classes
- 9 Pratique des SVM (Paramétrage Python et R)
- 10 SVM avantages et inconvénients

Avantages

- Capacité à traiter de grandes dimensionnalités (#variables élevé).
- Robuste même si rapport "# observations / # variables" est inversé.
- Traitement des problèmes non linéaires avec le choix des noyaux.
- Non paramétrique.
- Robuste par rapport aux pts aberrants (contrôlé avec le paramètre C).
- points supports donne une bonne indication de la complexité du problème traité.
- Souvent performant dans les comparaisons avec les autres approches.
- Paramétrage permet la souplesse (résistance au sur-apprentissage avec C).

Inconvénients

- Difficulté à identifier les bonnes valeurs des paramètres (et sensibilité aux paramètres).
- Difficulté à traiter les grandes bases avec #observations très élevé (capacité mémoire avec matrice de Gram).
- Problème lorsque les classes sont bruitées.
- Pas de modèle explicite pour les noyaux non linéaires (utilisation des points supports).
- Difficulté d'interprétations (ex. pertinence des variables).
- Le traitement des problèmes multi-classes reste une question ouverte.