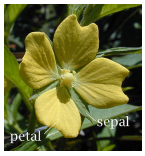
Apprentissage par optimisation numérique

Introduction

Un exemple classique



sépal		pétal		
long.	larg.	long.	larg.	type
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
.	.	.	.	.
7.0	3.2	4.7	1.4	versic.
6.4	3.2	4.5	1.5	versic.
.	.	.	.	.
6.3	3.3	6.0	2.5	virgin.
5.8	2.7	5.1	1.9	virgin.
.	.	.	.	.

$$f(x) = 0.202 \times \texttt{long.sep} + 0.264 \times \texttt{larg.sep} + 1.015 \times \texttt{long.pet} + 1.38 \times \texttt{larg.pet}$$

si $f(x) < 5.1$ alors setosa ;
si $5.1 < f(x) < 9.5$ alors versicolor ;
si $9.5 < f(x)$ alors virginica.

Un autre exemple

Classification d'images, un exemple = N pixels / niveaux de gris :

40 39 39 39 38 37 37 36 36 35 34 34 34 33 33 33
32 31 31 30 30 29 28 28 27 26 27 26 26 25 25 25
24 23 23 22 21 19 18 16 15 17 20 21 22 23 18 14
25 28 25 23 23 22 21 19 18 16 13 13 13 12 17 19
18 19 14 18 21 22 23 24 25 25 25 25 26 26 26 27
28 28 28 28 29 30 30 30 32 32 32 32 32 31 33 33
34 35 36 36 36 37 37 37 38 39 39 39 39 39 41 40
40 41 41 41 40 42 41 41 41 41 42 41 42 42 41 41
40 40 39 38 38 37 37 36 36 35 35 35 33 34 32 32
31 31 30 29 29 29 29 28 28 27 27 27 26 25 25 25
24 24 23 22 22 20 19 17 16 18 22 22 24 24 19 13
25 28 26 24 24 23 22 21 20 18 16 15 14 13 18 19

$\Rightarrow N$ variables numériques
On cherche une fonction $f(x_1, \ldots, x_N) \to \{-1, +1\}$ qui distingue par exemple les images d'un « a » des images d'une autre lettre.

Le cadre d'apprentissage

On a des *exemples* décrits par n attributs **numériques**, et par leur classe, qui peut être -1 ou $+1$:

- l'espace des entrées est $\mathcal{X} = \pm R^n$
- f est la fonction *cible* à apprendre, $f : \mathbf{R}^n \to \{-1, +1\}$
- un exemple a la forme (x, u), où : $\begin{cases} x = (x_1, \ldots, x_n) \in \mathbf{R}^n, \text{ et} \\ u \in \{-1, +1\} : \text{s'il n'y a pas de } \textit{bruit}, u = f(x) \end{cases}$
- $\mathcal{E} \subseteq \mathbf{R}^n \times \{-1, +1\}$

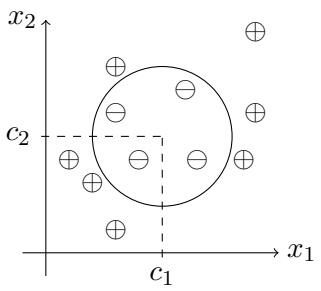
On cherche à classer, donc on cherche $h : \mathbf{R}^n \to \mathbf{R}$ telle que $h(x) \times u > 0$ pour tout $(x, u) \in \mathcal{E}$.

On est dans le cadre de la classification supervisée.

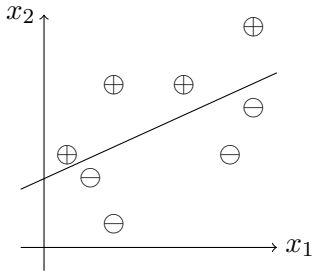
Remarque: on ne cherche pas exactement à avoir $h(x) = \pm 1$, on veut juste que $h(x)$ et y aient le même signe pour les exemples. On pourrait définir $h^s(x) = \texttt{signe}(h(x)) = \pm 1$.

En dimension 2, on peut représenter sur un plan:

- les exemples par un $\oplus$ ou un $\ominus$ suivant le signe de $f(x)$;
- la courbe d'équation $h(x) = 0$.



Séparation non linéaire
$h(x_1, x_2) = (x_1 - c_1)^2 + (x_2 - c_2)^2 - R$



Séparation linéaire
$h(x_1, x_2) = a_1 x_1 + a_2 x_2 + b$
$\mathcal{E}$ est *linéairement séparable*

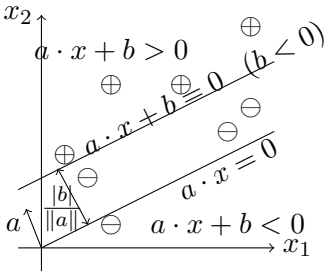
Apprentissage de séparation linéaire

- on cherche un hyperplan séparateur, d'équation

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n + b = a \cdot x + b = 0$$

- on cherche $a = (a_1, \ldots, a_n) \in \mathbf{R}^n$, $b \in \mathbf{R}$ tels que $(a \cdot x + b) \times u > 0$ pour tout $(x, u) \in \mathcal{E}$
- on note $h_{a,b}$ l'hypothèse définie par

$$h_{a,b}(x) = a \cdot x + b = a_1x_1 + a_2x_2 + \ldots + a_nx_n + b.$$



Le perceptron

L'algorithme

1. $a \leftarrow (0, \ldots, 0)$; $b \leftarrow 0$; pas_fini $\leftarrow$ vrai;
2. Tant que pas_fini faire
   (a) pas_fini $\leftarrow$ faux;
   (b) pour chaque $(x, u) \in \mathcal{E}$, si $(a \cdot x + b) \times u \leq 0$ alors
   $a \leftarrow a + \tau \times u \times x$;
   $b \leftarrow b + \tau \times u$;
   pas_fini $\leftarrow$ vrai;
3. Retourner (a, b).

($\tau > 0$ permet de régler la vitesse de convergence)

Propriétés

- algorithme *incrémental* : il suffit de le relancer si de nouveaux exemples sont ajoutés.
- convergence garantie si $\mathcal{E}$ est *linéairement séparable*, c'est-à-dire s'il existe une hypothèse linéaire qui permet de classer $\mathcal{E}$;
- ne converge pas dans le cas contraire.

Remarque : si on part de coefficients nuls, on peut exprimer le résultat de l'algorithme sous la forme

$$a^{(N)} = \sum_{(x,u)\in\mathcal{E}} \alpha_x x$$

où $\alpha_x = \tau N(x) u$

Descente en gradient

Dans la pratique, exemples pas toujours exacts (exemple : reconnaissance de caractères)
$\Rightarrow$ on n'a pas toujours des exemples séparables
$\Rightarrow$ chercher la *meilleure* séparation possible

Erreur empirique pour a, b et $\mathcal{E}$ donnés :

$$E(a, b, \mathcal{E}) = \frac{1}{2} \sum_{(x,u)\in\mathcal{E}} (a \cdot x + b - u)^2$$

On cherche a, b donnant une erreur minimale (dans le cas ou $\mathcal{E}$ n'est pas linéairement séparable, il n'est pas possible d'avoir une erreur nulle).
$\Rightarrow$ on « descend » le long de la surface représentant $E(a, b, \mathcal{E})$ en fonction de a et b : $\frac{\delta E}{\delta a}$ et $\frac{\delta E}{\delta b}$ donnent la direction de la « montée ».

Algorithme :

1. $a \leftarrow (0, \ldots, 0)$; $b \leftarrow 0$;
2. Tant que pas_fini faire
   (a) pour chaque $(x, u) \in \mathcal{E}$,
   $a \leftarrow a + \tau \times (u - (a \cdot x + b)) \times x$;
   $b \leftarrow b + \tau \times (u - (a \cdot x + b))$;
3. Retourner (a, b).

Propriétés

- convergence peut être très lente ;
- convergence même si les exemples ne sont pas linéairement séparables.

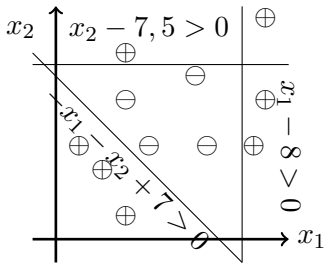
C'est un problème d'optimisation

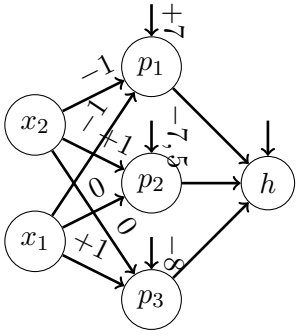
Trouver a, b qui minimisent $\frac{1}{2} \sum_{(x,u)\in\mathcal{E}} (a \cdot x + b - u)^2$:

- on peut utiliser d'autres types d'erreurs : $\log(1 + e^{-u(a\cdot x+b)})$ (perte logistique) mène à d'autres algorithmes
- régularisation = terme dans l'expression à minimiser
  impose des contraintes supplémentaires sur a et b, surtout pour éviter le surapprentissage (overfitting).
  par exemple : minimiser $\frac{1}{2} \sum_{(x,u)\in\mathcal{E}} (a \cdot x + b - u)^2 + C\|a, b\|$
  C = constante à régler
- nombreux algorithmes existent pour résoudre ces pb. d'optimisation spécifiques

Voir par exemple [YHL12].

Réseaux de classifieurs linéaires





Réseau de classifieurs linéaires = graphe acyclique orienté :

- n nœuds d'entrées $X_1, \ldots, X_n$, correspondant aux n variables $x_1, \ldots x_n$
- 1 nœud de sortie H : l'hypothèse à apprendre (on peut aussi avoir plusieurs nœuds de sortie)
- 1 ou plusieurs « couches cachées »

$\Rightarrow$ $\mathcal{N}$ = l'ensemble des nœuds, $\mathcal{N}^*$ = nœuds des couches cachées

- $\mathsf{Pa}(N)$ = ensemble des parents de N, pour $N \in \mathcal{N}^* \cup \{H\}$;
- a_{MN} = poids de l'arc allant de M à N ;
  b_N = poids de N

Calcul de $h(x)$ (propagation) : $N(x) = \mathsf{signe}(\sum_{M \in \mathsf{Pa}(N)} a_{MN} M(x) + b_N) = \mathsf{signe}(a_N \cdot \mathsf{Pa}(N)(x) + b_N)$
où a_N est le vecteur des poids des arcs de $\mathsf{Pa}(N)$ à N

$\Rightarrow$ pour $x = (x_1, \ldots, x_n) \in R^n$ donné, on calcule $h(x)$ en propageant des variables à H

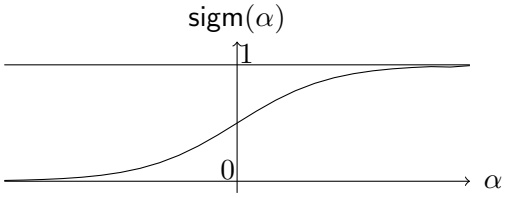
On montre que toute fonction continue de $\mathbf{R}^n$ dans $\mathbf{R}$ peut être approximée d'aussi près qu'on veut par un réseau de neurones à 1 couche cachée et une couche de sortie.

Apprentissage des coefficients

(Attention à l'initialisation : coefs $\neq 0$!)
Tant que « pas fini », faire :

1. Pour chaque $(x, u) \in \mathcal{E}$ faire
   (a) *// propagation*
   pour chaque $N \in \mathcal{N}^*$, en partant des successeurs des entrées et en allant vers la sortie, calculer $N(x)$;
   (b) *// calcul de la correction à apporter en H*
   $\delta H \leftarrow \tau \times (u - H(x))$;
   (c) *// rétropropagation de la correction*
   Pour chaque $N \in \mathcal{N}^*$, en partant des prédécesseurs de H et en allant vers les entrées, faire :
   $\delta N \leftarrow \sum_{N \in \mathsf{Pa}(M)} a_{NM} \delta M$;
   (d) *// mise à jour des coefficients de $\mathcal{N}$*
   Pour chaque $N \in \mathcal{N}^* \cup \{H\}$, faire :
   $a_N \leftarrow a_N + \delta N \times \mathsf{Pa}(N)(x)$; $b_N \leftarrow b_N + \delta N$;

Avec neurones Fonction sigmoïde
$\mathsf{sigm}(\alpha) = \frac{1}{1+e^{-\lambda\alpha}}$ où λ est une constante



Plus λ est grand, plus la courbe se rapproche d'une marche brusque en $\alpha = 0$.
$\Rightarrow N(x) = \mathsf{sigm}(a_N \cdot \mathsf{Pa}(N)(x) + b_N)$.
Remarque : on a maintenant $N(x) \in [0, 1]$.
Modification de l'algorithme d'apprentissage :
Descente en gradient pour minimiser $E(h, \mathcal{E}) = \frac{1}{2} \sum_{(x,u) \in \mathcal{E}} (h(x) - u)^2$.
Avec $\lambda = 1$: $\mathsf{sigm}'(\alpha) = \mathsf{sigm}(\alpha) \times (1 - \mathsf{sigm}(\alpha)) \Rightarrow$

$$\begin{aligned} \delta H &\leftarrow \tau \times H(x) \times (1 - H(x)) \times (u - H(x)) \\ \delta N &\leftarrow N(x) \times (1 - N(x)) \times \sum_{N \in \mathsf{Pa}(M)} a_{NM} \delta M \end{aligned}$$

Minima locaux: appliquer plusieurs fois l'algorithme en prenant des poids initiaux différents ; on garde alors le réseau qui donne la meilleure performance sur un ensemble d'exemples de validation

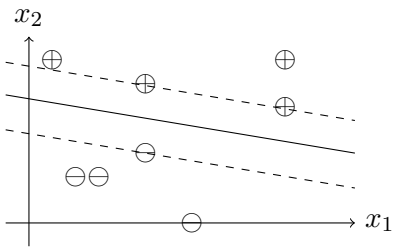
Surapprentissage: pour l'éviter, on peut surveiller la performance du réseau sur un ensemble de validation au cours de l'apprentissage, et arrêter d'apprendre lorsque la performance ne s'améliore pas.

Apprentissage de la structure: dans le cas où on n'a pas d'idée de la structure du réseau à apprendre, il faut apprendre aussi la structure. On peut pour cela partir d'un réseau avec peu de neurones en couche cachée, et ajouter un a un des neurones, en apprenant à chaque fois les poids du réseau: on arrête quand l'ajout de neurone n'améliore pas la performance.

Classes multiples: lorsqu'on doit apprendre un concept ayant un nombre fini de valeurs en sortie mais supérieur à 2, c'est-à-dire que $|\mathcal{Y}| > 2$, on peut apprendre un réseau de neurones ayant plusieurs neurones en sortie : un par classe.

Séparateurs linéaires à vaste marge

Maximisation de la marge



Marge associée à une droite d :
c'est la largeur du « tube » le + large centré sur la droite et qui ne contient aucun exemple.
Plusieurs séparatrices possibles, toutes n'ont pas la même marge.
On va chercher la droite associée à la plus grande marge
$\Rightarrow$ Séparateur à Vaste Marge (SVM).
Remarque: cette droite ne dépend en fait que de quelques exemples de $\mathcal{E}$, on les appelle les *exemples critiques*, ou *vecteurs supports*, *support vector* en Anglais
$\Rightarrow$ Support Vector Machine.

Pourquoi la marge la plus grande possible:

- graphiquement, semble satisfaisant
- semble la + sûre: si on fait une petite erreur sur la localisation de la droite, on obtiendra peu d'erreur.
- marche très bien en pratique

Optimisation quadratique

Distance d'un x qq à l'hyperplan d'équations $= a \cdot x + b$: $\frac{|a \cdot x + b|}{\|a\|}$

Problème initial

- $n+1$ inconnues : $a_1, \ldots, a_n, b$
- Objectif : minimiser $\|a\|^2$
- Contraintes : $u \times (a \cdot x + b) \geq 1$ pour tout $(x, u) \in \mathcal{E}$

Exemple $\mathcal{E} = \{((2,2),+1), ((4,2),-1), ((3,3),+1), ((3,1),-1)\}$

Problème dual

- $|\mathcal{E}|$ inconnues : un α_x pour chaque $(x, u) \in \mathcal{E}$
- Objectif : maximiser $\sum_{(x,u)\in\mathcal{E}} \alpha_x - \frac{1}{2}\| \sum_{(x,u)\in\mathcal{E}} \alpha_x u x\|^2$
- Contraintes : $0 \leq \alpha_x$, $\sum_{(x,u)\in\mathcal{E}} \alpha_x u = 0$

$\Rightarrow$ Solution du pb initial : $h(x) = \sum_{(x',u')\in\mathcal{E}} \alpha_{x'} u' (x' \cdot (x - x^{(0)})) + u^{(0)} = 0$
où $(x^{(0)}, u^{(0)})$ est un exemple critique, caractérisé par $\alpha_{x^{(0)}} \neq 0$.

Dans les deux cas : fonction d'objectif convexe, contraintes linéaires

$\Rightarrow$ une solution unique, qu'on calcule en $O(\texttt{nb\_variables}^3)$

Sous forme matricielle : problème d'*optimisation quadratique* $\Rightarrow$ algorithmes classiques (`Matlab`, `R`, ...)

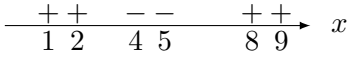
trouver $\alpha = \begin{pmatrix} \alpha_{x_1} \\ . \\ . \\ \alpha_{x_R} \end{pmatrix}$ qui maximise $\alpha^t H \alpha + f^t \alpha$ sous les contraintes: $\begin{cases} -I\alpha \leq 0 \\ C\alpha = 0 \end{cases}$

où $H_{ij} = -u_i u_j x_i x_j$, $f = \begin{pmatrix} 1 \\ . \\ . \\ 1 \end{pmatrix}$, $I = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ . & . & \ldots & . \\ 0 & . & \ldots & 1 \end{pmatrix}$, $C = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 0 & . & \ldots & 0 \\ . & . & \ldots & . \\ 0 & . & \ldots & 0 \end{pmatrix}$

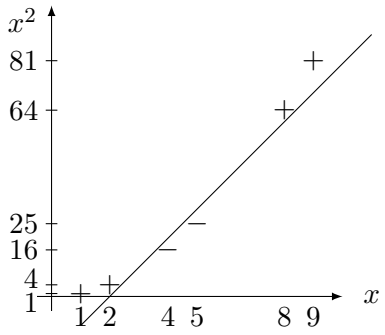
SVMs non linéaires

Un exemple en dimension 1

$\mathcal{E} = \{(1,+1),(2,+1),(4,-1),(5,-1),(8,+1),(9,+1)\}$



on « projette » chaque instance dans $\mathbf{R}^2$: $x \mapsto (x, x^2)$



Optim. dans un espace de dim. $> n$

$\Rightarrow$ Chercher $\Phi : \mathbf{R}^n \to R^m$, $m > n$,
telle qu'il existe $(a,b) \in \mathbf{R}^m \times R$ avec :
$u(a \cdot \Phi(x) + b) \geq 1$ pour tout $(x,u) \in \mathcal{E}$

$\Rightarrow$ Nouveau problème d'optimisation :

$$\begin{cases} \text{Maximiser} \\ \sum_{(x,u)\in\mathcal{E}} \alpha_x - \frac{1}{2} \sum_{(x,u),(x',u')\in\mathcal{E}} \alpha_x \alpha_{x'} u u' (\Phi(x)\cdot\Phi(x')) \\ \text{avec } 0 \leq \alpha_x \text{ pour tout } (x,u) \in \mathcal{E} \\ \text{et } \sum_{x\in\mathcal{E}} \alpha_x u = 0 \end{cases}$$

La fonction séparatrice a alors pour équation :

$$\sum_{x'\in\mathcal{E}} \alpha_{x'} u' (\Phi(x') \cdot (\Phi(x) - \Phi(x^{(0)}))) + u^{(0)} = 0$$

ou encore $a \cdot \Phi(x) + b = 0$
avec $a = \sum_{x'\in\mathcal{E}} \alpha_{x'} u' \Phi(x')$ et $b = u^{(0)} - \Phi(x^{(0)}) \cdot a$.

Problème : on arrive dans des espaces de très grande dimension $\Rightarrow$ le « kernel trick » !

Fonctions à noyau

$k : \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$ telle qu'il existe m et Φ :

- $\Phi : \mathbf{R}^n \to R^m$
- $k(x,x') = \Phi(x) \cdot \Phi(x')$

Problème d'optimisation associé :

- Trouver $(\alpha_x)_{(x,u)\in\mathcal{E}}$
- qui maximise $\sum_{x\in\mathcal{E}} \alpha_x - \frac{1}{2} \sum_{(x,u),(x',u')\in\mathcal{E}} \alpha_x \alpha_{x'} u u' k(x,x')$
- sous les contraintes: $\begin{cases} 0 \leq \alpha_x \text{ pour tout } x \in \mathcal{E} \\ \sum_{(x,u)\in\mathcal{E}} \alpha_x u = 0 \end{cases}$
- Solution :
  $h(x) = \text{signe}(\sum_{(x',u')\in\mathcal{E}} \alpha_{x'} y' k(x', x - x^{(0)}) + u^{(0)})$

Noyaux couramment utilisés

- Gaussien : $k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
- $k(x,y) = \tanh(\kappa x \cdot y - \delta)$ (cf réseaux de neurones)
- Polynômes contenant tous les termes $x_{i_1} \dots x_{i_k}$ jusqu'à un certain degré K.
- ...

Combinaisons de noyaux

si k et k' sont deux fonctions à noyaux, alors les fonctions suivantes sont à noyau (entre autres) :

- $ck(x,x')$
- $f(x)k(x,x')f(x')$
- $q(k(x,x'))$
- $\exp(k(x,x'))$
- $k + k'$
- $k \times k'$

...
où f est une fonction réelle quelconque, q est un polynome à coefficients ≥ 0.

Noyaux sur des espaces non numériques

Le noyau k doit représenter une notion de similarité entre exemples. Rien n'empêche de définir de telles fonctions sur des espaces autres que $\mathbf{R}^n$.
Une contrainte : la fonction k doit être

- symétrique :
- semi-définie positive : la matrice K des $k(x_i,x_j)$ doit vérifier

$$v^t \times K \times v \geq 0 \text{ pour tout } v \in \mathbf{R}^{|\mathcal{E}|}$$

Remarque : K est appelée la matrice de Gram

Par exemple, noyaux sur des ensembles :

- un ensemble de référence fini $\mathcal{A}$
  (Par exemple, pour la détection de spam, $\mathcal{A}$ est l'ensemble des mots du dictionnaire)
- un exemple est défini par une partie $A \subseteq \mathcal{A}$
- $k(A,A') = 2^{|A\cap A'|}$
- ou plus généralement $k(A,A') = \mu(A \cap A')$
  où μ est une probabilité sur $\mathcal{A}$.

On peut définir des noyaux sur des chaînes de caractères, des graphes, des arbres, ...

Bruit

En pratique : il y a des $(x, u) \in \mathcal{E}$ tels que $u \neq f(x)$ (où f est la fonction qu'on veut apprendre)

$\Rightarrow$ on risque de passer dans un espace de très grande dimension pour apprendre une fonction de risque empirique nul, et qui aura un risque réel élevé.

Il vaut mieux apprendre une fonction de risque réel non nul dans un espace de dimension plus faible.

$\Rightarrow$ On va essayer de minimiser $a \cdot a + C \times$ (distance entre mal classés et leur zone)
C est un paramètre à régler : plus il est grand, plus on pénalise les exemples mal classés.

$$\begin{cases} \text{Minimiser } \frac{1}{2}\|a\|^2 + C \sum\limits_{(x,u)\in\mathcal{E}} \epsilon_x \\ \text{avec } u(a \cdot x + b) \geq 1 - \epsilon_x \text{ pour tout } (x, u) \in \mathcal{E} \end{cases} \quad \text{ou} \quad \begin{cases} \text{Maximiser } \sum\limits_{(x,u)\in\mathcal{E}} \alpha_x - \frac{1}{2}\| \sum\limits_{(x,u)\in\mathcal{E}} \alpha_x u x\|^2 \\ \text{avec } 0 \leq \alpha_x \leq C \text{ pour tout } (x, u) \in \mathcal{E} \\ \text{et } \sum\limits_{(x,u)\in\mathcal{E}} \alpha_x u = 0 \end{cases}$$

Exemple : *Ranking SVM* [Joa02]

Le problème : un ensemble de documents D, une requête $q \Rightarrow$ ordonner les documents de D en fonction de la requête.

Pour apprendre à ordonner, exemples de la forme (q, d_1, d_2) :
« pour la requête q, le document d_1 est plus pertinent que d_2 ».

$\Rightarrow$ Paires de vecteurs $(\Phi(q, d_1), \Phi(q, d_2))$, où $\Phi(q, d)$ indique l'adéquation entre la requête q et le document d, avec par exemple le nombre de mots communs, le *page-rank* de d, etc...
On cherche un vecteur a tel que $a \cdot \Phi(q, d_1) > a \cdot \Phi(q, d_2)$ pour tout exemple de cette forme.

$\Leftrightarrow$ trouver a tq $a \cdot (\Phi(q, d_1) - \Phi(q, d_2)) > 0$, pour tout exemple (q, d_1, d_2)

Maximisation de la marge :

$$\begin{cases} \text{Minimiser } \frac{1}{2}\|a\|^2 + C \sum\limits_{e\in\mathcal{E}} \epsilon_e \\ \text{avec } a \cdot (\Phi(q, d_1) - \Phi(q, d_2) \geq 1 - \epsilon_e \text{ pour tout } e = (q, d_1, d_2) \in \mathcal{E} \end{cases}$$

Conclusions

- Méthode appliquée dans de très nombreux domaines
- Plusieurs librairies existent, permettant d'utiliser la méthode en `C++`, Java, ..., et dans d'autres logiciels : `R`, Matlab
- Les choix de la fonction à noyau et du paramètre C sont prépondérants
- Méthode « *boite noire* », c'est-à-dire qu'on ne peut pas « comprendre » le modèle appris
- Actuellement, difficile d'apprendre avec plus de 10^5 exemples
- Une fois le modèle appris, classification très facile à calculer

Bibliographie

[Joa02] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 133–142. ACM, 2002.

[YHL12] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584 –2603, sept. 2012.

Exercices

Exercice 1 Soit $\mathcal{E} = \{((2,2),+1),((4,2),-1),((3,3),+1),((3,1),-1)\}$: montrer qu'on obtient une séparatrice linéaire en 4 itérations avec l'algorithme du percéptron, et en une itération avec la descente en gradient.

Exercice 2 La méthode des Séparateurs à Vaste Marge utilise une fonction à *noyau* k, qui doit être exprimable sous la forme $k(x,x') = \varphi(x) \cdot \varphi(x')$ pour une certaine fonction $\varphi : \mathcal{X} \to \mathbf{R}^d$.

Pour classer des documents (méls, pages web, rapports,...) on peut définir un ensemble de mots $\mathcal{A}$ (on considère que ce sont les mots « importants »), associer à chaque mot $m \in \mathcal{A}$ un indice compris entre 1 et le nombre de mots de $\mathcal{A}$, puis associer à chaque document x un vecteur $\varphi_1(x)$, tel que $\varphi_1(x)_i = 1$ si le mot d'indice i est présent dans le document, et $\varphi_1(x)_i = 0$ sinon.

Question 2.1 Que représente $k_1(x,x') = \varphi_1(x) \cdot \varphi_1(x')$ dans ce cas ?

Question 2.2 Comment pourrait-on définir une fonction k_1' qui prenne en compte le nombre d'occurrences des mots dans un document ? Quel est l'intérêt d'une telle modification ?

Question 2.3 Pour classer des pages web, on peut aussi s'intéresser aux liens entre les pages. Expliquer comment on peut définir une fonction φ_2, et la fonction à noyau de similarité correspondante k_2, qui rende compte des liens entre les pages.

Question 2.4 Naturellement, on peut s'intéresser à la fois aux mots et aux liens que des pages web ont en commun. Définissez simplement, à partir de k_1 et k_2, une fonction à noyau k (et le noyau φ correspondant) permettant de prendre en compte les deux aspects.

Question 2.5 Si on a un ensemble d'exemples contenant D documents déjà classés, si on considère un vocabulaire ayant M mots, et si on suppose qu'il y a L liens possibles, quel est le nombre de variables du problème d'optimisation – dans la version duale – à résoudre pour définir le classifieur ? Et dans la version primale ? Quel est l'ordre de grandeur du temps de calcul de la matrice de Gramm correspondant à la fonction k ?

...