Cours IFT6266,
Modèles à noyau (*Kernel Machines*)

Modèles à noyau classiques

- K-plus-proches voisins:

$$f_D(x) = \sum_i y_i \frac{K(x, x_i)}{\sum_j K(x, x_j)} = \sum_i y_i \tilde{K}_D(x, x_i)$$

  avec $K(u, v) = 1_{v \in \mathcal{V}_k(u)}$, et $\mathcal{V}_k(v)$ le sous-ensemble de D contenant les k plus proches voisins de v dans D.

- Fenêtre de Parzen: même forme, mais $K(u, v)$ continu, e.g. avec le noyau Gaussien

$$K(u, v) = e^{-||u-v||^2/\sigma^2}.$$

  On s'en sert plutôt pour la régression.

- Plus généralement, les modèles à noyau ont cette forme:

$$f_D(x) = \sum_i \alpha_i K_D(x, x_i)$$

  où K_D peut dépendre des données D (généralement que des entrées) et les α_i sont donnés par l'algorithme (généralement en tenant compte des sorties désirées y_i).

Le Truc du Noyau (*kernel trick*)

- Le truc du noyau est une astuce qui permet de transformer beaucoup d'algorithmes "linéaires" (e.g. régression, classification, réduction de dimensionalité) en algorithmes "non-linéaires" tout en conservant les même propriétés d'optimisation facile (convexe).

- Dans ces algorithmes on a une notion de produit scalaire $x.y$, qu'on va remplacer par un produit scalaire dans un espace de haute dimension (infinie en général), $\phi(x).\phi(y)$. Par exemple la prédiction linéaire $w.x$ devient une prédiction non-linéaire $w.\phi(x)$.

- **Le TRUC** est qu'on a des **fonctions à noyau** $K(x, y)$ qui ont la propriété qu'elles correspondent à un produit scalaire: $\exists \phi : K(x, y) = \phi(x).\phi(y)$. On a plus jamais à calculer $\phi(.)$, seulement $K(.,.)$. La condition suffisante: $K(.,.)$ est positive semi-définie.

- La fonction à noyau la plus simple est le noyau Gaussien: $K(x, y) = e^{-\frac{1}{2}||x-y||^2/\sigma^2}$. On peut facilement composer (additionner, multiplier, etc...) des noyaux positifs semi-définis pour en obtenir d'autres. Il y a souvent des hyper-paramètres dans les noyaux (e.g. le σ). Le choix du noyau est crucial et correspond à un choix a priori sur la notion de similarité entre exemples.

Régression à noyau

Voici une application du truc du noyau à la régression linéaire, pour la rendre non-linéaire.

1. On se rappelle que dans la régression linéaire ordinaire avec n exemples en dimension d et pénalisation quadratique λ on veut minimiser

$$(\mathbf{y} - Xw)'(\mathbf{y} - Xw) + \lambda w'w$$

où la prédiction au point x est $f(x) = w'x$, et les exemples sont les paires (x_i, y_i), avec x_i la i-ème rangée de la matrice X, et w, $\mathbf{y}$ des vecteurs de taille d et n respectivement. Dans ce cas la solution est

$$w = (X'X + \lambda I)^{-1} X'\mathbf{y}$$

On peut montrer que w peut s'écrire comme une combinaison linéaire des x_k:

$$w = \sum_k \alpha_k x_k. \tag{1}$$

2. On va s'intéresser au cas où $d > n$, car la prédiction est non-linéaire en x: $f(x) = w'\phi(x)$ avec $dim(\phi(x)) = d > n$. De plus on va supposer que $K(x, y) = \phi(x)'\phi(y)$ peut se calculer en $dim(x)$ opérations plutôt qu'en $dim(\phi(x))$ opérations. On va écrire R la matrice $n \times d$ dont les rangées sont les $\phi(x_k)$.

   On va reparamétriser l'optimisation du critère en terme de α plutôt que w puisque $dim(\alpha) = n < dim(\phi(x)) = d$. Notons que

$$w = R'\alpha$$

   et la prédiction sur les exemples d'apprentissage s'écrit $Rw = RR'\alpha$. En annulant le gradieant par rapport à α,

   on obtient

$$\alpha = (RR' + \lambda I)^{-1}\mathbf{y}$$

   où RR' est la matrice de Gram M dont les éléments (i, j) sont $K(x_i, x_j)$.

3. La prédiction au point x s'écrit donc

$$f(x) = k(x)'(M + \lambda I)^{-1}\mathbf{y}$$

   quand on définit $k(x)$ le vecteur dont les éléments sont $K(x, x_i)$.

   C'est la même solution que pour l'espérance de Y étant donné x avec un processus Gaussien, mais sans les hypothèses distributionnelles explicites sur la loi à priori de f.

Régression par Processus Gaussiens

On peut généraliser la régression à noyau afin d'obtenir un estimateur de l'incertitude (au sens Bayesien) autour de la prédiction.

Voir la présentation d'Olivier Delalleau intitulée `Introduction aux Processus Gaussiens`, sur le site du cours.

Classificateurs à marge maximale

Le classificateur à marge maximale est une généralisation de l'algorithme du Perceptron. On introduit la notion de marge (une sorte de marge de sécurité) dans le critère d'apprentissage. On exige non seulement que les exemples soient bien classifiés, mais qu'ils soient à une distance au moins 1 de la surface de décision, tout en empêchant les poids de devenir trop grand. Dans le cas binaire, le critère est donc:

$$Q = \frac{1}{2}||w||^2 + C\sum_i (f(x_i)y_i - 1)_+$$

avec $f(x) = b + w'x$ dans le cas d'un classifieur linéaire à marge maximale.

L'avantage par rapport au Perceptron est une meilleure généralisation, dûe à une plus grande robustesse. Vapnik (livres de 1995) montre que l'on peut borner l'erreur de généralisation même si le nombre de dimension d'entrée est très grand, en autant que les deux termes ci-haut soient minimisés. Le premier contrôle la marge et le second les "erreurs".

On peut minimiser Q par descente de gradient stochastique, gradients conjugués, ou bien en l'exprimant comme un problème d'optimisation quadratique sous contraintes:

$$\min ||w||^2 + C\sum_i \xi_i$$

avec

$$f(x_i)y_i \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

Avec $C \to \infty$ on a une contrainte dure: tous les exemples doivent être du bon côté de la marge. En général une meilleure solution est obtenue avec un $C > 0$ fini. C'est un hyper-paramètre de régularisation.

SVMs

Un SVM généralise le classificateur à marge maximale, en y applicant le truc du noyau.

Avec le noyau linéaire $K(u, v) = u \cdot v$ on retombe sur le classifieur linéaire à marge maximale.

Le noyau le plus communément utilisé est le noyau Gaussien. Dans ce cas, on a donc deux hyper-paramètres: le C de régularisation, et le σ du noyau Gaussien.

La solution s'exprime sous la forme

$$f_D(x) = b + \sum_i \alpha_i K(x, x_i).$$

Avec le critère de marge, beaucoup de α_i sont 0: solution relativement sparse (mais asymptotiquement on reste généralement avec une fraction du total des exemples d'apprentissage, qui dépend du niveau de bruit dans la distribution génératrice). On appelle "points de support" (*support vectors*) les x_i correspondant à $\alpha_i \neq 0$.

Avec n exemples, l'implantation exacte requière $O(n^2)$ mémoire pour stocker tous les $K(x_i, x_j)$, et $O(n^3)$ calculs (au pire cas) pour résoudre le problème d'optimisation quadratique. De bonnes implantation donnent des temps de calcul entre $O(n^2)$ et $O(n^3)$ dans des cas pratiques.

Beaucoup d'implantations gratuites existent sur internet. Voir `www.kernel-machines.org`. C'est un des algorithmes d'aprentissage les plus populaires aujourd'hui: prédicteur non-linéaire dont l'optimisation est un problème convexe, donc sans minima locaux.

Il y a de la recherche pour faire des versions "on-line" approximatives, qui ne seraient pas $O(n^2)$ ou $O(n^3)$ dans le nombre d'exemples, mais plus près de $O(n)$. Le problème reste que le nombre de points de support tend à être élevé (p.e. comparé à un réseau de neurones profond).

On peut aussi généraliser les SVMs pour la régression. Pour le cas de l'erreur quadratique on obtient la régression à noyau classique (et pas de sparsité). Pour le cas du critère "tube-ϵ", on obtient un peu de sparsité. On minimise $|f(x_i) - y_i|_\epsilon$ plutôt que $(f(x_i)y_i - 1)_+$ ci-haut, avec $|x|_\epsilon = (|x| - \epsilon)_+$. On insiste pour que l'erreur soit au maximum ϵ mais on pénalise les dépassements linéairement. L'optimisation est encore quadratique (donc convexe).