IDIAP COMMUNICATION

IDIAP
Martigny - Valais - Suisse

Support Vector Machines
Théorie et Application

Ronan Collobert [1]

IDIAP–Com 00-03

septembre 2000

Institut Dalle Molle d'Intelligence Artificielle Perceptive • CP 592 • Martigny • Valais • Suisse

téléphone +41−27−721 77 11
télécopieur +41−27−721 77 12
adr.él. secretariat@idiap.ch
internet http://www.idiap.ch

[1] IDIAP, CP 592, 1920 Martigny, Switzerland, collober@idiap.ch

Table des matières

Remerciements

Je tiens à remercier Samy Bengio pour m'avoir proposé ce stage. Il a bien failli faire disparaitre mes préjugés sur les SVM. Sans lui rien n'aurait été. Je remercie Daniel Collobert[1] pour m'avoir trouvé ce stage. Sans lui rien ne serait.

Je remercie aussi Frank pour m'avoir trouvé un bureau digne d'un ministre corrompu, Nicolas pour ses discours philosophiques, Miguel pour son inestimable thé vert, Mikhael pour ses notions de russe, Johnny qui a failli me ruiner avec ses restaurants et m'achever avec ses fondues, et enfin Perry pour son expérience enrichissante, ses fêtes hallucinantes et ses ballades à vélo épuisantes. Bref je remercie le groupe *Machine Learning* qui m'a permis de travailler et surtout de survivre.

Je remercie aussi toute l'IDIAP pour son accueil, et tout particulièrement Datong pour son smash fulgurant au volley, Todd qui a réussi l'exploit (non renouvelé) de me faire lever avant 8h, Astrid pour ses gateaux (digne de l'appelation Bretonne contrôlée) et bien-sûr Hervé pour mon salaire.

Enfin je remercie les banque Suisses pour m'avoir fourni un compte inviolable, Compaq pour me prêter indéfiniment des machines 800Mhz et autres quadri-processeurs Alphas et *Matmatah* pour la musique d'ambiance sans laquelle les idées ne seraient jamais venues.

1. Toute ressemblance avec une personne de ma famille serait purement fortuite.

CHAPITRE 1 Introduction

Isaac Asimov quand il écrivit *Les Robots* [1] s'imaginait qu'avant la fin du second millénaire il y aurait dans toutes les maisons des machines douées de comportements humains, capables d'apprendre et capables de se sortir de situations non apprises à l'avance.

Il était bien optimiste.

Quand je vois que l'on n'est même pas capable de reproduire le comportement d'une mouche de manière artificielle, ça me fait bien rire. C'est simple : quand on observe un neurone biologique à la loupe on voit à peu près ce qu'il se passe, et quand on en observe deux connectés on ne comprend plus rien du tout.

Cependant les scientifiques ont essayé de modéliser du mieux qu'ils pouvaient quelques comportements de base des animaux. Plusieurs directions furent prises : celle de *l'intelligence artificielle* où l'idée était de construire une machine dont les comportements dits *intelligents* étaient modélisés par des manipulations de symboles. Cette filière, bien que très prometteuse au début, est aujourd'hui en voie d'extinction. L'autre direction prise (dans laquelle les scientifiques méprisent profondément ceux qui ont suivi la première) est celle de *l'apprentissage statistique*, dans laquelle on essaye de fabriquer des machines mûes par des algorithmes capable de généralisation, c'est-à-dire capable de réagir à une situation non prévue à l'avance. C'est ce qui nous intéresse dans ce mémoire.

Nous allons donc commencer par introduire la notion d'algorithme d'apprentissage, puis nous allons nous focaliser sur un algorithme en particulier : les *support vector machines*. Cet algorithme est très récent, la première publication [2] sur le sujet étant parue en 1992. Pour ceux qui connaissent un peu le domaine de l'*apprentissage statistique*, les *support vector machines* feraient au moins aussi bien que les *perceptrons multicouches*, d'après Vladimir Vapnik [29].

Nous verrons de près comment l'algorithme des *support vector machines* fonctionne, nous résoudrons les problèmes mathématiques qu'il amène et nous terminerons par un exemple d'application.

CHAPITRE 2 Théorie de l'apprentissage statistique

L'apprentissage, c'est-à-dire la capacité d'acquérir de nouveaux comportements ou d'en modifier d'anciens par expérience, est une propriété de presque tous les animaux, y compris ceux que l'on peut difficilement qualifier d'intelligents.
E. Kandel, *Cellular basis of behaviour*

2.1 Apprendre avec un nombre limité d'exemples

À la différence de E. Kandel, nous allons nous placer ici dans un cadre mathématique tel qu'introduit par Vapnik [28, 29]. Nous allons d'ailleurs commencer par définir la notion *d'algorithme* d'apprentissage.

- Soit Ω un espace probabilisé et $\mathcal{Z}$ un espace muni d'une mesure λ. Considérons le $l-$ *uplet*

$$D_l = \{Z_1, Z_2, \dots, Z_l\}$$

  où les Z_i sont des variables aléatoires sur Ω à valeurs dans $\mathcal{Z}$, identiquement et indépendemment distribuées, muni d'une **densité inconnue** F par rapport à λ. Pour $\omega_0 \in \Omega$ fixé à l'avance, on appellera *exemples d'apprentissage*

$$D_l^{\omega_0} = \{Z_1(\omega_0), Z_2(\omega_0), \dots, Z_l(\omega_0)\}$$

  un échantillon de taille l de la distribution de probabilité de densité F.
- Soit $\mathcal{F}$ un ensemble de fonctions. On définit sur $Z \times \mathcal{F}$ une fonction réelle que l'on appellera notre *critère local* :

$$Q : \mathcal{Z} \times \mathcal{F} \to \boldsymbol{R}$$

Un *algorithme d'apprentissage* est un algorithme[1] qui cherche à minimiser sur $\mathcal{F}$ l'application

$$R : f \longmapsto \int_{\mathcal{Z}} Q(z, f) F(z)\, d\lambda(z)$$

avec pour seule connaissance $D_l^{\omega_0}$.

1. Le mot «algorithme» sert à désigner toute méthode de calcul permettant de résoudre un problème et qui se présente sous la forme d'une suite d'opérations élémentaires obéissant à un enchaînement déterminé. L'origine du mot est issue du nom du mathématicien persan (IV siècle après J.-C.) qui introduisit la notion de zéro dans la culture occidentale : Muhammad ibn Musa abu Abdallah al-Khoreami al-Madjusi al-Qutrubulli, dont le nom usuel al Khorezmi fut latinisé au moyen-Âge en «Algorismus». On pouvait craindre pire.

On appelle $R(f)$ le *risque espéré* ou l'*erreur de généralisation* de f. Ce problème de minimisation est souvent appellé problème d'*induction*. Malheureusement ce n'est pas un «bête» problème d'optimisation car F est inconnue.

On distingue habituellement trois principaux problèmes d'apprentissage :

- Si $f \in \mathcal{F}$ est une fonction de la forme $f : \mathcal{X} \to \mathcal{Y}$, et $Z_i = (X_i, Y_i)$ est une variable aléatoire[2] à valeurs dans $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, on distingue deux catégories :
  - La *classification* : Y_i appartient à un ensemble fini, et pour $z = (x, y) \in \mathcal{Z}$, on pose en général $Q(z, f) = 0$ si $f(x) = y$ et 1 sinon.
  - La *régression* : $Y_i \in \boldsymbol{R}^n$ et on prend par exemple $Q(z, f) = (f(x) - y)^2$.
- On peut aussi faire de *l'estimation de densité* : $f \in \mathcal{F}$ est alors une fonction de $\mathcal{Z}$ à valeur dans $\boldsymbol{R}$, strictement positive, et telle que $\int_{\mathcal{Z}} f(z)\, d\lambda(z) = 1$. On prend par exemple $Q(f, z) = -\log f(z)$.

On s'intéresse donc à la minimisation du risque espéré. Comme on ne peut pas le calculer puisque F est inconnue, on définit le *risque empirique* ou *erreur d'apprentissage* pour $D_l^{\omega_0}$ et $f \in \mathcal{F}$ comme étant la quantité réelle

$$\hat{R}(f, D_l^{\omega_0}) = \frac{1}{l} \sum_{i=1}^{l} Q(Z_i(\omega_0), f)$$

qui mesure l'erreur de f sur l'ensemble d'exemples $D_l^{\omega_0}$.

Remarquez que $\hat{R}(f, D_l^{\omega_0})$ est une quantité tout à fait calculable et que $\hat{R}(f, D_l)$ est un estimateur non biaisé de $R(f)$ pour un f quelconque choisi *indépendemment* de D_l :

$$\boldsymbol{E}[\hat{R}(f, D_l)] = R(f)$$

Donc, plutôt que d'essayer de minimiser R, on essaye en général de *minimiser le risque empirique*, en espérant que l'on s'approche en même temps du minimum de R. C'est un principe à la base d'un grand nombre d'algorithmes d'apprentissage. On cherche donc :

$$f^\star_{D_l^{\omega_0}} = \operatorname*{arginf}_{f \in \mathcal{F}} \hat{R}(f, D_l^{\omega_0})$$

De l'égalité

$$\hat{R}(f^\star_{D_l^{\omega_0}}, D_l^{\omega_0}) = \inf_{f \in \mathcal{F}} \hat{R}(f, D_l^{\omega_0})$$

on déduit que $\hat{R}(f^\star_{D_l}, D_l)$ est un estimateur *optimiste* de $R(f^\star_{D_l^{\omega_1}})$, pour tout $\omega_1 \in \Omega$. En effet :

$$\boldsymbol{E}[R(f^\star_{D_l^{\omega_1}}) - \hat{R}(f^\star_{D_l}, D_l)] \geq 0$$

Donc

$$\boldsymbol{E}[\hat{R}(f^\star_{D_l^{\omega_1}}, D_l) - \hat{R}(f^\star_{D_l}, D_l)] \geq 0$$

Ainsi, la solution $f^\star_{D_l^{\omega_0}}$ obtenue en minimisant $\hat{R}(f, D_l^{\omega_0})$ est meilleure en générale sur $D_l^{\omega_0}$ que sur un autre ensemble $D_l^{\omega_1}$ tiré au hasard avec la distribution de densité F. De plus $f^\star_{D_l^{\omega_1}}$ ne peut pas être en générale identique à $f^\star_{D_l^{\omega_0}}$. *A fortiori* $f^\star_{D_l^{\omega_0}}$ et $f^\star_{D_l^{\omega_1}}$ ne peuvent pas être identique en général à la solution «optimale» qui minimise le risque espéré sur $\mathcal{F}$.

2. Pour un couple $(X_i(\omega_0), Y_i(\omega_0))$ donné, on appelle habituellement $X_i(\omega_0)$ *l'entrée désirée* et $Y_i(\omega_0)$ *la sortie désirée.*

2.2 Un problème de convergence uniforme

Nous allons essayer de montrer dans ce paragraphe que, sous certaines conditions, la minimisation du risque empirique nous permet de nous approcher du minimum du risque espéré. Pour cela nous allons dans un premier temps borner la différence entre le risque empirique et risque espéré.

Nous savons déjà par la loi faible des grands nombres que pour un $f \in \mathcal{F}$ donné

$$\forall \epsilon > 0 \quad \boldsymbol{P}\left[|\hat{R}(f, D_l) - R(f)| > \epsilon\right] \underset{l \to \infty}{\longrightarrow} 0$$

Ce résultat est affiné par l'inégalité de Hoeffding [12], qui s'applique si on connait un majorant τ de $(\sup Q - \inf Q)$:

$$\forall \epsilon > 0 \quad \boldsymbol{P}\left[|\hat{R}(f, D_l) - R(f)| > \tau\, \epsilon\right] \leq 2e^{-2\epsilon^2 l} \tag{2.1}$$

Mais on ne sait pas pour autant que le minimum du risque empirique converge vers le minimum du risque espéré. Rien ne prouve en effet que cette convergence se fait à la même vitesse pour deux f distincts. Rien ne laisse donc supposer que

$$R(f_1) < R(f_2) \Longleftrightarrow \hat{R}(f_1, D_l^{\omega_0}) < \hat{R}(f_2, D_l^{\omega_0})$$

pour l assez grand.

Si par contre nous avions la majoration

$$\sup_{f \in \mathcal{F}} |\hat{R}(f, D_l^{\omega_0}) - R(f)| \leq \epsilon$$

alors nous pourrions majorer

$$|R(f^\star_{D_l^{\omega_0}}) - \inf R(f)| \leq |R(f^\star_{D_l^{\omega_0}}) - \hat{R}(f^\star_{D_l^{\omega_0}}, D_l^{\omega_0})| + |\inf \hat{R}(f, D_l^{\omega_0}) - \inf R(f)| \leq 2\epsilon \tag{2.2}$$

Il nous faut donc trouver une condition nécessaire sur R pour avoir la convergence uniforme en probabilité

$$\forall \epsilon > 0 \quad \boldsymbol{P}\left[\sup_{f \in \mathcal{F}} |\hat{R}(f, D_l) - R(f)| > \epsilon\right] \underset{l \to \infty}{\longrightarrow} 0$$

qui nous permettrait d'obtenir ce que l'on veut.

Les travaux de Vapnik et Chervonenkis [28, 29] sur ce problème nous apprennent qu'une condition nécessaire pour cette convergence dépend ce qu'ils appellent la *capacité*[3] h de $\mathcal{F}$, qui est une mesure de sa complexité.

Par exemple pour la classification, la capacité h de $\mathcal{F}$ est le plus grand l tel qu'il existe un ensemble d'exemples D_l^ω tel qu'on puisse toujours trouver un $f \in \mathcal{F}$ qui donne la bonne réponse sur tous les exemples de D_l^ω, quel que soit l'étiquetage de ces exemples.

Vapnik et Chervonenkis énoncent alors le théorème suivant :

Théorème 2.1 *Si $\mathcal{F}$ est de capacité h* finie, *que $l > h$ et $\tau = \sup Q - \inf Q$ alors pour un $\eta > 0$ donné, on a*

$$\boldsymbol{P}\left[\sup_{f \in \mathcal{F}} |\hat{R}(f, D_l) - R(f)| \geq 2\tau \sqrt{\frac{h\left(\ln \frac{2l}{h} + 1\right) - \ln \frac{\eta}{9}}{l}}\right] \leq \eta \tag{2.3}$$

3. Encore appellée dimension de Vapnik-Chervonenkis (VC-dim)

En utilisant (2.2) il vient alors

$$\boldsymbol{P}\left[\; R(f^\star_{D_l}) - \inf R(f) \leq 4\tau\sqrt{\frac{h\left(\ln\frac{2l}{h}+1\right)-\ln\frac{\eta}{9}}{l}}\;\right] \geq 1-\eta \tag{2.4}$$

Autrement dit avec une confiance $1-\eta$ donnée, le risque espéré de la fonction qui minimise le risque empirique se rapproche du minimum sur $\mathcal{F}$ du risque espéré lorsque le nombre d'exemples d'apprentissage augmente, et ce, d'autant plus vite que la capacité de $\mathcal{F}$ est faible. Ce qui est bien le genre de résultat que l'on recherchait.

Deux remarques supplémentaires méritent d'être évoquées :

- Notez que l'on peut aussi déduire de (2.3) et (2.2) que

$$\boldsymbol{P}\left[\; |\hat{R}(f^\star_{D_l}, D_l) - \inf R(f)| \leq 3\tau\sqrt{\frac{h\left(\ln\frac{2l}{h}+1\right)-\ln\frac{\eta}{9}}{l}}\;\right] \geq 1-\eta \tag{2.5}$$

  De (2.4) et (2.5) on conclut que, à capacité finie, le risque espéré et le risque empirique de $f^\star_{D_l^{\omega_0}}$ convergent tous les deux *en probabilité* vers le minimum du risque espéré. On dit que l'algorithme d'apprentissage est *consistent*. Vapnik a en fait montré que la finitude de la capacité est aussi une condition *nécessaire* pour cette consistence.
- De (2.3) on déduit qu'avec une confiance $1-\eta$ on a

$$R(f^\star_{D_l^{\omega_0}}) \leq \hat{R}(f^\star_{D_l^{\omega_0}}, D_l) + 2\tau\sqrt{\frac{h\left(\ln\frac{2l}{h}+1\right)-\ln\frac{\eta}{9}}{l}} \tag{2.6}$$

  Cette équation nous donne une borne sur le risque espéré, après calcul du risque empirique. Comme il est facile de voir que le risque empirique augmente avec le nombre d'exemples d'apprentissage l, nous sommes en mesure de tracer l'allure des courbes données sur la FIG. 2.1. Ainsi, à l fixé nous avons une borne supérieure de l'erreur de généralisation (courbe rouge). De

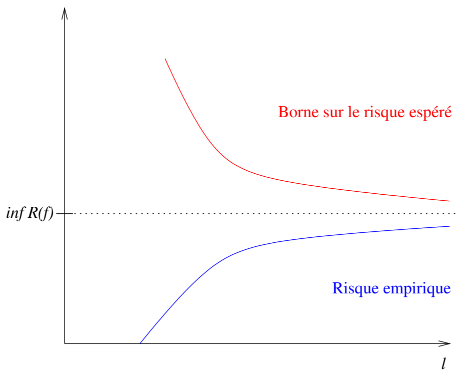

FIG. 2.1 – *Évolution de la borne sur le risque espéré et évolution du risque empirique à capacité fixe, en fonction du nombre d'exemples d'apprentissage.*

  plus, nous savons calculer le risque empirique et nous savons que celui-ci est croissant et converge vers le minimum du risque espéré : nous pouvons donc déduire que le risque espéré de $f^\star_{D_l^{\omega_0}}$ est *au dessus* du risque empirique. C'est une manière d'encadrer le risque espéré.

2.3 Le dilemme biais-variance

La majoration (2.4) est d'autant plus grossière que l'espace $\mathcal{F}$ est de capacité grande. Or rien ne sert d'avoir un meilleur optimum du risque empirique si cela n'améliore pas le risque espéré.

Pour avoir une idée de comment nous pouvons contrôler le risque espéré lorsque la capacité de $\mathcal{F}$ varie, munissons nous maintenant de N espaces de fonctions $\mathcal{F}_i$

$$\mathcal{F}_1 \subset \ldots \subset \mathcal{F}_N$$

dont les capacités h_i sont ordonnées (et finies)

$$h_1 \leq \ldots \leq h_N$$

En appelant $f^{\star,i}_{D_l^{\omega_0}}$ le minimum du risque empirique sur $\mathcal{F}_i$, on peut déduire de (2.3) que l'on a toujours, avec une probabilité $1-\eta$,

$$R(f^{\star,i}_{D_l^{\omega_0}}) \leq \hat{R}(f^{\star,i}_{D_l^{\omega_0}}, D_l^{\omega_0}) + \underbrace{2\tau\sqrt{\frac{h_i\left(\ln\frac{2l}{h_i}+1\right)-\ln\frac{\eta}{9}}{l}}}_{\text{Largeur de l'intervalle de confiance}}$$

Comme on a clairement

$$\hat{R}(f^{\star,N}_{D_l^{\omega_0}}, D_l^{\omega_0}) \leq \ldots \leq \hat{R}(f^{\star,1}_{D_l^{\omega_0}}, D_l^{\omega_0})$$

on en déduit sur la Fig. 2.2 l'allure de la borne sur l'erreur du risque espéré. Même si l'allure de la

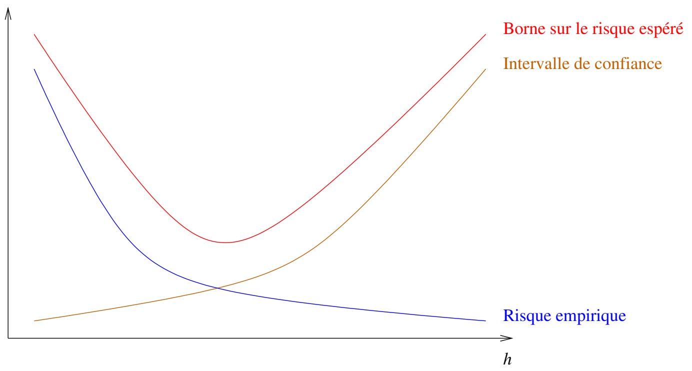

Fig. 2.2 – *Borne sur le risque espéré*

courbe qu'on obtient est celle d'une *borne* du risque espéré, l'expérience montre que le risque espéré *lui-même* suit ce genre de variations. Ainsi, dans un premier temps il décroit : on dit alors que f *underfit* (ne colle pas assez aux données). Puis le risque passe par un minimum et croît : on dit alors que f *overfit* (colle trop aux données).

Ce comportement s'explique souvent de manière plus simple par ce que l'on appelle *le dilemme biais-variance*. On décompose le risque espéré en trois parties :

- L'erreur dûe au fait que l'ensemble de fonctions $\mathcal{F}$ ne contient pas nécessairement la solution optimale au problème. On appelle parfois cette erreur *le biais*.

- L'erreur dûe au fait que $f^\star_{D_l^{\omega_0}}$ n'est pas forcément la meilleure fonction dans $\mathcal{F}$: elle minimise $\hat{R}(f, D_l^{\omega_0})$ mais pas forcément $R(f)$. On appelle parfois cette partie de l'erreur *la variance* car elle provient de la variabilité entre différents ensembles d'apprentissage possibles tirés de la distribution de densité F.
- L'erreur dûe au *bruit* : au fait que la solution optimale se trompe parfois. Par exemple, si l'on recherche une fonction de X vers Y, il n'existe pas de solution parfaite quand la relation entre X et Y n'est pas une fonction (si pour chaque valeur de X il y a plusieurs valeurs de Y possible).

Le dilemme biais-variance exprime le conflit entre le désir de réduire l'erreur de biais et de réduire l'erreur de variance. En effet lorsque l'on augmente la capacité de $\mathcal{F}$, l'erreur de biais diminue (parce qu'on a alors plus de chance d'obtenir une solution proche de la solution du problème), mais l'erreur de variance augmente (parce que le nombre de solutions adaptées à l'ensemble d'apprentissage augmente, et on a donc plus de chance d'obtenir une solution plus adaptée à cet ensemble, et moins adaptée globalement). Ce qui nous donne l'idée du comportement du risque espéré proposé à la FIG. 2.2.

2.4 Mesure de la qualité d'une solution

Malheureusement pour nous, la borne donnée par (2.4) (aussi bien que celle donnée par (2.6)) est en pratique quasiment inutilisable, à cause du fait que la capacité de $\mathcal{F}$ est souvent extrêmement élevée[4], voire infinie.

Pour estimer en pratique le risque espéré de $f \in \mathcal{F}$, et donc pour mesurer sa qualité, on dispose souvent d'un échantillon aléatoire $D_l^{\omega_1}$ (appelé *ensemble de test*) *différent* des exemples d'apprentissage. On peut alors encadrer le risque espéré en utilisant l'inégalité de Hoeffding (2.1) : avec une probabilité $1 - \eta$, nous avons

$$\hat{R}(f^\star_{D_l^{\omega_0}}, D_l^{\omega_1}) - \tau \sqrt{\frac{-\ln \frac{\eta}{2}}{2l}} \leq R(f^\star_{D_l^{\omega_0}}) \leq \hat{R}(f^\star_{D_l^{\omega_0}}, D_l^{\omega_1}) + \tau \sqrt{\frac{-\ln \frac{\eta}{2}}{2l}} \tag{2.7}$$

où τ est un majorant de $(\sup Q - \inf Q)$.

Ainsi, avec $\tau = 1$ (ce qui est le cas de la classification), il suffit de 15000 exemples[5] dans $D_l^{\omega_1}$ pour mesurer un taux d'erreur à $\pm 1\%$, avec une probabilité de 0.9.

Maintenant que nous savons estimer le risque espéré, nous verrons par la suite que dans la pratique celle-ci suit bien l'allure des courbes proposées par la théorie aux figures FIG. 2.2 et FIG. 2.1.

4. Auquel cas il nous faudrait énormément d'exemples d'apprentissage pour pouvoir utiliser (2.4), ce qui est la plupart du temps impossible à traiter avec les ordinateurs actuels.
5. Ce qui entre sans problème dans les possibilités des ordinateurs actuels.

CHAPITRE 3 Support Vector Machines

3.1 Un problème de classification

Soit $\mathcal{O}$ un ensemble d'*objets* quelconques décrits par un nombre fixe d de *caractéristiques* et munis d'une *classe*, prise parmi deux possibles. Prenons un sous-ensemble $\mathcal{S}$ de $\mathcal{O}$. Étant donné les paramètres d'un objet de $\mathcal{O} \setminus \mathcal{S}$, on veut pouvoir estimer le mieux possible la classe de cet objet avec la seule connaissance de $\mathcal{S}$.

Ici $\mathcal{S}$ est donc l'ensemble d'*apprentissage* et $\mathcal{O} \setminus \mathcal{S}$ l'ensemble de *test*.

Mathématiquement parlant, on représente $\mathcal{O}$ par un ensemble de couples $(x_i, y_i)_{1 \leq i \leq n}$ où x_i est un point dans $\boldsymbol{R}^d$ qui représente les paramètres, et $y_i = \pm 1$ représente la classe. Sans pertes de généralités, on prend $\mathcal{S}$ l'ensemble des couples $(x_i, y_i)_{1 \leq i \leq l}$ où $l < n$.

Il nous faut donc trouver un critère pour «séparer» le mieux possible les deux classes de $\mathcal{O}$ en utilisant au «maximum» les informations de $\mathcal{S}$.

Nous allons parler ici des idées introduites par Vapnik [28, 29] sur le sujet.

3.2 Une première idée de solution

Soit $\mathcal{S}$ un ensemble de points dans $\boldsymbol{R}^2$ de deux classes différentes, comme représentés Fig. 3.1. Il est clair qu'il existe une infinité de droites qui séparent les deux ensembles.

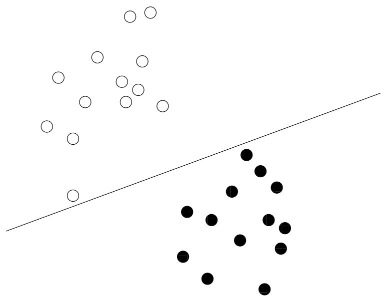

Fig. 3.1 – *Une droite séparatrice*

Dire que les deux ensembles sont séparables par une droite est équivalent à dire qu'il existe des constantes $w \in \boldsymbol{R}^2$ et $b \in \boldsymbol{R}$ telles que l'équation de la droite soit de la forme $(w \cdot x) + b = 0$ et que l'on a pour tout $1 \leq i \leq l$:

$$(w \cdot x_i) + b > 0 \; si \; y_i = +1$$

$$(w \cdot x_i) + b < 0 \; si \; y_i = -1$$

ce qui est équivalent à :

$$y_i((w \cdot x_i) + b) > 0 \; pour \; 1 \leq i \leq l$$

ce qui est équivalent à l'existence de constantes w et b telles que :

$$y_i((w \cdot x_i) + b) \geq 1 \; pour \; 1 \leq i \leq l \tag{3.1}$$

en «normalisant» par $\min_{1 \leq i \leq l} y_i((w \cdot x_i) + b)$ les variables précédentes.

Sur la Fig. 3.1 il est clair que la droite choisie n'est pas une très bonne séparatrice, du fait de sa proximité avec certains points. En effet, on peut imaginer des points de $\mathcal{O}$ qui seraient proches de ceux de $\mathcal{S}$ et qui pourraient être classés différemment, alors qu'intuitivement ils seraient de la même classe, comme le suggère les points grisés de la Fig. 3.2.

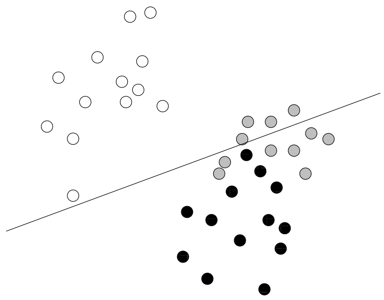

Fig. 3.2 – *Des points de $\mathcal{O}$ mal classés*

Pour pallier à ce genre de problème, on introduit une notion de *marge*, comme le propose la Fig. 3.3 : la marge ρ pour une droite séparatrice donnée normalisée comme en (3.1) est la distance entre la droite $(w \cdot x) + b = 1$ et la droite $(w \cdot x) + b = -1$. C'est-à-dire :

$$\rho = \left| \frac{1-b}{\|w\|} - \frac{-1-b}{\|w\|} \right| = \frac{2}{\|w\|} \tag{3.2}$$

Et une méthode, qui parait intuitivement intéressante pour choisir la droite séparatrice, est de choisir celle qui maximise cette marge.

Dans le cas de points dans $\boldsymbol{R}^d$ dont les classes sont *linéairement séparables*, on va simplement généraliser cette idée et prendre comme critère de séparation optimale un hyperplan qui maximise la marge entre les deux ensembles de points de classes différentes.

Le raisonnement ainsi que les équations restent les mêmes que dans $\boldsymbol{R}^2$.

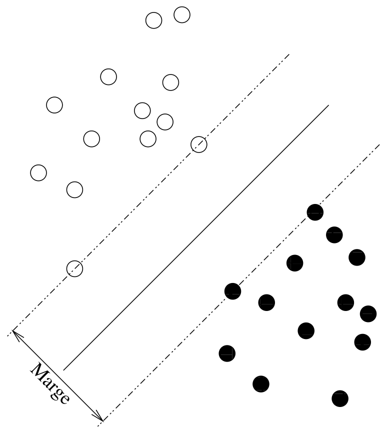

FIG. 3.3 – *Séparation optimale de deux classes de points*

3.3 À la recherche de l'hyperplan optimal dans le cas séparable

- Maintenant que l'idée est posée, il faut encore la résoudre mathématiquement. Il s'agit donc de trouver des constantes w et b vérifiant (3.1) qui maximisent (3.2). On peut utiliser une *méthode du Lagrangien*[1] avec contraintes :
  Posons :

  $$J(w,b) = \frac{\|w\|^2}{2} \tag{3.3}$$

  et

  $$\varphi_i(w,b) = 1 - y_i((w \cdot x_i) + b) \; pour \; 1 \leq i \leq l$$

  Notre problème est équivalent à minimiser (3.3) sous les contraintes :

  $$\varphi_i(w,b) \leq 0 \; pour \; 1 \leq i \leq l \tag{3.4}$$

  Posons maintenant le Lagrangien :

  $$L(w,b,\alpha) = J(w,b) + \sum_{1 \leq i \leq l} \alpha_i \, \varphi_i \tag{3.5}$$

- On sait d'après le THÉO. A.2 que si l'on trouve (w^o, b^o, α^o) point selle de L sur $(\boldsymbol{R}^d \times \boldsymbol{R}) \times \boldsymbol{R}_+^l$ alors (w^o, b^o) est solution de notre problème. Or les conditions **nécessaires** pour qu'il soit point selle sont, d'après la DÉF. A.1 :

  - (w^o, b^o) *est un minimum pour la fonction* $(w,b) \longrightarrow L(w,b,\alpha^o)$ *sur* $\boldsymbol{R}^d \times \boldsymbol{R}$. Comme celle-ci est clairement convexe et que l'on se place sur un ouvert, ceci est **équivalent** au fait que (w^o, b^o) annule sa dérivée. On doit donc avoir :

  $$\frac{\partial L(w^o, b^o, \alpha^o)}{\partial w} = 0 \iff w^o = \sum_{1 \leq i \leq l} \alpha_i^o y_i x_i \tag{3.6}$$

  $$\frac{\partial L(w^o, b^o, \alpha^o)}{\partial b} = 0 \iff \sum_{1 \leq i \leq l} \alpha_i^o y_i = 0 \tag{3.7}$$

1. Vous pouvez consultez l'annexe A ou [5, 11] pour la théorie sur l'optimisation non-linéaire.

- ∘ *α^o est un maximum pour la fonction $\alpha \longrightarrow L(w^o, b^o, \alpha)$ sur $\boldsymbol{R}_+^l$.* D'après ce qui précède, et si on substitue (3.6) et (3.7) dans (3.5), on obtient que α^o doit être **nécessairement** un maximum de...

$$\alpha \longrightarrow \sum_{1\le i\le l} \alpha_i - \frac{1}{2} \sum_{1\le i,j\le l} \alpha_i\alpha_j y_i y_j (x_i \cdot x_j) \tag{3.8}$$

  ...sous la contrainte (3.7).

- On sait de plus d'après le THÉO. A.1 que, puisque J et les contraintes φ_i sont convexes et dérivables, le fait que (w, b) soit un minimum de (3.3) sous les contraintes (3.4) **est équivalent** au fait qu'il existe des nombres α_i vérifiant les conditions suivantes, dites de Kuhn, Karush et Tucker (KKT) [11]...

$$J'(w,b) + \sum_{1\le i\le l} \alpha_i\, {\varphi'}_i(w,b) = 0$$

$$\alpha_i \geq 0 \textit{ pour } 1 \le i \le l \textit{ et } \sum_{1\le i\le l} \alpha_i \varphi_i(w,b) = 0$$

  Dans notre cas ces conditions de KKT sont équivalentes à :

$$\begin{aligned} w &= \sum_i \alpha_i y_i x_i && (3.9)\\ \sum_i \alpha_i y_i &= 0 && (3.10)\\ \alpha_i &\geq 0 && (3.11)\\ \alpha_i\,[y_i((w\cdot x_i)+b)-1] &= 0 && (3.12)\\ y_i((w\cdot x_i)+b)-1 &\geq 0 && (3.13) \end{aligned}$$

- Enfin, on sait aussi par le THÉO. A.2 que si (w, b) est solution de notre problème, alors il existe α tel que (w, b, α) soit un point selle de L. Nous allons donc nous intéresser à la recherche de point selle de L vérifiant les conditions de KKT. Ainsi :

  1. **Si on trouve** un α^o maximum de

  $$\sum_{1\le i\le l} \alpha_i - \frac{1}{2} \sum_{1\le i,j\le l} \alpha_i\alpha_j y_i y_j (x_i \cdot x_j)$$

     sous les contraintes

  $$\sum_{1\le i\le l} \alpha_i^o y_i = 0$$

     et

  $$\alpha_i^o \geq 0, \textit{ pour } 1 \le i \le l$$

  2. **Si on calcule** w^o par la formule :

  $$w^o = \sum_{1\le i\le l} \alpha_i^o y_i x_i$$

  3. **Si il existe** b^o vérifiant *toutes* les égalités et inégalités :

  $$\alpha_i^o\,[y_i((w^o\cdot x_i)+b^o)-1] = 0 \tag{3.14}$$

  $$y_i((w^o\cdot x_i)+b^o)-1 \geq 0 \tag{3.15}$$

...alors (w^o, b^o) sera solution de notre problème initial.
Plusieurs problèmes apparaissent :

- Nous devons trouver une méthode pour résoudre le 1). Cette résolution n'est pas aussi simple que l'on pourrait le croire, aussi nous en reparlerons dans un chapitre ultérieur.
- Il nous faut garantir l'existence d'un b^o vérifiant *les* équations (3.14) et (3.15).

- Une fois les α_i^o calculés, si l'on veut estimer la classe d'un point $x \in \mathcal{O}$ il suffit de calculer :

$$f(x) = \text{sign}\left[(w^o \cdot x) + b^o\right] = \text{sign}\left[\sum_{1 \leq i \leq l} \alpha_i^o y_i (x_i \cdot x) + b^o\right]$$

En fait vous pouvez remarquer grâce aux conditions de KKT que pour par exemple, $y_i = 1$ on a $\alpha_i = 0$ si $w \cdot x_i + b > 1$. Autrement dit α_i est nul si x_i est strictement en dehors de la marge telle que nous l'avons définie précédemment. Or, intuitivement comme le suggère la Fig. 3.3 (et en pratique) il y a peu de points x_i sur la marge. Et *a fortiori* il y a peu de points x_i avec un coefficient α_i non nul.
Ces points dont le coefficient est non nul s'appellent « points de support » ou plus couramment en anglais, *support vectors*.
Vu leur faible nombre en général, on *calculera* plutôt :

$$f(x) = \text{sign}\left[\sum_{support\ vectors} \alpha_i^o y_i (x_i \cdot x) + b^o\right] \tag{3.16}$$

3.4 Une autre idée de solution

Pour l'instant, nous avons donné une méthode pour trouver un hyperplan séparant de manière optimale deux classes de points, *à condition que ceux-ci soient linéairement séparables.* Ce qui a occasionné beaucoup de formules, mais qui n'est pas franchement glorieux.

Pour tenter de résoudre le problème non-linéairement séparable, une première idée très simple consiste à relâcher les contraintes [2] (3.1). Celles-ci deviennent :

$$y_i((w \cdot x_i) + b) \geq 1 - \xi_i \ pour\ 1 \leq i \leq l \tag{3.17}$$

avec

$$\xi_i \geq 0 \tag{3.18}$$

Autrement dit, pour chaque contrainte on s'autorise une erreur ξ_i positive que l'on veut, bien sûr, la plus petite possible.

- Nous voulons donc minimiser

$$J(w, b, \xi) = \frac{\|w\|^2}{2} + C \sum_{1 \leq i \leq l} \xi_i \tag{3.19}$$

sous les contraintes (3.17) et (3.18) où C est une constante fixée à l'avance qui permet de modifier l'importance de l'erreur que l'on s'autorise par rapport à la taille de la marge.

2. Cette technique est couramment appelée *soft margin.*

Nous utilisons la même méthode que précédemment : je ne vais faire que résumer les étapes. Le nouveau Lagrangien est :

$$L(w,b,\xi,\alpha,\mu) = J(w,b,\xi) + \sum_{1\le i\le l} \alpha_i \left[1 - \xi_i - y_i((w\cdot x_i)+b)\right] - \sum_{1\le i\le l} \mu_i \xi_i \tag{3.20}$$

(Où μ est un coefficient de Lagrange supplémentaire correspondant à la condition (3.18)).

- Les conditions nécessaires pour avoir un point selle de L sur $(\boldsymbol{R}^d \times \boldsymbol{R} \times \boldsymbol{R}^l) \times (\boldsymbol{R}^l_+ \times \boldsymbol{R}^l_+)$ deviennent :
  - D'une part :

$$\frac{\partial L(w^o,b^o,\xi^o,\alpha^o,\mu^o)}{\partial w} = 0 \iff w^o = \sum_{1\le i\le l} \alpha_i^o y_i x_i \tag{3.21}$$

$$\frac{\partial L(w^o,b^o,\xi^o,\alpha^o,\mu^o)}{\partial b} = 0 \iff \sum_{1\le i\le l} \alpha_i^o y_i = 0 \tag{3.22}$$

$$\frac{\partial L(w^o,b^o,\xi^o,\alpha^o,\mu^o)}{\partial \xi} = 0 \iff C - \alpha_i^o - \mu_i^o = 0 \tag{3.23}$$

  - D'après ce qui précède, et si on substitue (3.21), (3.22) et (3.23) dans (3.20), on obtient que α^o doit être **nécessairement** un maximum de...

$$\alpha \longrightarrow \sum_{1\le i\le l} \alpha_i - \frac{1}{2} \sum_{1\le i,j\le l} \alpha_i\alpha_j y_i y_j (x_i \cdot x_j) \tag{3.24}$$

    ...sous les contraintes (3.22) et...

$$\alpha_i^o \le C$$

    ... qui s'obtient à partir de (3.23) en tenant compte du fait que $\mu_i \ge 0$.

- Les conditions de KKT deviennent :

$$\begin{aligned}
w &= \sum_i \alpha_i y_i x_i && (3.25)\\
\sum_i \alpha_i y_i &= 0 && (3.26)\\
C - \alpha_i - \mu_i &= 0 && (3.27)\\
\alpha_i &\ge 0 && (3.28)\\
\mu_i &\ge 0 && (3.29)\\
\alpha_i \left[y_i((w\cdot x_i)+b) - 1 + \xi_i\right] &= 0 && (3.30)\\
\mu_i \xi_i &= 0 && (3.31)\\
y_i((w\cdot x_i)+b) - 1 + \xi_i &\ge 0 && (3.32)\\
\xi_i &\ge 0 && (3.33)
\end{aligned}$$

- Donc finalement...

  1. **Si on trouve** α^o maximum de

$$\sum_{1\le i\le l} \alpha_i - \frac{1}{2} \sum_{1\le i,j\le l} \alpha_i\alpha_j y_i y_j (x_i \cdot x_j) \tag{3.34}$$

  sous les contraintes

$$\sum_{1\le i\le l} \alpha_i^o y_i = 0$$

  et

$$0 \le \alpha_i^o \le C, \textit{ pour } 1 \le i \le l$$

2. **Si on calcule** w^o par la formule :

$$w^o = \sum_{1 \leq i \leq l} \alpha_i^o y_i x_i \tag{3.35}$$

3. **Si il existe** b^o, μ et ξ vérifiant les équations (3.27), (3.29), (3.30), (3.31), (3.32) et (3.33)

...alors (w^o, b^o) sera une solution de notre problème d'hyperplan optimal.

- Encore une fois, si l'on veut estimer la classe d'un point de $\mathcal{O}$, on utilise la formule (3.16). Vous remarquerez encore que les conditions sur b^o sont pratiquement inutilisables.

3.5 Une idée géniale

Maintenant nous avons une méthode pour trouver un hyperplan optimal séparant avec un minimum d'erreurs deux classes de points.

Mais ce qui est gênant dans cette méthode c'est le mot «hyperplan». Un hyperplan est quelque chose de mathématiquement très simple, et il serait complètement illusoire de se dire que l'on pourrait séparer efficacement des données de la vraie vie de cette manière. Par exemple si j'ai simplement des points comme la Fig. 3.4, l'hyperplan optimal tel que celui construit au paragraphe précédent va

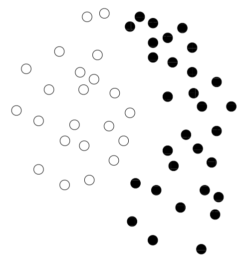

Fig. 3.4 – *Points à classer*

ressembler à ce qui est dessiné Fig. 3.5 : il y aura quelques erreurs de classification. Alors qu'une

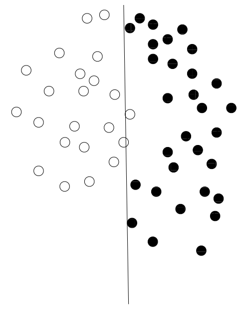

Fig. 3.5 – *Un hyperplan « optimal »*

classification comme celle de la Fig. 3.6 eût été bien plus naturelle...

La seule idée qui fait *vraiment* la force des SVM c'est de se dire « si je ne peux pas séparer linéairement mes deux classes de points dans l'espace où ils vivent, alors essayons donc de les séparer dans un espace plus gros ».

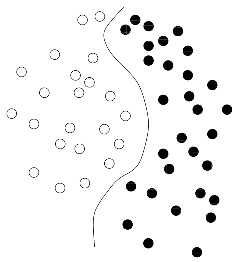

Fig. 3.6 – *Une classification idéale*

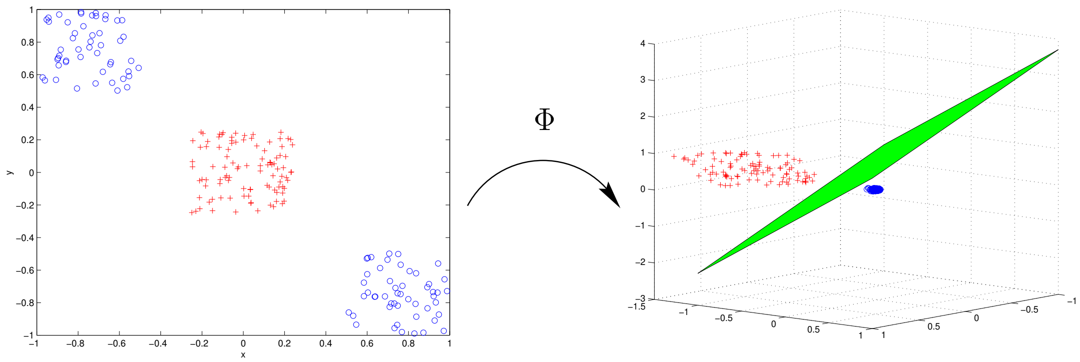

Fig. 3.7 – *Pour essayer de séparer les points, on les envoie dans un espace de dimension plus grande.*

On se munit donc d'une application
$$\Phi : \boldsymbol{R}^d \longrightarrow F$$
qui envoie les points à classer de leur espace natal ($\boldsymbol{R}^d$) dans un espace *Euclidien* F « plus gros » *i.e.* de dimension supérieure. On appelle habituellement F *feature space*.

Et tout ce qu'il nous reste à faire c'est de chercher l'hyperplan optimal de séparation dans l'espace F par la méthode précédente : on travaille simplement sur $(\Phi(x_i), y_i)_{1\leq i\leq l}$ à la place de $(x_i, y_i)_{1\leq i\leq l}$, en reprenant les formules précédentes, et en utilisant le produit scalaire de F au lieu de celui de $\boldsymbol{R}^d$.

Un exemple d'une telle séparation est donnée à la Fig. 3.7 où l'application Φ utilisée[3] est

$$\Phi : \left\{ \begin{array}{ccc} \boldsymbol{R}^2 & \longrightarrow & \boldsymbol{R}^3 \\ x = (x(1), x(2)) & \longmapsto & \begin{pmatrix} x(1)^2 \\ \sqrt{2}\, x(1)\, x(2) \\ x(2)^2 \end{pmatrix} \end{array} \right. \tag{3.36}$$

Là où le bât blesse, c'est que si l'espace F est de dimension très grande, le produit scalaire sera techniquement impossible à calculer *numériquement*. Il nous faut donc éviter de manipuler des objets dans F. Nous pouvons nous rendre compte que nous avons uniquement besoin de savoir calculer des *produits scalaires dans* F : plutôt que d'envoyer des points x et x' de $\boldsymbol{R}^d$ dans F puis de calculer $(\Phi(x) \cdot \Phi(x'))$, il serait intéressant pour nous qu'il existe une fonction peu coûteuse en calcul

$$k : \left\{ \begin{array}{ccc} \boldsymbol{R}^d \times \boldsymbol{R}^d & \longrightarrow & \boldsymbol{R} \\ (x, x') & \longmapsto & \Phi(x) \cdot \Phi(x') \end{array} \right. \tag{3.37}$$

3. Évidemment on ne peut pas prendre n'importe quelle application Φ pour séparer les points ! (Par exemple une application constante n'a pas lieu d'être) Par ailleurs, pour un problème donné, on ne connait aucune méthode générale pour choisir une telle application.

qui nous permettrait de calculer le produit scalaire sans passer par F.

En fait, plutôt que de choisir une application Φ, on préfère choisir une application $k : \boldsymbol{R}^d \times \boldsymbol{R}^d \longrightarrow \boldsymbol{R}$ telle qu'il existe un espace Euclidien F et une application $\Phi : \boldsymbol{R}^d \to F$ vérifiant la décomposition [4] (3.37). Une telle application k s'appelle ***kernel*** ou « noyau » en français.

Une fois l'application k choisie, on ne fait les calculs de produits scalaires qu'avec elle, escamotant ainsi *complètement* l'existence de F. En effet le problème (3.34) revient à chercher dans un premier temps α^o maximum de

$$\sum_{1\le i\le l} \alpha_i - \frac{1}{2} \sum_{1\le i,j\le l} \alpha_i\alpha_j y_i y_j \underbrace{k(x_i \cdot x_j)}_{\Phi(x_i)\cdot\Phi(x_j)}$$

sous les contraintes

$$\sum_{1\le i\le l} \alpha_i^o y_i = 0$$

et

$$0 \le \alpha_i^o \le C, \textit{ pour } 1 \le i \le l$$

puis de vérifier que l'on peut trouver b^o vérifiant les équations (3.27), (3.29), (3.30), (3.31), (3.32) et (3.33) pour $i = 1 \ldots l$, où (3.30) et (3.32) deviennent [5] :

$$\alpha_i[\sum_j y_i y_j k(x_i, x_j) + y_i\, b - 1 + \xi_i] = 0$$

$$\sum_j y_i y_j k(x_i, x_j) + y_i\, b - 1 + \xi_i \ge 0$$

Pour estimer la classe d'un point x il nous reste à utiliser [6]

$$f(x) = \text{sign}\left[\sum_{support\ vectors} \alpha_i^o y_i k(x_i, x) + b^o\right]$$

3.6 Construction de *kernels*

De la même façon qu'il n'existe pas de méthode pour choisir Φ, il n'existe pas de méthode pour choisir le *kernel* k. Par contre, il existe un théorème pour les caractériser comme le fait remarquer Vapnik dans [29] :

Théorème 3.1 (Mercer) *Pour qu'une fonction continue symétrique*

$$k : C \times C \longrightarrow \boldsymbol{R}$$

4. Vous noterez qu'aucune unicité de la décomposition n'est requise. Par exemple pour $k(x, x') = (x \cdot x')^2$, on peut prendre $F = \boldsymbol{R}^3$ et Φ comme au (3.36), aussi bien que $F = \boldsymbol{R}^4$ et

$$\Phi : x = (x(1), x(2)) \longmapsto \begin{pmatrix} x(1)^2 \\ x(1)\,x(2) \\ x(1)\,x(2) \\ x(2)^2 \end{pmatrix}$$

5. En remplaçant l'expression de w de (3.25) dans ces deux équations.
6. Cette fonction est obtenue à partir de (3.16).

(où C est un compact de $\boldsymbol{R}^d$) soit décomposable sous la forme :

$$k(x,x') = \sum_{i=1}^{\infty} a_i\, \psi_i(x)\psi_i(x')$$

(où les a_i sont des réels strictement positifs et $\psi_i \in \boldsymbol{L}^2(C)$), il est nécessaire et suffisant que la condition

$$\int_C \int_C k(x,x')g(x)g(x')\,dx\,dx' > 0$$

soit vérifiée pour toute application $g \in \boldsymbol{L}^2(C)$.

Il est très facile de voir [7] par des arguments d'approximation de fonctions que les conditions de Mercer sont équivalentes au fait que pour tout ensemble fini $\{x_1, \ldots, x_p\} \subset C$ la matrice K de coefficients $K_{ij} = k(x_i, x_j)$ est positive.

Avec cette remarque en tête, on en déduit par des arguments tout aussi simples :

Proposition 3.1 *Soient*[8] k_1 *et* k_2 *des* kernels *sur $C \times C$, où C est un compact de $\boldsymbol{R}^d$. Soient aussi $a \in \boldsymbol{R}^+$, h une fonction réelle sur C, et une fonction*

$$\Phi : C \to \boldsymbol{R}^m$$

correspondant à un kernel k_3 *sur $\boldsymbol{R}^m \times \boldsymbol{R}^m$ (où $m \in \boldsymbol{N}^\star$). Soit enfin B une matrice positive de $\boldsymbol{R}^{d\times d}$. Alors les fonctions suivantes sont des* kernels *sur $C \times C$:*

- $k(x,x') = k_1(x,x') + k_2(x,x')$
- $k(x,x') = a\, k_1(x,x')$
- $k(x,x') = k_1(x,x')\, k_2(x,x')$
- $k(x,x') = h(x)\, h(x')$
- $k(x,x') = k_3(\Phi(x), \Phi(x'))$
- $k(x,x') = x'\, B\, x$

On en déduit aussitôt :

Proposition 3.2 *Soit k_1 un* kernel *continu sur $C \times C$ où C est un compact de $\boldsymbol{R}^d$. Soit P une fonction polynôme avec des coefficients positifs. Alors les fonctions suivantes sont des* kernels *:*

- $k(x,x') = P(k_1(x,x'))$
- $k(x,x') = \exp(k_1(x,x'))$
- $k(x,x') = \exp(-\|x-x'\|^2/\sigma^2)$ *avec* $\sigma \in R_+^\star$

Le dernier de ces *kernels* est le *kernel gaussien*, très utilisé en pratique. On utilise aussi très souvent les *kernels* polynomiaux de la forme

$$k : (x,x') \mapsto (u\, x \cdot x' + v)^p$$

où $u \in \boldsymbol{R}$, $v \in \boldsymbol{R}$, $p \in \boldsymbol{N}^\star$.

7. Dans la suite de ce paragraphe, consultez [8] pour les détails.
8. Toutes les fonctions de cette proposition sont supposées continues pour pouvoir appliquer le théorème de Mercer.

3.7 Problèmes existentiels

Par construction, il est clair que notre problème de recherche d'hyperplan a une solution. Mais lors de notre résolution, nous avons vu que des petits problèmes apparaissent.

En particulier, on peut se demander si le problème de maximisation (3.34) a une solution, et si à chaque solution α^o il correspond un point selle.

Pour l'existence, c'est évident : étant donné que notre problème général a au moins une solution (w, b) et que d'après le Théo. A.2 il existe alors un vecteur α tel que (w, b, α) soit un point selle du Lagrangien (3.20) et que de plus tout point selle doit être solution du problème de maximisation (3.34), il n'y a aucun problème.

Maintenant, peut-on affirmer qu'à toute solution α de (3.34) on peut associer une solution (w, b) du problème général ? C'est quelque chose qui m'a franchement énervé, car tout le monde affirme «oui», ou plutôt tout le monde passe à coté du problème en disant « une fois que l'on a une solution de (3.34), on calcule b en utilisant (3.30), en remarquant que $\xi_i = 0$ si $\alpha_i < C$ ». Et je n'ai trouvé *aucun* article ni *personne* pour me dire pourquoi on peut faire cela.

J'ai fini par résoudre le problème par la méthode qui suit. Réécrivons (3.34) comme un problème de *minimisation* de

$$\alpha \longrightarrow \frac{1}{2}\alpha^{\mathrm{T}} Q\alpha - \alpha^{\mathrm{T}}\mathbf{1} \tag{3.38}$$

(Dans le cas général où l'on utilise un kernel, où la matrice Q a pour coefficients $Q_{ij} = y_i y_j k(x_i, x_j)$) Sous les contraintes :

$$\sum_{1\leq i\leq l} \alpha_i y_i = 0$$

et

$$0 \leq \alpha_i \leq C$$

Par le Théo. A.1, on sait que les conditions de KKT (de ce sous-problème particulier) sont **nécessaires**[9] à l'existence d'une solution de ce problème.

Autrement dit, si α^o est solution du problème, il existe des variables $\lambda^{eq} \in \boldsymbol{R}$, $\lambda^{low} \in \boldsymbol{R}^l$, $\lambda^{up} \in \boldsymbol{R}^l$ telles que :

$$Q\alpha^o - \mathbf{1} + \lambda^{eq}\, y - \lambda^{low} + \lambda^{up} = 0 \tag{3.39}$$

$$\lambda^{up} \geq 0$$

$$\lambda^{low} \geq 0$$

$$\lambda_i^{up}\,(\alpha_i^o - C) = 0$$

$$\lambda_i^{low}\alpha_i^o = 0$$

Prenons maintenant $b = \lambda^{eq}$, $\mu_i = C - \alpha_i$ et $\xi_i = 0$ si $0 \leq \alpha_i^o < C$, $\xi_i = \lambda_i^{up}$ si $\alpha_i^o = C$.

Avec ces valeurs, les conditions de KKT (3.25)-(3.33) du problème général sont vérifiées, autrement dit (w, λ^{eq}) (où w est défini par (3.25)) est solution du problème général.

9. En fait, vu que k doit vérifier les conditions de Mercer, on peut facilement voir que Q est positive, donc (3.38) est convexe. Les conditions de KKT sont donc aussi suffisantes.

Nous pouvons donc affirmer maintenant que la méthode suivante, qui est ce qu'on appelle un « *Support Vector Machine*[10] » fonctionne. Pour simplifier les notations, on pose encore Q la matrice de coefficients $Q_{ij} = y_i y_j k(x_i, x_j)$.

On cherche un α^o minimum de

$$\frac{1}{2}\alpha^{\mathrm{T}} Q \alpha - \alpha^{\mathrm{T}} \mathbf{1} \tag{3.40}$$

sous les contraintes

$$y^{\mathrm{T}} \alpha^o = 0 \tag{3.41}$$

et

$$0 \leq \alpha_i^o \leq C, \textit{ pour } 1 \leq i \leq l \tag{3.42}$$

On calcule b^o en utilisant une des égalités[†]...

$$(Q\alpha^o)_i + y_i b^o - 1 = 0 \tag{3.43}$$

... pour i tel que[††] $0 < \alpha_i < C$.

Pour estimer la classe d'un point $x \in \mathcal{O}$, on **calcule**

$$f(x) = \text{sign}\left[\sum_{support\ vectors} \alpha_i^o y_i k(x_i, x) + b^o\right] \tag{3.44}$$

[†] En remarquant avec les conditions de KKT que si $\alpha_i < C$ alors $\xi_i = 0$

[††] Le cas pathologique très rare où tous les α_i sont à 0 ou à C ne sera pas traité dans ce rapport. En fait il est facile de voir qu'il amène à une infinité de solutions [3].

3.8 Un petit exemple : le XOR

Maintenant que les principales idées ont été données, il nous reste des détails d'ordre pratique. En particulier, il nous faut résoudre le problème de minimisation (3.40). À ce sujet, un laboratoire de *France Telecom* m'a d'ailleurs demandé un petit exemple qui puisse se résoudre « à la main » : j'ai été très ennuyé pour inventer un problème facile. J'ai fini par leur fabriquer une solution au traditionnel exemple du XOR, pour leur montrer que rien n'est très simple, même avec très peu de points...

Il s'agit de séparer quatre points dans $\boldsymbol{R}^2$ comme représentés FIG. 3.8. Prenons par exemple $x_1 = (0{,}0)^{\mathrm{T}}$, $x_2 = (1{,}1)^{\mathrm{T}}$, $x_3 = (1{,}0)^{\mathrm{T}}$, $x_4 = (0{,}1)^{\mathrm{T}}$ et $y_1 = y_2 = -1$ ainsi que $y_3 = y_4 = 1$. Le problème n'étant pas linéairement séparable, il nous faut introduire un *kernel* adéquat. Prenons simplement $k : (x{,}y) \longmapsto (x \cdot y)^2$. La matrice du problème (3.40) devient alors :

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}$$

10. On note souvent «SVM». Il est difficile de traduire cette expression en français. Vous pouvez à la limite dire « Machine à points de support », ou à l'extrême rigueur, « Modèle à points de support »

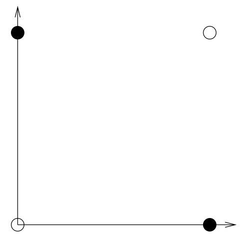

FIG. 3.8 – *Le problème du XOR*

Et après un rapide calcul, (3.40) devient (à un facteur $\frac{1}{2}$ près) :

$$4\alpha_2^2 + \alpha_3^2 + \alpha_4^2 - 2\alpha_2\alpha_3 - 2\alpha_2\alpha_4 - 2(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) \tag{3.45}$$

Nous devons donc minimiser (3.45) sous les contraintes (3.41) et (3.42). Pour simplifier les calculs, nous allons supposer que α_3 et α_4 sont égaux à l'optimum [11]. Posons donc

$$\beta = \alpha_3 = \alpha_4$$

et en remplaçant dans (3.45) et (3.41) cherchons à minimiser

$$4\alpha_2^2 + 2\beta^2 - 4\alpha_2\beta - 2\alpha_1 - 2\alpha_2 - 4\beta \tag{3.46}$$

sous les contraintes (3.42) et

$$\alpha_1 = -\alpha_2 + 2\beta \tag{3.47}$$

Maintenant, on voudrait minimiser (3.46) le long de (3.47), ce qui donne en insérant l'expression (3.47) de α_1 dans (3.46) :

$$h(\alpha_2, \beta) = 4\alpha_2^2 + 2\beta^2 - 4\alpha_2\beta - 8\beta$$

Essayons de chercher un minimum sans nous préoccuper des contraintes. Les conditions nécessaires sont alors :

$$\frac{\partial h(\alpha_2, \beta)}{\partial \alpha_2} = 0 \Longleftrightarrow 2\alpha_2 - \beta = 0$$

et

$$\frac{\partial h(\alpha_2, \beta)}{\partial \beta} = 0 \Longleftrightarrow \beta - \alpha_2 - 2 = 0$$

La solution de ce système est $\alpha_2 = 2$ et $\beta = 4$.

En utilisant (3.43) pour $i = 1$ par exemple, on en déduit immédiatement $b = -1$.

Les α_i et b trouvés vérifient (par hasard ?!) les conditions de KKT (3.25)-(3.33). Une solution de notre problème est donc $\alpha_1 = 6$, $\alpha_2 = 2$, $\alpha_3 = \alpha_4 = 4$.

Essayons maintenant de tracer la surface de décision $f(x, y) \geq 0$ dans $\boldsymbol{R}^2$, où f dans (3.44) vaut ici :

$$f(x, y) = \text{sign}\left[2x^2 + 2y^2 - 4xy - 1\right] = \text{sign}\left[2(x - y)^2 - 1\right]$$

La surface de décision est donc délimitée par les droites d'équation $y = x \pm \frac{1}{2}$. Son allure est donnée FIG. 3.9.

11. Si cette supposition vous choque, vous pouvez résoudre sans la faire.

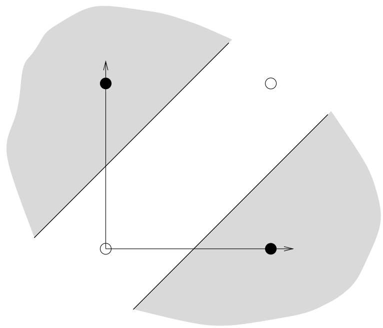

FIG. 3.9 – *La surface de décision*

Au lieu d'utiliser un noyau polynomial, nous aurions pu utiliser un noyau gaussien. Avec un noyau k de la forme

$$k(x, y) = e^{-\frac{(x-y)^2}{\sigma^2}}$$

la matrice Q du problème (3.40) devient (si on pose $a = e^{-\frac{1}{\sigma^2}}$)

$$Q = \begin{pmatrix} 1 & a^2 & -a & -a \\ a^2 & 1 & -a & -a \\ -a & -a & 1 & a^2 \\ -a & -a & a^2 & 1 \end{pmatrix}$$

Les calculs se compliquent fortement... sauf si l'on intuite que $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \beta$ à l'optimum. On cherche alors à maximiser (3.40) qui devient (à un coefficient multiplicateur 2 près) :

$$\beta \longmapsto \beta^2 \left[a^2 - 2a + 1\right] - 2\beta$$

Si l'on essaye de minimiser cette fonction sans se préoccuper des contraintes, il vient immédiatement

$$\beta = \frac{1}{(1-a)^2}$$

Avec cette valeur de β, les coefficients α_i vérifient les conditions de KKT (en prenant $b = 0$ obtenu avec (3.43)).

Nous avons donc trouvé une solution au problème. Nous pouvons encore tracer la surface de décision $f(x, y) \geq 0$ dans $\boldsymbol{R}^2$ avec (d'après (3.44)) :

$$f(x, y) = \text{sign}\left[\beta e^{-\frac{x^2+y^2}{\sigma^2}} \left(-1 - e^{-2\frac{1-x-y}{\sigma^2}} + e^{-\frac{1-2x}{\sigma^2}} + e^{-\frac{1-2y}{\sigma^2}}\right)\right]$$

Je vous épargne la vue des calculs :

On a donc...

$$f(x,y) \geq 0 \iff -1 - a^2XY + aX + aY \geq 0$$

... en posant $X = e^{-\frac{2x}{\sigma^2}}$ et $Y = e^{-\frac{y}{\sigma^2}}$. Il vient alors :

$$f(x,y) \geq 0 \iff aY\,[1 - aX] \geq 1 - aX$$

Donc $f(x,y) \geq 0$ *si et seulement si*

$$\begin{array}{lcl} Y \geq 1/a & et & X \leq 1/a \\ Y \leq 1/a & et & X \geq 1/a \end{array}$$

Ce qui revient à dire :

$$\begin{array}{lcl} y \leq 1/2 & et & x \geq 1/2 \\ y \geq 1/2 & et & x \leq 1/2 \end{array}$$

La surface de décision $f(x,y) \geq 0$ est donc telle que dessinée FIG. 3.10.

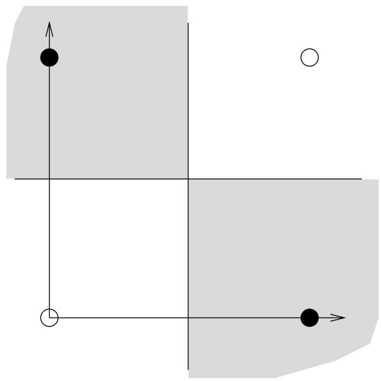

FIG. 3.10 – *Surface de décision avec* kernel *gaussien*

Pour conclure vous remarquerez :

- Que la surface de décision dépend complètement du *kernel* choisi.
- Que les calculs deviennent vite compliqués : une méthode numérique de recherche d'extrema *sous contraintes* s'impose.
- Que dans le cas étudié, nous avons eu de la chance de ne pas avoir à nous occuper des contraintes : il est clair qu'une grande partie de la difficulté de résolution viendra de là.

CHAPITRE 4 Support Vector Regression

Il est possible d'étendre facilement l'idée des SVM à la régression, comme le suggère Vapnik dans [28, 29].

Rappelons d'abord le problème «standard» de la régression linéaire : étant donné un ensemble de valeurs $(x_i,y_i)_{1\leq i\leq l} \in \boldsymbol{R}^d \times \boldsymbol{R}$, on cherche a trouver la fonction

$$f(x) = (w \cdot x) + b$$

qui minimise l'erreur

$$\sum_{i=1}^{l} \zeta(y_i - f(x_i)) \tag{4.1}$$

où ζ est une fonction de coût.

Dans le cas des *support vector machines*, on introduit la fonction de coût «ϵ-insensible» de Vapnik (avec $\epsilon > 0$) :

$$|x|_\epsilon = \max\{0,|x| - \epsilon\}$$

La caractéristique intéressante de celle-ci est qu'elle permet « d'ignorer » les erreurs inférieures à ϵ.

Intuitivement, en prenant $\zeta \equiv |.|_\epsilon$ dans (4.1), notre problème revient à chercher f telle que les données $(x_i,y_i)_{1\leq i\leq l}$ passent «au mieux» dans un tube d'erreur dont les bords sont définis par les hyperplans d'équations $x \mapsto f(x) - \epsilon$ et $x \mapsto f(x) + \epsilon$.

Finalement, plutôt que de chercher à résoudre le problème (4.1), nous allons minimiser

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} |y_i - f(x_i)|_\epsilon$$

où $\frac{1}{2}\|w\|^2$ est un terme de régularisation[1], et C est une constante fixée à l'avance qui permet de modifier l'importance de l'erreur par rapport au terme de régularisation.

1. Il est fréquent d'introduire ce genre de termes dans les algorithmes d'apprentissage, pour contrôler la capacité en induisant une préférence sur les solutions. Ici on l'introduit surtout pour pouvoir se ramener à un problème quadratique similaire au cas de la classification.

Cependant, si l'on se place dans le cadre de la minimisation des moindres carrés (introduite indépendemment par Gauss et Legendre au 18[ième] siècle), c'est-à-dire si l'on s'interesse à la recherche de (w,b) minimisant

$$L(w,b) = \sum_{i=1}^{l}(y_i - (w \cdot x_i) - b)^2 \tag{4.2}$$

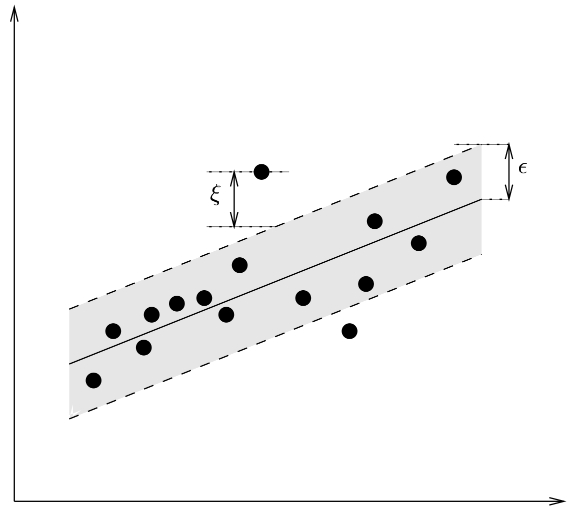

FIG. 4.1 – *On cherche un hyperplan tel que les données passent « au mieux » dans un « tube d'erreur » de largeur ϵ autour de cet hyperplan.*

Vu comme un problème d'optimisation sous contraintes, cela revient à minimiser

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^\star)$$

sous les contraintes (pour $i = 1 \ldots l$)

$$\begin{array}{lcl} ((w \cdot x_i) + b) - y_i & \leq & \epsilon + \xi_i \\ y_i - ((w \cdot x_i) + b) & \leq & \epsilon + \xi_i^\star \\ \xi_i,\, \xi_i^\star \geq 0 & & \end{array}$$

Encore une fois, *exactement* comme dans le cas de la classification, ce problème est résolu en introduisant un lagrangien, puis en utilisant des techniques de point selle.

---

alors en posant

$$\hat{w} = \begin{pmatrix} w \\ b \end{pmatrix}$$

et

$$\hat{X} = \begin{pmatrix} x_1^{\mathrm{T}} & 1 \\ & \vdots \\ x_l^{\mathrm{T}} & 1 \end{pmatrix}$$

il est facile de voir (voyez par exemple [8] pour les détails) que si $\hat{X}^{\mathrm{T}}\hat{X}$ est inversible, alors la solution de (4.2) est

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

Si $\hat{X}^{\mathrm{T}}\hat{X}$ n'est pas inversible Hoerl et Kennard [13] proposent (dans une méthode appelée *ridge regression*) de prendre plutôt

$$\hat{w} = (\hat{X}^{\mathrm{T}}\hat{X} + \lambda I_l)^{-1}\hat{X}^{\mathrm{T}}y$$

où $\lambda \in \boldsymbol{R}^+$ et I_l est la matrice identité de taille l. Et on peut encore facilement voir que ce $\hat{w}$ est solution du problème de minimisation

$$\tilde{L}(w,b) = \lambda(w \cdot w) + \sum_{i=1}^{l}(y_i - (w \cdot x_i) - b)^2$$

On ajoute ici un terme de régularisation similaire à celui qu'on introduit dans les *support vector machines* (qui ne tombe donc pas vraiment du ciel).

Je ne vais qu'écrire les conditions de KKT du problème : $(w, b, \xi, \xi^\star)$ est solution du problème de minimisation sous contraintes *si et seulement si* il existe des vecteurs $\alpha \in \boldsymbol{R}^d$, $\alpha^\star \in \boldsymbol{R}^d$, $\eta \in \boldsymbol{R}^d$, $\eta^\star \in \boldsymbol{R}^d$ tel que l'on ait (pour tout $i = 1 \dots l$)

$$\begin{array}{rcl}
w & = & \sum_i (\alpha_i^\star - \alpha_i) x_i \\
C - \alpha_i - \eta_i & = & 0 \\
C - \alpha_i^\star - \eta_i^\star & = & 0 \\
\alpha_i \left[(w \cdot x_i) + b - y_i - \epsilon - \xi_i \right] & = & 0 \\
\alpha_i^\star \left[(w \cdot x_i) + b - y_i + \epsilon + \xi_i^\star \right] & = & 0 \\
\eta_i \xi_i & = & 0 \\
\eta_i^\star \xi_i^\star & = & 0 \\
((w \cdot x_i) + b) - y_i & \leq & \epsilon + \xi_i \\
y_i - ((w \cdot x_i) + b) & \leq & \epsilon + \xi_i^\star
\end{array}$$

De ces conditions nous pouvons déduire facilement plusieurs remarques intéressantes. D'abord que :

- Si $\alpha_i = 0$ alors $(x_i, f(x_i))$ est « au dessus[2] » de l'hyperplan définit par l'équation $x \mapsto f(x) - \epsilon$.
- Si $\alpha_i = C$ alors $(x_i, f(x_i))$ est « au dessous » de l'hyperplan définit par l'équation $x \mapsto f(x) - \epsilon$.
- Si $0 < \alpha_i < C$ alors $(x_i, f(x_i))$ est *sur* l'hyperplan définit par l'équation $x \mapsto f(x) - \epsilon$.
- Si $\alpha_i^\star = 0$ alors $(x_i, f(x_i))$ est « au dessous » de l'hyperplan définit par l'équation $x \mapsto f(x) + \epsilon$.
- Si $\alpha_i^\star = C$ alors $(x_i, f(x_i))$ est « au dessus » de l'hyperplan définit par l'équation $x \mapsto f(x) + \epsilon$.
- Si $0 < \alpha_i^\star < C$ alors $(x_i, f(x_i))$ est *sur* l'hyperplan définit par l'équation $x \mapsto f(x) + \epsilon$.
- $\alpha_i \, \alpha_i^\star = 0$.

Et réciproquement :

- Si $(x_i, f(x_i))$ est strictement au dessus de l'hyperplan définit par l'équation $x \mapsto f(x) - \epsilon$, alors $\alpha_i = 0$.
- Si $(x_i, f(x_i))$ est strictement au dessous de l'hyperplan définit par l'équation $x \mapsto f(x) + \epsilon$, alors $\alpha_i^\star = 0$.

De ces deux dernière remarques, vous noterez que si le point $(x_i, f(x_i))$ est dans le tube d'erreur (*i.e* si $|f(x_i) - y_i| < \epsilon$) alors $\alpha_i = \alpha_i^\star = 0$. Ainsi, de la même façon qu'en classification, nous n'avons pas besoin de tous les x_i pour décrire w, et nous appellerons « *support vector* » tout point x_i qui a un de ses coefficients α_i ou $\alpha_i^\star$ non nul.

La généralisation à la régression non-linéaire est facilement réalisée en introduisant une fonction *kernel*, en complète analogie avec le cas de la classification.

Étant donné que tous les calculs, bien que légèrement plus fastidieux, suivent exactement ceux de la classification, je ne donnerais ici que le résultat final. Voici donc le shéma de fonctionnement d'un « Support Vector Machine en régression[3] », qui utilise un *kernel* noté k. Pour simplifier les notations, on note K la matrice de coefficient $K_{ij} = k(x_i, x_j)$.

2. Au sens large. (le point peut être aussi *sur* l'hyperplan).
3. Encore appelé « Support Vector Regression ».

On cherche un couple $(\alpha^o, \alpha^{\star,\,o})$ minimum de

$$(\alpha, \alpha^\star) \longmapsto \frac{1}{2}(\alpha^\star - \alpha)^{\mathrm{T}} K (\alpha^\star - \alpha) - (\alpha^\star - \alpha)^{\mathrm{T}} y + \epsilon (\alpha^\star + \alpha)^{\mathrm{T}} \mathbf{1} \tag{4.3}$$

sous les contraintes

$$(\alpha - \alpha^\star)^{\mathrm{T}} \mathbf{1} = 0$$

et

$$0 \leq \alpha_i, \alpha_i^\star \leq C, \;\; pour \;\; 1 \leq i \leq l$$

On calcule b^o en utilisant une des égalités † ...

$$\begin{aligned} b^o &= (K(\alpha^o - \alpha^{\star,\,o}))_i + y_i + \epsilon \quad pour \quad i \; tel \; que \; 0 < \alpha_i^o < C \\ b^o &= (K(\alpha^o - \alpha^{\star,\,o}))_i + y_i - \epsilon \quad pour \quad i \; tel \; que \; 0 < \alpha_i^{\star,\,o} < C \end{aligned}$$

Pour estimer la fonction recherchée en un point x, on **calcule**

$$f(x) = \sum_{support \;\; vectors} (\alpha_i^{\star,\,o} - \alpha_i^o) k(x_i, x) + b$$

† Le cas pathologique très rare où tous les α_i^o et $\alpha_i^{\star,\,o}$ sont à 0 ou C ne sera pas traité dans ce rapport. En fait il est facile de voir qu'il amène à une infinité de solutions [3].

CHAPITRE 5 *SVMTorch*

Il nous reste maintenant un problème capital à résoudre pour faire fonctionner les SVM : le problème quadratique (3.40) pour la classification et le problème (plus compliqué en pratique) quadratique (4.3) pour la régression.

Il existe en fait dans la littérature mathématique traditionnelle plusieurs méthodes (décrites par exemple dans [5] ou bien [11]) pour résoudre ce genre de problème : citons par exemple la méthode de Newton, celle du gradient conjugué (ces deux méthodes doivent être adaptées pour prendre en compte les contraintes du problème) ou bien celle du point intérieur, qui est un peu plus évoluée. Toutes ces méthodes ont un point commun déplorable : elles sont *extrêmement* lentes. Il est inimagineable avec les ordinateurs actuels de résoudre un problème de SVM avec seulement 10000 exemples d'apprentissage en utilisant ce genre de méthodes, principalement à cause de la complexité du problème qui est en $\mathcal{O}(l^2)$ (où l est la taille de l'ensemble d'apprentissage).

Heureusement plusieurs méthodes rapides (bien que de même complexité) ont été mises au point dans la communauté de *l'apprentissage statistique*, toutes basées sur une méthode dite de *décomposition*. Il s'agit de résoudre le problème quadratique par étapes : on fixe la plupart des inconnues du problème à leur valeurs courantes, et on résoud le problème de minimisation par rapport aux autres inconnues. Puis on recommence.

Parmi les algorithmes les plus connus basés sur cette idée citons *SMO* de Platt [25], *SVM-Light* de Joachims [14] (le plus rapide jusqu'à il y a quelques mois en classification) et *Nodelib* de Flake et Lawrence [10] (le plus rapide jusqu'à il y a quelques mois en régression). Comme je trouvais que l'algorithme de Flake et Lawrence était vraiment trop lent en régression, j'en ai élaboré un nouveau en m'inspirant de certaines idées de Joachims. J'ai alors créé un logiciel [1] de SVM baptisé *SVMTorch* avec cet algorithme pour la régression et l'algorithme de Joachims pour la classification.

SVMTorch est actuellement le programme le plus rapide du monde (à ma connaissance) aussi bien en classification qu'en régression. Pour la classification il est plus rapide (d'un facteur 2) que *SVM-Light* sans doute pour des questions d'implémentation (car l'idée mathématique est la même que celle de Joachims). Pour la régression mon algorithme fait toute la différence et j'ai observé jusqu'à un facteur 100 par rapport à l'algorithme de Flake et Lawrence, sur certaines données.

Comme *SVMTorch* est un programme extrêmement complexe (dans la version actuelle il y a plus de 7000 lignes de code) il est hors de question que je vous donne le listing ici. Par contre je vais vous livrer dans les deux chapitres suivant les deux publications [2] que j'ai réalisé avec Samy Bengio sur l'algorithme de régression.

La première publication traite de l'algorithme en lui-même, et la seconde traite de sa convergence. Notez que *SVMTorch* est actuellement le seul algorithme basé sur une méthode de décomposition

1. Disponible sur `http://www.idiap.ch/learning/SVMTorch.html`
2. Soumises au *Journal of Machine Learning Research* (`http://www.ai.mit.edu/projects/jmlr`).

pour lequel il existe une preuve de convergence en régression.

CHAPITRE 6

Support Vector Machines for Large-Scale Regression Problems

Authors: Ronan Collobert and Samy Bengio
Submitted to Journal of Machine Learning Research
Available on ***ftp://ftp.idiap.ch/pub/reports/2000/rr00-17.ps.gz***

Abstract. Support Vector Machines (SVMs) for regression problems are trained by solving a quadratic optimization problem which needs on the order of l^2 memory and time resources to solve, where l is the number of training examples. In this paper, we propose a decomposition algorithm, *SVMTorch* [1], which is similar to *SVM-Light* proposed by Joachims [14] for classification problems, but adapted to regression problems. With this algorithm, one can now efficiently solve large-scale regression problems (more than 20000 examples). Comparisons with *Nodelib*, another SVM algorithm for large-scale regression problems from Flake and Lawrence [10] yielded significant time improvements.

6.1 Introduction

Vapnik has proposed in [28] a method to solve regression problems using Support Vector Machines. It has yielded excellent performances on many regression and time series prediction problems (see for instance [23, 9]). This paper proposes an efficient implementation of SVMs for large-scale regression problems. Let us first recall how it works. Given a training set of l *examples* $(\boldsymbol{x}_i, y_i)$ with $\boldsymbol{x}_i \in E$ and $y_i \in \boldsymbol{R}$, where E is an Euclidean space with a scalar product denoted $(\,\cdot\,)$, we want to estimate the following linear regression:

$$f(\boldsymbol{x}) = (\boldsymbol{w} \cdot \boldsymbol{x}) + b$$

(with $b \in \boldsymbol{R}$) with a precision ϵ. For this, we minimize

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l} |y_i - f(\boldsymbol{x}_i)|_\epsilon$$

where $\frac{1}{2}\|\boldsymbol{w}\|^2$ is a regularization factor, C is a fixed constant, and $|.|_\epsilon$ is the ϵ-insensitive loss function defined by Vapnik:

$$|z|_\epsilon = \max\{0, |z| - \epsilon\}.$$

1. *SVMTorch* is available at `http://www.idiap.ch/learning/SVMTorch.html`.

Written as a constrained optimization problem, it amounts to minimizing

$$\tau(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^\star) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^\star)$$

subject to

$$((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) - y_i \leq \epsilon + \xi_i \tag{6.1}$$

$$y_i - ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) \leq \epsilon + \xi_i^\star \tag{6.2}$$

$$\xi_i, \xi_i^\star \geq 0.$$

To generalize to non-linear regression, we replace the dot product with a kernel $k(\cdot)$. Then, introducing Lagrange multipliers $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^\star$, the optimization problem can be stated as:

Minimize the function

$$\mathcal{W}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) = \frac{1}{2}(\boldsymbol{\alpha}^\star - \boldsymbol{\alpha})^{\mathrm{T}} K(\boldsymbol{\alpha}^\star - \boldsymbol{\alpha}) - (\boldsymbol{\alpha}^\star - \boldsymbol{\alpha})^{\mathrm{T}}\boldsymbol{y} + \epsilon(\boldsymbol{\alpha}^\star + \boldsymbol{\alpha})^{\mathrm{T}}\,\mathbf{1} \tag{6.3}$$

subject to

$$(\boldsymbol{\alpha} - \boldsymbol{\alpha}^\star)^{\mathrm{T}}\,\mathbf{1} = 0 \tag{6.4}$$

and

$$0 \leq \alpha_i^\star,\, \alpha_i \leq C, \quad i = 1...l \tag{6.5}$$

where K is the matrix with coefficients $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The estimate of the regression function at a given point is then

$$f(\boldsymbol{x}) = \sum_{i=1}^{l}(\alpha_i^\star - \alpha_i)k(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

where b is computed using the fact that (6.1) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$ and (6.2) becomes an equality with $\xi_i^\star = 0$ if $0 < \alpha_i^\star < C$.

Solving the minimization problem (6.3) under the constraints (6.4) and (6.5) needs resources on the order of l^2 and is thus difficult for problems with large l.

In this paper, we propose a method to solve such problems efficiently using a decomposition algorithm similar to the one proposed by Joachims [14] in the context of classification problems. In the next section, we give the general algorithm and explain in more details each if its main steps, as well as a discussion on some important implementation issues, such as a way to efficiently handle the kernel matrix computation. In the experiment section, we first compare this new algorithm to *Nodelib* [10], another SVM algorithm for large-scale regression problems, and then show how the size of the internal memory allocated to the resolution of the problem is related to the time needed to solve it.

6.2 The Decomposition Algorithm

As in the classification algorithm proposed by Joachims [14], which was based on a idea from Osuna *et al* [24], our regression algorithm is subdivided into the following four steps:

1. Select q variables as the new working set, called $\mathcal{S}$.

2. Fix the other variables $\mathcal{F}$ to their current values and solve the problem (6.3) with respect to $\mathcal{S}$.
3. Search for variables that are stuck to 0 or C and that will probably not change anymore. This is the *shrinking* phase.
4. Test if the optimization is finished or if we go back to the first step.

6.2.1 Selection of a New Working Set

We propose to select a new set of variables such that the overall criterion will be the most optimized. In order to select such working set, we use the same idea as Joachims [14]: simply search for the optimal gradient descent direction $\boldsymbol{p}$ which is *feasible* and which has only q non-null components. The variables corresponding to these components will be our new working set $\mathcal{S}$.

We thus need to minimize:

$$\mathcal{V}(\boldsymbol{p}) = \left(\mathcal{W}^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{\star})\right)^{\mathrm{T}} \boldsymbol{p} \tag{6.6}$$

with

$$\boldsymbol{p} = (d_1 \dots d_l, d_1^{\star} \dots d_l^{\star})^{\mathrm{T}}$$

subject to:

$$\mathbf{1}^{\mathrm{T}} \boldsymbol{d} - \mathbf{1}^{\mathrm{T}} \boldsymbol{d}^{\star} = 0 \tag{6.7}$$

$$\begin{array}{rll} d_i \geq 0 & \text{for} & i \text{ such that } \alpha_i = 0 \\ d_i^{\star} \geq 0 & \text{for} & i \text{ such that } \alpha_i^{\star} = 0 \\ d_i \leq 0 & \text{for} & i \text{ such that } \alpha_i = C \\ d_i^{\star} \leq 0 & \text{for} & i \text{ such that } \alpha_i^{\star} = C \end{array} \tag{6.8}$$

and

$$-\mathbf{1} \;\leq\; \boldsymbol{p} \;\leq\; \mathbf{1} \tag{6.9}$$

$$\mathrm{card}\{d_i \;:\; d_i \neq 0\} = q. \tag{6.10}$$

Since we are searching for an optimal descent direction, which is a direction where the scalar product with the gradient is the smallest, we want indeed to minimize (6.6). The conditions (6.7)-(6.8) are necessary to ensure the feasibility of the obtained direction. The condition (6.9) is there only to ensure that the problem has a solution. Finally, (6.10) is imposed because we are searching for a direction with only q non-null components.

Note that the derivative of $\mathcal{W}$ can be easily computed:

$$\mathcal{W}^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{\star}) = \begin{pmatrix} K(\boldsymbol{\alpha} - \boldsymbol{\alpha}^{\star}) + \boldsymbol{y} + \mathbf{1}\,\epsilon \\ K(\boldsymbol{\alpha}^{\star} - \boldsymbol{\alpha}) - \boldsymbol{y} + \mathbf{1}\,\epsilon \end{pmatrix}.$$

In order to solve this problem, it thus suffices to consider

$$\omega_i = \delta_i\, \mathcal{W}_i^{'}$$

where $\delta_i = 1$ for $1 \leq i \leq l$ and $\delta_i = -1$ for $l+1 \leq i \leq 2l$. Let us force q to be even, and let us sort the ω_i in reverse order. Let us then denote φ as the bijection of $\{1 \dots 2l\}$ into itself such that the $\omega_{\varphi(i)}$ are sorted. Let us then select the $q/2$ first indices $\varphi(i)$ such that:

$$\text{if } \varphi(i) \leq l, \text{ we have } 0 < \alpha_{\varphi(i)} \leq C$$

$$\text{if } \varphi(i) > l, \text{ we have } 0 \le \alpha^\star_{\varphi(i)-l} < C$$

and let us select also the $q/2$ last indices $\varphi(i)$ such that:

$$\text{if } \varphi(i) \le l, \text{ we have } 0 \le \alpha_{\varphi(i)} < C$$

$$\text{if } \varphi(i) > l, \text{ we have } 0 < \alpha^\star_{\varphi(i)-l} \le C.$$

Since we are searching for exactly q variables, the $\varphi(i)$ must be distinct. We could have to reduce q if one variable is selected twice.

Let us now denote c_i, $i = 1 \dots q$ the q indices we just chose, we note that the direction which has for j^{th} component 0 if $j \notin \{c_1 \dots c_q\}$, $-\delta_j$ if $j \in \{c_1 \dots c_{\frac{q}{2}}\}$ and δ_j if $j \in \{c_{\frac{q}{2}+1} \dots c_q\}$, is a solution of the minimization problem (6.6) [2].

Our new working set $\mathcal{S}$ is then composed of the q variables corresponding to the indices c_i (where an index $\le l$ corresponds to α_{c_i} and an index $> l$ corresponds to $\alpha^\star_{c_i - l}$).

6.2.2 Solving the Sub-Problem

We want to solve the problem (6.3) taking into account only variables $\mathcal{S}$. To simplify the notation, let us define

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\alpha} \\ -\boldsymbol{\alpha}^\star \end{pmatrix}$$

and

$$\tilde{K} = \begin{pmatrix} K & K \\ K & K \end{pmatrix}$$

as well as

$$\boldsymbol{b} = \begin{pmatrix} -\boldsymbol{y} - \mathbf{1}\,\epsilon \\ -\boldsymbol{y} + \mathbf{1}\,\epsilon \end{pmatrix}.$$

2. To see that, let us go back to the minimization problem of

$$(z_1, \dots z_l) \longmapsto \sum_{i=1\dots l} \omega_i \, z_i \tag{6.11}$$

subject to

$$\sum_i z_i = 0 \tag{6.12}$$

$$-1 \le z_i \le 1 \tag{6.13}$$

and

$$\operatorname{card}\{z_i, z_i \ne 0\} = q \tag{6.14}$$

with $z_i = \delta_i p_i$. (The reasoning is the same if we take the constraints (6.8) into account).

In the case where $l = q = 2r$, it is easy to see that the minimum is obtained for $z_i = -1$ if $i = 1 \dots r$ and $z_i = 1$ if $i = r+1 \dots q$: if for instance z_{i_0} is augmented by $\gamma \ge 0$ for a $i_0 \le r$, then one needs to compensate by $-\gamma$ another z_j to keep (6.12). Since we want to minimize (6.11), the best thing to do, knowing that the ω_i are sorted in reverse order and keeping in mind the constraint (6.13), is to modify z_q and thus to fix $z_q = 1 - \gamma$. Equation (6.11) is then augmented by $(\omega_{i_0} - \omega_q)\gamma$, which is a positive value because the ω_i are sorted and thus we get out of the minimum.

In the case where $l > q = 2r$, suppose we found a $\boldsymbol{z}$ which is a solution of (6.11). Let us denote $k_1 \dots k_q$ the q indices of the components of $\boldsymbol{z}$ which are non-nulls. Using the same argument as in the previous paragraph, it is clear that $z_{k_1} \dots z_{k_r} = -1$ and that $z_{k_r+1} \dots z_{k_q} = 1$. In other words, if $\boldsymbol{z}$ is a solution to our problem, then we necessarily have $z_{k_i} = \pm 1$. Considering again the order of the ω_i, is becomes evident that we have to take $(k_1 = 1) \dots (k_r = r)$ and $(k_{r+1} = l - r) \dots (k_q = l)$.

The problem (6.3) is thus equivalent to minimize

$$\tilde{\mathcal{W}}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}}\tilde{K}\boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{b} \tag{6.15}$$

subject to:

$$\boldsymbol{\beta}^{\mathrm{T}}\mathbf{1} = 0 \tag{6.16}$$

and

$$0 \leq \delta_i\,\beta_i \leq C,\ \ i = 1...2l \tag{6.17}$$

where again $\delta_i = 1$ for $1 \leq i \leq l$ and $\delta_i = -1$ for $l+1 \leq i \leq 2l$.

Now let us suppose we can decompose each of the following variables into two parts (after having reordered the variables accordingly): the first one corresponds to variables $\mathcal{S}$ and the second part corresponds to the fixed variables $\mathcal{F}$:

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_{\mathcal{S}} \\ \boldsymbol{\beta}_{\mathcal{F}} \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} \boldsymbol{b}_{\mathcal{S}} \\ \boldsymbol{b}_{\mathcal{F}} \end{pmatrix}$$

and

$$\tilde{K} = \begin{pmatrix} \tilde{K}_{\mathcal{S}\mathcal{S}} & \tilde{K}_{\mathcal{S}\mathcal{F}} \\ \tilde{K}_{\mathcal{F}\mathcal{S}} & \tilde{K}_{\mathcal{F}\mathcal{F}} \end{pmatrix}.$$

Replacing these variables in (6.15), (6.16) and (6.17), and taking into account the fact that $\tilde{K}_{\mathcal{S}\mathcal{F}}^{\mathrm{T}} = \tilde{K}_{\mathcal{F}\mathcal{S}}$, the minimization problem is now

$$\tilde{\mathcal{W}}(\boldsymbol{\beta}_{\mathcal{S}}) = \frac{1}{2}\boldsymbol{\beta}_{\mathcal{S}}^{\mathrm{T}}\tilde{K}\boldsymbol{\beta}_{\mathcal{S}} - \boldsymbol{\beta}_{\mathcal{S}}^{\mathrm{T}}\left(\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}}\right) \tag{6.18}$$

(removing the constants that depend only on $\mathcal{F}$), subject to

$$\boldsymbol{\beta}_{\mathcal{S}}^{\mathrm{T}}\mathbf{1} = -\boldsymbol{\beta}_{\mathcal{F}}^{\mathrm{T}}\mathbf{1} \tag{6.19}$$

and

$$0 \leq \tilde{\delta}_i\,\beta_{\mathcal{S}_i} \leq C, \quad i = 1...q \tag{6.20}$$

where $\tilde{\delta}_i = 1$ if the i^{th} variable in the set $\mathcal{S}$ corresponds to an α_i, $\tilde{\delta}_i = -1$ if it corresponds to an $\alpha_i^{\star}$.

Minimizing (6.18) under the constraints (6.19) and (6.20) can be realized using a constrained quadratic optimizer, such as a conjugate gradient method with projection or an interior point method [11]. Moreover, following Platt's idea in *SMO* [25], if one fixes the size of the working set $\mathcal{S}$ to two, the problem can also be solved analytically.

This particular case is important because experimental results show that it often gives the fastest convergence times. We thus detailed it here. Let us simplify again the notation:

$$\boldsymbol{\beta}_{\mathcal{S}} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

$$\boldsymbol{h} = \boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}}$$

$$\zeta = -\boldsymbol{\beta}_{\mathcal{F}}^{\mathrm{T}}\mathbf{1}$$

$$\tilde{K}_{\mathcal{S}} = \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}.$$

Minimizing (6.18) under the constraints (6.19) and (6.20) is thus equivalent to minimizing

$$(z_1, z_2) \longmapsto \frac{1}{2}\left(k_{11}\, z_1^2 + k_{22}\, z_2^2 + 2k_{12}\, z_1\, z_2\right) - h_1 z_1 - h_2 z_2 \tag{6.21}$$

subject to

$$z_1 + z_2 = \zeta \tag{6.22}$$

and

$$0 \leq \tilde{\delta}_1\, z_1,\, \tilde{\delta}_2\, z_2 \leq C. \tag{6.23}$$

We are searching for a minimum in (6.21) with respect to z_1 along the line (6.22). By inserting (6.22) into (6.21), and after some derivations, it is now equivalent to minimizing

$$\Lambda : z_1 \longmapsto \frac{1}{2}\,(k_{11} - 2k_{12} + k_{22})\, z_1^2 + [(k_{12} - k_{22})\,\zeta - h_1 + h_2]\, z_1.$$

In the case[3] where $\eta = k_{11} - 2k_{12} + k_{22} > 0$, this function has a unique minimum for

$$z_1^o = \frac{(k_{22} - k_{12})\,\zeta + h_1 - h_2}{\eta}.$$

Let us now consider the constraints (6.22) and (6.23). They force z_1 to stay between L and H where

$$\left.\begin{array}{l} L = \max(0,\, \zeta - C) \\ H = \min(C, \zeta) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = 1 \text{ and } \tilde{\delta}_2 = 1$$

$$\left.\begin{array}{l} L = \max(0,\, \zeta) \\ H = \min(C, \zeta + C) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = 1 \text{ and } \tilde{\delta}_2 = -1$$

$$\left.\begin{array}{l} L = \max(-C,\, \zeta - C) \\ H = \min(0, \zeta) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = -1 \text{ and } \tilde{\delta}_2 = 1$$

$$\left.\begin{array}{l} L = \max(-C,\, \zeta) \\ H = \min(0, \zeta + C) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = -1 \text{ and } \tilde{\delta}_2 = -1.$$

Thus, taking

$$z_1^{o,\,c} = \begin{cases} H & \text{if } z_1^o > H \\ z_1^o & \text{if } L \leq z_1^o \leq H \\ L & \text{if } z_1^o < L \end{cases}$$

and

$$z_2^o = \zeta - z_1^{o,\,c}$$

the minimum of (6.21) under the constraints (6.22) and (6.23) is obtained at $(z_1^{o,\,c}, z_2^o)$ if $\eta > 0$. In the pathological case where $\eta \leq 0$, it is clear that the solution

$$z_1^o = \begin{cases} L & \text{if } \Lambda(L) < \Lambda(H) \\ H & \text{if } \Lambda(L) \geq \Lambda(H) \end{cases}$$

and

$$z_2^o = \zeta - z_1^o$$

is the minimum.

3. Note that this is the most common case. For instance for a gaussian kernel with distinct examples $\boldsymbol{x}_i$, it is easy to see that is is always the case.

6.2.3 Shrinking

The idea of shrinking is to remove some variables that are stuck to the bounds 0 or C, and that will *a priori* not change anymore. To do this, we use the fact that $(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star)$ minimizes the problem (6.3) under the constraints (6.4) and (6.5) *if and only if* there exists numbers $\boldsymbol{\lambda}^{up} \in \boldsymbol{R}^{2l}$, $\boldsymbol{\lambda}^{low} \in \boldsymbol{R}^{2l}$, $\lambda^{eq} \in \boldsymbol{R}$ that verify the following *KKT conditions*:

$$\mathcal{W}'(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) + \lambda^{eq} \begin{pmatrix} \mathbf{1} \\ -\mathbf{1} \end{pmatrix} - \boldsymbol{\lambda}^{low} + \boldsymbol{\lambda}^{up} = \mathbf{0} \tag{6.24}$$

$$\lambda_i^{low}\, \alpha_i = 0,\;\; i = 1 \ldots l \quad \text{and} \quad \lambda_i^{low}\, \alpha_{i-l}^\star = 0,\;\; i = (l+1) \ldots 2l \tag{6.25}$$

$$\lambda_i^{up}\, (\alpha_i - C) = 0,\;\; i = 1 \ldots l \quad \text{and} \quad \lambda_i^{up}\, \left(\alpha_{i-l}^\star - C\right) = 0,\;\; i = (l+1) \ldots 2l \tag{6.26}$$

$$\boldsymbol{\lambda}^{low} \geq \mathbf{0} \tag{6.27}$$

$$\boldsymbol{\lambda}^{up} \geq \mathbf{0} \tag{6.28}$$

$$(\boldsymbol{\alpha} - \boldsymbol{\alpha}^\star)^{\mathrm{T}}\, \mathbf{1} = 0 \tag{6.29}$$

$$\mathbf{0} \leq \boldsymbol{\alpha}^\star,\, \boldsymbol{\alpha} \leq \boldsymbol{C}. \tag{6.30}$$

Note that if $\lambda_i^{low} > 0$, then the corresponding variable is equal to 0. Also, if $\lambda_i^{up} > 0$, then the corresponding variable is equal to C. The idea is thus to search at each iteration for variables $\boldsymbol{\lambda}^{up}$, $\boldsymbol{\lambda}^{low}$ and λ^{eq} that verify *as well as possible*[4] the equations (6.24)-(6.28), and to remove a variable which is stuck at 0 if its coefficient λ_i^{low} is strictly positive (or just below a constant ϵ_{shrink}) during a given number of iterations. Using the same idea, we also eliminate a variable which is stuck at C if its coefficient λ_i^{up} stays strictly positive for a sufficient number of iterations.

To estimate $\boldsymbol{\lambda}^{low}$ or $\boldsymbol{\lambda}^{up}$, we start by estimating λ^{eq} (note that if $0 < \alpha_i < C$ then $\lambda_i^{low} = \lambda_i^{up} = 0$ and if $0 < \alpha_i^\star < C$ then $\lambda_{i+l}^{low} = \lambda_{i+l}^{up} = 0$). Noting

$$A = \{i,\, 0 < \alpha_i < C\},\;\; B = \{i,\, 0 < \alpha_i^\star < C\}$$

we have (with $\hat{\lambda}$ standing for an estimation of λ) :

$$\hat{\lambda}^{eq} = \frac{1}{|A \cup B|} \left(\sum_{i \in B} \mathcal{W}_{i+l}'(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) - \sum_{i \in A} \mathcal{W}_i'(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) \right). \tag{6.31}$$

Then if we have

$$\begin{array}{rlcrcl} \alpha_i = 0 & \text{we compute} & \hat{\lambda}_i^{low} & = & \hat{\lambda}^{eq} & + \;\; \mathcal{W}_i' \\ \alpha_i^\star = 0 & \text{we compute} & \hat{\lambda}_{i+l}^{low} & = & -\hat{\lambda}^{eq} & + \;\; \mathcal{W}_{i+l}' \\ \alpha_i = C & \text{we compute} & \hat{\lambda}_i^{up} & = & -\hat{\lambda}^{eq} & - \;\; \mathcal{W}_i' \\ \alpha_i^\star = C & \text{we compute} & \hat{\lambda}_{i+l}^{up} & = & \hat{\lambda}^{eq} & - \;\; \mathcal{W}_{i+l}' \end{array} \tag{6.32}$$

and if a variable stays a sufficient number of iterations at 0 (or C) with its corresponding coefficient $\hat{\lambda}_j^{low} > \epsilon_{shrink}$ (or $\hat{\lambda}_j^{up} > \epsilon_{shrink}$), then we remove it from the problem.

4. If we were *really* able to find such variables, this would mean that $(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star)$ is a solution to our problem.

6.2.4 Termination Criterion

Given what has been said in the section on *shrinking*, if we can always have (6.29) and (6.30) during the resolution of a sub-problem, a *reasonable* termination criterion is to verify that the λ estimated by (6.31) and (6.32) verifies the conditions (6.24)-(6.28) with a given precision ϵ_{end}.

Thus we simply verify that

$$\begin{array}{rc} \text{for } i \text{ such that } 0 < \delta_i\,\beta_i < C: & \lambda^{eq} - \epsilon_{end} \leq -\delta_i \mathcal{W}_i^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) \leq \lambda^{eq} + \epsilon_{end} \\ \text{for } i \text{ such that } \beta_i = 0: & \mathcal{W}_i^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) + \delta_i \lambda^{eq} \geq -\epsilon_{end} \\ \text{for } i \text{ such that } \beta_i = C: & \mathcal{W}_i^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) + \delta_i \lambda^{eq} \leq \epsilon_{end} \end{array}$$

with $\delta_i = 1$ for $1 \leq i \leq l$, $\delta_i = -1$ for $(l+1) \leq i \leq 2l$ and

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\alpha} \\ -\boldsymbol{\alpha}^\star \end{pmatrix}.$$

6.2.5 Implementation Details

Note that in all the steps needed during one iteration, only two of them could be time consuming: the one that computes $\mathcal{W}_i^{'}$ and the one that computes $\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}}$ in (6.18).

We therefore propose to keep in memory a table of $\mathcal{W}_i^{'}$. Moreover, to update this variable, we can see that

$$\begin{array}{ll} \text{for } i \leq n & \mathcal{W}_i^{'\,(t+1)} = \mathcal{W}_i^{'\,(t)} + \sum_{j \in S_1} K_{ij} \left(\alpha_j^{(t+1)} - \alpha_j^{(t)}\right) - \sum_{j \in S_2} K_{ij} \left(\alpha_j^{\star\,(t+1)} - \alpha_j^{\star\,(t)}\right) \\ \text{for } i > n & \mathcal{W}_i^{'\,(t+1)} = \mathcal{W}_i^{'\,(t)} - \sum_{j \in S_1} K_{ij} \left(\alpha_j^{(t+1)} - \alpha_j^{(t)}\right) + \sum_{j \in S_2} K_{ij} \left(\alpha_j^{\star\,(t+1)} - \alpha_j^{\star\,(t)}\right) \end{array}$$

where

$$S_1 = \{i,\ \ \alpha_i \in \mathcal{S}\} \ \text{ and } \ S_2 = \{i,\ \ \alpha_i^\star \in \mathcal{S}\}.$$

For the computation of $\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}}$, we can use the following trick:

$$\begin{array}{rcl} \left(\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}}\right)_i & = & (\boldsymbol{b}_{\mathcal{S}})_i - \left(\tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}} + \tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_{\mathcal{S}}\right)_i + \left(\tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_{\mathcal{S}}\right)_i \\ & = & \left\{ \begin{array}{ll} -\mathcal{W}_i + \left(\tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_{\mathcal{S}}\right)_i & if\ i \leq n \\ \mathcal{W}_i + \left(\tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_{\mathcal{S}}\right)_i & if\ i > n. \end{array} \right. \end{array}$$

With these two ideas, one can reduce considerably the computational time: instead of computing all the lines of the matrix K, one can only compute the lines corresponding to the variables in $\mathcal{S}$.

Since we only need these lines for the computations, and since it quickly becomes intractable for large problems to keep all the matrix K in memory (the size of the matrix being quadratic with respect to the number of examples), it is interesting to implement a *cache* that keeps in memory the lines of K that corresponds to the most used variables instead of recomputing them at each iteration.

6.3 Experimental Results

6.3.1 Speed Comparisons

We compared our SVM implementation for regression problems (named *SVMTorch*) to the one from Flake and Lawrence [10] using their publicly available software *Nodelib*, which is an enhanced version of *SMO* [25]. Both these algorithms use an internal cache in order to be able to solve large-scale problems. All the experiments presented in this section have been done on a SPARC Ultra-10 440Mhz, with the *gcc* compiler. The parameters of the algorithms were not chosen to obtained the best generalization performances, since the goal was to *compare the speed* of the algorithms. Both programs used the same parameters with regard to cache, precision, etc... For *Nodelib*, the other parameters were set using the default values proposed by the authors. All the programs were compiled using *double* precision. Finally, each program was trained/tested on the same data for a given *benchmark*.

We compared the programs on two different tasks, using up to three different training set sizes. The first task was to predict the average number of sunspots of one year, given the average number of sunspots of the previous 12 years. Since we had access to daily data, we were able to artificially create one input/output pair for each day: at each day, we compute the yearly average of the year starting the next day, which has to be predicted using the 12 previous yearly averages. Using this technique, we had access to 43000 examples with input dimension equal to 12. The last 3000 examples served as the test set, while we created 3 different training sets of varying sizes: 5000, 20000 and 40000. The second task was created artificially using the daily sunspot series in order to test the algorithm with a big input dimension. We selected it to be equal to 240. The test set had 2000 examples while we created 2 different training sets of varying sizes: 5000 and 20000.

Table 6.1 gives the computation time results of all the experiments in CPU-seconds. In these experiments, the following SVM parameters were used: $C = 1000$ and precision $= 0.01$. As it can be seen, *SVMTorch* easily outperformed *Nodelib* in all experiments: the generalization performance of both algorithm as well as the number of support vectors found were similar but the time spent to find the solution was largely less for *SVMTorch* than for *Nodelib*. However, it seems that as the number of training examples grows, the speed difference between both algorithms narrows. The speed difference is also smaller when the input dimension is higher. A further analysis should probably be done in order to understand these difference variations.

6.3.2 Analysis of the Cache Size

In order to show the impact of the cache size on the time needed to train an SVM, we have done two series of experiments, one for each dataset, varying the size of the cache (from 10 to 50 Mb for the sunspots dataset and from 50 to 250 for the artificial dataset). Figure 6.1 shows the results: the more cache you can afford, the faster the algorithm will find a solution, but the optimal size of the cache is of course related to the number of training samples: after a given cache size, no more speed improvement can be achieved.

6.4 Conclusion

We have presented a new decomposition algorithm intended to efficiently solve large-scale regression problems using SVMs. This algorithm followed the same principles as the one from Joachims [14] for his classification algorithm. We also proposed to set the size of the sub-problems to 2 in order to solve

Yearly Sunspots ($\sigma = 900$, $\epsilon = 20$)					
	Train size	# of SV	Train MSE	Test MSE	# of CPU-seconds
Nodelib	5000	431	146.15	246.18	245
SVMTorch		432	146.14	246.15	3
Nodelib	20000	1151	126.59	287.60	1253
SVMTorch		1155	126.58	287.64	30
Nodelib	40000	1911	119.46	363.88	2499
SVMTorch		1871	119.76	369.22	94
Artificial Data ($\sigma = 10$, $\epsilon = 0.5$)					
	Train size	# of SV	Train MSE	Test MSE	# of CPU-seconds
Nodelib	5000	397	0.09	4.82	825
SVMTorch		394	0.10	4.66	90
Nodelib	20000	1013	0.08	0.60	5598
SVMTorch		1002	0.08	0.60	1037

Tab. 6.1 – *Comparative speed results between* SVMTorch *and* Nodelib *on two different databases and up to 3 different training set sizes. All the experiments used an internal cache of 100Mb.*

it analytically as it is done in SMO [25]. An internal cache keeping part of the kernel matrix in memory enables the program to solve large problems without the need to keep quadratic resources in memory and without the need to recompute every kernel evaluation, which leads to an overall fast algorithm. An experimental comparison with another algorithm has shown significant time improvement for large-scale problems.

Finally, note that in the available source code of *SVMTorch*, we also implemented the same algorithm for classification problems. The result is thus similar mathematically to the one proposed by Joachims in its *SVM-Light* software [14], but the speed obtained was twice as fast as *SVM-Light* (there are probably many implementation differences, such as the cache, which was not fully described in Joachims's paper).

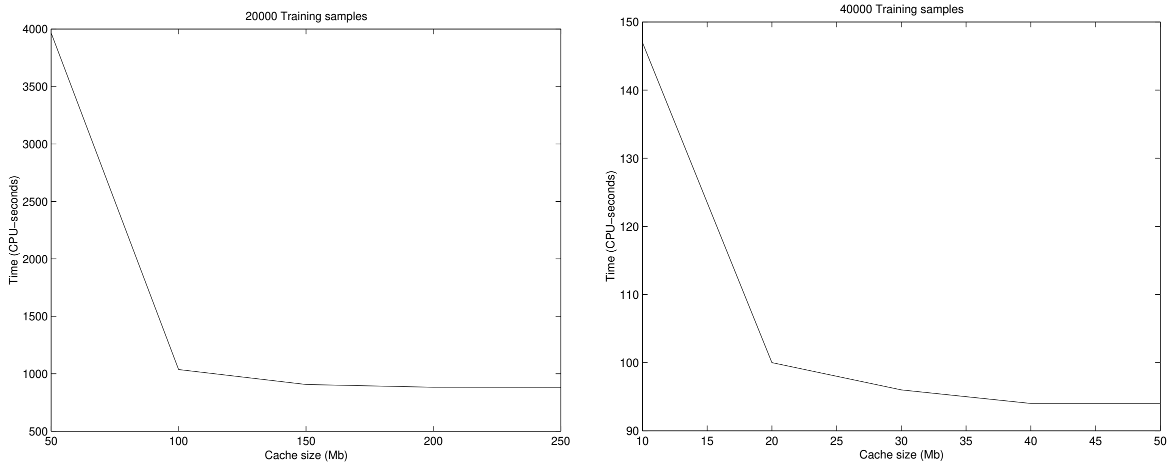

FIG. 6.1 – *Evolution of the convergence speed (in seconds) with respect to the size of the internal cache (in Mb). The left part gives the computational time for the artificial dataset trained with 20000 examples (in dimension 240), while the right part gives the computational time for the sunspots dataset trained with 40000 examples (in dimension 12).*

6.5 *Appendix:* Things that should have been in the paper

6.5.1 On the experiments

We forgot to specify that the kernel used in all the experiments was a Gaussian kernel. Thanks to Flake[5] for his remark on this.

In the conclusion of our paper, we said that we also implemented a classification version of the algorithm which was similar to the one proposed by Joachims [14] and that our implementation was twice as fast as *SVM-Light*. This assertion holds only for non-sparse data because *SVM-Light* has been specially designed for sparse data, while it was not the case in the first version of *SVMTorch*. The current version, which now includes sparse data format, is 1.33 times faster that *SVM-Light* for sparse data (and still 2 times faster for non-sparse data).

6.5.2 On the decomposition method

Since the publication of our technical report, we have been aware of many other decomposition algorithms for regression problems that we have not even cited. We try here to resume their work and the relation with our paper.

Shevade *et al* [26] proposed two modifications of the *SMO* algorithm for regression, based on a previous paper from the same team [16] for the classification problem. Laskov [17] proposed also a decomposition method for regression problems which is very similar to the second one from Shevade *et al.* In fact, it is easy to see that Laskov's method with a subproblem of size 2 uses the same selection algorithm as well as the same termination criterion.

Their method for selecting the working set is *very* similar to the one we proposed, but while we

5. `http://www.neci.nj.nec.com/homepages/flake/`

propose to select variables α_i independantly of their counterpart $\alpha_i^\star$, they propose to select simultaneously *pairs* of variables $\{\alpha_i,\, \alpha_i^\star\}$. Even if this seems to be a small difference, let us note that since $\alpha_i\, \alpha_i^\star = 0\ \forall i$, one of the two variables α_i or $\alpha_i^\star$ is always equal to 0, and choosing the α_i and the $\alpha_i^\star$ independantly can thus help to quickly eliminate half of the variables, thanks to the *shrinking* phase [6], which of course have a direct impact on the speed of our program.

Similarly, Smola and Schölkopf [27] also proposed earlier to use a decomposition algorithm for regression based on *SMO* using an analytical solution for the subproblem, but again they propose to select 2 pairs of variables (2 α and their corresponding $\alpha^\star$) instead of 2 variables, which leads to a different mathematical formulation. As for Laskov and Shevada *et al*, they do not use *shrinking* which is, in our opinion, the main speed gain of our algorithm. Finally, Flake and Lawrence [10] proposed again a modification of *SMO* for regression which uses the heuristics proposed by Platt [25] and those from Smola and Schölkopf [27] but works on a new variable $\lambda_i = \alpha_i - \alpha_i^\star$, which leads to a different analytical solution. In fact, all the analytical solutions proposed by these authors are different but need to handle multiple cases for the solution, except our method.

6.5.3 On the convergence of the algorithm

In our paper, we did not talked about the convergence of our algorithm. A paper from Chang *et al* [4] talks about the convergence of some SVM algorithms based on a decomposition method. However, using their arguments, we cannot conclude that our algorithm converges to the optimum, even when no shrinking is done. The hypothesis they use in their proof regarding their method to search for a feasible solution is slightly different from our method and thus we cannot use their proof in our case. Keerthi *et al* [15] also proposed a convergence proof for their method [16], but it applies only to their classification case. However, we will see in the next section that it also applies to our classification method *as well as* our regression method.

6.6 Remarks on the relation between many SVM algorithms

As we said in our paper, the algorithm we used in classification is the same, mathematically speaking, as the one proposed by Joachims [14]. Let us now consider [7] the paper from Keerthi *et al* [16] which proposes two algorithms based on Platt's algorithm, *SMO*. In particular, let us focus on the second method they propose and let us compare this method to the algorithm proposed by Joachims in the case of a working set of size 2. We strongly suggest to the reader to refer to the papers from Joachims and Keerthi *et al* in order to understand the following notations which will not be re-explained here.

At each iteration, Keerthi *et al* propose to start by selecting two variables in their working set. Following their notation, let us denote

$$\begin{array}{rcl} I_0 & = & \{i : 0 < \alpha_i < C\} \\ I_1 & = & \{i : y_i = 1,\, \alpha_i = 0\} \\ I_2 & = & \{i : y_i = -1,\, \alpha_i = C\} \\ I_3 & = & \{i : y_i = 1,\, \alpha_i = C\} \\ I_4 & = & \{i : y_i = -1,\, \alpha_i = 0\} \end{array}$$

6. this is verified in practice

7. Thanks to Patrick Haffner (`http://www.research.att.com/~haffner`) and Ryan Rifkin (`http://five-percent-nation.mit.edu/PersonalPages/rif/`) who have stimulated our interest on this.

and let us denote also i_{low} and i_{up} the index of the two selected variables. They verify

$$F_{i_{low}} = b_{low} = \max\{F_i : \ i \in I_0 \cup I_3 \cup I_4\}$$

and

$$F_{i_{up}} = b_{up} = \min\{F_i : \ i \in I_0 \cup I_1 \cup I_2\}$$

where

$$F_i = \sum_j \alpha_j y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) - y_i.$$

One can easily remark that $F_i = \omega_i$, where ω_i is the sorting variable used by Joachims in his paper. Joachims defines the following constraints

$$\begin{array}{l} d_i \geq 0, \ \ \forall i : \alpha_i = 0 \\ d_i \leq 0, \ \ \forall i : \alpha_i = C \end{array} \tag{6.33}$$

and select the following working set variables:

- the one that corresponds to the highest ω_i such that $0 < \alpha_i < C$ or such that $d_i = -y_i$ verify (6.33), and
- the one that corresponds to the smallest ω_i such that $0 < \alpha_i < C$ or such that $d_i = y_i$ verify (6.33).

This is indeed equivalent to the choice made by Keerthi *et al.*

Both algorithms then solve the sub-problem and test the optimality of the general problem. The algorithm from Keerthi *et al* stops when

$$b_{low} - b_{up} \leq \tau$$

where τ is a tolerance factor defined by the user. The algorithm from Joachims stops when the following conditions are verified:

$$\begin{array}{c} \forall i \text{ such that } 0 < \alpha_i < C, \ \ \lambda^{eq} - \tau \leq y_i - \sum_j \alpha_j y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) \leq \lambda^{eq} + \tau \\ \forall i \text{ such that } \alpha_i = 0, \ \ y_i(\sum_j \alpha_j y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) + \lambda^{eq}) \geq 1 - \tau \\ \forall i \text{ such that } \alpha_i = C, \ \ y_i(\sum_j \alpha_j y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) + \lambda^{eq}) \leq 1 + \tau \end{array} \tag{6.34}$$

where λ^{eq} is defined as follows

$$\lambda^{eq} = \frac{1}{|A|} \sum_{i \in A} \left[y_i - \sum_j \alpha_j y_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) \right]$$

with

$$A = \{i : 0 < \alpha_i < C\}.$$

It is easy to see that equations (6.34) are equivalent to

$$\begin{array}{c} \forall i \in I_0, \quad \lambda^{eq} - \tau \leq -F_i \leq \lambda^{eq} + \tau \\ \forall i \in I_1 \cup I_2, \quad F_i \geq -\lambda^{eq} - \tau \\ \forall i \in I_3 \cup I_4, \quad F_i \leq -\lambda^{eq} + \tau. \end{array}$$

Moreover, since $-b_{low} \leq \lambda^{eq} \leq -b_{up}$, if the optimality conditions from Keerthi *et al* are verified, then

$$\forall i \in I_0 \cup I_3 \cup I_4, \quad F_i \leq b_{up} + \tau \leq -\lambda^{eq} + \tau$$

and

$$\forall i \in I_0 \cup I_1 \cup I_2, \quad -F_i \leq -b_{low} + \tau \leq \lambda^{eq} + \tau$$

which implies the optimality conditions from Joachims.

Since the optimality test of *SVM-Light* is weaker that the one from Keerthi *et al* [15], it is easy to see that one can apply their theorem to show that *SVM-Light* converges for subproblems of size 2 (as well as our classification algorithm).

In the same way, one can show that the optimality test we used in our regression algorithm is weaker than the general algorithm proposed by Keerthi *et al* [15] and it is easy to see that given the fact that our algorithm uses a selection method that choose independantly the α and the $\alpha^{\star}$, the proof from Keerthi *et al also applies to our regression algorithm when the subproblem size is set to 2.*

6.7 Conclusion

In conclusion, we note that all these decomposition algorithms are extremely related. For instance, subset selection algorithms from Keerthi *et al* and Joachims are strictly identical if the *shrinking* is not used and for a working subset of size 2 in *SVM-Light*.

Moreover, it is very easy to see that the regression method from Laskov (again with a subset of size 2) is equivalent to the one from Shevade *et al*, which is the same as the classification one from Keerthi *et al*. Note also that the method from Flake and Lawrence could be modified using the second modification from Keerthi *et al* and would thus be enhanced.

Finally, we think that shrinking makes the main difference with regard to speed, and the selection method we have chosen simplifies the resolution of the analytic quadratic problem and enables to obtain a convergence proof for the regression problem.

CHAPITRE 7 On the Convergence of *SVMTorch*, an Algorithm for Large-Scale Regression Problems

Authors: Ronan Collobert and Samy Bengio
Submitted to Journal of Machine Learning Research
Available on ftp://ftp.idiap.ch/pub/reports/2000/rr00-24.ps.gz

Abstract. Recently, many researchers have proposed decomposition algorithms for SVM regression problems (see for instance [27, 10, 17, 26]). In a previous paper [6], we also proposed such an algorithm, named *SVMTorch*. In this paper, we show that while there is actually no convergence proof for any other decomposition algorithm for SVM regression problems to our knowledge, such a proof does exist for *SVMTorch* for the particular case where no shrinking is used and the size of the working set is equal to 2, which is the size that gave the fastest results on most experiments we have done. This convergence proof is in fact mainly based on the convergence proof given by Keerthi and Gilbert [15] for their SVM classification algorithm.

7.1 Introduction

Vapnik has proposed in [28] a method to solve regression problems using Support Vector Machines (SVMs). It has yielded excellent performances on many regression and time series prediction problems (see for instance [23, 9]). Recently, we have proposed a fast decomposition algorithm for large-scale regression problems [6] using SVMs. Unlike other decomposition algorithms for regression problems (see for instance [27, 10, 17, 26]), there exists a convergence proof for our algorithm, and the goal of this paper is to show such a proof. Let us first recall the general problem of SVM for regression and our method to solve it.

Given a training set of l *examples* $(\boldsymbol{x}_i, y_i)$ with $\boldsymbol{x}_i \in E$ and $y_i \in \boldsymbol{R}$, where E is an Euclidean space with a scalar product denoted $(\,\cdot\,)$, we want to estimate the following linear regression:

$$f(\boldsymbol{x}) = (\boldsymbol{w} \cdot \boldsymbol{x}) + b$$

(with $b \in \boldsymbol{R}$) with a precision ϵ. For this, we minimize

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l} |y_i - f(\boldsymbol{x}_i)|_\epsilon$$

where $\frac{1}{2}\|\boldsymbol{w}\|^2$ is a regularization factor, C is a fixed constant, and $|.|_\epsilon$ is the ϵ-insensitive loss function defined by Vapnik:

$$|z|_\epsilon = \max\{0,|z| - \epsilon\}.$$

Written as a constrained optimization problem, it amounts to minimizing

$$\tau(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^\star) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^\star)$$

subject to

$$((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) - y_i \leq \epsilon + \xi_i \tag{7.1}$$

$$y_i - ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) \leq \epsilon + \xi_i^\star \tag{7.2}$$

$$\xi_i, \xi_i^\star \geq 0.$$

To generalize to non-linear regression, we replace the dot product with a kernel[1] $k(\cdot)$. Then, introducing Lagrange multipliers $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^\star$, the optimization problem can be stated as:

Minimize the function

$$\mathcal{W}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) = \frac{1}{2}(\boldsymbol{\alpha}^\star - \boldsymbol{\alpha})^{\mathrm{T}} K (\boldsymbol{\alpha}^\star - \boldsymbol{\alpha}) - (\boldsymbol{\alpha}^\star - \boldsymbol{\alpha})^{\mathrm{T}} \boldsymbol{y} + \epsilon(\boldsymbol{\alpha}^\star + \boldsymbol{\alpha})^{\mathrm{T}} \mathbf{1} \tag{7.3}$$

subject to

$$(\boldsymbol{\alpha} - \boldsymbol{\alpha}^\star)^{\mathrm{T}} \mathbf{1} = 0 \tag{7.4}$$

and

$$0 \leq \alpha_i^\star,\, \alpha_i \leq C, \quad i = 1...l \tag{7.5}$$

where K is the matrix with coefficients $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The estimate of the regression function at a given point is then

$$f(\boldsymbol{x}) = \sum_{i=1}^{l}(\alpha_i^\star - \alpha_i) k(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

where b is computed using the fact that (7.1) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$ and (7.2) becomes an equality with $\xi_i^\star = 0$ if $0 < \alpha_i^\star < C$.

Let us denote

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\alpha} \\ -\boldsymbol{\alpha}^\star \end{pmatrix}$$

and

$$\tilde{K} = \begin{pmatrix} K & K \\ K & K \end{pmatrix}$$

as well as

$$\boldsymbol{b} = \begin{pmatrix} -\boldsymbol{y} - \mathbf{1}\,\epsilon \\ -\boldsymbol{y} + \mathbf{1}\,\epsilon \end{pmatrix}.$$

The optimization problem is thus (see [28, 6] for more details):

$$\tilde{\mathcal{W}}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}} \tilde{K} \boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{b} \tag{7.6}$$

1. note that this kernel needs to verify the Mercer's conditions in order for convergence proofs to work.

subject to

$$\boldsymbol{\beta}^{\mathrm{T}}\,\mathbf{1} = 0 \tag{7.7}$$

and

$$0 \le \delta_i\,\beta_i \le C, \;\; i = 1...2l \tag{7.8}$$

where $\delta_i = 1$ for $1 \le i \le l$ and $\delta_i = -1$ for $l+1 \le i \le 2l$.

Solving this optimization problem with a decomposition method (as proposed for instance by Osuna *et al* [24]) consists in an iterative procedure where at each iteration, one selects a set of variables in $\{\beta_1, \ldots, \beta_{2l}\}$ and then minimizes (7.6) with respect to the selected variables. Note that this selection scheme is particular to *SVMTorch*, while all the other decomposition algorithms we have seen for regression problems select pairs of variables $(\alpha_i, \alpha_i^\star)$. This apparently small difference enables us to show a convergence proof for our algorithm.

In the particular case where the number of selected variables at each iteration is fixed to 2, one can then use a analytical method to solve the subproblem, as we have shown in [6] and was also proposed for the *SMO* algorithm [25].

Going back to our method proposed in [6], in the case where no *shrinking*[2] is done and with the number of selected variables set to 2, our algorithm can then be written as:

Algorithm 1 *Given a $\tau > 0$,*

1. *Set $\boldsymbol{\beta} = \mathbf{0}$.*
2. *Select variables β_{i_0} and β_{i_1} where i_0 and i_1 verify*

$$\begin{array}{l} \delta_{i_0}\mathcal{W}'_{i_0}(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) = \max_{i\in A_1}\,\delta_i\mathcal{W}'_i(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) \\ \delta_{i_1}\mathcal{W}'_{i_1}(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) = \min_{i\in A_2}\,\delta_i\mathcal{W}'_i(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) \end{array} \tag{7.9}$$

   where again $\delta_i = 1$ for $1 \le i \le l$ and $\delta_i = -1$ for $l+1 \le i \le 2l$, and

$$A_1 = \{i \;:\; 0 < \beta_i \le C \textit{ for } i \le l\} \cup \{i \;:\; 0 \le \delta_i\beta_i < C \textit{ for } i > l\}$$

$$A_2 = \{i \;:\; 0 \le \beta_i < C \textit{ for } i \le l\} \cup \{i \;:\; 0 < \delta_i\beta_i \le C \textit{ for } i > l\}.$$

3. *Solve analytically the optimization problem (7.6) with respect to these two variables. Update $\boldsymbol{\beta}$ with respect to the obtained solution.*
4. *Verify the optimality conditions*

$$\begin{array}{rc} \textit{for } i \textit{ such that } 0 < \delta_i\,\beta_i < C: & \hat\lambda^{eq} - \tau \le -\delta_i\mathcal{W}'_i(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) \le \hat\lambda^{eq} + \tau \\ \textit{for } i \textit{ such that } \beta_i = 0: & \mathcal{W}'_i(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) + \delta_i\hat\lambda^{eq} \ge -\tau \\ \textit{for } i \textit{ such that } \delta_i\beta_i = C: & \mathcal{W}'_i(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) + \delta_i\hat\lambda^{eq} \le \tau \end{array} \tag{7.10}$$

   where

$$\hat\lambda^{eq} = \frac{1}{|A \cup B|}\left(\sum_{i\in B}\mathcal{W}'_{i+l}(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star) - \sum_{i\in A}\mathcal{W}'_i(\boldsymbol{\alpha},\boldsymbol{\alpha}^\star)\right) \tag{7.11}$$

   with

$$A = \{i,\, 0 < \alpha_i < C\}, \;\; B = \{i,\, 0 < \alpha_i^\star < C\}.$$

   When all optimality conditions are verified, stop the algorithm; otherwise, go back to step 2.

The aim of this paper is to show that such an algorithm converges. In section 7.2, we expose a recent convergence theorem from Keerthi and Gilbert [15] while in section 7.3 we show how it applies to our algorithm.

2. *Shrinking* is a heuristic used to eliminate variables that tend to be stuck at bounds 0 or C for many iterations.

7.2 The Convergence Theorem of Keerthi and Gilbert

In a recent paper, Keerthi and Gilbert [15] showed a convergence theorem for a modified version of *SMO* given by Keerthi *et al* [16] for SVM classification problems. In fact, in their paper, Keerthi and Gilbert talk about the general case where one wants to minimize

$$f(\boldsymbol{\eta}) = \frac{1}{2}\boldsymbol{\eta}^{\top} Q \boldsymbol{\eta} + \boldsymbol{p}^{\top}\boldsymbol{\eta} \tag{7.12}$$

subject to

$$a_i \leq \eta_i \leq b_i \ \forall i \ \text{ and } \ \sum_i z_i \eta_i = c$$

where Q is symmetric positive semi-definite, $a_i < b_i \ \forall i$ and $z_i \neq 0 \ \forall i$. They call $\mathcal{F}$ the feasible set of the quadratic problem. They suppose $\mathcal{F}$ non-empty, and f bounded below on $\mathcal{F}$. They denote

$$F_i(\boldsymbol{\eta}) = ([Q\boldsymbol{\eta}]_i + p_i)/z_i$$

and define the sets

$$\begin{array}{rcl}
I_0(\boldsymbol{\eta}) & = & \{i : a_i < \eta_i < b_i\} \\
I_1(\boldsymbol{\eta}) & = & \{i : z_i > 0, \, \eta_i = a_i\} \\
I_2(\boldsymbol{\eta}) & = & \{i : z_i < 0, \, \eta_i = b_i\} \\
I_3(\boldsymbol{\eta}) & = & \{i : z_i > 0, \, \eta_i = b_i\} \\
I_4(\boldsymbol{\eta}) & = & \{i : z_i < 0, \, \eta_i = a_i\} \\
 & & \\
I_{up}(\boldsymbol{\eta}) & = & I_0(\boldsymbol{\eta}) \cup I_1(\boldsymbol{\eta}) \cup I_2(\boldsymbol{\eta}) \\
I_{low}(\boldsymbol{\eta}) & = & I_0(\boldsymbol{\eta}) \cup I_3(\boldsymbol{\eta}) \cup I_4(\boldsymbol{\eta})
\end{array}$$

Moreover, they define (i, j) as being a *violating pair* if one of the following conditions is verified:

$$\begin{array}{llll}
i \in I_{up}(\boldsymbol{\eta}), & j \in I_{low}(\boldsymbol{\eta}) & \textit{and} & F_i(\boldsymbol{\eta}) < F_j(\boldsymbol{\eta}) - \tau \\
i \in I_{low}(\boldsymbol{\eta}), & j \in I_{up}(\boldsymbol{\eta}) & \textit{and} & F_i(\boldsymbol{\eta}) > F_j(\boldsymbol{\eta}) + \tau.
\end{array}$$

They then prove that *the following algorithm stops after a finite number of iterations* :

Algorithm 2 (GSMO) *Given a* $\tau > 0$

1. *Choose some* $\boldsymbol{\eta} \in \mathcal{F}$.
2. *If* $\boldsymbol{\eta}$ *satisfies*

$$\min_{i \in I_{up}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) \geq \max_{i \in I_{low}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) - \tau \tag{7.13}$$

   then stop.
3. *Choose* (j_0, j_1) *a* τ*-violating pair. Minimize* f *on* $\mathcal{F}$ *while varying only* η_{j_0} *and* η_{j_1}. *Set* $\boldsymbol{\eta}$ *to the point thus obtained. Go to step* 2.

7.3 Convergence of our Algorithm

Using the previous notation, let $\boldsymbol{\eta} = \boldsymbol{\beta}$, $\boldsymbol{p} = -\boldsymbol{b}$, $Q = \tilde{K}$, $z_i = 1 \ \forall i$, $c = 0$, and

$$\begin{array}{lll}
a_i = 0, & b_i = C & \text{for } 0 \leq i \leq l \\
a_i = -C, & b_i = 0 & \text{for } (l+1) \leq i \leq 2l.
\end{array}$$

Our optimization problem (7.6) is then totally equivalent to the one from Keerthi and Gilbert (7.12). Moreover, it is easy to see that the hypothesis needed by Keerthi and Gilbert are verified: the initial solution $\boldsymbol{\beta} = \mathbf{0}$ is in $\mathcal{F}$, K is positive semi-definite for a kernel k verifying Mercer's conditions, and *a fortiori* $\tilde{K}$ is also positive semi-definite. Moreover $z_i \neq 0\ \forall i$, $\mathcal{F}$ is clearly non-empty and compact. f is thus bounded below.

Let us now show that our stopping conditions (7.10) are *weaker* than those from Keerthi and Gilbert (7.13). It is easy to see that

$$F_i(\boldsymbol{\eta}) = \delta_i \mathcal{W}_i^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{\star})$$

and thus one can easily deduce that the conditions (7.10) are equivalent to

$$\begin{array}{ll} \forall i \in I_0(\boldsymbol{\eta}): & \hat{\lambda}^{eq} - \tau \leq -F_i(\boldsymbol{\eta}) \leq \hat{\lambda}^{eq} + \tau \\ \forall i \in I_1(\boldsymbol{\eta}): & F_i(\boldsymbol{\eta}) \geq -\hat{\lambda}^{eq} - \tau \\ \forall i \in I_3(\boldsymbol{\eta}): & F_i(\boldsymbol{\eta}) \leq -\hat{\lambda}^{eq} + \tau. \end{array} \tag{7.14}$$

Hence if $\boldsymbol{\eta}$ verifies the stopping conditions (7.13), and taking into account the fact that

$$-\max_{i \in I_{low}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) \leq \hat{\lambda}^{eq} \leq -\min_{i \in I_{up}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta})$$

then

$$\begin{array}{llcl} \forall i \in I_{low}(\boldsymbol{\eta}) & F_i(\boldsymbol{\eta}) & \leq & \min_{i \in I_{up}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) + \tau \\ & & \leq & -\hat{\lambda}^{eq} + \tau \end{array}$$

and

$$\begin{array}{llcl} \forall i \in I_{up}(\boldsymbol{\eta}) & -F_i(\boldsymbol{\eta}) & \leq & -\max_{i \in I_{low}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) + \tau \\ & & \leq & \hat{\lambda}^{eq} + \tau \end{array}$$

which implies the conditions (7.14). In other words, our stopping conditions are weaker than those from Keerthi and Gilbert.

Finally, noting that in our selected variables (7.9) one has [3]

$$A_1 = I_0(\boldsymbol{\eta}) \cup I_3(\boldsymbol{\eta}) = I_{low}(\boldsymbol{\eta})$$

$$A_2 = I_0(\boldsymbol{\eta}) \cup I_1(\boldsymbol{\eta}) = I_{up}(\boldsymbol{\eta})$$

one can see that we selected β_{i_0} and β_{i_1} such that

$$F_{i_0}(\boldsymbol{\eta}) = \max_{i \in I_{low}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta})$$

$$F_{i_1}(\boldsymbol{\eta}) = \min_{i \in I_{up}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}).$$

Moreover, if our algorithm has not stopped yet, in other words if our stopping conditions have not been verified yet, then since our stopping conditions are weaker than those from Keerthi and Gilbert, the latter are not verified either. Hence one then have

$$\min_{i \in I_{up}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) < \max_{i \in I_{low}(\boldsymbol{\eta})} F_i(\boldsymbol{\eta}) - \tau$$

and thus

$$F_{i_1}(\boldsymbol{\eta}) < F_{i_0}(\boldsymbol{\eta}) - \tau$$

and (i_0, i_1) is a *violating pair* (in fact it is the *worst violating pair*).

Our algorithm is thus a special case of the general algorithm proposed by Keerthi and Gilbert. It verifies all their hypothesis, excepted that our stopping criterion is weaker. Thus, our algorithm converges.

3. Indeed, one can see that in our case, one has always $z_i > 0$ and thus $I_2(\boldsymbol{\eta})$ and $I_4(\boldsymbol{\eta})$ are empty sets.

7.4 Conclusion

In this paper, we have shown that *SVMTorch*, a decomposition algorithm proposed to solve large-scale regression problems using support vectors machines [6], converges when the size of the subproblem is set to 2 and no *shrinking* is done. We showed this convergence using a general theorem recently given by Keerthi and Gilbert [15]. Finally, even if there is no proven convergence when using *shrinking*, empirical experiments [6] showed that it does speed up a lot convergence times.

CHAPITRE 8

Expériences

8.1 Présentation du problème

Je vais maintenant vous présenter un exemple concret d'application des *support vector machines* dans un domaine où ils excellent : la reconnaissance de caractères.

J'ai utilisé pour la démonstration qui suit la base de données très connue *MNIST* proposée par Yann LeCun [18], d'AT&T. Elle est composée de 60000 exemples d'apprentissage, et 10000 exemples de test. Chaque exemple est l'image d'un chiffre écrit à la main, comme les exemples proposés à la FIG. 8.1.

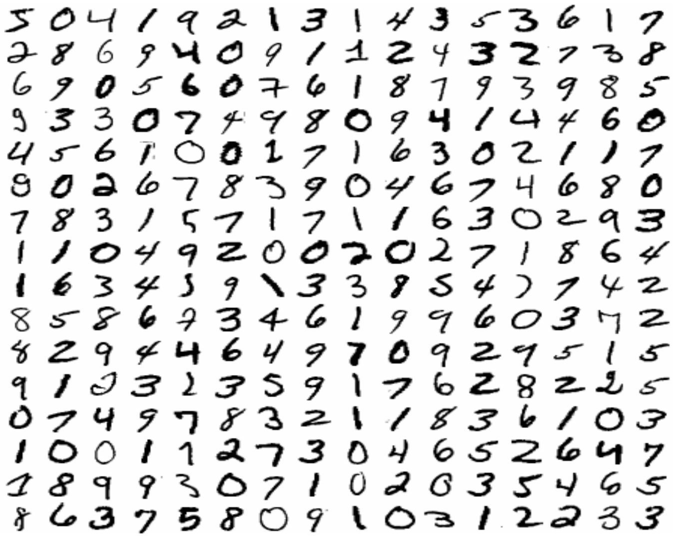

FIG. 8.1 – *Quelques exemples de chiffres parmi les 60000 exemples d'apprentissage*

Une image d'un chiffre est en fait composée de 28×28 pixels, chaque pixel étant représenté par un entier variant entre 0 et 255 correspondant à un niveau de gris (255 étant blanc et 0 étant noir).

Le problème est donc de « construire » une « machine » qui, quand on lui présente l'image d'un chiffre (représenté numériquement par un vecteur d'entiers[1] de taille $28 \times 28 = 784$) , donne en sortie un entier entre 0 et 9 correspondant à sa valeur.

1. Chaque entier étant donc compris entre 0 et 255 si vous me suivez...

Il s'agit donc de construire un classifieur qui sépare en dix ensembles adéquats l'ensemble des images de chiffres.

Malheureusement les SVM ont été conçus à la base pour faire de la classification à deux classes. Il existe des extensions au cas multiclasse [30], mais elles exigent de résoudre un problème de minimisation quadratique dont le nombre d'inconnues augmentent linéairement avec le nombre de classes. Comme la complexité de ce genre de problème est quadratique sur le nombres d'inconnues, ces méthodes sont invivables en pratique.

Cependant il existe dans la littérature plusieurs méthodes pour faire de la classification multiclasse à partir d'un (ou plusieurs) modèle de classifieur à deux classes. Citons par exemple la méthode du *one-per-class*, celle du *pairwise coupling* [22] ou encore celle appelée *pertinent dichotomies* [21].

Nous allons utiliser ici la méthode du *one-per-class* qui est une méthode simple qui donne d'assez bon résultats en général.

On construit donc dix modèles de SVM, chacun séparant (dans l'espace des images de chiffres) un chiffre en particulier de tous les autres[2].

Notons f_i la fonction apprise par le $i^{\text{ème}}$ SVM qui sépare le chiffre i de tous les autres. On suppose que si X est une image de i alors $f_i(X) > 0$, sinon $f_i(X) \leq 0$. Pour estimer la valeur d'une image Y de test, on présente cette image aux dix SVM et on prend comme réponse i_0 :

$$i_0 = \text{argmax}_i\, f_i(Y) \tag{8.1}$$

Ce choix est justifié par le fait que pour un point donné, plus la sortie (en valeur absolue) d'un SVM est élevée, plus ce point se situe loin de l'hyperplan séparateur, et donc plus nous sommes assurés d'une bonne classification.

Je vous propose ici d'illustrer les idées proposées par la théorie du premier chapitre en étudiant les erreurs de généralisation de chacun des modèles de SVM proposés ci-dessus.

8.2 La capacité en pratique

Intéressons nous dans un premier temps à l'évolution de l'erreur de généralisation en fonction du paramètre principal du SVM : le *kernel*. Comme il est dit par exemple dans [29], l'ensemble des SVM basés sur un *kernel* polynomial du type

$$k(x, x') = (ax \cdot x' + 1)^p$$

a une capacité de l'ordre de $O(d^p)$ où d est la dimension des vecteurs d'exemples. Ainsi, plus on augmente p, plus on augmente la capacité de l'ensemble des SVM correspondants. Et d'après ce qui a été suggéré au chapitre 2 sur la FIG. 2.2, l'erreur de généralisation d'un modèle décroît, passe par un minimum, puis croît lorsque l'on augmente la capacité du modèle considéré.

C'est exactement ce qui se passe avec chacun des dix SVM construits à partir des données *MNIST*, lorsque ceux-ci sont basés sur un *kernel* polynomial et que l'on augmente le degré du polynôme, comme le montre la FIG. 8.2. J'ai fait varier ici le degré du polynôme[3] entre 2 et 9, et construit les SVM sur des ensembles d'apprentissage de 5000 points[4]. L'erreur de généralisation a été estimée en comptant

2. Par exemple le premier SVM dira « oui c'est un 0 » ou « non ce n'est pas un 0 ».
3. Accessoirement j'ai pris (après plusieurs essais) un facteur de normalisation $a = 3.10^{-7}$.
4. J'ai diminué ici la taille de l'ensemble d'apprentissage pour que l'apprentissage soit plus rapide. Notez que la capacité optimale dépend de la taille de l'ensemble d'apprentissage, et que les résultats présentés ici n'ont pas pour but de choisir la capacité afin de résoudre le problème sur 60000 exemples : ils servent uniquement à illustrer la théorie.

le nombre d'erreurs de chaque modèle sur les 10000 points de test. Notez que les courbes d'erreur d'apprentissage n'ont pas été représentées, car dès $d = 2$, les SVM séparent parfaitement les 5000 points d'apprentissage.

Intéressons-nous maintenant à l'impact du nombre d'exemples d'apprentissage sur l'erreur de généralisation : si l'on fixe la capacité du SVM, *i.e.* si on fixe le degré du polynôme du *kernel* (ici $d = 1$) et que l'on augmente le nombre d'exemples (comme sur la FIG. 8.3 avec le cas particulier de la classe 9), vous remarquerez encore une fois que, comme ce qui a été dit au chapitre 2 sur la FIG. 2.1, l'erreur d'apprentissage augmente progressivement tandis que l'erreur de test diminue. Ces deux courbes semblent d'ailleurs vouloir se rejoindre à l'infini, ce qui est conforme à la théorie.

Les expériences sur la capacité nous montrent que l'on ne peut pas choisir n'importe quel *kernel* pour un problème donné : comme vous pouvez le voir sur la FIG. 8.2 dans le cas particulier de *MNIST* avec des *kernel* polynomiaux, le degré optimal du *kernel* dépend notamment de la classe étudiée.

De plus, nous avons vu sur la FIG. 8.3 l'importance du nombre d'exemples d'apprentissage : plus celui-ci est grand, plus on peut espérer une erreur de généralisation faible.

8.3 État de l'Art ?

La base de donnée *MNIST* a servi à tester beaucoup de modèles dans le domaine de l'*apprentissage statistique* [18]. En particulier les réseaux de neurones à deux couches classiques donnent des erreurs sur l'ensemble de test de l'ordre de 4% (lorsqu'on les entraîne sur les 60000 exemples d'entraînement, et si l'on ne fait aucun prétraitement sur les données). Il faut des réseaux de neurones *très* compliqués tels que *LeNet-4* de Yann LeCun *et al* [19, 20] (d'AT&T) pour arriver à une erreur de test de 1.1%, voire légèrement en dessous pour *LeNet-5*. Tandis que d'après Corrina Cortes [7] (aussi d'AT&T), en entraînant des SVM avec un *kernel* polynomial et en utilisant l'idée du *one-per-class*, on peut obtenir une erreur de test de 1.1%. Patrick Haffner (encore d'AT&T) m'a déclaré que ce résultat n'avait jamais été reproduit, mais que Burges et Shölkopf ont obtenu une erreur de 1.4% avec un *kernel* polynomial de degré 5 et que lui-même a obtenu une erreur de 1.36% avec un *kernel* gaussien [5]. Les SVM arrivent donc à résoudre extrêmement bien et de façon simple ce problème. J'ai moi même essayé d'entraîner les SVM sur *MNIST* avec un *kernel* gaussien, et j'ai fini par obtenir une erreur record de 1.27% avec la méthode (8.1). Je doute d'ailleurs que vous puissiez reconnaître sur la FIG. 8.4 tous les chiffres où le modèle a fait une erreur sur l'ensemble de test...

Pour conclure, notez que j'ai choisi (comme l'ont probablement fait les personnes citées précédemment) les paramètres d'apprentissage (en particulier σ) en regardant les résultats sur l'ensemble de test : cette performance de 1.27% est donc légèrement biaisée. Pour avoir une estimation correcte de l'erreur de généralisation il aurait fallu se servir uniquement des exemples d'apprentissage pour déterminer les paramètres optimaux. Mais de toutes façons, tous les résultats sur la base de données *MNIST* sont plus ou moins statistiquement identiques, car l'intervalle de confiance sur l'erreur de généralisation donné par l'inégalité de Hoeffding (2.7) au chapitre 2 est trop grand pour que l'on soit en mesure de se disputer des erreurs de l'ordre de 1%.

5. Ceci est une communication purement personnelle.

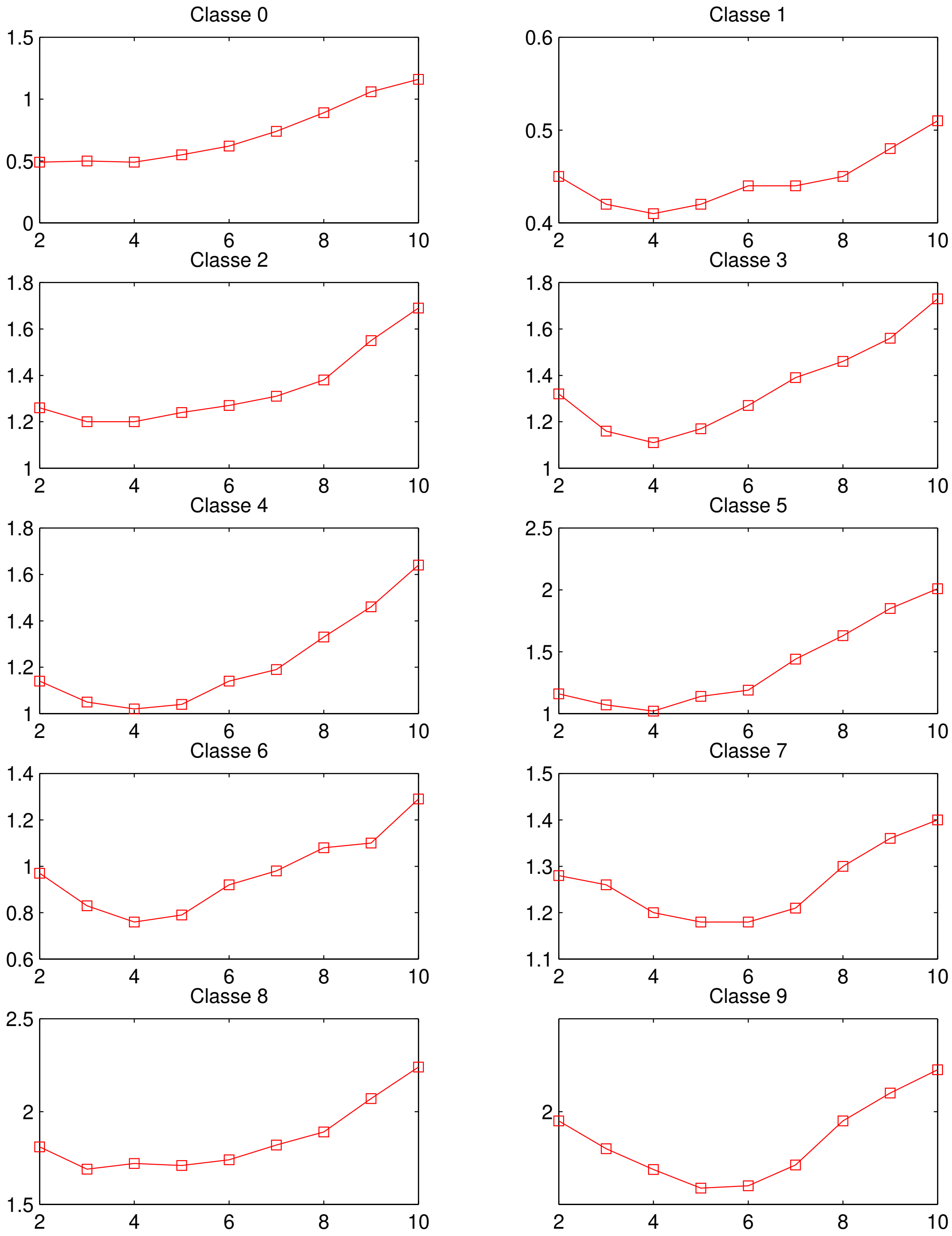

Fig. 8.2 – *Évolution de l'estimation du pourcentage d'erreur de généralisation en fonction du degré du polynôme du* kernel*, pour chaque SVM.*

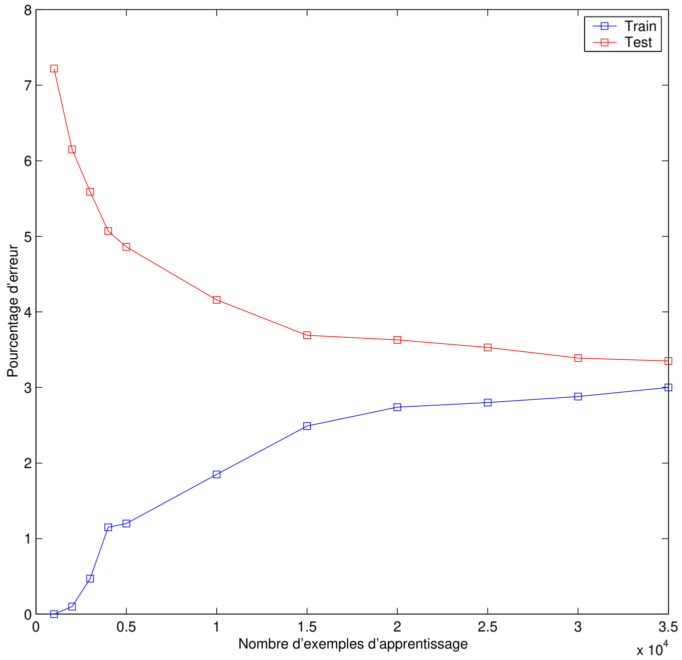

FIG. 8.3 – *Évolution des erreurs d'apprentissage et de généralisation, à capacité fixe, lorsqu'on augmente le nombre d'exemples d'apprentissage.*

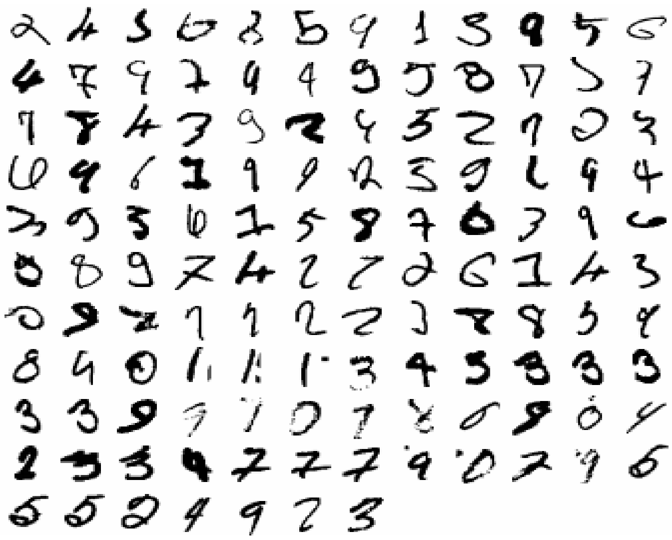

FIG. 8.4 – *Les* 127 *erreurs faites par le modèle de SVM*

CHAPITRE 9 Épilogue

Dans ce mémoire nous avons introduit la notion d'algorithme d'apprentissage, et nous en avons étudié un exemple dans les détails : les *support vector machines*. Nous avons aussi fourni un algorithme efficace pour résoudre le problème quadratique posé par ceux-ci en régression.

Il n'en reste pas moins que les *support vector machines* sont coûteux en temps d'apprentissage sur les *très* grandes bases de données. Il est encore difficilement envisageable d'entraîner un SVM sur un million d'exemples d'apprentissage[1]. On peut donc se demander à juste titre pourquoi les SVM ont eu tant de succès auprès de la communauté de *l'apprentissage statistique* ces dernières années. Pourquoi utiliser un SVM coûteux en calcul qui donnera un résultat à peine meilleur qu'un réseaux de neurone multicouche facile à entraîner, par exemple ?

Certains pensent que les SVM sont un phénomène de mode. Ils ont le vent en poupe actuellement principalement à cause du fait que c'est Vladimir Vapnik, légende vivante du domaine de la statistique, qui les a proposés. Si rien n'est fait pour améliorer le temps de calcul des SVM, peut-être tomberont-ils dans l'oubli. Si l'on arrive à entraîner un SVM sur des bases de données de plusieurs millions d'exemples en un temps raisonnable, on pourra alors se poser la question de l'utilité des réseaux de neurones.

En attendant, on nage dans le brouillard.

1. Des bases de données d'une telle taille sont de moins en moins rares : les problèmes en traitement de la parole ou en *data mining* en sont des exemples concrets.

ANNEXE A — Petite introdution à la programmation non-linéaire

Vous trouverez ici l'énoncé des théorèmes utilisés dans ce rapport, qui traitent de la programmation non-linéaire. Pour plus de détails vous pouvez vous reporter à [5, 11], qui sont les ouvrages plagiés ici.

La programmation non-linéaire s'intéresse à la recherche de u tel que

$$\left\{ \begin{array}{l} u \in U = \{v \in \boldsymbol{R}^n \ : \ \varphi_i(v) \leq 0,\, 1 \leq i \leq m\} \\ \\ \qquad\qquad J(u) = \inf_{v \in U} J(v) \end{array} \right. \tag{A.1}$$

Le théorème suivant établit des conditions nécéssaires et suffisantes de minimum en programmation *convexe.*

Théorème A.1 *Soit $J : \Omega \subset V \to \boldsymbol{R}$ une fonction définie sur un ouvert convexe Ω d'un espace de Hilbert V, et*

$$U = \{v \in \Omega \ : \ \varphi_i(v) \leq 0,\, 1 \leq i \leq m\}$$

une partie de Ω, les contraintes $\varphi_i : \Omega \subset V \to \boldsymbol{R}, 1 \leq i \leq m$ étant supposées convexes. Soit $u \in U$ un point en lequel les fonctions $\varphi_i, 1 \leq i \leq m$ et J sont dérivables.

- *Si la fonction J admet en u un minimum relatif par rapport à l'ensemble U, et si les contraintes sont qualifiées, alors il existe des nombres $\lambda_i(u)$,$1 \leq i \leq m$, tels que les relations de Kuhn Karush et Tucker (KKT) :*

$$\left\{ \begin{array}{l} J'(u) + \sum_{i=1}^m \lambda_i(u)\, \varphi_i'(u) = 0 \\ \lambda_i(u) \geq 0,\, 1 \leq i \leq m, \quad \sum_{i=1}^m \lambda_i(u)\, \varphi_i(u) = 0 \end{array} \right.$$

  soient satisfaites.
- *Réciproquement, si la fonction $J : U \to \boldsymbol{R}$ est convexe et s'il existe des nombres λ_i, $1 \leq i \leq m$, tels que les conditions de KKT soient satisfaites, alors la fonction J admet en u un minimum par rapport à l'ensemble U.*

Introduisons maintenant la notion de *point selle* :

Définition A.1 *Soient V et M deux ensembles quelconques et*

$$L : V \times M \to \boldsymbol{R}$$

une fonction. On dit que (u, λ) est point selle *de la fonction L si le point u est un minimum pour la fonction $v \in V \mapsto L(v, \lambda)$ et si le point λ est un maximum pour la fonction $\mu \in M \mapsto L(v, \mu)$. Autrement dit, si :*

$$\sup_{\mu \in M} L(u, \mu) = L(u, \lambda) = \sup_{v \in V} L(v, \lambda)$$

Introduisons aussi une notion générale de *Lagrangien* :

Définition A.2 *Le* Lagrangien *associé au problème (A.1) est la fonction* L définie *par*

$$L : \left\{ \begin{array}{ccc} V \times \boldsymbol{R}_+^m & \to & \boldsymbol{R} \\ (v, \mu) & \mapsto & J(v) + \sum_{i=1}^m \mu_i \varphi_i(v) \end{array} \right.$$

Et le théorème qui nous intéresse à ce sujet est le suivant, qui établit un lien entre la notion de point selle du Lagrangien L et la solution au problème (A.1).

Théorème A.2 *Avec les notations précédentes :*

- *Si* $(u, \lambda) \in V \times \boldsymbol{R}_+^m$ *est point selle du Lagrangien* L*, le point* u*, qui appartient à l'ensemble* U *est solution du problème (A.1).*
- *On suppose les fonctions* J *et* φ_i $1 \leq i \leq m$ *convexes, dérivables en un point* $u \in U$*, et les contraintes qualifiées. Alors si* u *est solution du problème (A.1), il existe au moins un vecteur* $\lambda \in \boldsymbol{R}_+^m$ *tel que le couple* $(u, \lambda) \in V \times \boldsymbol{R}_+^m$ *soit point selle du Lagrangien* L*.*

Notez qu'il existe d'autres théorèmes (dont la plupart sont cités par [5, 11]) pour établir facilement des solutions de (A.1), mais que malheureusement ils ne s'appliquent pas dans notre cas, les hypothèses de ces théorèmes étant trop restrictives.

Index

Bibliographie

[1] Isaac Asimov. *Les Robots.* Number 453. J'ai Lu, 1972.

[2] B. Boser, I. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifier. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. Pittsburgh ACM, 1992.

[3] C. Burges and D. Crisp. Uniqueness of the SVM solution. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12.* MIT Press, 1999.

[4] C. Chang, C. Hsu, and C. Lin. The analysis of decomposition methods for support vector machines. In *IJCAI'99, Workshop on Support Vector Machines*, 1999.

[5] P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

[6] R. Collobert and S. Bengio. Support vector machines for large-scale regression problems. IDIAP-RR 17, IDIAP, 2000. Available at `ftp://ftp.idiap.ch/pub/reports/2000/rr00-17.ps.gz`.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20, 1995.

[8] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machine and other kernel-based learning methods.* Cambridge University Press, 2000.

[9] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[10] G.W. Flake and S. Lawrence. Efficient SVM regression training with SMO. Submitted to Machine Learning. Available at `http://external.nj.nec.com/homepages/flake/smorch.ps`.

[11] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, 1987.

[12] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[13] A. Hoerl and R. Kennard. Ridge regression: biased estimation of non-orthogonal components. *Technometrics*, 12:55–67, 1970.

[14] Thorsten Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods.* MIT Press, 1999.

[15] S.S. Keerthi and E.G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. Technical Report CD-00-01, Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, 2000. Available at `http://guppy.mpe.nus.edu.sg/~mpessk/svm/conv_ml.ps.gz`.

[16] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. Technical Report CD-99-14, Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, 1999. To appear in Neural Computation. Available at `http://guppy.mpe.nus.edu.sg/~mpessk/smo_mod.ps.gz`.

[17] P. Laskov. An improved decomposition algorithm for regression support vector machines. In S.A. Solla, T.K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12.* MIT Press, 2000. Available at `http://www.cis.udel.edu/~laskov/publications/NIPS-99.ps.gz`.

[18] Y. LeCun. MNIST handwritten digit database. Available at http://www.research.att.com/~yann/ocr/mnist/.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufman, 1990. Available at http://www.research.att.com/~yann/publis/psgz/lecun-90c.ps.gz.

[20] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In F. Fogelman and P. Gallinari, editors, *International Conference on Artificial Neural Networks*, pages 53–60. EC2 & Cie, 1995. Available at http://www.research.att.com/~yann/publis/psgz/lecun-95b.ps.gz.

[21] Eddy Mayoraz and Miguel Moreira. On the decomposition of polychotomies into dichotomies. In *Proceedings of The Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997. IDIAP-RR 96-08, available at ftp://ftp.idiap.ch/pub/reports/1996/rr96-08.ps.gz.

[22] Miguel Moreira and Eddy Mayoraz. Improved pairwise coupling classification with correcting classifiers. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Artificial Intelligence*, pages 160–171. Springer, April 1998. IDIAP-RR 97-09, available at ftp://ftp.idiap.ch/pub/reports/1997/rr97-09.ps.gz.

[23] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks - ICANN'97*, pages 999–1004. Springer, 1997.

[24] Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, New York, 1997.

[25] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. MIT Press, 1999.

[26] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, and K.R.K. Murthy. Improvements to SMO algorithm for SVM regression. Technical Report CD-99-16, Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, 1999. To appear in IEEE Transaction on Neural Networks. Available at http://guppy.mpe.nus.edu.sg/~mpessk/smoreg_mod.shtml.

[27] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. Technical Report NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College,University of London, UK, 1998.

[28] Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer, second edition, 1995.

[29] Vladimir N. Vapnik. *Statistical learning theory.* John Wiley & Sons, 1998.

[30] J. Weston and C. Watkins. Multi-class support vector machines. Technical report, Royal Holloway, 1998. CSD-TR-98-04, available at http://www.cs.rhbnc.ac.uk/~jasonw.