
Boîte à outils SVM simple et rapide

Gaëlle Loosli^{*} – Stéphane Canu^{*} – Vishwanathan S.V.N.^{} – Alexander J. Smola^{**} – Manojit Chattopadhyay^{**}**

Laboratoire Perception, Systèmes, Information - FRE CNRS 2645
B.P. 08 - Place Emile Blondel
76131 - Mont Saint Aignan Cedex - France
gloosli@insa-rouen.fr

National ICT for Australia.
Canberra, ACT 0200 - Australia
vishy@axiom.anu.edu.au

RÉSUMÉ. Si les SVM (Support Vector Machines, ou Séparateurs à Vaste Marge) sont aujourd'hui reconnus comme l'une des meilleures méthodes d'apprentissage, ils restent considérés comme lents. Nous proposons ici une boîte à outils Matlab permettant d'utiliser simplement et rapidement les SVM grâce à une méthode de gradient projeté particulièrement bien adaptée au problème : SimpleSVM [VIS 03]. Nous avons choisi de coder cet algorithme dans l'environnement Matlab afin de profiter de sa convivialité tout en s'assurant une bonne efficacité, puisque Matlab utilise la bibliothèque ATLAS (Automatically Tuned Linear Algebra Software). La comparaison de notre solution avec l'état de l'art dans le domaine SMO (Sequential Minimal Optimization), montre qu'il s'agit là d'une solution dans certains cas plus rapide et d'une complexité moindre. Pour illustrer la simplicité et la rapidité de notre méthode, nous montrons enfin que sur la base de données MNIST, il a été possible d'obtenir des résultats satisfaisants en un temps relativement court (une heure et demie de calcul sur un PC sous linux pour construire 45 classifieurs binaires sur 60.000 exemples en dimension 576). Par ailleurs nous montrons comment étendre cet algorithme pour traiter des problèmes tels que les invariances, en découpant le problème tel qu'il soit possible d'obtenir une solution en seulement une phase : le Invariant SimpleSVM .

ABSTRACT. If SVM (Support Vector Machines) are now considered as one of the best learning methods, they are still considered as slow. Here we propose a Matlab toolbox that enables the usage of SVM in a fast and simple way. This is done thanks to the projected gradient method which is well adapted to the problem : SimpleSVM [VIS 03]. We chose to implement this algorithm with Matlab environment since it is user-friendly and efficient - it uses the ATLAS (Automatically Tuned Linear Algebra Software) library. The comparison to the state of the art in this field, SMO (Sequential Minimal Optimization) shows that in some cases, our solution is faster and less complex. In order to point out how fast and simple our method is, we give here results on the MNITS database. It was possible to compute a satisfying solution in a quite short time (one hour and a half on a PC with Linux distribution to compute 45 binary classifiers, with

60000 samples in dimension 576). Moreover, we show how this algorithm can be extended to problems like invariances, breaking down into small pieces the problem such that it is possible to get the solution running only once the Invariant SimpleSVM.

MOTS-CLÉS : Support Vector Machine, Séparateur à Vaste Marge, SVM, Apprentissage, Boîte à outils Matlab, Contraintes actives, Gradient projeté, Programmation quadratique sous contraintes de boîte, Invariances, USPS, MNIST.

KEYWORDS: Support Vector Machine, SVM, Machine Learning, Matlab toolbox, Active constraints, Projected Gradient, Quadratic Programming under box constraints, USPS, MNIST.

1. Introduction

La phase de mise en œuvre d'un algorithme est l'ultime test permettant d'éprouver ses qualités. Elle peut révéler un défaut majeur comme une certaine instabilité ayant échappé l'analyse théorique, souvent focalisée sur les aspects positifs de la méthode.

Nous allons présenter ici les différents aspects liés à la mise en œuvre d'un algorithme permettant de calculer la solution du problème des Séparateurs à Vaste Marge (SVM ou en anglais *Support Vector Machine*).

Cet algorithme du type « contraintes actives » ou « gradient projeté », publié sous le nom de *SimpleSVM* [VIS 03] [GIL 91], a passé l'épreuve avec succès. Il s'est révélé être non seulement très efficace en terme de temps de calcul et de précision mais aussi fertile en extensions potentielles.

Nous avons choisi comme première cible le langage Matlab car cet environnement est simple d'utilisation et particulièrement bien adapté au prototypage. L'ensemble des programmes dont nous allons parler est disponible en ligne¹. Le portage sur d'autres plates-formes comme R est en cours de réalisation.

Le succès des SVM a entraîné développement de nombreux algorithmes permettant leur mise en œuvre. La figure 10.1 de [SCH 02] donne un excellent point de vue sur ces différentes méthodes et leurs conditions d'utilisation. Parmi elles, l'algorithme *SMO* (*Sequential Minimal Optimization*) est souvent considéré comme le plus efficace. Il a semblé intéressant de comparer l'efficacité de notre algorithme avec celle de SMO. Afin de s'affranchir des contraintes liées au codage et à la machine nous avons choisi de montrer sur un problème donné comment les temps de calculs respectifs évoluaient en fonction de la taille du problème. Dans cet exemple l'évolution du temps de calcul en fonction du nombre de points de notre approche est $m^{1.2}$ contre $m^{2.5}$ pour SMO. Sa simplicité d'utilisation a été mise à l'épreuve d'un problème de taille significative : la reconnaissance de caractères manuscrits avec la base de données MNIST [LEC 98] qui contient 60.000 exemples en dimension 576 pour discriminer 10 classes. Sur cet exemple, il a été possible, en une heure et demie de temps de calcul sur un PC classique, d'obtenir des résultats raisonnables. Sur ce problème, la rapidité de la méthode ainsi que son caractère réentrant (c'est-à-dire qu'elle permet un démarrage à chaud et plus l'initialisation est proche de la solution plus la méthode converge rapidement) ont permis d'utiliser une technique de validation croisée pour identifier les hyper-paramètres du modèle. Ce caractère réentrant fait de *SimpleSVM* une méthode adaptée à l'utilisation en ligne.

En tirant parti de l'aspect itératif de la méthode, nous montrons comment traiter les invariances présentes dans les données au moment de la résolution du SVM. Contrairement aux méthodes usuelles, cette approche permet d'éviter une phase de pré-traitement des données ([Sch 98] et [CHA 02] pour les SVM invariants, respec-

1. <http://asi.insa-rouen.fr/~gloosli>

tivement linéaires et non linéaires) ainsi qu'une répétition de l'apprentissage sur une base de données étendue (*Virtual SVM*, [SCH 96]).

L'article est organisé de la manière suivante : après avoir rappelé la nature des SVM et les contraintes algorithmiques associées, nous présenterons l'algorithme *simpleSVM* en donnant une preuve de sa convergence. Nous présenterons ensuite le problème des invariances et l'algorithme *Invariant SimpleSVM* en soulignant sa similitude au *SimpleSVM*. Nous discuterons ensuite les différents détails liés à la programmation en Matlab. Enfin nous illustrerons à travers quelques expériences la rapidité et la simplicité de nos algorithmes.

Avant de rentrer dans le détail de la mise en œuvre proprement dite et pour en saisir les tenants et les aboutissants, il convient de commencer par poser le problème d'optimisation lié aux SVM et d'en étudier certaines caractéristiques.

2. Séparateurs à Vaste Marge

2.1. Minimisation quadratique sous contraintes

Il est possible, pour expliquer les SVM, de commencer en se donnant un noyau. Un noyau $k(x, y)$ est une fonction de deux variables, symétrique et positive (pour plus de détails se reporter à [ATT 99]). À partir du noyau est construit l'ensemble \mathcal{H}_0 de ses combinaisons linéaires finies :

$$\mathcal{H}_0 = \left\{ f : \mathbb{R}^d \rightarrow \mathbb{R} \mid \exists l \in \mathbb{N}, \mathbf{a} \in \mathbb{R}^l, \{\mathbf{x}_i\}_{i=1}^l \in \mathbb{R}^d ; f(\mathbf{x}) = \sum_{i=1}^l a_i k(\mathbf{x}, \mathbf{x}_i) \right\}$$

Pour tout couple de fonctions $f \in \mathcal{H}_0$ et $g \in \mathcal{H}_0$ s'écrivant $f(\mathbf{x}) = \sum_{i=1}^l f_i k(\mathbf{x}, \mathbf{x}_i)$ et $g(\mathbf{x}) = \sum_{i=1}^l g_i k(\mathbf{x}, \mathbf{x}'_i)$, la forme bilinéaire $\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^l \sum_{j=1}^l f_i g_j k(\mathbf{x}_i, \mathbf{x}'_j)$ définit un produit scalaire sur \mathcal{H}_0 . On note $\|f\|_{\mathcal{H}_0}^2 = \langle f, f \rangle_{\mathcal{H}_0}$ la norme associée. \mathcal{H}_0 muni du produit scalaire $\langle \cdot, \cdot \rangle_{\mathcal{H}_0}$ est un pré-hilbertien dans lequel la propriété de reproduction est vérifiée puisque $\langle f(\cdot), k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}_0} = f(\mathbf{x})$. Il reste à le compléter convenablement pour obtenir l'espace de Hilbert à noyau reproduisant $\mathcal{H} = \overline{\mathcal{H}_0}$ muni du même produit scalaire étendu.

Dans ce cadre, le séparateur à vaste marge du problème de discrimination à deux classes associé à l'échantillon $(\mathbf{x}_i, y_i), i \in [1, m]$ avec le codage $y_i \in \{-1, 1\}$, est la solution du problème d'optimisation sous contrainte suivant pour $f \in \mathcal{H}$:

$$\begin{cases} \min_{f, b, \boldsymbol{\xi}} & \frac{1}{2} \|f\|_{\mathcal{H}}^2 + C \sum_{i=1}^m \xi_i \\ \text{avec} & y_i (f(\mathbf{x}_i) + b) \geq 1 - \xi_i & i \in [1, m] \\ \text{et} & \xi_i \geq 0, & i \in [1, m] \end{cases} \quad (1)$$

où C est un scalaire permettant de régler la régularité de la fonction de décision, b un scalaire appelé le biais et les ξ_i sont des variables d'écart.

La solution de ce problème est aussi le point selle du lagrangien :

$$\mathcal{L}(f, b, \xi, \alpha, \beta) = \frac{1}{2} \|f\|_{\mathcal{H}}^2 + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i (y_i (f(\mathbf{x}_i) + b) - 1 + \xi_i) - \sum_{i=1}^m \beta_i \xi_i \quad (2)$$

avec $\alpha_i > 0$ et $\beta_i > 0$. On tire une partie des conditions de Kuhn et Tucker :

$$\begin{cases} \partial_f \mathcal{L}(f, b, \xi, \alpha, \beta) = 0 \\ \partial_b \mathcal{L}(f, b, \xi, \alpha, \beta) = 0 \\ \partial_{\xi} \mathcal{L}(f, b, \xi, \alpha, \beta) = 0 \end{cases} \Leftrightarrow \begin{cases} f(\mathbf{x}) - \sum_{i=1}^m \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) = 0 \\ \sum_{i=1}^m \alpha_i y_i = 0 \\ C - \alpha_i - \beta_i = 0 \quad i \in [1, m] \end{cases}$$

On en déduit que $f(\mathbf{x}) = \sum_{i=1}^m \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i)$. Cette relation permet de remplacer f dans le lagrangien pour aboutir à la formulation duale suivante, parfois appelée le dual de Wolfe :

$$\begin{cases} \max_{\alpha \in \mathbb{R}^m} & -\frac{1}{2} \alpha^\top G \alpha + \mathbf{e}^\top \alpha \\ \text{avec} & \alpha^\top \mathbf{y} = 0 \\ \text{et} & 0 \leq \alpha_i \leq C \quad i \in [1, m] \end{cases} \quad (3)$$

où G est la matrice d'influence de terme général $G_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ et $\mathbf{e} = [1, \dots, 1]^\top$. La solution du problème SVM sera alors donnée par la résolution d'un problème d'optimisation quadratique en dimension m sous contraintes de boîte.

2.2. Relations entre variables primales et duales

Dans le problème précédent, chaque inconnue α_i peut être interprétée comme l'influence de l'exemple (\mathbf{x}_i, y_i) dans la solution.

Du fait que seuls les points frontière sont importants pour la discrimination et qu'ils sont *a priori* peu nombreux, un grand nombre de ces coefficients α vont être nuls [STE 03].

Il sera alors pertinent de répartir les m inconnues du problème en trois groupes de points $[1, m] = I_s \cup I_c \cup I_0$, définis en fonction de la valeur du multiplicateur de Lagrange α associé :

– $[I_s]$ le groupe des points **supports** est celui des vecteurs supports candidats, c'est-à-dire pour lesquels $0 < \alpha_i < C$. Ces points sont à l'intérieur de la boîte et satisfont les contraintes. Il est aussi appelé l'ensemble de travail (*working set*), car il contient les variables sur lesquelles il faut « travailler » pour connaître leur valeur,

– $[I_c]$ le groupe des points **saturés**. Ces points sont sur le bord de la boîte. Si ces points sont trop proches de la classe opposée, voire complètement dans la classe opposée (erreur d'étiquetage par exemple), limiter leur contribution à la frontière de décision permet de régulariser la solution. Dans la solution finale ces points auront tous la valeur de leur α fixée à C : ils seront contraints et contribueront à la décision,

– $[I_0]$ le groupe des points **inactifs**. Ces points se situent également sur l'arête de la boîte. Dans ce cas, les vecteurs sont loin de la frontière entre les classes et ne contribuent en rien à la solution. Dans la solution finale ces points auront tous la valeur de leur α fixée à 0 : ils seront contraints.

Ces trois ensembles nous aident à reformuler le problème d'optimisation :

$$\left\{ \begin{array}{ll} \max_{\alpha \in \mathbb{R}^m} & -\frac{1}{2}\alpha^\top G\alpha + \mathbf{e}^\top \alpha \\ \text{avec} & \alpha^\top \mathbf{y} = 0 \\ \text{et} & 0 < \alpha_i < C \quad i \in I_s \\ \text{et} & \alpha_i = C \quad i \in I_c \\ \text{et} & \alpha_i = 0 \quad i \in I_0 \end{array} \right. \quad (4)$$

La relation entre les paramètres de la formulation duale que nous venons d'obtenir (problème 4) et les paramètres initiaux du problème (équation 1) est importante. Il est possible de la rendre explicite en passant une fois encore par l'écriture d'un lagrangien, cette fois celui du problème dual (problème 4) vu comme un problème de minimisation :

$$\mathcal{L}(\alpha, \lambda, \mu, \nu) = \frac{1}{2}\alpha^\top G\alpha - \mathbf{e}^\top \alpha + \lambda \alpha^\top \mathbf{y} - \nu^\top \alpha + \mu^\top (\alpha - C\mathbf{e}) \quad (5)$$

où les multiplicateurs de Lagrange α , μ et ν doivent être positifs. Ce lagrangien nous ramène au problème primal et peut être comparé à celui obtenu à partir du problème primal (équation 2) après avoir remplacé la variable f par α mais avant simplification, soit :

$$\mathcal{L}(\alpha, b, \xi, \beta) = \frac{1}{2}\alpha^\top G\alpha - \mathbf{e}^\top \alpha + b\alpha^\top \mathbf{y} + \xi^\top (\alpha + \beta - C\mathbf{e}) \quad (6)$$

Pour faire apparaître les équivalences entre paramètres, les trois cas possibles sont considérés (I_s , I_0 et I_c).

– $[0 < \alpha < C]$: dans ce cas la condition de Kuhn et Tucker stipulant l'annulation du gradient du lagrangien par rapport à α s'écrit : $G\alpha + \lambda\mathbf{y} - \mathbf{e} = 0$. Dans ce cas aussi la contrainte initiale est saturée, c'est-à-dire $y_i(f(\mathbf{x}_i) + b) = 1$ que l'on peut récrire en utilisant les mêmes notations que précédemment $G\alpha + b\mathbf{y} - \mathbf{e} = 0$. Le multiplicateur de Lagrange λ associé à la contrainte d'égalité est donc égal au biais $\lambda = b$.

– $[\alpha = C]$: dans ce cas $\xi \neq 0$, $\nu = 0$ et $\mu = -G\alpha - b\mathbf{y} + \mathbf{e} = \xi$. Si le multiplicateur de Lagrange est positif, la variable d'écart l'est aussi. Par conséquent $y_i(f(\mathbf{x}_i) + b) < 1$ doit être vérifié pour ce cas.

– $[\alpha = 0]$: dans ce cas $\xi = \mu = 0$ et $\nu = G\alpha + b\mathbf{y} - \mathbf{e}$. Si le multiplicateur de Lagrange est positif, alors $y_i(f(\mathbf{x}_i) + b) > 1$ doit être vérifié pour ce cas.

Ces relations sont résumées dans le tableau (1). Vérifier ces conditions d'optimalité revient à calculer $G\alpha + by - e$ puis à s'assurer que pour tous les points non supports ($\alpha = 0$) cette quantité est positive et que pour tous les points support saturés ($\alpha = C$) cette quantité est bien négative.

Ensemble	Contraintes initiales	Contraintes primales	Contraintes Duales
I_s	$y_i(f(\mathbf{x}_i) + b) = 1$	$0 < \alpha < C, \quad \xi = 0$	$\nu = 0, \quad \mu = 0$
I_c	$y_i(f(\mathbf{x}_i) + b) = 1 - \xi_i$	$\alpha = C, \quad \xi > 0$	$\nu = 0, \quad \mu > 0$
I_0	$y_i(f(\mathbf{x}_i) + b) > 1$	$\alpha = 0, \quad \xi = 0$	$\nu > 0, \quad \mu = 0$

Tableau 1. Situation des contraintes pour les trois types de variables.

Pour résoudre efficacement le problème, il est astucieux de prendre en compte la situation des points au regard des contraintes vérifiées ou non.

2.3. Les grandes lignes de la méthode proposée

L'objectif de tout algorithme SVM est double : il faut d'une part arriver à répartir l'ensemble d'apprentissage dans ces trois groupes, puis une fois la répartition connue, résoudre le problème. Il s'avère que cette seconde phase est relativement plus facile.

Supposons que l'on connaisse la répartition des points (I_s, I_c et I_0 donnés) : les contraintes d'inégalité ne sont plus nécessaires (elles sont contenues implicitement dans la définition des trois ensembles de points). Seules les valeurs des α_i pour $i \in I_s$ demeurent inconnues, car par définition $\alpha_i = C$ pour $i \in I_c$ et $\alpha_i = 0$ pour $i \in I_0$. La valeur des α_i pour $i \in I_s$ est ainsi donnée par la solution du problème d'optimisation suivant :

$$\left\{ \begin{array}{l} \max_{\alpha \in \mathbb{R}^{|I_s|}} \quad -\frac{1}{2}\alpha^\top G_s \alpha + \mathbf{e}_s^\top \alpha \\ \text{avec} \quad \alpha^\top \mathbf{y}_s + C \mathbf{e}_c^\top \mathbf{y}_c = 0 \end{array} \right. \quad (7)$$

avec $\mathbf{e}_s = \mathbf{e}(I_s) - CG(I_s, I_c)\mathbf{e}(I_c)$, $G_s = G(I_s, I_s)$, $\mathbf{y}_s = \mathbf{y}(I_s)$, $\mathbf{y}_c = \mathbf{y}(I_c)$ et \mathbf{e}_c un vecteur de un. Il est ici important de noter que la dimension de ce problème est égale au cardinal de l'ensemble I_s (qui peut être bien inférieur à m , la dimension initiale du problème). Les conditions de Kuhn et Tucker nous donnent le système à résoudre pour obtenir la valeur des coefficients α encore inconnus :

$$\begin{pmatrix} G_s & \mathbf{y}_s \\ \mathbf{y}_s^\top & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{e}_s \\ -C \mathbf{e}_c^\top \mathbf{y}_c \end{pmatrix} \quad (8)$$

Si la solution de ce système contient une composante violant les contraintes (une composante négative ou supérieure à C), elle indique que la répartition initiale des

points I_s , I_c et I_0 est fautive. Cette composante violant les contraintes doit être enlevée de I_s pour passer dans I_c ou I_0 .

Si maintenant toutes les composantes de la solution α vérifient les contraintes, il n'est pas encore certain que nous ayons trouvé l'optimum du problème initial. Il nous faut encore vérifier que les contraintes de positivité sont respectées pour les multiplicateurs de Lagrange associés (cf. paragraphe précédent). Alors, nous allons vérifier que pour tous les points de I_c , $G\alpha + by - e < 0$, et que pour tous les points de I_0 , $G\alpha + by - e > 0$. Si tel n'est pas le cas, le point violant les contraintes devra changer d'appartenance et quitter I_c ou I_0 pour passer dans I_s .

Nous venons là de donner le principe de notre algorithme. Il s'agit d'un algorithme itératif qui va, à chaque itération, ajouter ou enlever un seul point de l'ensemble I_s . Nous verrons qu'à chaque itération le coût décroît strictement, garantissant ainsi la convergence de la méthode. Enfin, puisqu'entre chaque itération les matrices G_s ne diffèrent que d'une ligne et d'une colonne, il est possible de calculer la nouvelle solution directement à partir de l'ancienne, réduisant ainsi la complexité de chaque itération de $\mathcal{O}(m^3)$ à $\mathcal{O}(m^2)$.

3. SimpleSVM - Méthode des contraintes actives

Dans cette partie nous allons introduire formellement l'algorithme *SimpleSVM* et détailler son fonctionnement. Cet algorithme met en œuvre la méthode des contraintes actives pour la résolution d'un problème de programmation quadratique sous contraintes avec des contraintes « de boîte ». Ce type de contrainte a un intérêt : il est facile de projeter dessus.

Une autre manière d'introduire l'algorithme est de le présenter comme une méthode de gradient projeté. Le principe consiste à partir de l'intérieur du domaine des contraintes admissibles et d'y rester en recherchant à chaque itération une direction de descente admissible qui nous amène sur la frontière du domaine. Une telle direction peut être obtenue en projetant le gradient sur cette frontière ([MIN 83] page 197). C'est cette projection qui va nous amener à faire passer un point de I_s vers I_0 ou I_c .

3.1. Activation d'une contrainte : projection

Il faut ici faire attention de ne pas mélanger contrainte active et point actif dans la solution. Si l'on active une contrainte, cela veut dire que l'on fixe α à l'une des bornes (0 ou C). Les points correspondant aux contraintes activées en 0 sont inactifs dans le classifieur (n'ont aucune influence).

Supposons que l'on connaisse un point admissible $\tilde{\alpha}_{\text{old}} = (\alpha_{\text{old}}, \lambda_{\text{old}})$. L'optimum non contraint $\tilde{\alpha}^*$ est donné par l'équation (8) :

$$\tilde{\alpha}^* = H^{-1}\tilde{\mathbf{e}} \quad \text{avec} \quad H = \begin{pmatrix} G_s & \mathbf{y}_s \\ \mathbf{y}_s^\top & 0 \end{pmatrix} \quad \text{et} \quad \tilde{\mathbf{e}} = \begin{pmatrix} \mathbf{e}_s \\ -C\mathbf{e}_c^\top \mathbf{y}_c \end{pmatrix} \quad (9)$$

On choisit la direction de descente $\mathbf{d} = \tilde{\alpha}^* - \tilde{\alpha}_{\text{old}}$, celle du gradient, qui garantit que le coût va décroître. Afin de rester dans le domaine admissible $0 \leq \alpha \leq C$, nous allons recenser toutes les composantes de α^* violant les contraintes et rechercher le pas de descente t^* minimisant le coût tout en nous maintenant dans le domaine admissible, c'est-à-dire :

$$t^* = \max_t (\alpha = \alpha_{\text{old}} + t\mathbf{d} \mid 0 \leq \alpha \leq C)$$

Pratiquement, le pas de descente est donné directement par la recherche du minimum suivant :

$$t^* = \min_t \left(-\frac{\alpha_{\text{old}}}{\mathbf{d}}, \frac{C - \alpha_{\text{old}}}{\mathbf{d}} \right)$$

Cela revient à projeter sur la frontière de la boîte et à éliminer de l'ensemble I_s la variable correspondant au plus grand pas possible. La nouvelle solution est alors :

$$\tilde{\alpha}_{\text{new}} = \tilde{\alpha}_{\text{old}} + t^*\mathbf{d} \quad (10)$$

Proposition 3.1 (Décroissance coût) *Le coût diminue strictement à chaque itération (étape 3.1 de l'algorithme 1).*

Démonstration. Cette propriété découle de la convexité du coût et nous donnons ici le détail. Considérons le coût :

$$q(\alpha) = \frac{1}{2}\alpha^\top H\alpha - \alpha^\top \tilde{\mathbf{e}}$$

la direction de descente \mathbf{d} est celle du gradient, elle vérifie :

$$\begin{aligned} \mathbf{d} &= \tilde{\alpha}^* - \tilde{\alpha}_{\text{old}} \\ &= H^{-1}\tilde{\mathbf{e}} - \tilde{\alpha}_{\text{old}} \end{aligned}$$

et donc $G\mathbf{d} = \mathbf{e} - G\tilde{\alpha}_{\text{old}}$. On a alors :

$$\begin{aligned} q(\tilde{\alpha}_{\text{new}}) - q(\tilde{\alpha}_{\text{old}}) &= q(\tilde{\alpha}_{\text{old}} + t\mathbf{d}) - q(\tilde{\alpha}_{\text{old}}) \\ &= \frac{1}{2}t^2\mathbf{d}^\top H\mathbf{d} + t(H\tilde{\alpha}_{\text{old}} - \tilde{\mathbf{e}})^\top \mathbf{d} \\ &= \frac{1}{2}t^2\mathbf{d}^\top H\mathbf{d} - t\mathbf{d}^\top H\mathbf{d} \\ &= \left(\frac{1}{2}t^2 - t\right)\mathbf{d}^\top H\mathbf{d} < 0 \end{aligned}$$

car d'une part le premier terme est négatif pour $0 < t < 2$ et par définition $0 < t^* \leq 1$; d'autre part le second est positif puisque la matrice G est définie positive et que le vecteur \mathbf{d} n'a pas toutes ses composantes nulles sauf la dernière. \square

3.2. Désactivation d'une contrainte

Désactiver une contrainte signifie permettre au α concerné de prendre une valeur libre. Cela revient à faire passer le vecteur correspondant dans le groupe I_s , des vecteurs supports candidats.

Cela s'avère indispensable lorsque le vecteur $\tilde{\alpha}^*$ calculé précédemment est admissible (lorsque $t = 1$). Dans ce cas, pour qu'il existe encore une direction de descente (pour que le coût continue à décroître) il faut trouver soit un point de I_0 pour lequel $y_i(f(\mathbf{x}_i) + b) - 1 < 0$, soit un point de I_c pour lequel $y_i(f(\mathbf{x}_i) + b) - 1 > 0$. Ces contraintes correspondent au fait que le gradient est convexe au point concerné et qu'il est possible de diminuer le coût en entrant dans le domaine admissible.

De manière intuitive, on vérifie dans le primal que la solution candidate classe bien les points du jeu d'apprentissage. Si ce n'est pas le cas, alors il faut réajuster la solution en ajoutant au groupe des actifs un des points mal « classé » (au sens de I_s , I_0 et I_c).

3.3. Convergence

L'algorithme 1 résume le principe des itérations de l'algorithme du *simpleSVM*. Nous allons reprendre la preuve de convergence de l'algorithme donnée dans [VIS 03].

Algorithme 1 : *simpleSVM*

```

1. ( $I_s, I_0, I_c$ ) ← initialiser
tant que minimumAtteint=FAUX
  2. ( $\alpha, \lambda$ ) ← résoudre système sans contrainte( $I_s$ )           équation (9)
  si  $\exists \alpha_i \leq 0$  ou  $\exists \alpha_i \geq C$ 
    3.1 projeter  $\alpha$  à l'intérieur du domaine admissible           équation (10)
    3.2 transférer le point associé de  $I_s$  vers  $I_0$  ou  $I_c$ 
  sinon
    4. chercher le meilleur point candidat  $x_{cand}$  dans  $I_c$  et  $I_0$ 
    si  $x_{cand}$  est trouvé
      5. transférer  $x_{cand}$  vers  $I_s$ 
    sinon
      6. minimumAtteint ← VRAI
    fin si
  fin si
fin tant que

```

L'algorithme s'arrête lorsque le vecteur $\tilde{\alpha}^*$ calculé précédemment est admissible et tous les points de I_0 et I_c vérifient leurs contraintes. Il n'existe alors plus de direction de descente admissible. Comme la solution dépend de la répartition des points dans les ensembles I_s , I_0 et I_c , qu'il n'existe qu'un nombre fini de points donc de

combinaisons, que le coût associé à l'algorithme décroît strictement à chaque itération et que l'optimum est atteint à chaque fois, l'algorithme ne peut pas boucler et va atteindre la solution globale en un temps fini.

4. Invariant SimpleSVM

Dans cette partie nous proposons une formalisation générale des invariances dans le cadre de la reconnaissance des formes. A cette fin nous allons présenter en premier lieu un exemple d'invariance commune en classification d'images, la rotation.

Définissons la plus petite unité de l'image (le pixel) comme l'application qui associe à tout couple de coordonnées un niveau de gris :

$$\begin{aligned} I : \mathbb{R}^2 &\rightarrow \mathcal{X} \in \mathbb{R} \\ (\ell, L) &\mapsto I(\ell, L) \end{aligned}$$

Une image est un vecteur de ces unités I défini $x \in \mathcal{X}^d$ avec $x_i = I(\ell_i, L_i), i \in [1, d]$. La rotation d'une image est une transformation qui agit sur les coordonnées :

$$\begin{aligned} R_\theta : \mathbb{R}^2 \times \Theta &\rightarrow \mathbb{R}^2 \\ (\ell, L), \theta &\mapsto R_\theta(\ell, L) \end{aligned}$$

Avec θ l'angle de la rotation, défini dans son domaine Θ (par exemple $[-\frac{\pi}{6}; \frac{\pi}{6}]$).

Considérant une discrimination binaire, nous définissons la fonction de décision $D(x)$:

$$\begin{aligned} D : \mathcal{X}^d &\rightarrow \{0, 1\} \\ x &\mapsto D(x) \end{aligned}$$

Si nous voulons que la fonction de décision soit invariante au regard de la rotation, nous imposons qu'elle donne la même décision pour une image et toutes ses rotations :

$$D(\{I(\ell_i, L_i), i = [1, d]\}) = D(\{I(R_\theta(\ell_i, L_i)), i = [1, d], \forall \theta \in \Theta\})$$

soit

$$D(x_0) = D(x_\theta), \forall \theta \in \Theta$$

La fonction de décision invariante devra donc être capable de classer correctement l'exemple ainsi que toutes les transformations générées par n'importe quel paramètre de transformation pris dans le domaine θ .

4.1. Les invariances en reconnaissance des formes

4.1.1. Définition

De manière plus générale nous avons besoin de définir une transformation et comment l'appliquer à tout type de formes (qui ne sont pas nécessairement images).

La forme x appartient à l'espace *de niveaux* \mathcal{L} (par exemple les niveaux de gris) et est dépendante de son support qui appartient à l'espace *support* \mathcal{S} (par exemple une grille de pixels). La transformation d'une forme est une application qui va d'une forme et des paramètres de transformation vers une forme :

$$T : \begin{array}{l} \mathcal{L} \times \Theta \rightarrow \mathcal{L} \\ x, \theta \mapsto T(x, \theta) \end{array}$$

Par ailleurs on pose $T(x, 0) = x$.

4.1.2. Types d'invariances

Nous passons ici en revue différents type d'invariances pour lesquelles on peut montrer qu'elles conviennent pour des manipulations algébriques [GRE 93]. D'une part la déformation de l'espace de niveaux agit sur les valeurs sans toucher au support.

Dans cette catégorie nous retrouvons les changements de contrastes, les manipulations de couleurs pour les images, etc. Pour un signal on trouvera les modifications de l'amplitude (effet d'échelle) ou de l'offset. D'autre part la déformation de l'espace support modifie la forme du support sans modifier les valeurs associées dans l'espace des niveaux.

Cette classe de transformations regroupe toutes les rotations, translations, changement d'échelle (par exemple le tempo d'une mélodie, ou le zoom sur une image) et distortions locales du support (effet de loupe sur une partie de l'image, ralentissement de tempo sur quelques mesures de la partition). Dans la formulation des SVM à suivre, nous ne ferons pas cette distinction

4.2. Formulation

Incorporer les invariances dans les SVM nécessite de distinguer les cas séparables (sans erreur) et non séparables (avec erreurs). Le cas séparable est solvable sans difficulté majeure alors que le non séparable entraîne des contraintes que l'on ne peut traiter directement.

4.2.1. Cas séparable

La formulation des SVM invariants dans le cas sans erreur est la suivante :

$$\left\{ \begin{array}{l} \min_{f,b} \frac{1}{2} \|f\|_{7t}^2 \\ \text{avec } y_i(f(T(\mathbf{x}_i, \theta)) + b) \geq 1 \quad i \in [1, m], \theta \in \Theta \end{array} \right. \quad (11)$$

La solution est comme précédemment obtenue en passant par le Lagrangien. La seule différence par rapport à la formulation habituelle est que le nombre de contraintes devient infini :

$$\left\{ \begin{array}{l} \max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{i,j=1}^m \iint_{\Theta} \alpha_i(\theta_1) \alpha_j(\theta_2) y_i y_j k(T(\mathbf{x}_i, \theta_1), T(\mathbf{x}_j, \theta_2)) d\theta_1 d\theta_2 + \sum_{i=1}^m \int_{\Theta} \alpha_i(\theta) d\theta \\ \text{avec} \quad \sum_{i=1}^m \int_{\Theta} \alpha_i(\theta) y_i d\theta = 0 \\ \text{et} \quad \alpha_i(\theta) \geq 0 \quad i \in [1, m], \theta \in \Theta \end{array} \right. \quad (12)$$

Il y a donc ainsi une infinité de multiplicateurs de Lagrange. Ceux-ci seront nuls pour tous les points sauf les vecteurs supports. Il est raisonnable de supposer que $\alpha_i(\theta)$ sera non nul seulement pour un nombre fini de paramètres θ . Cette hypothèse nous permet de simplifier de la manière suivante :

$$\left\{ \begin{array}{l} \max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{i,j,\theta_1,\theta_2} \alpha_i(\theta_1) \alpha_j(\theta_2) y_i y_j k(T(\mathbf{x}_i, \theta_1), T(\mathbf{x}_j, \theta_2)) + \sum_{i,\theta} \alpha_i(\theta) \\ \text{avec} \quad \sum_{i,\theta} \alpha_i(\theta) y_i = 0 \\ \text{et} \quad \alpha_i(\theta) \geq 0 \quad i \in [1, m], \theta \in \Theta \end{array} \right. \quad (13)$$

$$\left\{ \begin{array}{l} \max_{\boldsymbol{\gamma} \in \mathbf{R}^{m \times p}} -\frac{1}{2} \boldsymbol{\gamma}^T G \boldsymbol{\gamma} + \mathbf{e}^T \boldsymbol{\gamma} \\ \text{avec} \quad \boldsymbol{\gamma}^T \mathbf{y} = 0 \\ \text{et} \quad \gamma_i \geq 0 \quad i \in [1, mp] \end{array} \right. \quad (14)$$

où G est la matrice composée des blocs $G_{IJ} = y_i y_j K^{ij}$ avec $K_{kl}^{ij} = k(T(x_i, \theta_k), T(x_j, \theta_l))$, $\boldsymbol{\gamma} = [\alpha_1(\theta); \alpha_2(\theta); \dots; \alpha_m(\theta)]$ et p est la taille de Θ .

4.2.2. Cas non séparable

Pour les SVM *soft-margin*, utilisés dans les cas non séparables, nous faisons face à un autre problème.

$$\left\{ \begin{array}{l} \min_{f,b,\boldsymbol{\xi}} \frac{1}{2} \|f\|_{\mathcal{H}}^2 + C \sum_{i=1}^m \xi_i \\ \text{avec} \quad y_i (f(T(\mathbf{x}_i, \theta)) + b) \geq 1 - \xi_i \quad i \in [1, m], \theta \in \Theta, \\ \text{et} \quad \xi_i \geq 0, \quad i \in [1, m] \end{array} \right. \quad (15)$$

Ajouter les termes ξ_i permet de tolérer des erreurs et donne le système suivant :

$$\left\{ \begin{array}{l} \max_{\alpha} -\frac{1}{2} \sum_{i,j=1}^m \iint_{\Theta} \alpha_i(\theta_1) \alpha_j(\theta_2) y_i y_j k(T(\mathbf{x}_i, \theta_1), T(\mathbf{x}_j, \theta_2)) d\theta_1 d\theta_2 + \sum_{i=1}^m \int_{\Theta} \alpha_i(\theta) d\theta \\ \text{avec } \sum_{i=1}^m \int_{\Theta} \alpha_i(\theta) y_i d\theta = 0 \\ \text{et } \int_{\Theta} \alpha_i(\theta) d\theta \leq C \quad i \in [1, m] \\ \text{et } \alpha_i(\theta) \geq 0 \quad i \in [1, m], \theta \in \Theta \end{array} \right. \quad (16)$$

ce qui est similaire au cas séparable. Toutefois nous ne pouvons plus faire la supposition du nombre fini de multiplicateurs de Lagrange non-nuls. En effet si la trajectoire d'une transformation traverse les marge, cela donne une infinité de multiplicateurs de Lagrange bornés par $C : \int_{\Theta} \alpha_i(\theta) d\theta = C$. La résolution de ce problème implique de choisir la représentation des transformations. La solution consiste à discrétiser l'espace des paramètres de transformation Θ . On notera l'espace discrétisé $\Theta_{discret}$.

4.3. Formulation unificatrice

On identifie trois courants principaux dans les algorithmes alliant SVM et invariances, à savoir ceux fondés sur une extension artificielle du jeu de données, ceux qui modifient la fonction de coût de manière à y incorporer les invariances (cela implique un changement de la mesure des distances) et enfin ceux utilisant des approximations polynomiales pour représenter les trajectoires des transformations et classer celles-ci directement. La force de *Invariant SimpleSVM* est de contenir par définition toutes ces approches. En effet nous faisons très peu de suppositions sur la nature des transformations et en particulier nous ne faisons aucune approximation à priori dans le cas séparable.

4.3.1. Élargissement artificiel du jeu de données

Pour apprendre les invariances il est intuitif de chercher à les représenter dans la base d'apprentissage. Cette opération peut être menée avant l'utilisation d'un algorithme de discrimination en pré-calculant les transformations des exemples. Le jeu de données élargi (exemples réels et virtuels) contient ainsi l'information à priori sur les transformations. En dépit de sa simplicité, cette méthode présente l'inconvénient de devoir stocker un grand nombre de données et d'augmenter significativement la taille du problème à résoudre.

Virtual-SVM : Cette idée a été appliquée dans [SCH 96] avec *Virtual-SVM*. Les auteurs proposent un moyen de réduire la taille du problème qui tient compte du fait que toute l'information utile à la classification dans un SVM est contenue dans les vecteurs supports. L'idée est de ne générer les points virtuels que pour les vecteurs supports en supposant que les autres points sont trop éloignés des frontières pour que leurs transformations aient une influence sur la solution. Cette méthode requiert donc

trois phases : la sélection des vecteurs supports, la création des points virtuels et à nouveau la sélection des nouveaux vecteurs supports.

Le *Invariant SimpleSVM* réalise la même tâche si la transformation $T(x, \theta)$ est approchée par un nombre fini de points en fixant un nombre fini de valeurs pour $\theta \in \Theta_{discret}$

4.3.2. Adaptation de la mesure des distances

Présenté en 1993 dans [SIM 93] et appelé *tangent distance*, la motivation est de trouver une meilleure mesure que la distance euclidienne pour répondre au problème posé par les invariances. En effet, une légère translation dans une image engendre une grande différence avec une distance euclidienne sans que cela ait un réel sens pour la discrimination. L'algorithme *Tangent-Prop* a été proposé dans [SIM 96] et consiste à utiliser la distance tangente dans les réseaux de neurones. Pour ce qui est des SVM nous décrivons brièvement deux exemples d'application de cette distance.

Invariant SVM : L'idée est d'associer chaque vecteur avec un ou plusieurs vecteurs tangents et d'incorporer les invariances dans la fonction de coût. La première formulation a été pour les SVM linéaires dans [Sch 98] et a été étendue aux SVM non linéaires dans [CHA 02]. La fonction de coût à minimiser devient dans le cas linéaire :

$$\left\{ \begin{array}{l} \min_{f,b} (1 - \gamma) \|f\|_{\mathcal{H}}^2 + \gamma \sum_{i=1}^m \|f(dx_i)\|^2 \\ \text{avec} \quad y_i(f(x_i) + b) \geq 1 \quad i \in [1, m] \end{array} \right. \quad (17)$$

où γ est le paramètre de compromis ($\gamma = 0$ donne un SVM standard et $\gamma \rightarrow 0$ force l'hyperplan à être pratiquement orthogonal aux directions tangentes) et les dx_i sont les vecteurs tangents.

L'optimisation de cette fonction de coût revient à utiliser un SVM classique après une phase de pré-traitement où les données sont projetées linéairement. On obtient des résultats similaires dans le cas non linéaire avec une phase de pré-traitement consistant en une projection non linéaire.

Tangent Distance et Tangent Vector Kernels : En continuant dans l'idée d'utiliser la distance tangente (*TD-measure*) les noyaux intégrant les invariances ont été développés. Dans [HAA 02] les auteurs proposent les *Tangent Distance Kernels*. Brièvement, ils définissent des distances telles que *mean-TD* et *midpoint-TD* dérivées de la distance tangente et les substituent dans des noyaux classiques. Dans [POZ 03] les auteurs utilisent un noyau qui mesure la distance entre des trajectoires plutôt qu'entre des points.

Le *Invariant SimpleSVM* est similaire à ces méthodes si on approche les transformations par des polynômes du premier degré ($T(x, \theta) \simeq x + \nabla_{\theta} T(x, 0)\theta$), $\theta \in \Theta_{discret}$.

4.3.3. Approximation Polynomiale

Semi-definite Programming Machines : Avec les SDPM [GRA 04], les auteurs essaient d'apprendre les trajectoires des transformations différentiables points réels. Ils montrent que le problème peut être résolu pour des transformations qui peuvent être représentées ou approchées par des polynômes. Leur approche est fondée sur le théorème de Nesterov et ils formulent le problème d'apprentissage comme un problème de programmation semi-définie (SDP, semi-definite programming) sous contrainte polynomiale. Toutefois leur approche devient très vite trop complexe car elle requiert de résoudre des problèmes SDP de grande taille, ce qu'on ne sait pas très bien faire aujourd'hui.

Le *Invariant SimpleSVM* contient également cette approche si l'on approche les transformations par des polynômes de second degré ($T(x, \theta) \simeq x + \nabla_{\theta} T(x, 0)\theta + \frac{1}{2}\theta^{\top} H_{\theta}\theta$), $\theta \in \Theta_{discr\acute{e}t}$. Dans le cas séparable, nous faisons moins d'hypothèses et nous pouvons résoudre directement le problème. En revanche nous sommes pénalisés par le besoin de discrétiser l'espace des paramètres dans le cas non séparable. En effet, malgré la complexité de résolution, les SDPM travaillent dans l'espace original Θ .

4.3.4. Invariant SimpleSVM

Nous présentons ici l'implémentation de *Invariant SimpleSVM* qui correspond à l'ajout de points virtuels (l'équivalent du *Virtual SVM*). Comparé à *SimpleSVM* seule la phase 4 de l'algorithme 1 change. Au moment où dans *SimpleSVM* nous sélectionnons les points qui violent le plus les contraintes dans le dual (c'est à dire le point le plus mal classé dans le primal), *Invariant SimpleSVM* sélectionne la transformation de ce point qui viole encore plus les contraintes (cela peut être lui-même). Nous proposons ici plusieurs heuristiques pour le cas où l'espace des paramètres Θ est discrétisé :

- recherche complète : à chaque itération on ne regarde qu'un seul point et toutes ses transformations et on vérifie qu'aucun ne viole les contraintes,
- recherche incomplète : à chaque itération on regarde un groupe de p point et si l'un d'eux viole les contraintes alors on regarde ses transformations,
- recherche aléatoire : peut être appliquée dans les deux cas précédents. Plutôt que de considérer toutes les transformations d'un point, on en regarde quelques unes choisies au hasard. Cette heuristique diminue considérablement le temps de calcul mais ne garanti plus l'obtention de la meilleure solution.

5. Une boîte à outils en Matlab

Nous avons mis en œuvre les algorithmes présentés précédemment sous la forme d'un ensemble de programmes constituant une boîte à outils Matlab². Nous allons maintenant décrire cet ensemble de programmes. Les deux fonctions les plus importantes sont `svm <- TrainSVM(données, noyau, paramètres, modèle)` et `y`

2. Disponible sur <http://asi.insa-rouen.fr/~gloosli/>

<- `TestSVM(svm)`. Elles permettent respectivement d'apprendre à partir d'un échantillon contenu dans `donnees` et de calculer `y`. Le paramètre d'entrée `modele` est facultatif mais il est très utile, car il permet d'utiliser le caractère réentrant de l'algorithme en initialisant notre algorithme avec un point proche de la solution.

5.1. Initialisation

Le principe général de l'algorithme est par défaut de commencer par très peu de points. Ainsi, les premières itérations sont très rapides. La stratégie la plus simple pour initialiser le processus est le tirage aléatoire d'un point par classe. Elle est utilisée par défaut. Il existe d'autres critères pour choisir un point par classe. On peut par exemple les choisir de sorte que la distance qui les sépare soit la plus courte possible. On est ainsi sûr de commencer avec des points qui seront vecteurs supports [ROO 00].

On peut aussi procéder en projetant la solution calculée sans contraintes sur un sous-ensemble des points, à l'intérieur du domaine admissible. On constitue ainsi trois groupes de points (saturés à C , actifs et inactifs). On met dans le groupe des saturés tous les points tels que $\alpha \geq C$, dans le groupe des inactifs tous les points tels que $\alpha \leq 0$ et les derniers comme solution candidate dans le groupe des supports.

5.2. Résolution du système linéaire

Le cœur de l'algorithme, et aussi la partie la plus gourmande en temps de calcul est la résolution du système de la forme :

$$\begin{pmatrix} G & \mathbf{y} \\ \mathbf{y}^\top & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{e} \\ f \end{pmatrix}. \quad (18)$$

De façon à pouvoir garder le bénéfice de la positivité de la matrice G , le système est reformulé en découplant les variables. En définissant $M = G^{-1}\mathbf{y}$ et $N = G^{-1}\mathbf{e}$, $\boldsymbol{\alpha}$ et λ sont donnés par :

$$\begin{cases} \lambda = \frac{\mathbf{y}^\top N - f}{\mathbf{y}^\top M} \\ \boldsymbol{\alpha} = N - \lambda M \end{cases} \quad (19)$$

Deux stratégies différentes peuvent être utilisées pour résoudre le système linéaire de type $Gx = b$. Puisque nous savons la matrice G définie positive, la factorisation de Cholesky est la plus efficace. Mais dans le cas où la matrice G est singulière il est préférable d'utiliser la factorisation QR.

En pratique pour résoudre notre problème, voici comment utiliser la décomposition de Cholesky :

```
U = chol(G);
```

```

M = U\U'\y;           % ' désignant la transposée
N = U\U'\e; % et \ la "division matricielle" de Matlab
lambda=(y'*N-f)/(y'*M);
alpha=N-lambda*M;

```

La factorisation QR s'écrit $G = QR$ avec R une matrice triangulaire supérieure et Q une matrice orthogonale ($Q^T Q = I$). Nous utilisons cette décomposition de manière classique. Elle s'écrit en Matlab :

```

[Q,R] = qr(G);
M=R\Q'*y;
N=R\Q'*e;

```

5.3. Résolution Dynamique

Sachant que nous avons besoin de refaire ce calcul à chaque itération de l'algorithme avec seulement un point de différence par rapport à l'itération précédente, il est intéressant de regarder les méthodes incrémentales dont la complexité est $\mathcal{O}(n^2)$ plutôt que les méthodes directes en $\mathcal{O}(n^3)$. Les deux algorithmes de factorisation (Cholesky et QR) peuvent être traités de la sorte. Il est aussi possible de calculer incrémentalement G^{-1} l'inverse de la matrice G , en utilisant la formule de Sherman Morrison. Cette dernière solution s'avère pour l'instant la plus efficace.

5.3.1. Formulation de l'évolution de la matrice

Les formules précédentes s'appliquent lorsque la matrice B du nouveau système à résoudre s'écrit sous la forme $B = A + \mathbf{u}\mathbf{v}^T$, où A est la matrice d'un système dont on connaît solution avec \mathbf{u} et \mathbf{v} deux vecteurs donnés. Dans notre cas, lorsque la taille de la matrice augmente il faut écrire :

$$G^{new} = \begin{pmatrix} & & 0 \\ & G^{old} & \vdots \\ & & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} g_1^{new} \\ \vdots \\ g_n^{new} \\ -1 \end{pmatrix}^T + \begin{pmatrix} g_1^{new} \\ \vdots \\ g_n^{new} \\ g_{n+1}^{new} \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}^T$$

Pour diminuer la taille de la matrice on reprendra le même principe.

5.3.2. Factorisation incrémentale

Matlab propose une implémentation de la mise à jour dynamique de la factorisation de Cholesky. Elle n'est cependant pas proposée dans notre boîte à outils car elle s'est avérée peu efficace.

La décomposition QR a elle aussi sa version incrémentale. Il existe même deux méthodes pour arriver à mettre à jour les matrices Q et R . Matlab propose les fonctions

QRinsert et *QRdelete* qui permettent la mise à jour de la décomposition quand on ajoute ou supprime des lignes ou colonnes, ainsi que la méthode *QRupdate*.

5.3.3. Sherman Morrison

La méthode de Sherman Morrison permet de calculer incrémentalement l'inverse d'une matrice. La boîte à outils utilise la formule de Sherman Morrison sur des matrices symétriques de la façon suivante :

```
function [Hi] = sherman_morrison(Hi,ind,u)
v = [zeros(1,ind-1) 1 zeros(1,n-ind)];
if Hi(ind,ind)==0 % Hi matrice inversée
    Hi(ind,ind)=1; % à l'étape précédente
    u(ind) = (u(ind)-1)/2;
else
    u(ind) = u(ind)/2; % u contient les valeurs à ajouter
end
B = (Hi - ((Hi*u)*(v*Hi))/(1+v*Hi*u));
Hi = (B - ((B*v*t)*(u*t*B))/(1+u*t*B*v*t));
```

La complexité de cette méthode est $3n^2$.

5.4. Conseil sur le choix de la méthode

Dans la boîte à outils que nous proposons, le choix de la méthode de résolution du système linéaire est un paramètre laissé à la préférence de l'utilisateur. De manière générale nous pouvons toutefois donner les cas d'utilisation majeurs :

- base de donnée de petite taille et noyau classique (semi-défini positif) : Cholesky est alors un bon choix pour sa stabilité numérique et sa rapidité en Matlab. Dans la version actuelle de la boîte à outils, cette version est la seule à utiliser du code compilé et donc plus rapide,
- base de donnée de grande dimension et noyau classique : Sherman Morrison est alors la méthode la plus adaptée car c'est une version incrémentale,
- noyau non semi-défini positif : QR est alors la seule méthode à utiliser quelles que soient les données, car elle ne requiert aucun a priori sur la matrice à décomposer. En revanche cette méthode est plus lente (de l'ordre de $26n^2$ contre $\frac{1}{3}n^2$ pour Cholesky en version incrémentale.)

5.5. Ajout d'un point dans la solution

Plusieurs stratégies peuvent être adoptées lors de la phase d'ajout d'un point dans le groupe des supports. La première, la plus proche de la formulation mathématique, consiste à prendre tous les points des groupes saturés et inactifs et vérifier que les

contraintes sont bien satisfaites pour tous. Si l'on travaille sur une grande base de données, on réalisera qu'aussi bien en mémoire qu'en calculs, cette opération demande beaucoup de ressources. Notre idée est de segmenter cet ensemble de points et de chercher un point à ajouter sur des sous-groupes de p points. S'il n'y a aucun point à ajouter dans le premier groupe, on regarde dans le suivant jusqu'à trouver un point. Si après le passage en revue de tous les points on n'a rien trouvé, alors nous avons la solution finale. Au contraire dès que l'on trouve un point, on le rajoute sans regarder plus loin et on revient à la première étape avec notre nouveau jeu d'actifs. Plus le pas de segmentation p est petit, plus le calcul de vérification de l'optimalité est rapide. En revanche plus le pas est petit, plus on choisit de points qui seront rejetés par la suite car on ne prend pas à chaque itération le pire de tous, mais le pire de quelques uns. On se retrouve donc à devoir choisir un compromis entre le nombre d'itérations et la taille du calcul à chaque itération. En pratique nous avons constaté qu'un pas de l'ordre d'une centaine de points est en général un bon compromis.

5.6. Critère d'arrêt

Nous avons vu que l'algorithme converge (3.3), mais pour des raisons de précision machine nous définissons une borne d'erreur ϵ de l'ordre de 10^{-5} . L'algorithme est alors arrêté quand chacun des ν_i et μ_i est positif ou nul à ϵ près. Nous admettons par là que l'algorithme a convergé lorsque chacun des points contribue pour moins de ϵ au saut de dualité [SCH 02].

6. Expérimentations

Dans cette partie nous allons donner les résultats de différentes expériences visant à montrer les avantages de *SimpleSVM* et *Invariant SimpleSVM* en pratique. Pour des problèmes de taille réelle nous avons besoin d'outils rapides et dont la complexité permette d'utiliser des bases de données de grande dimension tout en restant dans des temps raisonnables. La première étude exposée ici est une étude pratique de complexité par rapport à SMO. La deuxième étude est la mise en œuvre de *SimpleSVM* sur la base de reconnaissance de caractère MNIST qui contient 60000 exemples de dix classes en dimension 576. La dernière étude compare les performances de *Invariant SimpleSVM* aux résultats connus sur la base de caractères USPS (9298 images en dimension 256). La machine utilisée pour mener ces épreuves est un PC sous linux (pentium 4 3GHz avec 1Go de RAM).

6.1. Étude de complexité et comparaison avec SMO

Le but de cette expérience est d'avoir un aperçu concret du comportement de *SimpleSVM* indépendamment la machine sur lequel il est mis en œuvre. notre démarche consiste à choisir un problème et de le résoudre pour différentes tailles. On peut ainsi

suivre l'évolution du temps d'exécution. La pente de la droite ainsi obtenue en coordonnées logarithmiques, nous donne une estimation de la complexité de l'algorithme. Pour comparer les performances et la complexité de notre solution, nous avons choisi de faire la même expérience avec un solveur reconnu comme étant parmi les plus rapide, SMO. SMO est une méthode proposée par [PLA 99]. Elle consiste à décomposer le problème en sous-problèmes de deux points et à optimiser ainsi les α deux à deux. De ce fait il n'y a plus de problème quadratique de grande taille à résoudre. Le choix des couples à optimiser est fait selon différentes heuristiques.

6.1.1. Cadre expérimental

Cette étude consiste à comparer l'évolution des temps de calculs en fonction de la taille du jeu d'apprentissage. L'expérience est effectuée sur des données en damier (problème des *checkers*, voir figure 1).

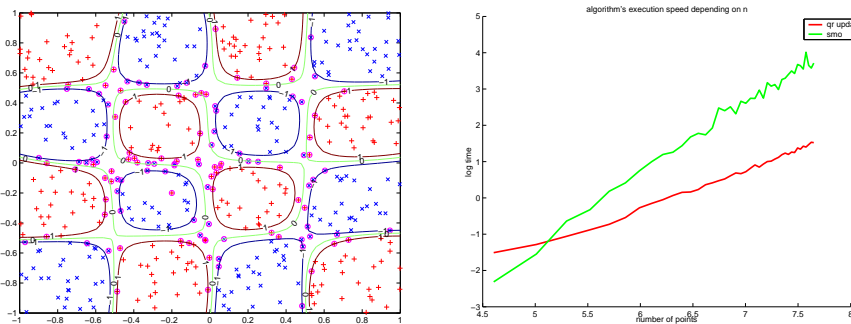


Figure 1. A gauche : forme des données utilisées pour comparer la complexité de SMO et SimpleSVM en fonction de la taille du jeu d'apprentissage. A droite : courbe de complexité de SMO et SimpleSVM sur le problème du damier. On représente ici le temps d'exécution (en log) en fonction du nombre de points en apprentissage (en log également). Les test sont effectués pour une taille d'apprentissage allant de 100 à 3000 points. Au delà de 200 points, SimpleSVM est plus rapide que SMO, avec une complexité de l'ordre de $n^{1,2}$ contre $n^{2,5}$ pour SMO.

Le code utilisé pour SMO est celui de la *toolbox* Matlab [CAW 00]. Le noyau et les paramètres sont identiques :

- Noyau Gaussien
- $\sigma = 0,25$
- $C = 500$

La méthode de résolution est *QR* dans sa version incrémentale (*QRupdate*).

6.1.2. Résultats et Discussion

La figure 1 montre que sur ce problème *SimpleSVM* est beaucoup plus efficace que SMO quand la taille du jeu d'apprentissage devient grand. On remarque par ailleurs que sur des petits jeux de données, SMO est plus rapide.

Sur ce problème concret nous voyons l'impact de la stratégie de *SimpleSVM* qui ne prend pas en compte les points non vecteurs supports lors de la résolution du problème. En effet en augmentant le nombre de points dans le damier, on ajoute très peu de points sur les frontières et beaucoup à l'intérieur des cases. Alors que SMO doit comparer de plus en plus de couples, *SimpleSVM* ne se sert des points non vecteurs supports qu'au moment de l'ajout d'un nouveau point.

Il faut noter ici que si l'on se trouve face à un problème où le nombre de vecteurs supports augmente avec la taille de la base d'apprentissage, alors *SimpleSVM* sera pénalisé. La complexité de *SimpleSVM* dépend plus du nombre de vecteurs supports que du nombre de points d'apprentissage. Ceint fera l'objet d'une étude expérimentale plus poussée dans de prochains travaux.

6.2. MNIST

Nous nous intéressons maintenant à l'utilisation de *SimpleSVM* dans le cadre de problèmes de taille réelle. Pour cela nous l'avons testé sur le problème de reconnaissance de caractères manuscrits MNIST. Ce jeu de données est constitué de 60000 images pour l'apprentissage et 10000 pour le test. Les images représentent les chiffres de 0 à 9 issus de codes postaux. Chaque image à une taille de 28×28 pixels dont la valeur est comprise entre 0 et 256.

6.2.1. Cadre expérimental

Tous les réglages ont été effectués sur l'ensemble d'apprentissage seul, celui-ci étant divisé en un ensemble d'apprentissage de 50000 points et un ensemble de validation de 10000 points. Les images ont été détournées de leur cadre blanc (on a donc retiré 2 pixels de chacun des 4 cotés). Les images sont ainsi réduites à 576 dimensions au lieu de 784. Par ailleurs la valeur des pixels a été translatée et réduite à l'intervalle $[-1, 1]$.

MNIST comprenant 10 classes, il a fallut choisir une méthode de classification multi-classes. Les deux principales, facilement mises en œuvre à partir de classifieurs binaires, sont le 1 contre 1 et le 1 contre tous. Dans le premier cas, nous obtenons 45 classifieurs binaires apprenant chacun sur une moyenne de 12000 points (l'équivalent de 2 classes), et la classification est donnée par un vote majoritaire sur l'ensemble des classifieurs. Dans l'autre cas nous obtenons 10 classifieurs qui apprennent chacun sur tous les points de la base d'apprentissage et la classification est donnée par le classifieur qui affiche la plus grande valeur de la fonction de décision. La complexité

Noyau	σ	C	temps d'apprentissage	erreur 1	erreur 2
Gaussien	0.002	20	1h45 (de 30s à 400s)	1.71%	0.50%
Gaussien	0.004	5	2h45 (de 60s à 720s)	1.48%	0.48%
polynomial ordre 5.	0.0005	20	1h15 (de 16s à 360s)	1.90%	0.69%

Tableau 2. Résultat des SVM sur la base de donnée MNIST. Entre parenthèses est précisé en seconde les temps minimum et maximum pris pour estimer un SVM. L'erreur 2 correspond à la prise en compte de la deuxième classe proposée lors du vote.

de *SimpleSVM* étant fortement liée au nombre de vecteurs supports et pour des raisons de capacité de mémoire, nous avons choisi d'implémenter le 1 contre 1.

Les résultats publiés pour MNIST avec des SVM utilisent le noyau gaussien ou le noyau polynomial. Nous avons choisi d'utiliser le noyau gaussien de la forme $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\sigma(\mathbf{x}_i - \mathbf{x}_j)^2)$. Nous avons également utilisé un noyau polynomial (dont l'ordre est arbitrairement fixé à 5) de la forme $k(\mathbf{x}_i, \mathbf{x}_j) = (\sigma * \mathbf{x}_i^\top * \mathbf{x}_j + 1)^5$. Dans tous les cas la méthode de résolution utilisée est Sherman-Morrison. Les deux paramètres essentiels à déterminer sont la largeur de bande du noyau (σ) et le paramètre de régularisation C . Le problème pour régler ces paramètres sur une base de données de cette dimension est le temps de calcul. Nous avons donc cherché à régler les paramètres sur un sous-ensemble des données et tenté d'utiliser ces valeurs optimales pour l'ensemble des données. Nous avons travaillé sur une grille de paramètres munie d'un masque (de façon à éviter les combinaisons de paramètres qui vont sur-régulariser (et mettre tous les points dans la marge) ou pour sur-apprendre (équivalent aux *hard margins* avec C inactif).

La figure 2 montre une carte obtenue pour un ensemble d'apprentissage de 5000 points et un ensemble de test de 10000 points, tirés au hasard dans le jeu d'apprentissage. Chaque point donne le taux de bonne classification du classifieur 1 contre 1 pour la combinaison de paramètres (σ, C). Les temps d'apprentissage dépendent des paramètres dans le sens où plus C est petit, plus il y a de vecteurs supports dont les multiplicateurs α sont bloqués à la valeur de C et plus σ est grand, plus la largeur de bande est petite, plus on apprend par cœurs les données (ce qui donne plus de vecteur supports). Donc sur la carte 2, les temps d'apprentissage sont les plus courts pour σ petit et C grand, et les plus longs dans l'angle opposé. Pour avoir un ordre d'idée, l'apprentissage le plus long a duré 28 minutes et le plus court 5 minutes (pour les 45 classifieurs). Il est donc clair que le gain en performance ici est coûteux en temps.

6.2.2. Résultats et discussion

Les résultats présentés dans le tableau (2) sont obtenus sur le jeu de test. Celui-ci n'a été utilisé que pour l'obtention de ces résultats finaux.

Les performances en taux de classification n'atteignent pas la hauteur des meilleurs résultats publiés. Cela s'explique en partie par l'absence de pré-traitement sur les images. Car notre objectif n'était de donner la meilleure solution mais de montrer

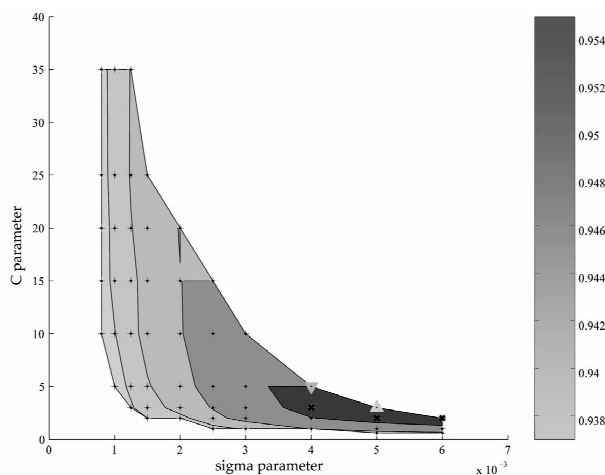


Figure 2. Carte obtenue pour un ensemble d'apprentissage de 5000 points et un ensemble de test de 10000 points, tirés au hasard dans le jeu d'apprentissage. Chaque point donne le taux de bonne classification du classifieur 1 contre 1 pour la combinaison de paramètres (σ, C) . Le meilleur taux de classification est montré par le triangle pointant vers le haut, le meilleur compromis performance/temps de calcul par le triangle pointant vers le bas.

qu'en un temps raisonnable (une heure de demi de calcul) et avec un environnement de calcul convivial comme Matlab, il est résoudre un problème réel de grande taille. En revanche, les performances en vitesse d'exécution sont satisfaisantes compte tenu du fait que l'on traite la base complète sans stratégie de *chunking* [OSU 97] ou autre. Il faut en moyenne 3 à 4 minutes pour entraîner un classifieur binaire avec 12000 points (ces temps dépendent des paramètres choisis dans le mesure où le temps de calcul dépend aussi du nombre de vecteurs supports).

6.3. Invariances et reconnaissance de caractères

6.3.1. Cadre Expérimental

Nous utilisons ici la base de données USPS qui contient 7291 images 16×16 pixels compris dans $[-1, 1]$ pour l'apprentissage et 2007 images de test, celle-ci étant réservée pour l'obtention des résultats finaux. Les images représentent les chiffres de 0 à 9 issus de codes postaux. Cette base de données est largement utilisée et de nombreux résultats sont disponibles pour la comparaison des performances. Par ailleurs ces données sont difficiles, l'erreur humaine est de 2,5%. Les transformations à considérer sont déduites de la nature des données. Un chiffre à la même signification indépendamment des translations, petites rotations ou l'épaisseur du trait. Par conséquent les

transformations utilisées dans les expériences sont les translations verticales et horizontales de 1 pixel, les rotations d'angles 10° , l'épaississement et l'affinement du trait. Ces transformations sont calculées à la volée quand nécessaire. Le but est d'éviter le stockage de toutes les transformations pour pouvoir traiter des grands jeux de données, cela augmente en revanche le temps de calcul.

6.3.2. Résultats et discussion

Notre objectif est ici d'une part de montrer que notre algorithme est rapide et efficace et d'autre part d'explorer différentes combinaisons de transformations (tâche jusque là rendue fastidieuse par les temps de calculs, les résultats publiés avec les SVM concernent souvent uniquement la translation de 1 pixel). Les résultats sont montrés dans le tableau 3 et ont été obtenus en appliquant l'heuristique de la recherche incomplète. Les paramètres ont été obtenus par validation croisée et le tableau 4 donne les principaux résultats publiés.

Noyaux	Largeur de bande	C	Transformation	Erreur	Temps
Gaussien	exp(-7)	1000	aucune	5.08	-
Poly 3	exp(-4)	0.1	aucune	4.04	270 sec
Poly 3	0.02	10	aucune	4.38	96 sec
Poly 3	0.02	10	tr	3.84	615 sec
Poly 3	0.02	10	tr+rot	3.79	-
Poly 5	0.1	10^{-5}	aucune	4.09	235 sec
Poly 5	0.1	10^{-4}	tr	3.44	1800 sec
Poly 5	0.1	10^{-4}	tr+rot	3.19	3200 sec
Poly 5	0.1	10^{-4}	tr+aff	3.14	-
Poly 5	0.1	10^{-4}	tr+aff+épai	3.24	2400 sec
Poly 5	0.1	10^{-4}	tr+rot+aff	2.99	2300 sec
Poly 5	0.1	10^{-4}	tr+rot+aff+épai	3.24	4800 sec
Poly 6	0.1	10^{-4}	aucune	4.33	336 sec
Poly 6	0.1	10^{-4}	tr+rot+aff	3.14	3300 sec

Tableau 3. Résultats sur USPS : Sont reportés dans ce tableau les résultats obtenus avec Invariant SimpleSVM. Les transformations appliquées sont la translation de 1 pixel (notée tr), la rotation (rot), l'affinement et l'épaississement des traits (respectivement aff et épai). Le meilleur résultat est obtenu en moins de 40 minutes.

Nous montrons ici que *Invariant SimpleSVM* est efficace. Remarquons ici que nous avons implémenté les version équivalentes aux *Virtual-SVM* [SCH 96] mais que nous obtenons toutefois de meilleurs résultats. Cela s'explique en particulier par le fait que nous sommes plus flexibles sur le choix des points pour lesquels nous générons des points virtuels. En effet nous regardons les transformations de chaque point qui est au moins une fois candidat pour être vecteur support, même s'il ne l'est pas au final. Le calcul des invariances ralentit considérablement les calculs (par un facteur 10 en moyenne en fonction des transformations considérées) mais la méthode reste rapide

Méthode	Erreur	Temps
Tangent Distance Kernel [HAA 02]	3.4 %	7109 sec
Tangent Vector and Local Rep. [KEY 02]	2.0 %	-
Virtual SVM [SCH 96]	3.2 %	-
Invariance Hyperplane + V-SVM [Sch 98]	3.0 %	-
Gaussian Mixture+TD+Virtual Samples [DAH 01]	2.4 %	-
<i>Invariant SimpleSVM</i> (ici)	3.0 %	2300 sec
Human performance	2.5 %	-

Tableau 4. Résultats publiés sur USPS : Sont reportés ici les principaux résultats et les temps de calcul (apprentissage+test) quand ils sont disponibles sur USPS.

comparée aux autres approches. Quant à l'effet des différentes transformations, il est difficile de conclure. On note toutefois que la translation améliore plus les résultats.

7. Conclusion

Les SVM sont connus comme étant performants mais lents. Ici nous montrons qu'il est possible de résoudre le problème d'optimisation des SVM dans des temps tout à fait raisonnables et ce en Matlab. Dans cet article nous avons présenté les aspects liés à la mise en oeuvre d'un algorithme rapide. L'algorithme SimpleSVM est expliqué et l'implémentation en Matlab détaillée. Les principaux résultats de cette étude sont d'une part une boîte à outils disponible qui fournit des méthodes rapides pour SVM et d'autre part des expérimentations montrant les capacités de cet algorithme en terme de temps de calcul et de l'évolution de ces temps. En outre, le fait de pouvoir initialiser la méthode à partir d'une solution candidate arbitraire permet de facilement régler les hyper paramètres par des méthodes de validation croisée. Par ailleurs, de part la structure de l'algorithme, il est possible de traiter des problèmes variés tels que les invariances. Là encore nous obtenons de bons résultats rapidement sans modifier fondamentalement l'algorithme mais simplement en y modifiant une étape.

D'un point de vue plus général, nous nous intéressons ici à la question de la complexité dans les méthodes d'apprentissage et les SVM en particulier. Notre objectif est de profiter des particularités du problème des SVM pour diminuer les temps de résolution à des ordres allant de $\mathcal{O}(m)$ à $\mathcal{O}(m^2)$.

Remerciements

Nous tenons à remercier Jean-Pierre Yvon qui nous avait initialement donné l'idée d'utiliser une méthode de contraintes actives sur ce problème, Olivier Bodard qui nous a aidé à l'améliorer, ainsi que Alice Briant et Yannick Pencilé qui nous ont aidés lors de la finalisation de l'article.

National ICT Australia (NICTA) est financé par l'initiative du gouvernement australien *Backing Australia's Ability* et en partie par le Conseil de la Recherche Australienne (ARC, Australian Research Council). Ce travail a été soutenu par des financements de l'ARC.

Ce travail est financé en partie par le Programme IST de la Communauté Européenne, avec le réseau d'excellence PASCAL, IST-2002-506778. Cette publication reflète uniquement le point de vue des auteurs.

8. Bibliographie

- [ATT 99] ATTEIA M., GACHES J., *Approxiation Hilbertienne*, Presses Universitaires de Grenoble, 1999.
- [CAW 00] CAWLEY G. C., « MATLAB Support Vector Machine Toolbox (v0.55 β) », University of East Anglia, School of Information Systems, Norwich, Norfolk, U.K. NR4 7TJ, 2000, <http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox>.
- [CHA 02] CHAPPELLE O., SCHÖLKOPF B., « Incorporating invariances in nonlinear SVMs », S. D. T. G. B., Z. G., Eds., *Advances in Neural Information Processing Systems*, vol. 14, Cambridge, MA, USA, 2002, MIT Press, p. 609-616.
- [DAH 01] DAHMEN J., KEYSERS D., NEY H., « Combined Classification of Handwritten Digits Using the 'Virtual Test Sample Method' », *Lecture Notes in Computer Science*, vol. 2096, 2001.
- [GIL 91] GILL P. E., MURRAY W., SAUNDERS M. A., WRIGHT M. H., « Inertia-Controlling Methods for General Quadratic Programming », *SIAM Review*, vol. 33, n° 1, 1991.
- [GRA 04] GRAEPEL T., HERBRICH R., « Invariant Pattern Recognition by Semi-Definite Programming Machines », THRUN S., SAUL L., SCHÖLKOPF B., Eds., *Advances in Neural Information Processing Systems 16*, MIT Press, Cambridge, MA, 2004.
- [GRE 93] GRENANDER U., *General Pattern Theory*, Oxford University Press, 1993.
- [HAA 02] HAASDONK B., KEYSERS D., « Tangent Distance Kernels for Support Vector Machines », *International Conference on Pattern Recognition*, Quebec City, Canada, août 2002.
- [KEY 02] KEYSERS D., PAREDES R., NEY H., VIDAL E., « Combination of Tangent Vectors and Local Representation for Handwritten Digit Recognition », *Lecture Notes in Computer Science*, vol. LNCS 2396, p. 538-547, SPR2002, International Workshop on Statistical Pattern Recognition, Windsor, Ontario, Canada, springer-vertag édition, Aug 2002.
- [LEC 98] LECUN Y., BOTTOU L., BENGIO Y., HAFNER P., « Gradient-based learning applied to document recognition », *Proceedings of the IEEE*, vol. 86, n° 11, 1998, p. 2278-2324, <http://yann.lecun.com/exdb/mnist/>.
- [LOO 04a] LOOSLI G., « Fast SVM Toolbox in Matlab based on SimpleSVM algorithm », 2004, <http://asi.insa-rouen.fr/~gloosli/>.
- [LOO 04b] LOOSLI G., CANU S., VISHWANATHAN S., SMOLA A. J., CHATTOPADHYAY M., « Une boîte à outils rapide et simple pour les SVM », LIQUIÈRE M., SEBBAN M., Eds., *CAP 2004 - Conférence d'Apprentissage*, Presses Universitaires de Grenoble, 2004, p. 113-128.

- [MIN 83] MINOUX M., *Programation mathématique. Théorie et algorithmes. Tome 1*, Dunod, 1983.
- [OSU 97] OSUNA E., FREUND R., GIROSI F., « An improved training algorithm for Support Vector Machines », PRINCIPE J., GILE L., MORGAN N., WILSON E., Eds., *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, 1997, p. 276-285.
- [PLA 99] PLATT J., « Fast training of support vector machines using sequential minimal optimization », SCHÖLKOPF B., BURGESS C., SMOLA A., Eds., *Advanced in Kernel Methods - Support Vector Learning*, MIT Press, 1999, p. 185-208.
- [POZ 03] POZDNOUKHOV A., BENGIO S., « Tangent Vector Kernels for Invariant Image Classification with SVMs », IDIAP-RR n° 75, 2003, IDIAP, Martigny, Switzerland, Submitted to International Conference on Pattern Recognition 2004.
- [ROO 00] ROOBAERT D., « DirectSVM : A simple support vector machine perceptron. », *Neural Network for Signal Processing X - Proceedings of the 2000 IEEE Workshop*, New York IEEE, 2000, p. 356-365.
- [SCH 96] SCHÖLKOPF B., BURGESS C., VAPNIK V., « Incorporating invariances in support vector learning machines. », C. VON DER MALSBERG W. VON SEELEN J. V., SENDHOFF B., Eds., *Artificial Neural Networks — ICANN'96*, vol. 1112, Berlin, 1996, Springer Lecture Notes in Computer Science, p. 47-52.
- [Sch 98] SCHÖLKOPF B., SIMARD P., SMOLA A., VAPNIK V., « Prior knowledge in support vector kernels », JORDAN M. I., KEARNS M. J., SOLLA S. A., Eds., *Advances in Neural Information Processing Systems 10*, Cambridge, MA, 1998, MIT Press, p. 640-646.
- [SCH 02] SCHÖLKOPF B., SMOLA A. J., *Learning with Kernels*, MIT Press, 2002.
- [SIM 93] SIMARD P., LECUN Y., J. D., « efficient pattern recognition using a new transformation distance », HANSON S., COWAN J., GILES L., Eds., *Advances in Neural Information Processing Systems*, vol. 5, Morgan Kaufmann, 1993.
- [SIM 96] SIMARD P., LECUN Y., DENKER J. S., VICTORRI B., « Transformation Invariance in Pattern Recognition-Tangent Distance and Tangent Propagation », *Neural Networks : Tricks of the Trade*, 1996, p. 239-27.
- [STE 03] STEINWART I., « Sparseness of Support Vector Machine », *Journal of Machine Learning Research*, vol. 4, 2003, p. 1071-1105.
- [VIS 03] VISHWANATHAN S. V. N., SMOLA A. J., MURTY M. N., « SimpleSVM », *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.