

SVM et apprentissage des très grandes bases de données*

Gaëlle Loosli¹, Stéphane Canu¹, Léon Bottou²

Laboratoire d'Informatique, Traitement de l'Information et Systèmes
INSA de Rouen
Avenue de l'Université, 76801 Saint Etienne du Rouvray
gaelle.loosli@insa-rouen.fr et stephane.canu@insa-rouen.fr
NEC laboratories of America
Princeton, NJ 08540, USA
leon@bottou.org

Résumé :

Le but de ce travail est de montrer qu'il est possible de faire de la discrimination à l'aide de Séparateurs à Vaste Marge (SVM) sur des très grandes bases de données (des millions d'exemples, des centaines de caractéristiques et une dizaine de classes). Pour traiter cette masse de données, nous nous proposons d'utiliser un algorithme « en ligne » où les exemples sont présentés les uns après les autres. Cette approche permet à la fois une mise à jour rapide de la solution (qui ne dépend que d'un seul exemple à la fois) et la gestion efficace de la base d'apprentissage (qui n'a pas à être entièrement en mémoire). L'application visée est la reconnaissance de caractères avec prise en compte des invariances dans les données. Pour cela, nous adaptons l'algorithme LASVM (Bordes *et al.*, 2005) (une méthode en ligne pour les SVM) en nous inspirant de (Loosli *et al.*, 2005) pour y intégrer la connaissance *a priori* sur l'invariance.

Mots-clés : SVM, apprentissage en ligne, grande échelle, méthodes actives, invariances, masses de données, discrimination.

1 L'apprentissage en ligne à grande échelle

Notre étude se place dans un contexte bien défini : nous nous intéressons à la question de l'usage des SVM pour l'apprentissage en ligne de très grandes bases de données. L'apport principal de ces travaux est de montrer que contrairement à une idée très répandue, il n'est pas exclu d'utiliser ce type de méthodes pour des problèmes de grandes dimensions.

*We would like to acknowledge support for this project from the NSF grant CCR-0325463. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

L'apprentissage en ligne pose un problème assez particulier. En effet on s'impose de ne jamais *revenir en arrière* dans les données. En pratique, tout point ne peut être considéré qu'une seule fois pour l'apprentissage et tout point non retenu est définitivement oublié. Il faut donc à la fois retenir suffisamment de points pour pouvoir résoudre le problème traité et en oublier suffisamment pour ne pas avoir à garder en mémoire trop d'information redondante (ce qui aurait pour effet de limiter la capacité à traiter plus de données). L'algorithme LASVM (Bordes *et al.*, 2005) répond à ces critères. Cette méthode s'appuie sur une sélection active des points à intégrer dans la solution et donne des résultats très satisfaisants pour une utilisation en ligne (bien qu'il soit également possible de l'utiliser dans un mode moins contraint sur ce point et d'autoriser plusieurs passes sur les données).

L'apprentissage de grandes bases de données est un objectif important car cela représente une grande part des problèmes réels. Les limitations bien connues sont la mémoire et le temps de calcul. Les enjeux sont donc de trouver des algorithmes qui conduisent à des solutions les plus parcimonieuses possibles (ce qui a pour effet immédiat une réduction de la mémoire et du temps nécessaire pour la classification et on peut espérer une réduction des besoins de l'apprentissage également).

L'apprentissage en ligne et l'apprentissage des grandes bases de données sont des objectifs compatibles et complémentaires. Si on admet que l'on a à disposition un algorithme en ligne qui donne des solutions équivalentes à un apprentissage hors ligne (c'est-à-dire que l'erreur en test est similaire, que l'on ait vu une ou plusieurs fois chaque exemple), alors on peut envisager d'utiliser cette méthode pour apprendre de grandes bases de données. De la même façon, plus on possède d'exemples pour un problème donné et plus on peut s'attendre à obtenir de meilleurs résultats. Ainsi les grandes bases de données sont (au même titre des les données réellement en ligne) traitables par les algorithmes en ligne.

2 SVM adaptées aux grandes bases de données

Afin d'expliquer le fonctionnement de LASVM, nous présentons dans un premier temps la formulation classique des SVM et discutons de la géométrie correspondante au problème de programmation quadratique (QP). Ensuite nous abordons les itérations qui constituent LASVM et la façon de les utiliser.

2.1 Support Vector Machines

On considère un problème de classification binaire, avec des exemples d'apprentissage $x_1 \dots x_n \in E$ associés à des étiquettes $y_1 \dots y_n \in \{+1, -1\}$. Nous nous donnons un noyau $k(\cdot, \cdot)$, la fonction symétrique positive représentant une mesure de similarité entre ses deux variables d'entrée. L'ensemble des fonctions représentées par combinaison linéaire finie du noyau forme l'espace \mathcal{H}_0 et s'écrit :

$$\mathcal{H}_0 = \left\{ f : E \mapsto \mathbb{R} \mid n \in \mathbb{N}, \{\alpha_i\}_1^n \in \mathbb{R}, \{x_i\}_1^n \in E, f(x) = \sum_{i=1}^n \alpha_i k(x_i, x) \right\}$$

Cet espace muni du produit scalaire

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^n \sum_{j=1}^m \alpha_i \alpha_j k(x_i, x_j)$$

est un espace pré-hilbertien, et de la norme induite est $\|f\|^2 = \langle f, f \rangle_{\mathcal{H}_0}$. L'espace des hypothèses \mathcal{H} , obtenu en complétant \mathcal{H}_0 ($\mathcal{H} = \overline{\mathcal{H}_0}$) est un espace de Hilbert à noyaux reproduisants. La propriété de reproduction du noyau k est donnée par la relation

$$\langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^n \alpha_i k(x_i, \cdot), k(\cdot, x) \right\rangle_{\mathcal{H}} = f(x)$$

La fonction de décision recherchée s'exprime par $D(x) = \text{signe}(f(x) + b)$ avec $f \in \mathcal{H}$ et le biais b . Le problème primal des SVM est alors :

$$\left\{ \begin{array}{l} \min_{f, b} \frac{1}{2} \|f\|^2 + C \sum_{i=1}^n \xi_i \\ y_i (f(x_i) + b) \geq 1 - \xi_i, \quad i = 1, n \\ \xi_i \geq 0, \quad i = 1, n \end{array} \right. \quad (1)$$

avec ξ_i les variables positives de relâchement de contraintes et C le paramètre de régularisation. Trouver $f \in \mathcal{H}$, un problème difficile, revient en fait à trouver les coefficients $\alpha \in \mathbb{R}^n$ (Reresenter Theorem, (Schölkopf & Smola, 2002)), ce qui est un problème plus simple en passant par la forme duale (1). Le minimum est donc atteint en résolvant le problème quadratique dual¹ :

$$\max_{\alpha} W(\alpha) = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i, j} \alpha_i \alpha_j k(x_i, x_j) \quad \text{avec} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = 0 \\ 0 \leq y_i \alpha_i \leq C \end{array} \right. \quad (2)$$

Un exemple x_i est appelé vecteur support quand le coefficient α_i correspondant est non nul. Le nombre de vecteurs supports représente asymptotiquement une proportion constante du nombre total d'exemples (Steinwart, 2004).

2.2 Directions admissibles

La géométrie du problème quadratique dual des SVM (2) est résumé par la figure 1. Les contraintes de boîte $0 \leq y_i \alpha_i \leq C$ restreignent la solution à un hypercube de dimension n . La contrainte d'égalité $\sum \alpha_i = 0$ restreint en plus la solution à un polytope \mathcal{F} de dimension $n - 1$. On considère un point admissible $\alpha \in \mathcal{F}$ et une droite contenant ce point. Cette droite indique une *direction admissible* si son intersection avec le polytope contient des points autres que α .

Les algorithmes de direction admissible mettent à jour de manière itérative la position d'un point admissible α_t . En premier lieu ils choisissent une direction u_t et ensuite ils suivent cette direction à la recherche du point admissible α_{t+1} qui maximise la fonction

¹ α_i est positif quand $y_i = +1$ et négatif quand $y_i = -1$.

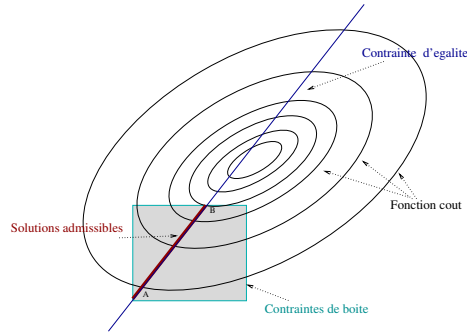


FIG. 1 – Géométrie du problème quadratique dual (2). Les contraintes de boîte $0 \leq y_i \alpha_u \leq C$ restreignent la solution à un hypercube de dimension n . La contrainte d'égalité $\sum \alpha_i = 0$ restreint de plus la solution à un polytope de dimension $n - 1$ qui est le segment $[AB]$ dans cette figure en 2 dimensions.

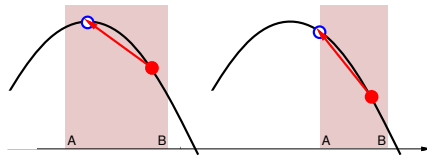


FIG. 2 – La fonction de coût quadratique limitée à la ligne de recherche peut avoir son maximum à l'intérieur (gauche) ou à l'extérieur (droite) de la boîte de contraintes.

coût. L'optimum est atteint quand aucune amélioration n'est plus possible (Zoutendijk, 1960).

La figure 2 illustre les deux configurations possibles pour la recherche de point admissible. La fonction de coût quadratique restreinte à la direction de recherche atteint son maximum soit à l'intérieur, soit à l'extérieur du polytope. La position du nouveau point admissible α_{t+1} se trouve grâce aux dérivées de premier et second ordre de α_t et aux emplacements des bornes A et B (qui elles sont données par les contraintes de boîte).

$$\alpha_{t+1} = \alpha_t + \max \left\{ A, \min \left\{ B, \frac{dW(\alpha_t + \lambda u_t)}{d\lambda} \left(\frac{d^2 W(\alpha_t + \lambda u_t)}{d\lambda^2} \right)^{-1} \right\} \right\} u_t \quad (3)$$

Le calcul de ces dérivées dans une direction arbitraire u_t peut coûter cher car cela implique tous les termes (n^2) de la matrice noyau K . Au lieu de chercher une direction admissible parmi toutes les directions possibles, il suffit en fait d'en choisir une dans un ensemble restreint de directions bien choisies (Bordes *et al.*, 2005, appendice). Ces directions peuvent avoir beaucoup de coefficients nuls ce qui simplifie le calcul des dérivées.

SMO (Platt, 1999) se sert de cela en ne considérant que les directions admissibles

qui ne modifient que deux coefficients α_i et α_j par des valeurs opposées. La variante la plus répandue du SMO repose sur un critère du premier ordre pour sélectionner les paires (i, j) qui définissent les directions successives de recherche :

$$i = \underset{\{s \mid \alpha_s < \max(0, y_s C)\}}{\arg \max} \frac{\partial W}{\partial \alpha_s} \quad j = \underset{\{s \mid \alpha_s > \min(0, y_s C)\}}{\arg \min} \frac{\partial W}{\partial \alpha_s} \quad (4)$$

2.3 Complexité de l'optimisation

Le temps requis pour résoudre le problème quadratique dual du SVM grandit en n^β avec $2 \leq \beta \leq 3$. Par ailleurs, quand n augmente, la matrice noyau devient très grande et ne peut plus être stockée en mémoire. Les solveurs SVM modernes maintiennent un cache des valeurs du noyau récemment calculées. Un ordre de grandeur est obtenu en évaluant quelle taille de cache est nécessaire pour éviter de recalculer systématiquement les mêmes valeurs du noyau. Soit n le nombre d'exemples et s le nombre de vecteurs supports (qui grandit linéairement avec le nombre d'exemples). Soit en outre $r \leq s$ le nombre de vecteurs supports « libres », c'est-à-dire tels que $0 < y_i \alpha_i < C$. Dans le cas de SMO, le cache doit être capable de contenir environ $n r$ entrées : les paires de coefficients (i, j) sont en général choisis parmi les r vecteurs supports libres ; chaque étape du SMO requière n valeurs distinctes du noyau pour chaque coefficient sélectionné.

Un problème d'apprentissage n'est pas exactement un problème d'optimisation. En particulier cela n'a pas beaucoup de sens d'optimiser la fonction objectif primale (eq. 1) avec une précision beaucoup plus importante que celle issue du fait de travailler avec un nombre fini d'exemples.

Des résultats formels existent pour l'apprentissage en ligne, qui utilisent une approximation du gradient stochastique. Ces algorithmes conçus pour le fonctionnement en ligne vont beaucoup plus vite que l'optimisation directe de la fonction objectif primale (Bottou & LeCun, 2004) : ils n'optimisent pas cette fonction de manière aussi précise mais ils atteignent une erreur en test équivalente plus rapidement.

A notre connaissance, il n'existe pas de preuve formelle pour les algorithmes en ligne résolvant la fonction duale de coût (eq. 2) et qui sont donc facilement utilisables avec des méthodes à noyaux. Certains algorithmes en ligne à noyaux existent (Freund & Schapire, 1998) mais ne sont pas très performants en pratique.

2.4 LASVM en ligne

Deux cadres ont été utilisés pour étudier les algorithmes en ligne, soit par les approximations stochastiques (Bottou, 1998) soit par le Perceptron (Novikoff, 1962). Plusieurs auteurs (Freund & Schapire, 1998; Frieß *et al.*, 1998; Gentile, 2001; Li & Long, 2002; Crammer *et al.*, 2004) ont modifié le Perceptron pour inclure une marge. D'autres variantes plus anciennes du Perceptron telles que le *minover* et l'*adatron* (Nadal, 1993, et références incluses), sont aussi très proches des SVM.

Le manque de résultats théoriques n'empêche pas la recherche d'algorithmes à noyaux performants en ligne. Le Budget Perceptron (Crammer *et al.*, 2004) montre qu'un algorithme à noyaux en ligne doit être capable à la fois d'insérer et de retirer un vecteur sup-

port de l'expansion courante. Le Huller (Bordes & Bottou, 2005) montre que ces deux actions sont dérivables de l'augmentation incrémentale de la fonction duale $W(\alpha)$.

LASVM est un SVM en ligne qui augmente incrémentalement la fonction objectif duale $W(\alpha)$. Il maintient un vecteur des coefficients courants α_t et l'ensemble des indices des vecteurs supports correspondants \mathcal{S}_t . Chaque itération de LASVM reçoit un nouvel exemple $(x_{\sigma(t)}, y_{\sigma(t)})$ et met à jour le vecteur de coefficients α_t en faisant deux étapes de SMO appelées *process* et *reprocess*.

- *Process* est une recherche de point admissible SMO (eq. 3) le long d'une direction définie par la paire de points formée de l'indice de l'exemple courant $\sigma(t)$ et d'un autre exemple choisi parmi les indices vecteurs supports courants \mathcal{S}_t en utilisant le critère de premier ordre (eq. 4). Cette opération donne un nouveau vecteur de coefficients α'_t et peut insérer $\sigma(t)$ dans l'ensemble d'indices \mathcal{S}'_t correspondant.
- *Reprocess* est une recherche de point admissible SMO (eq. 3) le long d'une direction définie par la paire de points (i, j) choisis dans les indices des vecteurs supports courants \mathcal{S}'_t en utilisant le critère de premier ordre (eq. 4). Cette opération donne un nouveau vecteur de coefficients α_{t+1} et peut enlever i ou j de l'ensemble des indices \mathcal{S}_{t+1} .

Répéter les itérations LASVM sur des exemples choisis aléatoirement converge vers la solution SVM avec une précision arbitraire. Toutefois, l'expérience montre que LASVM donne aussi de bons résultats après une seule présentation de chaque exemple. Après avoir présenté chaque exemple, les coefficients finaux sont affinés en faisant des *reprocess* jusqu'à convergence de la fonction duale $W(\alpha_t)$.

LASVM en ligne

- 1: Initialisation α_0
- 2: **tant que** il y a des points non traités **faire**
- 3: sélectionner un point non traité $(x_{\sigma(t)}, y_{\sigma(t)})$
- 4: *Process*($\sigma(t)$)
- 5: *Reprocess*
- 6: **fin tant que**
- 7: *Finition* : répéter *reprocess* jusqu'à convergence

Cet algorithme à une passe est plus rapide que SMO et demande beaucoup moins de mémoire. Pour éviter le calcul systématique des valeurs du noyau, le cache doit contenir sr entrées alors que SMO en demande nr , c'est à dire que l'on permet d'oublier toutes les valeurs concernant les points non vecteurs supports puisque l'on ne revient jamais dessus alors que dans un SVM classique, on doit garder ces valeurs. Comme en général $s \ll n$ on gagne beaucoup en mémoire.

2.5 LASVM actif

Chaque itération de LASVM en ligne sélectionne aléatoirement un exemple d'apprentissage $(x_{\sigma(t)}, y_{\sigma(t)})$. Des stratégies plus sophistiquées de sélection d'exemples conduisent à de meilleures performances de mise à l'échelle. On peut citer quatre stratégies :

- *Sélection aléatoire* : Prendre un exemple non vu au hasard.
- *Sélection par gradient* : Prendre l'exemple le moins bien classés (la plus petite valeur de $y_k f(x_k)$) dans un ensemble de points non vus. Ce critère est très proche de ce qui est fait dans (Loosli *et al.*, 2004).
- *Sélection active* : Prendre l'exemple qui est le plus proche de la frontière de décision (la plus petite valeur de $|f(x_k)|$) dans un ensemble de points non vus. Ce critère choisit un exemple indépendamment de son étiquette.
- *Sélection autoactive* : Tirer au plus 100 points non vus, mais arrêter dès que 5 d'entre eux sont à l'intérieur des marges. Parmi les 5, choisir le plus proche de la frontière de décision.

On montre empiriquement que la sélection *active* ou *autoactive* donnent des performances comparables ou meilleures en utilisant un nombre plus restreint de vecteurs supports. Cela se comprend car l'accroissement linéaire du nombre de vecteur support est lié au fait que tout exemple mal classé est automatiquement vecteur support dans la formulation classique du SVM. Sélectionner les points uniquement près de la frontière exclue de la solution un grand nombre d'outliers.

La partie suivante présente le problème des invariances et montre comment LASVM est adapté pour répondre à ce problème, en particulier grâce à ses particularités (actif et en ligne).

3 Application aux invariances

L'invariance dans les données est fréquente. Les images sont un exemple illustratif par excellence, puisqu'il est en effet très naturel de reconnaître un objet déplacé ou tourné dans une image alors que c'est une tâche complexe pour une machine. Il existe beaucoup de méthodes pour essayer de résoudre ce problème (Grenander, 1993; Simard *et al.*, 1993; Wood, 1996; Schölkopf *et al.*, 1996; Leen, 1995). Nous donnons une illustration de l'influence des invariances dans une tâche de classifications et discutons ensuite de l'approche que nous avons sélectionnée pour répondre à cette question.

3.1 De l'influence des invariances

Décrivons tout d'abord pourquoi nous avons besoin de nous occuper des invariances sur un exemple jouet. Prenons des points en deux dimensions. Nous savons qu'il existe une incertitude sur les mesures de ces points, se traduisant par une rotation centrale du plan. Nous illustrons cela sur la figure 3 : l'image (a) montre les points mesurés et la classification correspondante. L'image (b) montre la trajectoire de chaque point en tenant compte de l'information sur les mesures. On voit alors que la classification effectuée précédemment n'est plus si bonne. Toutefois, l'image (c) montre qu'il existe une frontière qui classe bien à la fois les points mesurés et leurs variations. Cette nouvelle frontière montre que l'on a besoin soit de savoir classer les perturbations des points, soit de savoir ajouter des points virtuels qui aideront à tenir compte de l'information à priori dont on dispose (image (d)).

Ce simple exemple donne une bonne idée de la complexité du problème. Apprendre les variations amène à des problèmes trop complexes (Graepel & Herbrich, 2004) et

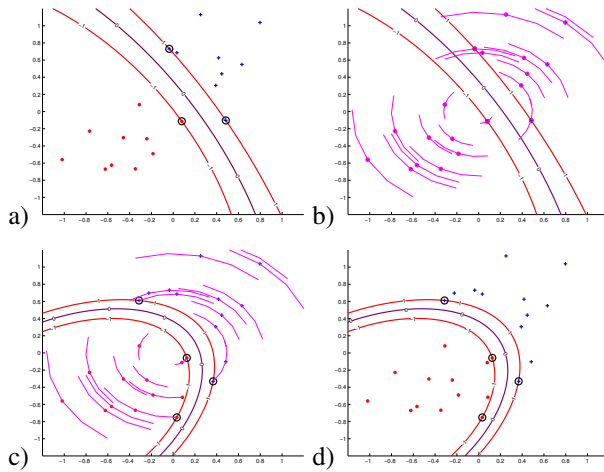


FIG. 3 – Ces figures illustrent l’influence de variations des points sur la frontière de décision pour un exemple jouet. L’image (a) montre les points mesurés et la classification correspondante. L’image (b) montre la trajectoire de chaque point en tenant compte de l’information sur les mesures. L’image (c) montre qu’il existe une frontière qui classe bien à la fois les points mesurés et leurs variations. L’image (d) tient compte de l’information à priori dont on dispose.

ajouter des points virtuels requiert beaucoup de mémoire ($n \times t$ avec t le nombre de transformations par points). Toutefois, pour une application réaliste, nous avons besoin d’une méthode peu complexe et qui ne stocke pas trop d’information supplémentaire inutile. En effet seules quelques transformations sont réellement utiles à la classification, les autres ne font qu’alourdir le système sans rien apporter. LASVM semble être un bon algorithme candidat pour résoudre les invariances. En effet, sa capacité de sélection active des points permet de ne choisir que les variations utiles des points d’origine et le fait d’être en ligne permet d’oublier les points inutiles et par conséquent d’économiser de la mémoire. De fait, nous pouvons envisager de résoudre un problème d’invariances utilisant beaucoup de transformations.

Plusieurs stratégies sont envisageables pour cette tâche, en excluant celles où l’on doit pré-calculer toutes les transformations avant de commencer l’apprentissage. Une première idée, proposée dans (Loosli *et al.*, 2005), est de calculer à la volée les transformations d’un point quand il devient candidat à être vecteur support. On choisit alors comme candidat soit l’original soit sa variation qui est meilleure candidate au sens d’un critère de sélection. Cette approche est intéressante mais présente un inconvénient majeur dans le cas où un point possède plusieurs variations également intéressantes pour la solution. Une façon de régler ce problème consiste à revenir plusieurs fois sur chaque point tant qu’il donne des variations utiles. Cette seconde stratégie n’est pas compatible avec la volonté de traiter les problèmes en ligne et par ailleurs, elle nécessite de recalculer à chaque fois toutes les transformations ou alors de la stocker, ce qui n’est pas avantageux en temps de calcul et en espace mémoire. On en vient donc à une troisième

stratégie, qui dissocie les exemples d'origine et leurs transformations. On considère alors chaque transformation comme un exemple à part entière. Cela résout d'une part le problème de plusieurs vecteurs supports issus du même exemple, mais cela apporte d'autre part une grande flexibilité dans l'algorithme. En effet, dans cette approche on apprend tout d'abord sur l'ensemble des originaux et on continue ensuite l'apprentissage en présentant des variations de points que l'on calcule à la volée. Ainsi, on peut arrêter d'apprendre quand on le souhaite (ou lorsque les limites de la machines sont atteintes). Plus on dispose de ressources et de temps, plus on peut tenir compte des variations, mais sans avoir à limiter à l'avance les possibilités d'introduire l'information à priori dont on dispose.

Après avoir donné une présentation plus formelle de l'intégration des invariances dans le problème des SVM, nous illustrons cette troisième approche sur la bien connue base d'OCR MNIST (LeCun *et al.*, 1998). Nous choisissons cette base pour les possibilités de comparaison à d'autres méthodes et aussi pour les capacités d'illustration du fait de travailler avec des images. Nous expliquons dans un premier temps comment nous avons incorporé les transformations de manière pratique et puis discuterons des résultats obtenus.

3.2 De la formulation des invariances

Nous proposons de redonner le cadre de formalisation des invariances en reconnaissance de formes. Soit $x \in E$ et Θ l'espace des paramètres de transformations applicables à x . Une transformation de forme est une application T qui associe une forme transformée $T(x, \theta)$ à une forme x et des paramètres de transformation $\theta \in \Theta$:

$$\begin{aligned} T : E \times \Theta &\rightarrow E \\ x, \theta &\mapsto T(x, \theta) \end{aligned}$$

De plus, on pose $T(x, 0) = x$.

Maintenant, si on considère un problème de classification, on définit la fonction de décision $D(x)$ comme l'application de \mathcal{X}^d dans $\{-1, 1\}$ qui associe $D(x)$ à x . Si on veut que cette fonction de décision soit invariante, il faut qu'elle donne la même décision pour la forme et pour toutes ses transformations :

$$D(T(x_i, 0)) = D(T(x_i, \theta)) \quad \forall \theta \in \Theta$$

Appliquées aux SVM dans le cas séparable, ces définitions donnent le problème primal suivant :

$$\left\{ \begin{array}{l} \min_{f, b} \frac{1}{2} \|f\|_{\mathcal{H}}^2 \\ y_i (f(T(\mathbf{x}_i, \theta)) + b) \geq 1 \end{array} \quad i \in [1, n], \theta \in \Theta \right. \quad (5)$$

Le problème dual associé est alors (Loosli *et al.*, 2005) :

$$\left\{ \begin{array}{l} \max_{\gamma \in \mathbb{R}^{n \times p}} -\frac{1}{2} \gamma^\top G \gamma + \mathbf{e}^\top \gamma \\ \gamma^\top \mathbf{y} = 0 \\ \gamma_i \geq 0 \end{array} \quad i \in [1, np] \right. \quad (6)$$

avec $\gamma = [\alpha_1(\theta); \alpha_2(\theta); \dots; \alpha_n(\theta)]$, G est la matrice bloc définie par $G_{IJ} = K^{ij}$ avec $K_{kl}^{ij} = k(T(x_i, \theta_k), T(x_j, \theta_l))$, e est un vecteur de 1 et p est la dimension de Θ .

3.3 Des invariances dans la pratique

Les invariances constituent un sujet déjà beaucoup étudié et il existe un grand nombre d'approches pour calculer les transformations (Simard *et al.*, 1993; Wood, 1996; Schölkopf *et al.*, 1996). Ici nous décrivons brièvement comment nous combinons ces approches de manière à les appliquer à un algorithme en ligne avec le moins de mémoire possible.

3.3.1 Vecteurs tangents

L'algèbre de Lie donne les outils pour calculer des transformations affines approchées en utilisant les vecteurs tangents (Simard *et al.*, 2000). Appliquer ces principes à une image consiste à calculer les dérivées horizontales et verticales de chaque image (ce qui revient en fait à faire la différence entre l'image et une translation sous-pixel d'elle-même, verticale ou horizontale). L'approximation des déformations affines (translation en x ou y, rotations, diagonales,...) est obtenue en faisant la combinaison linéaire pixel à pixel des vecteurs tangents et de l'image originale.

$$x_{af fine}(i, j) = x(i, j) + A_x * d_x * t_x(i, j) + A_y * d_y * t_y(i, j),$$

avec t_x et t_y respectivement les vecteurs tangents horizontaux et verticaux et $|A_x| = |A_y|$ les coefficients contrôlant la force des déformations appliquées. d_x et d_y sont 0 ou 1, de manière à pouvoir appliquer une transformation le long d'une seule dimension.

3.3.2 Épaisseur des traits

Les vecteurs tangents permettent également d'intervenir sur l'épaisseur des traits :

$$x_{epais}(i, j) = x(i, j) + B * \sqrt{t_x(i, j)^2 + t_y(i, j)^2},$$

avec β le coefficient qui règle la force de la transformation (si $B > 0$, les lignes seront plus épaisses et sinon elles seront plus fines).

3.3.3 Champs de déformation

En plus des transformations affines, nous considérons les transformations élastiques. L'idée est d'introduire des déformations locales aléatoires. Les champs de déformation sont utilisés pour changer les coordonnées d'un pixel. Les coefficients des champs de déformation sont générés aléatoirement et lissés par une gaussienne.

$$x_{deform e}(i, j) = x(i, j) + f_x(i, j) * t_x(i, j) + f_y(i, j) * t_y(i, j),$$

avec f_x et f_y les champs de déformation. Remarquons de chaque champ peut être utilisé en tant que champ de déformation horizontal ou vertical.

En plus des champs aléatoires, nous utilisons des champs réguliers et un champ nul. Ainsi nous pouvons obtenir des transformations affines et des transformations selon une seule direction. Enfin nous avons aussi généré des champs contrôlés, c'est-à-dire des champs réguliers (rotation par exemple) bruités.

3.3.4 Un nombre infini d'exemples

Puisque nous nous donnons pour cadre de ne pas pré-calculer et stocker les transformations, il nous faut une méthode rapide de calcul des variations des points. Pour ce faire nous compilons l'ensemble des transformations citées ci-avant en une seule expression :

$$x_t(i, j) = x(i, j) + A_x * f_x(i, j) * t_x(i, j) + A_y * f_y(i, j) * t_y(i, j) + B * \sqrt{t_x(i, j)^2 + t_y(i, j)^2},$$

Les coefficients d_x et d_y utilisés avec les vecteurs tangents sont remplacés par les champs de déformation, ce qui est possible puisque nous avons ajouté les champs réguliers, constants et nuls.

3.3.5 Translations

Toutes les transformations décrites sont des transformations sous-pixel. Bien que MNIST soit une base de données très propre avec des exemples centrés, il est utile d'ajouter des translations de 1 et 2 pixels dans toutes les directions.

4 Résultats sur MNIST et discussions

Nous avons appliqué nos transformations à MNIST, en essayant de faire un compromis entre temps de calcul et usage mémoire. Nous avons donc stocké les vecteurs tangents et des champs de déformation, mais toutes les transformations sont calculées à la demande en tirant au hasard des champs de déformations stockés et des coefficients de force. Par ailleurs nous avons mis en place une stratégie de cache pour les exemples transformés ainsi que pour le noyau.

4.1 Du réglage des déformations

En théorie, plus on prends en compte de déformations meilleur on est. Toutefois on risque d'introduire du bruit dans le cas de MNIST, compte tenu du fait que l'on ne cherchera pas à déformer les exemples de test et que la base est relativement régulière. Nous avons donc testé quelles déformations étaient les plus adaptées à notre problème en utilisant un jeu de validation. Nous avons par la même occasion fait une validation croisée sur le réglage du noyau. Les paramètres de la validation croisée sont les coefficients de force de déformation A , la largeur de bande γ du noyau que l'on choisie rbf (radial basis function). La figure 4.1 donne l'erreur en validation en fonction de A pour le meilleur noyau, appris sur 5000 points et 10 transformations par point. On voit que

changer l'épaisseur des traits n'est pas une bonne approche pour cette base. De la même manière, on peut se contenter des translations de 1 pixel au lieu de 2.

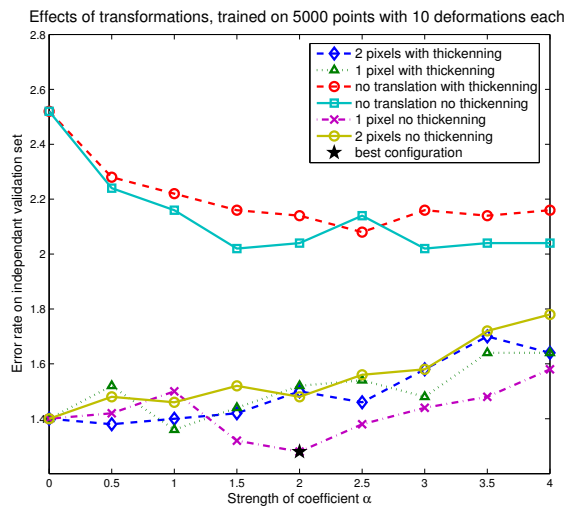


FIG. 4 – Effets des transformations sur la performance. Cette figure est obtenue sur un jeu de validation de 10000 points et appris avec 5000 points d'apprentissage, chacun déformé 10 fois (soit 55000 points en tout). Le noyau rbf a une largeur de bande $\gamma = 0.006$ obtenue par validation croisée. La meilleure configuration est obtenue pour les déformations élastiques de force $A = 2$ sans épaissement des traits et avec une translation possible de 1 pixel. La performance obtenue est de 1.28% et notons que le résultat sans transformation (obtenu pour $A = 0$, sans épaissement ni translation) est 2.52%

4.2 Du mode de sélection

La taille de la solution est un facteur de limitation. Nous cherchons donc le critère de sélection qui donnera les meilleurs résultats avec un bon compromis de parcimonie. La figure 5 reporte les résultats qui nous ont conduit à choisir le mode autoactif. En effet, les modes de sélection aléatoire (force brute) et autoactif donnent les meilleures performances en test (premier graphique) tandis qu'entre ces deux modes, autoactif donne la solution la plus parcimonieuse (deuxième graphique). Nous garderons donc le mode autoactif.

4.3 Des effets de l'optimisation complète

L'étape d'optimisation complète de LASVM est la dernière étape de l'algorithme. Dans notre cas, il n'y a pas de dernier point de la base puisque l'on peut toujours en générer. Dans un premier temps nous nous sommes contentés d'éliminer cette étape. Nous

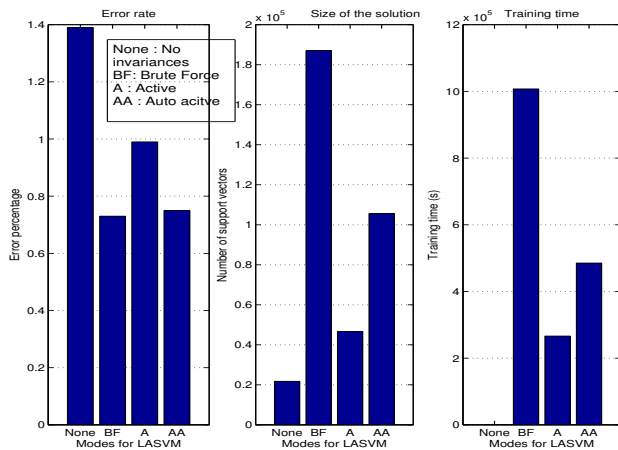


FIG. 5 – Ces graphiques comparent la performance (à gauche), la taille de la solution (centre) et le temps d'apprentissage (droite) pour différents modes de sélection de LASVM. La première colonne de chaque graphique reporte les résultats obtenus sans invariances, soit sur la base de donnée originale de 60000 points. Les autres colonnes donnent respectivement les résultats pour les modes aléatoires (force brute) actifs et autoactifs. Ces expériences sont conduites sur 100 transformations par point, soit 600000 points et selon les réglages obtenus précédemment. La sélection autoactive offre un bon compromis entre vitesse et performance.

avons remarqué qu'après un grand nombre de points d'apprentissage, le nombre de vecteurs supports diminuait. Il s'avère qu'après un certain temps, les nouveaux exemples présentés n'apportent plus d'information au classifieur, il n'y a donc plus l'étape de *process* qui ajoute des points à la solution, mais il y a toujours l'étape de *reprocess* qui continue à optimiser et à enlever des points. Dans un deuxième temps nous avons introduit une étape d'optimisation complète régulièrement au cours de l'apprentissage (tous les 600000 points). Nous avons observé le même phénomène mais plus tôt dans l'apprentissage. Se pose alors la question de la quantité d'optimisation nécessaire à chaque étape : est-il utile de moduler le nombre de *reprocess* par itération ?

4.4 Du nombre de reprocess

Le réglage de l'alternance entre *process* et *reprocess* est un point qui peut probablement améliorer les performance de LASVM. Le tableau 1 montre des résultats sur la qualité de la solution trouvée en fonction du nombre de *reprocess* effectué à chaque itération. Les résultats notés *nR/IP* désignent les expériences où *n reprocess* sont effectués à chaque fois qu'un *process* est fait. Les résultats notés *nR chaque* sont trouvés quand *n reprocess* sont fait à chaque fois qu'un nouveau point arrive, qu'il y ait ou non de phase de *process* pour lui. La principale conclusion de ces expériences est qu'il y a un compromis à faire et qu'il y a matière à investigation pour ce point. L'étude d'une automatisation de ce paramètre est en cours.

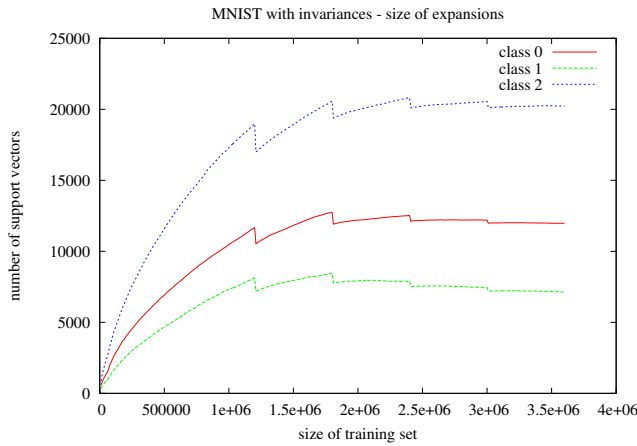


FIG. 6 – Cette figure montre l’évolution de la taille de la solution au cours de l’apprentissage. Ces résultats sont obtenus avec une sélection autoactive et en faisant régulièrement une phase de fin (optimisation complète). Chaque saut correspond à une phase d’optimisation complète. Nous remarquons ici que le nombre de vecteurs supports fini par décroître, ce qui peut être expliqué par la corrélation entre les exemples d’apprentissage.

	1R / 1P	2R / 1P	3R / 1P	4R / 1P	5R / 1P	1R chaque	2R chaque	3R chaque
Taille max	24531	21903	21436	20588	20029	23891	21660	20596
Points enlevés	1861	1258	934	777	537	1487	548	221
Prop. d’enlevés	7.6%	5.7%	4.3%	3.7%	2.6%	6.2%	2.5%	1.0%
Temps d’app.(sec)	1621	1548	1511	1482	1441	1857	1753	1685
Taux d’erreur	2.13%	2.08%	2.19%	2.09%	2.07%	2.06%	2.17%	2.13%

TAB. 1 – Effets des transformations sur les performances. Le tableau montre la comparaison entre différents compromis *process/reprocess*. Le nombre de *reprocess* consécutifs après chaque *process* varie, et de la même façon après chaque point, qu’il y ait eu ou pas une phase de *process*.

4.5 Des résultats globaux de LASVM pour les invariances sur MNIST

Le réglage pour les résultats finaux sont : noyau rbf avec une largeur de bande $\gamma = 0.006$ et $C = 1000$. Le coefficient de déformation est de $A = 2$. Nous ne touchons pas à l’épaississement des traits ($B = 0$). Le critère de sélection est le mode actif. Les meilleurs résultats obtenus sont 0.67%. De meilleures performances ont été publiées avec des réseaux à convolution ou en modifiant aussi le jeu de test (Simard *et al.*, 2003; Schölkopf & Smola, 2002). Cependant ces résultats sont aussi bons que les autres méthodes qui ne déforment en aucune façon les exemples de test. De plus, et c’est là notre principal propos, ils sont obtenus en entraînant 10 SVM binaires (1 contre tous) de 8 millions de points chacun en dimension 784. Le temps global d’apprentissage en test est de 8 jours sur une seule machine.

5 Conclusion

Cet article montre que SVM et grandes bases de données ne sont pas des notions incompatibles. En effet, grâce à des méthodes itératives de sélection d'exemples, nous avons construit un algorithme fonctionnant en ligne et rapidement. Nous présentons des résultats qui montrent qu'il est désormais possible d'utiliser les SVM pour résoudre des problèmes avec des millions d'exemples, même en relativement grande dimension (ici 784) et ce sur une seule machine. Cela est dû au caractère parcimonieux de la méthode. En effet, si dans le pire des cas, la complexité de la solution est d'ordre exponentiel, en pratique elle reste polynômiale en s , où s désigne le nombre de coefficients non nuls. Ainsi, c'est parce qu'il peut sélectionner efficacement de l'ordre de $s = 8000$ exemples sur les 8 millions présentés, qu'un classifieur peut traiter tant de points.

Références

- BORDES A. & BOTTOU L. (2005). The huller : a simple and efficient online svm. In *Machine Learning : ECML 2005*, Lecture Notes in Artificial Intelligence, LNAI 3720, p. 505–512 : Springer Verlag.
- BORDES A., ERTEKIN S., WESTON J. & BOTTOU L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, **6**, 1579–1619.
- BOTTOU L. (1998). Online algorithms and stochastic approximations. In D. SAAD, Ed., *Online Learning and Neural Networks*. Cambridge, UK : Cambridge University Press.
- BOTTOU L. & LECUN Y. (2004). Large scale online learning. In S. THRUN, L. SAUL & B. SCHÖLKOPF, Eds., *Advances in Neural Information Processing Systems 16*, Cambridge, MA : MIT Press.
- CORTES C. & VAPNIK V. (1995). Support vector networks. *Machine Learning*, **20**, pp 1–25.
- CRAMMER K., KANDOLA J. & SINGER Y. (2004). Online classification on a budget. In S. THRUN, L. SAUL & B. SCHÖLKOPF, Eds., *Advances in Neural Information Processing Systems 16*. Cambridge, MA : MIT Press.
- FREUND Y. & SCHAPIRE R. E. (1998). Large margin classification using the perceptron algorithm. In J. SHAVLIK, Ed., *Machine Learning : Proceedings of the Fifteenth International Conference*, San Francisco, CA : Morgan Kaufmann.
- FRIESS T., CRISTIANINI N. & CAMPBELL. C. (1998). The kernel Adatron algorithm : a fast and simple learning procedure for support vector machines. In J. SHAVLIK, Ed., *15th International Conf. Machine Learning*, p. 188–196 : Morgan Kaufmann Publishers.
- GENTILE C. (2001). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, **2**, 213–242.
- GRAEPEL T. & HERBRICH R. (2004). Invariant pattern recognition by semi-definite programming machines. In S. THRUN, L. SAUL & B. SCHÖLKOPF, Eds., *Advances in Neural Information Processing Systems 16*. Cambridge, MA : MIT Press.
- GRENANDER U. (1993). *General Pattern Theory*. Oxford University Press.
- LECUN Y., BOTTOU L., BENGIO Y. & HAFNER P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324. <http://yann.lecun.com/exdb/mnist/>.

- LEEN T. K. (1995). From data distributions to regularization in invariant learning. In G. TESAURO, D. TOURETZKY & T. LEEN, Eds., *Advances in Neural Information Processing Systems*, volume 7, p. 223–230 : The MIT Press.
- LI Y. & LONG P. M. (2002). The relaxed online maximum margin algorithm. *Mach. Learn.*, **46**(1-3), 361–387.
- LOOSLI G., CANU S., VISHWANATHAN S. & J.SMOLA A. (2005). Invariances in classification : an efficient svm implementation. In *ASMDA 2005 -Applied Stochastic Models and Data Analysis*.
- LOOSLI G., CANU S., VISHWANATHAN S., SMOLA A. J. & CHATTOPADHYAY M. (2004). Une boîte à outils rapide et simple pour les SVM. p. 113–128.
- NADAL J.-P. (1993). *Réseaux de neurones : de la physique à la psychologie*. Armand Colin, Collection 2a1.
- NEMIROVSKI A. (2005). Introduction to convex programming, interior point methods, and semi-definite programming. Machine Learning, Support Vector Machines, and Large-Scale Optimization Pascal Workshop.
- NOVIKOFF A. B. J. (1962). On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, p. 615–622 : Polytechnic Institute of Brooklyn.
- PLATT J. (1999). Fast training of support vector machines using sequential minimal optimization. In B. SCHÖLKOPF, C. J. C. BURGES & A. J. SMOLA, Eds., *Advances in Kernel Methods — Support Vector Learning*, p. 185–208, Cambridge, MA : MIT Press.
- SCHÖLKOPF B. & SMOLA A. J. (2002). *Learning with Kernels*. MIT Press.
- SCHÖLKOPF B., BURGES C. & VAPNIK V. (1996). Incorporating invariances in support vector learning machines. In J. V. C. VON DER MALSBERG, W. VON SEELEN & B. SENDHOFF, Eds., *Artificial Neural Networks — ICANN'96*, volume 1112, p. 47–52, Berlin : Springer Lecture Notes in Computer Science.
- SIMARD P., LECUN Y. & J. D. (1993). efficient pattern recognition using a new transformation distance. In S. HANSON, J. COWAN & L. GILES, Eds., *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann.
- SIMARD P., STEINKRAUS D. & PLATT J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, p. 958–962.
- SIMARD P. Y., LECUN Y., DENKER J. S. & VICTORRI B. (2000). Transformation invariance in pattern recognition – tangent distance and tangent propagation. *International Journal of Imaging Systems and Technology*, **11**(3).
- STEINWART I. (2004). Sparseness of support vector machines—some asymptotically sharp bounds. In S. THRUN, L. SAUL & B. SCHÖLKOPF, Eds., *Advances in Neural Information Processing Systems 16*. Cambridge, MA : MIT Press.
- WOOD J. (1996). Invariant pattern recognition : A review. *Pattern Recognition*, **29** Issue 1, 1–19.
- ZOUTENDIJK G. (1960). *Methods of Feasible Directions*. Elsevier.