Classification de grands ensembles de données avec un nouvel algorithme de SVM

Thanh-Nghi Do*, François Poulet**

*Equipe InSitu, INRIA Futurs, LRI, Bat.490, Université Paris Sud 91405 Orsay Cedex
Thanh-Nghi.Do@lri.fr
http://www.lri.fr/~dtnghi
**ESIEA-Ouest, 38, rue des Docteurs Calmette et Guérin, 53000 Laval
francois.poulet@esiea-ouest.fr
http://visu.egc.free.fr

Résumé. Le nouvel algorithme de boosting de Least-Squares Support Vector Machine (LS-SVM) que nous présentons vise à la classification de très grands ensembles de données sur des machines standard. Les méthodes de SVM et de noyaux permettent d'obtenir de bons résultats en ce qui concerne la précision mais la tâche d'apprentissage pour de grands ensembles de données demande une grande capacité mémoire et un temps relativement long. Nous présentons une extension de l'algorithme de LS-SVM proposé par Suykens et Vandewalle pour le boosting de LS-SVM. A cette fin, nous avons ajouté un terme de régularisation de Tikhonov et utilisé la formule de Sherman-Morrison-Woodbury pour traiter des ensembles de données ayant un grand nombre de dimensions. Nous l'avons ensuite étendu par application du boosting de LS-SVM afin de traiter des données ayant simultanément un grand nombre d'individus et de dimensions. Les performances de l'algorithme sont évaluées sur les ensembles de données de l'UCI, Twonorm, Ringnorm, Reuters-21578 et NDC sur une machine standard (PC-P4, 3GHz, 512 Mo RAM).

1 Introduction

Le volume de données stocké double actuellement tous les 9 mois (Lyman et al, 2003) et donc le besoin d'extraction de connaissances dans les grandes bases de données est de plus en plus important (Fayyad et al, 2004). La fouille de données (Fayyad et al, 1996) est confrontée au challenge de traiter de grands ensembles de données pour identifier des connaissances nouvelles, valides, potentiellement utilisables et compréhensibles. Elle utilise différents algorithmes pour la classification, la régression, le clustering ou les associations.

Nous nous intéressons plus particulièrement ici aux algorithmes de Séparateurs à Vaste Marge (SVM ou Support Vector Machine) proposé par (Vapnik, 1995) car ils se montrent particulièrement efficaces pour la classification, la régression ou la détection de nouveauté. On peut trouver de nombreuses applications des SVM comme la reconnaissance de visages, la catégorisation de textes ou la bioinformatique (Guyon, 1999). L'approche est systématique et motivée par la théorie de l'apprentissage statistique. Les SVM sont les plus connus parmi une classe d'algorithmes utilisant les méthodes de noyau (Cristianini et al, 2000). Les SVM

et les méthodes de noyaux permettent de construire des modèles précis et deviennent des outils de fouille de données de plus en plus populaires. Mais malgré ces qualités, les SVM ne peuvent pas traiter facilement des données volumineuses. Les solutions des SVM sont obtenues par résolution d'un programme quadratique, le coût de calcul d'une approche de SVM est au moins d'une complexité égale au carré du nombre d'individus de l'ensemble d'apprentissage et la quantité de mémoire nécessaire les rend impossible à utiliser sur de grands ensembles de données. Il y a donc besoin de permettre le passage à l'échelle de ces algorithmes pour traiter de grands ensembles de données sur des machines standard. Une heuristique possible pour améliorer l'apprentissage à l'aide de SVM est de décomposer le programme quadratique original en une série de plus petits problèmes (Boser et al, 1992), (Chang et al, 2003), (Osuna et al, 1997), (Platt, 1999). Les méthodes d'apprentissage incrémental (Cauwenberghs et al, 2001), (Do et Poulet, 2006), (Do et Poulet, 2003), (Fung et Mangasarian, 2002), (Poulet et Do, 2004), (Syed et al, 1999) permettent de traiter de grands ensembles de données par mise à jour des solutions partielles en augmentant l'ensemble d'apprentissage sans avoir à charger l'ensemble de données total en mémoire. Les algorithmes parallèles et distribués (Do et Poulet, 2006), (Poulet et Do, 2004) utilisent des machines connectées par internet pour améliorer le temps d'exécution de l'apprentissage de grands ensembles de données. Les algorithmes d'apprentissage actif (Do et Poulet, 2005), (Tong et Koller, 2000) permettent de choisir un sous-ensemble d'individus (ensemble actif) pour la construction du modèle. Nous présentons un nouvel algorithme de boosting de LS-SVM pour la classification de grands ensembles de données sur des machines standard. L'algorithme de LS-SVM proposé par (Suykens et Vandewalle, 1999) effectue un changement de la contrainte d'inégalité en égalité dans la résolution du problème d'optimisation permettant d'obtenir la solution par résolution d'un système d'équations linéaires au lieu du programme quadratique. Ce nouvel algorithme est donc beaucoup plus rapide en temps d'exécution. Nous avons étendu cet algorithme pour construire un nouvel algorithme de SVM incrémental, parallèle et distribué permettant de traiter des ensembles de données ayant de très grands nombres d'individus. Puis nous avons ajouté un terme de régularisation de Tikhonov (Tikhonov, 1943) et utilisé la formule de Sherman-Morrison-Woodbury (Golub et Van Loan, 1996) pour permettre au LS-SVM de traiter des ensembles de données ayant un très grand nombre de dimensions. Enfin nous avons appliqué la technique du boosting au LS-SVM pour obtenir un algorithme permettant la classification d'ensembles de données ayant simultanément un grand nombre d'individus et de dimensions. Les performances de l'algorithme sont évaluées sur des ensembles de données de l'UCI (Blake et Merz, 1998), Twonorm, Ringnorm (Delve, 1996), Reuters-21578 (Lewis, 1997) et NDC (Musicant, 1998). Les résultats sont comparés avec ceux obtenus avec LibSVM (Chang et Lin, 2003).

Le paragraphe 2 présente brièvement l'algorithme de LS-SVM, le paragraphe 3 décrit l'algorithme incrémental de LS-SVM. Dans le paragraphe 4 nous présentons l'algorithme de boosting de LS-SVM puis les résultats des tests numériques dans le paragraphe 5 avant la conclusion et les travaux futurs.

Quelques notations sont utilisées dans cet article. Tous les vecteurs sont représentés par des matrices colonne. Le produit scalaire de deux vecteurs x et y est noté x.y. La norme d'un vecteur v est $||v||$. La matrice A (de taille mxn) contient l'ensemble des m individus en dimension n. La classe (+1 ou –1) est stockée dans la matrice diagonale D (de taille mxm). e est un vecteur colonne de 1. w et b sont les coefficients et le scalaire de l'hyperplan, z est la variable de ressort et C est une constante positive. I représente la matrice identité.

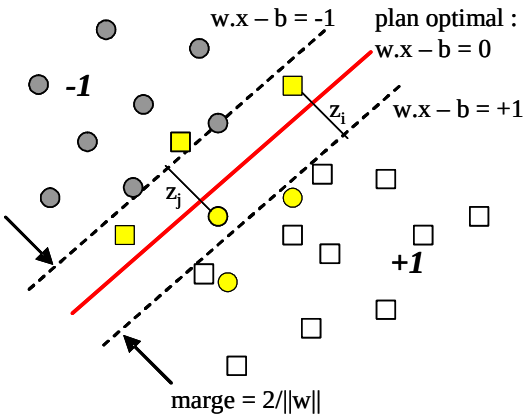

FIG. 1 – *Séparation linéaire des données en deux classes.*

2 L'algorithme de LS-SVM

Considérons une tâche de classification binaire linéaire comme représentée sur la figure 1 avec m points x_i ($i=1..m$) dans l'espace de R^n, representés par la matrice A avec les étiquettes de classe (+1 ou –1) stockées dans la matrice diagonale D. L'algorithme de SVM cherche le meilleur hyperplan de séparation des données (meilleur au sens du plus éloigné possible des deux classes). Cela revient à maximiser la marge qui est la distance entre les plans supports des deux classes. Le plan support de la classe +1 [resp. –1] sépare tous les individus de la classe +1 [resp. –1] des autres. Ceci peut s'écrire sous la forme suivante (1) :

$$D(Aw - eb) \geq e \quad (1)$$

La marge entre les plans support est 2/||w|| (où ||w|| représente la norme du vecteur w). Dans le cas non linéairement séparable, les contraintes doivent être relaxées pour permettre à un point d'être du mauvais côté du plan support de sa classe, une variable de ressort est alors ajoutée dans la partie gauche de l'équation 1. Ensuite tout point du mauvais côté de son plan support est considéré comme une erreur et à une valeur de z positive ($z_i > 0$).

Ensuite l'algorithme de SVM doit simultanément maximiser la marge et minimiser les erreurs. La formulation standard de l'algorithme de SVM avec un noyau linéaire est alors le programme quadratique (2):

$$\min \Psi(w, b, z) = (1/2)\ ||w||^2 + cz$$
$$\text{avec: } D(Aw - eb) + z \geq e \quad (2)$$

où z représente la variable de ressort et c est une constante positive pour régler les erreurs et la taille de la marge.

L'hyperplan (w,b) obtenu est la solution du programme quadratique (2). Ensuite la classification d'un nouvel individu x se base sur sa position par rapport à l'hyperplan obtenu classe(x) = signe (w.x-b). Les algorithmes de SVM peuvent utiliser d'autres types de fonctions pour la classification comme par exemple une fonction polynomiale de degré d, une fonction RBF (Radial Basis Function) ou une sigmoïde. Le passage de cas linéaire au cas non-linéaire se fait par l'utilisation d'une fonction de noyau à la place du produit scalaire dans

l'équation (2). Plus de détails sur les SVM et les méthodes de noyaux peuvent être trouvés dans (Cristianini et Shawe-Taylor, 2000). La solution des SVM est obtenue par résolution d'un programme quadratique donc le coût en temps d'exécution est au moins proportionnel au carré du nombre d'individus et la place mémoire nécessaire les rend incapables de traiter des ensembles de données très volumineux. L'algorithme de LS-SVM proposé par (Suykens et Vandewalle, 1999) utilise une égalité au lieu de l'inégalité dans le problème d'optimisation (2) avec la fonction Ψ suivante :

- minimisation de l'erreur avec $(c/2)||z||^2$ sous la contrainte : $D(Aw - eb) + z = e$

En substituant z dans la fonction objectif Ψ du programme quadratique (2), nous obtenons alors (3) :

$$min\ \Psi(w, b) = (1/2)||w||^2 + (c/2)||e - D(Aw - eb)||^2 \quad (3)$$

Pour résoudre ce problème d'optimisation, nous calculons les dérivées partielles en w et b. Cela nous donne donc le système linéaire de (n+1) inconnues $(w_1, w_2, \ldots, w_n, b)$ suivant :

$$\Psi'(w) = cA^T(Aw - eb - De) + w = 0 \quad (4)$$

$$\Psi'(b) = ce^T(-Aw + eb + De) = 0 \quad (5)$$

Les équations (4) et (5) peuvent être réécrites sous la forme suivante :

$$\left[w_1\ w_2\ w_3 \ldots w_n\ b\right]^T = \left(\frac{1}{c} I^\circ + E^T E\right)^{-1} E^T De \quad (6)$$

où E=[A -e] (juxtaposition de la matrice A avec une colonne de –1), I° est la matrice diagonale identité dont le dernier élément est 0.

La formulation du LS-SVM (6) nécessite la résolution d'un système linéaire à n+1 inconnues au lieu du programme quadratique (2), donc si le nombre de dimensions de l'ensemble de données est inférieur à 10^5, l'algorithme de LS-SVM (tableau 1) est capable de traiter un très grand nombre d'individus en un temps restreint sur une machine standard. Les tests numériques ont montré des résultats satisfaisants en comparaison à des algorithmes comme libSVM mais en se montrant beaucoup plus rapide. Par exemple, la classification d'un million de points en dimension 20 est effectuée en 1,3 seconde sur un PC (Pentium IV, 3GHz, 512Mo RAM). Pour traiter le cas de la classification non-linéaire, il faut remplacer la matrice A en entrée de l'algorithme par la matrice de noyau non linéaire K, par exemple :

- une fonction polynomiale de degré d de deux points x_i, x_j : $K[i,j] = (x_i.x_j + 1)^d$
- une fonction RBF de deux points x_i, x_j : $K[i,j] = exp(-\gamma|| x_i - x_j||^2)$

L'algorithme de LS-SVM utilisant une matrice de noyau nécessitera aussi un temps de calcul et une place mémoire importante.

Entrée :
- l'ensemble de données *A, D*
- le paramètre *c* (contrôle la marge et les erreurs)
Apprentissage :
- créer la matrice *E = [A -e]*
- résoudre le système linéaire (6)
- obtenir le plan (*w, b)*
Classification d'un nouvel individu *x* :
f(x) = signe (x.w - b)

TAB. 1 – *Algorithme de LS-SVM linéaire.*

3 LS-SVM incrémental

Bien que l'algorithme de LS-SVM soit efficace et rapide pour la classification de grands ensembles de données, il nécessite de charger l'ensemble des données en mémoire. Avec de très grands ensembles de données, par exemple un milliard de points en dimension 20, l'espace mémoire nécessaire est de 80Go. La plupart des algorithmes de classification actuels sont confrontés à ce problème. Nous allons nous intéresser à ce cas de figure, le traitement de très grands ensembles de données. Les algorithmes de classification incrémentaux (Do et Poulet, 2003, 2006), (Poulet et Do, 2004) sont une méthode très efficace pour traiter de très grands ensembles de données car ils ne nécessitent pas le chargement de la totalité des données en mémoire : seul un petit bloc de données est considéré à un instant donné et le modèle est construit par modifications successives.

3.1 LS-SVM incrémental en ligne

Supposons que nous avons à traiter un ensemble de données ayant un très grand nombre de points et un nombre plus restreint de dimensions, nous pouvons décomposer cet ensemble de données en blocs de lignes A_i, D_i. La version incrémentale en ligne de LS-SVM va calculer la solution de l'équation (6) de manière incrémentale. Considérons un exemple simple avec un ensemble de données décomposé en deux blocs de lignes A_1, D_1 et A_2, D_2 :

$$A=\begin{bmatrix}A_1\\A_2\end{bmatrix}, D=\begin{bmatrix}D_1 & 0\\0 & D_2\end{bmatrix}, e=\begin{bmatrix}e_1\\e_2\end{bmatrix}, E=\begin{bmatrix}A & -e\end{bmatrix}=\begin{bmatrix}A_1 & -e_1\\A_2 & -e_2\end{bmatrix}=\begin{bmatrix}E_1\\E_2\end{bmatrix}$$

Nous illustrons maintenant comment s'effectue le calcul de la solution du système d'équation linéaire (6) :

$$E^T De=\begin{bmatrix}E_1\\E_2\end{bmatrix}^T\begin{bmatrix}D_1 & 0\\0 & D_2\end{bmatrix}\begin{bmatrix}e_1\\e_2\end{bmatrix}=\begin{bmatrix}E_1{}^T & E_2{}^T\end{bmatrix}\begin{bmatrix}D_1 & 0\\0 & D_2\end{bmatrix}\begin{bmatrix}e_1\\e_2\end{bmatrix}$$

$$E^T De = E_1{}^T D_1 e_1 + E_2{}^T D_2 e_2 \quad (7)$$

$$E^T E=\begin{bmatrix}E_1\\E_2\end{bmatrix}^T\begin{bmatrix}E_1\\E_2\end{bmatrix}=\begin{bmatrix}E_1{}^T & E_2{}^T\end{bmatrix}\begin{bmatrix}E_1\\E_2\end{bmatrix} \quad (8)$$

$$E^T E = E_1{}^T E_1 + E_2{}^T E_2 \quad (9)$$

$$\begin{bmatrix}w_1 & w_2 & \dots & w_n & b\end{bmatrix}^T=\left(\frac{1}{c}I^\circ+\sum_{i=1}^{2}E_i{}^T E_i\right)^{-1}\sum_{i=1}^{2}E_i{}^T D_i e_i \quad (10)$$

A partir des équations (7), (9) et (10), on peut en déduire l'équation (11) de l'algorithme de LS-SVM incrémental en ligne, avec un ensemble de données décomposé en k blocs de lignes A_1, D_1, …, A_k, D_k :

$$\begin{bmatrix}w_1 & w_2 & \dots & w_n & b\end{bmatrix}^T=\left(\frac{1}{c}I^\circ+\sum_{i=1}^{k}E_i{}^T E_i\right)^{-1}\sum_{i=1}^{k}E_i{}^T D_i e_i \quad (11)$$

L'algorithme incrémental en ligne de LS-SVM du tableau 2 peut donc classifier des

données très volumineuses sur une machine standard. La précision de l'algorithme est exactement la même que celle de l'algorithme original. Si le nombre de dimensions de l'ensemble de données est inférieur à 10,000 alors l'algorithme est tout à fait capable de classifier des ensembles de données de plusieurs milliards d'individus sur une machine standard. Entre deux étapes successives de l'algorithme il n'est nécessaire de conserver en mémoire qu'une matrice de taille (n+1)x(n+1) et deux vecteurs de taille mx(n+1) (m étant le nombre d'individus et n le nombre de dimensions). Les tests numériques ont montré que l'algorithme est capable de classifier un ensemble d'un milliard d'individus en dimension 20 en 21 minutes sur un PC (Pentium IV, 3GHz, 512Mo RAM).

Entrée :
- les k blocs de données $A_1, D_1, \ldots, A_k, D_k$
- le paramètre c (contrôler la marge et des erreurs)

Apprentissage :
- initialiser : $E^TE = 0$, $d = E^TDe = 0$
- pour i de 1 à k faire
  - charger A_i et D_i en mémoire
  - calculer $E^TE = E^TE + E_i^TE_i$
  - calculer $d = d + d_i$ (où $d_i = E_i^TD_ie_i$)

  finpour
- résoudre le système linéaire (11)
- obtenir le plan (w, b)

Classification d'un nouvel exemple x :

f(x) = signe (x.w - b)

TAB. 2 – *Algorithme incrémental en ligne de LS-SVM linéaire.*

3.2 LS-SVM incrémental en colonne

Certaines applications comme la bioinformatique ou la fouille de textes nécessitent de traiter des données ayant un nombre très important de dimensions et un nombre d'individus plus réduit. Dans ce cas la matrice de taille (n+1)x(n+1) est trop importante et la résolution du système à (n+1) inconnues nécessite un temps de calcul élevé. Pour adapter l'algorithme à ce type de données nous avons appliqué la formule de Sherman-Morrison-Woodbury au système d'équations (6). Mais ce faisant nous avions une matrice singulière à inverser (I°). Nous avons donc ajouté un terme de régularisation de Tikhonov, ce qui est la méthode la plus couramment utilisée pour résoudre ce genre de problème. Avec le terme de Tikhonov ($\delta>0$) ajouté à (6) nous obtenons alors le système d'équations (12)

$$\left[w_1\ w_2\ w_3 \ldots w_n\ b\right]^T = \left(\frac{1}{c}I^\circ + \delta I + E^TE\right)^{-1} E^TDe \qquad (12)$$

Ce système peut être réécrit sous la forme suivante (13) :

$$\left[w_1\ w_2\ w_3 \ldots w_n\ b\right]^T = \left(H + E^TE\right)^{-1} E^TDe \qquad (13)$$

où H représente la matrice (n+1)x(n+1) diagonale dont le (n+1)ème terme est δ et les autres termes valent $(1/c)+\delta$. Ensuite nous appliquons la formule de Sherman-Morrison-Woodbury (14) dans la partie droite du système (13) :

$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^TA^{-1}U)^{-1}V^TA^{-1}$ (14)

$$\Rightarrow [w_1\ w_2\ w_3 \dots w_n\ b]^T = \left(H + E^T E\right)^{-1} E^T De$$

$$= \left(H^{-1} - H^{-1}E^T\left(I + EH^{-1}E^T\right)^{-1} EH^{-1}\right)E^T De$$

$$= \left(H^{-1}E^T - H^{-1}E^T\left(I + EH^{-1}E^T\right)^{-1} EH^{-1}E^T\right)De$$

$$= H^{-1}E^T\left(I - \left(I + EH^{-1}E^T\right)^{-1} EH^{-1}E^T\right)De$$

Nous obtenons le nouveau système d'équations (15) :

$$[w_1\ w_2\ w_3 \dots w_n\ b]^T = H^{-1}E^T\left(I - \left(I + EH^{-1}E^T\right)^{-1} EH^{-1}E^T\right)De \quad (15)$$

Ce nouveau système nécessite l'inversion de la matrice ($I + EH^{-1}E^T$) (de taille mxm) au lieu de la matrice ($I+E^TE$) de taille (n+1)x(n+1). Cette nouvelle formulation permet de traiter facilement des ensembles de données ayant un nombre de dimensions très important.

Nous avons ensuite construit une version incrémentale en colonne de l'algorithme de manière analogue à la version incrémentale en ligne. L'ensemble de données est découpé en blocs de colonnes E_i et le calcul du terme $EH^{-1}E^T$s'effectue de manière incrémentale. Entre deux étapes de calcul, nous ne conservons en mémoire que la matrice $EH^{-1}E^T$ de taille mxm.

4 Boosting de LS-SVM

Pour pouvoir traiter des ensembles de données ayant simultanément un grand nombre d'individus et de colonnes il y a au moins deux problèmes à résoudre : le temps d'apprentissage devient rapidement déraisonnable et la quantité de mémoire nécessaire dépasse les capacités de mémoire des machines courantes. Bien que les algorithmes incrémentaux de LS-SVM puissent efficacement charger en mémoire des petits blocs de données successifs, ils nécessitent l'inversion de matrice de taille (mxm) ou (n+1)x(n+1). La quantité de mémoire nécessaire et le coût de calcul deviennent trop importants. Pour pouvoir traiter des ensembles de données très volumineux, nous avons donc appliqué l'approche du boosting au LS-SVM de manière analogue à (Do et Poulet, 2004). Cette solution présente deux avantages : résoudre le problème de passage à l'échelle et conserver la précision de l'algorithme original. Plus de détails sur le boosting peuvent être trouvés dans (Freund et Schapire, 1999) ou sur le site www.boosting.org. Nous décrivons brièvement ici le mécanisme de boosting de LS-SVM. Dans les années 1990, Freund et ses collègues ont introduit le boosting pour améliorer la précision des algorithmes d'apprentissage. La méthode de boosting consiste à utiliser k fois un algorithme d'apprentissage basique en se concentrant à chaque étape sur les erreurs commises à l'étape précédente. Pour ce faire, il est nécessaire de tenir à jour une distribution de poids sur l'ensemble des individus de l'apprentissage. Initialement, tous les poids sont identiques et à chaque étape du boosting le poids des individus mal classifiés est augmenté pour obliger l'algorithme à les prendre en compte de manière plus significative. Nous considérons l'algorithme de LS-SVM comme l'algorithme d'apprentissage basique et à chaque étape du boosting nous échantillonnons un sous-ensemble d'individus en tenant compte de la distribution des poids. Il faut remarquer que le LS-SVM n'effectue l'apprentissage que sur ce sous-ensemble d'individus (de taille moindre que l'ensemble original). La taille de l'échantillon est inversement proportionnelle aux nombres d'étapes du boosting. Les algorithmes de LS-SVM incrémentaux en ligne ou colonne peuvent ainsi être adaptés au traitement de très grands ensembles de données (à la

fois en nombre d'individus et de dimensions), avec de bons résultats en précision et besoin en mémoire.

5 Quelques résultats

Nous avons développé le programme en C/C++ sous Linux en utilisant la librairie Lapack++ (Dongarra et al, 1993) pour bénéficier de bonnes performances en calcul matriciel. Le programme peut donc classifier de très grands ensembles de données efficacement. Nous allons en présenter une évaluation prenant en compte les critères suivants : la précision, le temps d'apprentissage et la place mémoire requise. Nous avons sélectionné 3 ensembles de données artificiels générés par Twonorm, Ringnorm et NDC et 8 ensembles de données de l'UCI. Les caractéristiques de ces ensembles sont décrites dans le tableau 3 (les attributs catégoriques des ensembles Adult et Mushroom ont été convertis en binaire).

Nous avons utilisé le nouvel algorithme de boosting de LS-SVM (Boost-LS-SVM) et LibSVM (l'un des algorithmes de SVM les plus efficaces) pour effectuer la classification sur un PC (Pentium IV, 3GHz et 512Mo RAM).

	Classes	Individus	Dimensions	Protocole de test
Twonorm	2	7400	20	300 Trn - 7100 Tst
Ringnorm	2	7400	20	300 Trn - 7100 Tst
Pima	2	768	8	10-fold
Bupa	2	345	6	10-fold
Ionosphere	2	351	34	10-fold
Mushroom	2	8124	22	10-fold
Tic-tac-toe	2	958	9	10-fold
Adult	2	48842	14	32561 Trn - 16281 Tst
Reuters-21578	135	10789	29406	7770 Trn - 3019 Tst
Forest cover type	7	581012	54	10-fold
Ndc	2	55000	20000	50000 Trn – 5000 Tst

TAB. 3 – *Description des ensembles de données.*

Les 8 premiers petits ensembles de données sont utilisés pour comparer la précision et le temps d'apprentissage (tableau 4). Boost-LS-SVM obtient de meilleures précisions dans tous les cas sauf un et un meilleur temps d'apprentissage dans la moitié des cas. On peut remarquer que le temps d'apprentissage de libSVM croit de manière très importante lorsque la taille des fichiers augmente. Par exemple sur l'ensemble de données Adult, Boost-LS-SVM est 190 fois plus rapide que libSVM.

Reuters-21578 est un ensemble de données réputé pour la catégorisation de textes. Nous avons utilisé Bow (McCallum, 1998) en prétraitement de ces données. Chaque document est vu comme un vecteur de mots, nous avons obtenu 29406 mots (dimensions) sans sélection de dimensions. Nous avons effectué la classification des 10 classes les plus nombreuses. Cet ensemble de données ayant plus de deux classes nous avons utilisé l'approche one-against-all. Les résultats sont présentés dans le tableau 5 avec la moyenne de la précision et du rappel (breakeven point) pour les 10 catégories. Boost-LS-SVM a obtenu une meilleure précision pour 9 des 10 catégories mais le temps d'exécution du Boost-LS-SVM est deux fois plus long

que celui de LibSVM car le nombre d'itérations est important pour arriver à la même précision.

	Temps (secs)		Précision (%)	
	Boost-LS-SVM	LibSVM	Boost-LS-SVM	LibSVM
Twonorm	0,05	**0,03**	**97,09**	97,01
Ringnorm	**0,11**	0,18	**75,07**	73,82
Pima	**0,07**	**0,07**	**78,03**	76,82
Bupa	0,03	**0,02**	**70,59**	68,41
Ionosphere	0,08	**0,04**	**88,57**	88,32
Mushroom	**1,22**	3,05	**100,00**	**100,00**
Tic-tac-toe	**0,07**	0,25	**66,00**	65,34
Adult	**10,28**	1977,59 (~33 min)	85,08	**85,29**

TAB. 4 – *Performances en terme de temps d'exécution et de taux de précision.*

	Précision (%)		Temps (secs)	
	Boost-LS-SVM	LibSVM	Boost-LS-SVM	LibSVM
Earn	**98,41**	98,02	**6,74**	7,52
Acq	**96,57**	95,66	**6,77**	8,62
Money-fx	**79,49**	75,72	6,14	**5,92**
Grain	**90,75**	89,33	7,88	**4,05**
Crude	**89,83**	86,62	6,44	**4,56**
Trade	**78,18**	77,46	6,48	**4,58**
Interest	**78,32**	75,57	10,66	**6,95**
Ship	**84,60**	83,00	11,37	**4,29**
Wheat	**86,38**	85,58	11,28	**2,83**
Corn	88,97	**88,99**	5,02	**3,03**

TAB. 5 – *Résultats sur 10 catégories de l'ensemble de données Reuters-21578.*

Deux grands ensembles de données sont utilisés pour évaluer le temps d'exécution et la quantité de mémoire nécessaire aux algorithmes. LibSVM nécessite de charger la totalité de l'ensemble de données en mémoire, nous avons donc étendu la capacité de la RAM. Pour l'ensemble de données Forest Cover Type, nous avons effectué la classification des deux classes les plus nombreuses (Spruce-Fire : 211840 individus et Lorgepole-Pine : 283301 individus en dimension 24). Nous avons généré un ensemble de données de 55000 individus en dimension 20000 (2 classes) avec le programme NDC. Pour les deux ensembles de données, LibSVM n'a pu donner de résultat : pour Forest Cover Type, le programme a tourné pendant 21 jours et pour le second, il n'a pas pu être chargé en mémoire (4Go).

Boost-LS-SVM n'a utilisé que 512Mo de mémoire mais il a du relire les données à chaque étape de boosting (96% du temps est ainsi passé à charger les données en mémoire vive). Les résultats présentés dans le tableau 6 montrent que Boost-LS-SVM est capable d'effectuer la classification d'ensembles de données ayant simultanément un grand nombre d'individus et de dimensions sur une machine standard dans un temps raisonnable.

	RAM (MB)	Précision (%)	Temps d'exécution (min)
Forest covertype	19	77,41	18.2
Ndc	193	81,18	1 192,8 (~ 20h)

TAB. 6 - *Performances des algorithmes sur les grands ensembles de données.*

6 Conclusion et perspectives

Nous avons présenté un nouvel algorithme de boosting de LS-SVM capable d'effectuer la classification de grands ensembles de données sur des machines standard. L'idée principale est d'étendre l'algorithme récent de Suykens et Wandewalle pour en construire une version incrémentale et un boosting de LS-SVM. La précision des nouveaux algorithmes est exactement la même que celle de l'algorithme original. La complexité de la version incrémentale en lignes est linéaire en nombre d'individus. Si le nombre de dimensions est suffisamment restreint (inférieur à 10000), il permet de classifier plusieurs milliards de données sur un simple PC. Quelques applications comme la bioinformatique ou la fouille de texte utilisent des données dont le nombre de dimensions est très important et le nombre d'individus plus faible, nous avons ajouté un terme de régularisation de Tikhonov et utilisé la formule de Sherman-Morrison-Woodbury pour construire la version incrémentale en colonne de l'algorithme de LS-SVM et traiter les ensembles de données ayant un grand nombre de dimensions. Puis nous avons étendu ces algorithmes en utilisant la technique du boosting pour la classification d'ensembles de données ayant simultanément un grand nombre de dimensions et d'individus. Les résultats des tests numériques montrent que le nouvel algorithme de boosting de LS-SVM est rapide et de bonne précision. Il permet le passage à l'échelle et obtient de bons taux de précision en comparaison à libSVM (l'un des algorithmes de SVM les plus efficaces). Pour des petits ensembles de données il présente un bon taux de précision et une bonne rapidité d'exécution. Pour des ensembles de données de très grandes tailles (à la fois en nombre de dimensions et d'individus), il a montré ses possibilités avec un bon taux de précision.

Une première extension de ces travaux va consister à étendre cet algorithme pour en faire une version parallèle et distribuée sur un ensemble de machines. Cette extension permettra d'améliorer le temps de la tâche d'apprentissage. Une seconde sera de proposer une nouvelle approche pour la classification non linéaire.

Remerciements. Nous tenons à remercier vivement Jason Rennie du MIT pour son aide sur la préparation de l'ensemble de données Reuters-21578.

Références

Blake, C. and C. Merz (1998). UCI Repository of Machine Learning Databases, http://www.ics.uci.edu/~mlearn/MLRepository.html.

Boser, B., I. Guyon, and V. Vapnik (1992). An Training Algorithm for Optimal Margin Classifiers, in proc. of 5th ACM Annual Workshop on Computational Learning Theory, Pittsburgh, Pennsylvania, 144-152.

Cauwenberghs, G. and T. Poggio (2001). Incremental and Decremental Support Vector Machine Learning, in *Advances in Neural Information Processing Systems*, MIT Press, 13:409-415.

Chang, C-C. and C-J. Lin (2003). LIBSVM -- A Library for Support Vector Machines, http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Cristianini, N. and J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press.

Delve (1996). Data for evaluating learning in valid experiments, http://www.cs.toronto.edu-/~delve.

Do, T-N. and F. Poulet (2006). Classifying one billion data with a new distributed SVM algorithm, in proc. of RIVF'06, 4th IEEE International Conference on Computer Science, Research, Innovation and Vision for the Future, Ho Chi Minh, Vietnam, 59-66.

Do, T-N. and F. Poulet (2003). Incremental SVM and Visualization Tools for Bio-medical Data Mining, in proc. of Workshop on Data Mining and Text Mining in Bioinformatics, ECML/PKDD'03, Cavtat-Dubrovnik, Croatia, 14-19.

Do, T-N. and F. Poulet (2004). Towards High Dimensional Data Mining with Boosting of PSVM and Visualization Tools, in proc. of ICEIS'04, 6th Int. Conf. on Entreprise Information Systems, 2:36-41, Porto, Portugal.

Do, T-N. and F. Poulet (2005). Mining Very Large Datasets with SVM and Visualization, in proc. of ICEIS'05, 7th Int. Conf. on Entreprise Information Systems, 2:127-134, Miami, USA.

Dongarra, J., R. Pozo and D. Walker (1993). LAPACK++: a design overview of object-oriented extensions for high performance linear algebra, in proc. of *Supercomputing'93*, *IEEE Press*, 162-171.

Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth (1996). From Data Mining to Knowledge Discovery in Databases, *AI Magazine*, 17(3):37-54.

Fayyad, U., G. Piatetsky-Shapiro, and R. Uthurusamy (2004). Summary from the KDD-03 Panel – Data Mining: The Next 10 Years, in *SIGKDD Explorations*, 5(2):191-196.

Freund, Y. and R. Schapire (1999). A Short Introduction to Boosting, in *Journal of Japanese Society for Artificial Intelligence*, 14(5):771-780.

Fung, G. and O. Mangasarian (2002). Incremental Support Vector Machine Classification, in proc. of the 2nd SIAM Int. Conf. on Data Mining SDM'2002 Arlington, Virginia, USA.

Golub, G. and C. Van Loan (1996). *Matrix Computations*, John Hopkins University Press, Balti-more, Maryland.

Guyon, I. (1999). Web Page on SVM Applications, http://www.clopinet.com/isabelle/-Projects/SVM/applist.html.

Lewis, D. (1997). Reuters-21578 Text Classification Test Collection, http://www.david-dlewis.com/resources/testcollections/reuters21578/.

Lyman, P., H-R. Varian, K. Swearingen, P. Charles, N. Good, L. Jordan, and J. Pal (2003). How much information, http://www.sims.berkeley.edu/research/projects/how-much-info-2003/.

McCallum, A. (1998). Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering, http://www-2.cs.cmu.edu/~mccallum/bow.

Musicant, D. (1998). NDC : Normally Distributed Clustered Data sets, http://www.cs.wisc-.edu/~musicant/data/ndc/.

Osuna, E., R. Freund, and F. Girosi (1997). An Improved Training Algorithm for Support Vector Machines, in *Neural Networks for Signal Processing VII*, J. Principe, L. Gile, N. Morgan, and E. Wilson Eds., 276-285.

Platt, J. (1999). Fast Training of Support Vector Machines Using Sequential Minimal Optimization, in *Advances in Kernel Methods -- Support Vector Learning*, B. Schoelkopf, C. Burges, and A. Smola Eds., 185-208.

Poulet, F. and T-N. Do (2004). Mining Very Large Datasets with Support Vector Machine Algorithms, in *Enterprise Information Systems V*, Camp O., Filipe J., Hammoudi S. And Piattini M. Eds., Kluwer Academic Publishers, 177-184.

Suykens, J. and J. Vandewalle (1999). Least Squares Support Vector Machines Classifiers, in *Neural Processing Letters*, 9(3):293-300.

Syed, N., H. Liu, and K. Sung (1999). Incremental Learning with Support Vector Machines, in proc. of the 6th *ACM SIGKDD* Int. Conf. on KDD'99, San Diego, USA.

Tikhonov, A-N. (1943). On the stability of inverse problems, *Dokl. Akad. Nauk SSSR*, 39(5):195-198.

Tong, S. and D. Koller (2000). Support Vector Machine Active Learning with Applications to Text Classification, in proc. of *ICML'00*, the 17th Int. Conf. on Machine Learning, 999-1006, Stanford, USA.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*, Springer-Verlag, New York.

Summary

The new Boosting of Least Squares Support Vector Machine (LS-SVM) algorithm proposed in this paper aims at classifying very large datasets on standard personal computers (PCs). SVM and kernel related methods have shown to build accurate models but the learning task usually needs a quadratic program so that the learning task for large datasets requires large memory capacity and long time. We extend the recent LS-SVM proposed by Suykens and Vandewalle for building a new boosting of LS-SVM algorithm. We added a Tikhonov regularization term and also used the Sherman-Morrison-Woodbury formula to adapt the LS-SVM to process datasets with a very large number of dimensions. We have extended this idea by applying boosting to LS-SVM for mining massive datasets with simultaneously very large number of datapoints and dimensions. We have evaluated its performance on UCI, Twonorm, Ringnorm, Reuters-21578 and NDC datasets on a PC (3 GHz Pentium IV, 512 MB RAM).