Fouille de Grands Ensembles de Données avec un Boosting de Proximal SVM

Thanh-Nghi Do*, François Poulet*

*ESIEA Recherche
38, rue des Docteurs Calmette et Guérin
Parc Universitaire de Laval - Changé
53000 Laval
(dothanh, poulet)@esiea-ouest.fr

Résumé. Les SVM (support vector machines) ont montré leur efficacité dans plusieurs domaines d'application. L'apprentissage des SVM se ramène à résoudre un programme quadratique, dont la mise en œuvre est en général coûteuse en temps. Une reformulation plus récente des SVM (proximal SVM), proposée par Fung et Mangasarian, ne nécessite que la résolution d'un système linéaire, cet algorithme de PSVM est plus efficace et permet de traiter des données dont le nombre d'individus est très important (10^9) et le nombre d'attributs plus restreint (10^4). Nous proposons d'utiliser la formule de Sherman-Morrison-Woodbury pour adapter le PSVM à la fouille d'ensembles de données dont le nombre d'attributs est très important et le nombre d'individus plus restreint sur un matériel standard. Puis nous présentons un algorithme de boosting de PSVM pour classifier des données de très grandes tailles en nombre d'individus et d'attributs. Nous évaluons les performances du nouvel algorithme sur les ensembles de données de l'UCI, Twonorm, Ringnorm, Reuters-21578 et Ndc.

1. Introduction

La fouille de données est un domaine récent de l'informatique dont le développement est lié aux masses de données de plus en plus importantes qui sont stockées à l'heure actuelle. Son but est de pouvoir extraire des informations intéressantes de ces bases de données. De nombreux algorithmes venant de différents domaines de recherche sont utilisés pour l'extraction de connaissances (intelligence artificielle, statistique, analyse de données, bases de données…). Parmi eux, on trouve les arbres de décision, les règles d'association, les SVM. Nous nous intéressons particulièrement à une classe d'algorithmes d'apprentissage supervisé : les SVM ou Séparateurs à Vaste Marge. En fournissant des outils performants de classification, régression et détection de nouveauté [Bennett et Campbell, 2000], les algorithmes de SVM ont été utilisés dans plusieurs applications comme : la reconnaissance de visages, la reconnaissance de chiffres manuscrits, la classification de textes ou la bioinformatique [Guyon, 1999]. En général, ils donnent de bons taux de précision. L'apprentissage des SVM se ramène à résoudre un programme quadratique, la mise en œuvre d'un algorithme de SVM est donc coûteuse en temps.

Récemment, Fung et Mangasarian ont proposé un algorithme de proximal SVM qui n'a besoin que de la résolution d'un système linéaire. Le PSVM offre des particularités intéressantes :

- c'est un algorithme incrémental, donc il peut traiter sans difficulté des fichiers de très grandes tailles sur des machines standard où les autres algorithmes nécessiteront des capacités en mémoire vive beaucoup plus importantes et donc des machines spécifiques (genre serveurs),
- il est très rapide, de l'ordre de plusieurs dizaines à plusieurs centaines de fois plus rapide que les SVM standard suivant les types de fichiers,
- son taux de précision est équivalent aux autres algorithmes,
- il est facilement parallélisable : [Poulet et Do, 2003] ont traités des fichiers d'un milliard d'individus en moins de deux minutes sur dix PC standard (P4-2,4 GHz, 256 Mo RAM).

La complexité de l'algorithme varie linéairement avec le nombre d'individus et avec le carré du nombre d'attributs. L'occupation mémoire est quant à elle proportionnelle au carré du nombre d'attributs.

Or certaines applications, comme par exemple la classification de textes ou de gènes, traitent des ensembles de données ayant un très grand nombre d'attributs (de l'ordre de 10^3 à 10^5) et un nombre plus restreint d'individus (de l'ordre de 10^2 à 10^4). Dans ce cadre d'utilisation, l'algorithme PSVM va très rapidement montrer ses limites. Pour remédier à ce problème, nous avons appliqué le théorème de Sherman-Morrison-Woodbury [Golub et Van Loan, 1996] à l'algorithme de PSVM linéaire. On obtient alors une complexité en fonction du nombre d'attributs et du carré du nombre d'individus avec une occupation mémoire fonction du carré du nombre d'individus. Cette nouvelle écriture de l'algorithme de PSVM nous permet donc de traiter des données avec un très grand nombre d'attributs et un nombre plus faible d'individus. Les performances de l'algorithme [Do et Poulet, 2003] ont été évaluées sur des ensembles de données bio-médicales.

Pour pouvoir traiter des ensembles de données de très grande taille en nombre d'individus et en nombre d'attributs, nous avons étendu l'algorithme de PSVM en utilisant un algorithme de boosting. Nous avons évalué les performances du nouvel algorithme sur les ensembles de données de l'UCI [Blake et Merz, 1998], Twonorm, Ringnorm [Breiman, 1996], Reuters-21578 [Lewis, 1997], Ndc [Musicant, 1998]. Nous avons également comparé le taux de précision et le temps d'exécution du boosting de PSVM avec ceux de l'algorithme standard LibSVM [Chang et Lin, 2003].

Le paragraphe 2 présente le principe de l'algorithme de PSVM linéaire. Ensuite, nous décrivons deux versions incrémentales de PSVM dans le paragraphe 3. Nous présentons la construction du boosting de PSVM dans le paragraphe 4. Les résultats numériques sont présentés dans le paragraphe 5 avant de conclure sur nos travaux.

Nous utilisons quelques notations dans cet article : tous les vecteurs sont des vecteurs colonnes; la norme-2 du vecteur x est notée par $\|x\|$ et x^T est la transposée de x. La matrice $A[mxn]$ représente m individus avec n attributs. La matrice diagonale de ±1 $D[mxm]$ représente les classes de m individus. I est la matrice identité. e est le vecteur colonne de 1; ν est la constante positive; z est la variable de ressort (slack) et w, b sont les coefficients et le scalaire de l'hyperplan.

2. Proximal support vector machine linéaire

Les algorithmes de SVM ont pour objectif de rechercher le meilleur hyperplan permettant de séparer les données en deux classes. Soit m individus dans l'espace en dimension n, ils sont représentés par la matrice A [mxn], leurs classes sont représentées par la matrice

diagonale D [mxm] de 1 ou -1 (D[i,i] = 1 si Ai ∈ A+, D[i,i] = -1 si Ai ∈ A-). La séparation est réalisée par deux plans supports $x^T.w - b = \pm 1$.

Ai.w – b ≥ + 1 pour les individus Ai correspondant à D[i,i] = +1 (1)
Ai.w – b ≤ – 1 pour les individus Ai correspondant à D[i,i] = -1 (2)

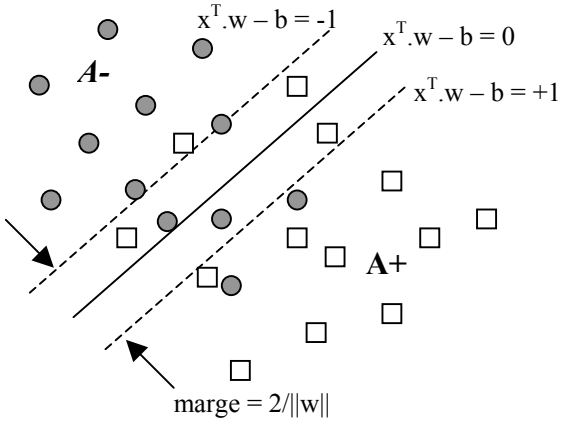

FIG. 1 – *Séparation linéaire des données en deux classes.*

On peut rassembler les deux formules (1), (2) en la formule (3) :

$$D(Aw - eb) \geq e \qquad (3)$$

L'algorithme standard de SVM (la maximisation de la marge et la minimisation des erreurs) est la solution du programme quadratique (4) :

$$\min_{z,\,w,\,b} \quad f = \nu e^T z + (1/2)\,\|w\|^2$$

$$\text{avec} \quad D(Aw - eb) + z \geq e \qquad (4)$$

où $z \geq 0$ est la variable de ressort (slack), et $\nu > 0$ est une constante est utilisée pour contrôler la marge et les erreurs.

L'hyperplan (w,b) est obtenu par la résolution de (4). La classification d'un nouvel individu x est calculée par : sign (x.w – b) (5)

Les SVM peuvent utiliser d'autres fonctions de noyau : fonction polynômiale de degré d, fonction RBF (Radial Basis Function), fonction sigmoïdale, … Le lecteur intéressé par les détails peut consulter [Cristianini et Shawe-Taylor, 2000] pour une explication complète

L'algorithme de proximal SVM [Fung et Mangasarian, 2001] modifie la formule de l'algorithme SVM standard en :
- maximisant la marge par $(1/2)\,\|w, b\|^2$
- minimisant les erreurs par $(\nu/2)\,\|z\|^2$

- remplaçant l'inégalité par l'égalité : $D(Aw - eb) + z = e$

En substituant z à w et b dans la fonction objectif f, (condition pour que f soit minimale), les dérivées premières en w, b sont nulles :

$$f'(w) = \nu A^T(Aw - eb - De) + w = 0 \qquad (6)$$
$$f'(b) = \nu e^T(-Aw + eb + De) + b = 0 \qquad (7)$$

(6) et (7) forment un système linéaire à (n+1) inconnues (les n coordonnées de w et le scalaire b). On peut réécrire (6) et (7) en (8) :

$$\left[w_1\ w_2\ w_3 \ldots w_n\ b\right]^T = \left(\frac{1}{\upsilon} I + E^T E\right)^{-1} E^T De \qquad (8)$$

où la matrice $E = [A \quad -e]$ est la juxtaposition de A avec une colonne de -1 et I est la matrice identité.

L'algorithme de PSVM n'a besoin que d'une seule résolution d'un système linéaire dont la complexité varie linéairement avec le nombre d'individus et avec le carré du nombre d'attributs. Le PSVM classifie $2x10^6$ individus avec 10 attributs et 2 classes en 15 secondes sur un PC (P4-2,4 GHz, 256 Mo RAM).

3. PSVM incrémental

Si l'ensemble de données a un très grand nombre d'individus (de l'ordre de 10^9) et un nombre plus restreint d'attributs (de l'ordre de 10^2 à 10^4), Fung et Mangasarian ont proposé de découper horizontalement (en lignes) l'ensemble de données A (les attributs des individus) et D (les classes des individus) en blocs Ai, Di. L'idée principale est ici de calculer de manière incrémentale [Fung et Mangasarian, 2002] (et parallélisée [Poulet et Do, 2003]) les valeurs $E^TE = \sum E_i^T E_i$ et $E^TDe = \sum E_i^T D_i e$. Les matrices E^TE, $E_i^TE_i$, sont de taille $(n+1)^2$, et les vecteurs E^TDe et $E_i^TD_ie$ sont de taille (n+1). Une fois ce calcul incrémental terminé, il ne reste plus qu'à inverser la matrice $I/\nu+E^TE$.

Théoriquement, cet algorithme est toujours valide, mais sa mise en œuvre va cependant poser des problèmes dans certains cas. Les limites vont venir des dimensions de l'ensemble de données traité en nombre de lignes (individus) et nombre de colonnes (attributs).

Si l'on se place dans le cas où les données sont telles que m >> n (le nombre d'individus est beaucoup plus important que le nombre d'attributs), la matrice E^TE conserve une taille raisonnable car seulement fonction du nombre d'attributs ce qui explique en partie les bonnes performances de l'algorithme de PSVM. Par exemple un milliard d'individus avec 10 attributs sont classifiés en deux classes en un quart d'heure sur un Pentium-4 (2,4 GHz, 256 Mo RAM).

Cependant, dans le cas inverse, m << n (le nombre d'individus est beaucoup plus faible que le nombre d'attributs), comme par exemple pour l'analyse d'expressions de gènes, on traite des ensembles de données ayant un très grand nombre d'attributs (de l'ordre de 10^3 à 10^5) et un nombre plus restreint d'individus (de l'ordre de 10^2 à 10^4). Les performances de l'algorithme deviennent alors "catastrophiques" : la matrice E^TE ne peut plus tenir en mémoire vive, la machine est obliger d'utiliser la mémoire secondaire (le disque dur) ce qui

ralentit de manière significative la vitesse d'exécution. A la limite, le système peut même être incapable de traiter une telle matrice. De plus le calcul de l'inverse de la matrice ($I/\nu + E^TE$) peut aussi poser problème à cause de sa taille.

A quantité de données égales : m x n, on a donc dans un cas (m >> n) un algorithme très rapide qui pourra traiter sans problème des données ayant un très grand nombre (plus d'un milliard) d'individus, sur une machine standard, et dans le cas inverse (m << n) un algorithme qui ne pourra peut-être même pas stocker les matrices permettant la résolution du problème.

Pour pouvoir traiter les données ayant un nombre d'attributs beaucoup plus important que le nombre d'individus, nous avons appliqué la formule (9) de Sherman-Morrison-Woodbury dans la partie droite de l'équation (8) permettant la résolution du problème.

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U\,(I + V^TA^{-1}U)^{-1}V^TA^{-1} \qquad (9)$$

Nous obtenons :

$$\begin{aligned}
&\left(\frac{1}{\upsilon}I + E^TE\right)^{-1}E^TDe \\
&= \left[\left(\frac{1}{\upsilon}I\right)^{-1} - \left(\frac{1}{\upsilon}I\right)^{-1}E^T\left(I + E\left(\frac{1}{\upsilon}I\right)^{-1}E^T\right)^{-1}E\left(\frac{1}{\upsilon}I\right)^{-1}\right]E^TDe \\
&= \upsilon E^TDe - \upsilon E^T\left(I + \upsilon EE^T\right)^{-1}\upsilon EE^TDe \\
&= \upsilon E^T\left[De - \left(I + \upsilon EE^T\right)^{-1}\upsilon EE^TDe\right] \\
&= \upsilon E^T\left[De - \left(\upsilon\left(\frac{1}{\upsilon}I + EE^T\right)\right)^{-1}\upsilon EE^TDe\right] \\
&= \upsilon E^T\left[De - \frac{1}{\upsilon}\left(\frac{1}{\upsilon}I + EE^T\right)^{-1}\upsilon EE^TDe\right] \\
&= \upsilon E^T\left[De - \left(\frac{1}{\upsilon}I + EE^T\right)^{-1}EE^TDe\right] \qquad (10)
\end{aligned}$$

On voit que la formulation de l'équation (10) nous permet de ne plus avoir à traiter que des matrices EE^T au lieu des matrices E^TE de l'équation (8). On passe ainsi d'une matrice de taille $(n+1)^2$ (le carré du nombre d'attributs) à une matrice de taille m^2 (le carré du nombre

d'individus). Dans le cas où n >> m, on a ainsi une matrice de taille raisonnable à manipuler assurant ainsi de bonnes performances en vitesse à l'algorithme de PSVM sans nécessiter des machines spécifiques avec énormément de mémoire vive.

On peut alors augmenter pratiquement à volonté le nombre d'attributs en utilisant le même type de technique incrémentale que celle décrite pour l'algorithme incrémental en ligne. Le découpage des données va cette fois-ci s'effectuer en colonne au lieu de ligne. On découpe donc verticalement (en colonne) l'ensemble de données A (donc E) en un ensemble de blocs Ai (Ei). Ensuite l'algorithme calcule de manière incrémentale les valeurs $EE^T = \sum E_i E_i^T$. La taille des matrices EE^T et $E_i E_i^T$ est de l'ordre de m^2 (le carré du nombre d'individus). Une fois ce calcul incrémental terminé, il ne reste plus qu'à calculer l'inverse de la matrice $(I/\nu + EE^T)^{-1}$. Cette nouvelle formulation de l'algorithme incrémental de PSVM nous permet de traiter des données avec un nombre d'attributs très supérieur au nombre d'individus sur des machines standard en un temps très restreint.

4. Boosting de PSVM

Malheureusement, si l'ensemble de données a simultanément un très grand nombre d'individus et d'attributs, il ne peut être traité par aucune des deux versions incrémentales. Par exemple, l'ensemble de données avec 50 000 individus et 20 000 attributs occupe plus de 4 Go en mémoire. Les PSVM ne chargent qu'un petit bloc de données à chaque étape, mais ils gardent en mémoire une matrice de taille soit le carré du nombre d'individus, soit le carré du nombre d'attributs et ensuite ils inversent cette matrice à la dernière étape. La tâche d'apprentissage nécessite alors énormément de mémoire vive et est coûteuse en temps d'exécution. Pour remédier à ce problème, nous proposons ici une extension de l'algorithme de PSVM : la construction d'un boosting de PSVM. Cette solution apporte non seulement de bonnes performances en temps d'exécution et occupation mémoire, mais aussi de bons taux de précision. Nous commençons à présenter sommairement la construction du boosting de PSVM comme suit :

Le boosting développé par Freund et ses collègues dans les années 90 est une méthode mettant en œuvre un ensemble de classifieurs faibles (weak learner) pour améliorer les performances de ces classifieurs. L'idée principale est de répéter l'apprentissage d'un classifieur faible à plusieurs reprises. A chaque étape de boosting, le classifieur faible se concentre sur les individus mal classifiés la fois précédente. Pour cela, on assigne un poids à chaque individu, ces poids sont mis à jour (augmentés pour les individus mal classifiés) après chaque tâche d'apprentissage. Initialement, les poids sont égaux, un classifieur faible construit un modèle H' à partir d'un échantillon A', ensuite la mise à jour des poids assignés aux individus vise à augmenter les poids des mal classifiés par le modèle H', on fait une boucle et on combine les modèles obtenus en un seul. Le boosting est simple à implémenter, il donne de bons résultats en pratique. Le lecteur intéressé pourra trouver des études détaillées sur le site www.boosting.org ou [Freund et Schapire, 1999]. Nous nous focalisons ici sur la construction d'un boosting de PSVM. L'idée est de considérer l'algorithme de PSVM comme un classifieur faible. Nous créons l'échantillon de l'ensemble de données pour construire le modèle en se basant sur les poids assignés aux individus. L'algorithme de PSVM peut ainsi traiter de très grands ensembles de données. Si les données ont un très grand nombre d'individus, l'algorithme incrémental en ligne de PSVM est utilisé comme classifieur faible. Dans le cas où l'ensemble de données a un très grand nombre d'attributs,

ou simultanément un grand nombre d'individus et d'attributs, l'algorithme incrémental en colonne est utilisé comme classifieur faible.

Le nouvel algorithme a de bonnes performances en temps d'exécution, taux de précision et occupation mémoire sur les ensembles de données de l'UCI, Twonorm, Ringnorm, Reuters-21578, Ndc. Nous avons comparé ces performances avec ceux de l'algorithme LibSVM.

5. Résultats

L'ensemble du programme est écrit en C/C++ sous IRIX (station SGI-O2) et Linux (PC). Nous nous intéressons ici à évaluer les performances en temps d'exécution, taux de bonne classification et occupation mémoire du nouvel algorithme de boosting de PSVM linéaire. L'algorithme LibSVM avec un noyau linéaire est utilisé pour le comparer avec notre algorithme. Ces tests ont été réalisés sur un PC Pentium-4 (2,4GHz, 256 Mo RAM, Linux Redhat 9.0). Une description des ensembles de données que nous avons utilisés est fournit dans le tableau 1.

	Classes	Individus	Attributs	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	8	581012	54	10-fold
Ndc	2	55000	20000	50000 Trn – 5000 Tst

TAB 1 – *Description des ensembles de données*.

	Temps (secs)		Précision (%)	
	BoostPSVM	LibSVM	BoostSVM	LibSVM
Twonorm	0,06	**0,03**	**97,04**	97,01
Ringnorm	**0,14**	0,23	**75,07**	73,82
Pima	0,088	**0,085**	**77,50**	76,82
Bupa	0,043	**0,019**	**69,12**	68,41
Ionosphere	0,106	**0,044**	88,00	**88,32**
Mushroom	**1,528**	3,75	**100,00**	**100,00**
Tic-tac-toe	**0,089**	0,323	**65,63**	65,34
Adult	**12,86**	2471,99 (~ 41 min)	85,25	**85,29**

TAB 2 – *Performances en terme de temps d'exécution et de taux de précision*.

Nous avons classifié les huit premiers ensembles de données en utilisant notre algorithme et LibSVM. Nous avons transformé les attributs nominaux des ensembles de données Adult et Mushroom en attributs binaires. Les résultats sont présentés dans le tableau 2.

Le temps d'exécution dans le tableau 2 est la moyenne d'une exécution sur l'ensemble d'apprentissage. Les meilleurs résultats sont notés en gras. Le boosting de PSVM est le meilleur en terme de précision dans 6 ensembles de données sur 8. En temps d'exécution, le boosting de PSVM est le meilleur dans 4 jeux d'essais sur 8. Remarquons que pour les ensembles de données ayant un grand nombre d'individus (plus de 10^4), le programme quadratique résolu par LibSVM nécessite beaucoup plus de temps de calcul. Par exemple avec Adult, le temps d'exécution de LibSVM augmente de manière significative, le boosting de PSVM est alors 190 fois plus rapide que LibSVM.

Le jeu de données Reuters-21578 est très utilisé pour évaluer les algorithmes de classification de textes. Les SVM donnent de bons résultats sur ces données. Donc, nous nous intéressons à comparer les performances du nouvel algorithme avec celles de LibSVM sur les données Reuters-21578. Nous avons utilisé le programme Bow [McCallum, 1998] pour extraire des termes à partir des documents. Aucune sélection de termes n'est nécessaire, nous traitons les 29406 termes obtenus. Un document est représenté par un vecteur de fréquences des termes de ce document. Le vecteur est considéré comme un individu en entrée des algorithmes d'apprentissage. Enfin, nous avons classifié 10 catégories de l'ensemble de données Reuters-21578.

	Précision (%)		Rappel (%)		Temps (secs)	
	Boost PSVM	Lib SVM	Boost PSVM	Lib SVM	Boost PSVM	Lib SVM
Earn	**99,26**	**99,26**	**97,56**	96,77	**8,42**	9,4
Acq	**95,69**	94,71	**97,45**	96,60	**8,47**	10,78
Money-fx	**81,01**	74,86	**77,96**	76,57	7,68	**7,4**
Grain	**87,92**	85,91	**93,57**	92,75	9,86	**5,05**
Crude	**92,59**	87,30	**87,06**	85,94	8,05	**5,7**
Trade	**76,86**	74,36	79,49	**80,56**	8,01	**5,73**
Interest	**77,10**	75,57	**79,53**	75,57	13,33	**8,69**
Ship	**83,15**	77,53	86,05	**88,46**	14,21	**5,36**
Wheat	**84,51**	83,10	**88,24**	88,06	14,10	**3,54**
Corn	**83,93**	82,14	94,00	**95,83**	6,28	**3,79**

TAB 3 – *Résultats sur 10 catégories de l'ensemble de données Reuters-21578*.

L'ensemble de données Reuters-21578 est un cas de classification multicatégorie (plus de 2), pour le traiter nous avons utilisé l'approche un-contre-le-reste. Nous avons 10 classes, la classification consiste à construire 10 modèles, où chaque modèle sépare une classe des 9 autres. Le tableau 3 présente les résultats concernant les taux de précision et de rappel obtenus par BoostPSVM et LibSVM. Le boosting de PSVM est toujours meilleur en terme de précision et le meilleur dans 7 catégories sur 10 en terme de rappel. En ce qui concerne les temps d'exécution, LibSVM est deux fois plus rapide que le boosting de PSVM. Ceci peut s'expliquer par le fait que le boosting de PSVM a nécessité plusieurs étapes de boosting pour atteindre les taux de précision et de rappel présentés.

Nous avons également mesuré les performances des algorithmes sur de grands ensembles de données. Les résultats sont présentés dans le tableau 4. LibSVM charge l'ensemble de données en mémoire, nous avons donc augmenté la capacité de mémoire pour que LibSVM puisse traiter les grands ensembles de données. Nous avons utilisé les 2 classes les plus grandes (nombre d'individus de la classe Spruce-Fir : 211840, nombre d'individus de la classe Lorgepole-Pine : 283301) de l'ensemble de données Forest cover type. Nous avons lancé LibSVM mais après 21 jours il n'y avait toujours pas de résultat. Ensuite, nous avons généré un grand ensemble de données (4 Go en mémoire) avec 55.000 individus, 20.000 attributs et 2 classes en utilisant le programme Ndc [Musicant, 1998]. Encore une fois, LibSVM ne peut pas traiter un tel ensemble de données parce qu'il nécessite plus de 4 Go pour stocker l'ensemble de données en mémoire, c'est pour cette raison que nous n'avons pas pu obtenir de résultat avec LibSVM.

	Mémoire (Mo)		Précision (%)		Temps (minutes)	
	Boost PSVM	Lib SVM	Boost PSVM	Lib SVM	Boost PSVM	Lib SVM
Forest cover type	**19**	354	**77.41**	N/A	**23,2**	31202,5
Ndc	**193**	> 4000	**81,18**	N/A	**1491,2 (~ 25h)**	N/A

TAB 4 – *Performances des algorithmes sur les grands ensembles de données*.

Quant à notre algorithme, aucune augmentation de mémoire n'est nécessaire, par contre il doit lire des données à chaque étape de boosting. Le temps de lecture occupe 96 % du temps d'exécution de l'algorithme. L'algorithme de boosting de PSVM a démontré sa capacité à pouvoir traiter des ensembles de données ayant simultanément un grand nombre d'individus et d'attributs (de l'ordre de 10^5) sur un PC standard dans un temps raisonnable.

6. Conclusion

Nous avons présenté dans cet article un nouvel algorithme de boosting de PSVM linéaire permettant de traiter de grands ensembles de données sur un matériel standard (Pentium-4, 2,4 GHz, 256 Mo RAM). A partir de l'algorithme initial de PSVM, nous avons utilisé le théorème de Sherman-Morrison-Woodbury pour adapter le PSVM à la fouille d'ensembles de données dont le nombre d'attributs est très important et le nombre d'individus plus restreint (ou l'inverse) sur un matériel standard. Ensuite, nous avons construit le boosting de PSVM permettant de traiter des données ayant simultanément un très grand nombre d'individus et d'attributs (de l'ordre 10^5) sur des machines standard dans un temps restreint. Ce nouvel algorithme a de bonnes performances en temps d'exécution, taux de précision et occupation mémoire. Ses résultats sont comparés à ceux obtenus par LibSVM.

L'extension la plus immédiate de ce travail est la parallélisation de l'algorithme pour gagner en temps d'exécution en se basant sur un réseau d'ordinateurs. Ensuite nous essayerons le boosting de PSVM non linéaire. Enfin, le résultat en sortie des algorithmes de SVM et de boosting est peu compréhensible, nous étudierons également les possibilités d'utiliser des méthodes graphiques interactives permettant d'améliorer la compréhensibilité du résultat obtenu par l'algorithme.

Remerciements

Nous remercions vivement Jason Rennie, MIT AI Lab NE43-733 200 pour son aide sur le prétraitement de l'ensemble de données Reuters-21578.

Références

[Bennett et Campbell, 2000] K. Bennett et C. Campbell. Support Vector Machines: Hype or Hallelujah ?. *SIGKDD Explorations*, 2(2), pp. 1-13, 2000.

[Blake et Merz, 1998] C. Blake et C. Merz. UCI Repository of Machine Learning Databases. 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Breiman, 1996] L. Breiman. *Bias, variance and arcing classifiers*. Technical Report 460, Statistics Department, University of California, 1996.

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

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

[Do et Poulet, 2003] T-N. Do et F. Poulet. Incremental SVM and Visualization Tools for Bio-medical Data Mining. Proceedings of Workshop on Data Mining and Text Mining in Bioinformatics, ECML/PKDD'03, Cavtat-Dubrovnik, pp. 14-19, 2003.

[Freund et Schapire, 1999] Y. Freund et R. Schapire. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5), pp. 771-780, 1999.

[Fung et Mangasarian, 2001] G. Fung et O. Mangasarian. Proximal Support Vector Machine Classifiers. Proceedings of the 7th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining, San Francisco, pp. 77-86, 2001.

[Fung et Mangasarian, 2002] G. Fung et O. Mangasarian. Incremental Support Vector Machine Classification. Proceedings of the 2nd SIAM, International Conference on Data Mining SDM'02, Virginia, pp. 247-260, 2002.

[Golub et Van Loan, 1996] G. Golub et C. Van Loan. Matrix Computations. John Hopkins University Press, Balti-more, Maryland, 3rd edition, 1996.

[Guyon, 1999] I. Guyon. SVM Application List. 1999. http://www.clopinet.com/isabelle-/Projects/SVM/applist.html.

[Lewis, 1997] D. Lewis. Reuters-21578 Text Classification Test Collection. 1997. http://ww-w.daviddlewis.com/resources/testcollections/reuters21578/.

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

[Musicant, 1998] D. Musicant. NDC: Normally Distributed Clustered Datasets. 1998. http://www.cs.wisc.edu/~musicant/data/ndc/.

[Pavlov et Mao, 2000] D. Pavlov et J. Mao. Scaling-up Support Vector Machines Using Boosting Algorithm. Proceedings of the 15th International Conference on Pattern Recognition, Barcelona, pp. 219-222, 2000.

[Poulet et Do, 2003] F. Poulet et T-N. Do. Mining Very Large Datasets with Support Vector Machine Algorithms. Proceedings of the 5th International Conference on Enterprise Information Systems, Angers, pp. 140-147, 2003.

[Tresp, 2001] V. Tresp. Scaling Kernel-based Systems to Large Data Sets. *Data Mining and Knowledge Discovery*, 5(3), pp. 197-211, 2001.
[Vapnik, 1995] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

Summary

In the recent years support vector machines (SVM) have been successfully applied to a large number of applications. Training a SVM usually needs a quadratic programming, so that the learning task for large data sets requires large memory capacity and a long time. Proximal SVM proposed by Fung and Mangasarian is a new SVM formulation; it is very fast to train because it requires only the solution of a linear system. We have used the Sherman-Morrison-Woodbury formula to adapt the PSVM to process data sets with a very large number of attributes. We have extended this idea by applying boosting to PSVM for mining massive data sets with simultaneously very large number of data points and attributes. We have evaluated its performance on UCI, Twonorm, Ringnorm, Reuters-21578 and Ndc data sets.