SVM incrémental, parallèle et distribué pour le traitement de grandes quantités de données

Thanh-Nghi Do*, François Poulet**

*College of Information Technology, Cantho University
1 Ly Tu Trong street, Cantho City, Vietnam
dtnghi@cit.ctu.edu.vn
**ESIEA - Pôle ECD
38, rue des Docteurs Calmette et Guérin, 53000 Laval - France
poulet@esiea-ouest.fr

Résumé. Nous présentons un nouvel algorithme de SVM (Support Vector Machine ou Séparateur à Vaste Marge) linéaire et non-linéaire, parallèle et distribué permettant le traitement de grands ensembles de données dans un temps restreint sur du matériel standard. A partir de l'algorithme de Newton-GSVM proposé par Mangasarian, nous avons construit un algorithme incrémental, parallèle et distribué permettant d'améliorer les performances en temps d'exécution et mémoire en s'exécutant sur un groupe d'ordinateurs. Ce nouvel algorithme a la capacité de classifier un million d'individus en 20 dimensions et deux classes en quelques secondes sur un ensemble de dix PC.

1 Introduction

A l'heure actuelle, les données arrivent plus vite que la capacité de traitement des algorithmes de fouille de données ne permet de les traiter. L'amélioration des performances des algorithmes de fouille de données est indispensable pour traiter de grands ensembles de données. Nous nous intéressons au cas de la classification supervisée et plus particulièrement à une classe d'algorithmes : les SVM [Vapnik, 1995]. En règle générale, ils donnent de bons taux de précision mais, l'apprentissage des SVM se ramène à résoudre un programme quadratique et est donc coûteux en temps et mémoire. Pour remédier à ce problème, les méthodes de décomposition [Platt, 1999], [Chang et Lin, 2003] travaillent sur des sous-ensembles arbitraires de données, on utilise alors des heuristiques [Do et Poulet, 2005] permettant de choisir les sous-ensembles de données. D'autres travaux visent à construire des algorithmes incrémentaux [Fung et Mangasarian, 2002] dont le principe est de ne charger qu'un petit bloc de données en mémoire à la fois, de construire un modèle partiel et de le mettre à jour en chargeant consécutivement des blocs de données. Les SVMs parallèles et distribués [Poulet et Do, 2004] utilisent un réseau de machines pour améliorer les performances. Nous présentons un nouvel algorithme de SVM linéaire et non-linéaire pour traiter de grands ensembles de données dans un temps restreint sur du matériel standard. A partir de l'algorithme de Newton-GSVM [Mangasarian, 2001], nous avons construit un algorithme incrémental, parallèle et distribué permettant d'améliorer les performances en temps d'exécution et mémoire en s'exécutant sur un groupe d'ordinateurs. Les résultats

expérimentaux montrent que ce nouvel algorithme a la capacité de traiter sans difficulté des très grandes quantités de données. Nous utilisons quelques notations dans cet article. Tous les vecteurs sont des vecteurs colonnes, la norme-2 du vecteur w est notée par $||w||$ et x^T est la transposée de x. La matrice $A[mxn]$ représente les m individus notés par x_i $(i = 1, 2, ..., m)$ dans l'espace réel en dimension n. La matrice diagonale de ± 1 $D[mxm]$ représente les classes y_i des m individus. I est la matrice identité, e est le vecteur colonne de 1, c est une constante positive, z est la variable de ressort (slack) et w, b sont les coefficients et le scalaire de l'hyperplan. Le paragraphe 2 présente le principe de l'algorithme de Newton-GSVM. Ensuite, nous décrivons une version incrémentale de l'algorithme de Newton-GSVM dans le paragraphe 3 puis la construction de l'algorithme parallèle et distribué de Newton-GSVM. Les résultats numériques sont présentés dans le paragraphe 5 avant de conclure sur nos travaux.

2 Méthode de Newton pour GSVM

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. Les SVM recherchent un hyperplan (w, b) permettant de séparer les individus en deux classes. On considère le cas où les individus ne sont pas linéairement séparables. La séparation est réalisée par deux plans support exprimée par :

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

où e est un vecteur colonne de 1

Les distances des erreurs sont notées par des variables de ressort $(z_i \geq 0 \ ; \ i=1, 2, ..., m)$. Si l'individu x_k est du bon côté de son plan support, alors z_k est égal à 0. La recherche de l'hyperplan optimal se ramène à simultanément maximiser la marge et minimiser les erreurs. La formulation primale du problème est exprimée par le programme quadratique (2) :

$$\min \ \Psi(w, b, z) = (1/2) \ ||w||^2 + cz \quad (2)$$

avec : $D(Aw - eb) + z \geq e$ et $z \geq 0$

où une constante $c > 0$ est utilisée pour contrôler la marge et les erreurs. Le plan optimal (w, b) est obtenu par la résolution du programme quadratique (2) dont la mise en œuvre est coûteuse en temps et mémoire vive. L'algorithme de generalized SVM (GSVM) [Mangasarian, 1998] modifie l'algorithme de SVM en maximisant la marge par $(1/2) \ ||w, b||^2$ et minimisant les erreurs par $(c/2) \ ||z||^2$. On obtient alors la formule primale de GSVM :

$$\min \ \Psi(w, b, z) = (1/2) \ ||w, b||^2 + (c/2) \ ||z||^2 \quad (3)$$

avec : $D(Aw - eb) + z \geq e$ et $z \geq 0$

où une constante $c > 0$ est utilisée pour contrôler la marge et les erreurs. Les contraintes dans (3) sont réécrites de la manière suivante (4) :

$$z = [e - D(Aw - eb)]_+ \quad (4)$$

avec $[o]+$ remplaçant les valeurs négatives par zéro.

En substituant z par w et b de la formule (4) dans la fonction objectif Ψ du programme quadratique (3), on obtient un problème d'optimisation non contrainte (5) :

$$\min \ \Psi(w, b) = (1/2) \ ||w, b||^2 + (c/2)||[e - D(Aw - eb)]_+||^2 \quad (5)$$

En notant $X = [w_1 \ w_2 \ ... \ w_n \ b]^T$ et $E = [A \quad -e]$, on peut réécrire le problème (5) en :

$$\min \ \Psi(X) = (1/2) \ X^T X + (c/2)||[e - DEX]_+||^2 \quad (6)$$

[Mangasarian, 2001] a proposé d'utiliser la méthode itérative de Newton pour résoudre efficacement le problème d'optimisation (6). Le principe de la méthode de Newton est de

minimiser successivement les approximations au second ordre de la fonction objectif Ψ en se basant sur un développement de Taylor au second ordre au voisinage de X_v.

$$\Psi(X) = \Psi(X_v) + \Psi'(X_v)(X - X_v) + (1/2)(X - X_v)^T \Psi''(X_v)(X - X_v) \quad (7)$$

On minimise la fonction quadratique $\Psi(X)$, ce qui fournit (8) pour que la dérivée première $\Psi'(X)$ soit égale à zéro.

$$\Psi'(X) = \Psi'(X_v) + \Psi''(X_v)(X - X_v) = 0 \rightarrow X = X_v - [\Psi''(X_v)]^{-1} \Psi'(X_v) \quad (8)$$

A l'itération p, on construit Ψ_p, approximation quadratique Ψ au voisinage de X_p, que l'on minimise pour obtenir X_{p+1}, défini par (8). Pour minimiser la fonction $\Psi(X)$ dans le problème d'optimisation (6), on calcule d'abord la dérivée première $\Psi'(X)$ de $\Psi(X)$ et ensuite le Hessien (dérivée seconde $\Psi''(X)$) de $\Psi(X)$.

$$\Psi'(X) = c(-DE)^T([e - DEX]_+) + X \quad (9)$$

$$\Psi''(X) = c(-DE)^T diag([e - DEX]^*)(-DE) + I \quad (10)$$

Avec *diag([e – DEX]*)* matrice diagonale *(m)x(m)* dont le i-ième élément diagonal est un sous gradient de $([e - DEX]_+)$. L'algorithme itératif de Newton pour GSVM est construit à partir des calculs de la dérivée première (9) et du Hessien (10). Cet algorithme converge vers la solution après un relativement faible nombre (de 5 à 8) d'itérations. Il est très efficace pour des problèmes avec un nombre restreint de dimensions (de l'ordre de 10^2 à 10^4), lorsque le calcul du Hessien est facile. Pour pouvoir classifier des données non linéairement séparables, l'algorithme de Newton-GSVM utilise une matrice de noyau *K[mxm]* au lieu de la matrice *A[mxn]* représentant les données. On peut construire une matrice de noyau *K[mxm]* en entrée du Newton-GSVM en utilisant l'ensemble des *m* individus (c'est-à-dire que toutes les données sont utilisées comme vecteurs support). La taille de cette matrice varie avec le carré du nombre d'individus, donc la mise en œuvre dans le cas non linéaire est très coûteuse en temps et mémoire. Pour remédier à ce problème, l'algorithme de Reduced SVM [Lee et Mangasarian, 2000] utilise un échantillon aléatoire de taille *s* (comme ensemble de vecteurs support) pour créer une matrice de noyau *K[mxs] (s << m)*. Le RSVM réduit la taille du problème et donne également de bons résultats (taux de précision).

3 Algorithme incrémental de Newton-GSVM

L'algorithme de Newton-GSVM classifie très efficacement des ensembles de données ayant un grand nombre d'individus et un nombre plus restreint de dimensions, il est beaucoup plus rapide que les algorithmes de SVM standard. La version incrémentale en ligne consiste à calculer de manière incrémentale la dérivée première (11) et le Hessien (12).

L'ensemble de données (A, D) a un très grand nombre d'individus (de l'ordre de 10^9) et un nombre plus restreint de dimensions (de l'ordre de 10^2 à 10^4), on découpe horizontalement (en lignes) l'ensemble de données A (les individus) et D (les classes des individus) en k petits blocs $A_1, D_1, \ldots, A_k, D_k$.

La dérivée première $\Psi'(X) = c\sum_{i=1}^{k}(-D_i E_i)^T([e - D_i E_i X]_+) + X$ (11)

Le Hessien $\Psi''(X) = c\sum_{i=1}^{k}(-D_i E_i)^T diag([e - D_i E_i X]^*)(-D_i E_i) + I$ (12)

L'algorithme incrémental en ligne de Newton-GSVM linéaire peut traiter de très grands ensembles de données sans difficulté. Entre deux étapes incrémentales ne sont conservées en

mémoire que la dérivée première $\Psi'(X)$, vecteur colonne de taille $(n+1)$ et le Hessien $\Psi''(X)$, matrice de taille $(n+1)x(n+1)$. Si l'on se place dans le cas où les données sont telles que le nombre d'individus est beaucoup plus important que le nombre de dimensions, la matrice $\Psi''(X)$ conserve une taille raisonnable car seulement fonction du nombre de dimensions ce qui explique les bonnes performances de l'algorithme de Newton-GSVM linéaire incrémental dans ce cadre d'utilisation.

4 Parallélisation et distribution de l'algorithme incrémental de Newton-GSVM

La quantité de données stockées ne cesse de croître, à l'heure actuelle elle dépasse parfois les possibilités de traitement. Pour pouvoir faire face à cet afflux, une solution est de paralléliser et distribuer le processus de fouille. Nous avons parallélisé l'algorithme incrémental de Newton-GSVM. Les deux caractéristiques incrémentale et parallèle de cet algorithme permettent à la fois d'optimiser au mieux l'utilisation de la mémoire (grâce à l'aspect incrémental) et le temps d'exécution (grâce à l'aspect exécution en parallèle).

Soit un grand ensemble de données découpé en k blocs lignes A_1, D_1, …, A_k, D_k et distribué sur plusieurs machines distantes (PC_1, PC_2, ... PC_k). Sur chaque machine distante, le bloc de données peut être traité en une seule fois ou être encore re-décomposé en blocs lignes. Cet aspect est intéressant pour la mise en œuvre du calcul sur un ensemble de machines disparates : on adapte les données à la capacité mémoire disponible. Le calcul du modèle (partiel) est effectué sur chaque machine distante et le résultat envoyé au serveur qui effectue la mise à jour de la solution pour chaque itération de l'algorithme. Le résultat final est absolument identique à ce que l'on aurait obtenu en utilisant l'algorithme séquentiel sur l'ensemble de données. Pour chaque itération on calcule parallèlement et indépendamment sur les machines distantes les $P_i = (-D_jE_j)^T([e - D_jE_jX_i]_+)$ et $Q_i = (-D_jE_j)^T diag([e - D_jE_jX_i]^*)(-D_jE_j)$. Ces résultats sont envoyés au serveur qui met à jour la solution et renvoie ce résultat aux machines distantes pour le calcul de la prochaine itération si la condition d'arrêt ($\Psi'(X) \cong 0$) n'est pas vérifiée. Nous avons choisi une solution très simple basée sur le mécanisme XML-RPC pour pouvoir travailler directement sur des entrepôts de données hétérogènes distribués sur le WEB, et donc lancer des portions de calcul sur les sites distants quel que soit le système d'exploitation et l'architecture des machines.

5 Résultats

L'ensemble du programme est écrit en C/C++ sous Linux (PC) avec la librairie Lapack++. Nous nous intéressons ici à évaluer les performances en temps d'exécution et taux de bonne classification en fouille de grands ensembles de données. Les jeux de test ont donc été effectués avec les ensembles de très grandes tailles : nous avons utilisé Ringnorm [Delve, 1996] pour générer des ensembles de données de dix mille à 1 milliard d'individus en dimension 20 avec 2 classes où la classe 1 a une moyenne égale à 0 et une variance égale à 4 et la classe -1 a une moyenne égale à 2/sqrt(20) et une variance égale à 1. Une description des ensembles de données est fournit dans le tableau 1. Les tests ont été réalisés sur des PCs Pentium-4 (3 GHz, 512 Mo RAM).

Nous avons découpé l'ensemble de données en blocs (égaux pour chaque machine). Ensuite nous avons fait varier la taille des blocs (les données sur une machine peuvent être

traitées en une fois ou par morceaux). Dans le pratique, si la taille des blocs est trop grande alors il y a saturation de la mémoire, le système d'exploitation passe tout son temps à « swapper ». La taille des blocs a donc une forte influence sur la vitesse d'exécution des algorithmes, de même que les caractéristiques matérielles de la machine utilisée. Dans la classification non linéaire des données Ringnorm, [Do et Poulet, 2005] a trouvé qu'environ 250 individus en tant que vecteurs support permettaient d'obtenir de bons résultats. Nous avons obtenu de bons résultats avec un échantillon aléatoire d'environ 200 individus (utilisés comme vecteurs support) représentant l'ensemble des données. Une matrice de noyau RBF, K est construite à partir de l'ensemble de données et des 200 vecteurs support. L'algorithme incrémental, parallèle et distribué a la capacité de traiter linéairement ou non linéairement les très grands ensembles de données. Nous avons aussi fait varier le nombre de machines et la taille des ensembles de données. Le temps d'exécution de l'algorithme varie linéairement avec la taille des ensembles de données et le nombre de machines utilisées et avec le carré du nombre de dimensions. Le tableau 1 présente les résultats de la classification des données sur 10 PCs Pentium-4 (3 GHz, 512 Mo RAM, Linux). L'algorithme incrémental, parallèle et distribué de Newton-GSVM a pu classifier linéairement un milliard d'individus en 20 dimensions en 2585 secondes (43 minutes) et non linéairement en 90100 secondes (25 heures) sur 10PCs. Ces résultats illustrent que notre algorithme a la capacité de traiter efficacement de très grandes quantités dans les cas linéairement ou non linéairement séparable en un temps restreint sur des machines standard.

	#individus	#individus	Linéaire		Non linéaire	
	apprentissage	test	T (sec)	Préc.(%)	T (sec)	Préc.(%)
Ens.1	10 000	1 000	0,025	77,00 %	0,913	98,90 %
Ens.3	100 000	10 000	0,255	76,22 %	8,90	98,67 %
Ens.5	1 000 000	100 000	2,58	76,36 %	90,3	98,57 %
Ens.7	10 000 000	1 000 000	25,8	76,31 %	908,5	98,56 %
Ens.9	100 000 000	10 000 000	259	76,32 %	9088	98,55 %
Ens.11	1 000 000 000	100 000 000	2585	76,32 %	90100	98,55 %

TAB 1 – *Description des ensembles de données et résultats de la classification sur 10 PCs*

6 Conclusion et perspectives

Nous avons présenté un nouvel algorithme incrémental, parallèle et distribué de Newton-GSVM pour traiter linéairement et non linéairement des grands ensembles de données dans un temps restreint sur du matériel standard : un milliard d'individus en 20 dimensions sont classifiés linéairement en deux classes en 43 minutes et non linéairement en 25 heures sur dix Pentium-4. L'apprentissage incrémental permet de traiter de très grandes quantités de données sans difficulté de mémoire sur une machine standard. Le traitement parallèle et distribué utilise un groupe de machines standard pour améliorer les performances en temps d'exécution. La complexité de l'algorithme varie linéairement avec le nombre d'individus de l'ensemble de données, le nombre de machines utilisées et le carré du nombre de dimensions. Avec une tâche de classification linéaire, un milliard d'individus (voire plus) sont classifiés sans difficulté sur des PCs. Dans le cas non linéaire, l'algorithme prend une matrice de noyau en entrée au lieu de la matrice de données. La complexité de l'algorithme et la qualité du modèle dépendent de l'ensemble des vecteurs support en entrée. Il faut rechercher de bonnes

heuristiques pour la sélection des vecteurs support afin d'obtenir de bonnes performances en terme de complexité et préserver aussi de bonnes qualités au modèle.

Références

Bennett K., C. Campbell (2000). Support vector machines: hype or hallelujah ?, in *SIGKDD Explorations*, 2(2), pp 1-13.

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

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

Do, T-N., 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, Vol. 2, pp 36-41, Porto, Portugal.

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

Fung, G., O. Mangasarian (2002). Incremental Support Vector Machine Classification, in proc. of the 2nd SIAM Int. Conf. on Data Mining SDM'2002 Arlington, Virginia, USA, pp 247-260.

Lee, Y-J., O. Mangasarian (2000). RSVM: Reduced Support Vector Machines, Data Mining Institute TR 00-07, Computer Sciences Department, Univ. of Wisconsin, Madison, USA.

Mangasarian, O. (1998). Generalized Support Vector Machines, Data Mining Institute Technical Report 98-14, Computer Sciences Department, Univ.of Wisconsin, USA.

Mangasarian, O. (2001). A Finite Newton Method for Classification Problems, Data Mining Institute Technical Report 01-11, Computer Sciences Department, University of Wisconsin, Madison, USA.

Mangasarian, O., D. Musicant (2001). Lagrangian Support Vector Machines, in *Journal of Machine Learning Research*, Vol. 1, pp 161-177.

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., pp 185-208.

Poulet F., T.N. Do (2004). Mining very large datasets with support vector machine algorithms, in *Enterprise Information Systems V*, Camp O., Filipe J., Hammoudi S. et Piattini M. Eds., Kluwer Academic Publishers, pp 177-184.

Vapnik V. (1995). *The nature of statistical learning theory*, Springer-Verlag.

Summary

The new incremental, parallel and distributed SVM algorithm using linear and non linear kernels proposed in this paper aims at classifying very large datasets on standard personal computers. We extend the recent finite Newton classifier for building an incremental, parallel and distributed SVM algorithm. The new algorithm is very fast and can handle very large datasets in linear or non-linear classification tasks. An example of the effectiveness is given with the linear classification into two classes of two million data-points in 20-dimensional input space in some seconds on ten personal computers.