SVM incrémental et parallèle sur GPU

François Poulet*, Thanh-Nghi Do**, Van-Hoa Nguyen***

*IRISA-Texmex
Campus de Beaulieu, 35042 Rennes Cedex
francois.poulet@irisa.fr
http://www.irisa.fr/texmex/people/poulet/index_fr.php

** Dpt LUSSI, Télécom Bretagne
Technopôle Brest-Iroise CS 83818, 29238 Brest Cedex 3
tn.do@telecom-bretagne.eu
http://perso.enst-bretagne.fr/tndo

***IRISA Symbiose
Campus de Beaulieu, 35042 Rennes Cedex
vhnguyen@irisa.fr
http://www.irisa.fr/symbiose/old/people/nguyen/

Résumé. Nous présentons un nouvel algorithme incrémental et parallèle de Séparateur à Vaste Marge (SVM ou Support Vector Machine) pour la classification de très grands ensembles de données en utilisant le processeur de la carte graphique (GPUs, Graphics Processing Units). Les SVMs et les méthodes de noyaux permettent de construire des modèles avec une bonne précision mais ils nécessitent habituellement la résolution d'un programme quadratique ce qui requiert une grande quantité de mémoire et un long temps d'exécution pour les ensembles de données de taille importante. Nous présentons une extension de l'algorithme de Least Squares SVM (LS-SVM) proposé par Suykens et Vandewalle pour obtenir un algorithme incrémental et parallèle. Le nouvel algorithme est exécuté sur le processeur graphique pour obtenir une bonne performance à faible coût. Les résultats numériques sur les ensembles de données de l'UCI et Delve montrent que notre algorithme incrémental et parallèle est environ 70 fois plus rapide sur GPU que sur CPU et significativement plus rapide (plus de 1000 fois) que les algorithmes standards tels que LibSVM, SVM-perf et CB-SVM.

1 Introduction

Les algorithmes de Séparateurs à Vaste Marge proposés par (Vapnik, 1995) et les méthodes de noyaux permettent de construire des modèles précis et deviennent des outils de classification de données de plus en plus populaires. On peut trouver de nombreuses applications des SVM comme la reconnaissance de visages, la catégorisation de textes ou la bioinformatique (Guyon, 1999). Cependant, les SVM demandent la résolution d'un

programme quadratique dont le coût de calcul 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 à l'heure actuelle (Lyman et al., 2003). Il y a donc besoin de permettre le passage à l'échelle des SVM pour traiter de grands ensembles de données sur des machines standard. Des méthodes heuristiques efficaces permettent d'améliorer le temps de calcul en décomposant le programme quadratique en une série de plus petits problèmes (Boser et al, 1992), (Chang et Lin, 2003), (Platt, 1999). Au niveau de la mise en œuvre, les méthodes d'apprentissage incrémental (Cauwenberghs et al, 2001), (Do et Poulet, 2006), (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 chargeant successivement les sous-ensembles d'apprentissage en mémoire sans avoir à charger l'ensemble total. Les algorithmes parallèles et distribués (Do et Poulet, 2006), (Poulet et Do, 2004) utilisent des machines en réseaux pour améliorer le temps d'exécution de l'apprentissage. Les algorithmes d'apprentissage actif (Do et Poulet, 2005), (Tong et Koller, 2000) choisissent un sous-ensemble d'individus (ensemble actif) représentatif pour la construction du modèle. Les algorithmes de boosting de PSVM et LS-SVM (Poulet et Do, 2004) se basent sur l'échantillonnage et le théorème de Sherman-Morrison-Woodbury (Golub et Van Loan, 1996) pour classifier simultanément un grand nombre d'individus et de dimensions.

Dans cet article, nous présentons un algorithme incrémental et parallèle de LS-SVM (Least Squares SVM) pour la classification de très grands ensembles de données sur GPUs, par exemple une carte graphique NVidia GeForce 8800GTX. Notre point de départ est donc l'algorithme de LS-SVM proposé par (Suykens et Vandewalle, 1999). L'algorithme de LS-SVM obtient la solution par résolution d'un système d'équations linéaires au lieu du programme quadratique. Il permet donc de classifier beaucoup plus rapidement des ensembles ayant de très grands nombres d'individus. Nous avons étendu cet algorithme de deux manières : nous en avons développé une version incrémentale pour pouvoir traiter de très grands ensembles de données (plus d'un milliard d'individus) et nous utilisons la carte graphique (GPU) qui est une architecture massivement parallèle pour obtenir une bonne puissance de calcul à faible coût.

Les performances des algorithmes en temps d'exécution et précision sont évaluées sur de grands ensembles de données provenant de l'UCI (Asuncion & Newman, 2007) et Delve (Delve, 1996) comme Forest Covertype, KDDCup 1999, Adult et RingNorm. Les résultats montrent que la version GPU de notre algorithme est environ 70 fois plus rapide que sa version CPU. Un exemple de son efficacité est la classification de l'ensemble de données de la KDD Cup 1999: 5 millions d'individus en dimension 41 sont classifiés en deux classes en 0.7 seconde sur une carte graphique NVidia GeForce 8800 GTX (comparé aux 55.7 secondes nécessaires sur CPU, Intel core 2, 2.6 GHz, 2 GB RAM). Nous avons aussi comparés nos résultats avec ceux obtenus avec LibSVM (Chang & Lin, 2003), Perf-SVM (Joachims, 2006) et CB-SVM (Yu et al., 2003).

Le paragraphe 2 présente brièvement l'algorithme de LS-SVM, puis le paragraphe 3 décrit l'algorithme incrémental permettant la classification de grands ensembles de données sur CPU. Dans le paragraphe 4 nous présentons la version incrémentale et parallèle sur GPU et quelques résultats 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[mxn] représente les m points en dimension n.

2 Algorithme de LS-SVM

Soit la tâche de classification binaire linéaire représentée sur la figure 1, avec *m* individus x_i *(i=1..m)* en *n* dimensions. Ils sont représentés par la matrice *A[mxn]* et leurs classes (+1 ou –1) stockées dans la matrice diagonale *D[mxm]*. L'algorithme de SVM cherche le meilleur hyperplan *(w, b)* de séparation des données, c'est-à-dire le plus éloigné à la fois de la classe +1 et de la classe -1. Il se ramène à d'une part maximiser la marge qui est la distance entre les plans supports (*2/||w||*) des deux classes (le plan support de la classe +1 [resp. –1] sépare tous les individus de la classe +1 [resp. –1] des autres) et d'autre part minimiser les erreurs (les individus du mauvais côté de leur plan support). Les distances au plan des erreurs sont notées par les variables $z_i \geq 0$ *(i=1..m)*. Si l'individu x_k est du bon côté de son plan support, alors z_k est égal à 0. L'algorithme de SVM revient au programme quadratique (1) :

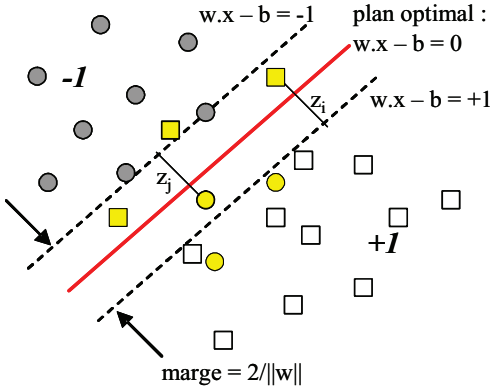

FIG. 1 – *SVM linéaire pour la classification des données en deux classes*

$$\begin{aligned} &\min\ \Psi(w, b, z) = (1/2)\ ||w||^2 + cz \\ &\quad \text{avec :} \\ &\qquad D(Aw - eb) + z \geq e \\ &\qquad z \geq 0 \end{aligned} \qquad (1)$$

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 (1). Ensuite, la classification d'un nouvel individu *x* est obtenue par le signe de (w.x-b).

Les SVM peuvent utiliser d'autres fonctions de classification comme par exemple une fonction polynomiale de degré *d*, une fonction RBF (Radial Basis Function) ou une fonction sigmoïdale. Le passage du cas linéaire au cas non linéaire se fait simplement en utilisant une fonction de noyau dans le produit scalaire de (1).

Cependant, le coût de calcul de la solution de (1) est au moins de l'ordre de $O(m^2)$, avec *m* représentant le nombre de points, ce qui fait qu'ils ne peuvent pas traiter de grands ensembles de données.

L'algorithme de LS-SVM proposé par (Suykens et Vandewalle, 1999) remplace la contrainte d'inégalité de (1) en égalité et modifie également l'algorithme de SVM (1) en minimisant les erreurs par min $(c/2)\ ||z||^2$ sous la contrainte : $D(Aw - eb) + z = e$

En substituant z dans la fonction objectif Ψ du programme quadratique (1), nous obtenons alors (2) :

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

Pour ce problème d'optimisation (2), nous calculons les dérivées partielles en w et b. Cela nous donne donc le système d'équations linéaires à $(n+1)$ inconnues $(w_1, w_2, ..., w_n, b)$ suivant :

$$\Psi'(w) = cA^T(Aw - eb - De) + w = 0 \quad (3)$$

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

(3) et (4) sont réécrits sous la forme du système d'équations linéaires (5) :

$$(w_1\ w_2\ ...\ w_n\ b)^T = (I°/c + E^TE)^{-1}E^TDe \quad (5)$$

où $I°$ est la matrice diagonale identité dont le dernier élément est 0 et E = [A -e].

L'algorithme de LS-SVM ne nécessite donc que la résolution d'un système d'équations linéaires (5) à $n+1$ inconnues au lieu du programme quadratique (1). Si le nombre de dimensions de l'ensemble de données n'est pas trop important, il est capable de traiter un très grand nombre d'individus (plus d'un milliard) en un temps restreint sur une machine standard.

Les résultats numériques ont montré que cet algorithme obtient la même précision que les algorithmes de SVM de références comme LibSVM, mais l'algorithme de LS-SVM est beaucoup plus rapide. Un exemple de son efficacité est décrit dans (Do et Poulet, 2006) avec la classification en deux classes d'un milliard d'individus en dimension 20 en seulement 13 secondes sur un PC standard (Pentium 4, 3 GHz, 512 Mo RAM).

3 Algorithme incrémental de LS-SVM

Bien que l'algorithme de LS-SVM soit rapide et efficace pour la classification de grands ensembles de données, il nécessite de charger la totalité de l'ensemble de données en mémoire pour effectuer la classification. Avec de grands ensembles de données, par exemple un milliard de points en dimension 20, il nécessite 80Go de RAM. Tous les algorithmes d'apprentissage ont des difficultés face au challenge des très grands ensembles de données. Nous allons nous concentrer sur le passage à l'échelle de ces algorithmes pour pouvoir traiter de grandes quantités de données sur des machines standard.

Les algorithmes incrémentaux sont une solution efficace pour le traitement de grands ensembles de données car ils évitent de charger la totalité des données en mémoire : seul un sous-ensemble de données est traité à un instant t et la solution partielle est mise à jour au fur et à mesure.

On suppose que l'on a un très grand ensemble de données décomposé en blocs de lignes Ai, Di. L'algorithme incrémental de LS-SVM va simplement calculer la solution du système d'équations linéaires (5) de manière incrémentale. Dans le cas le simple, supposons que l'ensemble de données à traiter est décomposé en deux blocs de lignes A_1, D_1 et A_2, D_2 (6) :

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}, e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$
$$E = \begin{bmatrix} A & -e \end{bmatrix} = \begin{bmatrix} A_1 & -e_1 \\ A_2 & -e_2 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} \quad (6)$$

Le calcul incrémental de la solution du système d'équations linéaires va alors s'effectuer de la manière suivante :

$$E^T De = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix}^T \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} E_1^T & E_2^T \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \quad (7)$$

$$E^T De = E_1^T D_1 e_1 + E_2^T D_2 e_2$$

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

$$E^T E = E_1^T E_1 + E_2^T E_2$$

$$\begin{bmatrix} w_1 w_2 ... w_n b \end{bmatrix}^T = \left(\frac{1}{c} I^o + \sum_{i=1}^{2} E_i^T E_i \right)^{-1} \sum_{i=1}^{2} E_i^T D_i e_i \quad (9)$$

A partir des équations (7), (8) et (9), on peut généraliser pour obtenir le calcul du LS-SVM incrémental (10) avec un ensemble de données décomposé en k blocs de lignes A_1,D_1, …, A_k, D_k :

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

Entrée :
- ensemble d'apprentissage en k blocs :A_1, D_1, …, A_k, D_k
- constante c pour régler la marge et les erreurs

Apprentissage :
- initialisation $E^TE=0$, $d=E^TDe=0$
- pour i de 1 à k
  - lit A_i et D_i
  - calcul de $E^TE= E^TE+ E_i^TE_i$ et $d=d+d_i$ (avec $d_i=E_i^TD_ie_i$)
- fin pour
- résolution du système d'équation (10)
- on obtient les coefficients du plan $(w,b)=w_1, w_2, …, w_n, b$

Classification d'un nouvel individu *x* basée sur : f(x)= signe(w.x-b)

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

Donc l'algorithme incrémental de LS-SVM présenté dans le tableau 1 peut traiter des ensembles de données de très grandes tailles sur une machine standard. La précision de l'algorithme est exactement la même que celle de l'algorithme original. Il ne nécessite que le stockage en mémoire d'une petite matrice de taille (n+1)x(n+1) et deux vecteurs de taille (n+1) entre deux étapes successives de l'algorithme. Les tests ont montré que l'algorithme incrémental peut effectuer la classification d'un milliard d'individus en dimension 20 en 21 minutes (sans compter le temps de lecture des données) sur un PC (Pentium 4, 3GHz, 512 Mo RAM).

4 Algorithme de LS-SVM incrémental et parallèle sur GPU

L'algorithme incrémental que nous venons de décrire peut traiter de très grands ensembles de données sur un simple PC, mais il ne s'exécute que sur un seul processeur. Nous l'avons étendu pour en obtenir une version parallèle sur GPU (Graphics Processing Unit).

Durant la dernière décennie les GPUs (Wasson, 2006) se sont développés avec des processeurs spécialisés pour l'accélération graphique. Le GPU a plusieurs avantages par rapport à l'architecture CPU actuelle pour le calcul intensif parallèle. Parmi ces avantages, on peut citer un meilleur taux de transfert mémoire, de significativement meilleurs calculs en flottant et l'utilisation de centaines d'unités parallèles de calcul en mode SIMD (Single Instruction Multiple Data). Les GPUs peuvent être une alternative intéressante au cluster de CPUs pour le calcul intensif. Les GPUs récents incluent de nouvelles possibilités de programmation et peuvent être utilisés pour des calculs non graphiques (GP-GPU : General Purpose GPU) comme des simulations de physique, du traitement du signal, des calculs géométriques ou de la fouille de données.

Entrée :
- ensemble d'apprentissage en k blocs : A1, D1, …, Ak, Dk
- constante c pour régler la marge et les erreurs

Apprentissage :
- initialisation $E^TE=0$, $d=E^TDe=0$ sur GPU
- pour i de 1 à k
  - charge Ai et Di en mémoire CPU
  - copie Ai et Di en mémoire GPU
  - calcul de $E^TE= E^TE+ E_i^TE_i$ et $d=d+d_i$ ($d_i=E_i^TD_ie_i$) sur GPU
- fin pour
- copie E^TE et E^TDe en mémoire CPU
- résolution du système d'équation (10) en CPU
- on obtient les coefficients du plan $(w,b)=w_1, w_2, \ldots, w_n, b$

Classification d'un nouvel individu *x* basée sur : $f(x)= signe(w.x-b)$

TAB. 2 – *Algorithme incrémental et parallèle de LS-SVM*

NVidia a récemment sorti une nouvelle carte graphique, la GeForce 8800 GTX et une bibliothèque en C appelée CUDA (NVidia Cuda, 2007) (Compute Unified Device Architecture). L'architecture de la NVidia GeForce 8800 GTX est basée sur 16 multi-processeurs. Chaque multi-processeur comporte 8 unités de calcul soit un total de 128 processeurs. Chaque groupe de 8 processeurs partage un cache mémoire L1. Chaque processeur possède une unité arithmétique (ALU : Arithmetic Logic Unit) et peut effectuer des opérations en flottants. Les instructions sont exécutées en mode SIMD. La Nvidia GeForce 8800 GTX possède 768 Mo de mémoire avec des performances en calcul de 330 GFLOPS et 86Go/s de taux de transfert mémoire. Cette architecture spécialisée peut être mise à profit pour du calcul intensif massivement parallèle. De plus la bibliothèque CUDA développée par NVidia en langage C permet d'utiliser la carte graphique pour des calculs non-graphiques (GP-GPU). Avec CUDA la GPU est utilisée comme support pour l'exécution en parallèle. CUDA inclut un pilote de la carte, une API et des librairies mathématiques de haut niveau pour un usage non-graphique comprenant notamment CUBLAS (NVidia Cublas, 2007) (Basic Linear Algebra Subprograms). Le modèle de base d'une application de la librairie CUBLAS est la création de matrices et vecteurs dans la mémoire GPU, l'appel de fonctions CUBLAS et finalement récupération des résultats de la GPU vers la machine hôte. Le taux de transfert entre la mémoire GPU et CPU est de 2Go/s.

Nous avons donc développé une version parallèle de l'algorithme incrémental de LS-SVM sur GPU pour bénéficier de bonnes performances de calcul à faible coût. L'implémentation parallèle et incrémentale décrite dans le tableau 2 utilise la librairie CUBLAS pour les calculs matriciels sur l'architecture massivement parallèle de la GPU. Le programme peut être utilisé sur n'importe quelle carte graphique compatible avec CUDA/CUBLAS (soit aujourd'hui plus de 200 cartes différentes, toutes de chez NVidia). Avec l'utilisation de CUDA/CUBLAS, la parallélisation est effectuée de manière implicite.

L'ensemble de données est tout d'abord découpé en petits blocs Ai, Di. A chaque étape de l'algorithme incrémental, un bloc de données, Ai, Di est chargée en mémoire CPU, puis transféré de la mémoire CPU à la mémoire GPU, ensuite les sommes des $Ei^{T}Ei$ et $di=Ei^{T}Diei$ sont calculées en parallèle. Finalement les résultats $Ei^{T}Ei$ et di sont recopiés de la mémoire GPU à la mémoire CPU pour résoudre le système d'équations linéaires. La précision de l'algorithme incrémental et parallèle est exactement la même que celle de l'algorithme original.

5 Quelques résultats

Pour l'expérimentation, nous avons utilisé un PC, Intel Core 2, 2.6GHz, 2Go RAM avec une carte graphique NVidia GeForce 8800GTX et un pilote NVidia version 6.14.11.6201 et CUDA 1.1 sous Linux Fedora Core 6. Nous avons développé deux versions du code (CPU et GPU) de l'algorithme incrémental et parallèle de LS-SVM en C/C++ avec les bibliothèques CUDA/CUBLAS et la librairie Lapack++ (Dongarra et al, 1993) pour bénéficier de bonnes performances en calcul matriciel. Les résultats de la version GPU sont comparés à la version CPU sous Linux Fedora Core 6. On ne prend en compte que le temps de calcul sans le temps nécessaire à la lecture des données et pour la version GPU, le temps de transfert CPU/GPU est pris en compte (donc le temps de calcul pur est inférieur de 10 à 50% à ce qui est présenté).

On se focalise sur les tests avec les plus grands ensembles de données de l'UCI comprenant les ensembles de données Forest Cover Type, KDD Cup 1999 et Adult (tableau 3). On a aussi créé deux grands ensembles de données en utilisant le programme RingNorm avec 20 dimensions, 2 classes et 1 et 10 millions d'individus.

	Classes	Individus	Dimensions	Protocole de test
Adult	2	48842	110	32561 Trn - 16281 Tst
Forest cover type	7	581012	54	495141 Trn – 45141 Tst
KDD cup 1999	5	5 209 458	41	4 898 429 Trn - 311 029 Tst
Ringnorm-1M	2	1000000	20	1000000 Trn - 100000 Tst
Ringnorm-10M	2	10000000	20	10000000 Trn - 1000000 Tst

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

	Temps (s)			Précision (%)
	GPU	CPU	ratio	
Adult	0.03	2.00	66.67	85.08
Forest cover type	0.10	10.00	100	76.41
KDD cup 1999	0.70	55.67	79.53	91.96
Ringnorm-1M	0.07	3.33	47.57	75.07
Ringnorm-10M	0.61	31.67	51.92	76.68

TAB. 4 – *Résultats de la classification*

Nous avons tout d'abord découpés les ensembles de données en petits blocs de lignes. Pour la version CPU, nous avons fait varier la taille du bloc traité à chaque étape de l'algorithme de 1000 points jusqu'à occuper toute la mémoire disponible pour comparer le temps nécessaire pour chaque étape de la classification. Les meilleurs résultats ont été obtenus avec une petite taille de bloc n'utilisant pas la mémoire secondaire (le disque). Les ensembles de données ont donc été découpés en blocs de 5000 lignes pour assurer les meilleures performances.

Les résultats de la classification versions CPU et GPU de l'algorithme incrémental et parallèle de LS-SVM sont présentés dans le tableau 4. La version GPU est en moyenne 70 fois plus rapide que la version CPU.

Pour l'ensemble de données Forest Cover Type, nous avons essayé d'effectuer la classification avec LibSVM, mais après 21 jours de calcul nous n'avions toujours pas de résultat. Cependant des résultats publiés récemment ont montré que l'algorithme SVM-Perf a effectué la classification de cet ensemble de données en 171 secondes sur processeur Intel Xeon, 3.6GHz, 2Go RAM. Cela signifie que notre version GPU de l'algorithme de LS-SVM est environ 1500 fois plus rapide que SVM-Perf.

L'ensemble de données de la KDD-Cup 1999 contient des données réseau indiquant si la connexion réseau est "normale" (classe négative) ou constitue une attaque (classe positive). LibSVM ne peut s'exécuter par manque de mémoire. CB-SVM a effectué la classification de l'ensemble de données avec 90% de précision en 4750 secondes sur un Pentium III, 800MHz avec 1Go de RAM alors que notre algorithme effectue la même classification avec 91% de précision en seulement 0.7 secondes. Il s'avère donc 6000 fois plus rapide que CB-SVM.

Les résultats numériques montrent l'efficacité de notre algorithme pour la classification de grands ensembles de données sur GPU.

De plus, nous avons dit que notre algorithme était incrémental, mais il est aussi décrémental et peut donc être particulièrement adapté à la fouille de flux de données. On peut ajouter un bloc de nouvelles données et supprimer un bloc d'anciennes données, le modèle est alors mis à jour à partir de l'étape précédente. Il est ainsi possible d'avoir n'importe quelle fenêtre temporelle sur le flux de données et calculer le modèle correspondant.

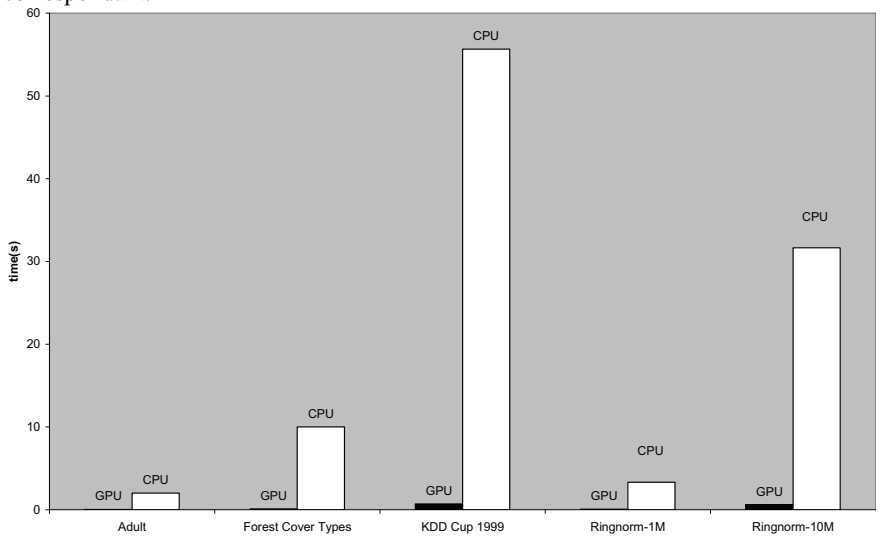

FIG. 2 – *Temps d'exécution des versions CPU/GPU*

(Domingos & Hulten, 2003) ont listés les critères qu'un algorithme de fouille de données doit suivre pour pouvoir traiter efficacement des flux de grandes quantités de données :

- il doit prendre un faible temps constant par individu : notre algorithme a une complexité linéaire en nombre d'individus,
- il doit utiliser une quantité de mémoire constante quelque soit la quantité de données à traiter : la mémoire nécessaire à notre algorithme est fixe et déterminée par la taille du bloc,
- il doit être capable de construire le modèle en ne lisant qu'une fois les données : on ne lit et traite chaque individu qu'une fois,
- il doit permettre d'obtenir un modèle des données à tout moment : on peut construire le modèle en faisant la somme à n'importe quel moment en prenant en compte les données déjà traitées,
- il doit produire un modèle équivalent à un algorithme qui n'aurait pas ces contraintes : le résultat que l'on obtient est exactement le même que celui de la version originale séquentielle,
- quand les données évoluent au cours du temps, on doit pouvoir faire évoluer le modèle et garder trace de ces changements : notre algorithme est incrémental et décrémental et peut donc suivre aisément tout changement.

Notre algorithme vérifie donc tous les critères listés ce qui nous garantit de pouvoir traiter

aisément de très grandes quantités de données.

6 Conclusion et perspectives

Nous avons présenté un nouvel algorithme incrémental et parallèle de LS-SVM pour la classification de très grands ensembles de données sur des processeurs graphiques (GPU). L'idée principale est d'étendre de deux manières l'algorithme proposé par Suykens et Wandewalle. Nous en avons développé une version incrémentale pour la classification de très grands ensembles de données. Notre algorithme évite de charger tout l'ensemble de données en mémoire vive : à un instant seul un sous-ensemble des données est en mémoire et le modèle est mis à jour au fur et à mesure du traitement. Nous avons ensuite développé une version parallèle de cet algorithme incrémental basée sur les processeurs graphiques (GPU) pour bénéficier de bonnes performances à faible coût.

Nous en avons évalué les performances en précision et temps de calcul sur les ensembles de données de l'UCI et de Delve. Les résultats montrent que la version GPU de notre algorithme est environ 70 fois plus rapide que sa version CPU. Nous l'avons aussi comparé à l'un des algorithmes de référence LibSVM et à deux autres algorithmes récents : SVM-Perf et CB-SVM. Notre version GPU est environ 1000 fois plus rapide que LibSVM, SVM-Perf et CB-SVM avec une précision similaire.

Nous avons aussi appliqué la même méthode aux algorithmes proposés par Mangasarian et ses collègues : Newton-SVM (Mangasarian, 2001) et Lagrangian-SVM (Mangasarian & Musicant, 2001) car ils ont les mêmes propriétés que l'algorithme LS-SVM. Les nouvelles implémentations de ces algorithmes parallèles sont très efficaces pour la classification de très grands ensembles de données.

Nous avons seulement utilisé un processeur graphique dans nos expérimentations et le temps d'exécution est déjà 1000 fois plus rapide que les algorithmes actuels tels que LibSVM, SVM-Perf et CB-SVM. Une amélioration évidente et immédiate va consister à utiliser simultanément plusieurs processeurs graphiques (GPU). Nous avons déjà montré pour la version CPU (Poulet et Do, 2004) que le temps de calcul était divisé par le nombre de machines utilisé. Avec la même propriété pour la version GPU, si on utilise 10 GPUs alors le temps de calcul sera divisé par 10000 par rapport aux algorithmes courants actuels. Cela signifie qu'une tâche de classification qui nécessitait une année de calcul est effectuée en une heure sur 10 GPUs (voire moins car les dernières cartes graphiques sorties sont trois fois plus rapides que celle utilisée ici). C'est une amélioration significative des possibilités des algorithmes de classification et peut ouvrir la voie à de nouvelles applications de fouille de données.

Une autre voie à explorer est l'extension de la méthode à des algorithmes de SVM non-linéaires.

Références

Asuncion, A. & Newman D.J.(2007). 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/.

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

Do, T-N. and Poulet F.(2006). Classifying one Billion Data wih a New Distributed SVM Algorithm, in proc. of RIVF'2006, 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 (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.

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.

Domingos P. & Hulten G (2003). A General Framework for Mining Massive Data Streams, in Journal of Computational and Graphical Statistics, 12(4):945-949.

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

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/app-list.html.

Joachims, T. (2006). Training Linear SVMs in Linear Time, in proc. of the *ACM SIGKDD* Int. Conf. on KDD, pp. 217-226.

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/.

Mangasarian, O. (2001). A finite newton method for classification problems. Data Mining Institute Technical Report 01-11, Computer Sciences Department, Univ. of Wisconsin.

Mangasarian O, Musicant D (2001). Lagrangian Support Vector Machines, Journal of Machine Learning Research, Vol.1, 217-226.

NVidia Cuda (2007). Cuda Programming Guide 1.1.

NVidia Cublas (2007). Cuda Cublas Library 1.1.

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.

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.

S.Wasson (2006), NVidia's GeForce 8800 graphics processor, Technical Report, PC Hardware Explored.

Yu, H., J. Yang, and J. Han (2003). Classifying large data sets using SVMs with hierarchical clusters, in proc. of the *ACM SIGKDD* Int. Conf. on KDD, pp. 306-315.

Summary

We present a new parallel and incremental Support Vector Machine (SVM) algorithm for the classification of very large datasets on graphics processing units (GPUs). SVM and kernel related methods have shown to build accurate models but the learning task usually needs a quadratic program so that this task for large datasets requires large memory capacity and long time. We extend a recent Least Squares SVM (LS-SVM) proposed by Suykens and Vandewalle for building incremental and parallel algorithm. The new algorithm uses graphics processors to gain high performance at low cost. Numerical test results on UCI and Delve dataset repositories showed that our parallel incremental algorithm using GPUs is about 70 times faster than a CPU implementation and significantly faster (over 1000 times) than state-of-the-art algorithms like LibSVM, SVM-perf and CB-SVM.