# Optimisation du Primal pour les SVM.

**Conference Paper** · January 2008

Source: DBLP

CITATIONS
0

READS
545

**2 authors**, including:

Thierry Artieres
Ecole Centrale de Marseille
**154** PUBLICATIONS **1,848** CITATIONS

SEE PROFILE

# Optimisation du Primal pour les SVM

Trinh-Minh-Tri Do*, Thierry Artières*

*LIP6, Université Pierre et Marie Curie
104 avenue du Président Kennedy, Paris, France
{Trinh-Minh-Tri.Do, Thierry.Artieres}@lip6.fr

**Résumé.** L'apprentissage de SVM par optimisation directe du primal est très étudié depuis quelques temps car il ouvre de nouvelles perspectives notamment pour le traitement de données structurées. Nous proposons un nouvel algorithme de ce type qui combine de façon originale un certain nombre de techniques et idées comme la méthode du sous-gradient, l'optimisation de fonctions continues non partout différentiables, et une heuristique de shrinking.

## 1 Introduction

Les Machines à Vecteurs de Support (SVM) sont une méthode très populaire d'apprentissage supervisé pour la classification et la régression. Dans sa forme la plus simple pour la classification bi-classes, cette méthode est basée sur un classificateur linéaire séparant deux ensembles de points par un hyperplan. L'idée originale est de trouver un hyperplan séparant "au mieux" les points par la maximisation de la marge entre l'hyperplan séparateur et les points dans la base d'apprentissage. Cette formulation conduit à un problème d'optimisation d'une fonction convexe sous des contraintes linéaires. Récemment des extensions de cette technique de base et de l'approche de maximisation de la marge ont été proposées pour le traitement de données structurées comme les séquences, les arbres etc (Tsochantaridis et al., 2004).

La méthode originale de Vladimir Vapnik pour résoudre le problème d'optimisation avec contraintes des SVMs consiste à introduire des multiplicateurs de Lagrange pour chaque contrainte, et d'optimiser le problème dual équivalent. Cet algorithme est coûteux en temps et en mémoire. Par exemple, l'espace mémoire nécessaire (la matrice noyau est de taille N au carré, si N est le nombre d'exemples). Ces caractéristiques de complexité rendent difficile l'emploi de machines à vecteurs support et plus généralement de méthodes de maximisation de la marge dans certaines situations, lorsque l'on traite des données structurées ou bien lorsque l'on dispose de très grandes quantités de données d'apprentissage. Plusieurs voies ont été suivies pour dépasser les problèmes posés par l'optimisation dans ce cadre.

Certains travaux ont porté sur l'optimisation efficace du dual, par le contrôle du nombre de contraintes actives (Joachims, 2006), ou par la décomposition du problème d'apprentissage (Osuna et al., 1997). Dans ce dernier cas, l'algorithme SMO ou SVMLight par exemple, on ne s'intéresse à une itération donnée qu'à un nombre limité de variables actives.

Des travaux plus récents ont porté sur l'optimisation directe de la forme primale par l'usage de la fonction hinge(z)=max(0,z). Cela permet de se ramener à un problème d'optimisation sans contraintes où la fonction objectif est convexe. La difficulté de ces dernières approches

vient du fait que la fonction hinge n'est pas dérivable en 0. Divers travaux ont alors proposé d'utiliser une version dérivable partout de cette fonction hinge par lissage ou bien en utilisant un coût quadratique car dans ce cas des méthodes d'optimisation standard peuvent être appliquées. Par exemple, (Chapelle, 2007) a montré qu'il était possible d'utiliser la méthode de Newton ou bien du gradient conjugué.

Une autre approche pour optimiser le primal consiste à utiliser la méthode du sous-gradient directement sur le problème d'optimisation de la fonction objectif non différentiable (Zhang, 2004). L'avantage de cette approche est sa simplicité, mais la vitesse de convergence est très dépendante du réglage du pas de gradient. Une exception est l'algorithme Pegasos (Shalev-Shwartz et al., 2007) qui est le seul algorithme de ce type ne nécessitant pas le réglage d'un hyperparamètre pour le pas de gradient.

Dans ce travail, nous nous plaçons dans le cadre de l'optimisation de fonction continue non partout dérivable. L'idée principale est que la minimisation du primal peut être attaquée comme un problème minimax sur un ensemble de fonctions quadratiques convexes définies sur des sous-espaces de l'espace des paramètres. Nous montrons que la fonction objectif n'est pas dérivable sur les frontières entre ces sous-espaces de définition et que ces frontières correspondent à des hyperplans associés à chaque exemple d'apprentissage. En analysant ces hyperplans dans l'espace des paramètres, nous décrivons une méthode efficace pour calculer la direction de plus grande pente et estimer le pas optimal. En exploitant ces résultats, nous proposons un nouvel algorithme d'optimisation qui se compare favorablement aux algorithmes de type Pegasos en mode batch et en mode on-line.

Nous décrivons tout d'abord le cadre d'optimisation dans lequel nous nous plaçons et les outils que nous allons utiliser. Ensuite nous décrivons notre algorithme en détails puis le validons expérimentalement en le comparant à des algorithmes de référence.

# 2 Préliminaires

Nous formalisons tout d'abord le problème que nous attaquons ici et qui consiste à optimiser une fonction objectif non partout différentiable. Nous rappelons ensuite quelques résultats sur ce type de fonctions et décrivons brièvement une méthode classique pour leur optimisation.

## 2.1 Formalisation

Considérons un problème de classification de données d'entrée $x$ en 2 classes $y \in \{-1, +1\}$. Considérons une base d'apprentissage $BA = \{(x_1, y_1), ..., (x_N, y_N)\}$ de $N$ exemples, où $x_i \in R^d$ et $y_i \in \{-1, +1\}$. Nous nous intéressons à l'apprentissage du classifieur linéaire : $h_w(x) = sign(\langle x, w \rangle)$ où $w \in R^d$ est l'ensemble des paramètres du classifieur à apprendre. Notons que $w$ divise $R^d$ en deux sous-espaces, la frontière est un hyperplan d'équation $H_w : \langle x, w \rangle = 0$. L'idée principale de l'apprentissage vaste marge est de trouver un hyperplan séparant "au mieux" les points par maximisation de la marge entre l'hyperplan séparateur et les points de la base d'apprentissage. Cette formulation conduit à un problème d'optimisation d'une fonction convexe sous des contraintes linéaires :

$$\begin{array}{lll} \min_w & \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i=1..N}\xi_i & \\ sous\,les\,contraintes & y_i\langle x_i, w\rangle \geq 1-\xi_i & \forall i \\ & \xi_i \geq 0 & \forall i \end{array} \quad (1)$$

où $\lambda$ est un hyper-paramètre de l'algorithme qui permet de régler l'importance des deux termes de la fonction objectif. Les $\xi_i$ sont des variables non-négatives qui représentent des pénalités pour les cas où la contrainte de marge n'est pas respectée pour l'exemple $x_i$.

En introduisant la fonction $hinge(z) = \max(0, z)$, on obtient un problème équivalent sans contraintes :

$$\min_w f(w) = \min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i=1..N}\max(0, 1 - y_i\langle x_i, w\rangle) \quad (2)$$

Notons que cette fonction objectif est quadratique par morceau et convexe mais qu'elle n'est pas différentiable (par rapport à $w$) en certains points, sur les hyperplans $H^i : 1 - y_i\langle x_i, w\rangle = 0$. Il y a un hyperplan par exemple d'apprentissage. En des points qui n'appartiennent à aucun de ces hyperplans la fonction objectif est localement quadratique.

Chaque hyperplan $H^i$ divise l'espace des $w$ en deux sous-espaces : $H^i_+ : \{w|1 - y_i\langle x_i, w\rangle \geq 0\}$ et $H^i_- : \{w|1 - y_i\langle x_i, w\rangle \leq 0\}$. Ces $N$ hyperplans divisent ainsi l'espace en hypercubes $C^k$ définis par : $C^k = \bigcap_{i=1..N} H^i_{\sigma^k_i}$ où $\sigma^k_i \in \{+, -\}$. Dans chaque hypercube $C^k$, la fonction objectif a une forme quadratique :

$$f(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i\in I^k}(1 - y_i\langle x_i, w\rangle) \quad \forall w \in C^k \quad (3)$$

où $I^k = \{i|\sigma^k_i = +\}$ est l'ensemble des indices des exemples qui violent la contrainte de la marge pour $w \in C^k$. Par construction cet ensemble est identique pour tous les $w$ d'un même hypercube $C^k$, c'est pourquoi nous ne marquons pas la dépendance de $I^k$ à $w$.

Avant de présenter notre algorithme en détail, nous commençons par quelques résultats théoriques sur l'optimisation de fonctions non-différentiables.

## 2.2 Généralités sur l'optimisation de fonctions non différentiables

Dans cette section, nous présentons quelque résultats concernant l'optimisation et l'analyse des fonctions convexes non-différentiables. Ces résultats, et leurs dérivations, peuvent être trouvés dans (Bertsekas et al., 2003; Demyanov et Vasilev, 1985). Nous introduisons tout d'abord la définition du sous-gradient et de la sous-différentielle d'une fonction convexe.

Soit $f : R^d \rightarrow R$ une fonction convexe, un vecteur $d \in R^d$ est appelé sous-gradient au point $x_0$ si :

$$f(x) \geq f(x_0) + \langle x - x_0, d\rangle \quad \forall x \in R^d \quad (4)$$

L'ensemble de tous les sous-gradients en $x_0$ est appelé la sous-différentielle en $x_0$, et est noté par $\partial f(x_0)$. La sous-différentielle est un ensemble compact, non vide et convexe.

**Théorème 1 (Demyanov and Vasilev) :** Une condition nécessaire pour une fonction continue non partout dérivable, éventuellement non convexe, $f(x) : R^d \rightarrow R$ d'atteindre un minimum en $x^*$ est que $0 \in \partial f(x^*)$. Pour une fonction convexe, la condition est également suffisante. Si $0 \notin \partial f(x^*)$ alors la direction $\Delta = -arg\min_{d \in \partial f(x)} \|d\|$ est la direction du sous-gradient de plus grande pente.

**Proposition 1 :** Soit $\{f_j : R^d \rightarrow R, j = 1, ..., m\}$ un ensemble de fonctions convexes et $f = \sum_{j=1..m} f_j$ , alors

$$\partial f(x) = \left\{ u = \sum_{j=1..m} d_j \Big| (d_1, .., d_m) \in \partial f_1(x) \times \partial f_2(x) \times ... \times \partial f_m(x) \right\} \tag{5}$$

**Théorème 2 (Danskin's) :** Soit $\{f_j : R^d \rightarrow R, j = 1, ..., m\}$ un ensemble de fonctions convexes différentiables, alors la sous-différentielle de $f = \max_{j=1..m} f_j$ est

$$\partial f(x) = conv\{\nabla f_j(x) | j \in I(x)\} \tag{6}$$

où $I(x) = \{i | f_i(x) = \max_j f_j(x)\}$, $\nabla f_j(x)$ est le gradient de $f_j(x)$ en $x$, et $conv(.)$ signifie enveloppe convexe.

### 2.3 Méthode du sous-gradient

Un façon d'optimiser le problème de l'équation (2) est d'utiliser la méthode du sous-gradient, qui converge vers le minimum global (Bertsekas et al., 2003). Nous comparerons notre algorithme à une méthode de référence de ce type (Shalev-Shwartz et al., 2007). A chaque itération, le nouveau vecteur de paramètres $w_t$ est calculé par :

$$w_{t+1} = w_t - \eta_t \nabla_t \tag{7}$$

où $\eta_t$ est le pas de sous-gradient et $\nabla_t \in \partial f(w_t)$ est un sous-gradient quelconque de la fonction $f$ à $w_t$.

## 3 Descente de sous-gradient pour l'optimisation du primal

Dans ce travail, nous proposons d'appliquer un algorithme du type descente de sous-gradient. La différence avec la méthode du sous-gradient est que nous choisissons la direction du sous-gradient selon la plus grande pente (plutôt que de prendre un sous-gradient quelconque dans la sous-différentielle), et que nous proposons une méthode optimale pour déterminer le pas de gradient. L'algorithme est résumé par le pseudo code décrit ce-dessous.

Cet algorithme a comme nous le verrons une complexité linéaire dans le nombre d'exemples d'apprentissage à chaque itération. En effet, il passe en revue tous les exemples d'apprentissage pour déterminer la direction de descente de plus grande pente et pour ensuite calculer le pas de gradient optimal dans cette direction. Nous décrivons maintenant ces étapes de calcul de la direction de plus grande pente du gradient et de calcul du pas de gradient optimal.

**Entrées** : $BA = \{(x_1, y_1), ..., (x_N, y_N)\}$
**Sorties** : $w$
Initialization : $w_1 = 0; t = 1$ ;
**tant que** *vrai* **faire**
  Calculer la direction du sous-gradient de plus grande pente $\Delta_t$;
  **si** $\|\Delta_t\| = 0$ **alors** return $w_t$; ;
  Calculer le pas de gradient optimal $\eta_t$ ;
  Mise à jour : $w_{t+1} = w_t + \eta_t \Delta_t$ ;
  $t = t + 1$;
**fin**

## 3.1 Direction du sous-gradient de plus grande pente

Rappelons que nous souhaitons minimiser la fonction objectif sous sa forme primale donnée dans l'Eq. (2). Nous cherchons dans un premier temps à déterminer la sous-différentielle de $f(w)$ en un "point" $w$. Pour cela nous commençons par nous intéresser à la sous-différentielle de la fonction élémentaire $\max(0, \frac{1-y_i\langle x_i,w\rangle}{N})$. Cette fonction est différentiable sauf pour les points $w$ tel que $y_i \langle x_i, w\rangle = 1$. Pour un point de ce type on obtient en appliquant le théorème 2 :

$$\partial\left(max(0, \frac{1 - y_i \langle x_i, w\rangle}{N})\right) = conv\left\{0, -\frac{y_i x_i}{N}\right\} = \left\{-\beta_i \frac{y_i x_i}{N} \middle| \beta_i \in [0, 1]\right\} \tag{8}$$

Par ailleurs en un point $w$ tel que $y_i \langle x_i, w\rangle \neq 1$ la fonction est dérivable et sa sous-différentielle est réduite à sa dérivée. La sous-différentielle de la fonction élémentaire s'exprime donc suivant les cas par :

$$\partial\left(\max(0, \frac{1 - y_i \langle x_i, w\rangle}{N})\right) = \begin{cases} 0 & si\ y_i \langle x_i, w\rangle > 1 \\ \left\{-\beta_i \frac{y_i x_i}{N} / \beta_i \in [0, 1]\right\} & si\ y_i \langle x_i, w\rangle = 1 \\ -\frac{y_i x_i}{N} & si\ y_i \langle x_i, w\rangle < 1 \end{cases} \tag{9}$$

Finalement, en utilisant la proposition 1 de la section précédente, on obtient :

$$\partial f(w) = \left\{ \lambda w + \sum_{i:y_i\langle x_i,w\rangle<1} -\frac{y_i x_i}{N} + \sum_{i:y_i\langle x_i,w\rangle=1} (-\beta_i \frac{y_i x_i}{N}) \;\middle|\; \beta_i \in [0, 1] \forall i \right\} \tag{10}$$

Il est intéressant de noter que cette formulation est proche de celle utilisée dans (Eizinger et Plach, 2003). Cela vient de la présence de fonctions $hinge$ dans la formalisation de l'apprentisage du perceptron, qui peut s'écrire comme la minimisation d'une fonction linéaire par morceaux, en fait la somme des fonctions $hinge(-y_i \langle x_i, w\rangle)$. A noter que dans ce cas la fonction objectif est non strictement convexe et qu'il peut exister, dans le cas séparable, une infinité de solutions. Dans notre formalisation la fonction objectif comporte un terme quadratique $\|w\|^2$ et des termes $hinge(1 - y_i \langle x_i, w\rangle)$, ce qui rend le problème strictement convexe. Par ailleurs, notre fonction objectif vise à maximiser la marge ce qui doit conduire à de meilleures propriétés de généralisation.

La recherche de la direction du sous-gradient de plus grande pente peut être vu comme le problème des moindres carrés suivant :

$$\beta* = argmin_{\beta=(\beta_{i_1},\beta_{i_2},...,\beta_{i_K})\in[0,1]^K} \ \|d_0 - \sum_{i\in\{i_1,...,i_K\}} \beta_i y_i x_i\| \tag{11}$$

où nous notons $\{i_1, ..., i_K\} = \{i | y_i \langle x_i, w\rangle = 1\}$ et en posant $d_0 = \lambda w + \sum_{i:y_i\langle x_i,w\rangle<1}(-\frac{y_i x_i}{N})$ qui est indépendant des paramètres d'optimisation $\beta_i$. Ce problème peut être résolu par des procédures standard (voir (Boyd et Vandenberghe, 2004)). Une fois $\beta^*$ calculé on obtient la direction du gradient de plus grande pente par la direction donnée dans le Théorème 1 avec les valeurs de $\beta^*$ données dans l'Eq. (11).

Notons pour terminer que si $K = 0$ alors la fonction est différentiable en $w$ et la direction du gradient de plus grande pente est simplement $\Delta_t = -d_0$.

## 3.2 Pas de gradient optimal

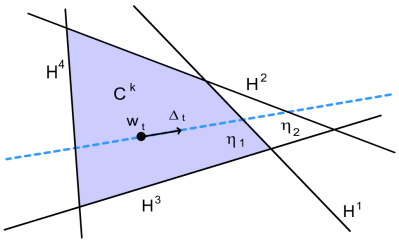
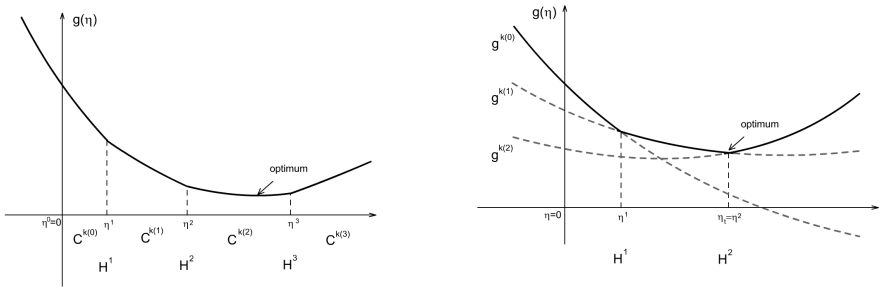



FIG. 1 – *L'espace des $w$.*

Une fois la direction de recherche $\Delta_t$ déterminée comme décrit à la section précédente, nous proposons ici une méthode pour déterminer le pas de gradient de façon optimale. Cela revient à résoudre le problème d'optimisation unidimensionnel suivant :

$$\eta_t = arg\min_{\eta} f(w_t + \eta\Delta_t) \tag{12}$$

Nous nous intéressons à la droite $D_t = \{w = w_t + \eta\Delta_t | \eta \in R\}$ et au comportement de $f(w)$ sur cette droite. Si l'on "avance" sur la droite $D_t$ dans la direction de $\Delta_t$, c'est à dire qu'on examine les $w = w_t + \eta\Delta_t$ pour $\eta$ croissant, alors $w$ va successivement croiser des hyperplans $H^i$ frontières entre hypercubes et traverser d'autres hypercubes. Au passage, la fonction $f(w)$ change d'une fonction quadratique d'un hypercube à une autre (voir figure 1). La fonction de $\eta$ que nous cherchons à optimiser, $g(\eta) = f(w_t + \eta\Delta_t)$ est une fonction quadratique par morceaux (voir figure 2). Les points non-différentiables correspondent aux intersections entre $w_t + \eta\Delta$ et les hyperplan $H^i$ séparateurs.




(a) (b)

FIG. 2 – *Recherche du pas de gradient optimal.*

On peut caractériser l'intersection, si elle existe, entre la droite $D_t$ et un hyperplan $H^i$ : $1 - y_i \langle x_i, w \rangle = 0$ par une valeur particulière de $\eta$, que nous notons $\eta^i$. Elle est déterminée par :

$$\begin{array}{lll} & 1 - y_i \left\langle x_i, (w_t + \eta^i \Delta_t) \right\rangle & = 0 \\ \iff & y_i \langle x_i, w_t \rangle + \eta^i y_i \langle x_i, \Delta_t \rangle & = 1 \\ \iff & \eta^i & = \frac{1 - y_i \langle x_i, w_t \rangle}{y_i \langle x_i, \Delta_t \rangle} \end{array} \tag{13}$$

Une valeur positive de $\eta^i$ signifie que l'hyperplan $H^i$ est "devant" $w_t$ (dans la direction $\Delta_t$) et une valeur négative de $\eta^i$ signifie le contraire.

Imaginons, sans perte de généralité, que la droite décrite par $w = w_t + \eta \Delta_t$ en faisant croître $\eta$ de zéro vers l'infini traverse successivement les hyperplans $H^1$, $H^2$, $H^3$... (voir Figure 2a). Sur le segment $\left[\eta^n, \eta^{n+1}\right]$, $g(\eta)$ vaut :

$$g^n(\eta) = \frac{\lambda}{2} \|w_t + \eta \Delta_t\|^2 + \frac{1}{N} \sum_{i \in I^{k(n)}} \left(1 - y_i \left\langle x_i, (w_t + \eta \Delta_t) \right\rangle\right) \tag{14}$$

où $k(n)$ est l'indice de l'hypercube correspondant au $n^{ieme}$ segment. Sa dérivée par rapport à la variable $\eta$ est :

$$\begin{array}{ll} \frac{\partial g^n(\eta)}{\partial \eta} & = \lambda \left\langle (w_t + \eta \Delta_t), \Delta_t \right\rangle + \frac{1}{N} \sum_{i \in I^{k(n)}} \left(-y_i \left\langle x_i, \Delta_t \right\rangle\right) \\ & = \lambda \left\langle w_t, \Delta_t \right\rangle + \lambda \eta \left\langle \Delta, \Delta_t \right\rangle - \sum_{i \in I^{k(n)}} \frac{y_i \langle x_i, \Delta_t \rangle}{N} \end{array} \tag{15}$$

A l'optimum cette dérivée, si elle existe, est nulle. Dans ce cas :

$$\eta^n_{opt} = \frac{\sum_{i \in I^{k(n)}} \frac{y_i \langle x_i, \Delta_t \rangle}{N} - \lambda \left\langle w_t, \Delta_t \right\rangle}{\lambda \left\langle \Delta_t, \Delta_t \right\rangle} \tag{16}$$

Si $\eta^n_{opt}$ appartient effectivement au $n^{ieme}$ segment, et que $g(\eta)$ est dérivable en cette valeur alors le pas optimal vaut $\eta^n_{opt}$. Cependant, $g(\eta)$ peut ne pas être dérivable à l'optimum (cf. figure 2b), cela signifie que l'optimum correspond à un $\eta$ sur une frontière entre segments (i.e. le $w$ optimal est sur une frontière, un hyperplan séparateur, entre hypercubes). Dans ce cas, pour tout $n$ l'optimum de $g^n(\eta)$ est "en dehors" du $n^{ieme}$ segment. Une méthode simple pour

vérifier si le $n^{ieme}$ point d'intersection $\eta^n$ est la solution est de calculer $\eta_{opt}^{n-1}$ et $\eta_{opt}^n$, si $\eta_{opt}^n \leq \eta^n \leq \eta_{opt}^{n-1}$ alors $\eta^n$ est la solution, sinon $\eta^n$ n'est pas la solution. Au final, notre algorithme pour trouver le pas de gradient optimal est décrit par le pseudo code ci-dessous. Dans ce code, nous notons $L$ le nombre d'hyperplans à envisager (a priori inconnu mais nécessairement $L \leq N$), c'est à dire ceux correspondant à des exemples pour lesquels il existe une intersection.

**Entrées** : $BA, w_t, \Delta_t$
**Sorties** : $\eta_t$
Initialization;
Estimer $I^0 \leftarrow \{i | y_i \langle x_i, w \rangle \leq 1 \forall w \in [w_t, w_t + \eta^1 \Delta]\}$;
Estimer les $\eta$ corespondant aux intersections de $D_t$ avec les hyperplans $H^i$, sélectioner des $\eta$ non-negatives et les trier dans l'ordre croissant : $\eta^1, \eta^2, \eta^3 ..., \eta^L$;
On note $i^1, i^2, ..., i^L$ les indices des hyperplans (i.e. exemples d'apprentissage) correspondents;
$n = 0; \eta^0 = 0$;
**tant que** *vrai* **faire**
  $\eta_{opt}^n = \frac{\sum_{i \in I^{k(n)}} \frac{y_i \langle x_i, \Delta_t \rangle}{N} - \lambda \langle w_t, \Delta_t \rangle}{\lambda \langle \Delta_t, \Delta_t \rangle}$;
  **si** $(\eta_{opt}^n \leq \eta^n)$ **alors** return $\eta^n$ ;
  **sinon si** $(n = L)$ *or* $(\eta^n \leq \eta_{opt}^n \leq \eta^{n+1})$ **alors** return $\eta_{opt}^n$;
  **sinon** $I^{n+1} = I^n \; xor \; \{i_{n+1}\}; n = n + 1$
**fin**

# 4 Shrinking et Complexité

L'algorithme itératif présenté dans la section précédente se compare avantageusement à d'autres algorithmes d'optimisation batch proposés récemment, en termes de performance et de complexité algorithmique. Nous proposons ici une variante beaucoup plus rapide de cet algorithme. Rappelons que l'algorithme passe en revue tous les exemples d'apprentissage (ou plutôt les hyperplans correspondants) pour déterminer la direction de descente de plus grande pente et pour ensuite calculer le pas de gradient optimal dans cette direction. Or en pratique, il y a le plus souvent très peu d'exemples qui contribuent à l'estimation de la direction de recherche et à l'estimation du pas de gradient optimal. Ce sont des exemples que nous appelons des exemples actifs, les autres étant passifs. On peut donc espérer casser la complexité de l'algorithme en réduisant le problème d'estimation à chaque itération en ne considérant que les exemples actifs. Nous proposons ici d'utiliser une méthode de *shrinking* similaire à ce qui est utilisé dans les techniques d'optimisation du dual pour sélectionner à une itération donnée un nombre restreint de variables actives (Joachims, 1999).

Pour minimiser la fonction objectif de l'Eq. (2), nous n'allons considérer à chaque itération qu'un nombre limité d'hyperplans actifs, en cherchant à optimiser la fonction :

$$f(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{i \in L_I} (1 - y_i \langle x_i, w \rangle) + \frac{1}{N} \sum_{i \in L_A} \max(0, 1 - y_i \langle x_i, w \rangle) \qquad (17)$$

où $L_A$ représente la liste des exemples actifs et $L_I$ représente la liste des exemples inactifs violant la marge. L'itération d'optimisation est réalisée comme décrit à la Section 3, la seule différence venant du fait que certains hyperplans ne sont pas considérés (les termes correspondants sont rajoutés à $d_0$, cf. Eq. (11)) et que l'on cherche une solution $w_{t+1}$ dans l'espace délimité par les hyperplans actifs. La procédure de sélection des exemples/hyperplans est heuristique. A l'initialisation tous les exemples sont actifs. A une itération $t$ on considère comme actifs les $K_t$ hyperplans les plus proches de la solution courante. Afin d'obtenir une optimisation très rapide, on réduit de moitié le nombre $K_t$ d'hyperplans actifs à chaque itération.

Pour garantir que la solution est correcte vis à vis des hyperplans sélectionnés, on restreint de plus l'espace de recherche à une boule $B(w_t, R_t)$ centrée autour de la solution courante et de rayon $R_t$, qui est calculée comme la distance maximale entre la solution courante et les hyperplans actifs sélectionnés. Il s'agit d'une heuristique qui permet d'éviter la plupart du temps de traverser un hyperplan inactif lors de la mise à jour de $w$. Si c'est le cas la solution trouvée est identique à celle que l'on trouverait avec tous les hyperplans actifs. Si ce n'est pas le cas on n'a plus cette garantie. C'est la raison pour laquelle nous avons choisi, régulièrement, de tout réinitialiser en réactivant tous les hyperplans, soit lorsque le nombre $K_t$ est inférieur à un seuil (e.g. 10) soit lorsque le rayon $R_t$ tombe à zéro. Cette procédure permet de garantir la convergence comme pour l'algorithme originel de la Section 3.

**Entrées** : $BA$
**Sorties** : $w$
Initialisation $t = 1; L_A = \{H^i | i = 1..N\}; L_I = \emptyset; R_t = inf$
**tant que** *vrai* **faire**
  1. Estimer la direction de recherche optimale $\Delta_t$ en considérant la fonction objectif de l'Eq. (11), pour les hyperplans actifs de $L_A$
  2. **si** $|\Delta_t| = 0$ **alors**
    Arrêt
  **fin**
  3. Estimer le pas optimal $\eta_t$ pour la fonction objectif de l'Eq. (12), dans la direction $\Delta_t$ et pour les hyperplans actifs de $L_A$
  4. Contraindre la nouvelle solution à appartenir à $B(w_t, R_t) : \eta_t = \min(\eta_t, \frac{R_{t-1}}{\|\Delta_t\|})$
  5. Mise à jour de $w : w_{t+1} = w_t + \eta_t \Delta_t$;
  6. $t = t + 1$;Mise à jour $L_A$, $L_I$, et $R_t$
  7. **si** *(*$|L_A| < 10$ *or* $(R_t = 0)$*)* **alors**
    $L_A = \{H^i | i = 1..N\}; L_I = \emptyset; R_t = inf$
  **fin**
**fin**

**Complexité** La complexité de chaque itération de l'algorithme sans shrinking est en $O(2Nd)$. En effet, les opérations coûteuses à chaque itération sont l'estimation de $(y_i \langle x_i, w \rangle)$ et de $y_i \langle x_i, \Delta_t \rangle$. Il y a $2 \times N$ calculs de ce type. En utilisant le shrinking, avec le même raisonnement la complexité d'une itération est en $2 \times O(|L_A|d)$. L'algorithme est une itération de plusieurs cycles. Dans chaque cycle, on part de $N$ hyperplans actifs, puis on divise par 2 le

nombre d'hyperplans actifs à chaque itération jusqu'à arriver à un nombre faible (e.g. <10). La complexité d'un cycle est donc en $2 \times O(Nd + (N/2)d + (N/4)d + ...) = 4 \times O(Nd)$.

# 5 Expériences

Nous décrivons ici des résultats expérimentaux obtenus en classification d'images de chiffres manuscrits sur la base MNIST[1]. Elle contient 60000 exemples d'apprentissage et 10000 exemples de test, les images sont en dimension 28x28. Nous avons prétraité les données via une analyse en composantes principales (ACP) afin de réduire la dimension des données à 50 dimensions (on ne garde que les 50 composantes des images sur les 50 axes principaux d'inertie). Il s'agit d'un prétraitement standard sur ces données, décrit par exemple dans (LeCun et al., 1998).

Nous avons comparé notre algorithme noté HyperPass (Hyperplane Passenger) avec l'algorithme Pegasos, basé sur la méthode de sous-gradient. Ce dernier s'est montré expérimentalement très efficace par rapport aux méthodes d'optimisation du dual pour les bases de données de grand dimension. Tout d'abord nous comparons la vitesse de convergence de Pegasos en mode batch et de notre algorithme sans la stratégie de shrinking. La figure 3 montre l'évolution du primal en fonction du nombre de passages sur la base de données. On voit que notre algorithme HyperPass converge beaucoup plus rapidement que Pegasos. Notons toutefois que chaque passage de la base de données correspond à une itération de HyperPass avec complexité $2 \times O(Nd)$, tandis que la complexité d'une itération de Pegasos est en $O(Nd)$. On voit également que la courbe de Pegasos est large car la valeur primal oscille beaucoup entre itérations successives.

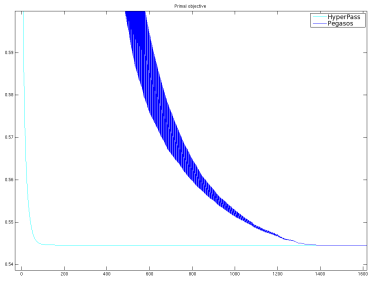



FIG. 3 – *MNIST '0' vs all - Primal objectif.*

Nous avons également étudié sur le même problème la version HyperPass avec shrinking et observé que cet algorithme converge après quelques cycles d'itérations (l'algorithme se termine au $7^{ieme}$ cycle, ce qui correspond à une complexité de $28 \times O(Nd)$). Nous comparons les solutions de HyperPass Shrinking et de la version on-line de Pegasos (beaucoup plus rapide que la version batch) pour une même complexité algorithmique (tableau 1). Le paramètre $K$ de Pegasos représente le nombre d'exemples utilisés pour faire un pas de sous-gradient,

[1] http ://yann.lecun.com/exdb/mnist/index.html

$K = N$ correspond au mode batch. On voit dans ce tableau que que la solution de HyperPass est meilleure que celles de Pegasos en termes de valeur du primal et de taux d'erreur de classification en test.

| Méthodes | nombre d'itérations | complexité | valeur du primal | erreur de test (%) |
|---|---|---|---|---|
| **HyperPass Shrinking** | **7 cycles** | **28**$\times O(Nd)$ | **0.5446187** | **18.40** |
| Pegasos K=N | 28 | 28$\times O(Nd)$ | 5.0695504 | 33.97 |
| Pegasos K=100 | 28$\times N/10$ | 28$\times O(Nd)$ | 0.5464795 | 18.87 |
| Pegasos K=10 | 28$\times N/10$ | 28$\times O(Nd)$ | 0.5464416 | 18.65 |

**TAB. 1** – *Comparaison de HyperPass Shrinking et de Pegasos avec une même complexité en temps.*

Enfin, nous comparons la solution de HyperPass Shrinking à celles obtenues avec l'algorithme Pegasos en le faisant tourner jusqu'à arriver à une valeur du primal seuil égale à celle obtenue par HyperPass +$\epsilon$, pour différentes valeurs de $\epsilon$. Le tableau 2 montre que Pegasos descend très vite au debut lorsque $K$ est petit, mais que la vitesse de convergence est plus faible ensuite. Par exemple, dans le cas $K = 10$, il faut $51$ passages pour arriver à une solution avec $\epsilon = 0.001$ tandis qu'il faut $246$ passages pour arriver à une solution avec $\epsilon = 0.0001$.

| Méthodes | nombre d'itérations | complexité | valeur du primal | erreur de test (%) |
|---|---|---|---|---|
| **HyperPass Shrinking** | **7 cycles** | **28**$\times O(Nd)$ | **0.5446187** | **18.40** |
| **Pegasos K=N** | | | | |
| $\epsilon = 0.01$ | 911 | 911$\times O(Nd)$ | 0.5545929 | 18.63 |
| $\epsilon = 0.001$ | 1273 | 1273$\times O(Nd)$ | 0.5455888 | 18.47 |
| $\epsilon = 0.0001$ | 1361 | 1361$\times O(Nd)$ | 0.5447137 | 18.45 |
| $\epsilon = 0.00001$ | 1376 | 1376$\times O(Nd)$ | 0.5446285 | 18.42 |
| **Pegasos K=100** | | | | |
| $\epsilon = 0.01$ | 18$\times N/100$ | 18$\times O(Nd)$ | 0.5491473 | 19.02 |
| $\epsilon = 0.001$ | 40$\times N/100$ | 40$\times O(Nd)$ | 0.5455721 | 18.70 |
| $\epsilon = 0.0001$ | 320$\times N/100$ | 320$\times O(Nd)$ | 0.5447172 | 18.48 |
| $\epsilon = 0.00001$ | 767$\times N/100$ | 767$\times O(Nd)$ | 0.5446287 | 18.49 |
| **Pegasos K=10** | | | | |
| $\epsilon = 0.01$ | 8$\times N/10$ | 8$\times O(Nd)$ | 0.5540219 | 18.79 |
| $\epsilon = 0.001$ | 51$\times N/10$ | 51$\times O(Nd)$ | 0.5455784 | 18.51 |
| $\epsilon = 0.0001$ | 246$\times N/10$ | 246$\times O(Nd)$ | 0.5447186 | 18.46 |
| $\epsilon = 0.00001$ | 667$\times N/10$ | 667$\times O(Nd)$ | 0.5446285 | 18.48 |

TAB. 2 – *Comparaison de HyperPass Shrinking et de Pegasos avec tolérance $\epsilon$ (voir texte).*

# 6 Conclusions

Nous avons proposé dans ce travail un nouvel algorithme d'apprentissage vaste marge pour des machines de type SVM. Cet algorithme optimise la fonction objectif sous sa forme primale en combinant la méthode du sous-gradient, des résultats sur l'optimisation de fonctions non

partout différentiables, et la nature de la fonction objectif que l'on cherche à minimiser et qui s'exprime comme un maximum de fonctions quadratiques. Nous avons proposé une version de base de cet algorithme et une version exploitant une stratégie de shrinking beaucoup plus rapide et tout aussi performante. Ces algorithmes héritent des propriétés de convergence des algorithmes de sous-gradient. Nous les avons comparé aux algorithmes les plus récents et nous avons démontré sur ces expériences leur supériorité.

Ces résultats sont d'ores et déjà très encourageants et nous travaillons en ce moment à étendre ce travail dans plusieurs directions, la kernélisation de la méthode permettant d'apprendre des classifieurs non linéaires, et le calcul de la vitesse de convergence.

# Références

Bertsekas, D. P., A. Nedic, et A. E. Ozdaglar (2003). *Convex analysis and optimization*.

Boyd, S. et L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press.

Chapelle, O. (2007). Training a support vector machine in the primal. *Neural Computation 19*(5), 1155–1178.

Demyanov, V. et L. Vasilev (1985). Nondifferentiable optimization.

Eizinger, C. et H. Plach (2003). A new approach to perceptron training. In *IEEE Transactions on Neural Networks*, pp. 216–221.

Joachims, T. (1999). Making large-scale svm learning practical. In *Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.)*. MIT Press.

Joachims, T. (2006). Training linear SVMs in linear time. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pp. 217 – 226.

LeCun, Y., L. Bottou, Y. Bengio, et P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

Osuna, E., R. Freund, et F. Girosi (1997). Training support vector machines : An application to face detection. In *CVPR'97*.

Shalev-Shwartz, S., Y. Singer, et N. Srebro (2007). Pegasos : Primal estimated sub-gradient solver for svm. In *ICML '07*, New York, NY, USA, pp. 807–814. ACM Press.

Tsochantaridis, I., T. Hofmann, T. Joachims, et Y. Altun (2004). Support vector machine learning for interdependent and structured output spaces. In *ICML '04*, pp. 104–112.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML '04*, New York, NY, USA, pp. 116. ACM Press.

# Summary

Learning SVM through direct optimization of primal is a recent and much studied problem since it opens new perspectives, for instance for dealing with structured outputs. We propose a new algorithm of this kind that combines in an original way a number of techniques and ideas, such as sub-gradient methods, optimization of non differentiable functions, and shrinking heuristics.