ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
"Antonio Ruberti"
CONSIGLIO NAZIONALE DELLE RICERCHE

G. Liuzzi, S. Lucidi, A. Manno, F. Rinaldi

A SPACE TRANSFORMATION DECOMPOSITION TECHNIQUE FOR SINGLY LINEARLY CONSTRAINED PROBLEMS SUBJECT TO SIMPLE BOUNDS

R. 11, 2015

Giampaolo Liuzzi – Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti" (IASI - CNR) — Via dei Taurini 19, 00185 Rome, Italy (giampaolo.liuzzi@iasi.cnr.it).

Stefano Lucidi – Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti", "Sapienza" Univerità di Roma — Via Ariosto 25, 00185 Rome, Italy (lucidi@dis.uniroma1.it).

Andrea Manno – Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti", "Sapienza" Univerità di Roma — Via Ariosto 25, 00185 Rome, Italy (manno@dis.uniroma1.it).

Francesco Rinaldi – Dipartimento di Matematica, Univerità di Padova — Via Trieste 63, 35121 Padova, Italy (rinaldi@math.unipd.it).

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR

via dei Taurini 19, 00185 ROMA, Italy

tel. ++39-06-49937101/02
fax ++39-06-49937106
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it

Abstract

In this work we propose a new decomposition algorithm for singly linearly constrained problems subject to simple bounds. In particular we focus on the training of Support Vector Machines. Our method, by exploiting a new space transformation technique combined with an intense caching strategy, aims to speed up the training task for large dimensional dataset. The practical implementation of the algorithm and some numerical experiments in comparison with two state-of-the-art training algorithms, are presented.

Key words: Decomposition Algorithm, Big Data, Support Vector Machine, Machine Learning

1. Introduction

The training of SVMs is usually formulated as a convex quadratic programming problem of the form:

$$\begin{aligned} \min \quad & f(x) = \frac{1}{2} x^T G x - e^T x \\ \text{s.t.} \quad & y^T x = 0, \\ & 0 \leq x \leq Ce, \end{aligned} \tag{1}$$

where G is a $n \times n$ symmetric positive semidefinte matrix, $e \in \mathbb{R}^n$ is a vector of all ones and C is a positive scalar. Furthermore, the generic element g_{ij} of matrix G is given by $y^i y^j K(x^i, x^j)$, where $K(a, b) = \phi(a)^T \phi(b)$ is the kernel connected to the nonlinear function ϕ mapping the data from the input space into the feature space, and (x^i, y^i) $i = 1, \ldots, l$ with $x^i \in \mathbb{R}^n$ and $y^i \in \{-1, 1\}$ are the input-target pairs of a given training set.
Since in the vast majority of applications the number l of training samples is huge, the dimensions of Hessian matrix G are such that it is impossible to fully store the matrix in the memory. In order to overcome this difficulty, decomposition techniques are usually adopted. These strategies consist in decomposing the problem (1) into a sequence of smaller subproblems obtain by fixing a subset of variables. More specifically, at each iteration k, given the current solution x^k,only a subset of variables W with cardinality $|W| \geq 2$, the so called working set, is involved in the minimization process, while the set of the remaining variables, $\overline{W}$, is kept fixed at the current value. Formally, the following subproblem is solved:

$$\begin{aligned} \min_{x_W} \quad & f(x_W, x^k_{\overline{W}}) \\ \text{s.t.} \quad & y_W^T x_W = -y_{\overline{W}}^T x_{\overline{W}}, \\ & 0 \leq x_W \leq Ce_W, , \end{aligned} \tag{2}$$

thus obtaining the new iterate $x^{k+1} = (x^\star_W, x^k_{\overline{W}})$.
The decomposition strategies used in this context can be divided into two classes: Sequential Minimal Optimization (SMO) methods [10],[2], where, at each iteration, only two variables are included in the set W, and methods where more than two variables at a time are used [5]. The advantage of using SMO approaches relies on the fact that the solution of subproblem (2) can be determined analytically, while in the second class of methods, the major cost of the single iteration, due to the use of optimization procedures to solve the subproblem, is balanced with the possibility of updating a larger number of variables at a time, thus reducing the number of iterations needed to converge.
As the major computational effort at each iteration is the evaluation of the kernel to determine the columns of the Hessian matrix, another commonly used strategy (both in SMO and non-SMO frameworks) is the caching technique, which consists in allocating some memory space (cache) to store the recently used columns of G. This procedure usually allows to avoid expensive kernel computations.
In this paper, we propose a new non-SMO first-order decomposition method for solving problem (1) that, by means of a suitably developed space transformation technique and a new caching strategy, enables to reduce the computational burden at a single iteration connected with the use of more than two variables at a time.

The paper is organized as follows. In Section 2 we give some notations and preliminary results. In Section 3, we describe the proposed method, its practical implementation and show some numerical results.

2. Notation and preliminary results

In this section we introduce some definitions and results needed in the sequel of the paper .

For every feasible point x, we denote the following sets of indices of active bounds:

$$L(x) = \{i : \ x_i = 0\}, \quad U(x) = \{i : \ x_i = C\}.$$

Further, at a feasible point x, the set of the feasible directions is the cone

$$D(x) = \{d \in R^n : \ y^T d = 0, \ d_i \geq 0, \ \forall \ i \in L(x), \ \text{and } d_i \leq 0, \ \forall \ i \in U(x)\}.$$

Given a feasible point x, we further introduce sets:

$$R(x) = \{0 < x_i < C\} \cup \{x_i = C \ \textit{and} \ y_i = -1\} \cup \{x_i = 0 \ \textit{and} \ y_i = +1\}$$
$$S(x) = \{0 < x_i < C\} \cup \{x_i = C \ \textit{and} \ y_i = 1\} \cup \{x_i = 0 \ \textit{and} \ y_i = -1\}.$$

which can be rewritten in the following way:

$$R(x) = \{0 < x_i \leq C \quad \textit{and} \ y_i = -1\} \cup \{0 \leq x_i < C \quad \textit{and} \ y_i = 1\} \tag{3}$$
$$S(x) = \{0 < x_i \leq C \quad \textit{and} \ y_i = 1\} \cup \{0 \leq x_i < C \quad \textit{and} \ y_i = -1\}. \tag{4}$$

2.1. Optimality Condition

The previous sets play an important role in defining a optimality condition for Problem (1)

Proposition 2.1 (Optimality conditions [6]) *A feasible point x^* is a solution of Problem (1) if and only if*

$$\min_{h \in R(x^*)} \frac{\nabla f(x^*)_h}{y_h} \geq \max_{h \in S(x^*)} \frac{\nabla f(x^*)_h}{y_h}. \tag{5}$$

For every feasible point x, which is not an optimal solution of Problem (1), a pair (i, j), with $i \in R(x)$, $j \in S(x)$, such that

$$\frac{\nabla f(x)_i}{y_i} < \frac{\nabla f(x)_j}{y_j} \tag{6}$$

is said a *violating pair*.

In order to characterize the pairs which most violates the optimality condition (5), the following sets of indices are defined:

$$I(x) = \left\{ i : \ i \in \arg\min_{h \in R(x)} \frac{\nabla f(x)_h}{y_h} \right\}, \tag{7}$$

$$J(x) = \left\{ j : \ j \in \operatorname{argmax}_{h \in S(x)} \frac{\nabla f(x)_h}{y_h} \right\}. \tag{8}$$

Then, at every no optimal feasible point x, a pair (i, j) such that $i \in I(x)$, $j \in J(x)$ it is said a *maximal violating pair*.

2.2. Sets of Sparse Feasible Directions

Now, we recall a special class of feasible directions having only two nonzero components and their important properties.

Given $i, j \in \{1, \ldots, n\}$, with $i \neq j$, we define a vector $d^{i,j} \in R^n$ in the following way

$$d_h^{i,j} = \begin{cases} 1/y_i, & \text{if } h = i \\ -1/y_j, & \text{if } h = j \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

For every $x \in \mathcal{F}$ we denote by $D_{RS}(x)$ the set of directions $d^{i,j}$ with $i \in R(x)$ and $j \in S(x)$, namely

$$D_{RS}(x) = \bigcup_{\substack{i \in R(x) \\ j \in S(x) \\ i \neq j}} d^{i,j}. \tag{10}$$

In the following proposition we recall some properties of the directions $d^{i,j}$ proved in [8] and [9].

Proposition 2.2. *Let $\hat{x}$ be a feasible point then*

i) for each pair $i \in R(\hat{x})$ and $j \in S(\hat{x})$, the direction $d^{i,j} \in R^n$ is a feasible direction at $\hat{x}$, i.e. $d \in D(\hat{x})$;

ii) $\hat{x}$ is a minimum point of Problem (1) *if and only if*

$$\nabla f(\hat{x})^T d^{i,j} \geq 0 \qquad \forall d^{i,j} \in D_{RS}(\hat{x}); \tag{11}$$

iii)

$$\begin{aligned} &D_{RS}(\hat{x}) \subseteq D(\hat{x}), \\ &\text{cone}\{D_{RS}(\hat{x})\} = D(\hat{x}). \end{aligned}$$

Furthermore, if the feasible point x is not a minimum point for Problem (1) then

- the definition (6) implies that the every violating pair $(\tilde{i}, \tilde{j})$ defines a feasible direction $d^{\tilde{i},\tilde{j}}$ which is a descent direction for the objective function of Problem (1) namely:

$$\nabla f(x)^T d^{\tilde{i},\tilde{j}} < 0.$$

- the definitions (7) and (8) imply that the pair $(\bar{\imath}, \bar{\jmath})$ such that $\bar{\imath} \in I(x)$ and $\bar{\jmath} \in J(x)$ defines a $d^{\bar{\imath},\bar{\jmath}}$ which solves the following problem

$$\begin{aligned} \min \quad & \nabla f(x)^T d \\ s.t. \quad & d \in D_{RS}(x). \end{aligned}$$

3. A new first-order method for SVM training

In this section, the method proposed is described. The followed approach is a compromise between that of LIBSVM [2] and that SVMLight [5]. In particular, at each iteration, the new algorithm differs from LIBSVM since it can update a larger number of variables and differs from SVMLight since it solves smaller and simpler subproblems.

3.1. Space Transformation Decomposition technique

One of the major difficulties for the solution of problem (1) concerns the treatment of the linear equality constraint

$$y^T x = 0. \tag{12}$$

Starting from a feasible point x^0, the proposed approach is based on determining, at each iteration k, a finite set

$$\mathcal{D}^k = \left\{ d^{i_\ell, j_\ell} \in R^n : \; i_\ell \in R(x^k), j_\ell \in S(x^k), \; \ell = 1, \ldots, q^k \right\}. \tag{13}$$

Definitions (3), (4), (9) and Proposition 2.2 ensure that these directions are into the nullspace $\mathbb{N}(y^T)$. In addition to maintaining the feasibility respect to (12), these directions allow to update only a subset of variables of the problem, following the decomposition guidelines justified in the introduction. Furthermore, in order to get simple subproblems to solve at each iterations, the important point is to use directions which are disjoint in the sense that

$$(i_1) \neq (j_1) \neq (i_2) \neq (j_2) \neq, \ldots, \neq (i_{q^k}) \neq (j_{q^k}).$$

If we move simultaneously along these directions, their particular structures guarantee that we only need to deal with the simpler box constraints.
The idea of our algorithm is to minimize the objective function by optimizing the steplenghts along these directions. At each iteration k, after having determined set $\mathcal{D}^k$, we define as new variables the steplenghts $\alpha_1, \alpha_2, \ldots, \alpha_{q^k}$ and we compute the new point x^{k+1} by solving the following box constrained formulation:

$$\begin{aligned} \min_{\alpha_1, \alpha_2, \ldots, \alpha_{q^k}} \quad & f(x_k + \sum_{\ell=1}^{q^k} \alpha_\ell \; d^{i_\ell j_\ell}) \\ 0 \leq x_k + \textstyle\sum_{\ell=1}^{q^k} \alpha_\ell \; d^{i_\ell j_\ell} \leq C. & \end{aligned} \tag{14}$$

obtaining $\alpha_1^k, \alpha_2^k, \ldots, \alpha_{q^k}^k$. Then, the new iterate is computed as

$$x^{k+1} = x^k + \sum_{\ell=1}^{q^k} \alpha_\ell^k \; d^{i_\ell j_\ell}.$$

3.2. The set of search directions

The idea is to choose q^k directions that are most descent for the objective function of Problem 1. To compute such directions, we draw our inspiration from the working set selection (WSS) strategy proposed in [7], that we report below.

Algorithm 1. WSS SVMLight

Data: x^k *feasible for problem* (1), $2q \geq 2$.

Step 1: *Sort* $\frac{\nabla f(\mathbf{x}^k)_i}{y_i}$ *in decreasing order.*

Step 2: *From the top of the sorted list sequentially set* $\bar{d}_i = -\frac{1}{y_i}$ *if* $x_i^k \in S(\mathbf{x}^k)$*; else set* $\bar{d}_i = 0$ *and bypass it.*
From the bottom of the sorted list sequentially set $\bar{d}_i = \frac{1}{y_i}$ *if* $x_i^k \in R(\mathbf{x}^k)$*; else set* $\bar{d}_i = 0$ *and bypass it.*
The assignment of $\bar{d}_i = -\frac{1}{y_i}$ *and* $\frac{1}{y_i}$ *is done symmetrically until either*

a) q elements of $\bar{\mathbf{d}}$ *are assigned to be* $-y_i$ *from the top and q elements of* $\bar{\mathbf{d}}$ *are assigned to be* y_i *from the bottom; or*

b) we cannot find $\bar{d}_i = -\frac{1}{y_i}$ *from the top and* $\bar{d}_i = \frac{1}{y_i}$ *from the bottom at the same time.*

Step 3: *Elements of* $\bar{\mathbf{d}}$ *not considered yet are assigned to be zeros.*

Such technique ([11]) finds the solution of problem

$$\begin{aligned} &\min_d \nabla f(x^k)^T d \\ &y^T d = 0, \quad -1 \leq d_i \leq 1, \quad i = 1 \dots l \\ &d_i \geq 0 \quad \text{if} \quad x_i^k = 0, \quad d_i \leq 0 \quad \text{if} \quad x_i^k = C, \\ &|\{d_i : d_i \neq 0\}| \leq 2q^k. \end{aligned} \tag{15}$$

Given a feasible point x^k for problem (1), the rationale of (15) is to find a feasible direction d^k that minimizes the linear approximation of the objective function in x^k, with at most $2q^k$ nonzero components, where q^k is a given positive integer.

Starting from the solution $\bar{d} = 0$, the (WSS) strategy of SVMLight algorithm, at each step ℓ finds a pair of indices i_ℓ from $S(x^k)$ and j_ℓ from $R(x^k)$ such that $d^{i_\ell j_\ell}$ is the most descent direction among the direction d^{ij}.

Then the algorithm sets $\bar{d}_{i_\ell} = -1/y_{i_k}$ and $\bar{d}_{j_\ell} = 1/y_{j_k}$; so the number of nonzero components of $\bar{d}$ increases of 2, and variables i_ℓ and j_ℓ are not anymore considered in the next iterates of the algorithm.

If it is impossible to find such a pair of variables or $|\{\bar{d}_i : \bar{d}_i \neq 0\}| = 2q^k$, the algorithm stops.

In SVMLight algorithm, the resulting descent direction $\bar{d}$ is not used as a search direction, but the variables corresponding to the nonzero components of the vector $\bar{d}$ are inserted in the working set, namely

$$W^k = \left\{ i_\ell, j_\ell : \ \ell = 1, \dots, q^k \right\}$$

Our method follows the same procedure with the difference that it stores all the produced directions $d^{i_\ell j_\ell}$, $\ell = 1, \dots, q^k$ which are descent, disjoint and linearly independent. These directions can be considered as columns of the fllowing matrix D^k:

$$D^k = \left[d^{i_1 j_1} \quad d^{i_2 j_2} \quad \dots \quad d^{i_{q^k} j_{q^k}}. \right] \tag{16}$$

By using this matrix it possible to perform the following space transformation:

$$x(\alpha) = x^k + D^k\alpha,$$

where $\alpha \in R^{q^k}$.

In [7] it is proved that $\bar{d}$ exactly solves problem (15) and that a given point $\bar{x}$ is optimal for (1) if and only if $\bar{d}$ is the null vector. Therefore the WSS technique generates no directions and the proposed algorithm does not produce a matrix D^k with at least one column if and only if $\bar{x}$ is optimal for (1).

3.3. SVM Reformulation

By using the matrix defined in the previous section, we can rewrite (14) as

$$\begin{aligned} \min_{\alpha} \quad & f(x^k + D^k\alpha) = \frac{1}{2}(x^k + D^k\alpha)^T G(x^k + D^k\alpha) - e^T(x^k + D^k\alpha) \\ & 0 \le (x^k + D^k\alpha) \le C, \end{aligned} \tag{17}$$

Then, by noticing that each row of D^k has at most one nonzero component and that $D^T e = 0$ we can rewrite (17) as follows

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T({D^k}^T G D^k)\alpha + \alpha^T({D^k}^T G x^k) \\ & l_\ell \le \alpha_\ell \le u_\ell \quad \ell = 1, \ldots, q^k, \end{aligned} \tag{18}$$

As α_i refers to a couple of variables say x_{i_ℓ} and x_{j_ℓ}, l_ℓ and u_ℓ are determined as

$$\begin{array}{llllll} l_\ell = \max\{-x_{i_\ell}, x_{j_\ell} - C\}, & u_\ell = \min\{x_{j_\ell}, C - x_{i_\ell}\} & \textit{if} & y_{i_\ell} = 1, & y_{j_\ell} = 1 \\ l_\ell = \max\{-x_{j_\ell}, x_{i_\ell} - C\}, & u_\ell = \min\{x_{i_\ell}, C - x_{j_\ell}\} & \textit{if} & y_{i_\ell} = -1, & y_{j_\ell} = -1 \\ l_\ell = \max\{-x_{i_\ell}, -x_{j_\ell}\}, & u_\ell = \min\{C - x_{j_\ell}, C - x_{i_\ell}\} & \textit{if} & y_{i_\ell} = 1, & y_{j_\ell} = -1 \\ l_\ell = \max\{x_{j_\ell} - C, x_{i_\ell} - C\}, & u_\ell = \min\{x_{i_\ell}, x_{j_\ell}\} & \textit{if} & y_{i_\ell} = -1, & y_{j_\ell} = 1 \end{array}$$

where we notice that $l_\ell \le 0$ and $u_\ell \ge 0$.

By solving Problem (18), using a suitably chosen algorithm, it is possible to obtain the vector $\alpha^k \in \mathbb{R}^{q^k}$ which produces the new point

$$x^{k+1} = x^k + D^k\alpha^k.$$

Since α^k is a minimum point of Problem (17) (or Problem (18)), by the KKT condition, we have:

$$\begin{aligned} {d^{i_\ell j_\ell}}^T \nabla f(x^{k+1}) &= \frac{\nabla f(x^{k+1})_{i_\ell}}{y_{i_\ell}} - \frac{\nabla f(x^{k+1})_{j_\ell}}{y_{j_\ell}} \ge 0 \quad \textit{if} \quad \alpha_\ell = 0 \\ {d^{i_\ell j_\ell}}^T \nabla f(x^{k+1}) &= \frac{\nabla f(x^{k+1})_{i_\ell}}{y_{i_\ell}} - \frac{\nabla f(x^{k+1})_{j_\ell}}{y_{j_\ell}} \ge 0 \quad \textit{if} \quad \alpha_\ell = 0 \end{aligned}$$

In conclusion, in the proposed method, the new point x^{k+1} is computed by solving a q^k dimensional box constrained subproblem and this is the main difference from the SVMLight approach where the new point derives from the solution of a $2q^k$ dimensional minimization problem with a linear equality constraint and box constraints on the variables.

3.4. Implementation

In order to speed up the optimization process, we adopted a caching strategy that consists in dedicating a part of the available memory to store the most recently used columns G_i of the hessian matrix (so that some kernel elements may not need to be recalculated). In particular, we used a new two-level caching technique. The first level tries to exploit intensively the advantages of the cache memory by using only descent directions related to elements already stored in the cache. If the first-level caching cannot find a sufficiently large number of descent directions, a second-level-caching is used to determine the largest possible number of directions considering all indexes of variables of the original problem; even in this case elements already available in cache are not recomputed.
Once all directions are determined, the box-constrained subproblem (17) is solved by a Newton-type method.
Below we report the detailed schemes of First-level-caching WSS, Second-level-caching WSS and STDA-SVM.

Algorithm 2. **First-level-caching WSS**

Data: x^k, $q > 1 \in \mathbb{N}$, $\eta \in (0, 1]$, $\mathcal{D}^k$, $R(x^k)$, $S(x^k)$ *and* Δ^k.

Set $m^k = 0$, $SC^{m^k} = \{i \in S(x^k) : i \in cache\}$ *and* $RC^{m^k} = \{i \in R(x^k) : i \in cache\}$

If $(SC^{m^k} = \emptyset$ ***or*** $RC^{m^k} = \emptyset)$ ***then*** ***return.***

While $(m^k \leq q)$ ***do***

If $(SC^{m^k} \neq \emptyset)$ ***then*** *set* $\bar{\imath} = \arg\max_{i \in SC^{m^k}} \dfrac{\nabla f_i(x^k)}{y_i}$, $SC^{m^k+1} = SC^{m^k} \setminus \{\bar{\imath}\}$

else *(STOP)*

If $(RC^{m^k} \neq \emptyset)$ ***then*** *set* $\bar{\jmath} = \arg\min_{j \in RC^{m^k}} \dfrac{\nabla f_j(x^k)}{y_j}$, $RC^{m^k+1} = RC^{m^k} \setminus \{\bar{\jmath}\}$

else *(STOP)*

If $\left(\dfrac{\nabla f_{\bar{\imath}}(x^k)}{y_{\bar{\imath}}} - \dfrac{\nabla f_{\bar{\jmath}}(x^k)}{y_{\bar{\jmath}}} > \eta\Delta^k\right)$ *set* $d_{\bar{\imath}}^{m^k} = -\dfrac{1}{y_{\bar{\imath}}}$ *and* $d_{\bar{\jmath}}^{m^k} = \dfrac{1}{y_{\bar{\jmath}}}$, *set* $d_s^{m^k} = 0 \quad \forall s \neq \bar{\imath}, \bar{\jmath}$ ***Set*** $\mathcal{D}^k = \mathcal{D}^k \cup \{d^{m^k}\}$.

else *(STOP)*

Set $m^k = m^k + 1$.

End While

if $(m^k \geq 2)$ ***then return***

else

Set $\mathcal{D}^k = \emptyset$ *and* ***return.***

Algorithm 3. Second-level-caching WSS

Data: x^k, $q > 1 \in \mathbb{N}$, $\mathcal{D}^k$, $R(x^k)$, $S(x^k)$, i_{mvp} *and* j_{mvp}.

Set $t^k = 1$, $d^1_{i_{mvp}} = -\dfrac{1}{y_{i_{mvp}}}$ *and* $d^1_{j_{mvp}} = \dfrac{1}{y_{j_{mvp}}}$, *set* $d^1_s = 0 \quad \forall s \neq \{i_{mvp}, j_{mvp}\}$, $\mathcal{D}^k = \{d^1\}$, $S^{t^k} = S(x^k) \setminus \{i_{mvp}\}$ *and* $R^{t^k} = R(x^k) \setminus \{j_{mvp}\}$.

While $(t^k \leq q)$ ***do***

If $(S^{t^k} \neq \emptyset)$ ***then*** *set* $\bar{\imath} = \arg\max\limits_{i \in S^{t^k}} \dfrac{\nabla f_i(x^k)}{y_i}$, $\quad S^{t^k+1} = S^{t^k} \setminus \{\bar{\imath}\}$

else *(STOP)*

If $(R^{t^k} \neq \emptyset)$ ***then*** *set* $\bar{\jmath} = \arg\min\limits_{j \in R^{t^k}} \dfrac{\nabla f_j(x^k)}{y_j}$, $\quad R^{t^k+1} = R^{t^k} \setminus \{\bar{\jmath}\}$

else *(STOP)*

If $\left(\dfrac{\nabla f_{\bar{\imath}}(x^k)}{y_{\bar{\imath}}} > \dfrac{\nabla f_{\bar{\jmath}}(x^k)}{y_{\bar{\jmath}}}\right)$ *set* $d^{t^k+1}_{\bar{\imath}} = -\dfrac{1}{y_{\bar{\imath}}}$ *and* $d^{t^k+1}_{\bar{\jmath}} = \dfrac{1}{y_{\bar{\jmath}}}$, *set* $d^{t^k+1}_s = 0 \quad \forall s \neq \bar{\imath}, \bar{\jmath}$

Set $\mathcal{D}^k = \mathcal{D}^k \cup \{d^{t^k+1}\}$.

else *(STOP)*

Set $t^k = t^k + 1$

End While

return.

Algorithm 4. **STDA-SVM**

Data: $x^0 = 0^n$, $q > 1 \in \mathbb{N}$, $\eta \in (0, 1]$, $\epsilon > 0$, $k = 0$.

Step 1: ***Set*** $\mathcal{D}^k = \emptyset$, $p^k = 0$ *and* ***Determine*** $S(x^k)$, *and* $R(x^k)$ *as in* (4) *and* (3) *and* ***Sort*** *in decreasing order* $\dfrac{\nabla f(x^k)}{y_i} \quad i = 1, \ldots, l.$

Determine $i_{mvp} = \arg \max_{i \in S(x^k)} \dfrac{\nabla f_i(x^k)}{y_i}$ *and* $j_{mvp} = \arg \min_{j \in R(x^k)} \dfrac{\nabla f_j(x^k)}{y_j}$.

Set $\Delta^k = \dfrac{\nabla f_{i_{mvp}}(x^k)}{y_{i_{mvp}}} - \dfrac{\nabla f_{j_{mvp}}(x^k)}{y_{j_{mvp}}}$.

If $(\Delta^k < \epsilon)$ ***then*** *(STOP),* x^k *is optimal.*

Step 2: ***Update*** $\mathcal{D}^k$ *by First-Level-Caching WSS.*

If $(\mathcal{D}^k = \emptyset)$ ***then Update*** $\mathcal{D}^k$ *by Second-Level-Caching WSS.*

Step 3: ***Set*** $p^k = card(\mathcal{D}^k)$ *and* ***Set*** *elements in* $\mathcal{D}^k$ *as columns of matrix* $D^k \in \mathbb{R}^{l \times p^k}$

Determine $\alpha^k \in \mathbb{R}^{p^k}$ *by solving problem:*

$$\min_{\alpha} \quad \tfrac{1}{2}\alpha^T ({D^k}^T G D^k)\alpha + \alpha^T ({D^k}^T G x^k)$$

$$l_i \leq \alpha_i \leq u_i \quad i = 1, \ldots, p^k.$$

Step 4: ***Set*** $x^{k+1} = x^k + D^k \alpha^k$, $k = k + 1$ *and* ***Go to*** *Step 1.*

STDA-SVM algorithm begins with the feasible starting point $x^0 = 0^n$ and takes as input parameter the positive integer $q > 1$ that is the maximum number of directions that the working set selection technique tries to find at each iteration.

At **Step 1**, the sorted list of scalars $\dfrac{\nabla f(x^k)}{y_i}$ and sets $S(x^k)$, $R(x^k)$ allows to determine the "most violating pair"; algorithm check optimality by the condition

$$\Delta^k = \frac{\nabla f_{i_{mvp}}(x^k)}{y_{i_{mvp}}} - \frac{\nabla f_{j_{mvp}}(x^k)}{y_{j_{mvp}}} < \epsilon \tag{19}$$

if (19) is satisfied there is no violation of the KKT conditions and x^k can be considered optimal (under a certain tolerance ϵ set by the user).

If the stopping criterion fails, at **Step 2**, the First-level caching WSS tries to find the largest possible number m^k (with $0 \leq m^k \leq q$) of descent directions, with nonzero indexes $\bar{\imath}$, $\bar{\jmath}$ related to columns already available in the cache memory, which satisfy the following condition:

$$\left(\frac{\nabla f_{\bar{\imath}}(x^k)}{y_{\bar{\imath}}} - \frac{\nabla f_{\bar{\jmath}}(x^k)}{y_{\bar{\jmath}}} > \eta \Delta^k \right), \tag{20}$$

with $\eta \in (0, 1]$. Parameter η allows to control how frequently the first-level-caching strategy is used: for large values of η it is hard to find descent directions satisfying condition (20), so that the algorithm tends to use more the second level caching (implying more kernel evaluations but better directions). On the other side, small values of η relax condition (20) and make it easier

to find in the cache a sufficient number of descent directions to compose matrix D^k (implying less kernel evaluations but less steep directions). The Second-level-caching WSS searches for the descent directions considering all indexes of variables (including the "most violating pair"); even if not specified in the scheme, the algorithm is implemented in order to exploit the cache also in this phase: for example, for a selected descent direction, if only one of the two indexes involving nonzero components corresponds to a column stored in the cache, that column will be not recalculated to construct the quadratic term ${D^k}^T G D^k$.
Once the set of directions $\mathcal{D}^k = \{d_1^k, d_2^k, \ldots, d_{p^k}^k\}$ is determined by the working set selection procedure, in **Step 3** the transformation matrix $D^k \in \mathbb{R}^{n \times p^k}$ is defined as $D^k = [d_1^k, d_2^k, \ldots, d_{p^k}^k]$. We notice that matrix D^k is full-column-rank because the directions are linearly indipendent by construction. Then the optimal vector $\alpha^k \in \mathbb{R}^{p^k}$ of steplenghts along these directions is calculated by solving problem (17) by a box-constrained Newton-type technique (see e.g. [1], [3] and the references therein). In **Step 4** the current iterate is updated as $x^{k+1} = x^k + D^k \alpha^k$ and the iterative process come back to **Step 1**.

3.5. Experiments

STDA-SVM code was obtained by suitably modifying the C++ code of LIBSVM. All the experiments have been carried out on a 64-bit Intel Core Duo with 3.16 Ghz $\times$ 2 and 4 gigabyte of RAM in a Linux environment.

Dealing with problems formulated as (1), in our experimentation we focused on datasets exploiting as much as possible the advantages of the dual form of the SVMs formulation ([4]), i.e. instances with a number of training samples much larger than the number of features. This type of datasets (available at LIBSVM site `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`) are not, in general, linearly separable and for this reason we used the nonlinear gaussian kernel for all tests.
The numerical results are rapresented in tables 1 and 2, where we report the number of training samples (n), the number of features of the dataset ($feat.$), the value of the upper bounds on the dual variables (C), the number of support vectors at the end of the minimization process (NSV), the number of decomposition iterations ($iter.$) and the runtime expressed in seconds ($time$).
For STDA-SVM, in all tests we set $q = 15$, $\epsilon = 0.001$ and $\eta = 0.1$; the small value of η depends on the fact that in preliminary experiments we noticed that an intense use of first-level-caching generally improves the time performances.
First we analyze the numerical results against SVMLight, reported in table 1. We decided to make this comparison because both STDA-SVM and SVMLight are designed to have a working set dimension scalable by the user in relation to the dimension of the problem; moreover (as mentioned above) the rationale underneath the working set selection of SVMLight is similar to that of STDA-SVM: the minimization of the linear model of the objective function in point x^k. The main difference between the two approaches is that while WSS of STDA-SVM produces directions that will be used in the minimization of the subproblem, the working set selection of SVMLight is only used to determine the variables to insert in W^k, though the relative subproblem is solved by some constrained nonlinear solver.
For SVMLight we set the stopping criterion tolerance $\epsilon = 0.001$ and $q = 30$ as maximum size of the subproblem as in STDA-SVM we use at most 15 directions so at most 30 variables are involved at each decomposition. For both training algorithms we adopt (as already anticipated)

a gaussian kernel with the same parameters. SVMLight was equipped with the LOQO solver for the solution of the subproblem.
As we can see in table 1, STDA-SVM outperforms SVMLight on almost all test with runtime savings often far above the 50% .

data_set	n	feat.	C	algorithm	NSV	iter.	time
a1a	32561	123	1	SVMLight	22265	8918	1502.34
				STDA-SVM	22249	3113	468.75
a1a	32561	123	10	SVMLight	21673	29141	3933.00
				STDA-SVM	21411	30248	779.22
a9a	48842	123	1	SVMLight	17663	7142	836.39
				STDA-SVM	17651	2409	294.52
a9a	48842	123	10	SVMLight	17077	21112	1187.59
				STDA-SVM	17014	24698	482.89
w3a	49749	300	1	SVMLight	2963	1264	61.90
				STDA-SVM	2942	774	41.71
w3a	49749	300	10	SVMLight	2504	2541	73.85
				STDA-SVM	2043	3641	60.57
ijcnn1	141691	22	1	SVMLight	22771	15156	2683.67
				STDA-SVM	22762	5403	765.03
ijcnn1	141691	22	10	SVMLight	16145	28138	5193.83
				STDA-SVM	16144	15401	759.41

Table 1: Comparison between STDA-SVM and SVMLight

The comparison with LIBSVM (reported in table 2) is motivated by the fact that LIBSVM is one of the most efficient training algorithm for nonlinear SVMs. LIBSVM adopt a so called SMO (Sequential Minimal Optimization) technique that consists in decomposing the original problem in a sequence of subproblem sized 2 so that the subproblem can be solved analytically. Since STDA-SVM (similarly to SVMLight) uses first-order information for the working set selection, in order to better verify the effectiveness of the space transformation approach, we used in our tests the 'first-order version' of LIBSVM.
As showed in table 2, STDA-SVM also gives better results in terms of CPU time with respect to LIBSVM. Anyway, differently from the previous comparison, the CPU time performances of the two codes are closer to each other. A particular trend can be noticed in the table: as we increase the value of parameter C, the CPU time savings of STDA-SVM with respect to LIBSVM improve significantly; in order to highlight this aspect we added in table 2 the results of the tests carried out with $C = 20$. This behavior is highly remarkable in the case of **ijcnn1.svm**, where STDA-SVM has worse performances than LIBSVM with $C = 1$ and great CPU time savings for higher values of C. This trend is even more evident when the number of support vector decreases as C grows. As the support vectors represent the variables $x_i > 0$, this behavior shows how STDA-SVM is much faster that a SMO approach to solve a problem with a sparse solution, as depicted in the figure.

data_set	n	features	C	algorithm	NSV	iter	time
a1a.svm	32561	123	1	LIBSVM	22236	17314	491.66
				STDA-SVM(15)	22249	3113	468.75
a1a.svm	32561	123	10	LIBSVM	21413	85406	872.31
				STDA-SVM(15)	21411	30248	779.22
a1a.svm	32561	123	20	LIBSVM	21204	164977	2029.21
				STDA-SVM(15)	21185	57516	1266.01
a9a.svm	48842	123	1	LIBSVM	17653	13162	305.31
				STDA-SVM(15)	17651	2409	294.52
a9a.svm	48842	123	10	LIBSVM	17021	64996	531.59
				STDA-SVM(15)	17014	24698	482.89
a9a.svm	48842	123	20	LIBSVM	16873	134411	1004.5
				STDA-SVM(15)	16868	48037	745.44
w3a.svm	49749	300	1	LIBSVM	2948	3295	40.89
				STDA-SVM(15)	2942	774	41.71
w3a.svm	49749	300	10	LIBSVM	2427	15309	68.35
				STDA-SVM(15)	2043	3641	60.57
w3a.svm	49749	300	20	LIBSVM	2284	24037	93.61
				STDA-SVM(15)	2165	6371	87.44
ijcnn1.svm	141691	22	1	LIBSVM	22767	17936	763.8
				STDA-SVM(15)	22762	5403	765.03
ijcnn1.svm	141691	22	10	LIBSVM	16145	50598	1118.02
				STDA-SVM(15)	16144	15401	759.41
ijcnn1.svm	141691	22	20	LIBSVM	13983	73910	1450.71
				STDA-SVM(15)	13982	17121	802.79

Table 2: Comparison between STDA-SVM and LIBSVM

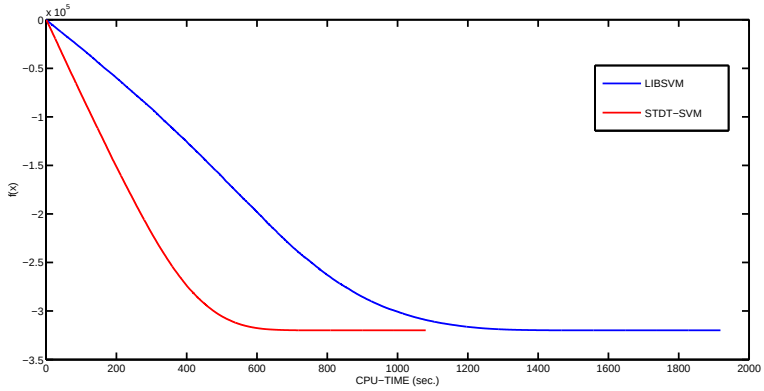

Figure 1: ijccn1 with C=20: cpu-time vs $f(x)$

References

[1] D. P. Bertsekas, (1982) Projected Newton methods for optimization problems with simple constraints *SIAM Journal on control and Optimization, 20(2), 221-246.*

[2] C.-C. Chang, C.-J. Lin, (2001) LIBSVM: A Library for Support Vector Machines *Software available at http://www.csie.ntu.edu//tildecjlin/libsvm.*

[3] M. De Santis, G. Di Pillo,S. Lucidi, (2012) An active set feasible method for large-scale minimization problems with bound constraints. *Computational Optimization and Applications, 53(2), 395-423*

[4] C.-W. Hsu, C.Wei, C.-C. Chang, and C.-J. Lin.(2003) A practical guide to support vector classification.

[5] T. Joachims, (1998) Making Large-Scale SVM Learning Practical. *Advances in Kernel Methods Support Vector Learning, B. Sch lkopf, C.J.C. Burges and A. Smola, eds., MIT Press, Cambridge, MA..*

[6] S. Keerthi and E. Gilbert, *Convergence of a generalized SMO algorithm for SVM*, Machine Learning, 46, pp. 351–360, 2002.

[7] C.J. Lin (2001) On the Convergence of the Decomposition Method for Support Vector Machines. *Neural Networks, IEEE Transactions.*

[8] S. Lucidi, L. Palagi, A. Risi, and M. Sciandrone, A convergent decomposition algorithm for support vector machines. *Computational Optimization and Applications*, Vol. 38, pp. 217-234, 2007.

[9] C.J. Lin, S. Lucidi, L. Palagi, A. Risi, and M. Sciandrone, ADecomposition Algorithm Model for Singly Linearly-Constrained Problems Subject to Lower and Upper Bounds. *Journal on Optimization Theory and Applications*, Vol. 141, pp. 107126, 2009.

[10] J.C.PLatt(1998) Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Support Vector Learning , B. Schlkopf, C. J. C. Burges, and A. J. Smola.*

[11] G.Zoutendijk(1960) Methods of Feasible Directions. *Amsterdam,The Netherlands: Elsevier.*