IDIAP RESEARCH REPORT

# Support Vector Machines for Large-Scale Regression Problems

Ronan Collobert [1] Samy Bengio [2]

IDIAP–RR 00-17

July 25, 2000

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 − 27 − 721 77 11
fax +41 − 27 − 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

1 IDIAP, CP 592, 1920 Martigny, Switzerland, collober@idiap.ch
2 IDIAP, CP 592, 1920 Martigny, Switzerland, bengio@idiap.ch

IDIAP Research Report 00-17

# Support Vector Machines for Large-Scale Regression Problems

Ronan Collobert Samy Bengio

July 25, 2000

**Abstract.** Support Vector Machines (SVMs) for regression problems are trained by solving a quadratic optimization problem which needs on the order of $l^2$ memory and time resources to solve, where $l$ is the number of training examples. In this paper, we propose a decomposition algorithm, *SVMTorch*[1], which is similar to *SVM-Light* proposed by Joachims [4] for classification problems, but adapted to regression problems. With this algorithm, one can now efficiently solve large-scale regression problems (more than 20000 examples). Comparisons with *Nodelib*, another SVM algorithm for large-scale regression problems from Flake and Lawrence [2] yielded significant time improvements.

[1] *SVMTorch* is available at `http://www.idiap.ch/learning/SVMTorch.html`.

# 1 Introduction

Vapnik has proposed in [8] a method to solve regression problems using Support Vector Machines. It has yielded excellent performances on many regression and time series prediction problems (see for instance [5, 1]). This paper proposes an efficient implementation of SVMs for large-scale regression problems. Let us first recall how it works.

Given a training set of $l$ *examples* $(\boldsymbol{x}_i, y_i)$ with $\boldsymbol{x}_i \in E$ and $y_i \in \mathbb{R}$, where $E$ is an Euclidean space with a scalar product denoted $(\,\cdot\,)$, we want to estimate the following linear regression:

$$f(\boldsymbol{x}) = (\boldsymbol{w} \cdot \boldsymbol{x}) + b$$

(with $b \in \mathbb{R}$) with a precision $\epsilon$. For this, we minimize

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l} |y_i - f(\boldsymbol{x}_i)|_\epsilon$$

where $\frac{1}{2}\|\boldsymbol{w}\|^2$ is a regularization factor, $C$ is a fixed constant, and $|.|_\epsilon$ is the $\epsilon$-insensitive loss function defined by Vapnik:

$$|z|_\epsilon = \max\{0, |z| - \epsilon\}.$$

Written as a constrained optimization problem, it amounts to minimizing

$$\tau(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^\star) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^\star)$$

subject to

$$((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) - y_i \leq \epsilon + \xi_i \tag{1}$$

$$y_i - ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) \leq \epsilon + \xi_i^\star \tag{2}$$

$$\xi_i, \xi_i^\star \geq 0.$$

To generalize to non-linear regression, we replace the dot product with a kernel $k(\cdot)$. Then, introducing Lagrange multipliers $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^\star$, the optimization problem can be stated as:

Minimize the function

$$\mathcal{W}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) = \frac{1}{2}(\boldsymbol{\alpha}^\star - \boldsymbol{\alpha})^{\mathrm{T}} K(\boldsymbol{\alpha}^\star - \boldsymbol{\alpha}) - (\boldsymbol{\alpha}^\star - \boldsymbol{\alpha})^{\mathrm{T}}\boldsymbol{y} + \epsilon(\boldsymbol{\alpha}^\star + \boldsymbol{\alpha})^{\mathrm{T}}\mathbf{1} \tag{3}$$

subject to

$$(\boldsymbol{\alpha} - \boldsymbol{\alpha}^\star)^{\mathrm{T}}\mathbf{1} = 0 \tag{4}$$

and

$$0 \leq \alpha_i^\star,\ \alpha_i \leq C, \quad i = 1...l \tag{5}$$

where $K$ is the matrix with coefficients $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The estimate of the regression function at a given point is then

$$f(\boldsymbol{x}) = \sum_{i=1}^{l}(\alpha_i^\star - \alpha_i)k(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

where $b$ is computed using the fact that (1) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$ and (2) becomes an equality with $\xi_i^\star = 0$ if $0 < \alpha_i^\star < C$.

Solving the minimization problem (3) under the constraints (4) and (5) needs resources on the order of $l^2$ and is thus difficult for problems with large $l$.

In this paper, we propose a method to solve such problems efficiently using a decomposition algorithm similar to the one proposed by Joachims [4] in the context of classification problems. In the next section, we give the general algorithm and explain in more details each if its main steps, as well as a discussion on some important implementation issues, such as a way to efficiently handle the kernel matrix computation. In the experiment section, we first compare this new algorithm to *Nodelib* [2], another SVM algorithm for large-scale regression problems, and then show how the size of the internal memory allocated to the resolution of the problem is related to the time needed to solve it.

# 2 The Decomposition Algorithm

As in the classification algorithm proposed by Joachims [4], which was based on a idea from Osuna *et al* [6], our regression algorithm is subdivided into the following four steps:

1. Select $q$ variables as the new working set, called $\mathcal{S}$.
2. Fix the other variables $\mathcal{F}$ to their current values and solve the problem (3) with respect to $\mathcal{S}$.
3. Search for variables that are stuck to 0 or $C$ and that will probably not change anymore. This is the *shrinking* phase.
4. Test if the optimization is finished or if we go back to the first step.

## 2.1 Selection of a New Working Set

We propose to select a new set of variables such that the overall criterion will be the most optimized. In order to select such working set, we use the same idea as Joachims [4]: simply search for the optimal gradient descent direction $\boldsymbol{p}$ which is *feasible* and which has only $q$ non-null components. The variables corresponding to these components will be our new working set $\mathcal{S}$.

We thus need to minimize:

$$\mathcal{V}(\boldsymbol{p}) = \left(\mathcal{W}'(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star)\right)^{\mathrm{T}} \boldsymbol{p} \tag{6}$$

with

$$\boldsymbol{p} = (d_1 \ldots d_l,\, d_1^\star \ldots d_l^\star)^{\mathrm{T}}$$

subject to:

$$\mathbf{1}^{\mathrm{T}} \boldsymbol{d} - \mathbf{1}^{\mathrm{T}} \boldsymbol{d}^\star = 0 \tag{7}$$

$$\begin{array}{lll} d_i \geq 0 & \text{for} & i \text{ such that } \alpha_i = 0 \\ d_i^\star \geq 0 & \text{for} & i \text{ such that } \alpha_i^\star = 0 \\ d_i \leq 0 & \text{for} & i \text{ such that } \alpha_i = C \\ d_i^\star \leq 0 & \text{for} & i \text{ such that } \alpha_i^\star = C \end{array} \tag{8}$$

and

$$-\mathbf{1} \;\leq\; \boldsymbol{p} \;\leq\; \mathbf{1} \tag{9}$$

$$\text{card}\{d_i \;:\; d_i \neq 0\} = q. \tag{10}$$

Since we are searching for an optimal descent direction, which is a direction where the scalar product with the gradient is the smallest, we want indeed to minimize (6). The conditions (7)-(8) are necessary to ensure the feasibility of the obtained direction. The condition (9) is there only to ensure that the problem has a solution. Finally, (10) is imposed because we are searching for a direction with only $q$ non-null components.

Note that the derivative of $\mathcal{W}$ can be easily computed:

$$\mathcal{W}^{'}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{\star}) = \left( \begin{array}{c} K\,(\boldsymbol{\alpha} - \boldsymbol{\alpha}^{\star}) + \boldsymbol{y} + \mathbf{1}\,\epsilon \\ K\,(\boldsymbol{\alpha}^{\star} - \boldsymbol{\alpha}) - \boldsymbol{y} + \mathbf{1}\,\epsilon \end{array} \right).$$

In order to solve this problem, it thus suffices to consider

$$\omega_i = \delta_i\, \mathcal{W}_i^{'}$$

where $\delta_i = 1$ for $1 \le i \le l$ and $\delta_i = -1$ for $l+1 \le i \le 2l$. Let us force $q$ to be even, and let us sort the $\omega_i$ in reverse order. Let us then denote $\varphi$ as the bijection of $\{1 \dots 2l\}$ into itself such that the $\omega_{\varphi(i)}$ are sorted. Let us then select the $q/2$ first indices $\varphi(i)$ such that:

$$\text{if } \varphi(i) \le l, \text{ we have } 0 < \alpha_{\varphi(i)} \le C$$

$$\text{if } \varphi(i) > l, \text{ we have } 0 \le \alpha^{\star}_{\varphi(i)-l} < C$$

and let us select also the $q/2$ last indices $\varphi(i)$ such that:

$$\text{if } \varphi(i) \le l, \text{ we have } 0 \le \alpha_{\varphi(i)} < C$$

$$\text{if } \varphi(i) > l, \text{ we have } 0 < \alpha^{\star}_{\varphi(i)-l} \le C.$$

Since we are searching for exactly $q$ variables, the $\varphi(i)$ must be distinct. We could have to reduce $q$ if one variable is selected twice.

Let us now denote $c_i,\ i = 1 \dots q$ the $q$ indices we just chose, we note that the direction which has for $j^{\text{th}}$ component 0 if $j \notin \{c_1 \dots c_q\}$, $-\delta_j$ if $j \in \{c_1 \dots c_{\frac{q}{2}}\}$ and $\delta_j$ if $j \in \{c_{\frac{q}{2}+1} \dots c_q\}$, is a solution of the minimization problem (6) [1].

Our new working set $\mathcal{S}$ is then composed of the $q$ variables corresponding to the indices $c_i$ (where an index $\le l$ corresponds to $\alpha_{c_i}$ and an index $> l$ corresponds to $\alpha^{\star}_{c_i - l}$).

[1] To see that, let us go back to the minimization problem of

$$(z_1, \dots z_l) \longmapsto \sum_{i=1\dots l} \omega_i\, z_i \tag{11}$$

subject to

$$\sum_i z_i = 0 \tag{12}$$

$$-1 \le z_i \le 1 \tag{13}$$

and

$$\text{card}\{z_i,\, z_i \neq 0\} = q \tag{14}$$

with $z_i = \delta_i p_i$. (The reasoning is the same if we take the constraints (8) into account).

In the case where $l = q = 2r$, it is easy to see that the minimum is obtained for $z_i = -1$ if $i = 1 \dots r$ and $z_i = 1$ if $i = r+1 \dots q$: if for instance $z_{i_0}$ is augmented by $\gamma \ge 0$ for a $i_0 \le r$, then one needs to compensate by $-\gamma$ another $z_j$ to keep (12). Since we want to minimize (11), the best thing to do, knowing that the $\omega_i$ are sorted in reverse order and keeping in mind the constraint (13), is to modify $z_q$ and thus to fix $z_q = 1 - \gamma$. Equation (11) is then augmented by $(\omega_{i_0} - \omega_q)\gamma$, which is a positive value because the $\omega_i$ are sorted and thus we get out of the minimum.

In the case where $l > q = 2r$, suppose we found a $\boldsymbol{z}$ which is a solution of (11). Let us denote $k_1 \dots k_q$ the $q$ indices of the components of $\boldsymbol{z}$ which are non-nulls. Using the same argument as in the previous paragraph, it is clear that $z_{k_1} \dots z_{k_r} = -1$ and that $z_{k_r+1} \dots z_{k_q} = 1$. In other words, if $\boldsymbol{z}$ is a solution to our problem, then we necessarily have $z_{k_i} = \pm 1$. Considering again the order of the $\omega_i$, is becomes evident that we have to take $(k_1 = 1) \dots (k_r = r)$ and $(k_{r+1} = l - r) \dots (k_q = l)$.

## 2.2 Solving the Sub-Problem

We want to solve the problem (3) taking into account only variables $\mathcal{S}$. To simplify the notation, let us define

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\alpha} \\ -\boldsymbol{\alpha}^\star \end{pmatrix}$$

and

$$\tilde{K} = \begin{pmatrix} K & K \\ K & K \end{pmatrix}$$

as well as

$$\boldsymbol{b} = \begin{pmatrix} -\boldsymbol{y} - \boldsymbol{1}\,\epsilon \\ -\boldsymbol{y} + \boldsymbol{1}\,\epsilon \end{pmatrix}.$$

The problem (3) is thus equivalent to minimize

$$\tilde{\mathcal{W}}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}}\tilde{K}\boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{b} \tag{15}$$

subject to:

$$\boldsymbol{\beta}^{\mathrm{T}}\,\boldsymbol{1} = 0 \tag{16}$$

and

$$0 \le \delta_i\,\beta_i \le C, \quad i = 1...2l \tag{17}$$

where again $\delta_i = 1$ for $1 \le i \le l$ and $\delta_i = -1$ for $l+1 \le i \le 2l$.

Now let us suppose we can decompose each of the following variables into two parts (after having reordered the variables accordingly): the first one corresponds to variables $\mathcal{S}$ and the second part corresponds to the fixed variables $\mathcal{F}$:

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_{\mathcal{S}} \\ \boldsymbol{\beta}_{\mathcal{F}} \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} \boldsymbol{b}_{\mathcal{S}} \\ \boldsymbol{b}_{\mathcal{F}} \end{pmatrix}$$

and

$$\tilde{K} = \begin{pmatrix} \tilde{K}_{\mathcal{S}\mathcal{S}} & \tilde{K}_{\mathcal{S}\mathcal{F}} \\ \tilde{K}_{\mathcal{F}\mathcal{S}} & \tilde{K}_{\mathcal{F}\mathcal{F}} \end{pmatrix}.$$

Replacing these variables in (15), (16) and (17), and taking into account the fact that $\tilde{K}_{\mathcal{S}\mathcal{F}}^{\mathrm{T}} = \tilde{K}_{\mathcal{F}\mathcal{S}}$, the minimization problem is now

$$\tilde{\mathcal{W}}(\boldsymbol{\beta}_{\mathcal{S}}) = \frac{1}{2}\boldsymbol{\beta}_{\mathcal{S}}^{\mathrm{T}}\tilde{K}\boldsymbol{\beta}_{\mathcal{S}} - \boldsymbol{\beta}_{\mathcal{S}}^{\mathrm{T}}\left(\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_{\mathcal{F}}\right) \tag{18}$$

(removing the constants that depend only on $\mathcal{F}$), subject to

$$\boldsymbol{\beta}_{\mathcal{S}}^{\mathrm{T}}\,\boldsymbol{1} = -\boldsymbol{\beta}_{\mathcal{F}}^{\mathrm{T}}\,\boldsymbol{1} \tag{19}$$

and

$$0 \le \tilde{\delta}_i\,\beta_{\mathcal{S}_i} \le C, \quad i = 1...q \tag{20}$$

where $\tilde{\delta}_i = 1$ if the $i^{\mathrm{th}}$ variable in the set $\mathcal{S}$ corresponds to an $\alpha_i$, $\tilde{\delta}_i = -1$ if it corresponds to an $\alpha_i^\star$.

Minimizing (18) under the constraints (19) and (20) can be realized using a constrained quadratic optimizer, such as a conjugate gradient method with projection or an interior point method [3].

Moreover, following Platt's idea in *SMO* [7], if one fixes the size of the working set $\mathcal{S}$ to two, the problem can also be solved analytically.

This particular case is important because experimental results show that it often gives the fastest convergence times. We thus detailed it here. Let us simplify again the notation:

$$\boldsymbol{\beta}_{\mathcal{S}} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

$$\boldsymbol{h} = \boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{S}\mathcal{F}} \boldsymbol{\beta}_{\mathcal{F}}$$

$$\zeta = -\boldsymbol{\beta}_{\mathcal{F}}^{\mathrm{T}} \mathbf{1}$$

$$\tilde{K}_{\mathcal{S}} = \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}.$$

Minimizing (18) under the constraints (19) and (20) is thus equivalent to minimizing

$$(z_1,\, z_2) \longmapsto \frac{1}{2}\left(k_{11}\, z_1^2 + k_{22}\, z_2^2 + 2k_{12}\, z_1\, z_2\right) - h_1 z_1 - h_2 z_2 \tag{21}$$

subject to

$$z_1 + z_2 = \zeta \tag{22}$$

and

$$0 \leq \tilde{\delta}_1\, z_1,\ \tilde{\delta}_2\, z_2 \leq C. \tag{23}$$

We are searching for a minimum in (21) with respect to $z_1$ along the line (22). By inserting (22) into (21), and after some derivations, it is now equivalent to minimizing

$$\Lambda : z_1 \longmapsto \frac{1}{2}\left(k_{11} - 2k_{12} + k_{22}\right) z_1^2 + \left[\left(k_{12} - k_{22}\right)\zeta - h_1 + h_2\right] z_1.$$

In the case[2] where $\eta = k_{11} - 2k_{12} + k_{22} > 0$, this function has a unique minimum for

$$z_1^o = \frac{\left(k_{22} - k_{12}\right)\zeta + h_1 - h_2}{\eta}.$$

Let us now consider the constraints (22) and (23). They force $z_1$ to stay between $L$ and $H$ where

$$\left.\begin{array}{l} L = \max(0,\ \zeta - C) \\ H = \min(C,\, \zeta) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = 1 \text{ and } \tilde{\delta}_2 = 1$$

$$\left.\begin{array}{l} L = \max(0,\ \zeta) \\ H = \min(C,\, \zeta + C) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = 1 \text{ and } \tilde{\delta}_2 = -1$$

$$\left.\begin{array}{l} L = \max(-C,\ \zeta - C) \\ H = \min(0,\, \zeta) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = -1 \text{ and } \tilde{\delta}_2 = 1$$

$$\left.\begin{array}{l} L = \max(-C,\ \zeta) \\ H = \min(0,\, \zeta + C) \end{array}\right\} \quad \text{if } \tilde{\delta}_1 = -1 \text{ and } \tilde{\delta}_2 = -1.$$

Thus, taking

$$z_1^{o,\,c} = \begin{cases} H & \text{if } z_1^o > H \\ z_1^o & \text{if } L \leq z_1^o \leq H \\ L & \text{if } z_1^o < L \end{cases}$$

[2]Note that this is the most common case. For instance for a gaussian kernel with distinct examples $\boldsymbol{x}_i$, it is easy to see that is is always the case.

and

$$z_2^o = \zeta - z_1^{o,\,c}$$

the minimum of (21) under the constraints (22) and (23) is obtained at $(z_1^{o,\,c},\, z_2^o)$ if $\eta > 0$. In the pathological case where $\eta \leq 0$, it is clear that the solution

$$z_1^o = \begin{cases} L & \text{if } \Lambda(L) < \Lambda(H) \\ H & \text{if } \Lambda(L) \geq \Lambda(H) \end{cases}$$

and

$$z_2^o = \zeta - z_1^o$$

is the minimum.

## 2.3 Shrinking

The idea of shrinking is to remove some variables that are stuck to the bounds 0 or $C$, and that will *a priori* not change anymore. To do this, we use the fact that $(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star)$ minimizes the problem (3) under the constraints (4) and (5) *if and only if* there exists numbers $\boldsymbol{\lambda}^{up} \in \mathbb{R}^{2l}$, $\boldsymbol{\lambda}^{low} \in \mathbb{R}^{2l}$, $\lambda^{eq} \in \mathbb{R}$ that verify the following *KKT conditions*:

$$\mathcal{W}'(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) + \lambda^{eq} \begin{pmatrix} \mathbf{1} \\ -\mathbf{1} \end{pmatrix} - \boldsymbol{\lambda}^{low} + \boldsymbol{\lambda}^{up} = \mathbf{0} \tag{24}$$

$$\lambda_i^{low}\, \alpha_i = 0, \quad i = 1 \ldots l \quad \text{and} \quad \lambda_i^{low}\, \alpha_{i-l}^\star = 0, \quad i = (l+1) \ldots 2l \tag{25}$$

$$\lambda_i^{up}\,(\alpha_i - C) = 0, \quad i = 1 \ldots l \quad \text{and} \quad \lambda_i^{up}\,\left(\alpha_{i-l}^\star - C\right) = 0, \quad i = (l+1) \ldots 2l \tag{26}$$

$$\boldsymbol{\lambda}^{low} \geq \mathbf{0} \tag{27}$$

$$\boldsymbol{\lambda}^{up} \geq \mathbf{0} \tag{28}$$

$$(\boldsymbol{\alpha} - \boldsymbol{\alpha}^\star)^{\mathrm{T}}\, \mathbf{1} = 0 \tag{29}$$

$$\mathbf{0} \leq \boldsymbol{\alpha}^\star,\ \boldsymbol{\alpha} \leq \boldsymbol{C}. \tag{30}$$

Note that if $\lambda_i^{low} > 0$, then the corresponding variable is equal to 0. Also, if $\lambda_i^{up} > 0$, then the corresponding variable is equal to $C$. The idea is thus to search at each iteration for variables $\boldsymbol{\lambda}^{up}$, $\boldsymbol{\lambda}^{low}$ and $\lambda^{eq}$ that verify *as well as possible*[3] the equations (24)-(28), and to remove a variable which is stuck at 0 if its coefficient $\lambda_i^{low}$ is strictly positive (or just below a constant $\epsilon_{shrink}$) during a given number of iterations. Using the same idea, we also eliminate a variable which is stuck at $C$ if its coefficient $\lambda_i^{up}$ stays strictly positive for a sufficient number of iterations.

To estimate $\boldsymbol{\lambda}^{low}$ or $\boldsymbol{\lambda}^{up}$, we start by estimating $\lambda^{eq}$ (note that if $0 < \alpha_i < C$ then $\lambda_i^{low} = \lambda_i^{up} = 0$ and if $0 < \alpha_i^\star < C$ then $\lambda_{i+l}^{low} = \lambda_{i+l}^{up} = 0$). Noting

$$A = \{i,\ 0 < \alpha_i < C\}, \quad B = \{i,\ 0 < \alpha_i^\star < C\}$$

we have (with $\hat{\lambda}$ standing for an estimation of $\lambda$) :

$$\hat{\lambda}^{eq} = \frac{1}{|A \cup B|} \left( \sum_{i \in B} \mathcal{W}'_{i+l}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) - \sum_{i \in A} \mathcal{W}'_i(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star) \right). \tag{31}$$

[3] If we were *really* able to find such variables, this would mean that $(\boldsymbol{\alpha}, \boldsymbol{\alpha}^\star)$ is a solution to our problem.

Then if we have

$$\begin{array}{llllll} \alpha_i = 0 & \text{we compute} & \hat{\lambda}_i^{low} & = & \hat{\lambda}^{eq} & + & \mathcal{W}_i^{'} \\ \alpha_i^\star = 0 & \text{we compute} & \hat{\lambda}_{i+l}^{low} & = & -\hat{\lambda}^{eq} & + & \mathcal{W}_{i+l}^{'} \\ \alpha_i = C & \text{we compute} & \hat{\lambda}_i^{up} & = & -\hat{\lambda}^{eq} & - & \mathcal{W}_i^{'} \\ \alpha_i^\star = C & \text{we compute} & \hat{\lambda}_{i+l}^{up} & = & \hat{\lambda}^{eq} & - & \mathcal{W}_{i+l}^{'} \end{array} \tag{32}$$

and if a variable stays a sufficient number of iterations at 0 (or $C$) with its corresponding coefficient $\hat{\lambda}_j^{low} > \epsilon_{shrink}$ (or $\hat{\lambda}_j^{up} > \epsilon_{shrink}$), then we remove it from the problem.

## 2.4 Termination Criterion

Given what has been said in the section on *shrinking*, if we can always have (29) and (30) during the resolution of a sub-problem, a *reasonable* termination criterion is to verify that the $\lambda$ estimated by (31) and (32) verifies the conditions (24)-(28) with a given precision $\epsilon_{end}$.

Thus we simply verify that

$$\begin{array}{rc} \text{for } i \text{ such that } 0 < \delta_i\, \beta_i < C: & \lambda^{eq} - \epsilon_{end} \le -\delta_i \mathcal{W}_i^{'}(\boldsymbol{\alpha},\, \boldsymbol{\alpha}^\star) \le \lambda^{eq} + \epsilon_{end} \\ \text{for } i \text{ such that } \beta_i = 0: & \mathcal{W}_i^{'}(\boldsymbol{\alpha},\, \boldsymbol{\alpha}^\star) + \delta_i \lambda^{eq} \ge -\epsilon_{end} \\ \text{for } i \text{ such that } \beta_i = C: & \mathcal{W}_i^{'}(\boldsymbol{\alpha},\, \boldsymbol{\alpha}^\star) + \delta_i \lambda^{eq} \le \epsilon_{end} \end{array}$$

with $\delta_i = 1$ for $1 \le i \le l$, $\delta_i = -1$ for $(l+1) \le i \le 2l$ and

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\alpha} \\ -\boldsymbol{\alpha}^\star \end{pmatrix}.$$

## 2.5 Implementation Details

Note that in all the steps needed during one iteration, only two of them could be time consuming: the one that computes $\mathcal{W}_i^{'}$ and the one that computes $\boldsymbol{b}_\mathcal{S} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_\mathcal{F}$ in (18).

We therefore propose to keep in memory a table of $\mathcal{W}_i^{'}$. Moreover, to update this variable, we can see that

$$\begin{array}{ll} \text{for } i \le n & \mathcal{W}_i^{'\,(t+1)} = \mathcal{W}_i^{'\,(t)} + \sum_{j\in S_1} K_{ij}\left(\alpha_j^{(t+1)} - \alpha_j^{(t)}\right) - \sum_{j\in S_2} K_{ij}\left(\alpha_j^{\star\,(t+1)} - \alpha_j^{\star\,(t)}\right) \\ \text{for } i > n & \mathcal{W}_i^{'\,(t+1)} = \mathcal{W}_i^{'\,(t)} - \sum_{j\in S_1} K_{ij}\left(\alpha_j^{(t+1)} - \alpha_j^{(t)}\right) + \sum_{j\in S_2} K_{ij}\left(\alpha_j^{\star\,(t+1)} - \alpha_j^{\star\,(t)}\right) \end{array}$$

where

$$S_1 = \{i,\ \ \alpha_i \in \mathcal{S}\} \ \text{ and } \ S_2 = \{i,\ \ \alpha_i^\star \in \mathcal{S}\}.$$

For the computation of $\boldsymbol{b}_\mathcal{S} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_\mathcal{F}$, we can use the following trick:

$$\begin{array}{rcl} \left(\boldsymbol{b}_\mathcal{S} - \tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_\mathcal{F}\right)_i & = & (\boldsymbol{b}_\mathcal{S})_i - \left(\tilde{K}_{\mathcal{S}\mathcal{F}}\boldsymbol{\beta}_\mathcal{F} + \tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_\mathcal{S}\right)_i + \left(\tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_\mathcal{S}\right)_i \\ & = & \begin{cases} -\mathcal{W}_i + \left(\tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_\mathcal{S}\right)_i & if\ i \le n \\ \mathcal{W}_i + \left(\tilde{K}_{\mathcal{S}\mathcal{S}}\boldsymbol{\beta}_\mathcal{S}\right)_i & if\ i > n. \end{cases} \end{array}$$

With these two ideas, one can reduce considerably the computational time: instead of computing all the lines of the matrix $K$, one can only compute the lines corresponding to the variables in $\mathcal{S}$.

Since we only need these lines for the computations, and since it quickly becomes intractable for large problems to keep all the matrix $K$ in memory (the size of the matrix being quadratic with respect to the number of examples), it is interesting to implement a *cache* that keeps in memory the lines of $K$ that corresponds to the most used variables instead of recomputing them at each iteration.

# 3 Experimental Results

## 3.1 Speed Comparisons

We compared our SVM implementation for regression problems (named *SVMTorch*) to the one from Flake and Lawrence [2] using their publicly available software *Nodelib*, which is an enhanced version of *SMO* [7].

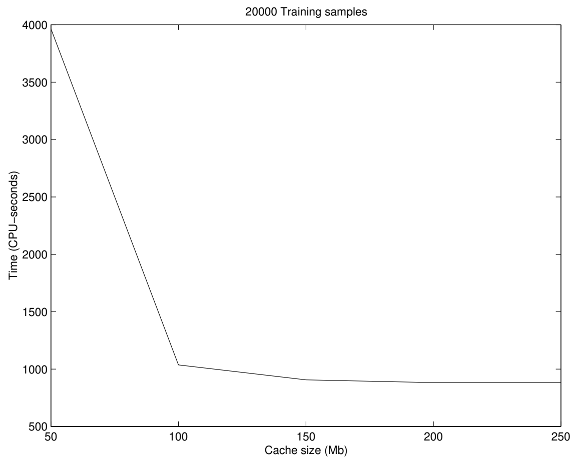
Both these algorithms use an internal cache in order to be able to solve large-scale problems. All the experiments presented in this section have been done on a SPARC Ultra-10 440Mhz, with the *gcc* compiler. The parameters of the algorithms were not chosen to obtained the best generalization performances, since the goal was to *compare the speed* of the algorithms. Both programs used the same parameters with regard to cache, precision, etc... For *Nodelib*, the other parameters were set using the default values proposed by the authors. All the programs were compiled using *double* precision. Finally, each program was trained/tested on the same data for a given *benchmark*.

We compared the programs on two different tasks, using up to three different training set sizes. The first task was to predict the average number of sunspots of one year, given the average number of sunspots of the previous 12 years. Since we had access to daily data, we were able to artificially create one input/output pair for each day: at each day, we compute the yearly average of the year starting the next day, which has to be predicted using the 12 previous yearly averages. Using this technique, we had access to 43000 examples with input dimension equal to 12. The last 3000 examples served as the test set, while we created 3 different training sets of varying sizes: 5000, 20000 and 40000. The second task was created artificially using the daily sunspot series in order to test the algorithm with a big input dimension. We selected it to be equal to 240. The test set had 2000 examples while we created 2 different training sets of varying sizes: 5000 and 20000.

Table 1 gives the computation time results of all the experiments in CPU-seconds. In these experiments, the following SVM parameters were used: $C = 1000$ and precision $= 0.01$. As it can be seen, *SVMTorch* easily outperformed *Nodelib* in all experiments: the generalization performance of both algorithm as well as the number of support vectors found were similar but the time spent to find the solution was largely less for *SVMTorch* than for *Nodelib*. However, it seems that as the number of training examples grows, the speed difference between both algorithms narrows. The speed difference is also smaller when the input dimension is higher. A further analysis should probably be done in order to understand these difference variations.

| **Yearly Sunspots** ($\sigma = 900$, $\epsilon = 20$) | | | | | |
|---|---|---|---|---|---|
| | Train size | # of SV | Train MSE | Test MSE | # of CPU-seconds |
| Nodelib | 5000 | 431 | 146.15 | 246.18 | 245 |
| SVMTorch | | 432 | 146.14 | 246.15 | 3 |
| Nodelib | 20000 | 1151 | 126.59 | 287.60 | 1253 |
| SVMTorch | | 1155 | 126.58 | 287.64 | 30 |
| Nodelib | 40000 | 1911 | 119.46 | 363.88 | 2499 |
| SVMTorch | | 1871 | 119.76 | 369.22 | 94 |
| **Artificial Data** ($\sigma = 10$, $\epsilon = 0.5$) | | | | | |
| | Train size | # of SV | Train MSE | Test MSE | # of CPU-seconds |
| Nodelib | 5000 | 397 | 0.09 | 4.82 | 825 |
| SVMTorch | | 394 | 0.10 | 4.66 | 90 |
| Nodelib | 20000 | 1013 | 0.08 | 0.60 | 5598 |
| SVMTorch | | 1002 | 0.08 | 0.60 | 1037 |

Table 1: Comparative speed results between *SVMTorch* and *Nodelib* on two different databases and up to 3 different training set sizes. All the experiments used an internal cache of 100Mb.



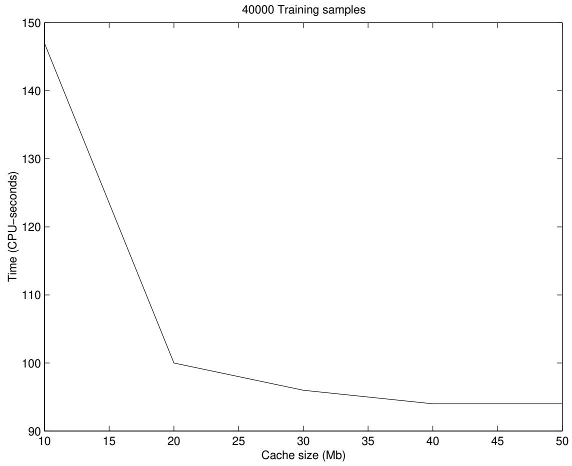




Figure 1: Evolution of the convergence speed (in seconds) with respect to the size of the internal cache (in Mb). The left part gives the computational time for the artificial dataset trained with 20000 examples (in dimension 240), while the right part gives the computational time for the sunspots dataset trained with 40000 examples (in dimension 12).

### 3.2 Analysis of the Cache Size

In order to show the impact of the cache size on the time needed to train an SVM, we have done two series of experiments, one for each dataset, varying the size of the cache (from 10 to 50 Mb for the sunspots dataset and from 50 to 250 for the artificial dataset). Figure 1 shows the results: the more cache you can afford, the faster the algorithm will find a solution, but the optimal size of the cache is of course related to the number of training samples: after a given cache size, no more speed improvement can be achieved.

## 4 Conclusion

We have presented a new decomposition algorithm intended to efficiently solve large-scale regression problems using SVMs. This algorithm followed the same principles as the one from Joachims [4] for his classification algorithm. We also proposed to set the size of the sub-problems to 2 in order to solve it analytically as it is done in SMO [7]. An internal cache keeping part of the kernel matrix in memory enables the program to solve large problems without the need to keep quadratic resources in memory and without the need to recompute every kernel evaluation, which leads to an overall fast algorithm. An experimental comparison with another algorithm has shown significant time improvement for large-scale problems.

Finally, note that in the available source code of *SVMTorch*, we also implemented the same algorithm for classification problems. The result is thus similar mathematically to the one proposed by Joachims in its *SVM-Light* software [4], but the speed obtained was twice as fast as *SVM-Light* (there are probably many implementation differences, such as the cache, which was not fully described in Joachims's paper).

## References

[1] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information*

*Processing Systems 9*, pages 155–161. MIT Press, 1997.

[2] G.W. Flake and S. Lawrence. Efficient SVM regression training with SMO. Submitted to Machine Learning. Available at `http://external.nj.nec.com/homepages/flake/smorch.ps`.

[3] R. Fletcher. *Practical Methods of Optimization*. John Wiley and sons, 1987.

[4] Thorsten Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. MIT Press, 1999.

[5] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks - ICANN'97*, pages 999–1004. Springer, 1997.

[6] Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, New York, 1997.

[7] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. MIT Press, 1999.

[8] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer, second edition, 1995.