Journal of Machine Learning Research 1 (2001) 143-160 Submitted 7/00; Published 2/01

SVMTorch: Support Vector Machines for Large-Scale Regression Problems

Ronan Collobert COLLOBER@IDIAP.CH
IDIAP
CP 592, rue du Simplon 4
1920 Martigny, Switzerland
tel: +41 27 721 77 31
fax: +41 27 721 77 12

Samy Bengio BENGIO@IDIAP.CH
IDIAP
CP 592, rue du Simplon 4
1920 Martigny, Switzerland
tel: +41 27 721 77 39
fax: +41 27 721 77 12

Editor: Robert C. Williamson

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 (1999) 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 publicly available SVM algorithm for large-scale regression problems from Flake and Lawrence (2000) yielded significant time improvements. Finally, based on a recent paper from Lin (2000), we show that a convergence proof exists for our algorithm.

1. Introduction

Vapnik (1995) has proposed a method to solve regression problems using support vector machines. It has yielded excellent performance on many regression and time series prediction problems (see for instance Müller et al., 1997, or Drucker et al., 1997). 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:

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

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

(with $b \in \mathbb{R}$) with a precision ϵ. 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 ϵ-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 objective 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 $\mathbf{1}$ is the unit vector and 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 (1999) in the context of classification problems. In the next section, we give the general algorithm and explain in more detail each of its main steps and how they differ from other published algorithms, as well as a discussion on convergence and on some important implementation issues, such as a way to efficiently handle the kernel matrix computation. In the experiment section, we compare this new algorithm on small and large datasets to *Nodelib*, another SVM algorithm for large-scale regression problems proposed by Flake and Lawrence (2000), 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, and finally how our algorithm scales with respect to l.

2. The Decomposition Algorithm

As in the classification algorithm proposed by Joachims (1999), which was based on a idea from Osuna et al. (1997), our regression algorithm is subdivided into the following four steps, which are explained afterward in the following subsections:

1. Select q variables α_i *or* $\alpha_i^\star$ 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 whose values have been at 0 or C for a long time and that will probably not change anymore. This optional step is the *shrinking* phase, as these variables are removed from the problem.
4. Test whether the optimization is finished; if not, return to the first step.

Many other decomposition algorithms for regression have been published recently, and a comparison is given later in section 2.6.

2.1 Selection of a New Working Set

We propose to select a new set of q variables such that the overall criterion will be optimized. In order to select such a working set, we use the same idea as Joachims (1999): 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 are chosen to be the 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

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

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

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) and (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:

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

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

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

where $\delta_i = 1$ for $1 \leq i \leq l$ and $\delta_i = -1$ for $l+1 \leq i \leq 2l$. To simplify, let q be even, and let us sort the ω_i in decreasing order. Let us then denote φ as the bijection of $\{1 \ldots 2l\}$ into itself such that the $\omega_{\varphi(i)}$ are sorted. Let us then select the $q/2$ first indices $\varphi(i)$ such that:

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

$$\text{if } \varphi(i) > l, \text{ we have } 0 \leq \alpha^\star_{\varphi(i)-l} < C \;\;;$$

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

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

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

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.

Lemma 1 *Let us now denote $c_i,\; i = 1 \ldots q$, the q indices we just chose, then the direction ρ such that*

$$\rho_j = \begin{cases} -\delta_j \text{ if } j \in \{c_1 \ldots c_{\frac{q}{2}}\} \\ \delta_j \text{ if } j \in \{c_{\frac{q}{2}+1} \ldots c_q\} \\ 0 \text{ otherwise} \end{cases}$$

is a solution of the minimization problem (6).

Proof: Let us go back to the minimization problem of the function

$$(z_1,\, \dots\, z_{2l}) \longmapsto \sum_{i=1\dots 2l} \omega_i\, z_i \tag{11}$$

subject to

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

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

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 $2l = q = 2r$, 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 $\gamma \geq 0$ for a $i_0 \leq r$, then one needs to compensate by $-\gamma$ another z_j to maintain (12). Since we want to minimize (11), the best thing to do, knowing that the ω_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 ω_i are sorted and thus we get out of the minimum.

In the case where $2l > q = 2r$, 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-null. 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 of our problem, then we necessarily have $z_{k_i} = \pm 1$. Considering again the order of the ω_i, it becomes evident that we have to take $(k_1 = 1) \dots (k_r = r)$ and $(k_{r+1} = 2l - r) \dots (k_q = 2l)$.
$\square$

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

2.2 Solving the Subproblem

We want to solve the problem (3) taking into account variables $\mathcal{S}$ only. 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

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

The problem (3) is thus equivalent to minimizing

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

subject to

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

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 part 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

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

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

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

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^{th} variable in the set $\mathcal{S}$ corresponds to an α_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 (Fletcher, 1987). Moreover, following Platt's idea in *SMO* (Platt, 1999), if one fixes the size of the working set $\mathcal{S}$ to 2, the problem can also be solved analytically.

This particular case is important because experimental results show that it always gives the fastest convergence times. We explain it here *because it is a different minimization problem* from the one proposed by previous authors such as Smola and Schölkopf (1998); in fact, it is easier because it only has 2 variables and not 4.

Let us again simplify the notation:

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

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

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

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

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

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

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

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

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

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

and

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

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 it is always the case.

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 $\eta \le 0$, it is clear that the solution

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

and

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

is the minimum.

2.3 Shrinking

The idea of shrinking is to remove some variables whose values have been equal to the bounds 0 or C for a long time, and that will *probably* 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*:

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

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

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

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

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

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

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

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 λ^{eq} that verify *as well as possible*[3] the equations (24)-(28), and to remove a variable whose value is equal to 0 if its coefficient λ_i^{low} is strictly positive (or just above a constant ϵ_{shrink}) during a given number of iterations. Using the same idea, we also eliminate a variable whose value is equal to C if its coefficient λ_i^{up} stays strictly positive for a sufficient number of iterations.

To estimate $\boldsymbol{\lambda}^{low}$ or $\boldsymbol{\lambda}^{up}$, we start by estimating λ^{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 λ) :

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

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

Then if we have

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

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. Note that this can lead to an incorrect solution if ϵ_{shrink} or the number of iterations before removing a variable is too small. This is verified in the experimental section.

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 subproblem, a *reasonable* termination criterion is to verify that the λ estimated by (31) and (32) verifies the conditions (24)-(28) with a given precision ϵ_{end}.

Thus, we simply verify that

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

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

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

2.5 Implementation Details

Note that in the algorithm described here, only two steps might be time consuming: the one that computes $\mathcal{W}_i^{'}$ and the one that computes $\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{SF}} \boldsymbol{\beta}_{\mathcal{F}}$ in (18).

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

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

where

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

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

$$\begin{array}{rcl} \left(\boldsymbol{b}_{\mathcal{S}} - \tilde{K}_{\mathcal{SF}} \boldsymbol{\beta}_{\mathcal{F}}\right)_i & = & (\boldsymbol{b}_{\mathcal{S}})_i - \left(\tilde{K}_{\mathcal{SF}} \boldsymbol{\beta}_{\mathcal{F}} + \tilde{K}_{\mathcal{SS}} \boldsymbol{\beta}_{\mathcal{S}}\right)_i + \left(\tilde{K}_{\mathcal{SS}} \boldsymbol{\beta}_{\mathcal{S}}\right)_i \\ & = & \left\{ \begin{array}{ll} -\mathcal{W}_i + \left(\tilde{K}_{\mathcal{SS}} \boldsymbol{\beta}_{\mathcal{S}}\right)_i & if \;\; i \leq l \\ \mathcal{W}_i + \left(\tilde{K}_{\mathcal{SS}} \boldsymbol{\beta}_{\mathcal{S}}\right)_i & if \;\; i > l. \end{array} \right. \end{array}$$

With these two ideas, one can reduce considerably the computational time: instead of computing all the lines of the matrix K, one can compute only 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 the whole 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.

2.6 Comparisons with Other Algorithms

Recently, many authors have proposed decomposition algorithms for regression. For instance, Shevade et al. (2000) proposed two modifications of the *SMO* algorithm from Platt (1999) for regression, based on a previous paper from the same team (Keerthi et al., 1999) for classification problems. Laskov (2000) proposed also a decomposition method for regression problems which is very similar to the second modification proposed by Shevade et al. In fact, it is easy to see that Laskov's method with a subproblem of size 2 uses the same selection algorithm as well as the same termination criterion as Shevade et al.

Their method for selecting the working set is *very* similar to the one we show in this paper, but while we propose to select variables α_i independently of their counterparts $\alpha_i^\star$, they propose to select simultaneously *pairs* of variables $\{\alpha_i,\ \alpha_i^\star\}$. Even if this seems to be a small difference, let us note that $\alpha_i\,\alpha_i^\star = 0\ \forall i$ at the optimal solution as well as during the optimization process, as proved by Lin (2000) in the case of algorithms such as the one we propose here.[4] Thus, one of the two variables α_i or $\alpha_i^\star$ is always equal to 0, and choosing the α_i and $\alpha_i^\star$ independently can thus help to quickly eliminate many variables, thanks to the *shrinking* phase,[5] which, of course, has a direct impact on the speed of our algorithm. In fact, working with pairs of variables would force the algorithm to do many computations with null variables until the end of the optimization process.

Similarly, Smola and Schölkopf (1998) also proposed earlier to use a decomposition algorithm for regression based on *SMO*, using an analytical solution for the subproblems, but again they proposed to select two pairs of variables (two α and their corresponding $\alpha^\star$) instead of two variables as we propose in this paper.

Finally, Flake and Lawrence (2000) proposed a modification of *SMO* for regression that uses the heuristics proposed by Platt (1999) and those from Smola and Schölkopf (1998), but their modification uses a new variable $\lambda_i = \alpha_i - \alpha_i^\star$. Once again, this forces the use of the pairs $\{\alpha_i,\ \alpha_i^\star\}$ during the computations.

The originality of our algorithm is thus to select *independently* the variables α_i and $\alpha_i^\star$, which has the side effect of efficiently adapting the shrinking step proposed in classification by Joachims (1999) (it is indeed less easy to think of an efficient shrinking method in the context of pairs of variables). This also helps to simplify the resolution of the subproblem in the case where $q = 2$. Finally, experiments given in section 3 suggest that this idea leads to faster convergence times.

4. Note also that testing whether the α_i and $\alpha_i^\star$ are non-zero at the same time would be a waste of time.
5. This is verified in practice.

2.7 Convergence

In a recent technical report (Collobert & Bengio, 2000), we have shown that our algorithm converges in the case where the working set size is equal to 2 and without shrinking, for any kernel that verifies Mercer's conditions. To do so, we have used a theorem proved by Keerthi and Gilbert (2000). Note however that more recently, Lin (2000) has shown the convergence of our algorithm for any value of the working set size (but again without shrinking), under the following hypothesis:

Assumption 1 *The matrix K satisfies*

$$\min_I(\min(eig(K_{II}))) > 0$$

where I is any subset of $\{1, \ldots, l\}$ with $|I| \leq q$, K_{II} is a square sub-matrix of K, and $\min(eig(.))$ *is the smallest eigenvalue of a matrix.*

Note finally that shrinking is a heuristic and thus using it should speed up the algorithm but no convergence proof will hold anymore.

3. Experimental Results

We compared our SVM implementation for regression problems (*SVMTorch*) to the one from Flake and Lawrence (2000) using their publicly available software *Nodelib*. This is interesting because *Nodelib* is based on *SMO* where the variables α_i and $\alpha_i^\star$ are selected simultaneously, which is not the case for *SVMTorch*. Note also that *Nodelib* includes some enhancements compared to *SMO* which are different from those proposed by Shevade et al. (2000).

Both these algorithms use an internal cache in order to be able to solve large-scale problems. All the experiments presented here have been done on a LINUX Pentium III 750Mhz, with the *gcc* compiler. The parameters of the algorithms were not chosen to obtain the best generalization performances, since the goal was to *compare the speed* of the algorithms. However, we have chosen them in order to obtain reasonable results. 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.[6] All the programs were compiled using *double* precision. We compared the programs on five different tasks :

Kin This dataset[7] represents a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm. The task is to predict the distance of the end-effector from a target, given features like joint positions, twist angles, etc.

Sunspots Using a series representing the number of sunspots per day, we created one input/output pair for each day: the yearly average of the year starting the next day had to be predicted using the 12 previous yearly averages.

6. For 5000 examples, we used `-clever -best -lazy`, while for more than 5000 examples, we used `-clever -best -lazy -ssz 200`.
7. Available on `http://www.cs.toronto.edu/~delve/data/kin/desc.html`.

Artificial This is an easy artificial dataset based on **Sunspots**: we create a daily series where each value is the yearly average centered on that day. The task is to predict a value given the 100 previous ones.

Forest This dataset[8] is a classification task with 7 classes, where only the first 35000 examples were used here. We transformed it into a regression task where the goal was to predict +1 for examples of class 2 and −1 for the other examples. Note that since class 2 was over-represented in the dataset, this transformation leads to a more balanced problem.

MNIST This dataset[9] is a classification task with 10 classes (representing handwritten digits). Again, we transformed it in a regression problem where the goal was to predict +1 for examples of classes 0 to 4 and −1 for the other examples.

Note also that *Forest* and *MNIST* are respectively 78% and 81% sparse (contain respectively 78% and 81% null values in their input matrices). Since *SVMTorch* can handle sparse data (as can *SVM-Light*), we tested this option in the experiments described in Tables 3 and 4. The parameters used to train the datasets can be found in Table 1. Note that all experiments used a Gaussian kernel[10] and a value of $C = 1000$, and the termination criterion was the verified KKT conditions with a precision of 0.01.

	# Train	# Test	Dim	σ	ϵ
Kin	6192	2000	32	100	0.5
Artificial	20000	2000	100	100	0.5
Forest	25000	10000	54	400	0.7
Sunspots	40000	2500	12	900	20
MNIST	60000	10000	784	1650	0.5

Table 1: Parameters used for each dataset: # Train = maximum number of training examples, taken at the begining of the dataset, # Test = number of test examples, taken just following the training set examples, Dim = input dimension, σ = parameter of the Gaussian kernel, ϵ = value used in the ϵ-insensitive loss function.

For the experiments involving *SVMTorch*, we have tested a version with shrinking but without verifying at the end of the optimization whether all the suppressed variables verified the KKT conditions (*SVMTorch*), with no shrinking (*SVMTorch*N), and a version with shrinking and verification at the end of the optimization, as done in *SVM-Light* (*SVMTorch*U). As it will be seen in the results, the first method has a big speedup advantage, but only a small negative impact on the generalization performance *in general*. However, sometimes the default value of 100 iterations before a variable is removed by shrinking must be changed to obtain the correct solution.

8. Available on `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/covtype.info`.
9. Available on `http://www.research.att.com/~yann/ocr/mnist/index.html`.
10. The kernel is $k(x, y) = \exp(-\|x - y\|^2/\sigma^2)$.

3.1 Working Set Size

Using the first 10000 examples of each dataset (or 6192 for *Kin* which is too small), we trained different models using various values of q, from 2 to 100. We used a fixed cache size of 100Mb and turned on the shrinking, but did not use the sparse mode. The optimizer used to solve the subproblems of size $q > 2$ was a conjugate gradient method with projection[11]. Table 2 gives the results of these experiments. It is clear that $q = 2$ is always faster than any other value of q. Thus, in the following experiments, we have always used $q = 2$.

	Working set size				
	2	4	10	50	100
Kin	11	14	16	28	54
Artificial	98	149	190	629	1537
Forest	272	406	462	670	981
Sunspots	7	11	15	45	89
MNIST	573	664	829	1657	2213

Table 2: Training time (in seconds) as a function of the working set size, for non-sparse data.

3.2 Small Datasets

Let us now compare *SVMTorch* and *Nodelib* on small datasets. In the results given in Table 3, only the first 5000 training examples were used. The size of the cache was set to 300Mb, so that the whole kernel matrix could be kept in memory. For each problem, we also computed the output median (which minimizes the mean absolute error (MAE) without any information about the inputs) on the training set and computed the performance of this median over the training and the test sets in order to have a better idea of the performance of the various algorithms.

As can be seen, for all the datasets, *SVMTorch* is usually many times faster than *Nodelib* (except for *Artificial* in the case of *SVMTorch* without shrinking). Since the whole matrix of the quadratic problem was in memory, handling of the cache had no effect on the speed results. Thus one can conclude that one of the main differences between these algorithms is the selection of the subproblem, and that selecting the α_i independently of the $\alpha_i^\star$ is very efficient. However, *Nodelib* gave slightly better results in terms of the objective function (probably due to the fact that their termination criterion is stronger than ours; see (Collobert & Bengio, 2000) for a comparison between termination criteria), but not in terms of test error. Note that for problems of this size, shrinking does not cause the performance to deteriorate too much (check the values of the objective function as well as the training and test set performances) but does speed the algorithm up a lot. Note also that using the sparse format was not good in the case of *Forest* but was good for *MNIST*, which has a higher input dimension: this means that the cost induced by the use of sparse

11. Actually, the original source code of this optimizer has been done by Leon Bottou.

Dataset	Model	Time NSP	Time SP	# SV	Objective Function	Model Train	Model Test	Median Train	Median Test
	SVMTorch	7	–	936	-173977.85	0.30	0.31		
Kin	*SVMTorch*U	15	–	936	-173977.85	0.30	0.31	0.37	0.38
	*SVMTorch*N	45	–	941	-173982.65	0.30	0.31		
	Nodelib	157	–	932	-174019.67	0.30	0.31		
	SVMTorch	31	–	342	-13594.01	0.25	0.52		
Artificial	*SVMTorch*U	166	–	367	-13703.07	0.24	0.51	25.16	27.95
	*SVMTorch*N	448	–	370	-13701.41	0.24	0.51		
	Nodelib	231	–	342	-13707.29	0.24	0.51		
	SVMTorch	21	21	993	-1012.46	0.51	0.80		
Forest	*SVMTorch*U	65	43	1051	-1030.78	0.41	0.80	0.38	1.59
	*SVMTorch*N	110	115	1058	-1030.43	0.41	0.80		
	Nodelib	542	–	1032	-1031.54	0.41	0.80		
	SVMTorch	2	–	420	-3489571.13	9.65	10.12		
Sunspots	*SVMTorch*U	9	–	422	-3489630.53	9.65	10.11	33.02	52.58
	*SVMTorch*N	38	–	422	-3489628.27	9.64	10.11		
	Nodelib	327	–	422	-3489630.65	9.64	10.11		
	SVMTorch	118	79	1861	-190.77	0.39	0.48		
MNIST	*SVMTorch*U	147	98	1861	-190.77	0.39	0.48	0.98	0.97
	*SVMTorch*N	152	104	1861	-190.77	0.39	0.48		
	Nodelib	2216	–	1878	-190.80	0.39	0.48		

Table 3: Experiments on small training sets. *SVMTorch*N = *SVMTorch* without shrinking, *SVMTorch*U = *SVMTorch* with shrinking and unshrinking, Time NSP = time (in seconds) for non-sparse data format, Time SP = time (in seconds) for sparse data format, # SV = number of support vectors, Objective Function = value of (3) at the end of the optimization, Model Train = mean absolute error (MAE) over the training set, Model Test = MAE over the test set, Median Train = MAE over the training set with the median as predictor, Median Test = MAE over the test set with the median as predictor.

format is balanced by the gain obtained in the kernel computation only when the input size is large enough.

3.3 Large Datasets

Let us now turn to experiments using large datasets. Table 4 shows the results using the whole training sets for all datasets, again using a cache size of 300Mb. Since the problems are now too big to be kept in memory, the implementation of the cache becomes very important and comparisons of the algorithms used in *SVMTorch* and *Nodelib* become more difficult. Nevertheless, it is clear that *SVMTorch* is always faster, except again for *Artificial* in the cases with no shrinking or with unshrinking, but the performance on the test sets is similar. However, note that shrinking sometimes leads to very poor results in terms of test set performance, as is the case on *Forest*. It is thus clear that shrinking should be used with care, *particularly for large datasets*, and the parameter that decides when to eliminate

Dataset	Model	Time NSP	Time SP	# SV	Objective Function	Model Train	Model Test	Median Train	Median Test
Kin	*SVMTorch*	11	–	1140	-212439.78	0.30	0.31	0.37	0.38
	*SVMTorch*U	32	–	1140	-212439.78	0.30	0.31		
	*SVMTorch*N	86	–	1140	-212439.78	0.30	0.31		
	Nodelib	273	–	1138	-212478.38	0.30	0.31		
Artificial	*SVMTorch*	235	–	706	-39569.14	0.21	0.34	27.29	14.25
	*SVMTorch*U	4394	–	817	-40025.98	0.20	0.33		
	*SVMTorch*N	9182	–	824	-40016.55	0.20	0.34		
	Nodelib	2653	–	764	-40043.94	0.20	0.33		
Forest	*SVMTorch*	4573	4392	3019	-56266.94	1.63	1.82	0.81	1.59
	*SVMTorch*U	40669	37769	4080	-78297.27	0.40	0.93		
	*SVMTorch*N	79237	73045	4233	-78294.56	0.39	0.93		
	Nodelib	87133	–	4088	-78384.15	0.39	0.93		
Sunspots	*SVMTorch*	67	–	1771	-11215476.03	8.97	12.72	33.02	52.57
	*SVMTorch*U	1290	–	1822	-11229107.83	8.96	12.59		
	*SVMTorch*N	2606	–	1820	-11229098.49	8.96	12.59		
	Nodelib	24022	–	1818	-11229124.45	8.96	12.59		
MNIST	*SVMTorch*	9874	6460	8532	-1289.54	0.25	0.27	0.98	0.97
	*SVMTorch*U	33644	21482	8642	-1290.66	0.25	0.27		
	*SVMTorch*N	32095	20951	8634	-1290.57	0.25	0.27		
	Nodelib	$> 10^6$	–	–	–	–	–		

Table 4: Experiments on large training sets. See Table 3 for the description of the fields.

a variable should be tuned carefully before running a series of experiments on the same dataset. Note also that *Nodelib* was not able to solve *MNIST* after 11 days.

3.4 Size of the Cache

We also did some experiments to measure the effect of the size of the cache on the training time. Table 5 shows the results for different cache sizes, from 10 to 100Mb. In these experiments, we used the first 10000 examples of each dataset (6192 for the smaller *Kin*) and used the non-sparse format. The only clear conclusion from these experiments is that the higher the size of the cache, the faster *SVMTorch* is, but the relation is completely problem dependent.

	Size of the cache (in Mb)									
	10	20	30	40	50	60	70	80	90	100
Kin	11	11	11	11	11	11	11	11	11	11
Artificial	359	182	106	100	99	98	98	98	98	98
Forest	869	715	622	519	450	391	355	316	305	272
Sunspots	7	7	7	7	7	7	7	7	7	7
MNIST	729	724	716	704	686	665	647	621	597	573

Table 5: Training time (in seconds) with respect to the size of the cache (in Mb).

3.5 Scaling with Respect to the Size of the Training Set

Finally, we tried to evaluate how *SVMTorch* (with shrinking and in non-sparse format) and *Nodelib* scaled with respect to the size of the training set. In order not to be influenced by the implementation of the cache system, we computed the training time for training sets of sizes 500, 1000, 2000, 3000, 4000, and 5000, so that the whole matrix of the quadratic problem could be kept in memory. Given the results, we did a linear regression of the log of the time given the log of the training size and Table 6 gives the slope of this linear regression for each problem, which gives an idea of how *SVMTorch* scales: it appears to be slightly better than quadratic, and slightly better than *Nodelib*.

	Kin	Artificial	Forest	Sunspots	MNIST
Scale *SVMTorch*	1.81	1.72	1.82	1.85	1.64
Scale *Nodelib*	1.83	1.93	2.09	2.44	1.75

Table 6: Scaling of *SVMTorch* and *Nodelib* for each dataset. Results give the slope of the linear regression in the log-log domain of time versus training size.

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 those used by Joachims (1999) in his classification algorithm. Compared to previously proposed decomposition algorithms for regression, we have proposed an original method to select the variables in the working set. We have shown how to solve analytically subproblems of size 2, as it is done in *SMO* (Platt, 1999). 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. We have also shown that there exists a convergence proof for our algorithm. Finally, an experimental comparison with another algorithm has shown significant time improvement for large-scale problems and training time generally scaling slightly less than quadratically with respect to the number of examples.

References

Collobert, R., & Bengio, S. (2000). *On the Convergence of SVMTorch, an Algorithm for Large-Scale Regression Problems* (IDIAP-RR No. 24). IDIAP. (Available at `ftp://www.idiap.ch/pub/reports/2000/rr00-24.ps.gz`)

Drucker, H., Burges, C., Kaufman, L., Smola, A., & Vapnik, V. (1997). Support vector regression machines. In M. Mozer, M. Jordan, & T. Petsche (Eds.), *Advances in Neural Information Processing Systems 9* (pp. 155–161). The MIT Press.

Flake, G., & Lawrence, S. (2000). *Efficient SVM regression training with SMO.* (Submitted to Machine Learning. Available at http://external.nj.nec.com/homepages/flake/smorch.ps)

Fletcher, R. (1987). *Practical Methods of Optimization.* Chichester: John Wiley and Sons.

Joachims, T. (1999). Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, & A. Smola (Eds.), *Advances in Kernel Methods.* The MIT Press.

Keerthi, S. S., & Gilbert, E. G. (2000). *Convergence of a Generalized SMO Algorithm for SVM Classifier Design* (Tech. Rep. No. CD-00-01). Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore. (Available at http://guppy.mpe.nus.edu.sg/~mpessk/svm/conv_ml.ps.gz)

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (1999). *Improvements to Platt's SMO Algorithm for SVM Classifier Design* (Tech. Rep. No. CD-99-14). Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore. (To appear in Neural Computation. Available at http://guppy.mpe.nus.edu.sg/~mpessk/smo_mod.ps.gz)

Laskov, P. (2000). An improved decomposition algorithm for regression support vector machines. In S. Solla, T. Leen, & K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems 12.* The MIT Press.

Lin, C. (2000). *On the Convergence of the Decomposition Method for Support Vector Machines* (Tech. Rep.). National Taiwan University. (Available at http://www.csie.ntu.edu.tw/~cjlin/papers/conv.ps.gz)

Müller, K.-R., Smola, A., Rätsch, G., Schölkopf, B., Kohlmorgen, J., & Vapnik, V. (1997). Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, & J.-D. Nicoud (Eds.), *Artificial Neural Networks - ICANN'97* (pp. 999–1004). Springer.

Osuna, E., Freund, R., & Girosi, F. (1997). An improved training algorithm for support vector machines. In J. Principe, L. Giles, N. Morgan, & E. Wilson (Eds.), *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop* (pp. 276–285). New York: IEEE Press.

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, & A. Smola (Eds.), *Advances in Kernel Methods.* The MIT Press.

Shevade, S. K., Keerthi, S. S., Bhattacharyya, C., & Murthy, K. R. K. (2000). Improvements to the SMO algorithm for SVM regression. *IEEE Transaction on Neural Networks*, *11*(5), 1188–1183.

Smola, A., & Schölkopf, B. (1998). *A Tutorial on Support Vector Regression* (Tech. Rep. No. NeuroCOLT NC-TR-98-030). Royal Holloway College,University of London, UK.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory* (second ed.). Springer.