On-line Support Vector Machines for Function Approximation

Mario Martin
email: mmartin@lsi.upc.es
Software Department, Universitat Politècnica de Catalunya,
Jordi Girona 1-3, Campus Nord, C6.
08034 Barcelona, Catalonia, Spain.

Abstract

This paper describes an *on-line* method for building ε-insensitive support vector machines for regression as described in (Vapnik, 1995). The method is an extension of the method developed by (Cauwenberghs & Poggio, 2000) for building incremental support vector machines for classification. Machines obtained by using this approach are equivalent to the ones obtained by applying exact methods like quadratic programming, but they are obtained more quickly and allow the incremental addition of new points, removal of existing points and update of target values for existing data. This development opens the application of SVM regression to areas such as on-line prediction of temporal series or generalization of value functions in reinforcement learning.

1 Introduction

Support Vector Machines, from now on SVM, (Vapnik, 1995) have been one of the most developed topics in Machine Learning in the last decade. Some reasons that explain this success are their good theoretical properties in generalization and convergence – see(Cristianini & Shawe-Taylor, 2000) for a review. Another reason is their excellent performance in some hard problems –see for instance (Osuna et al., 1997; Dumais et al., 1998).

Although SVMs are being used mainly for classification tasks, they can also be used to approximate functions (what is called SVM regression). One problem that prevents a wider use of SVMs for function approximation is that, though their good theoretical approaches, they are not applicable on-line, that is, in cases where data is sequentially obtained and learning has to be done from the first data.

One paradigmatic example is the on-line prediction of temporal series. When new data arrive, learning has to begin from scratch with the whole data set (or data has to be incorporated using "ad hoc" techniques that return approximate but not exact results).

SVMs for regression have not been either suitable for problems where the target values of existing observations change quickly, for instance, in reinforcement learning (Sutton & Barto., 1998). In reinforcement learning, function approximation is needed to learn *value functions*, that is, functions that return for each state the future expected reward if the agent follows the current policy from that state. SVMs are not used to approximate value functions because these functions are continuously update as the agent learns and changes its policy. One time, the estimated future reinforcement from state s is y, but later (usually very soon) a new estimation returns another value for the same state. Using SVM regression in this case implies again learning from scratch.

In order to allow the application of SVMs for regression to these areas, this paper describes the *first* (at our best knowledge) *exact on-line learning algorithm for SVM function approximation.* The algorithm is based in three actions that allow respectively (1) incrementally add new data to the SVM, (2) remove data from the SVM, and (3) update target values for existing data in the SVM.

The algorithm we propose is an extension of the work proposed in (Cauwenberghs & Poggio, 2000) for incremental SVM learning for classification tasks, but now applied to function approximation. In brief, the key idea of the algorithm consists in finding the appropriate *Kuhn-Tucker* (KT) conditions for new or updated data by modifying its influence (β) in the regression function while maintaining consistence in the KT conditions for the rest of the data used for learning. This idea is fully explained throughout the paper.

2 Reformulation

Specifically, we propose in this paper a method for the on-line building of ε-insensitive support vector machines for regression. The goal of this kind of machines is to find a function that presents at most ε deviation from the target values (Vapnik, 1995) while being as "flat" as possible. This version of SVM regression is appealing because not all vectors become support vectors, which is not the case in other approaches (Smola & Schölkopf, 1998).

SVMs for regression are usually solved by resorting to a standard dualization method using Lagrange multipliers. The dual formulation for ε-insensitive support vector regression is to find values for α, α^* that minimize the following quadratic objective function:

$$\begin{aligned} W = \frac{1}{2}\sum_{ij}(\alpha_i - \alpha_i^*)Q_{ij}(\alpha_j - \alpha_j^*) - \\ \sum_i y_i(\alpha_i - \alpha_i^*) + \varepsilon \sum_i (\alpha_i + \alpha_i^*) \end{aligned} \tag{1}$$

subject to the following constraints:

$$0 \le \alpha_i, \alpha_i^* \le C \tag{2}$$

$$\sum_i (\alpha_i - \alpha_i^*) = 0 \tag{3}$$

where Q is the positive-definite kernel matrix $Q_{ij} = K(x_i, x_j)$, and $\varepsilon > 0$ is the maximum deviation allowed.

Including in (1) a Lagrange multiplier for constraint (3), we get the following formulation:

$$\begin{aligned} W = \frac{1}{2}\sum_{ij}(\alpha_i - \alpha_i^*)Q_{ij}(\alpha_j - \alpha_j^*) - \\ \sum_i y_i(\alpha_i - \alpha_i^*) + \varepsilon \sum_i (\alpha_i + \alpha_i^*) + b\sum_i(\alpha_i - \alpha_i^*) \end{aligned} \tag{4}$$

with first order conditions for W:

$$g_i = \frac{\partial W}{\partial \alpha_i} = \sum_j Q_{ij}(\alpha_j - \alpha_j^*) - y_i + \varepsilon + b \tag{5}$$

$$g_i^* = \frac{\partial W}{\partial \alpha_i^*} = -\sum_j Q_{ij}(\alpha_j - \alpha_j^*) + y_i + \varepsilon - b = \\ -g_i + 2\varepsilon \tag{6}$$

$$\frac{\partial W}{\partial b} = \sum_j (\alpha_j - \alpha_j^*) = 0 \tag{7}$$

Renaming $(\alpha_i - \alpha_i^*)$ to β_i for simplicity, we have:

$$g_i = \frac{\partial W}{\partial \alpha_i} = \sum_j Q_{ij}\beta_j - y_i + \varepsilon + b \tag{8}$$

$$g_i^* = \frac{\partial W}{\partial \alpha_i^*} = -\sum_j Q_{ij}\beta_j + y_i + \varepsilon - b = -g_i + 2\varepsilon \tag{9}$$

$$\frac{\partial W}{\partial b} = \sum_j \beta_j = 0 \tag{10}$$

2.1 Separation of Data

The first order conditions for W lead to the *Kuhn-Tucker* (KT) conditions, that will allow the reformulation of SVM for regression by dividing the whole training data set D into the following sets: *margin support vectors* S (where $g_i = 0$ or $g_i^* = 0$), *error support vectors* E (where $g_i < 0$), *error star support vectors* E^* (where $g_i^* < 0$), and the *remaining vectors* R. Specifically, centering on g_i, KT conditions are:

$$\left\{ \begin{array}{ccccc} 2\varepsilon < g_i & \rightarrow & g_i^* < 0 & \beta_i = -C & i \in E^* \\ g_i = 2\varepsilon & \rightarrow & g_i^* = 0 & -C < \beta_i < 0 & i \in S \\ 0 < g_i < 2\varepsilon & \rightarrow & 0 < g_i^* < 2\varepsilon & \beta_i = 0 & i \in R \\ g_i = 0 & \rightarrow & g_i^* = 2\varepsilon & 0 < \beta_i < C & i \in S \\ g_i < 0 & \rightarrow & g_i^* > 2\varepsilon & \beta_i = C & i \in E \end{array} \right.$$

Figure 1 shows the geometrical interpretation of these sets in the feature space. Note that $\sum_j Q_{ij}\beta_j + b - y_i$ is the error of the target value for vector i. Thus g_i and g_i^* can be thought as thresholds for error in both sides of the ε-tube.

The division of the data set into subsets and the characterization of β values for each subset, allow us to rewrite equations (8), (9) and (10), for all vectors $i \in D$, as follows:

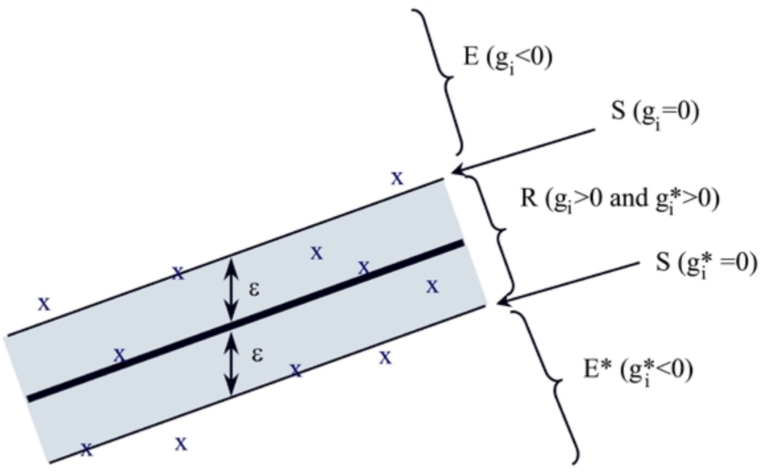

Figure 1: Decomposition of D following KT conditions into *margin support vectors* S, *error support vectors* E, *error support vectors* star E^* and *remaining vectors* R. Cross marks represent vectors in the feature space. S vectors are exactly on the margin lines, R vectors are inside the ε-tube (grey zone), and E and E^* vectors are outside the ε-tube.

$$g_i = \sum_{j\in S} Q_{ij}\beta_j + C\sum_{j\in E} Q_{ij} - C\sum_{j\in E^*} Q_{ij} - y_i + \varepsilon + b \tag{11}$$

$$g_i^* = -g_i + 2\varepsilon \tag{12}$$

$$\sum_{j\in S}\beta_j + C|E| - C|E^*| = 0 \tag{13}$$

3 On-line support vector regression

In order to build exact on-line support vector machines for regression, we need to define three incremental actions:

add one new vector: One new observation x_c is added to the data set D with the target value y_c. This operation should include the corresponding vector in the feature space with the "exact" β_c value but without beginning from scratch.

remove one vector: One existing observation x_c in D with target value y_c is removed from the data set. The resulting SVM should be the same that would be training from scratch a SVM with $D - \{c\}$.

update one vector: One existing observation x_c in D with target value y_c changes the target value to y_c'. As in the previous cases the resulting machine should be the same that would be training from scratch a SVM with exact methods.

In this section we will describe how these actions can be efficiently implemented. Addition and update actions will consist in finding a consistent KT condition for the vector being added or updated. Removal will be based on diminishing the influence of the vector being removed on the regression tube until it vanishes.

3.1 Adding one new vector

A new vector c is added by inspecting g_c and g_c^*. If both values are positive, c is added as an R vector because that means that the new vector lies inside the ε-tube (see KT conditions). When g_c or g_c^* are negative, the new vector is added by setting its initial influence on the regression (β_c) to 0. *Then this value is carefully modified* (incremented when $g_c < 0$ or decremented when $g_c^* < 0$) *until its g_c, g_c^* and β_c values become consistent with a KT condition* (that is, $g_c < 0$ and $\beta_c = C$, or $g_c^* < 0$ and $\beta_c = -C$, or $0 < \beta_c < C$ and $g_c = 0$, or $-C < \beta_c < 0$ and $g_c^* = 0$).

3.1.1 Modification of β_c

Variations in the β_c value of the new vector c, influence g_i, g_i^* and β_i values of the other vectors in D, and thus, can force the transfer of some vectors from one set S, R, E or E^* to another set. This transfer means that g_i, g_i^* and β_i values for vector i become no longer consistent with the KT conditions of the set where vector i is currently assigned, but become consistent with the KT conditions of another set.

The modification of β_c must take into account these transfers between sets. This section describes how the modification of β_c influences g_i, g_i^* and β_i values of the vectors in D *while sets S, E, E^* and R remain constant.* In the next section we describe how to deal with vector migrations between sets.

From equations (11), (12), and (13) it is easy to calculate the variation in g_i, g_i^* and β_i when a new vector c with influence β_c is added without migration of vectors between sets S, E, E^* and R:

$$\Delta g_i = Q_{ic}\Delta\beta_c + \sum_{j\in S} Q_{ij}\Delta\beta_j + \Delta b \tag{14}$$

$$\Delta g_i^* = -\Delta g_i \tag{15}$$

$$\Delta\beta_c + \sum_{j\in S}\Delta\beta_j = 0 \tag{16}$$

Note that while one vector remains in E, E^* or R sets, its β value does not change.

In particular, if margin support vectors must remain in S, then $\Delta g_i \equiv 0$ for $i \in S$. Thus, if we isolate $\Delta\beta_c$ terms in equations (14) and (16) for vectors $i \in S$, we get:

$$\sum_{j\in S} Q_{ij}\Delta\beta_j + \Delta b = -Q_{ic}\Delta\beta_c \tag{17}$$

$$\sum_{j\in S}\Delta\beta_j = -\Delta\beta_c \tag{18}$$

That, assuming $S = \{S_1, S_2, \cdots, S_l\}$, can be matricialy formulated as follows:

$$\mathcal{Q}\cdot\begin{bmatrix}\Delta b\\ \Delta\beta_{S_1}\\ \vdots\\ \Delta\beta_{S_l}\end{bmatrix} = -\begin{bmatrix}1\\ Q_{S_1c}\\ \vdots\\ Q_{S_lc}\end{bmatrix}\Delta\beta_c \tag{19}$$

where $\mathcal{Q}$ is defined as:

$$\mathcal{Q} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{S_1,S_1} & \cdots & Q_{S_1,S_l} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{S_l,S_1} & \cdots & Q_{S_l,S_l} \end{bmatrix} \tag{20}$$

From (19),

$$\begin{bmatrix} \Delta b \\ \Delta\beta_{S_1} \\ \vdots \\ \Delta\beta_{S_l} \end{bmatrix} = -\mathcal{Q}^{-1} \cdot \begin{bmatrix} 1 \\ Q_{S_1 c} \\ \vdots \\ Q_{S_l c} \end{bmatrix} \Delta\beta_c \tag{21}$$

and thus,

$$\begin{aligned} \Delta b &= \delta \Delta\beta_c & & (22) \\ \Delta\beta_j &= \delta_j \Delta\beta_c \qquad \forall j \in S & & (23) \end{aligned}$$

where

$$\begin{bmatrix} \delta \\ \delta_{S_1} \\ \vdots \\ \delta_{S_l} \end{bmatrix} = -\mathcal{R} \begin{bmatrix} 1 \\ Q_{S_1 c} \\ \vdots \\ Q_{S_l c} \end{bmatrix} \tag{24}$$

and $\mathcal{R} = \mathcal{Q}^{-1}$.

Equations (22) and (23) show how the variation in the β_c value of a new vector c influences β_i values of vectors $i \in S$. The δ values are named *coefficient sensitivities* from (Cauwenberghs & Poggio, 2000). Note that β values for vectors not in S do not change while these vectors do not transfer to another set. Thus, we can extend equation (23) to all vectors in D by setting $\delta_i \equiv 0$ for $i \notin S$.

Now, we can obtain for vectors $i \notin S$ how g_i and g_i^* change as β_c changes. From equation (14), we replace $\Delta\beta_j$ and Δb by their equivalence in equations (22) and (23).

$$\begin{aligned} \Delta g_i = Q_{ic}\Delta\beta_c + \sum_{j\in S} Q_{ij}\Delta\beta_j + \Delta b &= \\ Q_{ic}\Delta\beta_c + \sum_{j\in S} Q_{ij}\delta_j\Delta\beta_c + \delta\Delta\beta_c &= \\ \big(Q_{ic} + \sum_{j\in S} Q_{ij}\delta_j + \delta\big)\Delta\beta_c &= \\ \gamma_i\Delta\beta_c & \qquad (25) \end{aligned}$$

where

$$\gamma_i = Q_{ic} + \sum_{j\in S} Q_{ij}\delta_j + \delta \qquad \forall i \notin S \tag{26}$$

The γ values are named *margin sensitivities* and are defined only for non margin support vectors because for $i \in S$, $\Delta g_i = 0$. As we have done with coefficient sensitivities, if we extend equation (25) to all vectors in D, we must set $\gamma_i \equiv 0$ for $i \in S$.

Equation (25) shows how g_i changes as β_c changes, but indirectly also shows how g_i^* changes, because equation (15) states that $\Delta g_i^* = -\Delta g_i$.

Summarizing, equation (25) shows, for vectors not in S, how g_i and g_i^* values change as β_c changes (note that their β value does not change). Equation (22) shows how β_i for vectors $i \in S$ change as β_c changes (note that Δg_i and Δg_i^* is 0 for these vectors). Finally, equation (23) shows how b varies as β_c changes.

All these equations are valid while vectors do not migrate from set R, S, E or E^* to another one. But in some cases, in order to reach a consistent KT conditions for the new vector c, it could be necessary to change first the membership of some vectors to these sets. Well, do not worry. *Modify β_c in the right direction (increment or decrement) until one migration is forced. Migrate the vector updating S, E, E^* and R sets adequately, and then continue the variation of β_c.*

3.1.2 Migration of vectors between sets

This section describes all possible different kinds of migrations between sets S, E, E^* and R, and how they can be detected. One vector can migrate only from its current set to a neighbor set. Figure 1 shows the geometrical interpretation of each set and from it we can infer the following possible migrations.

from E to S: One error support vector becomes a margin support vector. This migration can be detected when updating g_i for $i \in E$ following equation (25), g_i (that was negative) becomes 0.

The maximum variation in β_c that does not imply migrations from E to S can be calculated as follows: The maximum Δg_i allowed for one vector $i \in E$ is $(0 - g_i)$, that is, from $g_i < 0$ to $g_i = 0$. From equation (25) we have, $\Delta\beta_c = \Delta g_i \gamma_i^{-1}$. Thus, the maximum variation allowed without the migration of vector i from E to S can be equated as: $(0 - g_i)\gamma_i^{-1}$. Calculating this value for all vectors in E and selecting the minimum value, we obtain the maximum variation allowed in β_c that does not force migration of vectors from E to S.

from S to E: One margin support vector becomes an error support vector. This migration is detected when, updating β_i for $i \in S$ following equation (23), β_i (that was $0 < \beta_i < C$) becomes C.

Similarly to the previous case, from equation (23), $\Delta\beta_c = \Delta\beta_i \delta_i^{-1}$. Thus, the maximum variation allowed without the migration of vector i from S to E can be formulated as: $(C - \beta_i)\delta_i^{-1}$. Calculating this value for all vectors in S and selecting the minimum value, we obtain the maximum variation allowed in β_c that does not force migration of vectors from S to E.

from S to R: One margin support vector becomes a remainder vector. This happens when updating β_i for $i \in S$ following equation (23), β_i (that was $0 < \beta_i < C$ or $-C < \beta_i < 0$) turns into 0.

The maximum variation allowed without the migration of vector i from S to R can be formulated as in the previous case as follows: $(0 - \beta_i)\delta_i^{-1}$. Calculating this value for all vectors in S and selecting the minimum value, we obtain the maximum variation allowed in β_c that does not force migration of vectors from S to R.

from R to S: One remainder vector becomes a margin support vector. This case is detected when the update of g_i or g_i^* for $i \in R$ (thus with $g_i > 0$ and $g_i^* > 0$) causes that one value becomes 0.

The maximum variation in β_c that does not imply migrations from R to S is calculated by collecting $(0 - g_i)\gamma_i^{-1}$ and $(0 - g_i^*)\gamma_i^{-1}$ for all vectors in R and selecting the minimum value. This is the maximum variation allowed in β_c that does not force migration of vectors from R to S.

from S to E^*: One margin support vector becomes an error support vector. This case is detected when, in the update of β_i for $i \in S$ the value changes from $-C < \beta_i < 0$ to $-C$.

The maximum variation in β_c that does not imply migrations from S to E^* is calculated by collecting $(-C - \beta_i)\delta_i^{-1}$ for all vectors in S and selecting the minimum value.

from E^* to S: One error support vector becomes a margin support vector. This last case is detected when updating g_i^* for vectors $i \in E^*$, the value for one vector becomes $g_i^* = 0$.

The maximum variation in β_c that does not imply migrations from E^* to S is calculated by collecting $(0 - g_i^*)\gamma_i^{-1}$ for all vectors in E^* and selecting the minimum value.

The only memory resources required in order to monitorize KT conditions fulfilled by vectors in D are: g_i and g_i^* for vectors $i \notin S$, and β_i for vectors $i \in S$. In addition, in order to efficiently update these variables we also need to maintain Q_{ij} for $i, j \in S$ –needed in equation (26)–, and $\mathcal{R}$ –needed in equation (24).

Note that each possible migration is *from S* or *to S* and thus, after any migration, S must be updated. This implies that, in addition to the update of g_i and g_i^* for vectors $i \notin S$, and the update of β_i for $i \in \beta_i$, also matrixes Q_{ij} for $i, j \in S$ and $\mathcal{R}$, must be updated. To update matrix Q is easy because it only consists in adding/removing the row and column with the kernel values of the margin support vector added/removed. But the efficient update of matrix $\mathcal{R}$ is not obvious. In the following section we describe how to efficiently maintain matrix $\mathcal{R}$.

3.1.3 Updating $\mathcal{R}$

Matrix $\mathcal{R}$ is defined in (24) as the inverse of $\mathcal{Q}$, which at the same time, is defined in (20). Note that we only need $\mathcal{R}$ for the update of β values, not $\mathcal{Q}$. When one vector becomes a margin support vector (for instance due to a migration from another set) matrix $\mathcal{Q}$ should be updated and, thus, $\mathcal{R}$ should be updated too. The naive idea of maintaining $\mathcal{Q}$ and calculate its inverse to obtain $\mathcal{R}$ is expensive in memory and time resources. Instead of this, we will work on $\mathcal{R}$ directly.

The updating procedure is an adaptation of the method proposed by (Cauwenberghs & Poggio, 2000) for classification to the regression problem.

On one hand, when we are adding one margin support vector c, matrix $\mathcal{R}$ is updated as follows:

$$\mathcal{R} := \left[\begin{array}{ccc|c} & & & 0 \\ & \mathcal{R} & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 0 \end{array}\right] + \frac{1}{\delta_c}\begin{bmatrix} \delta \\ \delta_{S_1} \\ \vdots \\ \delta_{S_l} \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \delta & \delta_{S_1} & \cdots & \delta_{S_l} & 1 \end{bmatrix} \tag{27}$$

On the other hand, when margin support vector k is removed, matrix $\mathcal{R}$ is updated as follows:

$$\mathcal{R}_{ij} := \mathcal{R}_{ij} - \mathcal{R}_{kk}^{-1}\mathcal{R}_{ik}\mathcal{R}_{kj} \qquad \forall j, i \neq k \in [0..l] \tag{28}$$

where the index 0 refers to the b-term.

Finally, to end the recursive definition of the $\mathcal{R}$ matrix updating, it remains to define the base case. When adding the first margin support vector, the matrix is initialized as follows:

$$\mathcal{R} := \mathcal{Q}^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & Q_{cc} \end{bmatrix}^{-1} = \begin{bmatrix} -Q_{cc} & 1 \\ 1 & 0 \end{bmatrix} \tag{29}$$

3.1.4 Procedure for adding one new vector

Taking into account the considerations of the previous sections, the procedure for the incremental addition of one vector results as follows:

```
1. Set $\beta_c$ to 0
2. If $g_c > 0$ and $g_c^* > 0$ Then add $c$ to $R$ and exit
3. If $g_c \leq 0$ Then
     Increment $\beta_c$, updating $\beta$ for $i \in S$ and
     $g_i, g_i^*$ for $i \notin S$, until one of the following
     conditions holds:
        - $g_c = 0$: add $c$ to $S$, update $\mathcal{R}$ and exit
        - $\beta_c = C$: add $c$ to $E$ and exit
        - one vector migrates from/to sets $E, E^*$
          or $R$ to/from $S$: update set memberships
          and update $\mathcal{R}$ matrix.
   Else $\{g_c^* \leq 0\}$
     Decrement $\beta_c$, updating $\beta$ for $i \in S$ and
     $g_i, g_i^*$ for $i \notin S$, until one of the following
     conditions holds:
        - $g_c^* = 0$: add $c$ to $S$, update $\mathcal{R}$ and exit
        - $\beta_c = -C$: add $c$ to $E^*$ and exit
        - one vector migrates from/to sets $E, E^*$
          or $R$ to/from $S$: update set memberships
          and update $\mathcal{R}$ matrix.
4. Return to 3
```

In this procedure, the influence on the regression of vector c to be added (β_c) is incremented until it reaches a consistent KT condition. Increments in β_c are done monitoring g_i, g_i^* and β_i of the whole set of vectors D. When one vector i does no longer fulfill the KT conditions associated with the set where it was assigned, the vector is transferred to the appropriate set and variables are updated as necessary.

This procedure always converges. The time cost to add one vector is linear in time with the number of vectors in D. The memory resources needed are quadratic in the number of vectors in S, because of matrix $\mathcal{R}$.

3.2 Removing one vector

The procedure for removing one vector from D uses the same principles that the procedure for adding one new vector.

One vector c can be safely removed from D only when it does not have any influence on the regression tube. This only happens when the vector lies inside the ε-tube, or in other words, when $\beta_c = 0$.

If β_c is not 0, the value must be incremented or decremented (depending on the sign of β_c) until it reaches 0. As in the case of adding one new vector, the modification of β_c can change the membership to E, E^*, R and S of some other vectors in D. Thus, the modification of β_c must be done carefully, keeping an eye on possible migrations of vectors between sets. The algorithm for the on-line removal of one vector is the following:

```
1. If $g_c > 0$ and $g_c^* > 0$ Then remove
     $c$ from $R$ and exit
2. If $g_c \leq 0$ Then
     Decrement $\beta_c$, updating $\beta$ for $i \in S$ and
     $g_i, g_i^*$ for $i \notin S$, until one of the following
     conditions holds:
       - $\beta_c = 0$: remove $c$ from $R$ and exit
       - one vector migrates from/to sets $E, E^*$
         or $R$ to/from $S$: update set memberships
         and update $\mathcal{R}$ matrix.
   Else $\{g_c^* \leq 0\}$
     Increment $\beta_c$, updating $\beta$ for $i \in S$ and
     $g_i, g_i^*$ for $i \notin S$, until one of the following
     conditions holds:
       - $\beta_c = 0$: remove $c$ from $R$ and exit
       - one vector migrates from/to sets $E, E^*$
         or $R$ to/from $S$: update set memberships
         and update $\mathcal{R}$ matrix.
3. Return to 2
```

As in the case of on-line addition of one vector, the procedure always converge. The time cost is linear in $|D|$ while the memory cost is quadratic in $|S|$.

3.3 Updating target value for existing data

The obvious way to update the target value for one existing vector c in D consists in making good use of the previous actions. In order to update the pair $<x_c, y_c>$ to $<x_c, y'_c>$ we can follow this procedure:

```
1. on-line removal of $<x_c, y_c>$
2. on-line addition of $<x_c, y'_c>$
```

Equations (8) and (9) show that the update of the target value y_c changes g_c and g_c^*. Thus, usually after an update, g_c, g_c^* and β_c values are no longer consistent with KT conditions. Thus, an alternative way of updating the target value would consist in varying β_c until this value becomes KT-consistent with g_c and g_c^*. We left as an exercise to the reader the implementation of this procedure.

4 Conclusions

In this paper, we have shown the first *on-line* procedure for building ε-insensitive SVMs for regression. An implementation of this method for Matlab is available at `http://www.lsi.upc.es/~mmartin/svmr.html`.

The aim of this paper is to open the door to SVM function approximation for applications that receive training data in an incremental way, for instance on-line prediction of temporal series, and to applications where the target for the training data changes very often, for instance reinforcement learning.

In addition to the on-line property, the proposed method presents some interesting features when compared with other exact methods like QP. First, the memory resources needed are quadratic in the number of *margin support vectors*, not quadratic on the total number of vectors. Second, empirical tests of the algorithm on several regression sets show comparable (or better) speeds in convergence, which means that the on-line learning procedure presented here is adequate even when the on-line property is not strictly required.

Acknowledgements

I would thank in general to the people of the SVM seminar at the LSI department in the UPC for the enlightening discussions about SVM topics, but specially to Cecilio Angulo for his comments and interest in this paper. I would also thank to Gert Cauwenberghs for making the incremental SVMc program available. This work has been partially supported by the TIC 2000-1011 project from CICyT, Spain.

References

Cauwenberghs, G., & Poggio, T. (2000). Incremental and decremental support vector machine learning. *Fourteenth conference on Advances in Neural Infomation Processing Systems, NIPS* (pp. 409–415).

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge University Press.

Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. *7th International Conference on Information and Knowledge Management, ACM-CIKM98* (pp. 148–155).

Osuna, E., Freund, R., & Girosi, F. (1997). Training support vector machines: an application to face detection. *International Conference on Computer Vision and Pattern Recognition, CVPR97* (pp. 30–136).

Smola, A., & Schölkopf, B. (1998). *A tutorial on support vector regression* (Technical Report NC2-TR-1998-030). NeuroCOLT2.

Sutton, R., & Barto., A. (1998). *Reinforcement learning*. MIT Press.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Heidelberg, DE: Springer Verlag.