Abstract

Conventional batch implementations of Support Vector Regression (SVR) are inefficient when used for applications such as online learning or cross-validation, because one must retrain from scratch every time the training set is modified. We introduce an Accurate Online Support Vector Regression (AOSVR) algorithm which efficiently updates a trained SVR function whenever a sample is added to or removed from the training set. The updated SVR function is identical to the one that would be produced by a batch algorithm. Examples are presented that compare the performance of AOSVR to a batch algorithm in an online and in a cross-validation scenario.

Keywords: Accurate Online Support Vector Regression; Support Vector Regression; Online Time-series Prediction; Leave-one-out Cross-validation.

1. Introduction

Support Vector Regression (SVR) fits a continuous-valued function to data in a way that shares many of the advantages of support vector machine (SVM) classification. Most algorithms for SVR (Smola et al. 1998; Chang et al. 2002) require that training samples be delivered in a single batch. For applications such as online time-series prediction or leave-one-out cross-validation, a new model is desired each time a new sample is added to (or removed from) the training set. Retraining from scratch for each new data point can
be very expensive. *Approximate* online training algorithms have previously been proposed for SVMs (Syed et al.1999; Csato et al. 2001; Gentile 2001; Graepel et al. 2001; Herbster 2001; Li et al. 1999; Kivinen et al. 2002; Ralaivola et al. 2001). We propose an accurate online support vector regression (AOSVR) algorithm that follows the approach of (Cauwenberghs et al. 2001) for incremental SVM classification.

This paper is organized as follows. The formulation of the SVR problem, and the development of the Karush-Kuhn-Tucker (KKT) conditions that its solution must satisfy, are presented in Section 2. The incremental SVR algorithm is derived in Section 3, and a decremental version is described in Section 4. Two applications of the AOSVR algorithm are presented in Section 5.

2. Support Vector Regression and The Karush-Kuhn-Tucker conditions

A more detailed version of the following presentation of SVR theory can be found in Smola et al. (1998).

Given a training set $T = \{(x_i, y_i), i=1...l\}$, where $x_i \in \mathbb{R}^N$, and $y_i \in \mathbb{R}$, we construct a linear regression function

$$f(x) = \mathbf{W}^T \Phi(x) + b$$

(1)
on a feature space $F$. Here, $\mathbf{W}$ is a vector in $F$, and $\Phi(x)$ maps the input $x$ to a vector in $F$. The $\mathbf{W}$ and $b$ in (1) are obtained by solving an optimization problem:

$$\min_{\mathbf{W},b} \quad P = \frac{1}{2} \mathbf{W}^T \mathbf{W} + C \sum_{i=1}^{l} (\xi_i + \xi_i^*)$$

s.t. $y_i - (\mathbf{W}^T \Phi(x) + b) \leq \varepsilon + \xi_i$

$(\mathbf{W}^T \Phi(x) + b) - y_i \leq \varepsilon + \xi_i^*$

$\xi_i, \xi_i^* \geq 0, \quad i = 1\cdots l$  

(2)
The optimization criterion penalizes data points whose $y$-values differ from $f(x)$ by more than $\varepsilon$. The slack variables, $\xi$ and $\xi^*$, correspond to the size of this excess deviation for positive and negative deviations, respectively, as shown in Figure 1.

![Figure 1. The $\varepsilon$-insensitive loss function and the role of the slack variables $\xi$ and $\xi^*$](image)

Introducing Lagrange multipliers $\alpha$, $\alpha^*$, $\eta$ and $\eta^*$, we can write the corresponding Lagrangian as:

$$
L_p = \frac{1}{2} W^T W + C \sum_{i=1}^l \left( \xi_i + \xi_i^* \right) - \sum_{i=1}^l \left( \eta_i \xi_i + \eta_i^* \xi_i^* \right) - \\
\sum_{i=1}^l \alpha_i (\varepsilon + \xi_i + y_i - W^T \Phi(x_i) - b) - \sum_{i=1}^l \alpha_i^* (\varepsilon + \xi_i^* - y_i - W^T \Phi(x_i) + b)
$$

s.t. $\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0$

This in turn leads to the dual optimization problem:

$$
\min_{\alpha, \alpha^*} \quad D = \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l Q_{ij} (\alpha_i - \alpha_i^*) (\alpha_j - \alpha_j^*) + \varepsilon \sum_{i=1}^l (\alpha_i + \alpha_i^*) - \sum_{i=1}^l y_i (\alpha_i - \alpha_i^*)
$$

s.t. $0 \leq \alpha_i, \alpha_i^* \leq C \quad i = 1 \cdots l,$

$$
\sum_{i=1}^l (\alpha_i - \alpha_i^*) = 0
$$

(3)

where $Q_{ij} = \Phi(x_i)^T \Phi(x_j) = K(x_i, x_j)$. Here $K(x_i, x_j)$ is a kernel function (Smola et al. 1998). Given the solution of (3), the regression function (1) can be written:
The Lagrange formulation of (3) can be represented as:

\[
L_D = \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} Q_{ij}(\alpha_i - \alpha_j^*)(\alpha_j - \alpha_j^*) + \varepsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) - \sum_{i=1}^{l} y_i (\alpha_i - \alpha_i^*) \\
- \sum_{i=1}^{l} (\delta_i \alpha_i + \delta_i^* \alpha_i^*) + \sum_{i=1}^{l} [u_i (\alpha_i - C) + u_i^* (\alpha_i^* - C)] + \beta \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)
\]

where \( \delta_i^{(*)}, \ u_i^{(*)}, \) and \( \beta \) are the Lagrange multipliers. Optimizing this Lagrangian leads to the Karush-Kuhn-Tucker (KKT) conditions:

\[
\frac{\partial L_D}{\partial \alpha_i} = \sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) + \varepsilon - y_i + \beta - \delta_i + u_i = 0 \\
\frac{\partial L_D}{\partial \alpha_i^*} = -\sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) + \varepsilon + y_i - \beta - \delta_i^* + u_i^* = 0
\]

\[
\delta_i^{(*)} \geq 0, \quad \delta_i^{(*)} \alpha_i^{(*)} = 0 \quad u_i^{(*)} \geq 0, \quad u_i^{(*)} (\alpha_i^{(*)} - C) = 0
\]

Note that \( \beta \) in (6) is the same as \( b \) in (1) and (4) at optimality (Chang et al. 2002).

Define a margin function \( h(x_i) \) for the \( i \)th sample \( x_i \) as:

\[
h(x_i) \equiv f(x_i) - y_i = \sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) - y_i + b.
\]

By combining (6), and (7), \( h(x_i) \) can be rearranged as:

\[
\begin{cases}
h(x_i) \geq -\varepsilon, & \alpha_i = 0 \\
h(x_i) = -\varepsilon, & 0 < \alpha_i < C \\
h(x_i) \leq -\varepsilon, & \alpha_i = C \\
h(x_i) \leq \varepsilon, & \alpha_i^* = 0 \\
h(x_i) = \varepsilon, & 0 < \alpha_i^* < C \\
h(x_i) \geq \varepsilon, & \alpha_i^* = C
\end{cases}
\]

According to the KKT conditions (6), at most one of \( \alpha_i \) and \( \alpha_i^* \) will be nonzero, and both are nonnegative. That is,
\[
\alpha_i > 0 \Rightarrow \alpha_i^* = 0 \quad \text{and} \quad \alpha_i^* > 0 \Rightarrow \alpha_i = 0
\]  \hspace{1cm} (9)

Therefore, we can define a coefficient difference \( \theta_i \) as

\[
\theta_i \triangleq \alpha_i - \alpha_i^*
\]  \hspace{1cm} (10)

and note that \( \theta_i \) determines both \( \alpha_i \) and \( \alpha_i^* \).

Combining (8), (9), and (10), we can obtain:

\[
\begin{cases}
  h(x_i) > \varepsilon, & \theta_i = -C \\
  h(x_i) = \varepsilon, & -C < \theta_i < 0 \\
  -\varepsilon \leq h(x_i) \leq \varepsilon, & \theta_i = 0 \\
  h(x_i) = -\varepsilon, & 0 < \theta_i < C \\
  h(x_i) \leq -\varepsilon, & \theta_i = C
\end{cases}
\]  \hspace{1cm} (11)

Equation (11) suggests that the samples in training set \( T \) can be classified into three subsets.

**E Set:** Error Support Vectors: \( E = \{i \mid |\theta_i| = C\} \)

**S Set:** Margin Support Vectors: \( S = \{i \mid 0 < |\theta_i| < C\} \)  \hspace{1cm} (12)

**R Set:** Remaining Samples: \( R = \{i \mid \theta_i = 0\} \)

3. **Incremental Algorithm**

The incremental algorithm updates the trained SVR function whenever a new sample \( x_c \) is added to the training set \( T \). The basic idea is to gradually change the coefficient difference \( \theta_c \) corresponding to the new sample \( x_c \) until it meets the KKT conditions, while ensuring that the existing samples in \( T \) continue to satisfy the KKT conditions. In this section, we first derive the relation between the change of \( \theta_c \), or \( \Delta \theta_c \), and the change of other coefficients under the KKT conditions, and then propose a method to determine
the largest allowed $\Delta \theta_c$ for each step. A pseudo-code description of this algorithm is provided in the Appendix.

### 3.1 Derivation of the Incremental Relations

Let $x_c$ be a new training sample that is added to $T$. We initially set $\theta_c = 0$ and then gradually change (increase or decrease) the value of $\theta_c$ under the KKT conditions (11).

According to (6), (7), (8), and (11), the incremental relation between $\Delta h(x_i)$, $\Delta \theta_i$, and $\Delta b$ is given by:

\[
\Delta h(x_i) = Q_{ic} \Delta \theta_c + \sum_{j=1}^{j=1} Q_{ij} \Delta \theta_j + \Delta b
\]

From the equality condition in (3), we have

\[
\theta_c + \sum_{i=1}^{j=1} \theta_i = 0
\]

Combining (11), (12), (13), and (14), we obtain:

\[
\sum_{j \in S} Q_{ij} \Delta \theta_j + \Delta b = -Q_{ic} \Delta \theta_c \quad \text{where} \quad i \in S
\]

\[
\sum_{j \in S} \Delta \theta_j = -\Delta \theta_c
\]

If we define the index of the samples in the $S$ set as:

\[
S = \{s_1, s_2, \cdots s_i\}
\]

Equation (15) can be represented in matrix form as:

\[
\begin{bmatrix}
0 & 1 & \cdots & 1 \\
1 & Q_{s_1 s_1} & \cdots & Q_{s_1 s_i} \\
\vdots & \vdots & \ddots & \vdots \\
1 & Q_{s_i s_1} & \cdots & Q_{s_i s_i}
\end{bmatrix}
\begin{bmatrix}
\Delta b \\
\Delta \theta_{s_1} \\
\vdots \\
\Delta \theta_{s_i}
\end{bmatrix} =
\begin{bmatrix}
1 \\
\vdots \\
Q_{s_i c}
\end{bmatrix}
\]

That is,
\[
\begin{bmatrix}
\Delta b \\
\Delta \theta_{n_1} \\
\vdots \\
\Delta \theta_{n_k}
\end{bmatrix} = \beta \Delta \theta_c 
\]

where

\[
\beta = \beta_{n_1} \cdots \beta_{n_k} = -R \begin{bmatrix}
\beta_{n_1} \\
\vdots \\
\beta_{n_k}
\end{bmatrix} = -R \begin{bmatrix}
Q_{n_1c} \\
\vdots \\
Q_{n_kc}
\end{bmatrix} = -R \begin{bmatrix}
1 & Q_{n_1} & \cdots & Q_{n_1n_k} \\
1 & Q_{n_2} & \cdots & Q_{n_2n_k} \\
\vdots & \vdots & \ddots & \vdots \\
1 & Q_{n_k} & \cdots & Q_{n_kn_k}
\end{bmatrix}^{-1} \begin{bmatrix}
1 \\
Q_{n_1c} \\
\vdots \\
Q_{n_kc}
\end{bmatrix} 
\]

(19)

Define a non-S, or N set, as \(N = E \cup R = \{n_1, n_2, \ldots, n_k\}\). Combining (11), (12), (13), and (18), we obtain

\[
\begin{bmatrix}
\Delta h(x_{n_1}) \\
\Delta h(x_{n_2}) \\
\vdots \\
\Delta h(x_{n_k})
\end{bmatrix} = \gamma \Delta \theta_c 
\]

(20)

where,

\[
\gamma = \begin{bmatrix}
Q_{n_1c} \\
Q_{n_2c} \\
\vdots \\
Q_{n_kc}
\end{bmatrix} + \begin{bmatrix}
1 & Q_{n_1} & \cdots & Q_{n_1n_k} \\
1 & Q_{n_2} & \cdots & Q_{n_2n_k} \\
\vdots & \vdots & \ddots & \vdots \\
1 & Q_{n_k} & \cdots & Q_{n_kn_k}
\end{bmatrix} \beta 
\]

(20b)

In special case, when S set is empty, according to (13) and (14), Equation (20) simplifies to:

\[
\begin{bmatrix}
\Delta h(x_{n_1}) \\
\Delta h(x_{n_2}) \\
\vdots \\
\Delta h(x_{n_k})
\end{bmatrix} = \begin{bmatrix}
1 \\
1 \\
\vdots \\
1
\end{bmatrix} \Delta b 
\]

(21)
Given $\Delta \theta$, we can update $\theta_i, i \in S$ and $b$ according to (18), and update $h(x_i), i \in N$ according to (20). Moreover, (11) suggests that $\theta_i, i \in N$ and $h(x_i), i \in S$ are constant if the $S$ set stays unchanged. Therefore, the results presented in this section enable us to update all the $\theta_i$ and $h(x_i)$ given $\Delta \theta_c$. In the next section, we address the question of how to find an appropriate $\Delta \theta_c$.

3.2. AOSVR Bookkeeping Procedure

Equations (18) and (20) hold only when the samples in the $S$ set do not change. Therefore, $\Delta \theta_c$ is chosen to be the largest value that either can maintain the $S$ set unchanged or lead to the termination of the incremental algorithm.

The first step is to determine whether the change $\Delta \theta_c$ should be positive or negative. According to (11),

$$
\text{sign}(\Delta \theta_c) = \text{sign}(y_c - f(x_c)) = \text{sign}(-h(x_c))
$$

The next step is to determine a bound on $\Delta \theta_c$ imposed by each sample in the training set. To simplify exposition we only consider the case $\Delta \theta_c > 0$, and the case $\Delta \theta_c < 0$ can be obtained similarly.

For the new sample $x_c$, there are two cases:

Case 1: $h(x_c)$ changes from $h(x_c) < -\varepsilon$ to $h(x_c) = -\varepsilon$, and the new sample $x_c$ is added into $S$ set, and the algorithm terminates.

Case 2: If $\theta_c$ increase from $\theta_c < C$ to $\theta_c = C$, the new sample $x_c$ is added into $E$ set, and the algorithm terminates.

For each sample $x_i$ in the set $S$,
Case 3: If $\theta_i$ changes from $0 < |\theta_i| < C$ to $|\theta_i| = C$, sample $x_i$ changes from $S$ set to $E$ set; 
If $\theta_i$ changes to $\theta_i = 0$, sample $x_i$ changes from $S$ set to $R$ set.

For each sample $x_i$ in the set $E$, 

Case 4: If $h(x_i)$ changes from $|h(x_i)| > \varepsilon$ to $|h(x_i)| = \varepsilon$, $x_i$ is moved from $E$ set to $S$ set.

For each sample $x_i$ in the set $R$, 

Case 5: If $h(x_i)$ changes from $|h(x_i)| < \varepsilon$ to $|h(x_i)| = \varepsilon$, $x_i$ is moved from $R$ set to $S$ set.

The bookkeeping procedure is to trace each sample in the training set $T$ against these five cases, and determine the allowed $\Delta \theta_c$ for each sample according to (18) or (20). The final $\Delta \theta_c$ is defined as the one with minimal absolute value among all the possible $\Delta \theta_c$.

3.3. Efficiently Updating $R$ Matrix

When the $S$ set is not example, the matrix $R$ that is used in (19)

$$ R = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{s_1} & \cdots & Q_{s_{k_1}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{s_{k_s}} & \cdots & Q_{s_{k_s}} \end{bmatrix}^{-1} $$

(23)

must be updated whenever the $S$ set is changed. Following (Cauwenberghs et al. 2001) we can efficiently update $R$ without explicitly computing the matrix inverse. When the $k^{th}$ sample $x_{s_k}$ in $S$ set is removed from the $S$ set, the new $R$ can be obtained as follows:

$$ R_{\text{new}}^{\text{new}} = R_{1,1} - \frac{R_{1,k} R_{k,1}}{R_{k,k}}, \text{ where } I = [1 \cdots k \ k+2 \cdots S_{k}+1] $$

(24)

When the new sample $x_{c}$, or a sample $x_i$ in $E$ set or $R$ set, is added to $S$ set, the new $R$ can be updated as follows:
\[
R^{\text{new}} = \begin{bmatrix}
R & 0 \\
0 & \ddots & 0 \\
0 & \cdots & 0
\end{bmatrix} + \frac{1}{\gamma_i} \begin{bmatrix}
\beta \\
1
\end{bmatrix} \begin{bmatrix}
\beta^T \\
1
\end{bmatrix}
\]

where \( \beta \) is defined as \( \beta = -R Q_{y,i} \), and \( \gamma_i \) is defined as \( \gamma_i = \begin{bmatrix}
\beta^T \\
1
\end{bmatrix} \begin{bmatrix}
Q_{y,i} \\
\vdots \\
Q_{s,i}
\end{bmatrix} \) when the sample \( x_i \) was moved from \( E \) set or \( R \) set. In contrast, when the sample \( x_c \) is the sample added to \( S \) set, \( \beta \) is can be obtained according to (18), and \( \gamma_i \) is the last element of \( \gamma \) defined in (20).

3.4. Initialization of the Incremental Algorithm

An initial SVR solution can be obtained from a batch SVR solution, and in most cases that is the most efficient approach. But it is sometimes convenient to use AOSVR to produce a full solution from scratch. An efficient starting point is the two-sample solution, which can be written analytically. Assume training set \( T = \{(x_1, y_1), (x_2, y_2)\} \), and \( y_1 \geq y_2 \). The solution of (3) will be

\[
\begin{align*}
\theta_1 &= \max(0, \min(C, \frac{y_1 - y_2 - 2\varepsilon}{2(K_{11} - K_{12})})) \\
\theta_2 &= -\theta_1 \\
b &= (y_1 + y_2) / 2
\end{align*}
\]

4. Decremental Algorithm

The decremental (or “unlearning”) algorithm is employed when an existing sample is removed from the training set. If a sample \( x_c \) is in the \( R \) set, then it does not contribute to
the SVR solution, and removing it from the training set is trivial; no adjustments are needed. If on the other hand, \( x_c \) has a nonzero coefficient, then the idea is to gradually reduce the value of the coefficient to zero, while ensuring all the other samples in training set continue to satisfy the KKT conditions.

The general algorithm is almost the same as the incremental algorithm except for a few small adjustments:

(i) The direction of the change of \( \theta_c \) is now changed to be:

\[
\text{sign}(\Delta \theta_c) = \text{sign}(f(x_c) - y_c) = \text{sign}(h(x_c)).
\]

(ii) There is no Case 1 because the removed \( x_c \) need not satisfy KKT conditions.

(iii) The condition in Case 2 becomes: \( \theta_c \) changing from \( |\theta_c| > 0 \) to \( \theta_c = 0 \).

5. Applications

The accurate online SVR (AOSVR) learning algorithm produces exactly the same SVR as the conventional batch SVR learning algorithm, and can be applied in all scenarios where batch SVR is currently employed. In this section, two particular applications of AOSVR are used to illustrate the particular efficiency of AOSVR for incremental learning.

In both applications, experimental results are presented to compare the efficiency of AOSVR with that of the batch SVR. Our version of AOSVR is implemented in Matlab, and for the batch SVR, we used LibSVM (Chang et al. 2001), which is implemented in C++. This leads to an apples-and-oranges comparison, but we found the currently available Matlab codes for batch SVR to be prohibitively slow. For example, although AOSVR should be slower than batch SVR on a batch SVR problem, we found that it took
AOSVR 4.34 seconds to train a predictor for the 292-point sunspot yearly time-series, while the Matlab SVM Toolbox (Gunn 1998) took 143.06 seconds. We expect a C++ implementation to be faster than Matlab, so the comparison with LibSVM gives batch SVR an advantage – but despite this, we find that AOSVR outperforms batch SVR in the online scenarios presented here.

5.1. Online Time-series Prediction

In recent years, the use of SVR for time-series prediction has attracted increased attention (Müller et al. 1997; Fernández 1999; Tay et al. 2001). In an online scenario, one updates a model from incoming data and at the same time makes predictions based on that model. This arises, for instance, in market forecasting scenarios. Another potential application is the (near) real-time prediction of electron density around a satellite in the magnetosphere, because high charge densities can damage satellite equipment (Friedel et al. 2002).

In time-series prediction, the prediction origin, denoted $O$, is the time from which the prediction is generated. The time between the prediction origin and the predicted data point is the prediction horizon, which for simplicity we will take as one time step.

A typical online time-series prediction scenario can be represented as follows (Tashman 2000):

1. Given a time series $\{x(t), \ t = 1,2,3\ldots\}$ and prediction origin $O$, construct a set of training samples, $A_{O,B}$, from the segment of time series $\{x(t), \ t = 1\ldots O\}$ as $A_{O,B} = \{(X(t), y(t)), \ t = B\ldots O - 1\}$, where $X(t) = \begin{bmatrix} x(t) & \cdots & x(t-B+1) \end{bmatrix}^T$, $y(t) = x(t+1)$, and $B$ is the embedding dimension of the training set $A_{O,B}$. 
(2) Train a predictor \( P(A_{O,B}; X) \) from the training set \( A_{O,B} \).

(3) Predict \( x(O+1) \) using \( \hat{x}(O+1) = P(A_{O,B}; X(O)) \).

(4) When \( x(O+1) \) becomes available, update the prediction origin: \( O = O+1 \).

Then, go to (1) and repeat the above procedure.

Note that the training set \( A_{O,B} \) keeps growing as \( O \) increases, so the training of the predictor in step (2) becomes increasingly expensive. Therefore, many SVR-based time-series predictions are implemented in a compromised way (Tay et al. 2001), with a fixed prediction origin \( O \). That is, after the predictor is obtained, it stays fixed, and is not updated as new data arrives. A direct consequence of this compromise is the degrading of the prediction performance, which is demonstrated by the experimental results listed in Table 2.

In contrast, an online prediction algorithm, such as AOSVR, can take advantage of the fact that the training set is augmented one sample at a time, and the enhanced efficiency that an online algorithm provides is shown in the next section.

5.1.1. Experiments

Two experiments were performed to compare the AOSVR algorithm with the batch SVR algorithm. We are careful to use the same algorithm parameters for online and batch SVR, but since our purpose is to compare computational performance, we did not attempt to optimize these parameters for each data set. In these experiments, the kernel function is a gaussian radial basis function, \( \exp(-\|X_i - X_j\|^2) \), the regularization coefficient \( C \) and the insensitivity parameter \( \epsilon \) in (2) are set to 10 and 0.1 respectively, and the embedding dimension, \( B \), of the training \( A_{O,B} \), is 5. Also, we scale all the time-series to [-1,1].
Three widely used benchmark time-series are employed in both experiments: (a) the Santa Fe Institute Competition time series A (Weigend et al. 1994), (b) the Mackey-Glass equation with $\tau=17$ (Mackey et al. 1977), and (c) the yearly average sunspot numbers recorded from 1700 to 1995. Some basic information about these time-series is listed in Table 1. The SV Ratio is the number of support vectors divided by the number of training samples. This is based on a prediction of the last data point using all previous data for training. In general, a higher SV ratio suggests that the underlying problem is harder (Vapnik 1998).

<table>
<thead>
<tr>
<th></th>
<th># Data Points</th>
<th>SV Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Santa Fe Institute</td>
<td>1000</td>
<td>4.52%</td>
</tr>
<tr>
<td>Mackey-Glass</td>
<td>1500</td>
<td>1.54%</td>
</tr>
<tr>
<td>Yearly Sunspot</td>
<td>292</td>
<td>41.81%</td>
</tr>
</tbody>
</table>

Table 1. Information Regarding Experimental Time Series

The first experiment demonstrates that using a fixed predictor produces less accurate predictions than using a predictor that is updated as new data becomes available. Two measurements are used to quantify the prediction performance: mean squared error (MSE), and mean absolute error (MAE). The predictors are initially trained on the first half of the data in the time-series. In the fixed case, the same predictor is used to predict the second half of the time-series. In the online case, the predictor is updated whenever a new data point is available. The performance measurements for both cases are calculated from the prediction and actual value of the second half data points in the time-series. As shown in Table 2, the online predictor outperforms the fixed predictor. We also note that the errors for the three time-series in Table 2 coincide with the estimated prediction difficulty in Table 1 based on SV Ratio.
Table 2. Performance Comparison For Online and Fixed Predictors

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>MSE</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Santa Fe Institute</td>
<td>Online</td>
<td>0.0072</td>
</tr>
<tr>
<td></td>
<td>Fixed</td>
<td>0.0097</td>
</tr>
<tr>
<td>Mackey-Glass</td>
<td>Online</td>
<td>0.0034</td>
</tr>
<tr>
<td></td>
<td>Fixed</td>
<td>0.0036</td>
</tr>
<tr>
<td>Yearly Sunspot</td>
<td>Online</td>
<td>0.0263</td>
</tr>
<tr>
<td></td>
<td>Fixed</td>
<td>0.0369</td>
</tr>
</tbody>
</table>

The second experiment illustrates that AOSVR is more efficient than a batch implementation in the online prediction scenario. For each benchmark time-series, an initial SVR predictor is trained on the first 20% of the data points using a batch SVR algorithm. Afterwards, both AOSVR and batch SVR algorithms are employed to work in the online prediction mode for the remaining 80% of the data points in the time-series. AOSVR and the batch SVR algorithm produce exactly the same prediction errors in this experiment, so the comparison is only of prediction speed. The experimental results of the three time-series are presented in Figures 2, 3, and 4 respectively. The x-axis of these plots is the number of data points, to which the online prediction model is applied.
Figure 2. Log and linear plots of prediction time of SFI time series

Figure 3. Log and linear plots of prediction time of Mackey-Glass time series
Figure 4. Log and linear plots of prediction time of yearly sunspot time series

These experimental results demonstrate that AOSVR algorithm is generally much faster than the batch SVR algorithm when applied to online prediction. This is because the batch SVR algorithm must train a new classifier from scratch every time a new point is added. Comparison of Figures 3 and 4 furthermore suggests that more speed improvement is achieved on the sunspot data than on the Mackey-Glass. We speculate that this is because the sunspot problem is “harder” than the Mackey-Glass – it has a higher support vector ratio – and that the performance of the AOSVR algorithm is less sensitive to problem difficulty.

To test this hypothesis, we compared the performance of AOSVR to batch SVR on a single dataset (the sunspots) whose difficulty was adjusted by changing the value of $\varepsilon$. A smaller $\varepsilon$ leads to a higher support vector ratio and a more difficult problem. Both the
AOSVR and batch SVR algorithms were employed for online prediction of the full time-series. The overall prediction times are plotted against $\varepsilon$ in Figure 5. Where AOSVR performance varied by a factor of about ten over the range of $\varepsilon$, the batch SVR performance varied by a factor of almost 200.

![Graph showing prediction time against $\varepsilon$ for AOSVR and batch SVR](image)

Figure 5. Log and linear plots of prediction time of yearly sunspot time series

5.1.2. Limited-Memory Version of the Online Time-series Prediction Scenario

One problem with online time-series prediction in general is the longer the prediction goes on, the bigger the training set $A_{o,b}$ will become, and the more SVs will be involved in SVR predictor (Kimeldorf et al. 1971). A complicated SVR predictor imposes both memory and computation stress on the prediction system. One way to deal with this problem is to impose a “forgetting” time $W$. When training set $A_{o,b}$ grows to this maximum $W$, then the decremental algorithm of AOSVR will be used to remove the oldest sample before the next new sample is added to the training set.
We note that this variant of the online prediction scenario is also potentially suitable for non-stationary time-series, as it can be updated in real-time to fit the most recent behavior of the time-series. More rigorous investigation in this direction will be a future effort.

5.2. Leave-One-Out Cross-validation

Cross-validation is a useful tool for assessing the generalization ability of a machine-learning algorithm. The idea is to train on one subset of the data, and then to test the accuracy of the predictor on a separate disjoint subset. In leave-one-out cross-validation (LOOCV), only a single sample is used for testing, and all the rest are used for training. Generally, this is repeated for every sample in the dataset. When the batch SVR is employed, LOOCV can be very expensive, since a full retraining is done for each sample. One compromise approach is to approximate LOOCV by estimating some related but less computation-demanding factors, such as the Xi-Alpha Bound (Joachims 2000), and Approximate Span Bound (Vapnik et al. 1999). Although (Lee et al. 2001) proposed a numerical solution to reduce the computation for directly implementing LOOCV, the amount of computation required is still considerable. Also, the accuracy of the LOOCV result obtained using this method can be potentially compromised due to the special parameter set employed by the method.

The decremental algorithm of AOSVR provides us with an efficient implementation of LOOCV for SVR:

(1) Given a dataset \( D \), construct the SVR function \( f(x) \) from the whole dataset \( D \) using batch SVR learning algorithm;
(2) For each non-support vector $x_i$ in the dataset $D$, calculate error $e_i$ corresponding to $x_i$ as: $e_i = y_i - f(x_i)$, where $y_i$ is the target value corresponding to $x_i$.

(3) For each support vector $x_i$ involved in the SVR function $f(x)$,

a. Unlearn $x_i$ from the SVR function $f(x)$ using the decremental algorithm to obtain the SVR function $f_i(x)$ which would be constructed from the dataset $D_i=D\setminus \{x_i\}$;

b. Calculate error $e_i$ corresponding to support vector $x_i$ as: $e_i = y_i - f_i(x_i)$, where $y_i$ is the target value corresponding to $x_i$.

(4) Knowing the error for each sample $x_i$ in $D$, it is possible to construct a variety of overall measures; a simple choice is the MSE:

$$MSE_{i,LOOCV}(D) = \frac{1}{N} \sum_{i}^{N} e_i^2$$

(28)

where $N$ is number of samples in dataset $D$. Other choices of error metric, such as MAE, can be obtained just by altering (28) appropriately.

5.2.1. Experiment

The algorithm parameters in this experiment are set the same as those in the experiments in Subsection 5.1.1. Two famous regression datasets, the auto-mpg and Boston housing datasets, are chosen from the UCI machine-learning repository. Some basic information of these datasets is listed in Table 3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Attributes</th>
<th># Samples</th>
<th>SV Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-MPG</td>
<td>7</td>
<td>392</td>
<td>41.07%</td>
</tr>
<tr>
<td>Boston Housing</td>
<td>13</td>
<td>506</td>
<td>36.36%</td>
</tr>
</tbody>
</table>

Table 3. Information Regarding Experimental Regression Datasets
The experimental results of both datasets are presented in Figure 6. The x-axis is the size of the training set, upon which the LOOCV is implemented. These plots show that AOSVR-based LOOCV is much faster than its batch SVR counterpart when the training set is relatively large.

![LOOCV Time Comparison of Auto-MPG Dataset](image1)

![LOOCV Time Comparison of Boston Housing Dataset](image2)

**Figure 6.** Linear plots of LOOCV time of Auto-MPG and Boston Housing dataset

**6. Conclusions**

We have developed and implemented an accurate online support vector regression (AOSVR) algorithm that permits efficient retraining when a new sample is added to, or when an existing sample is removed from, the training set. AOSVR is applied to online time-series prediction and to leave-one-out cross-validation, and the experimental results
demonstrate that the AOSVR algorithm is more efficient than the conventional batch SVR in these scenarios. Moreover, AOSVR appears less sensitive than batch SVR to the difficulty of the underlying problem.

Appendix

Pseudo-code for Incrementing AOSVR with a New Data Sample

**Inputs:**
- Training set $T = \{(x_i, y_i), i = 1..l\}$
- Coefficients $\{\theta_i, i = 1..l\}$, and bias $b$
- Partition of samples into sets $S, E,$ and $R$
- Matrix $R$ defined in (23)
- New sample $(x_c, y_c)$

**Outputs:**
- Updated coefficients $\{\theta_i, i = 1..l + 1\}$ and bias $b$
- Updated Matrix $R$
- Updated partition of samples into sets $S, E,$ and $R$

**AOSVR Incremental Algorithm:**
- Initialize $\theta_c = 0$
- Compute $f(x_c) = \sum_{i \in E \cup S} \theta_i Q_{ic} + b$
- Compute $h(x_c) = f(x_c) - y_c$
- If $|h(x_c)| \leq \epsilon$, assign $x_c$ to $R$, and terminate.
- Let $q = sign(-h(x_c))$ be the sign that $\Delta \theta_c$ will take
• Do until the new sample \( \mathbf{x}_c \) meets the KKT condition
  
  o Update \( \mathbf{\beta}, \mathbf{\gamma} \) according to (19) and (20b)
  
  o Start bookkeeping procedure:

    Check the new sample \( \mathbf{x}_c \),
    
    - \( L_{c1} = (-h(\mathbf{x}_c) - q\varepsilon) / \gamma_c \) (Case 1)
    
    - \( L_{c2} = qC - \theta_c \) (Case 2)

    Check each sample \( \mathbf{x}_i \) in the set \( S \) (Case 3)
    
    - If \( q\beta_i > 0 \) and \( C > \theta_i \geq 0 \), \( L^S_i = (C - \theta_i) / \beta_i \)
    
    - If \( q\beta_i > 0 \) and \( 0 > \theta_i \geq -C \), \( L^S_i = -\theta_i / \beta_i \)
    
    - If \( q\beta_i < 0 \) and \( C \geq \theta_i > 0 \), \( L^S_i = -\theta_i / \beta_i \)
    
    - If \( q\beta_i < 0 \) and \( 0 \geq \theta_i > -C \), \( L^S_i = (C - \theta_i) / \beta_i \)

    Check each sample \( \mathbf{x}_i \) in the set \( E \) (Case 4)
    
    - \( L^E_i = (-h(\mathbf{x}_i) - \text{sign}(q\beta_i)\varepsilon) / \beta_i \)

    Check each sample \( \mathbf{x}_i \) in the set \( R \) (Case 5)
    
    - \( L^R_i = (-h(\mathbf{x}_i) + \text{sign}(q\beta_i)\varepsilon) / \beta_i \)

    Set \( \Delta\theta_c = q \min(|L_{c1}|, |L_{c2}|, |L^S_i|, |L^E_i|, |L^R_i|) \),

    where \( L^S = \{L^S_i, i \in S\} \), \( L^E = \{L^E_i, i \in E\} \), and \( L^R = \{L^R_i, i \in R\} \).

    Let \( \text{Flag} \) be the case number that determines \( \Delta\theta_c \).

    Let \( \mathbf{x}_i \) be the particular sample in \( T \) that determines \( \Delta\theta_c \).

  o End Bookkeeping Procedure.
o Update $\theta_i$, $b$, and $\theta_j, i \in S$ according to (18)

o Update $h(x_i), i \in E \cup R$ according to (20)

o Switch $Flag$

  ($Flag = 1$):
  
  Add new sample $x_e$ to set $S$; update matrix $R$ according to (25)

  ($Flag = 2$):
  
  Add new sample $x_e$ to set $E$

  ($Flag = 3$):
  
  If $\theta_i = 0$, move $x_i$ to set $R$; update $R$ according to (24)
  
  If $|\theta_i| = C$, move $x_i$ to set $E$; update $R$ according to (24)

  ($Flag = 4$):
  
  Move $x_i$ to set $S$; update $R$ according to (25)

  ($Flag = 5$):
  
  Move $x_i$ to set $S$; update $R$ according to (25)

o End Switch $Flag$

o If $Flag \leq 2$, terminate; otherwise continue the Do-Loop.

- Terminate incremental algorithm; ready for the next sample.

**Acknowledgements**

We would like to thank Professor Chih-Jen Lin in National University of Taiwan for his suggestion on some implementation issues. This work is supported by the NASA project NRA-00-01-AISR-088 and by the Los Alamos Laboratory Directed Research and Development (LDRD) program.
References


