Expert Systems with Applications 39 (2012) 10933–10942

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Recurrent sparse support vector regression machines trained by active learning in the time-domain

V. Ceperic [a,b,*], G. Gielen [a], A. Baric [b]

[a] Department of Electrotechnical Engineering, Katholieke, Universiteit Leuven, Kasteelpark Arenberg 10, Leuven, Belgium
[b] Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia

ARTICLE INFO

Keywords:
Support vector machines
Support vector regression
Recurrent models
Sparse models
Active learning

ABSTRACT

A method for the sparse solution of recurrent support vector regression machines is presented. The proposed method achieves a high accuracy versus complexity and allows the user to adjust the complexity of the resulting model. The sparse representation is guaranteed by limiting the number of training data points for the support vector regression method. Each training data point is selected based on the accuracy of the fully recurrent model using the active learning principle applied to the successive time-domain data. The user can adjust the training time by selecting how often the hyper-parameters of the algorithm should be optimised. The advantages of the proposed method are illustrated on several examples, and the experiments clearly show that it is possible to reduce the number of support vectors and to significantly improve the accuracy versus complexity of recurrent support vector regression machines.

1. Introduction

(Cortes & Vapnik (1995)) presented support vector machines (SVM), a set of related supervised learning methods used for classification and regression. Since then, support vector machines have generated a lot of interest in the machine learning community due to their excellent performance in a variety of learning problems, e.g. text categorisation (Kuo & Yajima, 2010; Hao, Chiang, & Tu, 2007), image analysis (Guo, Gunn, Damper, & Nelson, 2008; Chang, Wang, & Li, 2009), solving problems in bioinformatics (Ben-Hur, Ong, Sonnenburg, Schölkopf, & Rätsch, 2008; Zhong, He, Harrison, Tai, & Pan, 2007), bankruptcy prediction (Härdle, Moro, & Hoffmann, 2010; Shin, Lee, & Kim, 2005). Some of the additional reasons for the wide use of SVMs are: theoretical guarantees about their performance, lower susceptibility to local minima and higher immunity to increased model complexity that is usually associated with adding extra dimensions to the model input space.

SVMs were initially applied for classification problems (Cortes & Vapnik, 1995), but soon after the introduction, the formulation was extended to regression problems (Drucker, Burges, Kaufman, Smola, & Vapnik, 1996; Vapnik, Golowich, & Smola, 1996). Support vector machines seem to offer excellent generalisation properties on real-life regression and classification problems while they are still capable of producing sparse models. The common formulation of SVM regression is Vapnik's ε-tube support vector regression (ε-SVR) (Drucker et al., 1996; Vapnik et al., 1996). The model produced by ε-SVR depends only on a subset of the training data while ignoring any training data within the threshold ε to the model prediction. This step unveils the potential problem: if the threshold ε is small, then the model will depend on a larger number of the overall training data, thus making the solution non-sparse, as shown in Guo, Zhang, and Zhang (2010).

In general, sparse regression models are simplified models that are characterised by low complexity. There are several benefits to sparse regression models:

- tendency to avoid over-fitting: if the model is less complex, it is less likely that the model will over-fit the data;
- decrease of computational burden in the phase of active use: the evaluation time for the SVM model is directly proportional to the number of support vectors and, if that number is decreased, the execution speed is increased;
- ability to generalise well: generalisation and over-fitting are two closely related terms in such a way that if the chance for over-fitting is decreased, the generalisation quality of the model is increased.

It should be noted that the sparsity of the model, by itself, is not a guarantee that the model will generalise well. One of the common techniques for improving generalisation and overcoming over-fitting is regularisation (Neumaier, 1998). Support vector machines in their training formulation have a regularisation term which helps to reduce over-fitting.

* Corresponding author at: Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia.
E-mail address: vladimir.ceperic@ieee.org (V. Ceperic).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2012.03.031

Several approaches for sparse modelling have already been presented, e.g. relevance vector machines (RVM) (Tipping, 2001), adaptive sparse supervised learning (ASSL) (Figueiredo, 2003), kernelised least absolute shrinkage and selection operator regression (KLASSO) (Roth, 2004), fixed-size least squares support vector machines (FS-LSSVM) (Brabanter, Brabanter, Suykens, & Moor, 2010; Suykens, 2002), Schölkopf's ν-SVR (Schölkopf, Smola, Williamson, & Bartlett, 2000), SESVR and SNSVR (Guo et al., 2010). All mentioned models achieve high accuracy while reducing complexity but are not designed for time-domain modelling or for the training of recurrent models.

The problem how to build a sparse, parsimonious model is amplified if the resulting model has to be recurrent. Several authors addressed the recurrent modelling problem by introducing rather complex models, e.g. evolution of recurrent systems with optimal linear output (EVOLINO) (Schmidhuber, Wierstra, Gagliolo, & Gomez, 2007), various types of echo state networks (ESN) (Holzmann & Hauser, 2010; Jaeger, Lukosevicius, Popovici, & Siewert, 2007; Xue, Yang, & Haykin, 2007), recursive Bayesian recurrent neural networks (Mirikitani & Nikolaev, 2010), long short-term memory (LSTM) networks (Graves & Schmidhuber, 2005). Due to the favourable properties of support vector machines, several recurrent models with SVM have been developed, e.g. recurrent least squares SVM (Suykens & Vandewalle, 2000), regularised recurrent least squares SVM (Qu, Oussar, Dreyfus, & Xu, 2009), evolution of systems with kernel-based outputs (EVOKE) (Schmidhuber, Gagliolo, Wierstra, & Gomez, 2006), recurrent support vector regression with immune algorithm hyper-parameter determination (RSVRIA) (Hong & Lai, 2008). The mentioned models might be inapplicable to real-time and embedded applications due to their complexity, i.e. they are not designed to be sparse or parsimonious.

Due to the favourable properties of ε-SVR, the goal of this paper is to stay within the formulation of ε-SVR, while reducing the model complexity and expanding the model for accurate recurrent modelling in the time-domain. To achieve this goal, the active learning principle is adapted for the time-domain recurrent modelling and applied to ε-tube support vector regression machines. The active learning principle has been reported and used for SVM classification (Jiang et al., 2007; Tong & Koller, 2002). The basic procedure of the active learning approach for classification is as follows: the algorithm creates an initial SVM classification model on a small sample of the training data, evaluates the model on the remaining training data, adds to its working set the data record that is closest to the current decision boundary (hyperplane), and then locally refines the decision hyperplane. However, this procedure is not applicable to function approximation or recurrent time-domain models.

In this paper, the sparsity is guaranteed by limiting the number of training data points for ε-SVR while sacrificing the least accuracy. To this end, the training data set is iteratively constructed, based on the influence of each training data point on the accuracy of the recurrent model. In this way, the active learning principle can be used to reduce the complexity of the SVM, which enables the development of sparse solutions based on the least number of support vectors. Additionally, the incremental construction procedure allows the user to adjust the training time by selecting in which iterations the optimisation of the algorithm's hyper-parameters is performed.

The paper is organised as follows: Section 2 gives a brief overview of support vector regression. Section 3 presents sparse recurrent support vector regression modelling by using the active learning principle. In Section 4 several modelling experiments are reported. Finally, conclusions are given in Section 5.

2. Support vector regression

The ε-tube support vector regression is briefly introduced, as it is the base of the modelling proposed in this paper.

Firstly the case of linear regression is considered. The training data is denoted as:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathbb{R}\}_{i=1}^{l}, \tag{1}$$

where y_i is the real number associated to the p-dimensional input vector $\mathbf{x}_i$. Let X denote the space of input patterns, $X \subset \mathbb{R}^p$. Let l in (1) denote the number of training data samples.

The first goal is to find a linear function f, which will later be extended to the nonlinear case, that best describes (1) in the form of

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{2}$$

where $\mathbf{w} \in X$ is the weight vector, $b \in \mathbb{R}$ is a constant and $\langle \cdot, \cdot \rangle$ denotes a dot product in X.

The function f should be as flat as possible. Flatness of the function f means that one seeks a small weight vector $\mathbf{w}$. An overview of flatness definitions of such functions is given in Schölkopf and Smola (2001). A simple way to ensure required flatness is to minimise the norm $\|\mathbf{w}\|^2 = \langle \mathbf{w}, \mathbf{w} \rangle$. This problem can be written as a quadratic optimisation problem:

$$\begin{aligned} &\text{minimise} && \tfrac{1}{2}\|\mathbf{w}\|^2, \\ &\text{subject to} && \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \varepsilon, \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i \varepsilon, \end{cases} \end{aligned} \tag{3}$$

for $i = 1, \ldots, l$.

It is assumed in (3) that the function f as in (2) exists and that it can approximate the training data pairs with accuracy ε (Vapnik et al., 1996). If this assumption is correct, the optimisation problem in (3) can be solved. Sometimes this may be impossible, or in other cases some extra error could be allowed in order to improve generalisation. In that situation, analogously to the "soft margin" loss function in Cortes and Vapnik (1995), the slack variables ξ_i and ξ_i^* can be introduced (Drucker et al., 1996; Vapnik et al., 1996; Smola & Schölkopf, 2004) (Fig. 1) and described as:

$$\begin{aligned} &\text{minimise} && \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*), \\ &\text{subject to} && \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b & \varepsilon + \xi_i, \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y & \varepsilon + \xi_i^*, \\ \xi_i, \xi_i^* & 0, \end{cases} \end{aligned} \tag{4}$$

for $i = 1, \ldots, l$.

The constant C in (4) determines the balance between the flatness of f and the amount of *tolerated* deviations larger than ε. In other words, the constant C can be considered as a tuning parameter between model complexity and required training data set model accuracy.

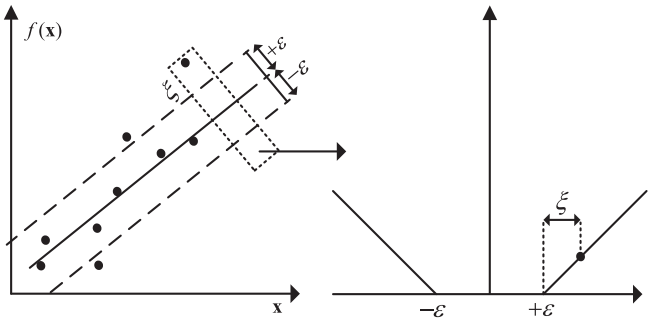

Fig. 1. The soft margin loss setting for a linear SVM.

The optimisation problem in (4) can be solved more easily in its dual formulation (Smola & Schölkopf, 2004):

$$\text{maximise} \quad \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i-\alpha_i^*)\left(\alpha_j-\alpha_j^*\right)\langle \mathbf{x}_i,\mathbf{x}_j\rangle, \\ -\varepsilon\sum_{i=1}^{l}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{l}y_i(\alpha_i-\alpha_i^*) \end{cases}, \tag{5}$$

$$\text{subject to} \quad \textstyle\sum_{i=1}^{l}(\alpha_i-\alpha_i^*)=0 \text{ and } \alpha_i,\alpha_i^*\in[0,C],$$

where α_i, α_i^* are Lagrangian multipliers with the following constraint:

$$\alpha_i,\alpha_i^* \geqslant 0. \tag{6}$$

From (5) it can be seen that the training complexity of a support vector machine is dependent on the number of α_i, α_i^* coefficients l which is the number of training data samples.

The support vector (SV) expansion (Smola & Schölkopf, 2004) can be written as

$$\mathbf{w}=\sum_{i=1}^{l}(\alpha_i+\alpha_i^*)\mathbf{x}_i \tag{7}$$

and therefore the regression function

$$f(\mathbf{x})=\sum_{i=1}^{l}(\alpha_i+\alpha_i^*)\langle \mathbf{x}_i,\mathbf{x}\rangle+b, \tag{8}$$

where $\mathbf{w}$ is the weight vector that can completely be described as a linear combination of the training patterns $\mathbf{x}_i$. Note that even when evaluating $f(\mathbf{x})$, no explicit computation of $\mathbf{w}$ is necessary. The $\mathbf{x}_i$ vectors corresponding to non-zero $(\alpha_i+\alpha_i^*)$ resulting from the optimisation of (5) are called the support vectors. From (8) it can be seen that the evaluation complexity of a support vector machine is dependent on the number of support vectors.

The computation of b in (8) can be accomplished by exploiting the Karush–Kuhn–Tucker (KKT) conditions (Karush et al., 1939; Kuhn et al., 1951). Further considerations about choosing the optimal parameter b can be found in Keerthi, Shevade, Bhattacharyya, and Murthy (2001) and Smola and Schölkopf (2004).

Up to now, we have discussed the linear regression case. The next step towards general function approximation is to make the SVM nonlinear. This can be achieved by simply preprocessing the training patterns $\mathbf{x}_i$ using the map function $\Phi: X \to F$. The function Φ transforms the input space into some feature space F where it is possible to apply the standard SV regression algorithm (Smola & Schölkopf, 2004). Unfortunately, the evaluation of the function Φ can be computationally expensive.

Boser, Guyon, and Vapnik (1992) proved that if we can find a new function that satisfies $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x})\rangle$ in the feature space F, it is possible to avoid the computationally expensive calculation of the function Φ. This can be done by using so called kernel functions.

A kernel function on two vectors $\mathbf{v}$ and $\mathbf{z}$ is the function $K: X \to \mathbb{R}$ that satisfies:

$$K(\mathbf{v},\mathbf{z})=\langle \Phi(\mathbf{v}),\Phi(\mathbf{z})\rangle. \tag{9}$$

In this paper, the radial basis function (RBF) is used as a kernel function:

$$\begin{aligned} K(\mathbf{v},\mathbf{z}) &= \exp(-\gamma\|\mathbf{v}-\mathbf{z}\|^2) \\ &= \exp(-\gamma(\langle \mathbf{v},\mathbf{v}\rangle+\langle \mathbf{z},\mathbf{z}\rangle-2\langle \mathbf{v},\mathbf{z}\rangle)), \quad \text{for } \gamma>0. \end{aligned} \tag{10}$$

Other kernel functions can be used with the proposed methods too.

For the ε-tube support vector regression machine, as in Fig. 1, two parameters are the most critical:

- the regularisation parameter C;
- the width of the ε-tube, i.e. the parameter ε.

Kernel functions might require the selection of additional parameters, e.g. γ in Eq. (10). The choice of the SVM parameters directly influences the sparsity of the model, e.g. if ε is set too small, the set of support vectors might include the whole training data set. The SVM parameters and the additional kernel parameters are together often referred to as the hyper-parameters, also in this paper.

The standard approach to find a (near) optimal C and ε is to perform a grid search (GS), as suggested in Hsu, Chang, and Lin (2003). It should be noted that in Hsu et al. (2003) the grid search for optimal parameters is performed for classification problems. This grid search approach was adopted by many researchers and used also for SVM regression, e.g. (Bao, Lu, & Zhang, 2004; Ceperic et al., 2004). Hsu et al. (2003) suggested exponentially growing sequences of SVM parameters as a practical method to identify optimal parameters. The problem with the grid search strategy is that it is very time consuming (Bao et al., 2004).

In the last few years, many researchers used the particle swarm optimisation (PSO) method to optimise the hyper-parameters of the SVM, e.g. (Cao & Xu, 2007; Fei, Liu, Zeng, & Miao, 2008; Jiang et al., 2007; Liu, Zhuang, & Liu, 2011). Particle swarm optimisation is an optimisation algorithm based on principles of swarm intelligence that finds a solution to an optimisation problem in the search space based on mimicking the social behaviour of swarms. The use of PSO enables faster identification of near-optimal hyper-parameters (Cao & Xu, 2007; Fei et al., 2008; Jiang et al., 2007; Liu et al., 2011) compared to convectional methods such as e.g. the grid search method.

The ε-SVR implementation used in this paper is based on the LIBSVM tool (Chang et al., 2001), while a variant of PSO from the toolbox PSwarmM (Vaz & Vicente, 2007) is used for the hyper-parameters optimisation.

3. Sparse recurrent support vector regression machines by using an active learning principle in the time-domain

The goal of this paper is to stay within the definition of ε-SVR, while adapting the formulation to:

- models with feedback connections, i.e. recurrent models;
- modelling in the time-domain;
- reducing the model complexity by limiting the maximum number of support vectors;
- improving the accuracy while keeping the complexity low.

The support vectors in the SVM function approximation are chosen from the set of training data points that is iteratively constructed by active learning.

The key idea behind active learning, as described in Settles et al. (2009), states that "a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns". In this paper, the active learning principle is extended to the time-domain and embedded in an algorithm named TASVR (Time-Domain Active Learning Support Vector Regression). The next point to include in the training data set is selected in the limited "future" time horizon from the current point in time, as shown in Fig. 2. The length of the time horizon is limited primarily by the computational cost introduced by each additional point in the time horizon. The TASVR algorithm is summarised in Algorithm 1 and presented in the flow chart in Fig. 3. The numbers in the flow chart correspond to the steps in Algorithm 1.

One of the key questions in the practical application of the active learning principle for support vector machines is how to select the optimal SVM hyper-parameters. The optimal choice of the SVM hyper-parameters is critical because an active learning principle

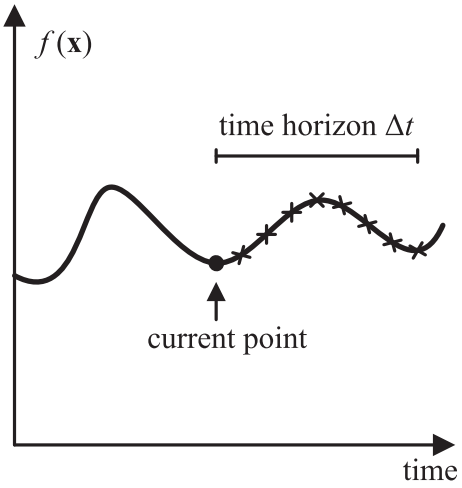

Fig. 2. Concept of active learning in the time-domain (TASVR).

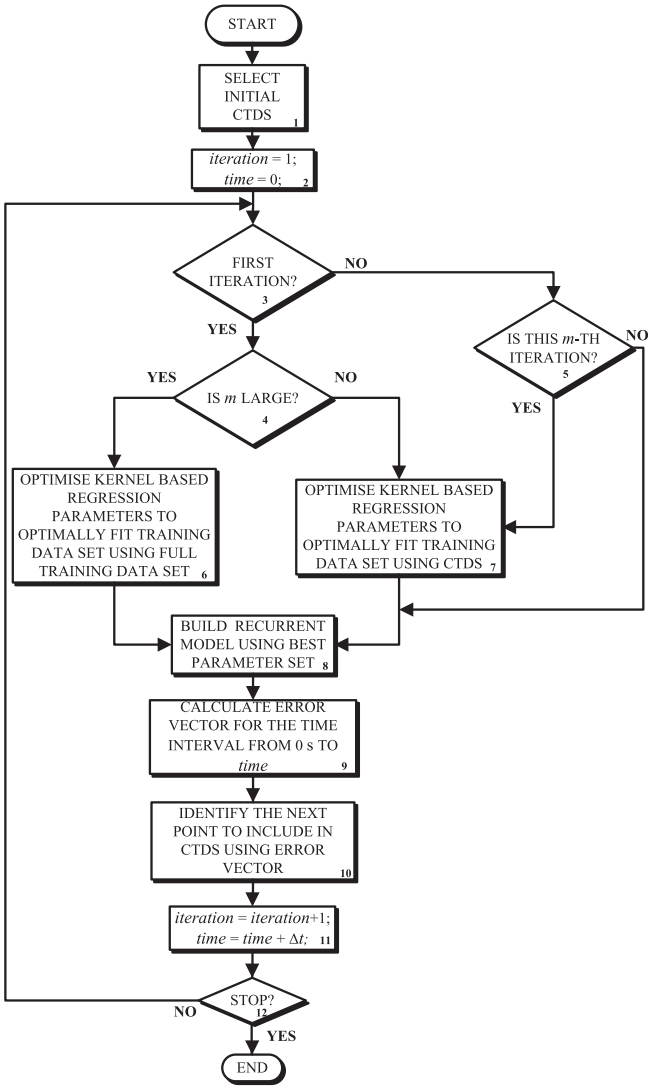

Fig. 3. TASVR training flow chart for the recurrent model.

relies on the tuned regressor in each iteration. In TASVR, the user can select how often the hyper-parameters should be optimised and can thus directly influence the overall training time and model accuracy. The selection of the optimal SVR hyper-parameters is embedded into the training algorithm: TASVR optimises the ε-SVR hyper-parameters in the first and then every mth iteration where m is a positive integer number.

Algorithm 1. Training algorithm for Time-Domain Active Learning Support Vector Regression (TASVR)

Step 1: Start with a limited number of data points in the current training data set (CTDS) (i.e. take the first input–output training data pair).
Step 2: Set the variable *iteration* to 1 and variable *time* to 0.
Step 3: Check if this is the first iteration. If yes, then go to Step 4, else go to Step 5.
Step 4: Check if m is large. If yes, go to Step 6, else go to Step 7.
Step 5: Check if this is the mth iteration. If yes, go to Step 7, else go to Step 8.
Step 6: Determine the optimal hyper-parameters on the full training data set using a series–parallel configuration and cross-validation or insert a set of hyper-parameters that is known from previous experiments to give reliable results. Go to Step 8.
Step 7: Thoroughly optimise the hyper-parameters to fit the training data set with (e.g. with PSO, as described in Section 2).
Step 8: Build the model using the best parameter set determined in Step 6 or Step 7.
Step 9: Calculate the error vector for the time interval from 0 to *time*.
Step 10: Identify the next point to include in the CTDS using the error vector.
Step 11: Increase the value of the variable *iteration* by 1. Increase the value of the variable *time* by a predetermined interval Δt. The Δt is defined by the time horizon, as shown in Fig. 2.
Step 12: If the stopping criterion is not met, go to Step 3.

The two extreme variants of the TASVR algorithm w.r.t. the hyper-parameters optimisation are named IDH-TASVR (Initially Determined Hyper-parameters for Time-Domain Active Learning Support Vector Regression) and OH-TASVR (Optimised Hyper-parameters for Time-Domain Active Learning Support Vector Regression). The main difference between these variants is how the model hyper-parameters are determined.

OH-TASVR is a sparse ε-SVR based algorithm for the training and construction of recurrent models where the active learning principle is used with the optimisation of the SVM hyper-parameters in each iteration of the algorithm, i.e. $m = 1$. This can be very time consuming if the necessary number of support vectors is large. On the other hand, the iterative optimisation of the hyper-parameters ensures a near-optimal model for the current training data set.

IDH-TASVR is a sparse ε-SVR based algorithm for the training and construction of recurrent models where the hyper-parameters are set in the first step and later kept fixed in the subsequent training steps, i.e. $m = \infty$. It should be noted that for large m (e.g. $\geqslant 100$), the quality of the initially chosen hyper-parameters can significantly deteriorate when the training process proceeds, if the hyper-parameters are determined on the limited training data set. In such cases, it is better to select the hyper-parameters based on the full training data set using series–parallel configuration either by cross-validation, experience or a trial-and-error procedure in the first iteration of the algorithm, as shown in Step 6 of the algorithm.

IDH-TASVR is more suitable when the necessary number of SVs for the required model accuracy is significant or when a fast training time is more important than a small loss in accuracy. In this case the optimisation of the hyper-parameters is performed only once, in Step 6 of the algorithm. The same hyper-parameters are then used in each iteration of the training algorithm.

The training data set is initially set to be very small for both of the proposed methods. This training data set is referred to as the current training data set (CTDS) to avoid confusion with the overall training data set. In Step 1, the initial CTDS is constructed iteratively from the first input–output training data pair, related to *time* = 0. Each input–output training data pair is referred onwards as a "point". The training data pairs should be sequential and can be non-equidistant in time.

The crucial step in TASVR is the addition of a new point to the CTDS in each iteration. The next point to be included in the CTDS is determined by the model performance on the training data set sequence from the first point in the data set to the last point of the time horizon. The model performance evaluation is limited to the end of the time horizon because the model is built only based on the information prior to the last point in the time horizon. The next point to be included in the training data set is selected in the limited time horizon, Δt forward from the current point in time. The point in this horizon which contributes the most to the model performance is included in the CTDS. In the experiments reported in Section 4, the time horizon is limited to 30 points, i.e. Δt is the time that corresponds to the time span of 30 points.

In each iteration, the model is built using a series–parallel configuration (Narendra & Parthasarathy, 1990),[1] but it is evaluated and tested in each iteration in fully recurrent (also called parallel) configuration. This simplification allows that standard SVM tools (e.g. LIBSVM) can be used for TASVR. An additional advantage is that the user does not have to change the code of the SVM tool, i.e. the SVM tool can be used to formulate ε-SVR solutions, while the iterative construction of the training data set and the recurrent model evaluation can be done in almost any programming language or environment, e.g. MATLAB.

The stopping criterion to end the iterative procedure in the TASVR algorithm (Step 12) is set in the following way:

- the model has reached the user-specified complexity limit, i.e. the maximum number of support vectors is reached; or
- the model has reached the user-specified desired accuracy (defined in this paper as the mean square error (MSE)).

Another important factor related to all machine learning algorithms is the training time and memory space requirement. A common quadratic programming optimisation problem (as found in Eq. (5)) has a training time and memory space requirement proportional to $O(l^3)$ and $O(l^2)$ (Guo et al., 2010), respectively, where l is the total number of variables. In this case l depends on the total number of training data points. Several algorithms exist, e.g. sequential minimal optimisation (SMO) (Platt et al., 1999), that reduce the training time requirement of the SVM problem. SMO has a typical training time requirement between $O(l)$ and $O(l^{2.3})$. The training time requirement is further reduced in LIBSVM, as described in Fan, Chen, and Lin (2005). If the following assumptions are made:

- the training time requirement of the algorithm used to solve the SVM problem is $O(l)$;
- it is presumed that one training point is added to the CTDS at each iteration based on solving one SVM problem per point in the time horizon;
- the hyper-parameters are determined initially and kept fixed in the training algorithm, i.e. $m = \infty$;
- there are k points in the time horizon;

then the training time requirement of the time-domain active learning approach to recurrent support vector regression is:

$$k \cdot O(1) + k \cdot O(2) + \cdots + k \cdot O(T) = k \cdot O\left(\frac{T^2 + T}{2}\right), \tag{11}$$

where T is the final number of training data in the CTDS. It can be noticed that if T/l is small, i.e. when the final number of selected points is small compared to the total number of training data points, the active learning approach is time efficient which is in line with the focus of this paper to develop sparse solutions based on the least number of support vectors.

The training time requirement of OH-TASVR (where m is set to 1) is approximately o-times higher than IDH-TASVR, where o is the number of optimisation algorithm iterations needed to optimise the algorithm's hyper-parameters, since these are optimised within each iteration of the proposed algorithm.

4. Modelling experiments

In this section, several experiments are presented to examine the sparsity and overall performance of the TASVR training algorithm in its two variants, IDH-TASVR and OH-TASVR.

Two sets of experiments on the following well known recurrent modelling problems are performed:

- the multiple superimposed oscillations task (Holzmann & Hauser, 2010; Schmidhuber et al., 2007; Xue et al., 2007);
- the 10th-order nonlinear autoregressive moving average (NARMA) system benchmark task (Atiya & Parlos, 2000).

The performance on these two sets of experiments is measured by the model accuracy and its complexity. The accuracy is described as the error of the model and the complexity is directly related to the number of support vectors in the model. One should note that forcing (any) algorithm to a specific complexity can deteriorate its performance and accuracy.

4.1. Multiple superimposed oscillations

In this section we test and analyse the performance of the IDH-TASVR and OH-TASVR algorithms on the multiple superimposed oscillations (MSO) task. This task is often studied in echo state networks (ESN) literature, e.g. (Holzmann & Hauser, 2010). As mentioned in Schmidhuber et al. (2007), Xue et al. (2007), Holzmann and Hauser (2010), standard echo state networks are unable to model functions composed of only two superimposed oscillators. Holzmann and Hauser (Holzmann & Hauser, 2010) notice that "the problem is that the dynamics of all reservoir neurons are coupled, while this task requires that different oscillations can be represented simultaneously by the internal state vector". Several authors addressed this problem by introducing rather complex models, e.g. EVOLINO (Schmidhuber et al., 2007), echo state networks with filter neurons and delay and sum readout (Holzmann & Hauser, 2010). As they are not sparse, these models might not be applicable to real-time and embedded applications.

The objective in this experiment is to model the multiple superimposed oscillations while retaining low complexity.

[1] In the training of series–parallel architectures, the true output is used instead of feeding back the estimated output.

The following multiple signals are investigated (as used in Holzmann & Hauser (2010), Xue et al. (2007) and others):

$$y_2 = \sin(0.2n) + \sin(0.311n), \tag{12}$$

$$y_3 = y_2 + \sin(0.42n), \tag{13}$$

$$y_4 = y_3 + \sin(0.51n), \tag{14}$$

$$y_5 = y_4 + \sin(0.74n), \tag{15}$$

where $n = 1,\ldots,700$.

The size of the training data set is set to 400, while the testing data set has 300 points. The maximally allowed complexity in this experiment is set to 15 support vectors. This is done by limiting the maximum number of points in the CTDS to 15. In this way the resulting model will have 15 or less support vectors, depending on the selection made by the SVM algorithm and the selected hyper-parameters.

The dimension of the input space is chosen to be 50 delayed output signals in the recurrent configuration. It is necessary to include these delayed outputs, which represent feedback, in order to capture the complex behaviour of higher-order multi-sine signals and their long periods. Indeed, superimposed sine waves with frequencies that are not integer multiplies of each other can have extremely long periods. The used number of 50 delayed outputs is just a small fraction of the signal period. Further increase in the number of delayed output signals could additionally increase accuracy.

Table 1 compares the OH-TASVR and IDH-TASVR mean square error performance on the test data set while Fig. 4 shows the absolute value of the model error per testing data sample for the above MSO data set. Both algorithms have a high accuracy (unlike the standard echo state networks) while retaining a low model complexity. The OH-TASVR algorithm produces more accurate models than IDH-TASVR, because the SVM hyper-parameters are tuned in each iteration of the algorithm and therefore the active learning process is more efficient.

In Xue et al. (2007) the two sines problem (y_2) was approximated with "decoupled echo state network with lateral inhibition", consisting of four reservoirs with 100 neurons each, i.e. the complexity is far greater than 9 and 13 SVs and a feedback with maximum 50 points as used here (Table 1). They trained the network after a "washout" period of 100 steps (to fill the states in the reservoirs due to the feedback), for 600 time steps, and the test error was calculated from the 701th to the 1000th time step. A mean square error of $3 \cdot 10^{-4}$ was obtained on the y_2 test data set, which is at least three orders or magnitude worse than the results reported in Table 1.

Schmidhuber et al. (2007) use PI-EVOLINO and EVOKE networks to model the MSO data set. The PI-EVOLINO network consisted of 30 LSTM memory cells and the size of the subpopulation was 40 for the two-, three- and four-sine problem, and 100 for the five-sine problem. They achieved the following results on the test data sets:

Table 1
Mean square error (MSE) for the multiple superimposed oscillations test data set.

	IDH-TASVR			OH-TASVR		
	MSE	NRMSE	SV	MSE	NRMSE	SV
y_2	2.03E−7	4.49E−4	9	3.57E−8	1.88E−4	13
y_3	3.25E−6	1.46E−3	12	2.33E−6	1.23E−3	15
y_4	6.99E−5	5.72E−3	14	1.75E−5	2.86E−3	15
y_5	8.41E−4	1.80E−2	15	9.16E−5	5.94E−3	15

- y_2 multi-sine: NRMSE = 4.15E−3 (compared to IDH-TASVR 4.49E−4 and OH-TASVR 1.88E−4);
- y_3 multi-sine: NRMSE = 8.04E−3 (compared to IDH-TASVR 1.46E−3 and OH-TASVR 1.23E−3);
- y_4 multi-sine: NRMSE = 1.01E−1 (compared to IDH-TASVR 5.72E−3 and OH-TASVR 2.86E−3);
- y_5 multi-sine: NRMSE = 1.66E−1 (compared to IDH-TASVR 1.80E−2 and OH-TASVR 5.94E−3).

Schmidhuber et al. note that EVOKE (the support vector machines based approach they use) achieved relatively low generalisation results on the double-sines problem (NRMSE of 1.03E−2), and gave unsatisfactory results for three or more sines. The results reported in Table 1 are considerably better despite the significantly reduced complexity of the proposed models.

Recently, Holzmann and Hauser (2010) proved that it is possible to obtain a better accuracy on the MSO data set, but at the expense of considerably increasing the model complexity (by introducing infinite impulse response (IIR) filter neurons that need to be "tuned" to different frequencies, additional delay and sum readouts) which is in contrast with the objective to build a sparse model characterised by a low complexity. It should be noted that one could introduce additional support vectors in the models proposed in this paper and therefore increase the accuracy; however, the sparsity of the model would be compromised in this case.

Fig. 5 shows the accuracy versus complexity plot of the IDH-TASVR and OH-TASVR training algorithms for the most complex MSO test case: the modelling of y_5. Accuracy is defined as the MSE and complexity is defined as the number of SVs in the model. Fig. 5 shows that the MSE decreases rapidly after the first 9 SVs in the model. As expected, OH-TASVR exhibits a higher accuracy than IDH-TASVR. It is interesting to note that the MSE of the OH-TASVR algorithm for 17 SVs is slightly higher than for 16 SVs. This could be the result of the hyper-parameters optimisation that managed to find a "better" optimum w.r.t the test data set for the model with 16 SVs than for 17 SVs.

4.2. 10th-order nonlinear autoregressive moving average system benchmark task

Atiya and Parlos (2000) note that "it is well known that for problems with long-term time dependencies recurrent networks are very hard to train" and therefore they consider the following tenth-order system as a benchmark for recurrent networks:

$$y(n) = 0.3y(n-1) + 0.05y(n-1)\left[\sum_{i=1}^{10} y(n-i)\right] + 1.5u(n-10)u(n-1) + 0.1, \tag{16}$$

where $u(n)$ is independent uniform noise (uniform in [0,0.5]).

After the introduction in Atiya and Parlos (2000), the 10th-order nonlinear autoregressive moving average (NARMA) system benchmark was analysed in various papers about echo state networks, for instance in Holzmann and Hauser (2010), Anon. (2002).

In Eq. (16) the term $u(n)$ and the uncertainty of its random part might lead to misleading conclusions and an inaccurate comparison of the results. Therefore, it is decided to test the algorithms on clearly defined data sets that are easily reproducible. The data generation in this experiment is based on pseudo-random number generation with the following setup:

- the 64-bit multiplicative lagged Fibonacci generator (Mascagni & Srinivasan, 2004) is used as pseudo-random number generating algorithm;

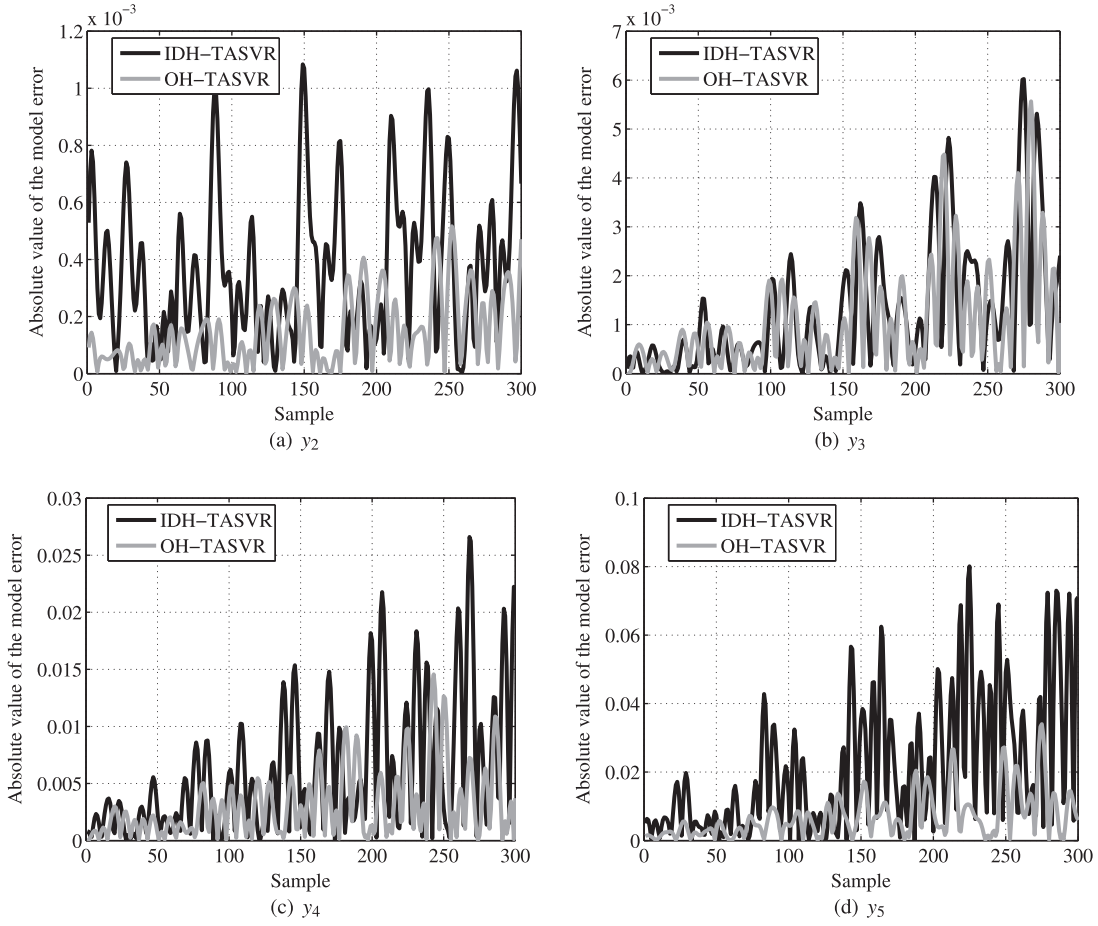

Fig. 4. Absolute value of the model error per testing data sample for the MSO data set.

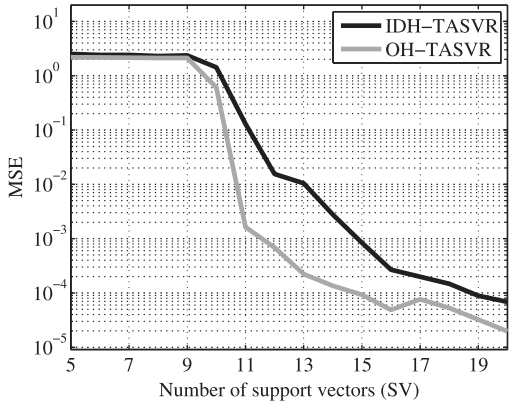

Fig. 5. Accuracy vs. complexity plot of the IDH-TASVR and OH-TASVR training algorithms for the y_5 MSO data set.

- the Ziggurat algorithm (Marsaglia & Tsang, 2000) is used to generate normal pseudo-random values from a number stream generated by the MT algorithm;
- the Matlab function *rand* is used as a generator of pseudorandom numbers from a uniform distribution;
- the output of the Matlab function *rand* is scaled to [0,0.5];
- the data set generation procedure is repeated 10 times, i.e. 10 different experiments are generated;
- the seed values at the beginning of each data set generation are equal to $seed = 1000 \cdot p$, where p is the number of the experiment (in this case $p = 1 \ldots 10$);
- $y(n)$ for $n = 1 \ldots 10$ is set to zero. The calculation of $y(n)$ starts for $n = 11$;
- the first 2500 points of the generated time series are neglected. In this way, the disrupting effect of initially setting the first 10 $y(n)$ to zero is negligible.

The inputs are constructed from:

- $u(n)$ which is the independent uniform noise (uniform in [0,0.5]);
- 10 delayed output signals constructing a feedback and corresponding to the 10th-order problem.

The size of the training data set is set to 400, while the testing data set has 2000 points. The maximally allowed complexity in this experiment is set to 20 SVs. As in the case of modelling the MSO data set, this is done by limiting the maximum number of points in the CTDS.

Table 2 compares the OH-TASVR and IDH-TASVR normalised root mean square error (NRMSE) performance on the 10th-order NARMA benchmark test data set. Fig. 6 shows the modelled values and the model error for the first 300 testing data samples in the experiment 1 (*seed* = 1000). Both algorithms succeed in modelling the 10th-order NARMA data set while retaining a low model complexity. As in the case of modelling the MSO data set, the OH-TASVR algorithm produced more accurate models than IDH-TASVR, because the SVM hyper-parameters are tuned in each iteration of the algorithm and therefore, the active learning process is more efficient.

Table 2
$NRMSE_{test}$ for the 10th-order NARMA system benchmark test data set.

	$NRMSE_{test}$	
Experiment	OH-TASVR	IDH-TASVR
1	0.189	0.285
2	0.213	0.345
3	0.267	0.324
4	0.154	0.292
5	0.203	0.290
6	0.226	0.326
7	0.289	0.254
8	0.265	0.278
9	0.159	0.266
10	0.143	0.244
$\overline{NRMSE}_{test}$	0.211	0.291

In Atiya and Parlos (2000), a recurrent neural network with 40 hidden nodes is considered for the modelling of the 10th-order NARMA system benchmark data set. According to Anon. (2002), the normalised mean square error on the training data set ($NMSE_{train}$) in Atiya and Parlos (2000) is miscalculated and should be equal to 0.241, which also agrees with the plots given in Atiya and Parlos (2000); $NMSE_{train} = 0.241$ corresponds to $NRMSE_{train} \approx 0.49$. When compared to Atiya and Parlos (2000), the results presented in Table 2.

- Are more accurate than reported in Atiya and Parlos (2000) despite tha fact that they are reported on the test data set (in contrast to Atiya & Parlos (2000) where the results are reported using the training data set);
- have considerably lower complexity (the maximum number of support vectors is just 20, while in Atiya & Parlos (2000) a network with 40 hidden nodes is used).

Anon. (2002) reports that $NRMSE_{test}$ for ESN is approximately equal to 0.556 ($NMSE_{test} \approx 0.31$) on 2000 samples, trained on 500 samples using 20-units (a complexity comparable to the 20 SVs in the models presented here). The results reported in Table 2, also using 20 SVs are considerably better.

Fig. 7 shows the accuracy versus complexity plot of the IDH-TASVR and OH-TASVR training algorithm for the first experiment in this set (*seed* = 1000) of the 10th-order NARMA system benchmark data set. The NRMSE value decreases rapidly after the first 7 SVs for the IDH-TASVR trained model. The increase of accuracy per SV is more consistent for OH-TASVR. It is interesting to notice that for this experiment, the accuracy of both algorithms is similar

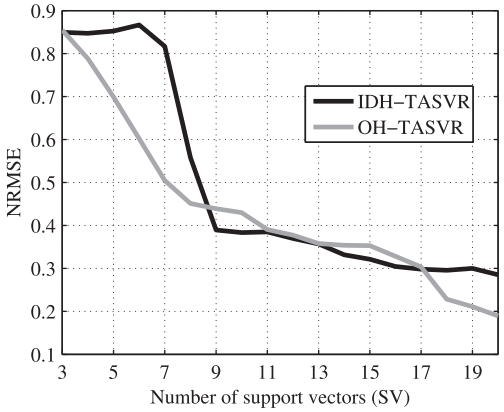

Fig. 7. Accuracy vs. complexity plot of the IDH-TASVR and OH-TASVR training algorithm for the 10th-order NARMA system benchmark data set.

for the models with 9–17 SVs, while the difference grows again for more SVs.

As in the case of the MSO data set, one could add SVs to the models proposed in this paper and therefore further increase the accuracy; but in doing so, one would lose the sparsity of the model.

Furthermore, it is worth noting that the performance of the models varies with each experiment, i.e. with each generation of a training and test data set, as seen on Table 2. This reaffirms the decision to clearly define the data set generation procedure rather than picking an arbitrary set or sets. This step allows a simple and accurate comparison with the results reported in this paper.

5. Conclusion

A training algorithm has been presented for the sparse solution of recurrent support vector regression machines by applying the active learning principle in the time-domain (TASVR). The proposed algorithm achieves a high accuracy while keeping the complexity low, i.e. the resulting models have a low number of support vectors, which makes the models efficient in execution. The maximum number of support vectors can be defined by the user. The optimisation procedure for the algorithms's hyper-parameters is built into the training algorithm. The user selects how often the hyper-parameters are optimised and thus directly influences the overall training time and model accuracy. The two extreme variants of the TASVR algorithm w.r.t. the hyper-parameter optimisation are investigated in detail: the OH-TASVR algorithm (where the hyper-parameters are optimised in each iteration) and the

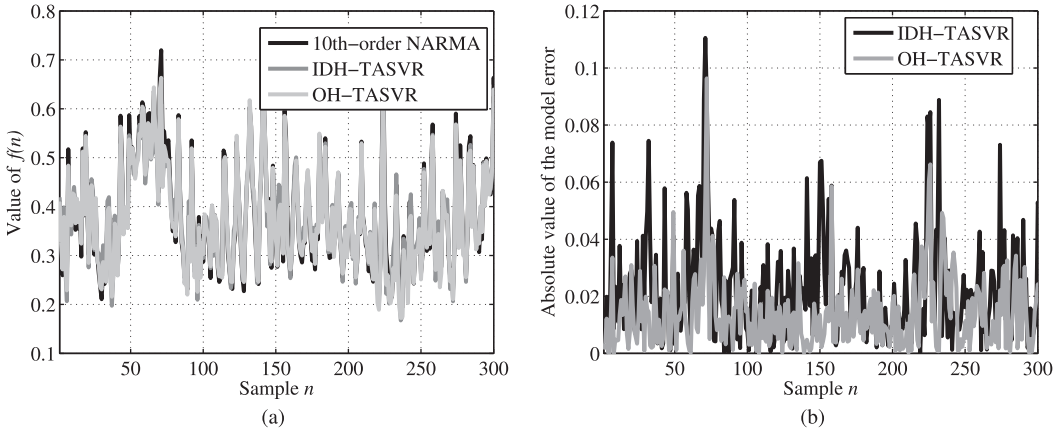

Fig. 6. Results for the 10th-order NARMA system benchmark data set, experiment 1 (*seed* = 1000): (a) modelled values for the first 300 testing data samples, (b) absolute value of the model error for the first 300 testing data samples.

IDH-TASVR algorithm (where initially optimised hyper-parameters are kept fixed in later iterations). The iterative optimisation of the hyper-parameters in the OH-TASVR variant ensures a near-optimal model for the current, iteratively constructed training data set. This results in a better accuracy versus complexity, but at the expense of extra CPU time needed to perform the hyper-parameter optimisation at each iteration of the algorithm. In the IDH-TASVR variant, the hyper-parameters are fixed in the first iteration and thus the IDH-TASVR algorithm has a faster training time than the OH-TASVR variant. Instead of performing the hyper-parameters optimisation only once (IDH-TASVR) or in every step (OH-TASVR), it is possible to do the optimisation in every mth iteration, thus reducing the CPU load, however at the expense of reduced accuracy.

The advantages of the algorithm and its variants have been illustrated on several well-known benchmark examples. The results show that it is possible to reduce the number of support vectors and significantly improve the accuracy versus complexity of recurrent ε-SVR machines.

Acknowledgements

The authors acknowledge the support of the IWT GoldenGates Project and ON Semiconductor, Belgium.

Vladimir Ceperic acknowledges and thanks for the financial support of the National Foundation for Science, Higher Education and Technological Development of Croatia within the programme "Scholarships for Doctoral Students".

References

Anon., 2002. Adaptive Nonlinear System Identification with Echo State Networks, MIT Press, Cambridge, MA.

Atiya, A. F., & Parlos, A. G. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks, 11*, 697–709.

Bao, Y., Lu, Y., & Zhang, J. (2004). Forecasting stock price by SVMs regression. In *Artificial Intelligence: Methodology, Systems, and Applications, Lecture Notes in Computer Science, Springer Berlin/ Heidelberg,* (pp. 295–303).

Ben-Hur, A., Ong, C. S., Sonnenburg, S., Schölkopf, B., & Rätsch, G. (2008). Support Vector Machines and Kernels for Computational Biology. *PLoS Computational Biology, 4*(10).

Boser, B. E., Guyon, I. M., & Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual ACM workshop on computational learning theory, ACM, New York, NY, USA,* (pp. 144–152).

Brabanter, K. D., Brabanter, J. D., Suykens, J., & Moor, B. D. (2010). Optimized fixed-size kernel models for large data sets. *Computational Statistics & Data Analysis, 54*(6), 1484–1504.

Cao, C., & Xu, J. (2007). Short-Term Traffic Flow Predication Based on PSO-SVM. In Q. Peng, K. C. P. Wang, Y. Qiu, Y. Pu, X. Luo, & B. Shuai (Eds.). *Proceedings of the First International Conference on Transportation Engineering* (Vol. 246, pp. 28). Chengdu, China: American Society of Civil Engineers (ASCE).

Ceperic, V., & Baric, A. (2004). Modeling of analog circuits by using support vector regression machines. In *Proceedings of the 11th IEEE international conference on electronics, circuits and systems, ICECS 2004,* (pp. 391–394).

Chang, C. -C., & Lin, C. -J. (2001). LIBSVM: A library for support vector machines, software available at <http://www.csie.ntu.edu.tw/cjlin/libsvm>.

Chang, C. -Y., Wang, H. -J., & Li, C. -F. (2009). Semantic analysis of real-world images using support vector machine. *Expert Systems with Applications, 36*(7), 10560–10569.

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning, 20*(3), 273–297.

Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A. J., & Vapnik, V. (1996). Support Vector Regression Machines. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems* 9, (pp. 155–161).

Fan, R.-E., Chen, P.-H., & Lin, C.-J. (2005). Working Set Selection Using Second Order Information for Training Support Vector Machines. *Journal of Machine Learning Research, 6*, 1889–1918.

Fei, S., Liu, C., Zeng, Q., & Miao, Y. (2008). Application of Particle Swarm Optimization-Based Support Vector Machine in Fault Diagnosis of Turbo-Generator. In *IITA '08: Proceedings of the 2008 second international symposium on intelligent information technology application, IEEE Computer Society, Washington, DC, USA,* (pp. 1040–1044).

Figueiredo, M. A. T. (2003). Adaptive sparseness for supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25*(9), 1150–1159.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks, 18*(5-6), 602–610. iJCNN 2005.

Guo, B., Gunn, S., Damper, R., & Nelson, J. (2008). Customizing kernel functions for SVM-based hyperspectral image classification. *IEEE Transactions on Image Processing, 17*(4), 622–629.

Guo, G., Zhang, J.-S., & Zhang, G.-Y. (2010). A method to sparsify the solution of support vector regression. *Neural Computing & Applications, 19*(1), 115–122.

Hao, P. -Y., Chiang, J. -H., & Tu, Y. -K. (2007). Hierarchically SVM classification based on support vector clustering method and its application to document categorization. *Expert Systems with Applications, 33*(3), 627–635.

Härdle, W. K., Moro, R., & Hoffmann, L. (2010). Learning Machines Supporting Bankruptcy Prediction, SFB 649 Discussion Paper 2010-032, Sonderforschungsbereich 649, Humboldt Universität zu Berlin, Germany.

Holzmann, G., & Hauser, H. (2010). Echo state networks with filter neurons and a delay & sum readout. *Neural Networks, 23*(2), 244–256.

Hong, W., & Lai, C. (2008). Reliability forecasting by recurrent Support Vector Regression. *International Journal of Artificial Intelligence and Soft Computing, 1*, 114–129.

Hsu, C. -W., Chang, C. -C., & Lin, C. -J. (2003). A Practical Guide to Support Vector Classification, Tech. Rep., Department of Computer Science and Information Engineering, National Taiwan University.

Jaeger, H., Lukosevicius, M., Popovici, D., & Siewert, U. (2007). Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks, 20*(3), 335–352. echo State Networks and Liquid State Machines.

Jiang, J., Ip, H. H. (2007). Dynamic Distance-Based Active Learning with SVM. In *MLDM '07: Proceedings of the 5th international conference on machine learning and data mining in pattern recognition, Springer-Verlag, Berlin, Heidelberg,* (pp. 296–309).

Jiang, M. -H., & Yuan, X. -C. (2007). Construction and Application of PSO-SVM Model for Personal Credit Scoring. In *ICCS '07: Proceedings of the 7th international conference on Computational Science, Part IV, Springer-Verlag, Berlin, Heidelberg,* (pp. 158–161).

Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints, Master's thesis, University of Chicago.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (2001). Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation, 13*(3), 637–649.

Kuhn, H. W., & Tucker, A. W. (1951). Nonlinear programming. In *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics, University of California Press, Berkeley*.

Kuo, T.-F., & Yajima, Y. (2010). Ranking and selecting terms for text categorization via SVM discriminate boundary. *International Journal of Intelligent Systems, 25*(2), 137–154.

Liu, L.-x., Zhuang, Y.-q., & Liu, X.-y. (2011). Tax forecasting theory and model based on SVM optimized by PSO. *Expert Systems with Applications, 38*(1), 116–120.

Marsaglia, G., & Tsang, W. W. (2000). The ziggurat method for generating random variables. *Journal of Statistical Software, 5*(8), 1–7.

Mascagni, M., & Srinivasan, A. (2004). Parameterizing parallel multiplicative lagged-Fibonacci generators. *Parallel Computing, 30*(7), 899–916.

Mirikitani, D., & Nikolaev, N. (2010). Recursive bayesian recurrent neural networks for time-series modeling. *IEEE Transactions on Neural Networks, 21*(2), 262–274.

Narendra, K., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks, 1*(1), 4–27.

Neumaier, A. (1998). Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM Review, 40*(3), 636–666.

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: Support vector learning, MIT Press, Cambridge, MA, USA,* (pp. 185–208).

Qu, H., Oussar, Y., Dreyfus, G., & Xu, W. (2009). Regularized Recurrent Least Squares Support Vector Machines. In *International joint conference on bioinformatics, systems biology and intelligent computing,* 2009. IJCBS '09, (pp. 508–511).

Roth, V. (2004). The generalized LASSO. *IEEE Transactions on Neural Networks, 15*(1), 16–28.

Schmidhuber, J., Gagliolo, M., Wierstra, D., & Gomez, F. (2006). Evolino for Recurrent Support Vector Machines. In M. Verleysen (Ed.), *ESANN '06 – 14 th European Symposium on Artificial Neural Networks* (pp. 593–598). Belgium: Evere.

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007). Training recurrent networks by evolino. *Neural Computation, 19*, 757–779.

Schölkopf, B., & Smola, A. (2001). *Learning with kernels: Support vector machines, regularization, optimization, and beyond (adaptive computation and machine learning)*. Cambridge, MA, USA: The MIT Press.

Schölkopf, B., Smola, A. J., Williamson, R. C., & Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation, 12*(5), 1207–1245.

Settles, B. (2009). Active Learning Literature Survey, Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Shin, K. -S., Lee, T. S., & Kim, H. -J. (2005). An application of support vector machines in bankruptcy prediction model. *Expert Systems with Applications, 28*(1), 127–135.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing, 14*(3), 199–222.

Suykens, J. A. K. (2002). *Least squares support vector machines*. River Edge, NJ: World Scientific.

Suykens, J., & Vandewalle, J. (2000). Recurrent least squares support vector machines. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 47*(7), 1109–1114.

Tipping, M. E. (2001). Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Reserach, 1*, 211–244.

Tong, S., & Koller, D. (2002). Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research, 2*, 45–66.

Vapnik, V., Golowich, S. E., & Smola, A. (1996). Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing. In *Advances in neural information processing systems* 9, MIT Press, (pp. 281–287).

Vaz, A., & Vicente, L. (2007). A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization, 39*(2), 197–219.

Xue, Y., Yang, L., & Haykin, S. (2007). Decoupled echo state networks with lateral inhibition. *Neural Networks, 20*(3), 365–376.

Zhong, W., He, J., Harrison, R., Tai, P. C., & Pan, Y. (2007). Clustering support vector machines for protein local structure prediction. *Expert Systems with Applications, 32*(2), 518–526.