Analysis of Switching Dynamics with Competing Support Vector Machines

Ming-Wei Chang and Chih-Jen Lin
Department of Computer Science and
Information Engineering
National Taiwan University
Taipei 106, Taiwan (cjlin@csie.ntu.edu.tw)
Ruby C. Weng
Department of Statistics
National Chenechi University
Taipei 116, Taiwan

Abstract

We present a framework for the unsupervised segmentation of switching dynamics using support vector machines. Following the architecture by Pawelzik et al. [21] where annealed competing neural networks were used to segment a non-stationary time series, in this article we exploit the use of support vector machines, a well-known learning technique. First, a new formulation of support vector regression is proposed. Second, an expectation-maximization (EM) step is suggested to adaptively adjust the annealing parameter. Results indicate that the proposed approach is promising.

Index Terms

Annealing, competing experts, expectation maximization, support vector machines, unsupervised time series segmentation.

I. Introduction

Recently support vector machines (SVMs) [26] have been a promising method for data classification and regression. For an account of various applications of the method, see [15], [16], and references therein. However, its application to unsupervised learning problems has not been exploited much. In this paper we aim to apply it to unsupervised segmentation of time series. Practical applications of unsupervised segmentation of time series include, for example, speech recognition [23], signal classification [6], and brain data [20], [9].

The topic of unsupervised segmentation has been investigated by researchers in various fields. See for example, an early survey on various approaches in [18], the fuzzy c-regression models (FCRM) [5] with applications to models with known forms and known distribution of the noise term, a combination of supervised and unsupervised learning using hidden Markov models based on neural networks [4], and other competing neural networks approaches in [14], [21], [8], [11], [9].

In [21], [14], annealed competing neural networks were used to segment a non-stationary time series, where non-stationarities are caused by switching dynamics. This method is called "Annealed Competition of Experts" (ACE). Unlike the mixtures of experts architecture [7] which used an input-dependent gating-network, the ACE method drives the competition of experts by an evaluation of prediction performance so that the underlying dynamics can have overlapping input domains. Two main features of this approach are $memory$ derived from a slow switching rate and $deterministically\ annealed$ competition of the experts during training. The assumption of slow switching rate is imposed to resolve problems caused by overlapping input-output relations. The idea of annealing is to avoid getting stuck in local minima and resolve the underlying dynamics in a hierarchical manner. (The deterministic annealing method was described in the context of clustering [24].) The neural network used is a Radial Basis Function (RBF) network of the Moody-Darken type [13].

The present paper aims to solve the same unsupervised segmentation problem as in [21], [14]. We propose a framework using competing support vector machines (SVMs). The standard SVMs assume equal weights on all error terms, which mean that each data point is equally important. However, due to the switching nature of the problem, the data points actually come from different sources and therefore, the contribution of each data point to each predictor would not be the same. In order to solve this problem, we propose a modified formulation of SVMs which allows different weights on the error terms. The dual problem and the implementation for this formulation are also new. Here the weights are adjusted by relative prediction performance, as in [21], [14]. In addition to the new formulation, this paper is novel in presenting an adaptive annealing method. A key observation is that the annealing parameter characterizes some statistical property of the error terms. Therefore, at each iteration during training, we treat the annealing parameter as an unknown parameter and estimate it based on the current weighting coefficients and error terms. This estimate is essentially the maximum likelihood estimate, which can be obtained by an expectation-maximization (EM) step. We shall call this annealing method $adaptive\ deterministic\ annealing$.

The paper is organized as follows. In Section II, we briefly describe the framework of [21], [14] and present a modified SVM formulation. In Section III, we give motivation behind the proposed annealing method and derive an estimate of the annealing parameter. The implementation of the modified SVM formulation is in Section IV. Section V demonstrates experimental results on some data sets. Then we give discussion in Section VI.

II. A Modified SVM Formulation

First, we describe the segmentation problem using input-output pairs and outline the approach of [21]. Then we review the standard SVMs and introduce a modified formulation.

Let $(x_t, y_t) = (x_t, f_{r_t}(x_t))$ be generated by m unknown functions, where $t = 1, \ldots, l$, $r_t \in \{1, \ldots, m\}$, and $r_t = j$ if the t-th data point is generated by f_j. If we set $x_{t+1} = f_{r_t}(x_t)$, we get a time series $\{x_t\}$. The task is to determine r_t, $t = 1, \ldots, l$, and functions f_j, $j = 1, \ldots, m$. For notational simplicity, we take $y_t = f_{r_t}(x_t)$ in this section. The readers should notice that the method presented below does not restrict to time series with embedding order 1. The extension to higher embedding order can be done by simply replacing the scalar x_t by vectors from the method of the time delay embedding of the time series [10], [21].

Define $p_i^t = 1$ if $r_t = i$ and 0 otherwise. Then p_i^t, f_i, $i = 1, \ldots, m$, $t = 1, \ldots, l$ is an optimal solution of the following non-convex optimization problem:

$$\begin{aligned} \min_{p,f} \quad & \sum_{t=1}^{l} \sum_{i=1}^{m} p_i^t (y_t - f_i(x_t))^2 \\ \text{subject to} \quad & \sum_{i=1}^{m} p_i^t = 1, p_i^t \geq 0, \quad t = 1, \ldots, l. \end{aligned} \tag{1}$$

The main computations for solving (1) can be divided into two steps: (i) minimize the objective function in (1) over f_i with p_i^t fixed; (ii) adjust p_i^t given current f_i. In [21], the authors use RBF networks for step(i); i.e. for $i = 1, \ldots, m$, they solve

$$\min_{\hat{f}_i} \quad \sum_{t=1}^{l} p_i^t (y_t - \hat{f}_i(x_t))^2, \tag{2}$$

where $\hat{f}_i$ are RBF networks. For step(ii), they assume that $\hat{f}_i(x_t)$ are distributed according to Gaussian and then by Bayes rules

$$p_i^t = \frac{\exp\left(-\beta (e_i^t)^2\right)}{\sum_{j=1}^{m} \exp\left(-\beta (e_j^t)^2\right)}, \tag{3}$$

where

$$e_i^t = y_t - \hat{f}_i(x_t) \tag{4}$$

and β is the annealing parameter which controls the degree of competition among predictors. To incorporate the property of low switching rates, the sequence $((x_{t-\Delta}, y_{t-\Delta}), \ldots, (x_{t+\Delta}, y_{t+\Delta}))$ are assumed to come from the same source. Using Bayes rules again, the weighting coefficients p_i^t can be updated as

$$p_i^t = \frac{\exp\left(-\beta \sum_{\delta=-\Delta}^{\Delta} (e_i^{t-\delta})^2\right)}{\sum_{j=1}^{m} \exp\left(-\beta \sum_{\delta=-\Delta}^{\Delta} (e_j^{t-\delta})^2\right)}. \tag{5}$$

We will further discuss β in Section III and the choice of Δ in Section V.

We summarize the algorithm in [21] as follows:

Algorithm II.1

1) *Given Δ and parameters of RBF networks. Randomly pick initial p_i^t which satisfies*

$$\sum_{i=1}^{m} p_i^t = 1;\ p_i^t \geq 0, \quad t = 1, \ldots, l.$$

2) *Solve* (2) *using the RBF networks and obtain $\hat{f}_i$.*

3) *Increase β slightly and update p_i^t by* (5).

4) *If certain stopping criteria are met, terminate the algorithm. Otherwise, go to step 2.*

Next we propose to modify support vector regression (SVR) for solving (2). The idea of SVR is to map input data into a higher dimensional space and find a function which approximates the hidden relationships of the given data. The standard formulation is as follows.

$$\begin{aligned} \min_{w,b,\xi,\xi^*} \quad & \frac{1}{2} w^T w + C \sum_{t=1}^{l} (\xi^t + \xi^{t,*}) \\ \text{subject to } \ & -\epsilon - \xi^{t,*} \leq y_t - (w^T \phi(x_t) + b) \leq \epsilon + \xi^t, \\ & \xi^t \geq 0, \xi^{t,*} \geq 0, t = 1, \ldots, l, \end{aligned} \tag{6}$$

where w and b are unknown parameters to be estimated, ξ and ξ^* indicate errors, ϵ is the tolerance, and ϕ is the function that maps data x_t to a higher dimensional space. SVR uses the so called "regularization term" $\frac{1}{2} w^T w$ in (6) to avoid overfitting. Without this term, for certain mapping functions ϕ, the solution of (6) always satisfies $-\epsilon \leq y_t - (w^T \phi(x_t) + b) \leq \epsilon$, see, for example, [2, Corollary1]. Thus it overfits the data.

If we take

$$f(x) = w^T \phi(x) + b,$$

then (6) can be rewritten as

$$\min_{w,b} \quad \frac{1}{2} w^T w + C \sum_{t=1}^{l} |y_t - f(x_t)|_\epsilon \tag{7}$$

where $|\cdot|_\epsilon$ is the ϵ-insensitive loss function.

The standard SVR considers a uniform penalty parameter C for all data. By comparing (2) and (7), we propose to use different weights Cp_i^t in each error term of the objective function. Thus for each $i = 1, \ldots, m$, we have the following modification.

$$\begin{aligned} \min_{w_i,b_i,\xi,\xi^*} \quad & \frac{1}{2} w_i^T w_i + C \sum_{t=1}^{l} p_i^t (\xi_i^t + \xi_i^{t,*}) \\ \text{subject to } \ & -\epsilon - \xi_i^{t,*} \leq y_t - (w_i^T \phi(x_t) + b_i) \leq \epsilon + \xi_i^t, \\ & \xi_i^t \geq 0, \xi_i^{t,*} \geq 0, t = 1, \ldots, l. \end{aligned} \tag{8}$$

Note that if we remove the term $\frac{1}{2} w_i^T w_i$ and set $\epsilon = 0$, then (8) is equivalent to (2) with ℓ_2-norm replaced by ℓ_1-norm. The role of the regularization term $\frac{1}{2} w_i^T w_i$ is similar to that in the standard

SVR. Without it, we will overfit the data because in this case $-\epsilon \le y_t - (w_i^T \phi(x_t) + b_i) \le \epsilon$ for all t with $p_i^t > 0$.

The stopping criterion of our algorithm is as follows

$$\frac{|\hat{obj} - obj|}{|obj|} \le 0.05, \tag{9}$$

where $\hat{obj}$ and obj are the objective values of two consecutive iterations. Here the objective function is defined as

$$\sum_{t=1}^{l} \sum_{i=1}^{m} p_i^t |y_t - f_i(x_t)|.$$

A summary of our changes from Algorithm II.1 is in the following:

Algorithm II.2

1' *Given Δ, SVR parameters C and ϵ, and SVR mapping function ϕ, solve* (8) *and obtain $\hat{f}_i$.*

3' *Update β by the method described in Section III. If (9) is satisfied, stop the algorithm.*

For an account on the relation between RBF networks and SVRs with RBF kernels, see [3], [25] and references therein.

III. Adjustment of β Using Maximum Likelihood Estimation

Recall that the annealing parameter β in (3) and (5) controls the degree of competition. If β is large, $p_i^t \approx 1$ for $i = \text{argmin}_j \sum_{\delta=-\Delta}^{\Delta} (e_j^{t-\delta})^2$ and 0 otherwise. This is the so-called hard competition (winner-takes-all). If one uses hard competition right from the beginning, it is very likely to get stuck in an undesired local minimum. In [21], this problem was resolved by using deterministic annealing.

In this section we describe an adaptive method for adjusting β. Let e_i^t, defined in (4), denote the difference between the observed value and the estimated value. From (3), β can be viewed as a parameter that characterizes certain statistical property about e_i^t. Therefore, at each iteration we suggest to update β by maximizing the conditional likelihood function given current $\hat{f}_i$. To do so, we assume that, given current $\hat{f}_i$, y_t follows a mixture of Gaussian distributions p_i with mean $\hat{f}_i(x_t)$, a common unknown variance τ, and mixing coefficients a_i. Then, the density of y_t given $\hat{f}_i$ is

$$\begin{aligned} p(y_t|\hat{f}_i, \tau) &= \sum_{i=1}^{m} a_i p_i(y_t|\hat{f}_i, \tau) \\ &= \sum_{i=1}^{m} \frac{a_i}{\sqrt{2\pi\tau}} \exp\{-(e_i^t)^2/(2\tau)\}. \end{aligned} \tag{10}$$

Let $\hat{\tau}$ be an estimate of τ. Then using (10) we can estimate p_i^t by Bayes rules

$$\begin{aligned} \hat{p}_i^t &\equiv p(r_t = i|\hat{f}, y_t, \hat{\tau}) \\ &= \frac{p(y_t, r_t = i|\hat{f}, \hat{\tau})}{\sum_{k=1}^{m} p(y_t, r_t = k|\hat{f}, \hat{\tau})} \\ &= \frac{a_i \exp\{-(e_i^t)^2/(2\hat{\tau})\}}{\sum_{k=1}^{m} a_k \exp\{-(e_k^t)^2/(2\hat{\tau})\}}. \end{aligned} \tag{11}$$

By comparing (5) and (11), we suggest to choose $1/(2\hat{\tau})$ as our next β. Since $\hat{\tau}$ is the measure of the variation of e_i^t, it is intuitively clear that the next $\hat{\tau}$ will decrease if $\hat{f}_i$ in the next iteration can better fit the data. So the new β is likely to increase, which corresponds to the annealing property used in [24], [21].

We propose to estimate τ in each iteration by the maximum likelihood principle. This can be obtained by Expectation Maximization method. To see how, let r_t be as in Section II, which is a latent variable (unobservable). We augment y_t with r_t and call $\{(y_t, r_t) : t = 1, \ldots, l\}$ the complete data. Then the complete-data log-likelihood function of τ is

$$\begin{aligned} L(\tau) &= \sum_{t=1}^{l} \log p(y_t, r_t | \hat{f}_i, \tau) \\ &= \sum_{t=1}^{l} \log a_{r_t} p_{r_t}(y_t | \hat{f}_i, \tau). \end{aligned} \tag{12}$$

Let $\tau^{(g)}$ and $p_i^{t(g)} \equiv p(r_t = i | \hat{f}, y_t, \tau^{(g)})$ be current estimates. Let $Y \equiv (y_1, \ldots, y_l)$ and $R \equiv (r_1, \ldots, r_l)$. The E-step calculates the expectation of the conditional log-likelihood function of (Y, R) given Y and $\tau^{(g)}$:

$$Q(\tau, \tau^{(g)}) \tag{13}$$

$$\begin{aligned} &\equiv E[\log p(Y, R | \hat{f}, \tau) | \hat{f}, Y, \tau^{(g)}] \\ &= \sum_{t=1}^{l} E[\log p(y_t, r_t | \hat{f}, \tau) | \hat{f}, Y, \tau^{(g)}] \\ &= \sum_{t=1}^{l} \sum_{i=1}^{m} \log(a_i p_i(y_t | \hat{f}_i, \tau)) p_i^{t(g)} \\ &= \sum_{t=1}^{l} \sum_{i=1}^{m} \{(\log a_i) p_i^{t(g)}\} + \sum_{t=1}^{l} \sum_{i=1}^{m} \{[\log p_i(y_t | \hat{f}, \tau)] p_i^{t(g)}\}, \end{aligned} \tag{14}$$

where the second equality follows from the independence of each observation and the third equality follows from (12) and the definition of $p_i^{t(g)}$.

The M-step finds the maximizer of Q:

$$\tau^{(g+1)} = \arg\max_{\tau} Q(\tau, \tau^g). \tag{15}$$

By (14) and Gaussian assumption, we find that the maximum of (14) occurs at

$$\begin{aligned} \tau^{(g+1)} &= \frac{\sum_{t=1}^{l} \sum_{i=1}^{m} \{(e_i^t)^2 p_i^{t(g)}\}}{\sum_{t=1}^{l} \sum_{i=1}^{m} p_i^{t(g)}} \\ &= \frac{\sum_{t=1}^{l} \sum_{i=1}^{m} \{(e_i^t)^2 p_i^{t(g)}\}}{l}, \end{aligned} \tag{16}$$

where $p_i^{t(g)}$ is obtained by replacing $\hat{\tau}$ in (11) by $\tau^{(g)}$.

To apply the above method, the Gaussian assumption in (10) is not necessary. For example, if we replace the mixture of Gaussians by a mixture of Laplace (double exponential) distributions with a

common scale parameter τ, i.e. $p_i(y_t|\hat{f}_i,\tau) \propto \tau^{-1}\exp(-|y_t-\hat{f}_i|/\tau)$, the expectation and maximization steps can be carried out similarly. The resulting estimate of τ is similar to (16), but with $(e_i^t)^2$ replaced by $|e_i^t|$. Because here a linear loss function is used for SVR, we suggest to use $|e_i^t|$ instead of $(e_i^t)^2$ when updating p_i^t. Therefore, at the $(g+1)$st iteration of our implementation, we update p_i^t by (5) with β replaced by $1/\tau^{(g+1)}$ and $(e_i^t)^2$ replaced by $|e_i^t|$. Note that we use (5) because the parameter Δ incorporates the slowly switching property [21] while a_i are less important in the implementation.

IV. Solving the Modified Support Vector Regression

In this section we discuss how to solve the new SVR formulation (8). As SVRs map data into higher dimensional spaces, w is a long vector variable. Thus, usually a dual problem is easier to be solved. The Lagrangian dual of (8) can be derived by a similar way as that of standard SVR so here we give only the resulting formulation:

$$\begin{aligned} \min_{\alpha,\alpha^*} \quad & \frac{1}{2}(\alpha-\alpha^*)^T K(\alpha-\alpha^*) + \epsilon\sum_{t=1}^{l}(\alpha_t+\alpha_t^*) + \sum_{t=1}^{l} y_t(\alpha_t-\alpha_t^*) \\ \text{subject to} \quad & \sum_{t=1}^{l}(\alpha_t-\alpha_t^*) = 0, \\ & 0 \le \alpha_t, \alpha_t^* \le C p_i^t, t = 1,\ldots,l, \end{aligned} \tag{17}$$

where K is a square matrix with $K_{t,o} = K(x_t,x_o) \equiv \phi(x_t)^T\phi(x_0)$. Here we mainly consider the RBF kernel $K(x_i,x_j) = e^{-\gamma||x_i-x_j||^2}$. The main difference is that each dual variable has its own upper bound Cp_i^t. At the optimal solution $w_i = \sum_{t=1}^{l}(\alpha_t-\alpha_t^*)\phi(x_t)$. So

$$\hat{f}_i(x) = \sum_{t=1}^{l}(\alpha_t-\alpha_t^*)K(x_t,x).$$

If $\alpha_t - \alpha_t^* \neq 0$, then x_t is a support vector of the modified SVR (17). The main difficulty on solving (17) is that K is a large dense matrix. This issue has occurred for the case of classification and some methods such as the decomposition method (e.g. [19]) have been proposed. The decomposition method avoids the memory problem by iteratively working on few variables. The extreme is the Sequential Minimal Optimization (SMO) [22] where in each iteration only two variables are updated. Here we consider the SMO-type implementation in LIBSVM [1]. Originally in LIBSVM the two variables are selected by the following rule: The first element is the α_t (or α_t^*) which has the smallest value of the following numbers:

$$\begin{aligned} &-(K(\alpha-\alpha^*))_t - \epsilon - y_t, \quad \text{if} \quad \alpha_t > 0 \text{ and} \\ &-(K(\alpha-\alpha^*))_t + \epsilon - y_t, \quad \text{if} \quad \alpha_t^* < C. \end{aligned}$$

The second element is the α_t (or α_t^*) which has the largest value of

$$\begin{aligned} &-(K(\alpha-\alpha^*))_t - \epsilon - y_t, \quad \text{if} \quad \alpha_t < C \text{ and} \\ &-(K(\alpha-\alpha^*))_t + \epsilon - y_t, \quad \text{if} \quad \alpha_t^* > 0. \end{aligned}$$

They are derived from the optimality condition of the dual of (7). We change C in the above formula to Cp_i^t while the convergence of the algorithm still holds. The modified code of LIBSVM is available upon request.

V. Experiments

A. Four Chaotic Time Series

We test the case of completely overlapping input manifold used in [21]. For all (x_t, y_t), $y_t = f_{r_t}(x_t)$. They consider $x \in [0, 1]$ and four different functions: $f_1(x) = 4x(1-x)$, $f_2(x) = 2x$ if $x \in [0, 0.5)$ and $2(1-x)$ if $x \in [0.5, 1]$, $f_3(x) = f_1(f_1(x))$, and $f_4 = f_2(f_2(x))$. It is easily seen that all the f_i map x from $[0, 1]$ to $[0, 1]$. They set $x_{t+1} = y_t = f_{r_t}(x_t)$ and activate the four functions consecutively, each for 100 time steps, giving an overall 400 time steps. They then repeat the procedure three times and obtain a time series of 1,200 points. An illustration of these functions is in Figure 1. As the number of functions is considered unknown, we start from six competing SVMs. In the end some SVMs represent the same function and the six SVMs represent four different functions. If we use only four competing SVMs in the beginning, we find it is easier to fall into local minima than using six SVRs.

For SVR parameters, we simply set $C = 1$ and the width of the insensitive tube ϵ to be 0.03 (actually, we may set ϵ to be any nonnegative small number, say 0 to 0.05). We set the parameter γ of the RBF kernel to be 50. Since a smaller γ means greater smoothness of predictors, in order to avoid getting trapped in a local minimum, we may start with a smaller γ and gradually increase it during training. Therefore, we also try to adaptively adjust γ during training by setting $\gamma^{(g)} = 1/(2\tau^{(g)})$, where $\tau^{(g)}$ is as in (16). It works well too. The choice of Δ is the same as that in [8]: as the small switching rate 1/100 guarantees a large probability for short sequences of length 7 to contain no switching event, a choice of $\Delta = 3$ would work. We also find that this parameter is not crucial during training. For this case the algorithm stops in about six iterations and the data points are well separated. See Figure 1 for the first four iterations. Here the initial p_i^t are chosen to be around 1/6. So in the beginning, the six predictors are about the same. As the iteration goes, the underlying structure resolves in a hierarchical manner. We then add noise $0.1N(0, 0.5)$ to these four functions. The algorithm stops in seven iterations. In Figure 2 we present the first six iterations. For this case we randomly assign the initial p_i^t to be 0 or 1 under the condition $\sum_{i=1}^6 p_i^t = 1$, $t = 1, \ldots, 1200$. By comparing the first graphs in Figures 1 and 2, we find that if the initial p_i^t are chosen to be around 1/6, the six predictors obtained at the first iteration are closer to each other.

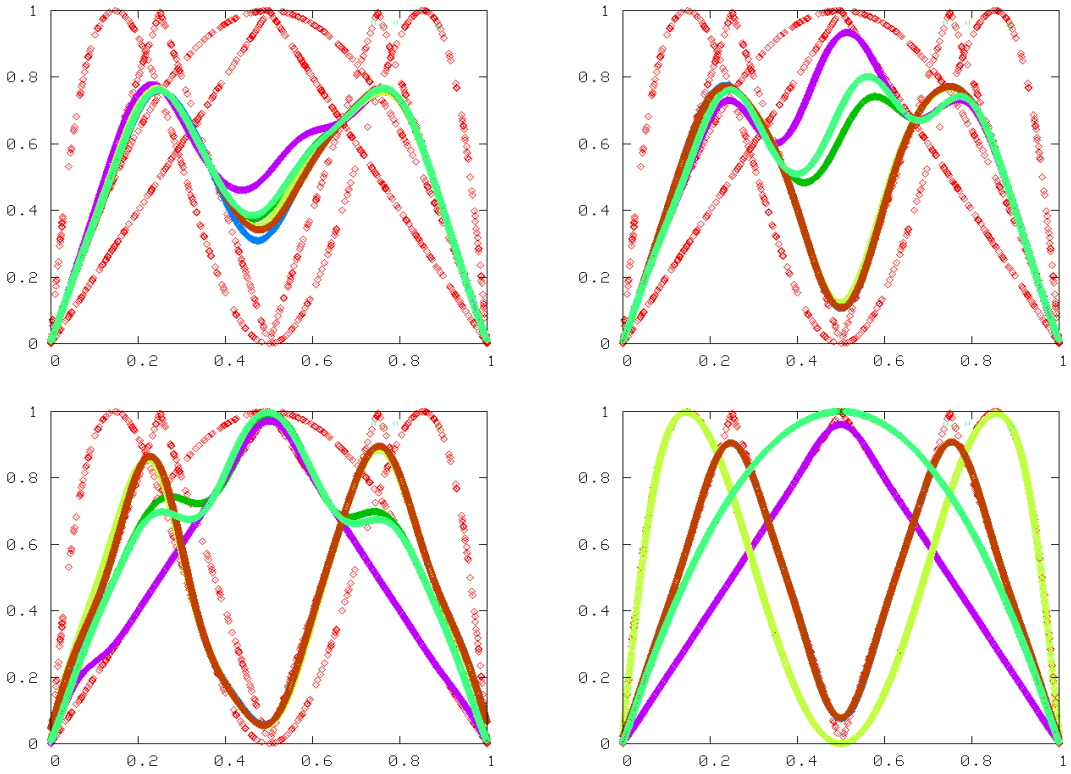

Fig. 1. First four iterations (data without noise)

B. Mackey-Glass chaotic system

Next we consider a high-dimensional chaotic time series obtained from the Mackey-Glass delay-differential equation [12]:

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-t_d)}{1+x(t-t_d)^{10}}.$$

Following earlier experiments, points are selected every six time steps. We generate 300 points with delay parameter $t_d = 23$. Then we switch t_d be to 17, 23, and 30, and generate also 300 points for each of the t_d. Totally there are 1,200 points from three underlying dynamics. The embedding dimension is $d = 6$. That is, y_t is the one-step ahead value of $x_t = (y_{t-1}, \ldots, y_{t-6})$. For this problem we set the $\gamma = 1$ and leave the other parameters the same as in the previous example. As can be seen in Figure 3, the underlying dynamics are well segmented.

C. Santa Fe Time Series: Data Set D

We also consider the benchmark Data Set D from the Santa Fe Time Series Prediction Competition [27]. This is an artificial data generated from a nine-dimensional periodically driven system with an asymmetrical four-well potential and a drift on the parameters. The system operates in one well for

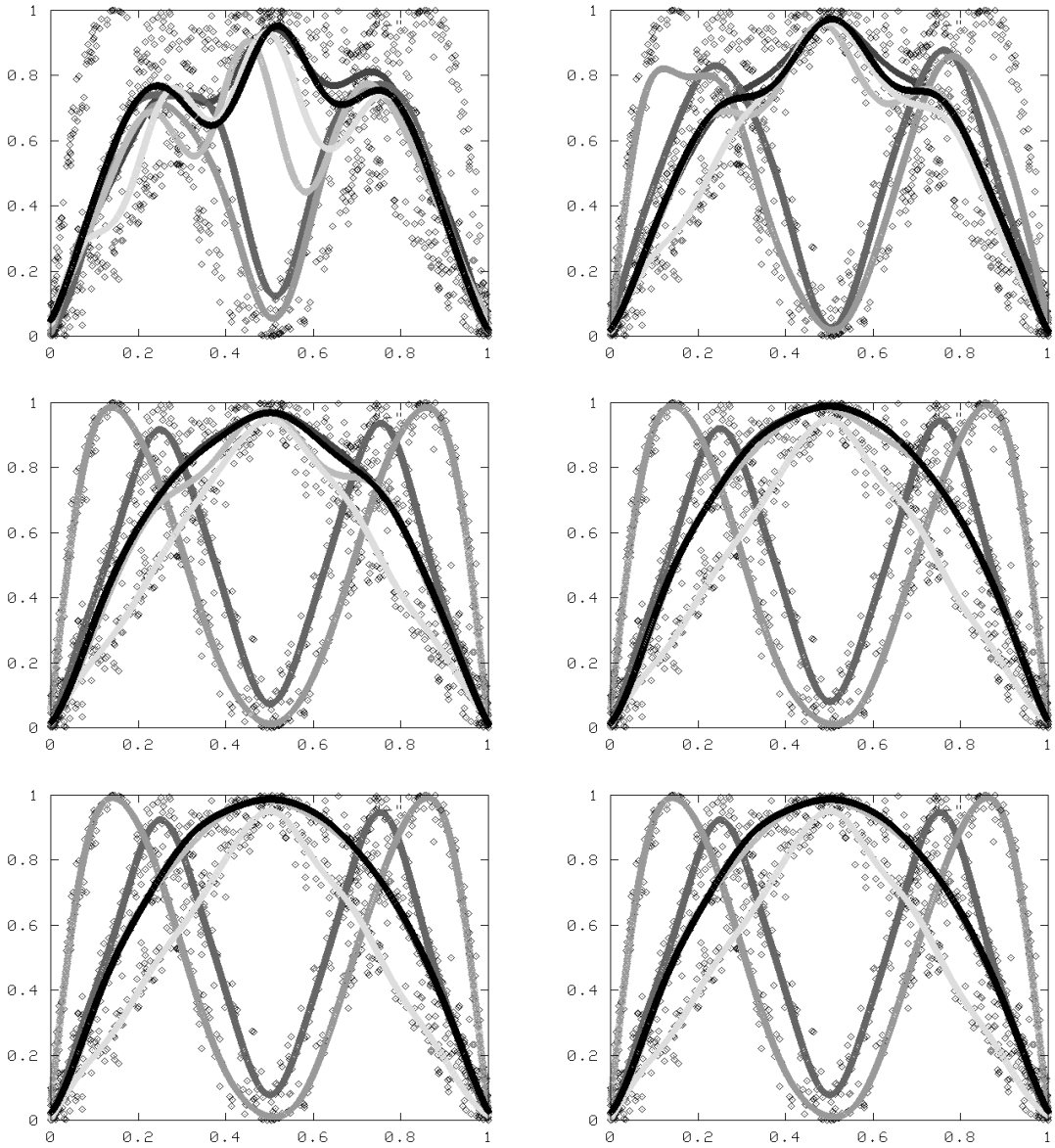

Fig. 2. First six iterations (data with noise)

some time and then switch to another well with a different dynamical behavior. Each participant is asked to predict the first 25 steps and the evaluation is based on the root mean squared error (RMSE) of the prediction. The RMSE obtained by the winner of the Santa Fe Competition, Zhang and Hutchinson [27, pp219-241], is 0.0665.

One may train a model on the whole dataset and then conduct predictions. But if data are actually from different sources, it would be better to do segmentation before prediction. [21] proposes to use the “Annealed Competition of Experts” (ACE) method to segment the time series into regimes of

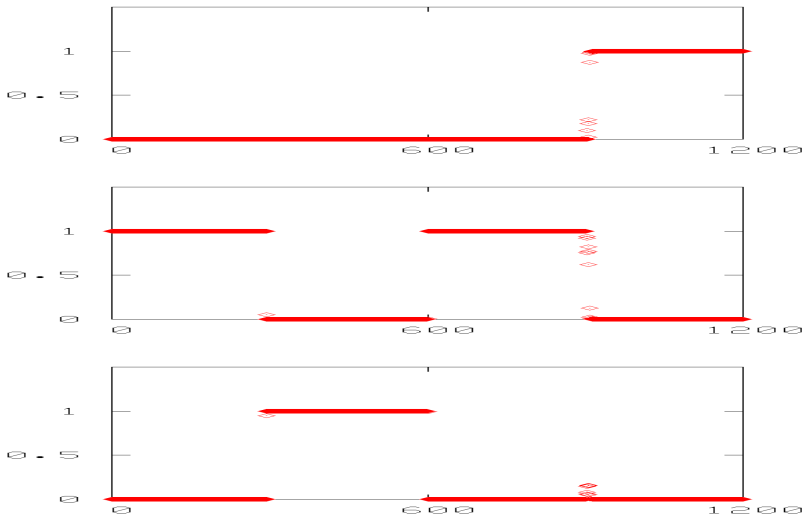

Fig. 3. $p_i^t, i = 1, \ldots, 3$ at each time point t (x-axis: time; y-axis: p_i^t)

approximately stationary dynamics by using six RBF networks. Then the prediction is simply done by iterating the particular predictor that is responsible for the generation of the latest training data. Similar to [21], in [17] they uses ACE with six RBF networks to segment the time series. Instead of RBF networks, they then use SVR to train the particular class of data that includes the data points at the end of the full training set. They also use SVR to train the full dataset and compare its predicting performance with that obtained by using segmentation.

Here we start with the standard SVR (6) and train a model on the whole dataset. We set the embedding dimension as 20 and the kernel parameter $\gamma = 1.5$ as in [17]. The SVR parameter ϵ is set to be 0 and C is determined by five fold cross validation on $\{2^{-4}, 2^{-3}, \ldots, 2^5\}$. By five fold cross validation we mean that after data are transformed into (x_t, y_t), they are randomly separated to five groups. Sequentially one group is validated by training the rest. We then use this model to predict the first 25 steps. Unfortunately, the C value with the smallest validation error does not always produce the best prediction. For example, a typical run shows that: the smallest validation error 0.0351 occurs at $C = 2^3$ and the prediction error is 0.0748; the smallest prediction error 0.0521 is attained at $C = 2^4$, but it has a larger validation error 0.0357. As the result of a single run is not always representative, we decide to investigate the long-run performance of this method by repeating the same procedure 30 times with different cross validations chosen randomly. The distribution of the selected C values is shown in Figure 4 (the light bars). In average, the RMSE of the 25-step prediction is 0.0871.

Next we use Algorithm II.1 with six competing SVRs to segment the time series. The parameters in the modified SVR (8) are set as $\gamma = 0.05$ (as in [17]), $\epsilon = 0$, and $C = 0.01$. Then we evaluate the p_i^t values for each predictor at the last 25 points of the training data. The predictor with the

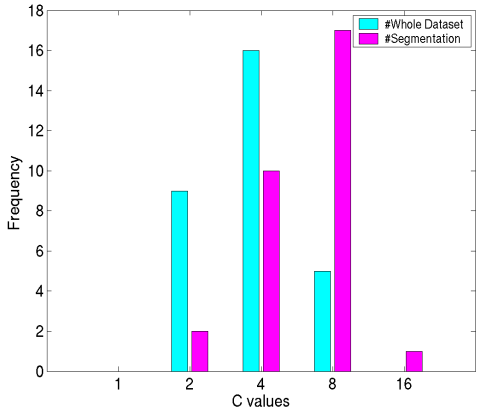

Fig. 4. The Distribution of the Selected C Values

largest p_i^t for the majority of the 25 points is chosen. After doing so, we use the standard SVR (6) to train this particular class of data. Here we set $\epsilon = 0$ and select C by five fold cross validation on $\{2^{-4}, 2^{-3}, \ldots, 2^5\}$. The resulting model is used to predict the first 25 steps of the testing data. To study the long-run performance of the method we repeat this procedure 30 times. The average of the RMSEs of the prediction errors is 0.0656. The distribution of the best C selected is given in Figure 4 (the dark bars). Of the 30 experiments, 15 of them outperform the result obtained by the winner. Due to different initial p_i^t, the number of data points chosen for after the segmentation varies. For our 30 runs, this number ranges from 400 to 1,100. The best prediction performances (RMSE = 0.038 to 0.042) occur at numbers around 800 to 1,000. Our finding is that the prediction result using segmentation is better than that using the whole dataset directly.

As mentioned in [17], determining the SVR parameters (C, ϵ) is computationally intensive. They suggest to determine them at the minimum of the one step prediction error measured on a randomly chosen validation set. To have a more stable parameter selection, here we use five fold cross validation to select C. Moreover, instead of setting $\epsilon = 0$, we also tried positive ϵ, but the results are not better. It is desirable to compare our parameter settings with those in [17]. However, they did not indicate their final choice of the parameters so we cannot make the comparison.

VI. Discussion

In this paper we present a framework for the unsupervised segmentation of non-stationary time series using SVMs. The problems we consider here are the same as those in [21], [14]. The method used in this paper is novel in two aspects. First, a new formulation of SVM and its implementation are proposed for the switching problems. Second, we use statistical reasoning to adjust the annealing parameter. The annealing property of the proposed method is given intuitively in Section II. From the

figures we obtained in Section V, our results are comparable to those in [21]. Moreover, as we adjust β adaptively, our algorithm requires much fewer iterations than theirs.

Several parameters in our algorithm must be chosen in the beginning. For the mapping function ϕ, we use the RBF kernel $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$, which is the most popular choice. The selection of C, the width of the insensitive tube ϵ in (8), and the kernel parameter γ is usually by try and error. For the two chaotic time series examples in Section V, we can easily select suitable C, ϵ, and γ, see discussions in that section. However, for the prediction of Data Set D from the Santa Fe Time Series Competition, it takes quite a long time to tune the parameter.

For a slowly changing dynamics, [21] is the first paper that proposes to use the parameter Δ for updating p_i^t. Intuitively, this method would work better for smaller switching rates. The switching rates of the first two examples in Section V are set to be 1/100 and 1/300 respectively, the same as those in [21]. For the chaotic time series example in Section V-A, an unreported experiment shows that our algorithm also works very well for a switching rate as high as 1/5.

As to the computational time, the major cost in each iteration of the proposed approach is on solving m different SVM problems. It is known that for most SVM implementations fewer support vectors (i.e. $\alpha_t - \alpha_t^* \neq 0$, where (α_t, α_t^*) is optimal for (8)) leads to less training time. We also consider free support vectors (i.e. $-Cp_i^t < \alpha_t - \alpha_t^* < 0$ or $0 < \alpha_t - \alpha_t^* < Cp_i^t$) because for some SVM implementations they are more related to the training time than the number of support vectors. The reason is that these implementations are iterative procedures and in final iterations mainly those free support vectors are still considered. More discussions can be found in [1]. Figure 5 presents the number of support vectors and the number of free support vectors at each iteration for the chaotic time series example in Section V-A. Here the data is generated without adding the noise. To interpret, in the first iteration the average number of support vectors of the six SVMs is 133, the average number of free support vectors is 11; in the second iteration the average number of support vectors is 701, that of free support vectors is 26, and so on.

Remember that there are 1,200 training data points and six competing SVRs. In this section, we randomly assign the initial p_i^t to be 0 or 1 under the condition $\sum_{i=1}^{6} p_i^t = 1$, $t = 1, \ldots, 1200$. Therefore, at the first iteration, approximately 200 data points are assigned to each predictor and the average number of support vectors of the six SVRs are less than 200. After updating p_i^t, for all t with $p_i^t > 0$ the corresponding data points are assigned to the i-th SVR. So the number of support vectors substantially increased at the second iteration. Then as the underlying structure gradually resolved, the number of support vectors reduces again. An important trick we used here is to set all $p_i^t \leq 0.01$ to be zero. This effectively decreases the number of support vectors.

As for the numbers of free support vectors, they are all small compared to support vectors. At an optimal solution of an SVM, some bounded variables have small values (i.e. small corresponding p_i^t)

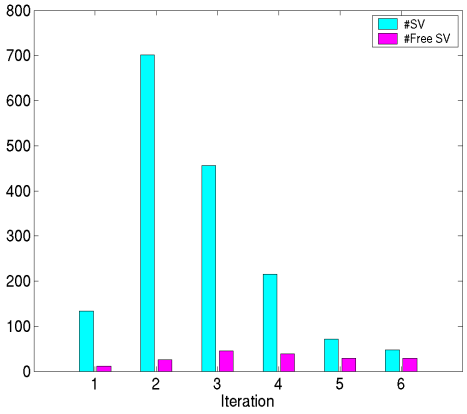

Fig. 5. Number of support vectors (i.e. $\alpha_t - \alpha_t^* \neq 0$) and number of free support vectors (i.e. $-Cp_i^t < \alpha_t - \alpha_t^* < 0$ or $0 < \alpha_t - \alpha_t^* < Cp_i^t$)

but some are large (i.e. corresponding p_i^t close to 1). For variables with small bounds, our current implementation, LIBSVM, can quickly identify them, so the training time of the first iteration is similar to that of the second iteration (less than two seconds on a Pentium III-500 machine). In other words, though the number of support vectors is dramatically increased, at the second iteration, many p_i^t are small so LIBSVM quickly identifies bounded variables. Then the decomposition method mainly works on free variables, so the difference between the two iterations is not that much.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Council of Taiwan via the grants NSC 90-2213-E-002-111 and NSC 91-2118-M-004-003. The second author thanks Danil Prokhorov and other organizers of IJCNN Challenge 2001 for bringing him to this subject. The authors also thank Jens Kohlmorgen for some helpful comments.

REFERENCES

[1] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] C.-C. Chang and C.-J. Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119–2147, 2001.

[3] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50, 2000.

[4] L. A. Feldkamp, T. M. Feldkamp, and D. V. Prokhorov. An approach to adaptive classification. In S. Haykin and B. Kosko, editors, *Intelligent signal processing*. IEEE Press, 2001.

[5] R. J. Hathaway and J. C. Bezdek. Switching regression models and fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 1(3):195–204, 1993.

[6] S. Haykin and D. Cong. Classification of radar clutter using neural networks. *IEEE Transactions on Neural Networks*, 2:589–600, 1991.

[7] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[8] A. Kehagias and V. Petridis. Time-series segmentation using predictive modular neural networks. *Neural Computation*, 9:1691–1709, 1997.

[9] J. Kohlmorgen, K.-R. Müller, J. Rittweger, and K. Pawelzik. Identification of nonstationary dynamics in physiological recordings. *Biological Cybernetics*, 83:73–84, 2000.

[10] W. Liebert, K. Pawelzik, and H. G. Schuster. Optimal embeddings of chaotic attractors from topological considerations. *Europhys. Lett.*, 14:521, 1991.

[11] S. Liehr, K. Pawelzik, J. Kohlmorgen, and K. R. Müller. Hidden markov mixtures of experts with an application to EEG recordings from sleep. *Theory in Biosci.*, 118(3-4):246–260, 1999.

[12] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

[13] J. Moody and C. J. Darken. Fast learning in networks of locally–tuned processing units. *Neural Computation*, 1:281–293, 1989.

[14] K.-R. Müller, J. Kohlmorgen, and K. Pawelzik. Analysis of switching dynamics with competing neural networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E78–A(10):1306–1315, 1995.

[15] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

[16] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings of ICANN97: : Proc. of the Int. Conf. on Artificial Neural Networks*, pages 999–1004, Berlin, 1997. Springer.

[17] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 243–254, Cambridge, MA, 1999. MIT Press.

[18] R. Murray-Smith and T. A. J. (Eds.). *Multiple Model Approaches to Modelling and Control*. Taylor and Francis, London, 1997.

[19] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, pages 130–136, New York, NY, 1997. IEEE.

[20] K. Pawelzik. Detecting coherence in neuronal data. In L. Van Hemmen and K. Schulten, editors, *Physics of neural networks*. Springer, 1994.

[21] K. Pawelzik, J. Kohlmorgen, and K.-R. Müller. Annealed competition of experts for a segmentation and classification of switching dynamics. *Neural Computation*, 8(2):340–356, 1996.

[22] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

[23] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[24] K. Rose, F. Gurewitz, and G. Fox. Statistical mechanics and phase transitions in clustering. *Physical Rev. Letters*, 65(8):945–948, 1990.

[25] B. Schölkopf, K.-K. Sung, C. J. C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765, 1997.

[26] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.

[27] A. S. Weigend and N. A. Gershenfeld, editors. *Time series prediction: forcasting the future and understanding the past*. Addison-Wesley, 1994.

Ming-Wei Chang received his B.S. double degree in Computer Science & Information Engineering and Mathematics in 2001 from National Taiwan University, where in 2003 he received his M.S. degree in Computer Science & Information Engineering. His research interests include machine learning, numerical optimization, and algorithm analysis.

Chih-Jen Lin (S'91-M'98) received his B.S. degree in Mathematics from National Taiwan University in 1993. From 1995 to 1998 he studied at the Department of Industrial and Operations Engineering at the University of Michigan and received his M.S. and Ph.D. degrees. He is currently associate professor in the Department of Computer Science and Information Engineering at National Taiwan University. His research interests include machine learning, numerical optimization, and applications of Operations Research.

Ruby Chiu-Hsing Weng received her B.S. and M.S. degrees in Mathematics from National Taiwan University in 1993 and 1995, and Ph.D. degree in Statistics from the University of Michigan in 1999. She is Assistant Professor, Department of Statistics, National Chengchi University, Taipei. Her research interests include sequential analysis and time series analysis.