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
E-mail: lincj@ccms.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 time series using support vector regression. It applies to non-stationary time series which alter in time. We follow the architecture by Pawelzik et al. [13] which consists of competing predictors. In [13] competing Neural Networks were used while here we exploit the use of Support Vector Machines, a new learning technique. Results indicate that the proposed approach is as good as that in [13]. Differences between the two approaches are also discussed.

I. Introduction

Recently support vector machines (SVM) [17] has been a promising method for data classification and regression. However, its use on other types of problems have not been exploited much. In the paper we will apply it to unsupervised segmentation of time series. We consider the case in Pawelzik et al. [13] where different samples (x_t, y_t) are generated by a number m of unknown functions $f_{r_t}, r_t \in \{1, \ldots, m\}$ which alternate according to r_t, i.e. $y_t = f_{r_t}(x_t)$. We then would like to determine functions f_r with their respective r_t given time series $\{x_t, y_t\}_{t=1}^{l}$. Therefore, it is likely that given points on different function surfaces, the task is to separate these points to different groups where each one corresponds to points on one surface.

Practical applications of time-series segmentation include, for example, speech recognition [14], signal classification [5], and brain data [12].

As without any training information, this problem must be considered in an unsupervised manner. In order to correctly separate these points, we cannot only count on the information of $\{x_t, y_t\}$. Previous approaches usually need some additional properties.

In [13], the authors assumed that time series have a low switching rate. That is, in general data before and after any given time point t are from the same time series. Therefore, in addition to the spatial relation of $x_t, t = 1, \ldots, l$, such an assumption provides more connections among x_t. Another example by Feldkamp et al. [4] does not consider $\{x_t, y_t\}$ to be from slowly changing time series. However, they assume, for example, there is a binary sequence like

$$01001011\cdots,$$

where each $\{x_t, y_t\}$ is associated with one of the four categories: 00, 01, 10, and 11. Hence if $\{x_t, y_t\}$ is in the 00 class, $\{x_{t+1}, y_{t+1}\}$ must be in 00 or 01. Such additional information are typically available based on different applications. Here we will focus on time series so the same condition on a slow changing rate is assumed.

Many papers have considered this issue by using neural networks. See [13], [10], [6] and references therein. Another important approach is via hidden Markov models where examples are [1], [7], [16]. Basically [13] proposed to use several competing neural networks weighted by their relative performance. Weights of different networks are adjusted in an annealed manner where we gradually increase the degree of competition. The neural network used is a radius basis function network of the Moody-Darken type [9]. Here we follow a similar framework but discuss it more from a point of view of solving a global minimization problem. In addition, instead of RBF networks we used SVM where their differences are also discussed.

This paper is organized as follows. In Section II, we discuss our approach and present how SVM can be incorporated. An important parameter in our algorithm is β whose calculation will be discussed in Section III. Section IV demonstrates experimental results on some data sets. We present some discussions in Section V.

0-7803-7278-6/02/$10.00 ©2002 IEEE

II. Annealed Competition of Support Vector Machines

If without considering noise, we have

$$y_t = f_{r_t}(x_t), \quad t = 1, \ldots, l.$$

If we assign

$$p_i^t = \begin{cases} 1 & \text{if } r_t = i, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

then

$$\sum_{t=1}^{l} \sum_{i=1}^{m} p_i^t (y_t - f_i(x_t))^2 = 0.$$

Therefore, p_i^t, f_i, $i = 1, \cdots, m$, $t = 1, \cdots, l$ is an optimal solution of the following non-convex optimization problem:

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

Of course we can always find a single function which fits all data so that the objective value of (2) is zero and then it is already optimal. What we need is to avoid overfitting and adjust values of p_i^t so that (1) is obtained. Then according to whether p_i^t is zero or one, we can find out which group a point (x_t, y_t) belongs to.

In [13], the authors consider an iterative process where in each iteration p_i^t are fixed first and m Radial-Basis function (RBF) networks are used to minimize the quadratic functions:

$$\min_{\hat{f}_i} \quad \sum_{t=1}^{l} p_i^t (y_t - \hat{f}_i(x_t))^2, \quad i = 1, \ldots, m, \tag{3}$$

where

$$\hat{f}_i(x) = \sum_{t=1}^{\bar{l}} \alpha_t e^{-\frac{\|x - c_t\|^2}{2\sigma^2}} \tag{4}$$

is the ith approximate predictor. Here $c_1, \ldots, c_{\bar{l}}$ are the "centers" used for constructing functions.

After new $\hat{f}_i$ are obtained, they update p_i^t by

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

where

$$e_i^t = y_t - \hat{f}_i(x_t) \tag{6}$$

and β is a parameter which controls the degree of competition.

This updating rule on p_i^t is from Bayes' rule with the assumption that $((x_{t-\Delta}, y_{t-\Delta}), \cdots, (x_{t+\Delta}, y_{t+\Delta}))$ are from the same time series. Therefore, using (5) we can put subsequent time series data into the same group. In addition, if β is large, then $p_i^t \approx 1$ for $i = \text{argmax}_j \sum_{\delta=-\Delta}^{\Delta} (e_j^{t-\delta})^2$. This is the so-called hard competition (winner-takes-all).

Here instead of (2), we consider

$$\begin{aligned} \min_{p,f} \quad & \sum_{t=1}^{l} \sum_{i=1}^{m} p_i^t |y_t - f_i(x_t)| \\ & \sum_{i=1}^{m} p_i^t = 1, p_i^t \geq 0, \quad t = 1, \ldots, l. \end{aligned} \tag{7}$$

When p_i^t is fixed, by considering

$$\hat{f}_i(x) = w_i^T \phi(x) + b,$$

we then solve

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

which is a modification of the standard support vector regression. Note that if without the term $\frac{1}{2} w_i^T w_i$, $\epsilon = 0$, and p_i^t are fixed, (8) is equivalent to (7). The original idea of support vector regression is to find a function which approximates the hidden relationships of the given data. Here data x_t are mapped into a higher dimensional space by the function ϕ. SVM uses the ϵ-insensitive loss function where data points in a tube with width ϵ are considered correctly approximated. Note that we have different weights Cp_i^t in each term of the objective function. The original SVM usually considers a uniform penalty parameter C for all data. It is essential to use the so called "regularization term" $\frac{1}{2} w_i^T w_i$ in (8). Otherwise, if the kernel matrix K with $K_{to} = \phi(x_t)^T \phi(x_o)$ is positive definite, the solution of (8) will have

$$y_t = w_i^T \phi(x_t) + b_i, \text{ if } p_i^t > 0. \tag{9}$$

That is, overfitting occurs and we are trapped at a local minimum which is not what we want. Adding $\frac{1}{2} w_i^T w_i$ remedies this problem so (9) does not happen in early iterations. Then we can calculate the error in (6) and use them for updating p_i^t in (5).

At the optimal solution of (8) we have

$$w_i = \sum_{t=1}^{l} \alpha_i^t \phi(x_t), \tag{10}$$

where α_i^t are obtained from the dual formulation described later. So the ith predictor is

$$\hat{f}_i(x) = \sum_{t=1}^{l} \alpha_i^t K(x_i, x),$$

where $K(x_t, x) = \phi(x_t)^T \phi(x)$ is usually called the kernel function.

Next we compare the use of RBF networks and SVM. If the RBF kernel is used for SVM,

$$K(x_t, x) = e^{-\frac{\|x_t - x\|^2}{2\sigma^2}}.$$

Therefore if we choose $\bar{l} = l$, $c_t = x_t$ in (4), $\epsilon = 0$ in (8), and use a quadratic loss function

$$C \sum_{t=1}^{l} p_i^t ((\xi_i^t)^2 + (\xi_i^{t,*})^2) \tag{11}$$

in (8), then (11) is in fact the same as (3). Another difference is on the regularization term. Usually RBF is implemented with an additional regularization term $\frac{1}{2}\sum_{t=1}^{\bar{l}} (\alpha_i^t)^2$. This is different from $\frac{1}{2} w^T w$ in SVR which can be rewritten as

$$\frac{1}{2} \sum_{t=1}^{l} \sum_{o=1}^{l} \alpha_i^t \alpha_i^o K(x_t, x_o).$$

A possible advantage of SVM is that with the linear ϵ-insensitive loss function, it automatically decides the number of nonzero α_i^t so that $\phi(x_t)$ will be used to construct $\hat{f}_i$. On the contrary, RBF networks have to decide $\bar{l}$ and c_t in advance. An example on comparing RBF networks and SVM for classification problems is in [15]. Of course how to set an appropriate ϵ is also an issue. More discussions on the relation between the RBF networks and SVM are in [3].

Implementations of RBF networks and SVM are also different. As (3) is an unconstrained minimization where its first derivative becomes a linear system, sometimes a direct method such as Gaussian elimination is used. But sometimes iterative methods using the steepest descent direction are considered. For the modified form of SVR (8), we usually consider its dual:

$$\begin{aligned} \min_{\bar{\alpha}, \bar{\alpha}^*} \quad & \frac{1}{2}(\bar{\alpha} - \bar{\alpha}^*)^T K (\bar{\alpha} - \bar{\alpha}^*) + \epsilon \sum_{t=1}^{l} (\bar{\alpha}_t + \bar{\alpha}_t^*) \\ & + \sum_{t=1}^{l} y_t (\bar{\alpha}_t - \bar{\alpha}_t^*) \\ & \sum_{t=1}^{l} (\bar{\alpha}_t - \bar{\alpha}_t^*) = 0, \qquad (12) \\ & 0 \le \bar{\alpha}_t, \bar{\alpha}_t^* \le C p_i^t, t = 1, \ldots, l, \qquad (13) \end{aligned}$$

where K is a square matrix with $K_{t,o} = K(x_t, x_o)$. Then $\bar{\alpha}_t - \bar{\alpha}_t^*$ is the α_i^t used in (10). The main difficulty on solving (1) is that K is a large dense matrix. This issue has occurred for the case of classification where some methods such as the decomposition method (e.g. [11]) has been proposed. The decomposition method starts from the zero vector and can avoid the memory problem if the percentage of support vectors (i.e. $\alpha^t - \alpha^{t,*} \neq 0$) is small. It has been extended to regression but we need further modifications for handling different upper bounds Cp_i^t. Here we consider the decomposition method used by the software LIBSVM [2] which can be easily modified for our purpose. The modified code of LIBSVM is available upon request.

When the RBF function is used, the parameter σ affects how data are fitted. It adjusts the smoothness of predictors. Thus if initially $1/(\sigma^2)$ is not small, each machine will fit some data and become saturated. That is, we are trapped in a local minimum. Hence, we can start from a small $1/(\sigma^2)$ and gradually increase it. Then in final steps when data have been correctly separated, $1/(\sigma^2)$ becomes large so that predictors try to fit different groups of data. Our experience also shows that if we use a fixed σ which in some sense is not too small or too large, the algorithm can also work. Of course the choice of such a σ depends on the smoothness of all functions $f_i, i = 1, \ldots, m$. For example, if a random sample of points on these functions has a high degree of nonlinearity, we are safe to use a large fixed $1/(\sigma^2)$ from the beginning.

The stopping criterion of our algorithm is as follows

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

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

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

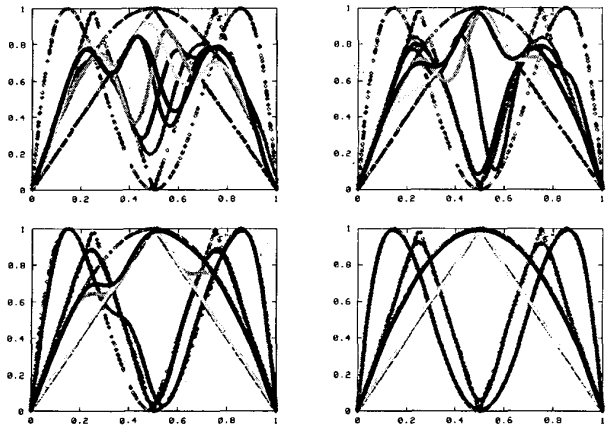

Fig. 1. First four iterations (data without noise)

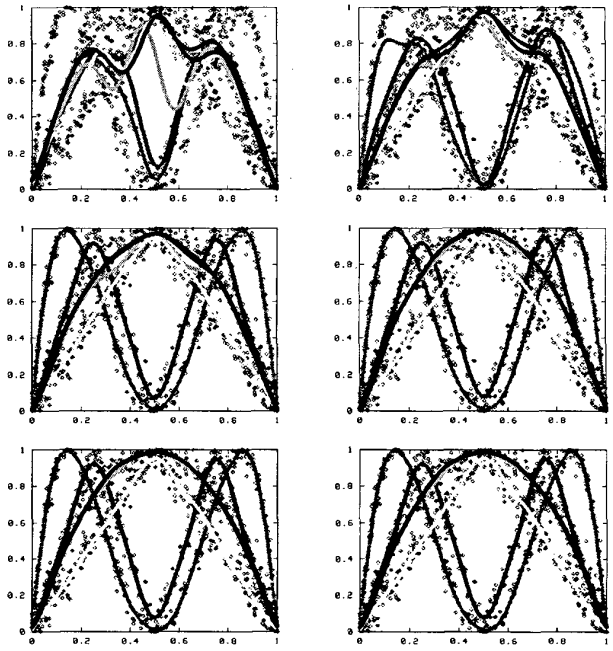

Fig. 2. First six iterations (data with noise)

III. The Adjustment of β

In this section we describe our method for adjusting β, an important parameter which controls the update of p_i^t. From (6) we have that

$$y_t = \hat{f}_i(x_t) + e_i^t, i = 1, \ldots, m, t = 1, \ldots, l.$$

Assume that e_i^t are i.i.d. $N(0, \tau)$.

Define a_i to be the percentage of data in the ith group:

$$a_i \equiv \frac{\#\text{ data with } r_t = i}{l};$$

that is, $a_i = p(r_t = i)$. The log-likelihood function of y is

$$L(\tau) = \sum_{t=1}^{l} \log p(y_t|x_t, \tau),$$

where

$$\begin{aligned} p(y_t|x_t, \tau) &= \sum_{i=1}^{m} p(y_t, r_t = i|x_t, \tau) \\ &= \sum_{i=1}^{m} a_i p_i(y_t|x_t, \tau), \end{aligned} \tag{14}$$

with

$$\begin{aligned} p_i(y_t|x_t, \tau) &\equiv p(y_t|r_t = i, x_t, \tau) \\ &= \frac{1}{\sqrt{2\pi\tau}} \exp\{-(y_t - \hat{f}_i(x_t))^2/(2\tau)\} \\ &= \frac{1}{\sqrt{2\pi\tau}} \exp\{-(e_i^t)^2/(2\tau)\}. \end{aligned} \tag{15}$$

Hence

$$p(y_t, r_t|x_t, \tau) = a_{r_t} p_{r_t}(y_t|x_t, \tau). \tag{16}$$

Let $\hat{\tau}$ be an estimate of τ. Then we can estimate p_i^t by

$$\begin{aligned} \hat{p}_i^t &\equiv p(r_t = i|x_t, y_t, \hat{\tau}) \\ &= \frac{p(y_t, r_t = i|x_t, \hat{\tau})}{\sum_{k=1}^{m} p(y_t, r_t = k|x_t, \hat{\tau})} \\ &= \frac{a_i \exp\{-(e_i^t)^2/(2\hat{\tau})\}}{\sum_{k=1}^{m} a_k \exp\{-(e_k^t)^2/(2\hat{\tau})\}} \end{aligned} \tag{17}$$

using (15) and (16). By comparing (5) and (17), we suggest to choose $1/(2\hat{\tau})$ as our next β. Since $\hat{\tau}$ is 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 (corresponding to the fact that the temperature is decreasing).

Let $\tau^{(g)}$ and $p_i^{t(g)} \equiv p(r_t = i|x_t, y_t, \tau^{(g)})$ be the information of the previous iteration. We shall show how to obtain $\tau^{(g+1)}$. Let $X \equiv (x_1, \ldots, x_l)$, $Y \equiv (y_1, \ldots, y_l)$, and $R \equiv (r_1, \ldots, r_l)$. Denote $Q(\tau, \tau^{(g)})$ as the conditional log-likelihood function of (Y, R) given X, Y, and

τ^g. Then

$$
\begin{aligned}
& Q(\tau, \tau^{(g)}) && (18)\\
\equiv\ & E[\log p(Y, R|X, \tau)|X, Y, \tau^{(g)}] \\
=\ & E[\log \prod_t p(y_t, r_t|x_t, \tau)|X, Y, \tau^{(g)}] && (19)\\
=\ & \sum_{t=1}^{l} E[\log p(y_t, r_t|x_t, \tau)|X, Y, \tau^{(g)}] \\
=\ & \sum_{t=1}^{l} E[\log(a_{r_t} p_{r_t}(y_t|x_t, \tau))|X, Y, \tau^{(g)}] \text{ (from (16))} \\
=\ & \sum_{t=1}^{l}\sum_{i=1}^{m} \log(a_i p_i(y_t|x_t, \tau)) p(r_t = i|X, Y, \tau^{(g)}) \\
=\ & \sum_{t=1}^{l}\sum_{i=1}^{m} \{(\log a_i) p(r_t = i|X, Y, \tau^{(g)})\} \\
& + \sum_{t=1}^{l}\sum_{i=1}^{m} \{[\log p_i(y_t|x_t, \tau)] p(r_t = i|X, Y, \tau^{(g)})\} \\
=\ & \sum_{t=1}^{l}\sum_{i=1}^{m} \{(\log a_i) p_i^{t(g)}\} \\
& + \sum_{t=1}^{l}\sum_{i=1}^{m} \{[\log p_i(y_t|x_t, \tau)] p_i^{t(g)}\}, && (20)
\end{aligned}
$$

where (19) and (20) follow from the independence of each observation. Note that from (15)

$$\log p_i(y_t|x_t, \tau) = \frac{-1}{2}[\log(2\pi) + \log\tau + \frac{(e_i^t)^2}{\tau}]. \quad (21)$$

Let

$$\tau^{(g+1)} = \text{argmax}_\tau Q(\tau, \tau^g).$$

Using (20), simple calculations show that the maximum of (20) occurs at

$$
\begin{aligned}
\tau^{(g+1)} &= \frac{\sum_{t=1}^{l}\sum_{i=1}^{m}\{(e_i^t)^2 p_i^{t(g)}\}}{\sum_{t=1}^{l}\sum_{i=1}^{m} p_i^{t(g)}} \\
&= \frac{\sum_{t=1}^{l}\sum_{i=1}^{m}\{(e_i^t)^2 p_i^{t(g)}\}}{l}, && (22)
\end{aligned}
$$

where

$$
\begin{aligned}
p_i^{t(g)} &= p(r_t = i|x_t, y_t, \tau^{(g)}) \\
&= \frac{a_i \exp\{-(e_i^t)^2/(2\tau^{(g)})\}}{\sum_{k=1}^{m} a_k \exp\{-(e_k^t)^2/(2\tau^{(g)})\}} && (23)
\end{aligned}
$$

can be obtained similarly to (17).

Therefore, at the $(g+1)$st iteration of the implementation, we replace β in (5) by $1/(2\tau^{(g+1)})$. Practically we do not really calculate (23) and use (5) instead. Therefore, we also do not have to worry about a_i, which is unknown in advance.

Because we are using a linear loss function in support vector regression, we feel that in all formulations linear instead of quadratic terms should be used. Therefore, in (5), (22), and (23), all $(e_i^t)^2$ is replaced by $|e_i^t|$. In other words, though the derivation in this section assumes that e_i^t is with a normal distribution, if we consider it to be with a Laplace (double exponential) distribution, we will get results using $|e_i^t|$.

IV. Experiments

A. Four Chaotic Time Series

We test the extreme case of completely overlapping input manifold used in [13]. For all (x_t, y_t), $y_t = f_{r_t}(x_t)$. They consider all $x_t \in [0, 1]$ 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 $f_4 = f_2(f_2(x))$. An illustration of these functions is in Figure 1. It is easily seen that all these functions map x from $[0, 1]$ to $[0, 1]$.

In the beginning we randomly assign p_i^t to be 0 or 1 while keeping the condition $\sum_{i=1}^{m} p_i^t = 1$, $t = 1, \cdots, l$. We set $1/(2\sigma^2)$ of the RBF kernel to be 50. For updating p_i^t, we consider $\Delta = 3$. Following [6], the four series are activated consecutively, each for 100 time steps, giving an overall 400 time steps. We use ten such periods so totally there are 1,200 steps.

For this case the algorithm stops in five iterations. We present the first four in Figure 1 where it can be seen that points are well separated. We assume the number of functions is unknown so we start from six competing SVMs for this case. Our experience indicates that if we use exactly four SVMs, sometimes it may fall into local minima. Thus, using more SVMs may be necessary.

We also consider cases where groups possess different number of data. Our implementation has been able to handle such data with different ratios.

To further test our implementation, we add noise on these four function using $0.1N(0, 0.5)$. The algorithm stops in seven iterations where the first six iterations are in Figure 2.

B. Mackey-Glass Time Series

Similar to earlier results, we also check time series obtained from the Mackey-Glass delay-differential equation [8]:

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

Following earlier experiments, points are selected every six time steps. Sequentially we generate 300 points in

each segment using the order of $t_d = 23, 17, 23, 30$. Thus, totally there are 1,200 point for testing. The embedding dimension is $d = 6$. That is, y_t is the one-step ahead value of six consecutive x_t. For this problem we set $1/(2\sigma^2)$ to be 1. Other settings are the same as the implementation in Section IV-A. Results are in Figure 3 where it can be seen that different segments are well separated.

V. Discussion

For SVM, the number of support vectors directly affects the training and testing time. A zero p_i^t means that α_i^t in (10) is not necessary so the corresponding variables in the dual problem can be removed. However, in theory p_i^t can never be zero due to (5). Thus, we use a threshold 0.01 for removing points with small p_i^t. Then the computational time can be largely reduced.

One difference between SVR and neural networks is the use of ϵ, the width of the insensitive tube in (8). SVR can be smoother and tolerate more noise using appropriate ϵ. In the above case with noise, setting $\epsilon = 0.05$ results in fewer support vectors and less running time.

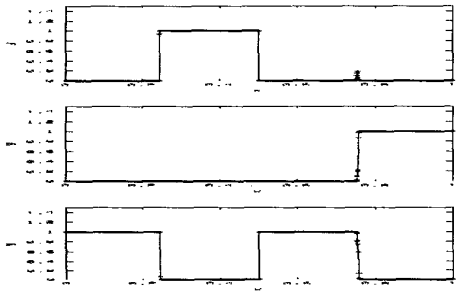

Fig. 3. $p_i^t, i = 1, \ldots, 3$ at each time point t

References

[1] T. W. Cacciatore and S. J. Nowlan. Mixtures of controllers for jump linear and non-linear plants. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 719–726. Morgan Kaufmann Publishers, Inc., 1994.

[2] 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`.

[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 adaptive classification. In S. Haykin and B. Kosko, editors, *Intelligent signal processing*, 2001.

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

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

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

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

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

[10] 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.

[11] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, 1997.

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

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

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

[15] 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.

[16] S. Shi and A. Weigend. Taking time seriously: Hidden markov experts applied to financial engineering. In *CIFEr '97: Proc. of the Conf. on Computational Intelligence for Financial Engineering*, pages 244–252. IEEE, 1997.

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