# 过滤窗最小二乘支持向量机的混沌时间序列预测*

赵永平 1)† 张丽艳 1) 李德才 2) 王立峰 2) 蒋洪章 2)

1) (南京理工大学, 智能弹药国防重点学科实验室, 南京 210094)

2) (国营一二一厂, 牡丹江 157013)

(2012年12月15日收到; 2013年2月15日收到修改稿)

传统的滑动窗策略只是简单且机械地将最远的数据移出窗口, 而将最近的数据移进窗口. 针对这种遗忘策略存在的缺陷, 提出了过滤窗策略. 过滤窗采用 "优胜劣汰" 的选择机制, 将对模型贡献比较大的数据留在窗口当中. 将过滤窗和最小二乘支持向量回归机相结合, 提出了过滤窗最小二乘支持向量回归机. 与滑动窗最小二乘支持向量回归机相比较, 过滤窗最小二乘支持向量回归机具有较小的计算量, 需要较短的窗口长度就能达到和滑动窗最小二乘支持向量回归机几乎相同的预测精度, 而较短的窗口长度又预示着较少的计算量和较好的实时性. 混沌时间序列在线建模和预测的实例表明了过滤窗最小二乘支持向量回归机的有效性和可行性.

**关键词:** 混沌时间序列, 支持向量机, 滑动窗, 过滤窗

**PACS:** 05.45.Tp, 05.45.−a, 02.50.Ey **DOI:** 10.7498/aps.62.120511

## 1 引 言

混沌时间序列是由非线性确定性系统产生的一种介于确定性与随机性之间的非线性动力学现象. 它是有序与无序、确定性与随机性的中间态, 宏观上表现为无序无律的混乱运动以及对初值十分敏感的 "蝴蝶效应", 微观上呈现无穷嵌套的几何自相似性. 不同于随机时间序列, 混沌时间序列具有短期的可预测性和长期的不可预测性 [1,2]. 混沌系统在现实生活中普遍存在, 近年来随着人们对混沌理论研究的不断深入, 它在许多实际系统中得以广泛应用, 如生物医学、声学、化学、通讯、交通、气候天气、电力负荷、财经金融等. 因此, 混沌时间序列的建模和预测已成为混沌信号处理领域中一个非常重要的研究方向. 按照建模原理的不同, 已有混沌时间序列的预测方法有多项式法 [3]、模糊理论法 [4,5]、信息熵法 [6]、回声状态网络法 [7,8]、支持向量机 [9,10] 等. 其中基于结构风险最小化原则的支持向量机 (support vector machine, SVM)[11,12] 因其在小样本的情况下具有较好的泛化能力而备受关注. 用支持向量机来进行混沌时间序列的建模和预测是一种有效的手段, 其中的一个典型代表就是滑动窗 (sliding window, SW) 支持向量机 [13−15], 其原理如图 1 所示.

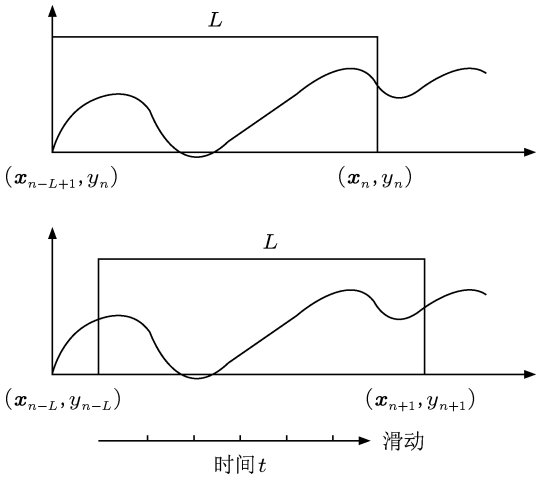


图 1 滑动窗原理示意图

滑动窗支持向量机首先定义一个窗口长度 $L$, 每次都是基于窗口内的数据 $\{(x_i, y_i)\}_{i=n-L+1}^{n}$ 来进行支持向量机的建模, 而窗口外面的数据则完全被

* 国家自然科学基金 (批准号: 51006052) 和南京理工大学 "卓越计划""紫金之星" 资助的课题.

† 通讯作者. E-mail: y.p.zhao@163.com

 

抛弃. 这其实是一种遗忘机制, 也就是说窗口外面的数据被完全遗忘掉. 而当有新的数据 $(\boldsymbol{x}_{n+1}, y_{n+1})$ 要进来时, 将离当前时刻最远的数据 $(\boldsymbol{x}_{n-L}, y_{n-L})$ 挤出窗口, 然后基于新窗口内的数据 $\{(\boldsymbol{x}_i, y_i)\}_{i=n-L}^{n+1}$ 对模型进行调整. 这种窗口更新机制的思想是离当前时刻最远的数据对下一个数据预测的影响比较小. 这种以时间作为衡量标准的窗口更新机制是有局限性, 如用离线学习方法进行混沌时间序列建模时, 在对当前时刻数据进行预测时, 前一刻的数据并没有用来建模, 而同样可以取得比较好的预测效果. 这就说明离当前时刻比较远的数据对混沌时间序列模型的建立是有贡献的, 并且这种贡献并不一定比离当前时刻比较近的数据差.

因此, 本文结合最小二乘支持向量回归机 (least squares support vector regression, LSSVR)[16,17] 提出一种新的窗口更新机制, 即过滤窗 (filtering window, FW), 这种窗口更新机制 "过滤" 数据时并不是以时间作为衡量准则, 而是以对混沌时间序列模型的贡献大小来作为衡量准则. 在这种机制里, 可以把窗口看成是一张 "网", 一个个数据可以看成是一个个 "粒子", "粒度" 的大小取决于数据对混沌时间序列模型的贡献程度, 贡献越大, "粒度" 也就越大, 反之则越小. 不过这张 "网" 的网空是自适应变化的, 目的是保持 "网" 中 "粒子" 的数目不变化. 刚落入 "网" 中的 "粒子" 是否能留在 "网" 中, 以及 "网" 中哪一个 "粒子" 将被漏掉, 完全取决于这些 "粒子" "粒度" 的大小. 这样以来, 在具有相同窗口尺度 $L$ 的情况下, 与滑动窗最小二乘支持向量回归机 (SW-LSSVR) 比较, 过滤窗最小二乘支持向量回归机 (FW-LSSVR) 所建立的混沌时间序列模型具有更好的预测精度, 这也同时意味着在相同混沌时间序列模型预测精度的情况下, FW-LSSVR 需要的窗口长度 $L$ 比较小, 其实时性自然较好. 另外, 即使在相同的窗口尺度下, FW-LSSVR 的模型更新速度也比 SW-LSSVR 快, 即实时性也比较好. 最后, 用一些典型的混沌时间序列例子来验证本文所提算法在混沌时间序列建模和预测中的有效性和可行性.

## 2 最小二乘支持向量回归机

对于回归问题, 假设给定样本集 $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, 其中 $\boldsymbol{x}_i \in \mathbb{R}^l$, $y_i \in \mathbb{R}$. 先用一个非线性映射 $\boldsymbol{\varphi}(\cdot)$ 将数据映射到一个高维特征空间中, 然后在高维特征空间中进行线性回归

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}) + b, \tag{1}$$

其中 $\boldsymbol{w}$ 为特征空间中线性方程的法向量, T 代表矩阵或向量的转置.

为了在最小二乘意义下求解 (1) 式, 在将支持向量回归机中的不等式约束转化为等式约束后, Suykens 等 [16,17] 给出了最小二乘支持向量回归机的数学模型

$$\min_{\boldsymbol{w},\boldsymbol{e}} J(\boldsymbol{w},\boldsymbol{e}) = \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \frac{C}{2}\sum_{i=1}^{N} e_i^2, \tag{2}$$

$$\text{s.t. } y_i = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}_i) + b + e_i \quad (i = 1, \cdots, N),$$

其中 $\boldsymbol{e} = [e_1, \cdots, e_N]^{\mathrm{T}}$, $e_i$ 代表系统测量值与预测值之间的误差, $C \in \mathbb{R}^+$ 为正则化参数, $b$ 为偏置量. 通过构建 Lagrange 函数可将 (2) 式这个约束优化问题转化为求解下列方程组:

$$\begin{bmatrix} 0 & \mathbf{1}_N^{\mathrm{T}} \\ \mathbf{1}_N & \boldsymbol{K} + \boldsymbol{I}/C \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix}, \tag{3}$$

其中 $\boldsymbol{K}$ 为核矩阵, $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}_j)$ 为满足 Mercer 条件的核函数, $\boldsymbol{I}$ 为与 $\boldsymbol{K}$ 同阶的单位矩阵, $\mathbf{1}_N = [1, 1, \cdots, 1]^{\mathrm{T}}$. 由 (3) 式可求得 $b$ 和 $\boldsymbol{\alpha}$. 于是, 可得到最小二乘支持向量回归机的数学模型:

$$y = f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{4}$$

## 3 过滤窗最小二乘支持向量回归机

假设时刻 $t$ 时, 过滤窗中已有 $L$ 个数据, 即 $\{(\boldsymbol{x}_{n_s}, y_{n_s})\}_{s=1}^{L}$, 令集合 $P = \{n_1, \cdots, n_s\}$, 则集合 $P$ 的势 $|P| = L$. 此时过滤窗最小二乘支持向量回归机为

$$y = f^n(\boldsymbol{x}) = \sum_{i \in P} \alpha_i^n k(\boldsymbol{x}_i, \boldsymbol{x}) + b^n, \tag{5}$$

其中 $\boldsymbol{\alpha}_P^n$ 和 $b^n$ 为 (6) 式的解

$$\begin{bmatrix} b^n \\ \boldsymbol{\alpha}_P^n \end{bmatrix} = \boldsymbol{R}^n \begin{bmatrix} 0 \\ \boldsymbol{y}_P \end{bmatrix}, \tag{6}$$

其中

$$\boldsymbol{R}^n = \begin{bmatrix} 0 & \mathbf{1}_{\mathrm{L}}^{\mathrm{T}} \\ \mathbf{1}_{\mathrm{L}} & \boldsymbol{K}_{PP} + \boldsymbol{I}/C \end{bmatrix}^{-1},$$

$$\boldsymbol{\alpha}_P^n = \begin{bmatrix} \alpha_{n_1} \\ \vdots \\ \alpha_{n_s} \end{bmatrix}, \quad \boldsymbol{y}_P = \begin{bmatrix} y_{n_1} \\ \vdots \\ y_{n_s} \end{bmatrix}.$$

### 3.1 “粒度”

为了实现过滤窗的过滤机制, 首先需要给过滤窗中的每一个 “粒子”, 也就是每一个数据定义一个反映其颗粒大小的 “粒度”, 其表达式为

$$\nu_{n_s}=\left|\sum_{i\in P\setminus\{n_s\}}\alpha_i^{(-n_s)}k(\boldsymbol{x}_i,\boldsymbol{x}_{n_s})+b^{(-n_s)}-y_{n_s}\right|, \quad (7)$$

其中 $\alpha_i^{(-n_s)}$ 和 $b^{(-n_s)}$ 为将 (6) 式中与 “粒子”$\boldsymbol{x}_{n_s}$ 相对应的行和列抽取掉后所对应的方程组的解, 即

$$\begin{bmatrix} b^{(-n_s)} \\ \boldsymbol{\alpha}_{P\setminus\{n_s\}}^{(-n_s)} \end{bmatrix}=\begin{bmatrix} 0 & \mathbf{1}^{\mathrm{T}} \\ \mathbf{1} & \boldsymbol{K}_{(P\setminus\{n_s\})(P\setminus\{n_s\})}+\boldsymbol{I}/C \end{bmatrix}^{-1} \times\begin{bmatrix} 0 \\ \boldsymbol{y}_{P\setminus\{n_s\}} \end{bmatrix}, \quad (8)$$

其中

$$\boldsymbol{\alpha}_{P\setminus\{n_s\}}^{(-n_s)}=\left[\alpha_{n_1}^{(-n_s)},\cdots,\alpha_{n_{s-1}}^{(-n_s)},\alpha_{n_{s+1}}^{(-n_s)},\cdots,\alpha_{n_{\mathrm{L}}}^{(-n_s)}\right]^{\mathrm{T}},$$

$$\boldsymbol{y}_{P\setminus\{n_s\}}=\left[y_{n_1},\cdots,y_{n_{s-1}},y_{n_{s+1}},\cdots,y_{n_{\mathrm{L}}}\right]^{\mathrm{T}}.$$

(7) 式定义的 “粒度” 的物理意义: 当将 “粒子”$\boldsymbol{x}_{n_s}$ 从过滤窗中 “漏掉” 后, 用其他 “粒子” 构建的模型来预测 $\boldsymbol{x}_{n_s}$ 时所产生的误差. 此误差 $\nu_{n_s}$ 越小, 说明此 “粒子” $\boldsymbol{x}_{n_s}$ 对过滤窗最小二乘支持向量回归机的贡献越小, 反之则越大. 换句话说, $\nu_{n_s}$ 反映的是用其他 “粒子” 来构建 “粒子”$\boldsymbol{x}_{n_s}$ 时的好坏程度. $\nu_{n_s}$ 越小, 说明其他 “粒子” 可以很好地构建 “粒子” $\boldsymbol{x}_{n_s}$, 则 $\boldsymbol{x}_{n_s}$ 对模型的贡献作用就越小, 可以被 “漏掉”, 也就是说 $\boldsymbol{x}_{n_s}$ 是冗余的.

计算过滤窗中每一个 “粒子” 的 “粒度”, 如果都按照 (8) 式那样直接进行求解的话, 这个计算代价是比较大的, 即 $O(L^4)$, 这对于在线学习算法 FW-LSSVR 来说是一个致命的缺陷. 下面给出一种简便快速计算 “粒子” “粒度” $\nu_{n_s}$ 的方法.

将 (8) 式简记为

$$\begin{bmatrix} b^{(-n_s)} \\ \boldsymbol{\alpha}_{P\setminus\{n_s\}}^{(-n_s)} \end{bmatrix}=\boldsymbol{R}_{P\setminus\{n_s\}}^{n}\begin{bmatrix} 0 \\ \boldsymbol{y}_{P\setminus\{n_s\}} \end{bmatrix}, \quad (9)$$

其中

$$\boldsymbol{R}_{P\setminus\{n_s\}}^{n}=\begin{bmatrix} 0 & \mathbf{1}^{\mathrm{T}} \\ \mathbf{1} & \boldsymbol{K}_{(P\setminus\{n_s\})(P\setminus\{n_s\})}+\boldsymbol{I}/C \end{bmatrix}^{-1}.$$

则 (7) 式变为

$$\nu_{n_s}=\left|\left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\begin{bmatrix} b^{(-n_s)} \\ \boldsymbol{\alpha}_{P\setminus\{n_s\}}^{(-n_s)} \end{bmatrix}-y_{n_s}\right|, \quad (10)$$

其中

$$\boldsymbol{k}_{n_s}=\Big[k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_1}),\cdots,k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_{s-1}}), k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_{s+1}}),\cdots,k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_{\mathrm{L}}})\Big]^{\mathrm{T}}.$$

将 (9) 式代入到 (10) 式中可得

$$\nu_{n_s}=\left|\left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\boldsymbol{R}_{P\setminus\{n_s\}}^{n}\begin{bmatrix} 0 \\ \boldsymbol{y}_{P\setminus\{n_s\}} \end{bmatrix}-y_{n_s}\right|. \quad (11)$$

由于

$$\begin{bmatrix} 0 \\ \boldsymbol{y}_{P\setminus\{n_s\}} \end{bmatrix}=\left((\boldsymbol{R}^n)^{-1}\right)_{(-n_s)}\begin{bmatrix} b^n \\ \boldsymbol{\alpha}_P^n \end{bmatrix}, \quad (12)$$

其中 $\left((\boldsymbol{R}^n)^{-1}\right)_{(-n_s)}$ 为 $(\boldsymbol{R}^n)^{-1}$ 抽去与 “粒子” $\boldsymbol{x}_{n_s}$ 相对应行后的矩阵, 即

$$\left((\boldsymbol{R}^n)^{-1}\right)_{(-n_s)}=\begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & k(\boldsymbol{x}_{n_1},\boldsymbol{x}_{n_1})+1/C & \cdots & k(\boldsymbol{x}_{n_1},\boldsymbol{x}_{n_{\mathrm{L}}}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & k(\boldsymbol{x}_{n_{s-1}},\boldsymbol{x}_{n_1}) & \vdots & k(\boldsymbol{x}_{n_{s-1}},\boldsymbol{x}_{n_{\mathrm{L}}}) \\ 1 & k(\boldsymbol{x}_{n_{s+1}},\boldsymbol{x}_{n_1}) & \vdots & k(\boldsymbol{x}_{n_{s+1}},\boldsymbol{x}_{n_{\mathrm{L}}}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & k(\boldsymbol{x}_{n_{\mathrm{L}}},\boldsymbol{x}_{n_1}) & \cdots & k(\boldsymbol{x}_{n_{\mathrm{L}}},\boldsymbol{x}_{n_{\mathrm{L}}})+1/C \end{bmatrix}. \quad (13)$$

将 (12) 式代入到 (11) 式中, 可得

$$\nu_{n_s}=\left|[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}]\boldsymbol{R}_{P\setminus\{n_s\}}^{n}\left((\boldsymbol{R}^n)^{-1}\right)_{(-n_s)}\begin{bmatrix} b^n \\ \boldsymbol{\alpha}_P^n \end{bmatrix}-y_{n_s}\right|. \quad (14)$$

由于

$$\left((\boldsymbol{R}^n)^{-1}\right)_{(-n_s)}\begin{bmatrix} b^n \\ \boldsymbol{\alpha}_P^n \end{bmatrix}=\left(\boldsymbol{R}_{P\setminus\{n_s\}}^{n}\right)^{-1}\left[b^n,\boldsymbol{\alpha}_{P\setminus\{n_s\}}^{n}\right]^{\mathrm{T}}+\begin{bmatrix} 1 \\ \boldsymbol{k}_{n_s} \end{bmatrix}\alpha_{n_s}^{n}, \quad (15)$$

将 (15) 式代入到 (14) 式中, 可得

$$\nu_{n_s}=\left|\left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\left[b^n,\boldsymbol{\alpha}_{P\setminus\{n_s\}}^{n}\right]^{\mathrm{T}}+\left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\boldsymbol{R}_{P\setminus\{n_s\}}^{n}\begin{bmatrix} 1 \\ \boldsymbol{k}_{n_s} \end{bmatrix}\alpha_{n_s}^{n}-y_{n_s}\right|. \quad (16)$$

由于

$$y_{n_s}=\left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\left[b^n,\boldsymbol{\alpha}_{P\setminus\{n_s\}}^{n}\right]^{\mathrm{T}}$$

$$+(k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_s})+1/C)\alpha_{n_s}^n, \tag{17}$$

将 (17) 式代入到 (16) 式中, 可得

$$\nu_{n_s}=\Big|(k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_s})+1/C) - \left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\boldsymbol{R}_{P\backslash\{n_s\}}^n\begin{bmatrix}1\\ \boldsymbol{k}_{n_s}\end{bmatrix}\Big|\Big|\alpha_{n_s}^n\Big|. \tag{18}$$

由 (21) 式可知,

$$R_{n_sn_s}=\left\{\left(k(\boldsymbol{x}_{n_s},\boldsymbol{x}_{n_s})+1/C\right) - \left[1,\boldsymbol{k}_{n_s}^{\mathrm{T}}\right]\boldsymbol{R}_{P\backslash\{n_s\}}^n\begin{bmatrix}1\\ \boldsymbol{k}_{n_s}\end{bmatrix}\right\}^{-1}, \tag{19}$$

其中 $R_{n_sn_s}$ 为 $\boldsymbol{R}^n$ 中与 "粒子" $\boldsymbol{x}_{n_s}$ 相对应的对角元素. 将 (19) 式代入到 (18) 式中, 可得

$$\nu_{n_s}=\left|\frac{\alpha_{n_s}^n}{R_{n_sn_s}}\right|. \tag{20}$$

若知道了 $\boldsymbol{R}^n$ 和 $\boldsymbol{\alpha}^n$ 后, 根据 (20) 式, 粒度 $\nu_{n_s}$ $(s=1,\cdots,L)$ 自然可以计算出来. 在过滤窗中 "粒子" 的 "粒度" 根据 (20) 式可以很容易计算出来, 随着时间的推移, 当 "粒子" $\boldsymbol{x}_{n+1}$ 到来时, 它的 "粒度" 该如何计算呢? 下面将进行介绍.

## 3.2 $\boldsymbol{x}_{n+1}$ 的 "粒度"

根据 (20) 式, 过滤窗中 "粒子" $\boldsymbol{x}_{n_s}$ $(s=1,\cdots,L)$ 的 "粒度" $\nu_{n_s}(s=1,\cdots,L)$ 很容易求得. 令 "粒子" $\boldsymbol{x}_{s_{\min}}$ 是过滤窗中 "粒度" 最小的 "粒子", 那么 "粒子" $\boldsymbol{x}_{s_{\min}}$ 也是过滤窗中最容易被 "漏掉" 的 "粒子". 如果 "粒子" $\boldsymbol{x}_{n+1}$ 能留在过滤窗中, 则其 "粒度" 应该大于 "粒子" $\boldsymbol{x}_{s_{\min}}$ 的 "粒度". 下面给出在 "粒子" $\boldsymbol{x}_{s_{\min}}$ 漏掉的情况下, 如何计算出 "粒子" $\boldsymbol{x}_{n+1}$ 的粒度.

看下面的 Sherman-Morrison 公式 [18]

$$\begin{bmatrix}\boldsymbol{A} & \boldsymbol{U}\\ \boldsymbol{V} & \boldsymbol{D}\end{bmatrix}^{-1}=\begin{bmatrix}\boldsymbol{A}^{-1}+\boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{D}-\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{D}-\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\\ -(\boldsymbol{D}-\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} & (\boldsymbol{D}-\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\end{bmatrix}, \tag{21}$$

假设 "粒子" $\boldsymbol{x}_{s_{\min}}$ 位于过滤窗的最后边 (不在最后面也没关系, 只需要经过一个简单的位置交换就可以将 "粒子" $\boldsymbol{x}_{s_{\min}}$ 排在最后面), 那么

$$\boldsymbol{R}^n=\begin{bmatrix}\boldsymbol{R}_{|P|}^n & \boldsymbol{r}_{n_{\min}}\\ \boldsymbol{r}_{n_{\min}}^{\mathrm{T}} & r\end{bmatrix}, \tag{22}$$

其中 $\boldsymbol{R}_{|P|}^n$ 为 $\boldsymbol{R}^n$ 的前 $|P|$ 阶方阵. 由 (21) 式可得

$$\begin{aligned}\boldsymbol{R}_{|P|}^n=&\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n+r\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n\begin{bmatrix}1\\ \boldsymbol{k}_{n_{\min}}\end{bmatrix}\\ &\times\begin{bmatrix}1 & \boldsymbol{k}_{n_{\min}}^{\mathrm{T}}\end{bmatrix}\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n,\\ \boldsymbol{r}_{n_{\min}}=&-r\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n\begin{bmatrix}1\\ \boldsymbol{k}_{n_{\min}}\end{bmatrix},\\ r=&\left(k(\boldsymbol{x}_{n_{\min}},\boldsymbol{x}_{n_{\min}})+\frac{1}{C}-\begin{bmatrix}1 & \boldsymbol{k}_{n_{\min}}^{\mathrm{T}}\end{bmatrix}\right.\\ &\left.\times\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n\begin{bmatrix}1\\ \boldsymbol{k}_{n_{\min}}\end{bmatrix}\right)^{-1},\end{aligned} \tag{23}$$

其中

$$\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n=\begin{bmatrix}0 & \boldsymbol{1}_{L-1}^{\mathrm{T}}\\ \boldsymbol{1}_{L-1} & \boldsymbol{K}_{(P\backslash\{n_{\min}\})(P\backslash\{n_{\min}\})}+\boldsymbol{I}/C\end{bmatrix}^{-1}.$$

因此,

$$\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n=\boldsymbol{R}_{|P|}^n-\frac{\boldsymbol{r}_{n_{\min}}\boldsymbol{r}_{n_{\min}}^{\mathrm{T}}}{r}, \tag{24}$$

则

$$\begin{bmatrix}b^{(-n_{\min})}\\ \boldsymbol{\alpha}_{P\backslash\{n_{\min}\}}^{(-n_{\min})}\end{bmatrix}=\boldsymbol{R}_{P\backslash\{n_{\min}\}}^n\begin{bmatrix}0\\ \boldsymbol{y}_{P\backslash\{n_{\min}\}}\end{bmatrix}. \tag{25}$$

由于

$$\boldsymbol{R}_{|P|}^n\begin{bmatrix}0\\ \boldsymbol{y}_{P\backslash\{n_{\min}\}}\end{bmatrix}=\begin{bmatrix}b^n\\ (\boldsymbol{\alpha}^n)_{P\backslash\{n_{\min}\}}\end{bmatrix}-\boldsymbol{r}_{n_{\min}}y_{\min}, \tag{26}$$

将 (24) 和 (26) 式代入到 (25) 式中, 得

$$\begin{aligned}\begin{bmatrix}b^{(-n_{\min})}\\ \boldsymbol{\alpha}_{P\backslash\{n_{\min}\}}^{(-n_{\min})}\end{bmatrix}=&\begin{bmatrix}b^n\\ (\boldsymbol{\alpha}^n)_{P\backslash\{n_{\min}\}}\end{bmatrix}\\ &-\boldsymbol{r}_{n_{\min}}\left(y_{\min}+\frac{\boldsymbol{r}_{n_{\min}}^{\mathrm{T}}}{r}\begin{bmatrix}0\\ \boldsymbol{y}_{P\backslash\{n_{\min}\}}\end{bmatrix}\right),\end{aligned} \tag{27}$$

那么 "粒子" $\boldsymbol{x}_{n+1}$ 的 "粒度" 为

$$\nu_{n+1}=\left|\boldsymbol{k}_{(P\backslash\{n_s\})(n+1)}^{\mathrm{T}}\boldsymbol{\alpha}_{P\backslash\{n_{\min}\}}^{(-n_{\min})}+b^{(-n_{\min})}-y_{n+1}\right|. \tag{28}$$

在计算得到 "粒子" $\boldsymbol{x}_{n+1}$ 的 "粒度" $\nu_{n+1}$ 后, 将比较 $\nu_{n+1}$ 和 $\nu_{n_{\min}}$. 如果 $\nu_{n+1}\leqslant\nu_{n_{\min}}$, 过滤窗口将 "粒

子" $\boldsymbol{x}_{n+1}$ 漏掉, 过滤窗口中的数据和 FW-LSSVR 都将不变化, 继续迎接下一个粒子的到来. 若 $v_{n+1} > v_{n_{\min}}$, 则要将 "粒子" $\boldsymbol{x}_{n_{\min}}$ 漏掉, 将 "粒子" $\boldsymbol{x}_{n+1}$ 保留在过滤窗口中. 由于有 "粒子" 漏掉, 同时又有新 "粒子" $\boldsymbol{x}_{n+1}$ 进入过滤窗口中, 此时将要更新窗口和 FW-LSSVR.

### 3.3 更新 FW-LSSVR

由于漏掉 "粒子"$\boldsymbol{x}_{n_{\min}}$ 后的 FW-LSSVR 可由 (24) 和 (27) 式获得, 同时 $P = P\backslash\{n_{\min}\}$. 下面介绍在捕获 "粒子"$\boldsymbol{x}_{n+1}$ 后如何更新 FW-LSSVR.

根据 (21) 式则下式成立

$$\boldsymbol{R}^{n+1} = \begin{bmatrix} \boldsymbol{R}^n_{P\backslash\{n_{\min}\}} & \boldsymbol{0} \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix} + \lambda \begin{bmatrix} \beta \\ -1 \end{bmatrix} \begin{bmatrix} \beta^{\mathrm{T}} & -1 \end{bmatrix}, \tag{29}$$

其中

$$\beta = R^n_{P\backslash\{n_{\min}\}} \begin{bmatrix} 1 \\ \boldsymbol{k}_{n+1} \end{bmatrix},$$

$$\lambda = \left(k(\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+1}) + 1/C - \begin{bmatrix} 1 & \boldsymbol{k}^{\mathrm{T}}_{n+1} \end{bmatrix} \beta\right)^{-1},$$

$$\boldsymbol{k}_{n+1} = [k(\boldsymbol{x}_{n+1}, \boldsymbol{x}_i), \cdots, k(\boldsymbol{x}_{n+1}, \boldsymbol{x}_j)]^{\mathrm{T}} \quad (i, j \in P),$$

得

$$\begin{aligned} \begin{bmatrix} b^{n+1} \\ \boldsymbol{\alpha}^{n+1} \end{bmatrix} &= \boldsymbol{R}^{n+1} \begin{bmatrix} 0 \\ \boldsymbol{y}_P \\ y_{n+1} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{R}^n_{P\backslash\{n_{\min}\}} & \boldsymbol{0} \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \boldsymbol{y}_P \\ y_{n+1} \end{bmatrix} \\ &\quad + \lambda \begin{bmatrix} \beta \\ -1 \end{bmatrix} \begin{bmatrix} \beta^{\mathrm{T}} & -1 \end{bmatrix} \begin{bmatrix} 0 \\ \boldsymbol{y}_P \\ y_{n+1} \end{bmatrix}. \end{aligned} \tag{30}$$

将 (25) 式代入到 (30) 式中, 可得

$$\begin{aligned} \begin{bmatrix} b^{n+1} \\ \boldsymbol{\alpha}^{n+1} \end{bmatrix} &= \begin{bmatrix} b^{(-n_{\min})} \\ \boldsymbol{\alpha}^{(-n_{\min})}_{P\backslash\{n_{\min}\}} \end{bmatrix} \\ &\quad + \lambda \begin{bmatrix} \beta \\ -1 \end{bmatrix} \left(\beta^{\mathrm{T}} \begin{bmatrix} 0 \\ y_P \end{bmatrix} - y_{n+1}\right), \end{aligned} \tag{31}$$

另外, $P = P \cup \{n+1\}$. 到此为止, FW-LSSVR 更新完毕.

下面对算法 FW-LSSVR 进行总结, 其实现过程如下.

**步骤 1** 初始化核参数和正则化参数 $C$, 选定窗口长度 $L$, 定义索引集合 $P = \emptyset$, 令 $n = 1$, 假设 "粒子" 总个数为 $M$;

**步骤 2** 如果 $n = 1$, 则

$$\begin{bmatrix} b^1 \\ \alpha^1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & k(\boldsymbol{x}_1, \boldsymbol{x}_1) + \dfrac{1}{C} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ y_1 \end{bmatrix},$$

且 $P = P \cup \{1\}$;

**步骤 3** 如果 $n < L$, 则

$$\boldsymbol{R}^{n+1} = \begin{bmatrix} \boldsymbol{R}^n & \boldsymbol{0} \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix} + \lambda \begin{bmatrix} \beta \\ -1 \end{bmatrix} \begin{bmatrix} \beta^{\mathrm{T}} & -1 \end{bmatrix},$$

$$\begin{bmatrix} b^{n+1} \\ \boldsymbol{\alpha}^{n+1} \end{bmatrix} = \begin{bmatrix} b^n \\ \boldsymbol{\alpha}^n \end{bmatrix} + \lambda \begin{bmatrix} \beta \\ -1 \end{bmatrix} \left(\beta^{\mathrm{T}} \begin{bmatrix} 0 \\ y_P \end{bmatrix} - y_{n+1}\right),$$

其中

$$\beta = \boldsymbol{R}^n \begin{bmatrix} 1 \\ \boldsymbol{k}_{n+1} \end{bmatrix},$$

$$\lambda = \left(k(\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+1}) + 1/C - \begin{bmatrix} 1 & \boldsymbol{k}^{\mathrm{T}}_{n+1} \end{bmatrix} \beta\right)^{-1},$$

$$\boldsymbol{k}_{n+1} = [k(\boldsymbol{x}_{n+1}, \boldsymbol{x}_1), \cdots, k(\boldsymbol{x}_{n+1}, \boldsymbol{x}_n)]^{\mathrm{T}},$$

令 $P = P \cup \{n+1\}$, $n = n+1$, 转到步骤 3; 否则转到步骤 4;

**步骤 4** 根据 (20) 式计算当前过滤窗口中 "粒子" 的 "粒度"$\boldsymbol{\nu}_P$, 找出 "粒度" 最小的 "粒子" $\boldsymbol{x}_{n_{\min}}$;

**步骤 5** 根据 (24),(27) 和 (28) 式计算出 "粒子" $\boldsymbol{x}_{n+1}$ 的 "粒度"$v_{n+1}$, 比较 $v_{n_{\min}}$ 和 $v_{n+1}$ 的大小;

**步骤 6** 如果 $v_{n+1} \leqslant v_{n_{\min}}$, 则令 $R^{n+1} = R^n$, $\boldsymbol{\alpha}^{n+1}_P = \boldsymbol{\alpha}^n_P$, $b^{n+1} = b^n$, $\boldsymbol{\nu}^{n+1}_P = \boldsymbol{\nu}^n_P$, $n = n+1$, 如果 $n = M$, 算法终止, 否则转到步骤 5;

**步骤 7** 如果 $v_{n+1} > v_{n_{\min}}$, 根据 (29) 和 (31) 式更新 $R^{n+1}$, $\boldsymbol{\alpha}^{n+1}$ 和 $b^{n+1}$, 令 $P = P \cup \{n+1\}$, $n = n+1$, 如果 $n = M$, 算法终止, 否则转到步骤 4.

## 4 混沌时间序列在线预测实例

为了验证本文所提算法的有效性, 下面以六种典型的混沌时间序列为例进行实验验证. 这六种典型混沌时间序列描述如下.

Kawakami 混沌时间序列:

$$x_{\mathrm{K}}(i+1) = x_{\mathrm{K}}(i)^2 - 0.1x_{\mathrm{K}}(i) - 1.6, \tag{32}$$

其初始值为 0.6.

Heron 混沌时间序列:

$$\begin{cases} x_{\mathrm{H}}(i+1) = 1 - Ax_{\mathrm{H}}(i)^2 + y_{\mathrm{H}}(i), \\ y_{\mathrm{H}}(i+1) = Bx_{\mathrm{H}}(i), \end{cases} \tag{33}$$

其中 $A = 1.37 + 0.05\sin(i/5)$, $B = 0.3$, 初始值为 $\begin{bmatrix} 1 & 0 \end{bmatrix}^{\mathrm{T}}$.

Logistic 混沌时间序列:

$$x_{\mathrm{L}}(i+1) = 4x_{\mathrm{L}}(i)(1 - x_{\mathrm{L}}(i)), \tag{34}$$

其初始值为 0.6.

Lorens 混沌时间序列:

$$\begin{cases} \dfrac{\mathrm{d}x_{\mathrm{Lo}}}{\mathrm{d}t} = -A(x_{\mathrm{Lo}} - y_{\mathrm{Lo}}), \\ \dfrac{\mathrm{d}y_{\mathrm{Lo}}}{\mathrm{d}t} = Cx - x_{\mathrm{Lo}}z_{\mathrm{Lo}} - y_{\mathrm{Lo}}, \\ \dfrac{\mathrm{d}z_{\mathrm{Lo}}}{\mathrm{d}t} = x_{\mathrm{Lo}}y_{\mathrm{Lo}} - Bz_{\mathrm{Lo}}, \end{cases} \tag{35}$$

其中 $A = 16$, $B = 4$, $C = 45.92$, 初始值为 $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^{\mathrm{T}}$.

Chen's 混沌时间序列:

$$\begin{cases} \dfrac{\mathrm{d}x_{\mathrm{C}}}{\mathrm{d}t} = A(y_{\mathrm{C}} - x_{\mathrm{C}}), \\ \dfrac{\mathrm{d}y_{\mathrm{C}}}{\mathrm{d}t} = (C - A)x_{\mathrm{C}} - x_{\mathrm{C}}z_{\mathrm{C}} + Cy_{\mathrm{C}}, \\ \dfrac{\mathrm{d}z_{\mathrm{C}}}{\mathrm{d}t} = x_{\mathrm{C}}y_{\mathrm{C}} - Bz_{\mathrm{C}}, \end{cases} \tag{36}$$

其中 $A = 35, B = 3, C = 28$, 初始值为 $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^{\mathrm{T}}$.

Rossler 混沌时间序列:

$$\begin{cases} \dfrac{\mathrm{d}x_{\mathrm{R}}}{\mathrm{d}t} = y_{\mathrm{R}} + Ax_{\mathrm{R}}, \\ \dfrac{\mathrm{d}y_{\mathrm{R}}}{\mathrm{d}t} = -(x_{\mathrm{R}} + z_{\mathrm{R}}), \\ \dfrac{\mathrm{d}z_{\mathrm{R}}}{\mathrm{d}t} = (y_{\mathrm{R}} - C)z_{\mathrm{R}} + B, \end{cases} \tag{37}$$

其中 $A = 0.2$, $B = 0.2$, $C = 4.7$, 初始值为 $\begin{bmatrix} -1 & 1 & 1 \end{bmatrix}^{\mathrm{T}}$.

对于每一种混沌时间序列, 产生 1000 个样本分别用 SW-LSSVR 和 FW-LSSVR 进行在线建模和预测, 其中嵌入维数 $m = 4$, 延迟时间 $\tau = 1$, 进行一步预测. 模型的核函数选择常用的 Gaussian 核, 即 $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathrm{e}^{-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\gamma^2}}$, 模型的正则化参数 $C$ 和核参数 $\gamma$ 用交叉验证中的留一法 [19] 分别从 $\{2^{-5}, \cdots, 2^{20}\}$ 和 $\{2^{-5}, \cdots, 2^{5}\}$ 中选取. 为了衡量预测的精度, 定义一个性能指标, 即均方根误差 (rooted mean squared errors, RMSE), 其描述为

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2}, \tag{38}$$

其中 $x_i$ 为混沌时间序列真实值, $\hat{x}_i$ 为 $x_i$ 的预测值.

为了便于比较, 在用算法 SW-LSSVR 和 FW-LSSVR 进行混沌时间序列建模和预测时, 窗口长度 $L$ 的变化从 10 开始, 以步长为 10 逐步增长到 300. 图 2 给出的是混沌时间序列预测精度 RMSE 随窗口长度 $L$ 变化的情况 (在这里只给出了前两种混沌时间序列的实验结果, 其他的由于结果类似而予以省略, 以后类同).

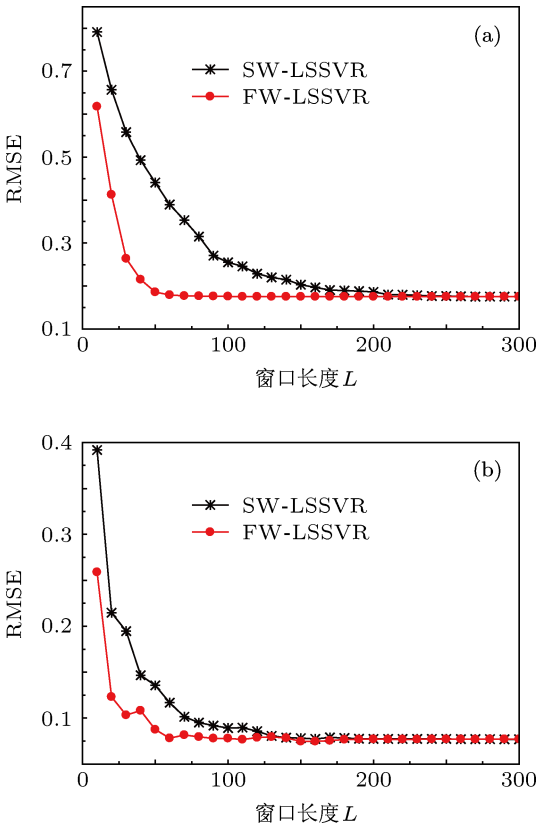


图 2 RMSE 随窗口长度 $L$ 变化的情况 (a) Kawakami; (b) Heron

无论对算法 SW-LSSVR, 还是对 FW-LSSVR, RMSE 都随着窗口长度 $L$ 的增加而逐渐减小, 最后趋于稳定而几乎接近相等, 也就是说混沌时间序列预测的精度随着窗口长度 $L$ 的增加而逐渐提高, 最后这两种算法的预测精度几乎相等. 可是, 相比较而言, FW-LSSVR 预测产生的 RMSE 随窗口长度 $L$ 的增加要减少得更快. 换句话说, FW-LSSVR 以更快的速度接近稳定. 这就会产生要达到相同的预测精度, FW-LSSVR 需要较短的窗口长度 $L$, 也就意味着需要较少的 "粒子", 而较少的 "粒子" 往往寓

意着较短的时间, 较好的实时性. 同时, 这也说明了过滤窗口 “过滤粒子” 的有效性. 图 3 给出了在预测完 1000 个样本后, 过滤窗中 “粒子” 的状态.

从图 3 可以看出, 对于混沌时间序列 Kawakami 来说, 过滤窗中的 “粒子” 大部分集中在前半部分; 对于 Heron 来说, 过滤窗中的 “粒子” 分布比较均匀 (其他混沌时间序列不是像 Kawakami 就是像 Heron, 没有过滤窗中大部分 “粒子” 都集中在后面的情况, 在此予以省略). 无论是哪一种, 过滤窗中的 “粒子” 随着时间的推移更新频率都比较低, 这就是 FW-LSSVR 的计算代价比 SW-LSSVR 低的一个重要原因. 由图 3 可知, 过滤窗中大部分 “粒子” 都分布在极值点, 也就是靠近图形的两边, 这与支持向量机中支持向量的分布类似 [7], 也就是说 FS-LSSVR 把比较重要的 “粒子” 留在了窗口当中, 证明了过滤窗口过滤 “粒子” 的有效性. 另外, 从图 3 中还可以看出, 位于图形中部过滤窗口中的 “粒子” 大部分集中在前面, 这主要是因为后面位于中间部分的粒子由于和前面中间部分的 “粒子” 相似而不能进入到过滤窗中, 这就减轻了过滤窗的更新频率, 减少了计算量, 提高了 FW-LSSVR 的实时性.

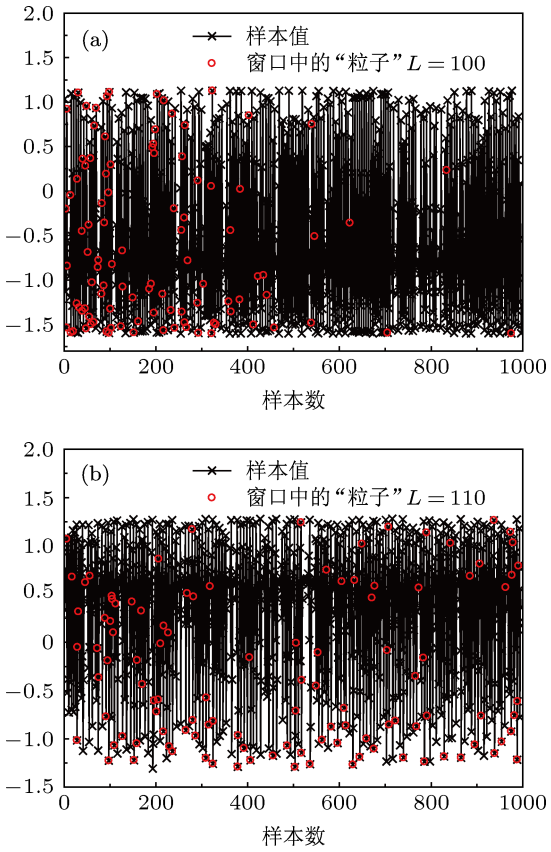


图 3 在预测完 1000 个样本后过滤窗中 “粒子” 的状态
(a) Kawakami; (b) Heron

表 1 对于不同混沌时间序列 FW-LSSVR 和 SW-LSSVR 的比较结果

| 混沌时间序列 | 算法 | 窗口长度 | RMSE | 在线预测时间/s |
|---|---|---|---|---|
| Kawakami | SW-LSSVR | 270 | $1.754\times10^{-1}$ | 8.236 |
| | SW-LSSVR | 100 | $2.550\times10^{-1}$ | 1.978 |
| | FW-LSSVR | **100** | $\mathbf{1.757\times10^{-1}}$ | **1.819** |
| Heron | SW-LSSVR | 290 | $7.706\times10^{-2}$ | 9.411 |
| | SW-LSSVR | 110 | $8.969\times10^{-2}$ | 2.855 |
| | FW-LSSVR | **110** | $\mathbf{7.670\times10^{-2}}$ | **2.422** |
| Logistic | SW-LSSVR | 280 | $9.389\times10^{-2}$ | 8.969 |
| | SW-LSSVR | 80 | $1.634\times10^{-1}$ | 0.491 |
| | FW-LSSVR | **80** | $\mathbf{9.341\times10^{-2}}$ | **0.317** |
| Lorenz | SW-LSSVR | 170 | 1.208 | 3.442 |
| | SW-LSSVR | 80 | 1.225 | 0.506 |
| | FW-LSSVR | **80** | **1.208** | **0.364** |
| Chen’s | SW-LSSVR | 300 | 1.886 | 9.783 |
| | SW-LSSVR | 170 | 1.980 | 3.425 |
| | FW-LSSVR | **170** | **1.886** | **3.163** |
| Rossler | SW-LSSVR | 200 | $4.056\times10^{-1}$ | 5.613 |
| | SW-LSSVR | 130 | $4.110\times10^{-1}$ | 3.119 |
| | FW-LSSVR | **130** | $\mathbf{4.059\times10^{-1}}$ | **2.828** |

从表 1 可以看出, 对同一个混沌时间序列, 当预测精度即 RMSE 趋于稳定时, FW-LSSVR 需要较短的窗口长度, 也就预示着 FW-LSSVR 需要较短的预测时间, 这一点可以从表 1 中每一个混沌时间序列实验的第一行和第三行看出来. 在预测精度几乎相同的情况下, 其预测误差情况如图 4 所示. 从图 4 可以看出, 在预测精度几乎相等的情况下, FW-LSSVR 需要较少的 “粒子”. 另外, 还可以从图 4 中看出, 在开始阶段预测误差较大, 而随着时间序列的推移, 精度趋于稳定. 这主要是因为, 在开始阶段, 算法刚刚启动, 窗口的长度不够, 算法没有足够的学习能力而导致预测误差较大, 而随着窗口长度的增长, 预测误差逐渐减小. 当窗口长度增长到某一个值时, 预测精度趋于稳定, 就不需要增长窗口长度了, 若继续增长, 只会增添算法的计算量, 影响其实时性. 在和 FW-LSSVR 窗口长度相等的情况下, SW-LSSVR 的预测误差如图 5 所示. 从图 5 可以看出, 在窗口长度相同的情况, SW-LSSVR 的预测误差较大, 这一点也可以从表 1 中得到印证. 同时, 这也说明了滑动窗口策略存在的缺陷, 而过滤窗口是有效的.

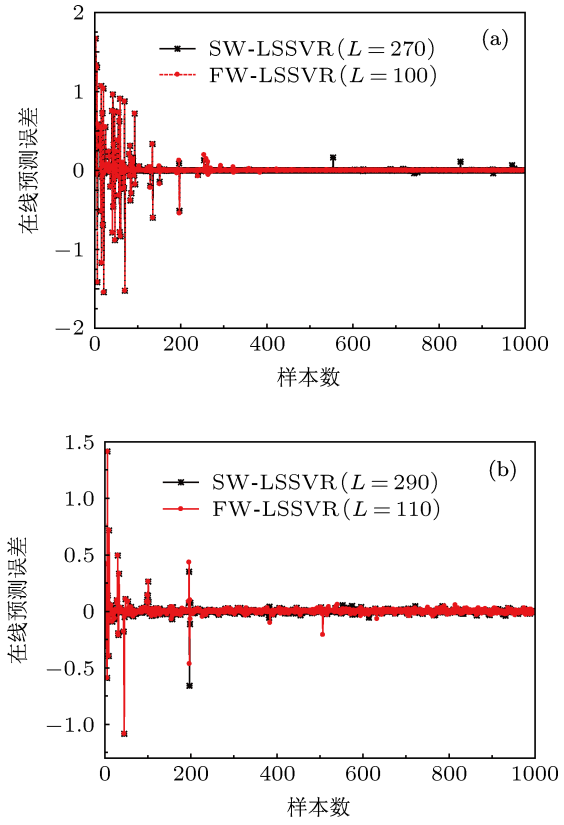


图 4 在预测精度趋于稳定时的预测误差情况 (a) Kawakami; (b) Heron

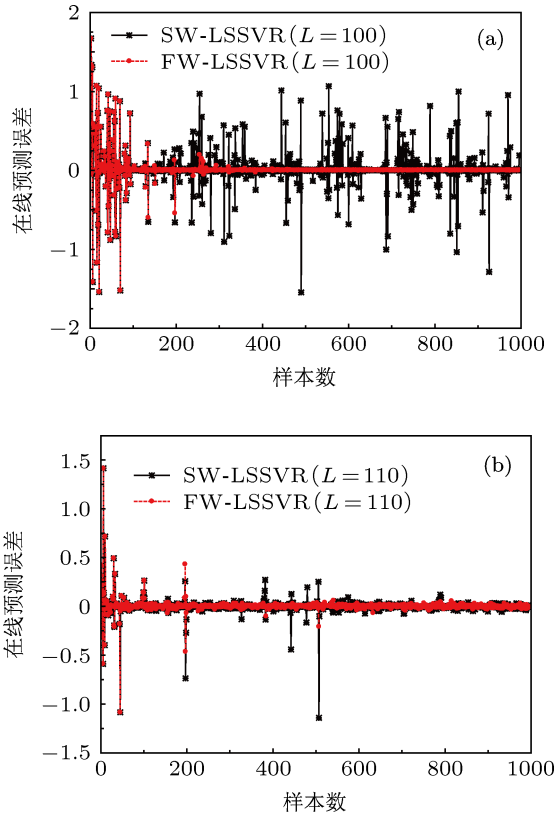


图 5 在相同窗口长度时的预测误差情况 (a) Kawakami; (b) Heron

## 5 结 论

滑动窗在滑动的过程中, 只是简单地将最远的数据给抛弃掉, 将最近的数据移入窗口. 而在建模的过程中, 最远的数据对模型的贡献程度并不一定比最近的差. 滑动窗这种将最远的数据给抛弃掉的做法存在不妥之处. 然而, 过滤窗口则不同, 它就像是张 “网”, 每一个数据就是一个 “粒子”, 同时给每一个 “粒子” 定义一个 “粒度”, 根据 “粒度” 的大小决定哪些 “粒子” 将留在 “网” 中, 哪个 “粒子” 将被遗漏. 新来的 “粒子” 能否留在 “网” 中就看其 “粒度” 的大小, 而不是看时间的远近. 而 “粒度” 就是根据 “粒子” 对模型的贡献程度来定义的, 这样选择出来的 “粒子” 自然对模型的贡献程度比较大, 过滤窗口这种淘汰机制自然也要比滑动窗口简单的遗忘机制要好. 因而, 与滑动窗最小二乘支持向量回归机相比较, 过滤窗最小二乘支持向量机具有较好的实时性和较高的精度.

[1] Zhang X, Wang H L 2011 *Acta Phys. Sin.* **60** 080504 (in Chinese) [张弦, 王宏力 2011 物理学报 **60** 080504]
[2] Cai J W, Hu S S, Tao H F 2007 *Acta Phys. Sin.* **56** 6820 (in Chinese) [蔡俊伟, 胡寿松, 陶洪峰 2007 物理学报 **56** 6820]
[3] Zhou Y D, Ma H, Lü W Y, Wang H Q 2007 *Acta Phys. Sin.* **56** 6809 (in Chinese) [周永道, 马洪, 吕王勇, 王会琦 2007 物理学报 **56** 6809]
[4] Joshi B P, Kumar S 2012 *Cybern. Syst.* **43** 34
[5] Mao J Q, Yao J, Ding H S 2009 *Acta Phys. Sin.* **58** 2220 (in Chinese) [毛剑琴, 姚健, 丁海山 2009 物理学报 **58** 2220]
[6] Zhang C T, Ma Q L, Peng H 2010 *Acta Phys. Sin.* **59** 7623 (in Chinese) [张春涛, 马千里, 彭宏 2010 物理学报 **59** 7623]
[7] Li D, Han M Wang J 2012 *IEEE Trans. Neural Netw. Learn. Syst.* **23** 787
[8] Song T, Li H 2012 *Acta Phys. Sin.* **61** 080506 (in Chinese) [宋彤, 李菡 2012 物理学报 **61** 080506]
[9] Zhang L, Zhou W D, Chang P C, Yang J W, Li F Z 2013 *Neurocomputing* **99** 411
[10] Zhang J F, Hu S S 2008 *Acta Phys. Sin.* **57** 2708 (in Chinese) [张军峰, 胡寿松 2008 物理学报 **57** 2708]
[11] Vapnik V N 1995 *The Nature of Statistical Learning Theory* (New York: Springer)
[12] Vapnik V N 1999 *IEEE Trans. Neural Netw.* **10** 1045
[13] Ye M Y, Wang X D, Zhang H R 2005 *Acta Phys. Sin.* **54** 2568 (in Chinese) [叶美盈, 汪晓东, 张浩然 2005 物理学报 **54** 2568]
[14] Zhang H R, Wang X D 2006 *Chin. J. Comput.* **29** 400 (in Chinese) [张浩然, 汪晓东 2006 计算机学报 **29** 400]
[15] Fan Y G, Li P, Song Z H 2006 *Control Decis.* **21** 1129 (in Chinese) [范玉刚, 李平, 宋执环 2006 控制与决策 **21** 1129]
[16] Suykens J A K, Vandewalle J 1999 *Neural Process. Lett.* **9** 293
[17] Suykens J A K, van Gestel T, de Brabanter J, de Moor B, Vandewalle J 2002 *Least Squares Support Vector Machines* (Singapore: World Scientific)
[18] Zhang X D 2004 *Matrix Analysis and Applications* (Beijing: Tsinghua University Press) (in Chinese) [张贤达 2004 矩阵分析与应用 (北京: 清华大学出版社)]
[19] An S, Liu W, Venkatesh S 2007 *Pattern Recognit.* **40** 2154

# Chaotic time series prediction using filtering window based least squares support vector regression*

Zhao Yong-Ping[1)†] Zhang Li-Yan[1)] Li De-Cai[2)] Wang Li-Feng[2)]
Jiang Hong-Zhang[2)]

1) ( *Zndy of Ministerial Key Laboratory, Nanjing University of Scinece and Technology, Nanjing* 210094, *China* )

2) ( *State Owned 121 Factory, Mudanjiang* 157013, *China* )

( Received 15 December 2012; revised manuscript received 15 February 2013 )

**Abstract**

When the traditional strategy of sliding window (SW) deals with the flowing data, the data far from current position are mechanically and briefly moved out of the window, and the nearest ones are moved into the window. To solve the shortcomings of this forgetting mechanism, the strategy of filtering window (FW) is proposed, in which adopted is the mechanism for selecting the superior and eliminating the inferior, thus resulting in the data making more contributions to the will-built model to be kept in the window. Merging the filtering window with least squares support vector regression (LSSVR) yields the filtering window based LSSVR (FW-LSSVR for short). As opposed to traditional sliding window based LSSVR (SW-LSSVR for short), FW-LSSVR cuts down the computational complexity, and needs smaller window size to obtain the almost same prediction accuracy, thus suggesting the less computational burden and better real time. The experimental results on classical chaotic time series demonstrate the effectiveness and feasibility of the proposed FW-LSSVR.

**Keywords:** chaotic time series, support vector machine, sliding window, filtering window

**PACS:** 05.45.Tp, 05.45.−a, 02.50.Ey **DOI:** 10.7498/aps.62.120511

* Project supported by the National Natural Science Foundation of China (Grant No. 51006052) and the Outstanding Scholar Supported Program of Nanjing University of Scinece and Technology, China.

† Corresponding author. E-mail: y.p.zhao@163.com