Recursive Multikernel Filters Exploiting Nonlinear Temporal Structure

Steven Van Vaerenbergh
Dept. Communications Engineering
University of Cantabria, Spain
steven.vanvaerenbergh@unican.es

Simone Scardapane
DIET Dept.
Sapienza University, Italy
simone.scardapane@uniroma1.it

Ignacio Santamaria
Dept. Communications Engineering
University of Cantabria, Spain
i.santamaria@unican.es

***Abstract*—In kernel methods, temporal information on the data is commonly included by using time-delayed embeddings as inputs. Recently, an alternative formulation was proposed by defining a γ-filter explicitly in a reproducing kernel Hilbert space, giving rise to a complex model where multiple kernels operate on different temporal combinations of the input signal. In the original formulation, the kernels are then simply combined to obtain a single kernel matrix (for instance by averaging), which provides computational benefits but discards important information on the temporal structure of the signal. Inspired by works on multiple kernel learning, we overcome this drawback by considering the different kernels separately. We propose an efficient strategy to adaptively combine and select these kernels during the training phase. The resulting batch and online algorithms automatically learn to process highly nonlinear temporal information extracted from the input signal, which is implicitly encoded in the kernel values. We evaluate our proposal on several artificial and real tasks, showing that it can outperform classical approaches both in batch and online settings.**

I. Introduction

In recent years, kernel adaptive filters (KAF) have become a popular approach for online machine learning and time-series prediction, thanks to numerous theoretical and practical advances [1]–[3]. Differently from deep recurrent neural networks [4], whose training is generally formulated in batch fashion, KAFs can be trained efficiently with a single pass over the training data. Popular examples of KAFs include kernel least-mean-square [2], [5] and kernel recursive least-squares [1], [6]. When operating on temporal data, most KAFs apply common kernel functions, e.g. Gaussian or polynomial, to time-delay embeddings of the input data. Choosing a specific embedding is not trivial in general, and it might be suboptimal in the case of non-stationary signals. These are known problems in other kernel methods as well, such as support vector machines (SVMs) [7] and kernel ridge regression (KRR).

In order to overcome these limitations, several authors have proposed kernelized extensions of classical recursive models, including the recurrent least-squares SVM [8], the autoregressive and moving average (ARMA) SVM [9], the kernel machine and space projection (KMSP) method [10], the kernel γ-filter [11], and, more recently, the kernel adaptive ARMA algorithm [12]. All of these works share a common methodology, which is composed of three major steps: (i) define a proper state-space model (SSM) in the input space; (ii) map the input and/or the state of the model to a high-dimensional feature map corresponding to a properly defined reproducing kernel Hilbert space (RKHS); and (iii) solve the resulting model by substituting all dot products with evaluations of the associated kernel function according to the so-called "kernel trick". Although this is a powerful methodology, it is hard to obtain insights from the model and, more importantly, selecting a proper embedding remains a crucial problem.

In this paper, we focus on the alternative methodology recently proposed in [13]. The basic idea consists in defining the SSM explicitly in a proper RKHS, where samples might not correspond one-to-one with the original inputs. In particular, it is possible to define a proper reproducing theorem such that the model can be expressed as a summation of kernel values, which can be computed *recursively* at each time instant. For the specific case of the γ-filter, the resulting model is particularly appealing because each "tap" in the RKHS corresponds to a filtering operation on a different time-scale of the original input [13]. Unlike previously proposed recursive kernels, e.g. [14], this class of kernels was found to work robustly in many problems, including time-series prediction and array processing. Some theoretical aspects of a related class of kernels were investigated independently in [15].

Here, we are interested in extending this methodology by focussing on a shortcoming of the original model. In particular, standard kernel methods consider a single kernel value for each input datum, while in this case, we obtain several kernel values per input. In [13], this problem was side-stepped by either averaging the values, or by computing inner products. This approach lacks in flexibility, however, because it implicitly assumes that all time-scales are equally important. In this paper, we put forth the idea of considering each kernel value as coming from a different kernel function, and to apply proper adaptive strategies to learn the dependency with respect to each of them. In the literature, the idea of adaptively combining kernel functions goes under the name of multiple kernel (MK) learning. MK algorithms originated in the SVM literature [16], and their benefits have also been proven by a number of authors for KAFs, including the MK normalized LMS [17], the mixture KMLS [18], the doubly regularized MKLMS [19], and Cartesian HYPASS [3].

In order to test the feasibility of our idea, we focus on a simple strategy in which multiple kernel estimators are linearly combined following a stacking-like [20] algorithm.

ISBN 978-0-9928626-7-1

This allows us to test the same formulation in both batch and online settings. In the experimental section, we show that by combining our procedure with the recursive kernel of [13], we obtain a performance that is comparable to or better than competing approaches. To this end, we evaluate several benchmarks using both artificial and real-world datasets.

The rest of the paper is organized as follows. Section II describes the RKHS γ-filter from [13], which is extended through the proposed MK strategy in Section III. We briefly consider computational considerations of the recursive kernel evaluation in Section IV. Experiments are presented in Section V, before giving some conclusive remarks in Section VI.

II. Recursive γ-filtering in RKHS

Let us denote by (x_n, y_n) a generic input-output pair observed at time instant n. We assume these data to be generated by the following nonlinear model

$$y_n = f(\langle w^i, x_n^i \rangle) + e_y \,, \tag{1a}$$
$$x_n^i = g(x_{n-1}^i, x_{n-2}^i, \ldots, y_n, y_{n-1}, \ldots) + e_x \,, \tag{1b}$$

where x_n^i is the input signal at the i-th filter tap, $\langle \cdot \rangle$ denotes the inner product, w^i are the filter weights, $f(\cdot), g(\cdot)$ are smooth nonlinear functions, and e_x, e_y represent state and output noise respectively. The main idea introduced in [13] is to model a similar process, defined instead in a proper Hilbert space $\mathcal{H}$

$$y_n = \hat{f}(\langle w^i, \phi_n^i \rangle_{\mathcal{H}}) + e_y \,, \tag{2a}$$
$$\phi_n^i = \hat{g}(\phi_{n-1}^i, \phi_{n-2}^i, \ldots, y_n, y_{n-1}, \ldots) + e_\phi \,, \tag{2b}$$

where w^i, ϕ_n^i are now samples in the (possibly infinite-dimensional) $\mathcal{H}$, and e_ϕ, e_y represent the state and output noise. Differently from previous works, it is not required for ϕ_n^i to have x_n^i as its preimage, in order to provide more flexibility to the model. Proving a representer theorem in general is not trivial, except for specific instantiations of Eq. (2). A particularly interesting case arises by assuming that the filtering operation in Eq. (2) is a γ-filter [11], [21] given by (omitting noise for simplicity)

$$y_n = \sum_{i=1}^{P} \langle w^i, \phi_n^i \rangle_{\mathcal{H}} \,, \tag{3a}$$
$$\phi_n^i = \begin{cases} \psi(x_n) & \text{if } i = 1 \,, \\ (1-\mu)\phi_{n-1}^i + \mu \phi_{n-1}^{i-1} & \text{if } 2 \leq i \leq P \end{cases} \tag{3b}$$

where $\psi(x_n)$ is some nonlinear transformation in $\mathcal{H}$, P is the filter length controlling the memory depth, and $0 < \mu \leq 1$ is a free parameter controlling stability. It is interesting to observe that each "tap" in the Hilbert space is defined by a recursive equation, so as to work over different temporal combinations of the (nonlinearly transformed) original input sequence. In [13], it is proved that, for a given sequence of length N, the filter weights w^i can be expressed as

$$w^i = \sum_{m=1}^{N} \beta_m^i \phi_m^i \,, \tag{4}$$

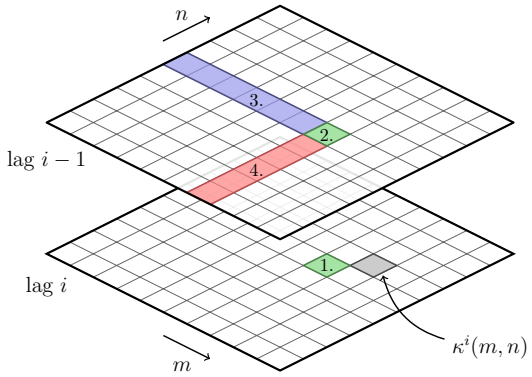

Fig. 1. Calculation of the recursive kernel matrix. The numbers 1 to 4 correspond to the four terms in Eq. (6), $i > 1$.

for some coefficients $\beta_1^i, \ldots, \beta_N^i \in \mathbb{R}$. After substituting (4) in (3), and making use of the kernel definition $\kappa^i(m,n) = \langle \phi_m^i, \phi_n^i \rangle_{\mathcal{H}}$, we obtain

$$\hat{y}_n = \sum_{i=1}^{P} \sum_{m=1}^{N} \beta_m^i \langle \phi_m^i, \phi_n^i \rangle = \sum_{i=1}^{P} \sum_{m=1}^{N} \beta_m^i \kappa^i(m,n) \,. \tag{5}$$

Again, it is worth underlining that, in general, $\kappa^i(m,n) \neq \kappa^i(x_m^i, x_n^i)$. In particular, due to the recursive model definition, the kernel itself can be defined recursively, and the closed-form expression is given by[1]

$$\kappa^i(m,n) = \begin{cases} \kappa(x_m, x_n), & i = 1 \\ \bar{\mu}^2 \kappa^i(m-1, n-1) \\ \quad + \mu^2 \kappa^{i-1}(m-1, n-1) \\ \quad + \mu^2 \sum_{j=2}^{m-1} \bar{\mu}^{j-1} \kappa^{i-1}(m-j, n-1) \\ \quad + \mu^2 \sum_{j=2}^{n-1} \bar{\mu}^{j-1} \kappa^{i-1}(m-1, n-j), & i > 1 \end{cases} \tag{6}$$

where $\bar{\mu} = 1 - \mu$, and $\kappa(x_m, x_n)$ is the classical (scalar) kernel function corresponding to $\langle \psi(x_m), \psi(x_n) \rangle_{\mathcal{H}}$. The calculation of the associated kernel matrix is illustrated in Fig. 1. More generally, we can replace x_m and x_n with the corresponding time-delayed embeddings $\boldsymbol{x}_m$ and $\boldsymbol{x}_n$ to obtain a more expressive model with similar computational complexity.

The recursive kernel γ-filter generalizes several known models, such as the classical γ-filter, recursive auto-regressive filters, and several others (see [13, Section III-C]). Differently from standard KRR and KAF algorithms, which require one kernel value for each time instant, the model structure in (5) requires P kernel values, each of which can be seen as operating at a different time-scale. The filtering approach proposed in [13] did not adapt all coefficients corresponding to all kernel values, but employed a simple combination of

[1] Note that the original formula in [13, Eq. (13)] was missing a summation from $j = 2$ to $n - 1$. The correct formula is given by Eq. (6).

ISBN 978-0-9928626-7-1

the kernels instead. One such combination is the composite kernel $\kappa(m,n) = \frac{1}{P}\sum_{i=1}^{P}\kappa^i(m,n)$. However, this approach has a drawback, namely, it assumes that all kernels are equally significant, and it may lose important information when combining them. In the next section, we will describe a more principled approach to combine the different kernels.

III. Proposed Adaptive Multi-Kernel Approach

We now describe an adaptive formulation that automatically weights the different kernels. The proposed algorithm is relatively inexpensive, and it can be implemented easily in both batch and online settings. Note, however, that nothing prevents the use of more advanced multi-kernel or ensemble strategies to further exploit the multi-kernel structure.

Let us consider the batch case first. Given N training samples (x_n, y_n), denote by $\mathbf{y}$ the vector of all N outputs, and by $\mathbf{K}^i$ the kernel matrix corresponding to the i-th tap in (6). A set of P KRR models is trained as

$$f^i(\mathbf{x}) = \mathbf{y}^T\left(\mathbf{K}^i + c\mathbf{I}\right)^{-1}\boldsymbol{\kappa}^i(x)\,, \tag{7}$$

where $\boldsymbol{\kappa}^i(x) = \left[\kappa^i(x,x_1),\ldots,\kappa^i(x,x_N)\right]$ and c is a regularization constant. We combine the basic models as $h(\mathbf{x}) = \sum_{i=1}^{P}\alpha^i f^i(x)$, where the coefficients $\boldsymbol{\alpha} = [\alpha^1,\ldots,\alpha^P]^T$ are found by minimizing

$$\min_{\alpha^1,\ldots,\alpha^P}\left\{\frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{i=1}^{P}\alpha^i f^i(x_n)\right)^2\right\}\,, \tag{8}$$

which has an immediate closed form solution

$$\boldsymbol{\alpha} = \mathbf{F}^{-1}\mathbf{y}\,, \tag{9}$$

where $\lfloor\mathbf{F}\rfloor_{ni} = f^i(x_n)$. This formulation is a basic form of what is known as "stacking" in the machine learning literature [20], [22]. When $P \ll N$, Eq. (8) does not require regularization. Otherwise, Eq. (8) can be replaced by a more general leave-one-out strategy as in the original stacking problem [20]. More generally, we can include several constraints and/or regularization terms to Eq. (8) in order to force a specific structure on $\boldsymbol{\alpha}$, such as the requirement to lie in a P-simplex (similar to an adaptive combination of filters [23]), or impose an ℓ_1-norm regularization to remove unnecessary lags.

A similar stacking strategy can be applied in the online case. In particular, given the new datum (x_n, y_n), we first update P KAFs in parallel, for instance employing P KLMS filters [5]

$$f_n^i = f_{n-1}^i + \eta\left[y_n - f_{n-1}^i(x_n)\right]\kappa^i(\cdot,x_n)\,, \tag{10}$$

where η is the step-size. Then, we update the current estimate $\boldsymbol{\alpha}_{n-1}$ of the weighting coefficients following an instantaneous descent on (8)

$$\boldsymbol{\alpha}_n = \boldsymbol{\alpha}_{n-1} + \nu\left(y_n - \sum_{i=1}^{P}\alpha^i f_n^i(x_n)\right)\boldsymbol{f}_n(x_n)\,, \tag{11}$$

where ν is a step-size parameter and $\lfloor\boldsymbol{f}_n(\cdot)\rfloor_i = f_n^i(\cdot)$.

Note that KLMS algorithms require calculating *arrays* of kernel evaluations in each iteration, which is feasible by applying the recursive formula discussed in the next section. KRLS algorithms, on the other hand, require calculating entire kernel *matrices*, which is less obvious in recursive settings. This, and other concepts, such as sparsity, require more investigation.

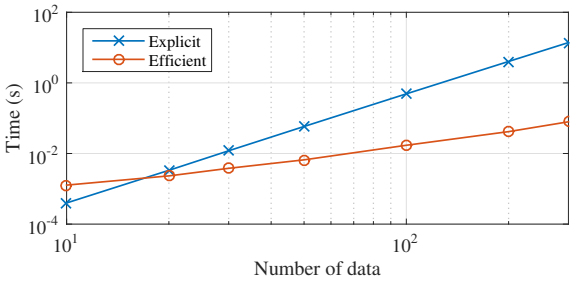

Fig. 2. Execution times for calculating recursive kernel matrices for different amounts of data. Explicit calculation, Eq. (6), vs. efficient computation.

IV. Efficient Computation of the Recursive Kernel

We now briefly outline an implementation to compute the recursive kernel using fast operations on column vectors.

The slowest operations in Eq. (6) are the summations of the third and fourth term. In order to speed up the calculation of the last term, we define the column vector $\boldsymbol{r}_n^i$ with elements

$$\boldsymbol{r}_n^i(m) = \mu^2\sum_{j=2}^{n-1}(1-\mu)^{j-1}\kappa^i(m-1,n-j). \tag{12}$$

This vector can be obtained recursively by observing that

$$\boldsymbol{r}_n^i(m) = (1-\mu)\left(\boldsymbol{r}_{n-1}^i(m) + \mu^2\kappa^i(m-1,n-j-1)\right). \tag{13}$$

A similar recursive calculation is not possible for the third term in Eq. (6), though this term can be obtained efficiently from the elements of the convolution $\boldsymbol{\mu}_{mem} * \boldsymbol{k}_n^i$, where $\boldsymbol{k}_n^i(m) = \kappa^i(m,n-1)$ and $\boldsymbol{\mu}_{mem} = \mu^2[1-\mu,(1-\mu)^2,...,(1-\mu)^N]^\top$.

Fig. 2 compares the computation times of the recursive kernel matrix with $P = 5$, for an explicit implementation of Eq. (6) and the discussed efficient implementation, both in Matlab R2015a on an Intel Core i7 PC with 3.4 GHz processor.

V. Experimental Results

The proposed batch and online algorithms are evaluated on six benchmark problems, three of which are defined on artificially generated datasets and the rest on real-world data.

The first artificial dataset is the Mackey-Glass time-series (with delay 30) taken from [13], denoted as MG30, on which we perform one-step-ahead prediction. The second task is a noisy version of a nonlinear prediction problem introduced in [24], denoted as "Narendra", which is defined by

$$y_n = 0.3y_{n-1} + 0.6y_{n-2} + f(e_n)\,, \tag{14}$$

where the unknown function $f(\cdot)$ has the form

$$f(e) = 0.6\sin(\pi e) + 0.3\sin(3\pi e) + 0.1\sin(5\pi e) \tag{15}$$

and $e_n = \sin((1+a)\omega_0 n)$, $\omega_0 = 2\pi/250$, a is uniformly distributed in the interval $[0.1, 2.9]$, and we set $y_{-1} = y_0 = 1$.

ISBN 978-0-9928626-7-1 © EURASIP 2017

TABLE I
EXPERIMENTAL COMPARISON OF SEVERAL KERNELS IN THE BATCH CASE, INCLUDING A STANDARD RBF KERNEL WITH TIME-DELAYED EMBEDDINGS, THE TWO COMPOSITE RECURSIVE KERNELS FROM [13], AND THE PROPOSED RECURSIVE MULTIKERNEL (RMK) STRATEGIES.

	Standard kernel	**Composite recursive kernel**		**Proposed recursive MK**		
Dataset	RBF	Average	Symmetric	SimpleMKL	Stacking	Sparse Stacking
MG30	-18.99 dB	-23.26 dB	-22.21 dB	$\mathbf{-23.42}$ dB	-20.05 dB	-19.92 dB
Narendra	-14.81 dB	-15.72 dB	-15.09 dB	-15.29 dB	$\mathbf{-17.01}$ dB	-16.78 dB
Wiener	$\mathbf{-18.20}$ dB	-17.58 dB	-17.23 dB	-17.56 dB	-18.19 dB	-18.19 dB
EEG	-5.92 dB	-3.69 dB	-3.60 dB	-6.23 dB	-6.94 dB	$\mathbf{-7.69}$ dB
Respiratory	-18.53 dB	-14.92 dB	-12.58 dB	-16.82 dB	$\mathbf{-18.84}$ dB	-18.64 dB
EUR-USD	-25.10 dB	-23.24 dB	-21.47 dB	—	$\mathbf{-25.83}$ dB	-25.68 dB

We additionally add Gaussian noise with variance 0.1 to the desired output during training.

The third task on artificial data is the identification of the nonlinear Wiener model described in [25], where the input is generated according to

$$x_n = bx_{n-1} + \sqrt{1-b^2}e_x\,, \tag{16}$$

with x_0 randomly generated according to a uniform distribution, e_x is Gaussian noise with variance 0.1, and we set $b = 0.8$. The output is given by first applying a linear filter to an embedding of the last 8 inputs, and then applying a soft nonlinearity on the resulting scalar value.

The first problem on real-world data considers the 4-step ahead prediction of the EEG dataset from [13], extracted from the MIT-BIH Polysomnographic Database[2]. We then consider the prediction of a respiratory motion trace recorded at the Georgetown University Hospital[3], originally described in [26]. Finally, the EUR-USD dataset contains the EUR vs. USD exchange rates in minute intervals, taken on the days January 2nd and 5th of 2009. The task is 2-step ahead prediction.

A. Batch experiments

In all batch experiments, we use 200 elements for training and 1000 separate elements for testing, except for EUR-USD, where we use 1440 samples for training and 1370 for testing, respectively. Parameters are fine-tuned following the same grid-search procedure described in [13], optimizing over a third, independent validation set.

We evaluate several KRR models, trained using (i) a standard RBF kernel with time-delayed embeddings on input, (ii) the recursive kernel with the two composition strategies described in [13] (averaging and symmetrization), and (iii) the stacking procedure described in Section III, both with ℓ_2 and ℓ_1 regularization. Additionally, we consider an extension of our idea by training a support vector regression model with the SimpleMKL algorithm [27], which finds a new kernel via an adaptive combination of the base kernel matrices. Denoting by E^2 the mean-squared error (MSE) over the test set, we evaluate the models using a normalized MSE on a logarithmic scale,

$$\mathrm{nMSE} = 10\log_{10}\left(\mathrm{E}^2/\hat{\sigma}_y^2\right)\,, \tag{17}$$

where $\hat{\sigma}_y^2$ is the empirical variance of the output computed over the test set.

The results for the different benchmarks are given in Table I, where the best result for each dataset is highlighted with a bold font. We observe that the proposed stacking procedures are outperforming the standard KRR models in 4 out of 6 benchmarks, while SimpleMKL achieves slightly better performance in the Mackey-Glass task. To confirm the results, we employ the corrected Friedman test described in [28], according to which the performance of the algorithms are statistically different with a confidence value of $\alpha = 0.05$. A set of Nemenyi post-hoc tests shows that the performance of the base stacking procedure is statistically better than the standard kernel and the two composite recursive kernels, while the sparse stacking procedure is statistically better than the standard kernel and the symmetric recursive kernel, only.

TABLE II
RESULTS FOR ADAPTIVE FILTERING WITH DIFFERENT KERNELS.

Dataset	KLMS	KLMS (RMK)
MG30	-12.50 dB	$\mathbf{-17.99}$ dB
Narendra	-5.65 dB	$\mathbf{-14.55}$ dB
Wiener	-13.42 dB	$\mathbf{-13.68}$ dB
EEG	-5.32 dB	$\mathbf{-7.33}$ dB
Respiratory	$\mathbf{-12.56}$ dB	-11.53 dB
EUR-USD	-21.35 dB	$\mathbf{-25.13}$ dB

B. Online experiments

In a second set of experiments we evaluate the online performance of the described KLMS algorithm with recursive multikernel, and classical KLMS [5], on the six benchmark problems. Both algorithms use the same kernel parameter and learning rate. Table II lists the nMSE results obtained after convergence, indicating clear benefits of the RMK strategy.

[2]https://www.physionet.org/physiobank/database/slpdb/
[3]http://signals.rob.uni-luebeck.de/

ISBN 978-0-9928626-7-1 © EURASIP 2017

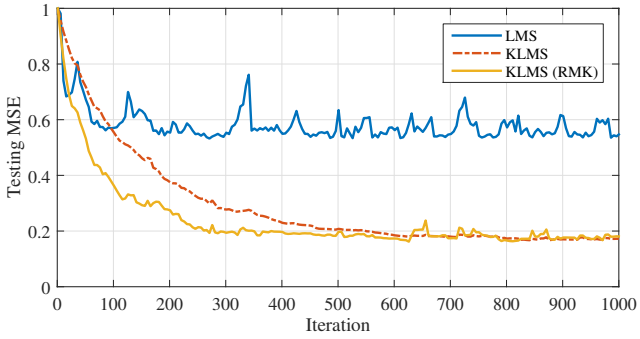

Fig. 3. Learning curves for the channel equalization experiment.

Finally, in Fig. 3 we reproduce the online learning experiment from [5, Sec. 2.11.2]. In this experiment, a binary signal is fed into a nonlinear communications channel, and the goal is to estimate the correct input signal given its output, i.e. to construct a *channel equalizer*. KLMS with RMK employs $P = 5$ lags and $\mu = 0.9$, and the same kernel parameter and learning rate as standard KLMS. While both nonlinear algorithms converge to a similar MSE, the RMK algorithm enjoys a much faster convergence rate.

VI. Conclusions

In this paper, we proposed a novel approach for exploiting multiple time-scale information in kernel regression and filtering problems by combining a previously introduced γ-filter (defined in a proper Hilbert space), with an adaptive strategy for combining kernel functions. In this way, the algorithm automatically adapts to the most significant time-scales of the original input signal. Additionally, the kernel functions are particularly suitable for online processing because they can be recursively computed from previous values, and we briefly discussed on their efficient implementation.

Experimental simulations show that the algorithm has similar or better performance on a wide range of tasks, when compared to several alternative strategies such as time-delayed embeddings of the input. In future works, we plan to further extend our idea by leveraging upon the recent literature on MK filters, and by exploring nonlinear combinations of the different kernels.

Acknowledgment

S. Van Vaerenbergh is supported by the Spanish Ministry of Economy and Competitiveness (under project TEC2014-57402-JIN). S. Scardapane is supported in part by Italian MIUR, "*Progetti di Ricerca di Rilevante Interesse Nazionale*", GAUChO project, under Grant 2015YPXH4W_004.

References

[1] S. Van Vaerenbergh, M. Lázaro-Gredilla, and I. Santamaría, "Kernel recursive least-squares tracker for time-varying regression," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1313–1326, Aug. 2012.

[2] S. Zhao, B. Chen, P. Zhu, and J. C. Príncipe, "Fixed budget quantized kernel least-mean-square algorithm," *Signal Process.*, vol. 93, no. 9, pp. 2759–2770, 2013.

[3] M. Yukawa, "Adaptive learning in Cartesian product of reproducing kernel Hilbert spaces," *IEEE Trans. Signal Process.*, vol. 63, no. 22, pp. 6037–6048, 2015.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[5] W. Liu, P. P. Pokharel, and J. C. Principe, "The kernel least-mean-square algorithm," *IEEE Trans. Signal Process.*, vol. 56, no. 2, pp. 543–554, 2008.

[6] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Sig. Proc.*, vol. 52, no. 8, pp. 2275–2285, 2004.

[7] S. Mukherjee, E. Osuna, and F. Girosi, "Nonlinear prediction of chaotic time series using support vector machines," in *Proc. 1997 IEEE Workshop on Neural Netw. for Signal Process.* IEEE, 1997, pp. 511–520.

[8] J. A. Suykens and J. Vandewalle, "Recurrent least squares support vector machines," *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 7, pp. 1109–1114, 2000.

[9] M. Martínez-Ramón, J. L. Rojo-Alvarez, G. Camps-Valls, J. Muñoz-Marí, E. Soria-Olivas, A. R. Figueiras-Vidal *et al.*, "Support vector machines for nonlinear kernel ARMA system identification," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1617–1622, 2006.

[10] G. Li, C. Wen, W. X. Zheng, and Y. Chen, "Identification of a class of nonlinear autoregressive models with exogenous inputs based on kernel machines," *IEEE Trans. Sig. Proc.*, vol. 59, no. 5, pp. 2146–2159, 2011.

[11] G. Camps-Valls, M. Martínez-Ramón, J. L. Rojo-Álvarez, and E. Soria-Olivas, "Robust γ-filter using support vector machines," *Neurocomputing*, vol. 62, pp. 493–499, 2004.

[12] K. Li and J. C. Príncipe, "The kernel adaptive autoregressive-moving-average algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 334–346, 2016.

[13] D. Tuia, J. Muñoz-Marí, J. L. Rojo-Álvarez, M. Martínez-Ramón, and G. Camps-Valls, "Explicit recursive and adaptive filtering in reproducing kernel Hilbert spaces," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 7, pp. 1413–1419, 2014.

[14] M. Hermans and B. Schrauwen, "Recurrent kernel machines: Computing with infinite echo state networks," *Neural Computation*, vol. 24, no. 1, pp. 104–133, 2012.

[15] M. Mouattamid and R. Schaback, "Recursive kernels," *Analysis in Theory and Applications*, vol. 25, no. 4, pp. 301–316, 2009.

[16] M. Gönen and E. Alpaydın, "Multiple kernel learning algorithms," *J. of Machine Learn. Research*, vol. 12, no. Jul, pp. 2211–2268, 2011.

[17] M. Yukawa, "Multikernel adaptive filtering," *IEEE Trans. Signal Process.*, vol. 60, no. 9, pp. 4672–4682, 2012.

[18] R. Pokharel, S. Seth, and J. C. Principe, "Mixture kernel least mean square," in *2013 Int. Joint Conf. on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–7.

[19] M. Yukawa and R.-i. Ishii, "Online model selection and learning by multikernel adaptive filtering," in *Proc. 21st European Signal Process. Conf. (EUSIPCO)*. IEEE, 2013, pp. 1–5.

[20] L. Breiman, "Stacked regressions," *Machine learning*, vol. 24, no. 1, pp. 49–64, 1996.

[21] J. C. Principe, B. De Vries, and P. G. De Oliveira, "The gamma filter - a new class of adaptive IIR filters with restricted feedback," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 649–656, 1993.

[22] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Machine learning*, vol. 54, no. 3, pp. 255–273, 2004.

[23] J. Arenas-Garcia, L. A. Azpicueta-Ruiz, M. T. Silva, V. H. Nascimento, and A. H. Sayed, "Combinations of adaptive filters: performance and convergence properties," *IEEE Signal Process. Mag.*, vol. 33, no. 1, pp. 120–140, 2016.

[24] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, 1990.

[25] S. Scardapane, M. Scarpiniti, D. Comminiello, and A. Uncini, "Diffusion spline adaptive filtering," in *Proc. 2016 24th European Signal Process. Conf. (EUSIPCO)*. IEEE, 2016, pp. 1498–1502.

[26] F. Ernst, *Compensating for quasi-periodic motion in robotic radiosurgery*. Springer, 2012.

[27] A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet, "SimpleMKL," *J. of Machine Learn. Research*, vol. 9, no. Nov, pp. 2491–2521, 2008.

[28] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. of Machine Learn. Research*, vol. 7, no. Jan, pp. 1–30, 2006.