A recursive least square algorithm for online kernel principal component extraction

João B.O. Souza Filho, Paulo S.R. Diniza

Abstract

The online extraction of kernel principal components has gained increased attention, and several algorithms proposed recently explore kernelized versions of the generalized Hebbian algorithm (GHA) [1], a well-known principal component analysis (PCA) extraction rule. Consequently, the convergence speed of such algorithms and the accuracy of the extracted components are highly dependent on a proper choice of the learning rate, a problem dependent factor. This paper proposes a new online fixed-point kernel principal component extraction algorithm, exploring the minimization of a recursive least-square error function, conjugated with an approximated deflation transform using component estimates obtained by the algorithm, implicitly applied upon data. The proposed technique automatically builds a concise dictionary to expand kernel components, involves simple recursive equations to dynamically define a specific learning rate to each component under extraction, and has a linear computational complexity regarding dictionary size. As compared to state-of-art kernel principal component extraction algorithms, results show improved convergence speed and accuracy of the components produced by the proposed method in five open-access databases.

1. Introduction

Kernel principal components analysis (KPCA) [2] is a simple but powerful nonlinear generalization of the widely used PCA technique [3]. Originally stated as a Gram-matrix eigendecomposition problem [2], thus solvable by classical linear algebra methods [4], this technique faces problems with large scale datasets, for which the computational burden involved in Gram-matrix construction and factorization may turn the extraction process infeasible.

To address these problems, several authors have proposed incremental [5–7] and more recently online kernel component extraction algorithms [8–10]. Some examples are the online kernel Hebbian algorithm (OKHA) [8] and the subset kernel Hebbian algorithm (SubKHA) [9], which are extensions of the kernel Hebbian algorithm (KHA) [6]. In both cases, component estimates are expanded using concise and dynamically built dictionaries, but following different rules for managing them. The kernel Hebbian algorithm (KHA) [6] is a nonlinear extension of the generalized Hebbian algorithm (GHA) [1].

A practical issue faced when using these algorithms is the choice of the learning rate factor, which critically affects the convergence speed and accuracy of extracted components. Usually, such relevant factor is set the same for all components under extraction by trial-and-error or following some metaheuristic procedure. In case of KHA, the work [11] showed that adopting individual learning rates to each component under extraction results in better convergence and accuracy [11]. Nonetheless, the meta-heuristic procedure explored in this work is computationally expensive and requires the adjustment of several experimental parameters. In a similar way, Tanaka [10] has proposed a recursive least-square online kernel PCA extraction algorithm based on the iterative weighted rule (IWR) [12]. The strategy adopted by this rule is somewhat similar to the one proposed in [13] for the extraction of principal components, resulting in learning rates dynamically adjusted to each component under extraction. The accuracy of this algorithm, however, seems to be dependent on the choice of the weighting constants [13].

Motivated by these drawbacks this paper proposes a low-complexity online kernel principal component extraction algorithm, named recursive least square kernel Hebbian algorithm (RLS-KHA). The algorithm

Communicated by Haiqin Yang

Keywords:
- Kernel principal components analysis
- Kernel methods
- Online kernel algorithms
- Machine learning
- Generalized Hebbian algorithm

Electrical Engineering Program (PEE), Alberto Luiz Coimbra Institute (COPPE), Federal University of Rio de Janeiro, Brazil

Electrical Engineering Postgraduate Program (PPEEL), Federal Center of Technological Education Celso Suckow da Fonseca, Avenida Maracanã, 229, Building E, 5th floor, Rio de Janeiro, Brazil

Department of Electronics and Computer Engineering, Polytechnic School, Federal University of Rio de Janeiro, Avenida Athos da Silveira Ramos, 149, Building H, 2nd floor, rooms H-219(20) and 221, Rio de Janeiro, Brazil

Corresponding author at: Department of Electronics and Computer Engineering, Polytechnic School, Federal University of Rio de Janeiro, Avenida Athos da Silveira Ramos, 149, Technological Center, Building H, 2nd floor, rooms H-219(20) and 221, Rio de Janeiro, Brazil.

E-mail addresses: jbfilho@poli.ufrj.br (J.B.O.S. Filho), diniza@smt.ufrj.br (P.S.R. Diniz).

Available online 13 December 2016

http://dx.doi.org/10.1016/j.neucom.2016.12.031

Received 20 May 2016; Received in revised form 23 November 2016; Accepted 8 December 2016

© 2016 Elsevier B.V. All rights reserved.
employs individual learning rates to each component under extraction, automatically tuned using a very simple iterative equation. The proposed RLS-KHA possesses iterative equations similar to KHA, and furthermore demands the setting of only one experimental parameter: the forgetting factor, exhibiting improved accuracy and convergence speed.

2. Kernel PCA

The KPCA [2] is a natural nonlinear extension of the PCA [3]. This technique implicitly produces mappings of realizations of a random vector \( \mathbf{x} \) in a feature space \( \mathbf{F} \) using a nonlinear function \( \Phi(\mathbf{x}) \), which is supposed to satisfy the Mercer theorem [14]. The principal kernel components [2] consists of optimal directions in the sense of representing these mappings in such space. These directions are defined by the eigenvectors, sorted by decreasing order of their corresponding eigenvalues, of the following covariance matrix [2]

\[
C_k = E[\Phi(\mathbf{x})\Phi^T(\mathbf{x})]
\]

where the function \( \Phi(\cdot) \) represents the centered mapping of \( \mathbf{x} \) in the feature space given by

\[
\Phi(\mathbf{x}) = \Phi(\mathbf{x}) - E[\Phi(\mathbf{x})]
\]

and the operator \( E[\cdot] \) designates the expected value.

An elegant result from KPCA theory is that the mapping \( \Phi(\mathbf{x}) \) does not have to be explicitly determined. Assume the vectors \( \mathbf{x}_1, \ldots, \mathbf{x}_N \) defining a set of realizations of \( \mathbf{x} \), and \( \Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_N) \) representing the mappings of such realizations in a feature space. The principal kernel components can be simply stated in terms of linear combinations of these mappings [2] as follows

\[
w_j = [\Phi(\mathbf{x}_1) \cdots \Phi(\mathbf{x}_N)]\sigma_j, \quad 1 \leq j \leq p \in \mathbb{N}
\]

where the vector \( \sigma_j \) are the weighting constants related to the \( j \)-th kernel component. Now assume that the Gram matrix [14] of these realizations is given by

\[
K_N = \begin{bmatrix}
\kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\
\vdots & \ddots & \vdots \\
\kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N)
\end{bmatrix}
\]

where an arbitrary kernel function \( \kappa(\mathbf{x}_a, \mathbf{x}_b) \) (\( 1 \leq a, b \leq N \)) satisfies the following equality \( \kappa(\mathbf{x}_a, \mathbf{x}_b) = \Phi^T(\mathbf{x}_a)\Phi(\mathbf{x}_b) \), i.e., it computes the inner-product of the mappings of the vectors \( \mathbf{x}_a, \mathbf{x}_b \) in the feature space. The vector \( \sigma_j \) corresponds to a scaled-version of the \( j \)-th dominant eigenvector of \( K_N \) as follows

\[
K_N\sigma_j = \lambda_j \sigma_j
\]

\[
\sigma_j = \frac{1}{\sqrt{\lambda_j}} \sigma_j
\]

3. The RLS-KHA

The proposed algorithm explores the least mean-square reconstruction principle (LMSER) [15], combined with an approximate deflation transform, both considering implicit mappings of input data in a feature space. For algorithm deduction, consider the following set of objective functions (\( O_j \))

\[
O_j = E[\|\Phi_j^0(\mathbf{x}) - w_j\Phi_j(\mathbf{x})\|^2], \quad j = 1 \ldots p
\]

where the parameter vector \( \mathbf{w}_j \) should be chosen to minimize the corresponding \( O_j \), and the vector \( \Phi_j(\mathbf{x}) \) corresponds to a deflated [4] version of \( \Phi(\mathbf{x}) \), i.e., it has null projections in the directions of higher-order components \( \mathbf{w}_i, 1 \leq i < j - 1 \). This vector corresponds to

\[
\Phi_j(\mathbf{x}) = \Phi(\mathbf{x}) - \sum_{i=1}^{j-1} \xi_i \Phi_i(\mathbf{x})
\]

with

\[
F_j = I - \sum_{i=1}^{j-1} \xi_i e_i^T
\]

where the vectors \( e_i \) represent the dominant eigenvectors of \( C_k \). As shown in Appendix A, the set of objective functions proposed in Eq. (7) have the \( p \) principal kernel components of \( C_k \) as minimum points.

Now, in order to iteratively estimate the expected value of \( \Phi(\mathbf{x}) \) at an arbitrary iteration \( k \), we will use an exponentially weighted moving average as follows

\[
E[\Phi(\mathbf{x})] \approx \tilde{\Phi}_k = \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\Phi(\mathbf{x}(i))
\]

\[
= (1 - \gamma)\Phi(\mathbf{x}(k)) + \gamma \sum_{i=1}^{k-1} \gamma^{k-i-1}\Phi(\mathbf{x}(i))
\]

\[
= (1 - \gamma)\Phi(\mathbf{x}(k)) + \gamma \tilde{\Phi}_{k-1}
\]

The variable \( \gamma \) is a forgetting factor [16] \( 0 < \gamma < 1 \). In this case, Eqs. (2) and (8) for an input vector \( \mathbf{x}(k) \) can be rewritten as

\[
\tilde{\Phi}(\mathbf{x}(k)) \approx \tilde{\Phi}(\mathbf{x}(k)) = \Phi(\mathbf{x}(k)) - \tilde{\Phi}_k
\]

\[
\tilde{O}_j(k) \approx \tilde{O}_j(k) = \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\|\Phi_j^0(\mathbf{x}(i)) - w_j(k)\Phi_j(\mathbf{x}(i))\|^2
\]

\[
= \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\|\tilde{\Phi}_j(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i))\|^2
\]

\[
= \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\|\Phi_j^0(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i))\|^2
\]

\[
\approx \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\|\Phi_j(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i))\|^2
\]

where

\[
\tilde{\Phi}_j(\mathbf{x}(i)) = \Phi_j^0(\mathbf{x}(i))
\]

The use of this weighted estimator is convenient for non-stationary data, as it defines a time-limited window of reconstruction errors used in component extraction, whose extension is dependent on the value of \( \gamma \) chosen. Besides, it is possible to approximate Eq. (13) by the following equality

\[
\tilde{O}_j(k) \approx \tilde{O}_j(k) = \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\|\Phi_j^0(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i))\|^2
\]

where we are assuming that the projections of input data mapped in the feature space on the directions of kernel component estimates do not vary significantly for this range of iterations, thus \( \tilde{O}_j(k) \approx \gamma^k \). A similar criterion was previously explored for principal subspace component extraction by the PAST algorithm [17] and by Tanaka algorithm for KPCA extraction [10].

The gradient \( \nabla_{w_j(k)} \tilde{O}_j(k) \) is given by

\[
\nabla_{w_j(k)} \tilde{O}_j(k) = 2(1 - \gamma)\sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i} (\tilde{\Phi}_j(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i)))
\]

\[
= \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i}\Phi_j^0(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i)))
\]

where

\[
\xi_j(k) = \sum_{i=1}^{k} (1 - \gamma)\gamma^{k-i} (\tilde{\Phi}_j(\mathbf{x}(i)) - w_j(k)\tilde{\Phi}_j(\mathbf{x}(i)))
\]
A more convenient form to write these equations is given by (see Appendix B)
\[
\mathbf{w}_i(k) = \mathbf{w}_i(k-1) + \eta(k)\tilde{y}_j(k-1, k)[\tilde{\Phi}_j^T(x(k)) - \tilde{y}_j(k-1, k)\mathbf{w}_i(k-1)]
\]
(19)
where
\[
\eta(k) = \frac{1}{\xi_j(k)}
\]
(20)
\[
\xi_j(k) = \tilde{y}_j^2(k) - 1 + \gamma \xi_j(k-1)
\]
(21)
These equations, however, still involve the explicit mapping of the input vector \(x(k)\) in the feature space, which can be avoided by the usage of the kernel trick [14]. For this, consider at the \(k\)th iteration, a dynamically defined dictionary having a set of \(m\) selected input samples as follows
\[
\mathbf{D}_{m,k} = [\mathbf{x}(a) \ldots \mathbf{x}(b)], \quad a \neq b \leq k
\]
(22)
The mappings of these dictionary members in the feature space can be written as
\[
\Psi_{m,k} = [\Phi(x(a)) \ldots \Phi(x(b))]
\]
(23)
The next step corresponds to approximate the mapping of \(x(k)\) in the feature space by a linear combination of matrix \(\Psi_{m,k}\) columns as follows
\[
\Phi(x(k)) \approx \tilde{\Phi}(x(k)) = \Psi_{m,k}\beta_{k,m}
\]
(24)
The vector \(\beta_{k,m}\), that minimizes the square-norm of the error produced in this approximative mapping corresponds to [8]
\[
\beta_{k,m} = \kappa_{m,k}^{-1}\kappa_{m,k}^{-1}
\]
(25)
Here, the matrix \(\kappa_{m,k}\) is the Gram-matrix [14] produced only using dictionary members, i.e.,
\[
\kappa_{m,k} = \begin{bmatrix}
\kappa(x(d_i), x(d_i)) & \cdots & \kappa(x(d_i), x(d_n)) \\
\cdots & \cdots & \cdots \\
\kappa(x(d_n), x(d_i)) & \cdots & \kappa(x(d_n), x(d_n))
\end{bmatrix}
\]
(26)
where the vectors \(d_1, d_2, \ldots, d_n\) correspond to the columns of the matrix \(\mathbf{D}_{m,k}\), and the vector \(\kappa_{m,k}^{-1}\) represents the empirical kernel mapping [14] of \(x(k)\) in this set, given by
\[
\kappa_{m,k}^{-1}(\cdot) = \tilde{\Phi}(x(k)) = [\kappa(x(k), d_1); \cdots; \kappa(x(k), d_n)]
\]
(27)
Note that the arbitrary kernel function denoted as \(k(\cdot, \cdot)\) is supposed to satisfy the Mercer theorem [14]. Similarly to Eq. (24) assume the following mappings
\[
\Psi_{m} = \Psi_{m}\tilde{\beta}_{k,m}
\]
(28)
\[
\tilde{\Phi}(x(k)) = \Psi_{m}\tilde{\beta}_{k,m}
\]
(29)
Combining Eqs. (10), (24) and (28), as well as assuming no dictionary augmentation at iteration \(k\), i.e., \(\Psi_{m,k} = \Psi_{m,k-1}\), the following relation results
\[
\Psi_{m}\tilde{\beta}_{k,m} = (1 - \gamma)\Psi_{m-1}\tilde{\beta}_{k,m} + \gamma \Psi_{m,k-1}
\]
\[
\tilde{\beta}_{k-1,m} = (1 - \gamma)\tilde{\beta}_{k,m} + \gamma \tilde{\beta}_{k-1,m}
\]
(30)
In addition we consider expressing the kernel component estimates using the matrix \(\Psi_{m,k}\) as follows
\[
\mathbf{w}_i(k) = \Psi_{m,k}\mathbf{a}_{k,m}, \quad 1 \leq j \leq \rho
\]
(31)
Using Eqs. (10), (11), (27) and (31), Eq. (14) can be rewritten in the following form
\[
\tilde{y}_j(k-1, k) = \alpha_{j,m}^T(k-1)\Psi_{m,k}\tilde{\Phi}(x(k))
\]
\[
= \alpha_{j,m}^T(k-1)\Psi_{m,k}\tilde{\Phi}(x(k)) - \mathbf{v}_j
\]
\[
= \alpha_{j,m}^T(k-1)[\mathbf{v}_m^T - \mathbf{v}_k]
\]
\[
= \alpha_{j,m}^T(k-1)\tilde{\kappa}_{m,k}
\]
(32)
where
\[
\tilde{\kappa}_{m,k} = \kappa_{m,k} - \kappa_{k}
\]
(33)
\[
\kappa_k = (1 - \gamma)\kappa_{m,k} + \gamma \kappa_{k-1}
\]
(34)
Then, we can use at each iteration an estimate of the matrix \(\mathbf{F}_j\), denoted as \(\tilde{\mathbf{F}}_j(k)\), where the true eigenvectors are approximated by the component estimates produced by the algorithm as follows
\[
\tilde{\mathbf{F}}_j(k) \approx \tilde{\mathbf{F}}_j(k) = \mathbf{I} - \sum_{i=1}^{j-1} \mathbf{w}_i(k-1)\mathbf{w}_i^T(k-1)
\]
(35)
Using Eqs. (11), (12), (14), (28), (29), (31) and (35), the following relations can be deduced
\[
\tilde{\Phi}(x(k)) = \tilde{\mathbf{F}}_j(k)\tilde{\Phi}(x(k)) = \tilde{\Phi}(x(k)) - \sum_{i=1}^{j-1} \mathbf{w}_i(k-1)\mathbf{w}_i^T(k-1)\tilde{\Phi}(x(k))
\]
\[
\Psi_{m,k}\tilde{\beta}_{k,m} = \Psi_{m,k}\left[\tilde{\beta}_{k,m} - \tilde{\beta}_{k,m} - \sum_{i=1}^{j-1} \mathbf{a}_{k,m}(k-1)\tilde{y}_j(k-1, k)\right]
\]
\[
\tilde{\beta}_{k,m} = \tilde{\beta}_{k,m} - \sum_{i=1}^{j-1} \mathbf{a}_{k,m}(k-1)\mathbf{a}_{k,m}(k-1)
\]
(36)
where
\[
\tilde{\beta}_{k,m} = \tilde{\beta}_{k,m} - \tilde{\beta}_{k,m}
\]
(37)
Similarly, using Eqs. (29) and (31) in Eq. (19), it is possible to deduce that
\[
\Psi_{m,k}\mathbf{a}_{k,m}(k) = \Psi_{m,k}\mathbf{a}_{k,m}(k) + \eta(k)\tilde{y}_j(k-1, k)\Psi_{m,k}\tilde{\beta}_{k,m} - \tilde{y}_j(k-1, k)\mathbf{a}_{k,m}(k-1)
\]
\[
\mathbf{a}_{k,m}(k) = \mathbf{a}_{k,m}(k-1) + \eta(k)\mathbf{a}_{k,m}(k)\tilde{\beta}_{k,m} - \tilde{y}_j(k-1, k)\mathbf{a}_{k,m}(k-1)
\]
(38)
Defining the matrix \(\mathbf{A}\) as follows
\[
\mathbf{A}_m(k) = [\mathbf{a}_{1,m}^T(k); \mathbf{a}_{2,m}^T(k); \cdots; \mathbf{a}_{\rho,m}^T(k)]
\]
(39)
the proposed algorithm can be expressed in the following matrix form
\[
\mathbf{A}_m(k) = \mathbf{A}_m(k-1) + \mathbf{A}(k)\tilde{y}(k)\tilde{\Phi}(x(k)) - LT[\tilde{y}(k)\tilde{y}(k)]\mathbf{A}_m(k-1)
\]
(40)
with
\[
\tilde{y}(k) = \mathbf{a}_{m}(k-1)\tilde{\kappa}_{m,k}
\]
(41)
\[
\mathbf{A}(k) = \text{diag}\left[\frac{1}{\xi_1(k)}, \frac{1}{\xi_2(k)}, \ldots, \frac{1}{\xi_\rho(k)}\right]
\]
(42)
where \(\text{LT}[\cdot]\) represents the lower-triangular matrix operator and \(\text{diag}[]\) represents a diagonal matrix, whose entries are determined by the values \(\xi_i(k)\) defined by Eq. (21). This equation can be also rewritten in the following vector form
\[
\tilde{\xi}(k) = \tilde{\xi}(k) + \rho \tilde{\xi}(k-1)
\]
(43)
where the vector \(\tilde{\xi}(k)\) is given by
\[
\tilde{\xi}(k) = [\xi_1(k); \xi_2(k); \cdots; \xi_\rho(k)]
\]
(44)

3.1. Dictionary construction

The proposed RLS-KHA explores the approximative linear dependence (ALD) [18] for dictionary augmentation, i.e., it evaluates the reconstruction error produced by the approximative mapping proposed...
in Eq. (24) as follows
\[ e^2_k = \| \Phi(x) - \Psi_{m+1} \beta_{m+1} \|^2 \] (45)

The value of \( e^2_k \) can be incrementally determined using the value of \( k \) and the variable \( \kappa_{\Psi\Phi x} \).

If this quantity is greater than a user defined threshold, here denoted as \( \nu \), the approximation error is considered unacceptable, resulting in the inclusion of the input vector \( x(k) \) in the dictionary as follows
\[
D_{n+1} = D_{n}, k(x(k))
\] (47)

As a consequence, a new matrix \( \Psi_{m+1,k} \) is defined, corresponding to \( \Psi_{m+1,k} = [\Psi_{m,k} \Phi(x(k))] \)

Dictionary augmentation also results in the update of the following algorithm variables

- **Vector** \( \beta_{m+1} \)
  - Based on Eqs. (24) and (48), we have
  \[
  \hat{\Phi}(x(k)) = \Psi_{m+1,k} \beta_{m+1} = [\Psi_{m,k} \Phi(x(k))] \beta_{m+1}
  \]
  \[
  \beta_{m+1} = \begin{bmatrix} 0_m \\ \end{bmatrix}
  \] (49)
  where the column-vector \( 0_m \) has all \( m \) components equal zero.

- **Vector** \( \bar{\beta}_{k-m+1} \)
  - Similarly to Eq. (49), considering Eq. (28), the following equation results
  \[
  \bar{\beta}_{k-m+1} = \begin{bmatrix} \bar{\beta}_{k-m+1} \\ \end{bmatrix}
  \] (50)

- **Vector** \( \kappa_{m+1} \)
  - Similarly to Eq. (50), we have
  \[
  \kappa_{m+1} = \begin{bmatrix} \kappa_{m} \\ \end{bmatrix}
  \] (51)

- **Vectors** \( \alpha_{m} \)
  - Consider kernel component estimates before and after dictionary augmentation denoted as \( w_{m} \) and \( w_{m+1} \) (1 \( \leq j \leq p \)), respectively. Since dictionary member inclusion should not alter component estimates from the previous iteration, using Eq. (31), the following expression results
  \[
  w_{m+1}(k) = w_{m}(k-1)
  \]
  \[
  \Psi_{m+1,k} \alpha_{m+1}(k-1) = \Psi_{m,k} \alpha_{m}(k-1)
  \]
  \[
  \alpha_{m+1}(k-1) = \begin{bmatrix} \alpha_{m}(k-1) \\ 0 \end{bmatrix}
  \] (52)

Thus, if we consider the matrices \( A_{m}(k-1) \) and \( A_{m+1}(k-1) \) having lines defined by the vectors \( \alpha_{m}(k-1) \) and \( \alpha_{m+1}(k-1) \) (1 \( \leq j \leq p \)), respectively, the following structure results
\[
A_{m}(k-1) = [A_{m}(k-1) \ 0_p]
\] (53)

- **Vector** \( \kappa_{m+1} \)
  - Considering Eqs. (27) and (48), we have
  \[
  \kappa_{m+1} = [\Psi_{m+1,k} \Phi(x(k))] = [\Psi_{m,k} \Phi(x(k))] \Phi(x(k))
  \]
  \[
  \kappa_{m+1} = \begin{bmatrix} k(x(k), x(k)) \\ \end{bmatrix}
  \] (54)

- **Matrix** \( K_{m+1} \)
  - The matrix \( K_{m+1} \) can be incrementally determined using the following formula [8]
  \[
  K_{m+1} = \begin{bmatrix} K_{m} & 0_m \\ 0_m & 0 \end{bmatrix} - \begin{bmatrix} \beta_{m} \\ \end{bmatrix} \begin{bmatrix} \beta_{m} \\ \end{bmatrix} - \begin{bmatrix} \bar{\beta}_{k-m+1} \\ \end{bmatrix}
  \] (55)

In synthesis, the RLS-KHA involves three main conceptual phases: dictionary evaluation, dictionary augmentation and the update of kernel component estimates. Table 1 summarizes the algorithm for convenience, where separate boxes emphasize the steps corresponding to each phase. In the first phase, the error produced by the approximate mapping of the current input vector is calculated, which involves \( O(m^2) \) operations, mainly due to step 5. The second phase corresponds to the update of both dictionary and algorithm variables, requiring \( O((m+1)^2) \) operations in step 9. Lastly, the update of kernel component estimates requires \( O(mp) \) and \( O \left( \frac{m^2}{2} \right) \) operations in steps 21 and 24, respectively. If \( m < \frac{2}{5} \), the number of algorithm operations in this phase becomes linear with respect to dictionary size. In this case, the remaining phases dominate the overall computational cost of the algorithm.

### Table 1
The RLS-KHA algorithm.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define the subspace distance threshold limit (( \nu )), the number of components to be extracted (( p )), and the forgetting factor (( \gamma )).</td>
</tr>
<tr>
<td>2</td>
<td>Initialize ( m = 1 ), ( K_{1} = 1/4(\beta_{1}, x(1)) ), ( D_{1} = {x(1)} ), ( \beta_{1} = 0 ), ( \epsilon_{1} = 0 ), ( \xi(1) = 0_{p} ), and ( A = {a_{1}; a_{2}; \ldots; a_{p}} ) where the scalars ( a_{1}, \ldots, a_{p} ) are small-random values.</td>
</tr>
<tr>
<td>3</td>
<td>for all ( k \geq 2 ) do</td>
</tr>
<tr>
<td></td>
<td>Evaluating a dictionary member candidate</td>
</tr>
<tr>
<td>4</td>
<td>( \kappa_{m+1} = [\kappa(x(k), d_{1}); \kappa(x(k), d_{2}); \ldots; \kappa(x(k), d_{p})] ) where ( d_{i} ) is the ( i )-th column of ( D_{k} ) at step 27.</td>
</tr>
<tr>
<td>5</td>
<td>( \beta_{k,m} = K_{m} \kappa_{m+1} ) (25)</td>
</tr>
<tr>
<td>6</td>
<td>( \epsilon_{k} = [\kappa(x(k), x(k)) - \beta_{k,m}^{T} \kappa_{m+1}] ) (46)</td>
</tr>
<tr>
<td></td>
<td>if ( \epsilon_{k} \geq \nu ) then</td>
</tr>
<tr>
<td>8</td>
<td>Including a dictionary member</td>
</tr>
<tr>
<td>9</td>
<td>( m = m + 1 )</td>
</tr>
<tr>
<td>10</td>
<td>( \beta_{k,m} = K_{m} \kappa_{m+1} ) (37)</td>
</tr>
<tr>
<td>11</td>
<td>( \beta_{k,m} = [\beta_{k,m} - \beta_{k,m} - \beta_{k,m} - \beta_{k,m}] ) (49)</td>
</tr>
<tr>
<td>12</td>
<td>( D_{k+1} = D_{k} + x(k) ) (47)</td>
</tr>
<tr>
<td>13</td>
<td>( A_{m}(k-1) = A_{m+1}(k-1) ) (53)</td>
</tr>
<tr>
<td>14</td>
<td>( \kappa_{m+1} = [\kappa(x(k), x(k))] ) (54)</td>
</tr>
<tr>
<td>15</td>
<td>( \kappa_{m+1} = [\kappa(x(k), x(k))] ) (51)</td>
</tr>
<tr>
<td></td>
<td>end if</td>
</tr>
<tr>
<td>16</td>
<td>RLS-KHA update equations</td>
</tr>
<tr>
<td>17</td>
<td>( \beta_{k,m} = (1 - \gamma) \beta_{k,m} + \gamma \beta_{k,m} ) (30)</td>
</tr>
<tr>
<td>18</td>
<td>( \beta_{k,m} = \beta_{k,m} - \beta_{k,m} ) (37)</td>
</tr>
<tr>
<td>19</td>
<td>( \beta_{k,m} = (1 - \gamma) \kappa_{m} + \gamma \kappa_{m} ) (34)</td>
</tr>
<tr>
<td>20</td>
<td>( \kappa_{m+1} = K_{m} \kappa_{m} - \kappa_{m} ) (33)</td>
</tr>
<tr>
<td>21</td>
<td>( \kappa_{m+1} = A_{m}(k-1) \kappa_{m+1} ) (41)</td>
</tr>
<tr>
<td>22</td>
<td>( \kappa_{m+1} = \kappa_{m} - 0 ) (43)</td>
</tr>
<tr>
<td>23</td>
<td>( \kappa_{m+1} = \kappa_{m} - \kappa_{m} ) (42)</td>
</tr>
<tr>
<td>24</td>
<td>( A_{m}(k) = A_{m}(k-1) + A_{m}(k-1 + A_{m}(k-1) \beta_{k,m}^{T} \kappa_{m+1} - \text{LT}(\kappa(k) \beta_{k,m}^{T}) A_{m}(k-1) ) (40)</td>
</tr>
<tr>
<td></td>
<td>end for</td>
</tr>
</tbody>
</table>

where the vector \( \beta_{k,m} \) and the variable \( \epsilon_{k} \) are given by Eqs. (25) and (46), respectively.
3.2. Relations to other KPCA extraction methods

Most of the recently proposed online kernel PCA extraction algorithms exploit a common framework, involving the following phases: input evaluation, dictionary management, and the update of kernel component estimates. In the first, the algorithms evaluate if the current input should become a dictionary member. Regarding this aspect, the proposed RLS-KHA and the OKHA are similar, since both methods employ the ALD, whereas the Tanaka algorithm considers the coherence criterion [19]. Unlike ALD and OKHA, the SubKHA includes all input data vectors into the dictionary until a maximum number of members defined by the user is reached.

Relative to member inclusion, the dictionary update phase follows a very similar approach in all algorithms. A reduced computational cost formula (Eq. (55)) produces the inverse of the Gram matrix of dictionary members. The exception is the SubKHA, which exchanges the least representative dictionary member by the current input when the dictionary is full and a replacement criterion based on ALD is met. Note that the OKHA, the Tanaka algorithm, and the RLS-KHA do not replace dictionary members.

Regarding the iterative equations for kernel component extraction, the OKHA and the SubKHA exploit kernelizations of the GHA algorithm, employing the same user defined learning rate for all components under extraction, usually maintained fixed or adjusted according to some heuristic procedure. The proposed RLS-KHA and Tanaka algorithm use an individual learning rate to each component under extraction, dynamically tuned, resulting in impressive gains in convergence speed.

Concerning the last phase, Tanaka algorithm minimizes a weighted cost function (IWR), initially proposed for generalized eigendecomposition in [12], whereas the proposed RLS-KHA considers the minimization of multiple simple objective functions following the LMSER principle, coupled by approximated deflations of input data based on component estimates. The latter method involves simpler extraction equations than the former and produces more accurate results, as observed in our simulations. Besides, analyzing Equations (40) and (41), if we ignore the process of centering the input data mapped in the feature space, the RLS-KHA assumes similar extraction equations to KHA, OKHA, and SubKHA. Thus, for this particular case, the proposed RLS-KHA solves an open problem stated in [8], which is the determination of an optimal learning rate for the OKHA algorithm. Table 2 summarizes the main characteristics of the algorithms here mentioned.

### Table 2
Main characteristics of some online kernel algorithms.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Algorithm</th>
<th>OKHA</th>
<th>RLS-KHA</th>
<th>Tanaka</th>
<th>SubKHA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary evaluation</td>
<td>ALD</td>
<td>Coherence</td>
<td>None until dictionary is full</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dictionary management</td>
<td>Just inclusion</td>
<td>Inclusion and exclusion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning rate</td>
<td>User defined</td>
<td>Automatically tuned</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm derivation</td>
<td>GHA</td>
<td>Particular</td>
<td>IWR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 3
Datasets and extraction parameters used in the evaluation of the methods (see text).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>N</th>
<th>d</th>
<th>e</th>
<th>p</th>
<th>ne</th>
<th>ep</th>
</tr>
</thead>
<tbody>
<tr>
<td>USPS</td>
<td>300</td>
<td>8</td>
<td>703.2 kB</td>
<td>8</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>ISOLET</td>
<td>7797</td>
<td>917</td>
<td>463.8 MB</td>
<td>10</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>MNIST</td>
<td>22008</td>
<td>784</td>
<td>3.61 GB</td>
<td>8.5</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>COREL</td>
<td>68040</td>
<td>33</td>
<td>34.55 GB</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>COVER</td>
<td>581012</td>
<td>54</td>
<td>2.46 TB</td>
<td>1.0</td>
<td>30</td>
<td>4</td>
</tr>
</tbody>
</table>

For such cases, we recommend values of $\gamma$ in the range of $0.9995$. Nonetheless, lower values of $\gamma$ may be employed in non-stationary environments, speeding up algorithm convergence. Clearly, the requirements of convergence speed and accuracy of the target application dictate the choice of the factor $\gamma$. Motivated by USPS results, we adopted the value of $\gamma$ equal to 0.999 for all remaining datasets.

Note that in our experiments involving the proposed RLS-KHA, dataset vectors were randomly sampled, configuring a stationary data process. For such cases, we recommend values of gamma in the range of 0.99–0.9995. Nonetheless, lower values of $\gamma$ may be employed in non-stationary environments, speeding up algorithm convergence.
Table 4

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RLS-KHA/OKHA (v)</th>
<th>Tanaka (β)</th>
<th>SubKHA (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>USPS-FD</td>
<td>0.01</td>
<td>1.0000</td>
<td>300</td>
</tr>
<tr>
<td>USPS-SD</td>
<td>0.25</td>
<td>0.7420</td>
<td>49</td>
</tr>
<tr>
<td>ISOLET</td>
<td>0.62</td>
<td>0.4380</td>
<td>71</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.43</td>
<td>0.5515</td>
<td>90</td>
</tr>
<tr>
<td>COREL</td>
<td>0.47</td>
<td>0.5900</td>
<td>50</td>
</tr>
<tr>
<td>COVERTYPE</td>
<td>0.50</td>
<td>0.5750</td>
<td>156</td>
</tr>
</tbody>
</table>

The quality extraction function proposed in [9], here named as empirical square reconstruction error (ESRE), was the figure of merit employed in our analysis. Ignoring the centering of input data mappings in the feature space, we can express this function for an arbitrary iteration k as

\[ ESRE(k) = \frac{1}{N} \sum_{i=1}^{N} [x_i(x_i - 2y_i^T(k)y_i(k) + t_i^T(k)K_{m}t_i(k))] \]  

with

\[ y_i(k) = A_i(k)x_i^n \]  

\[ t_i(k) = A_i(k)y_i(k) \]  

In this formula, the estimates of kernel components define the columns of the matrix \( A \), the variable \( N \) corresponds to the cardinality of the evaluation set, and dataset vectors are denoted as \( x_i \) (\( 1 \leq i \leq N \)).

The performance plots for method comparison considered the average value of ESRE at each iteration. We also evaluated the average (\( \bar{\mu} \)) and standard deviation (\( \bar{\sigma} \)) of the steady-state values attained by each method, reported as \( \bar{\mu} \pm \bar{\sigma} \). To identify statistically significant differences on method performances, we employed the non-parametric Kruskal-Wallis (KW) test [25], assuming a significance level of \( \alpha = 10\% \), and the Tukey’s HSD [25] post-hoc test. We conducted the statistical analysis as follows: if the KW test returned a \( p \)-value lower than 0.1, the differences on the results are supposed to be significant, and we analyzed the \( p \)-value associated with each pairwise comparison, refusing the hypothesis of similar performance (null hypothesis) for all pairs satisfying \( p < 0.01 \).

More details regarding the datasets and conducted experiments are presented in the following.

4.1. USPS dataset

The USPS dataset is composed of grayscale handwritten digit images with size 16×16, coded using integer numbers in the range of 0–255. We reproduced an experiment conducted in [11], taking the first 100 images related to the digits 1–3, and considering the same values for the kernel width and the number of the components adopted in [11]. The selected images were converted to column-vectors with dimension 256, normalized to be in 0 to 1 range, and presented to the algorithms.

Initially, we examined the effect of the forgetting factor choice on the convergence behavior of the proposed RLS-KHA. Figs. 1 and 2 summarize the experiments, which considered values of \( \gamma \) ranging from 0.985 to 0.9995, as well as full and small dictionaries. The proposed algorithm became unstable in some trials when values of \( \gamma \) lower than 0.985 were adopted.

The use of higher values of \( \gamma \) resulted in slower convergence, but improved accuracy, as can be observed by the reduction in the steady-state ESRE. The mean value attained by the proposed RLS-KHA for the full dictionary case was \( 0.1670 \pm 0.0025 \) (\( \gamma = 0.985 \)) and \( 0.1415 \pm 0.0011 \) (\( \gamma = 0.995 \)), whereas \( 0.1909 \pm 0.0054 \) (\( \gamma = 0.985 \)) and \( 0.1768 \pm 0.0054 \) (\( \gamma = 0.995 \)), when considering the small dictionary. In both cases, KW test confirmed statistically significant differences in algorithm performances (\( \chi^2 > 57.45 \), \( p < 0.001 \)). Notice that a good trade-off between accuracy and convergence speed was achieved using \( \gamma = 0.999 \). This factor produced statistically similar steady-state ESRE values as for \( \gamma = 0.995 \) (\( 0.9995 \)) (\( p > 0.05 \)) and \( \gamma = 0.9995 \) (\( p = 0.5555 \)) for small and full dictionaries, respectively.

In the sequence, we compared the proposed RLS-KHA using \( \gamma = 0.999 \) with other KPCA methods for both dictionary sizes. Figs. 3 and 4 depict the results. The differences observed in steady-state ESRE were relevant (\( \chi^2 > 99.44 \), \( p < 0.001 \)). Besides, the use of fixed learning rates resulted in slightly better results than exponentially decaying ones for both dictionary sizes.

In the case of full dictionary, OKHA and SubKHA exhibited superimposed curves, as expected. Regarding online methods, Tanaka converged faster than the proposed RLS-KHA until iteration 6,000, but attained a highly biased steady-state ESRE (0.1815 ± 0.0001). As the KHA employs all dataset members since the beginning of the extraction, this method initially exhibited higher values of ESRE. However, after 200 iterations, the KHA was the fastest method, being outperformed by the proposed RLS-KHA at the iteration 20,000. The RLS-KHA also attained
the lowest mean steady-state ESRE \((0.1417 \pm 0.0017)\), performing equivalently \((p=0.1049)\) to OKHA/SubKHA with exponentially decaying learning rate \((D)\) and better than the remaining techniques \((p < 0.006)\). Moreover, the proposed RLS-KHA converged six times faster \((45,000 \text{ iterations})\) than OKHA and SubKHA in this experiment.

Concerning small dictionary results, the SubKHA initially behaved similar to OKHA, however, it attained the highest mean steady-state ESRE \((0.2516 \pm 0.0062)\), similarly to Tanaka algorithm \((0.2101 \pm 0.0047, p = 0.1049)\). The values attained by the proposed algorithm, OKHA \((D)\), and OKHA \((F)\) were \(0.1768 \pm 0.0054, 0.1770 \pm 0.0054,\) and \(0.1776 \pm 0.0045\), respectively, practically the same \((p > 0.9934)\). Moreover, the proposed RLS-KHA converged in about 35,000 iterations, performing approximately 4.6 times faster than OKHA \((F)\).

### 4.2. ISOLET dataset

The ISOLET dataset \([21]\) is composed of 617 features extracted from voice signals produced by 150 subjects pronouncing all alphabet letters twice \([26]\). Simulations considered both training and testing archives \((\text{isolet1 to isolet5})\), involving 7,797 examples. Based on trials applying classical KPCA extraction on subsamples of this dataset, we defined the kernel width and number of components. The choice of such parameters envisaged to produce a consistent energy curve \([3]\) and to result in a set of kernel components retaining at least 80% from the energy \([11]\) from data (implicitly) mapped in the feature space. Results are summarized in Fig. 5.

According to KW test, the methods performed differently \((\chi^2 = 28.74, p < 0.001)\), but the results were more similar due to the lower \(\chi^2\) attained. Up to the iteration 6000, the best performing technique was the Tanaka algorithm. The proposed RLS-KHA outperformed the OKHA, SubKHA, and Tanaka algorithm at iterations 3600, 5000, and 6000, respectively. The replacement of dictionary members in SubKHA \((EN)\) resulted in a slight improvement in convergence speed, but the steady-state values were equivalent \((p=0.8768)\) to SubKHA \((DS)\). Our method achieved the lowest mean steady-state ESRE \((0.4986 \pm 0.0090)\), performing better than OKHA \((0.5500 \pm 0.0284, p < 0.001)\) and SubKHA \((DS)\) \((0.5169 \pm 0.0065, p = 0.6204)\), and equivalently to Tanaka \((0.5126 \pm 0.0284, p = 0.6204)\) and SubKHA \((EN)\) \((0.5106 \pm 0.0058, p = 0.4040)\).

### 4.3. MNIST dataset

Digital pictures of handwritten digits with size 28×28 \([22]\) compose the MNIST dataset. Reproducing an experiment from \([8]\), we selected the training and testing samples associated with the numbers “1”, “2”, and “3”, resulting in 22,008 images. We converted the selected images to vectors with dimension 784, which were submitted to the algorithms. Unlike \([8]\), we did not add noise to these pictures. The kernel width, number of components extracted, and dictionary size were the same adopted in an experiment from \([8]\). Fig. 6 exhibits the corresponding results.

KW test reported different steady-state ESRE values \((\chi^2 = 45.44, p < 0.001)\). Results show that the proposed RLS-KHA performed better than Tanaka algorithm in the entire simulation interval. Our method surpassed the OKHA and SubKHA \((EN, DS)\) at iterations 3000 and 6000, respectively, achieving the lowest mean steady-state ESRE \((0.2720 \pm 0.0094)\). It performed better than SubKHA \((DS)\) \((0.3004 \pm 0.0018, p = 0.0496)\), OKHA \((0.3347 \pm 0.0106, p < 0.001)\) and Tanaka algorithm \((0.3859 \pm 4.6476, p < 0.001)\), but performed similarly to SubKHA \((EN)\) \((0.2962 \pm 0.0043, p = 0.3228)\).
4.4. COREL dataset

The COREL dataset consists of four sets of features (color histogram, color histogram layout, color moments and co-occurrence texture), extracted from 68,040 photos images belonging to various categories, and obtained from Corel Corporation. We considered only the color histogram features to our experiments. The kernel width, number of components, and dictionary size were set similarly to ISOLET experiments. We have also included a modified (M) version of the RLS-KHA algorithm, employing the same dictionary management criterion of SubKHA (DS) to enlighten some findings. Fig. 7 exhibits the results.

Excluding RLS-KHA (M), the steady-state values were different according to KW test ($\chi^2 = 37.39$, and $p < 0.001$). The convergence behavior of the proposed RLS-KHA was better than OKHA and Tanaka, but worse than SubKHA. In terms of steady-state errors, our algorithm attained a mean steady-state ESRE of $0.1416 \pm 0.007$, performing better than OKHA ($0.1581 \pm 0.0060$, $p = 0.0807$), similarly to Tanaka ($0.1492 \pm 0.0068$, $p = 0.9480$) and SubKHA (DS) ($0.1323 \pm 0.0082$, $p = 0.5203$), and worse than SubKHA (EN) ($0.1214 \pm 0.0068$, $p = 0.0167$). However, the best convergence curve belongs to the RLS-KHA (M), which attained a steady-state ESRE value of $0.1118 \pm 0.0049$, similarly to SubKHA (EN) ($p = 0.0853$), and better than the remaining methods ($p < 0.0855$).

We conclude that the SubKHA performed better in this experiment since it fills its dictionary in much less iterations than RLS-KHA, as the former employs a more conservative criterion for member inclusion. Note that we have observed a similar phenomenon in USPS experiments when comparing the KHA with online KPCA algorithms.

4.5. Cover type dataset

The cover type UCI dataset involves 54 cartographic variables (elevation, slope, distance to hydrology, etc.) regarding wilderness areas located in the Roosevelt National Forest in northern Colorado. Each one of the 581,012 feature vectors relates to a 30×30 m cell, determined by US Forest Service (USFS). The kernel width, number of components, and dictionary size were set similarly to ISOLET experiments. We normalized each variable to be within 0–1 range, and also included the modified RLS-KHA as in COREL experiments. Fig. 8 shows the results.

Excluding the RLS-KHA (M), initially, Tanaka algorithm was the fastest technique, being surpassed by the proposed RLS-KHA, SubKHA (EN) and SubKHA (DS) at iterations 120,000, 180,000, and 230,000, respectively. From iteration 120,000 to 300,000, the proposed RLS-KHA was the best performing method, after what it was outperformed by SubKHA (EN). In terms of steady-state ESRE, the methods performed differently ($\chi^2 = 12.06$, $p = 0.0169$), but the proposed RLS-KHA, SubKHA (DS), and SubKHA (EN) attained the following equivalent ($p > 0.8983$) values: $0.2513 \pm 0.0022$, $0.2369 \pm 0.0134$, and $0.255 \pm 0.0213$, respectively. In turn, the values achieved by SubKHA (EN) were lower than OKHA ($0.2755 \pm 0.0150$, $p = 0.0884$) and Tanaka algorithm ($0.2806 \pm 0.0060$, $p = 0.0235$). Similar to COREL dataset, the RLS-KHA (M) converged faster than all methods ($\chi^2 = 17.62$, $p = 0.0035$), attaining a final mean steady-state ESRE ($0.2263 \pm 0.0031$) similar to SubKHA (EN, DS) ($p > 0.4657$), and better than OKHA and Tanaka algorithm ($p < 0.0203$). Thus, the use of a more aggressive criterion for dictionary construction may improve the performance of the proposed RLS-KHA, especially when dealing with large datasets, an issue to be explored in a future algorithm version.

5. Conclusions

In this work, we have proposed a new online kernel principal component extraction algorithm: the RLS-KHA. The method involves...
extraction equations as simple as the KHA algorithm and employs an individual learning rate to each component under extraction, which is dynamically tuned by a computationally inexpensive iterative equation. Results show that convergence speed and accuracy are conflicting issues in this approach, but the use of a suitably chosen forgetting factor leads to impressive gains in both requisites, especially as compared to state-of-the-art KPCA extraction methods.

Appendix A. The minimum value of Eq. (7)

Using straightforward algebra, Eq. (7) may be rewritten as

\[ O_j = \nabla_j \frac{1}{2} \mathbf{w}_j^T \mathbf{E} \mathbf{w}_j \]

where the trace operator is represented by \( \text{tr} \cdot \). The gradient of \( O_j \) with respect to \( \mathbf{w}_j \), denoted as \( \nabla_{\mathbf{w}_j} O_j \), is given by

\[ \nabla_{\mathbf{w}_j} O_j = -4\mathbf{E} \mathbf{w}_j \]

Assuming that the vector \( \mathbf{w}_j \) has an arbitrary dimensionality of \( n \), and considering Eqs. (8) and (9), the following expression results

\[ \mathbf{E} (\mathbf{w}_j \mathbf{w}_j^T) = \mathbf{F}_j \mathbf{E} (\mathbf{w}_j \mathbf{w}_j^T) = \left( 1 - \sum_{i=1}^k \xi_i \right) \sum_{i=1}^k \xi_i \mathbf{e}_i \mathbf{e}_i^T = \mathbf{F}_j \mathbf{E} (\mathbf{w}_j \mathbf{w}_j^T) \]

Making \( \nabla_{\mathbf{w}_j} O_j = 0 \) and considering Eq. (A.3), results in

\[ \mathbf{C}_j \mathbf{w}_j = \delta \mathbf{w}_j \]

for

\[ \delta = \mathbf{w}_j^T \mathbf{C}_j \mathbf{w}_j \]

i.e., the minimum value of \( O_j \) is achieved for \( \mathbf{w}_j = \alpha \mathbf{e}_j \), which is a scaled-version of the \( j \)-th unit norm dominant eigenvector of \( \mathbf{C}_j \). Substituting this result into Eq. (A.4), we have

\[ a \mathbf{C}_j \mathbf{e}_j = (\alpha \mathbf{e}_j^T \mathbf{C}_j \mathbf{e}_j) \alpha \mathbf{e}_j \]

\[ \mathbf{C}_j \mathbf{e}_j = \alpha^2 \lambda_j \mathbf{e}_j \]

thus \( a = \pm 1 \).

Appendix B. Proof of Eq. (19)

Expanding Eq. (17), we have

\[ \mathbf{w}_j(k) = \left[ \frac{1}{\xi_j(k)} \left( \gamma \mathbf{F}_j^T \mathbf{y}_j(k-1, k) + \lambda \mathbf{F}_j^T \mathbf{y}_j(k-1, i) \right) \right] \]

Similarly, Eq. (18) can be written as

\[ \xi_j(k) = \frac{\gamma^2}{\xi_j(k-1, k)} + \frac{\lambda}{\xi_j(k-1, k)} \]

\[ \xi_j(k-1) = \frac{1}{\gamma^2} \xi_j(k) \]

Substituting Eq. (B.2) in (B.1) results in Eq. (19).

References

datasets/ISOLET, 2015.


João B.O. Souza Filho received the Electronic Engineering degree in 2001, and the M.Sc and the D.Sc degrees in Electrical Engineering in 2002 and 2007, respectively, all from Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil. He was with the Federal Center of Technological Education Celso Suckow da Fonseca (CEFET-RJ) from 2005 to 2015. Now he is a full professor of the Department of Electronic Engineering from UFRJ. His research interests are in pattern recognition, machine learning, artificial neural networks, adaptive systems, digital signal processing, embedded systems and electronic instrumentation.

Paulo S.R. Diniz received the Electronics Eng. degree (Cum Laude) from the Federal University of Rio de Janeiro (UFRJ) in 1978, the M.Sc. degree from COPPE/UFRJ in 1981, and the Ph.D. from Concordia University, Montreal, Canada, in 1984 (all in Electrical Engineering). He is with the Department of Electronic Engineering and the Postgraduate Program in Electrical Engineering from UFRJ. Diniz is an IEEE Fellow. He served as associate editor of some IEEE journals and received many career awards, having published numerous refereed papers and three textbooks. His research interests are in adaptive signal processing, digital and wireless communications and multirate systems.