Applied Soft Computing Journal 96 (2020) 106622

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

A generalized framework for ANFIS synthesis procedures by clustering techniques

Stefano Leonori [*], Alessio Martino, Massimiliano Luzi, Fabio Massimo Frattale Mascioli, Antonello Rizzi

Department of Information Engineering, Electronics and Telecommunications - University of Rome "La Sapienza", Via Eudossiana 18, 00184 Rome, Italy

ARTICLE INFO

Article history:
Received 10 March 2020
Received in revised form 30 July 2020
Accepted 4 August 2020
Available online 8 August 2020

Keywords:
Machine learning
Function approximation
Clustering
ANFIS
Neuro-fuzzy classifiers

ABSTRACT

The application of machine learning and soft computing techniques for function approximation is a widely explored topic in literature. Neural networks, evolutionary algorithms and support vector machines proved to be very effective, although these models suffer from very low level of interpretability by human operators. Conversely, Adaptive Neuro Fuzzy Inference Systems (ANFISs) demonstrated to be very accurate models featured by a considerable degree of interpretability. In this paper, a general framework for ANFIS training by clustering is proposed and investigated. In particular, different derivative-free ANFIS synthesis procedures are considered for performance evaluation, by taking into account different clustering algorithms, dissimilarity measures and by including an additional neuro-fuzzy classifier downstream the clustering phase targeted to rule base refinement. The resulting ANFISs have been compared, in terms of effectiveness and efficiency, on several benchmark datasets against three suitable competitors, namely a Support Vector Regression, MultiLayer Perceptron and a K-Nearest Neighbour decision rule. Computational results show that the proposed techniques tend to outperform competing strategies while, at same time, featuring models with lower structural complexity. A complete software suite implementing the proposed framework is freely available under an open-source licence.

1. Introduction

In the last years, with the breakthrough of machine learning and soft computing techniques, the trade-off between accuracy and interpretability has been the subject of active research lines. Especially in industrial applications, soft computing algorithms are often required to be as interpretable as possible since they are in charge to manage complex and critical systems, whose failure can result in service interruptions and/or resource waste. Furthermore, operators are likely personnel with no background on soft computing, hence the model interpretability, readability and reproducibility are in many cases strict requirements.

On this topic, models based on fuzzy logic such as Fuzzy Inference Systems (FISs) and Adaptive Neuro-Fuzzy Inference Systems (ANFISs) have been employed successfully in numerous applications for decision-making, medical diagnosis [1], control system [2–5], real time energy management [6–10], prediction [11–13] and other data mining problems..

Conversely to some variants of Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs), considered as *black box models* because of their inaccessibility [14], FISs and ANFISs are often addressed in the literature as *grey box models* thanks to their rule-based nature and the ability to deal with linguistic variables, as claimed in [10,15–17].

Indeed, the rule-based architecture of FISs and ANFISs allows an implicit interpretation of the architecture itself, as well as insights on the computational process to be modelled. Concerning this topic, in [18] a neural network featured by sigmoid activation function is designed in order to be approximated by an equivalent Takagi–Sugeno type fuzzy system in order to improve its interpretability.

ANFISs, in particular, have shown remarkable results as universal approximators, being able to deal with the uncertainty in training suitable models, as well as to take advantage of expert knowledge by means of the intrinsic rule-based architecture [19]. As argued by Jang himself, ANFISs are hybrid intelligent systems which integrate both the fuzzy logic's qualitative approach and the ANNs adaptive capabilities towards better performance [20,21]. Like ANNs, Takagi–Sugeno rules based ANFISs are supervised

[*] Corresponding author.
E-mail addresses: stefano.leonori@uniroma1.it (S. Leonori), alessio.martino@uniroma1.it (A. Martino), massimiliano.luzi@uniroma1.it (M. Luzi), fabiomassimo.frattalemascioli@uniroma1.it (F.M. Frattale Mascioli), antonello.rizzi@uniroma1.it (A. Rizzi).

https://doi.org/10.1016/j.asoc.2020.106622
1568-4946/© 2020 Elsevier B.V. All rights reserved.

modelling systems able to perform piece-wise linear approximations of non-linear functions. Moreover, its pseudo-ANN architecture allows the application of gradient descents optimization procedures, unlike plain Mamdani FIS [22].

As their interpretability is concerned, in [15] and [17], the authors discuss about the potentials and issues of human–computer cooperation and problem solving efficiency of ANFISs. In particular, they state that although ANFISs are introduced in order to preserve the FIS linguistic variables and rule properties, without the injection of expert knowledge these can lose their level of interpretability going *de-facto* towards plain ANNs (*i.e.*, black box models). However, it is important to note that, even in extreme cases, their rule-based behaviour always allows a better human–machine cooperation, unlike ANNs.

In [23] authors well analyse and illustrates how to quantify a fuzzy system level of interpretability listing four main factors here briefly summarized:

- Complexity at the rule base level: rules number (*i.e.* model conditions)
- Complexity at the fuzzy partition level: number of Membership Functions (MFs) (*i.e.* model features)
- Semantics at the rule base level: rules fired at the same time
- Semantics at the fuzzy partition level: completeness or coverage and distinguishability.

The first and the second points are related to the model complexity, namely looking at the quantity of rules and MFs. The third and the fourth points, as instead, aim to ensure semantic integrity by studying the coverage and distinguishability of the MFs and the rules in their respective domain.

Concerning fuzzy systems training procedures, in [1], the authors suggest naturally-inspired heuristics, such as Genetic Algorithms (GAs), as effective optimization tools to get a good trade-off between model interpretability and accuracy. These paradigms demonstrated effective solution in renewable energy control, prediction and management systems as reviewed in [2,3] for wind turbine and PV systems and in [7,24–26] for hybrid renewable energy systems and microgrids, where other optimization algorithms are mentioned, such as chaotic optimization, particle swarm optimization (PSO), cuckoo search and differential evolution.

However, paradigms like FIS-GA revealed to be difficult to converge to a suitable solution in case of huge lack of knowledge regarding the problem at hand [15], especially for high-dimensional problems.

In [19], the authors suggest that, in order to keep a good level of interpretability, the MFs can be intuitively defined by means of a grid partitioning of the input domain and therefore the rule base system can be defined again by an evolutionary based algorithm.

However, such procedures can critically increase the model complexity by over-generating the number of inefficient MFs, hence the number of rules: indeed, a grid sampling of the input domain without considering the underlying data distribution might lead to the synthesis of MFs in low-density regions, which are unlikely to be fired. To address this problem and limit the *curse of dimensionality* dilemma, Hierarchical FIS-GA have been proposed in [8] to automatically prune the input MFs, hence the rule-based system and simplifying the synthesis procedure by dividing it into several steps. Also the implementation of an ensemble of ANFISs can improve the model performances, especially for time series problems [11,12,27].

An alternative approach involves the use of clustering algorithms for partitioning the input space for a more practical and effective determination of bell-shaped fuzzy sets, especially for high-dimensional input spaces. Indeed, clustering-based techniques allow to centre and model MFs according to clusters' data distribution, whereas the same is not true for grid partitioning techniques. In [28–30] authors well highlights that clustering techniques well support evolving structures when adopting split and/or merging operations in order to solve Big Data problems. This procedure allows the integration of knowledge iteratively or "on-the-fly", and would let evolve a fuzzy model, unlike evolutionary based systems which usually have all the data in an iterative optimization procedure. Moreover, the use of clustering-based training procedures not only is derivative-free, but it does not need any evolutionary-based optimization either: this results in affordable computational efforts.

As regards recent works on ANFISs, a multi-ANFIS system is employed in [5] to model the complex dynamics of a high-speed electric multiple unit and to design a predictive controller. The ANFISs MFs are initialized by means of a subtractive clustering algorithm, which can take full advantage of the intrinsic data distribution, and later optimized via backpropagation. In [31], an ANFIS is used to model the voltage response of electrochemical cells. In this case, the ANFIS is trained by means of PSO, and it demonstrates a very effective performance at dealing with the uncertainties of electrochemical cells. In [32], an effective method to synthesize higher order Takagi–Sugeno–Kang (TSK) fuzzy systems using first-order TSK models is described. The proposed model allows to train high order TSK fuzzy systems in popular softwares (*e.g.*, MATLAB®) that notably allow the implementation of low order TSK fuzzy systems only. In [12] an ensemble learning ANFIS model is tested for chaotic time series prediction. The ensemble structure allows to vary the training procedure to each ANFIS in order to better simulate an expert system. An ensemble of three interval type-2 fuzzy neural network models with optimization of Fuzzy integrators is employed for time series prediction in [11] by means of a GA, comparing its performances with PSO.

In [33], a recurrent mechanism for generating an ANFIS by means of a kernel-based fuzzy clustering method is used for identifying dynamic response of the magnetorheological damper. The ANFIS is built upon a filtered dataset from impulse noise. The recurrent mechanism consists in updating the filter by the ANFIS and in re-building itself with the new filtered dataset until the stop criteria is reached. A novel paradigm named Extreme Learning ANFIS (ELANFIS) is proposed in [4] for control and prediction problems. In ELANFIS, the fast learning procedure of extreme learning machines and reservoir networks is exploited for training an ANFIS. The advantage of using ELANFISs over simple extreme learning machines relies on MFs (*i.e.*, explicit knowledge representation), allowing to reduce the randomness of the generated model [34]. In [35] a general purpose online self-organizing ANFIS is compared with a neural network, emphasizing the ease of implementation. The ANFIS rules are generated by a nearest-neighbourhood based clustering algorithm and a pruning method is used to allow the online generation of new rules, deleting those considered no more effective. In [36] different ANFIS optimization algorithms are evaluated considering back-propagation, mini-lots, and ADAM algorithms.

It has been discussed that synthesizing ANFISs by means of clustering algorithms is an effective, derivative-free, and computationally efficient method, as well as it allows to determine automatically the most relevant input fuzzy sets for addressing the input–output relationship of the process at hand. Thus, this work proposes a generalized framework for synthesizing ANFIS networks by means of clustering techniques, following two baseline works [37,38]. Specifically, given a labelled pattern set, an ANFIS function approximator with multidimensional MFs and first-order hyperplane rule consequents has been implemented and trained considering different k-means-based techniques. In this paper, we extend these two baseline works by considering several combinations of clustering techniques (one partitional and two hierarchical strategies), dissimilarity measures (squared Euclidean and Mahalanobis) and the possibility to refine the clustering solutions by means of different neuro-fuzzy Min–Max classifiers [39]. These degrees of customization allow to sketch the ANFIS synthesis as a properly-defined framework, whose building blocks analysis is the main focus of this paper. Specifically, with respect to our previous works:

- we consider a more general test campaign for addressing the performance of the proposed ANFIS design by relying on benchmark datasets and real-world dataset, whereas in [37] we tailored our analysis on the synthesis of an energy management system and in [38] only synthetic datasets have been used;
- in [37] we considered only hyperplane clustering strategies for synthesizing the ANFIS, whereas in this work the input space will also be taken into account by exploiting different dissimilarity measures;
- in [37] only a specific neuro-fuzzy Min–Max classifier variant (PARC) has been employed and in [38] only another variant has been used (ARC), whereas in this work we use two variants (PARC and GPARC) along with the possibility of skipping the clusters refining step by neuro-fuzzy classifiers;
- in [38] only a k-means-like clustering procedure has been adopted for synthesizing the ANFIS, whereas in this work we use three different clustering variants.

Furthermore, in order to analyse the effectiveness and the competitiveness of the considered methods, all synthesized ANFISs have been compared with baseline models for function approximation, namely MultiLayer Perceptron (MLP), Support Vector Regression (SVR) [40] and K-Nearest Neighbours (K-NN) [41].

The paper is organized as follows. In Section 2 the ANFIS architecture is introduced, while in Section 3 the main actors involved in the ANFIS synthesis procedure are described (namely, clustering algorithms, hyperplane clustering and Min–Max classifiers). In Section 4, the details of the synthesis procedure and the experimental settings are explained, along with the three competitors (MLP, SVR, K-NN). Section 5 shows the computational results, followed by the conclusions, in Section 6.

The software implementing the proposed framework is available at https://gitlab.com/labcoin/anfis-toolbox.

2. ANFIS architecture

ANFIS architectures are usually represented as a pseudo-ANN with five layers where the first layer implements the fuzzification process for determining suitable input MFs. In the second layer, the rule fire strengths are calculated and then normalized in the third layer. Later, in layer four, each Rule Consequent (RC) hyperplane is evaluated and, finally, their weighted sum is considered for evaluating the crisp output.

In this work, we refer to a simplified ANFIS version illustrated in Fig. 1 which is defined by just three layers since the MF, the rule weights and the rule premise parts have been incorporated. This model simplification can be obtained by relying on the definition of multivariate Gaussian MFs, defined as

$$\phi^{(i)}(\mathbf{u}) = e^{-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu}^{(i)})\cdot\mathbf{C}^{(i)^{-1}}\cdot(\mathbf{u}^T-\boldsymbol{\mu}^{(i)^T})} \quad (1)$$

where $\phi^{(i)}$ is the generic ith MF, defined by $\boldsymbol{\mu}^{(i)}$ and $\mathbf{C}^{(i)}$, namely the ith cluster centroid and the covariance matrix, respectively. $\mathbf{u}$ is the crisp input vector normalized in the unitary hypercube $[0, 1]^m$, being m the number of real-valued input features (input space dimension).

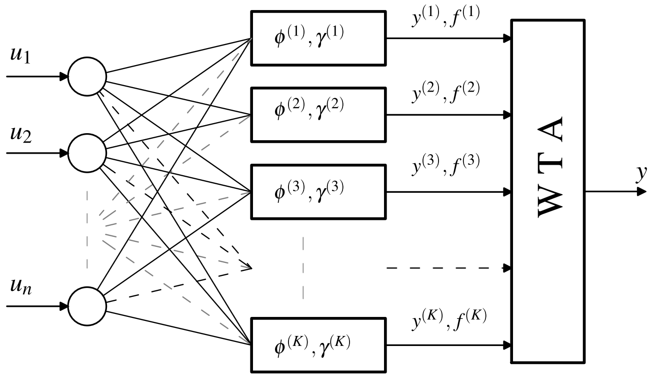

u_1
u_2
u_n
$\phi^{(1)}, \gamma^{(1)}$
$\phi^{(2)}, \gamma^{(2)}$
$\phi^{(3)}, \gamma^{(3)}$
$\phi^{(K)}, \gamma^{(K)}$
$y^{(1)}, f^{(1)}$
$y^{(2)}, f^{(2)}$
$y^{(3)}, f^{(3)}$
$y^{(K)}, f^{(K)}$
WTA
y

Fig. 1. TKS-ANFIS architecture.

Due to the MF multidimensionality, the ith MF $\phi^{(i)}$ fixes the rule antecedent set of the ith rule, defined as follows:

$$\text{IF } \mathbf{u} \text{ is } \phi^{(i)} \text{ THEN } y^{(i)} = \mathbf{u}^T \cdot \gamma^{(i)} \quad (2)$$

where, in turn, $\gamma^{(i)}$ is the ith RC, defined by a properly tuned hyperplane coefficients vector. The absence of AND-OR operators in the premise part of the rule allow to collapse the firing rule strength f to the MF value, namely $f^{(i)} \equiv \phi^{(i)}$.

As shown in Fig. 1, a Winner Takes All (WTA) strategy is adopted for computing the overall FIS output as the output of the most firing rule (highest MF value).

3. ANFIS synthesis

FISs synthesis by clustering is a technique widely explored in the literature. As argued in [38], the synthesis involves two steps. During the first part, clustering algorithms can leverage on the pattern set (input space) for generating the support domain of each MFs. In a second step, a linear least square is used to find the hyperplane that fits the corresponding output values. However, it is important to note that this approach can be restrictive because the information carried out by the output values is not used to generate the MFs, but only (in a second moment) with the consequent generation. For this reason, it has been underlined in [38] the possibility to extend the clustering algorithms to work in the hyperplane space or, better, in the joint input-hyperplane space in order to exploit the joint input–output information also in the clustering phase.

3.1. Partitional clustering

The most straightforward technique to synthesize the MFs (see Eq. (1)) is by means of partitional clustering algorithms such as k-means [42] or k-medoids [43]. Such algorithms, however, have to be re-adapted in order to work in a joint input-hyperplane space fashion, since plain clustering algorithms notably work in an unsupervised fashion [43–45]. As in [37] and [46], let us consider a given cluster to be represented by a prototype vector $\boldsymbol{\mu}$ in the input space (*e.g.*, centroid, medoid) and the hyperplane coefficient vector $\boldsymbol{\theta}$. The distance between a given labelled pattern $\langle \mathbf{x}, y\rangle$ and a given cluster $\langle \boldsymbol{\mu}, \boldsymbol{\theta}\rangle$ is defined as follows:

$$\hat{d}(\langle \mathbf{x}, y\rangle, \langle \boldsymbol{\theta}, \boldsymbol{\mu}\rangle) = \varepsilon d(\mathbf{x}, \boldsymbol{\mu}) + (1-\varepsilon)(y - (\boldsymbol{\theta}^T\mathbf{x} - \theta_0))^2 \quad (3)$$

where $\varepsilon \in [0, 1]$ weights the contribution between the leftmost term, namely the distance $d(\cdot, \cdot)$ in the input space (*e.g.*, squared Euclidean distance), and the rightmost term, namely the distance in the hyperplane space (*i.e.*, the approximation error due to the hyperplane). Specifically, if $\varepsilon = 0$, the algorithm collapses into a proper hyperplane clustering. Further, it is worth stressing that hyperplanes are affine subspaces as they also include the intercept θ_0.[1]

The partitional clustering procedure follows the widely-known Voronoi iterations: after a random initialization, each point is assigned to the closest cluster according to Eq. (3) and then clusters' representatives are updated. In case of k-means the centroid is defined as the centre of mass of the cluster; for k-medoid, the medoid is defined as the cluster element which minimizes the sum of pairwise distances; in both cases the hyperplane coefficient vector can be evaluated by a plain ordinary least square estimator. The assignments and updates steps iterate until a stopping criterion is met. Algorithm 1 summarizes the partitional clustering procedure.

Algorithm 1: Pseudocode for joint input-hyperplane clustering k-means

```
Input : Dataset instance matrix $\mathbf{X}$, Output values vector $\mathbf{y}$, number of clusters $k$, maximum number of iteration $I$, tradeoff weight $\varepsilon$
Output: Set of $k$ clusters
$\mathbf{X} = [\mathbf{1}\ \mathbf{X}]$;
numPatterns = size($\mathbf{X}$, 'rows');
hyperplanes = initialHyperplanes();
centroids = initialCentroids();
for iter in range(1, $I$) do
    for $i$ in range(1, numPatterns) do
        for $j$ in range(1, $k$) do
            $D[i,j] = \varepsilon \cdot d(\mathbf{X}[i,:], \text{centroids}[j,:]) +$
            $+(1-\varepsilon)\cdot(\mathbf{y}[i] - (\text{hyperplanes}[j,:]^T\ \mathbf{X}[i,:]))^2$;
        end
    end
    for $j$ in range(1, numPatterns) do
        assignments[$j$] = argmin(D[$j$, :]);
    end
    for $j$ in range(1, $k$) do
        $\mathbf{X}_j = \mathbf{X}[\text{assignments} == j, :]$;
        $\mathbf{y}_j = \mathbf{y}[\text{assignments} == j]$;
        hyperplanes[$j$, :] = $(\mathbf{X}_j^T\mathbf{X}_j)^{-1}\mathbf{X}_j^T\mathbf{y}_j$;
        centroids[$j$, :] = mean($\mathbf{X}_j$,'columns');
    end
    if stoppingCriteria() is True then
        break;
    end
end
```

3.2. Hierarchical clustering

Alongside plain partitional clustering algorithms, two hierarchical procedures are also considered for the ANFIS synthesis, originally proposed in [37].

The first variant (divisive) works in a top-down fashion by starting with a single root node corresponding to the entire dataset. A 2-means aims at generating the first set of (two) leaf nodes. Then, iteratively, the following steps are performed until a desired number of $k_{\max}$ leaf nodes is returned:

1. for each leaf node, evaluate the within-cluster sum of distances (cf. Eq. (3))
2. select the leaf node corresponding to the highest within-cluster sum of distances
3. run a 2-means on the latter and generate two offsprings.

The second variant (agglomerative) works in a bottom-up fashion by starting with a desired number of $k_{\max}$ leaf nodes obtained using the partitional strategy described in Section 3.1. Then, iteratively, the following steps are performed until a single root node corresponding to the entire dataset is returned:

1. evaluate $\mathbf{D}$, the pairwise distance matrix between centroids
2. evaluate $\mathbf{E}$, the pairwise error matrix that encodes the hyperplane approximation error obtained if any two clusters had to be merged together
3. evaluate $\bar{\mathbf{E}}$, the pairwise overall error matrix that encodes both the intra-cluster (sum of distances) and the inter-cluster ($\mathbf{E}$) errors obtained if any two clusters had to be merged together
4. define the score matrix $\mathbf{S} = \max(\mathbf{D}, \bar{\mathbf{E}})$ and select the (i, j) pair leading to the minimum score as the two clusters to merge
5. merge the two clusters and evaluate the new centroid $\boldsymbol{\mu}$ and hyperplane coefficient vector $\boldsymbol{\theta}$.

Fig. 2 sketches the two hierarchical variants for a simple problem where $k_{\max} = 3$. The rightmost tree shows the divisive variant, where one starts with the entire dataset, runs a 2-means on the latter and obtains two leaf nodes. By supposing that cluster 1 leads to the higher within-cluster error, a 2-means is performed on the cluster and two further clusters are returned, whereas cluster 2 remains unaltered. The set of leaf nodes after each splitting procedure returns a 'layer' with cardinality k. Similarly, the leftmost tree shows the agglomerative variant in which the first set of $k_{\max} = 3$ nodes is returned by the plain partitional variant. For the resulting clusters, the pairwise matrices $\mathbf{D}$, $\mathbf{E}$, $\bar{\mathbf{E}}$ and $\mathbf{S}$ are evaluated and by supposing that clusters $(1, 2)$ lead to the lowest score in $\mathbf{S}$, the merging procedure takes place by stacking patterns and output values belonging to the two clusters and then updating centroid and hyperplane coefficient. Clusters not involved in the merging procedure remain unaltered. As per the divisive counterpart, the set of leaf nodes after k merging procedures identifies a 'layer' with k clusters. Algorithms 2, 3 and 4 summarize the agglomerative clustering procedure. Similarly, Algorithms 5, 6 and 7 summarize the divisive clustering procedure.

3.3. Clusters refining by Min–Max classifiers

Regardless of the specific clustering technique (partitional, agglomerative, divisive), whether clusters can be described by means of their prototype vector $\boldsymbol{\mu}$, covariance matrix $\mathbf{C}$ and hyperplane vector $\boldsymbol{\theta}$, they can be used to synthesize the ANFIS MFs according to Eqs. (1)–(2). However, for some problems, the following undesired scenario may happen: two clusters well-separated in the input space are approximated by the same hyperplane. As the MFs are concerned, this results in ambiguous MF definitions as heavy overlaps can be observed. The toy problem [37] shown in Fig. 3-(a) helps in visualizing such scenario: four clusters lying on four different lines are represented by black dots in figure, a single (blue) hyperplane approximates two clusters well separated in the input space, the orange hyperplane represents a third cluster between them, whereas the green one covers the fourth cluster on the right. However, the regions pertaining the first two MFs (orange and blue Gaussian curves in the bottom panel of Fig. 3-(a)) are mostly overlapped, leading to ambiguity in the selection of the winning rule for a wide interval of the input axis. Specifically, the blue MF almost completely prevails on the orange MF causing a bad approximation of samples falling onto the input space region pertaining to orange cluster.

[1] In order to explicitly consider the intercept, one shall augment the input patterns by appending a leading 1, hence $\mathbf{x} := [1, \mathbf{x}]$.

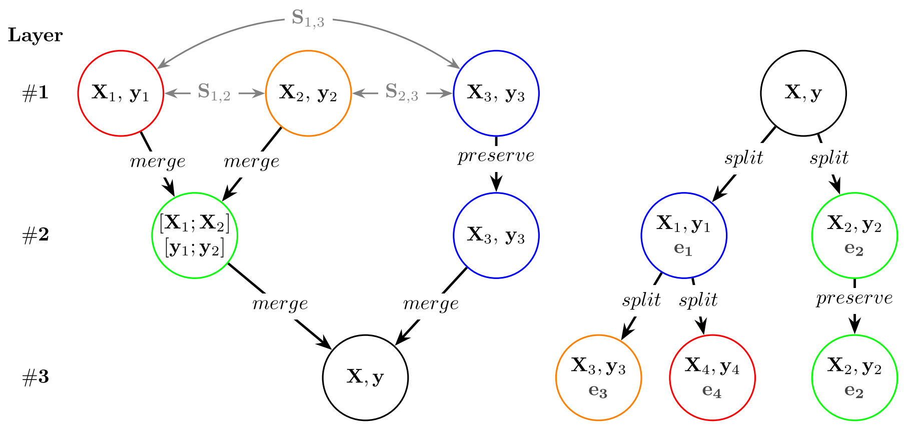
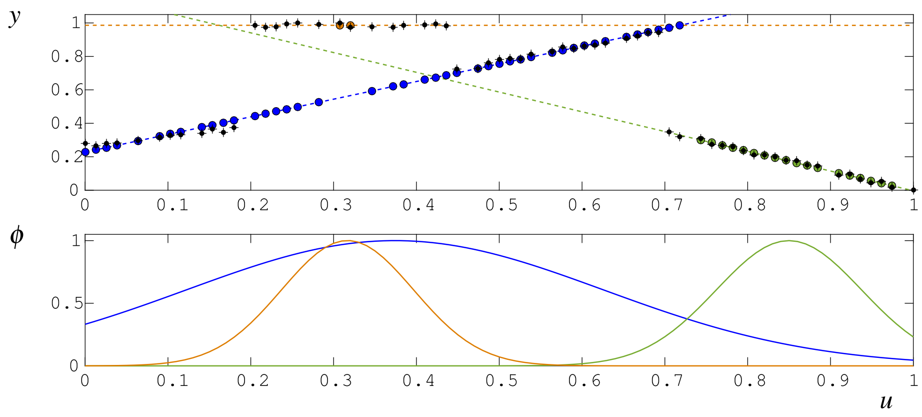
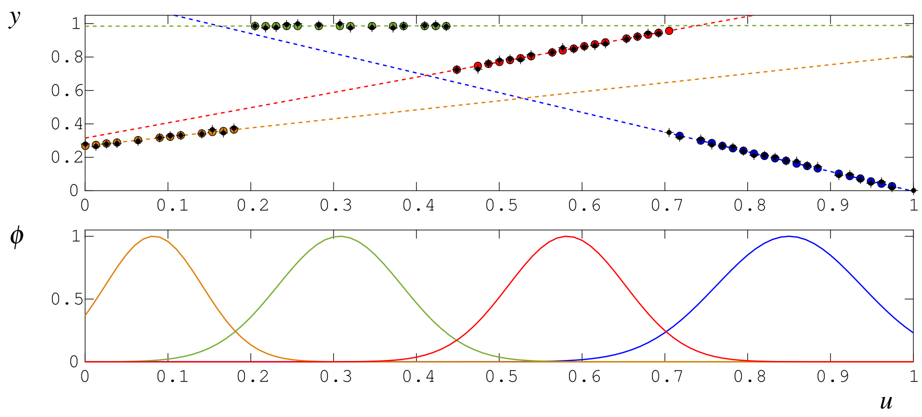


Fig. 2. Agglomerative (left) and divisive (right) clustering variants scheme, both having three layers depth.



(a) ANFIS function approximation by *k*-means clustering. Top panel: the training input-output samples as crossed black dots while dashed lines represents RCs. Input-output samples represented with the same colour fall in the same cluster hyperplane. Bottom panel: MFs built upon each cluster. MF are plotted with the corresponding cluster colour on top panel.



(b) ANFIS synthesis function approximation by *k*-means clustering supported by Min-Max PARC classification system.

Fig. 3. Relying on a simple toy problem, the above panels show the role of PARC procedure in avoiding overlapping MFs.

In order to overcome this problem, a classifier can be used right after the clustering procedure in order to refine the MF definitions [38]. Specifically, a Min–Max classifier is trained on a labelled dataset where each pattern is labelled with the cluster ID to which the pattern has been assigned to the clustering algorithm. With this new re-labelled pattern set, the Min–Max classifier is able to return a set of hyperboxes, each one labelled with their respective cluster ID, returning this way a better tessellation of the input space. In particular, the hyperboxes are generated in order to avoid any overlapping between pairs of hyperboxes associated with different class labels. The output of a Min–Max classifier consists in a set of hyperboxes, each of which is represented by its min and max vertex coordinates. Hence, in order to properly re-estimate the MFs, each pattern from the training set is reassigned to the hyperbox in which

Algorithm 2: Selection Routine for Agglomerative Hierarchical Clustering

```
Input  : A set of $k$ clusters $\mathscr{S}_i|_{i=1}^{k}$ of the form $\langle \mathbf{X}_i, \mathbf{y}_i \rangle|_{i=1}^{k}$, their centroids $\boldsymbol{\mu}^{(i)}|_{i=1}^{k}$ and hyperplanes $\boldsymbol{\theta}^{(i)}|_{i=1}^{k}$, tradeoff weight $\varepsilon$
Output: IDs of the two clusters selected for merging
1  for $i$ in range(1, $k$) do
2      $e[i] = \sum_{j=1}^{|C_i|} \varepsilon d(\mathbf{x}^{(j)}, \boldsymbol{\mu}^{(i)}) + (1-\varepsilon)(y^{(j)} - (\boldsymbol{\theta}^{(i)T}\mathbf{x}^{(j)} - \theta_0^{(i)}))^2$;
3  end
4  for $i$ in range(1, $k$) do
5      for $j$ in range(1:$k$) do
6          $D[i,j] = \|\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}\|_2^2$;
7      end
8  end
9  for $i$ in range(1, $k$) do
10     for $j$ in range(1, $k$) do
11         $\mathbf{X}_{\text{tmp}} = [\mathbf{X}_i\ ;\ \mathbf{X}_j]$;
12         $\mathbf{y}_{\text{tmp}} = [\mathbf{y}_i\ ;\ \mathbf{y}_j]$;
13         $\boldsymbol{\theta}_{\text{tmp}} = (\mathbf{X}_{\text{tmp}}^T\mathbf{X}_{\text{tmp}})^{-1}\mathbf{X}_{\text{tmp}}^T\mathbf{y}_{\text{tmp}}$;
14         $E[i,j] = \sum(\mathbf{y}_{\text{tmp}} - (\boldsymbol{\theta}_{\text{tmp}}^T \cdot [\mathbf{1}\ \mathbf{X}_{\text{tmp}}]))^2$;
15     end
16 end
17 for $i$ in range(1, $k$) do
18     for $j$ in range(1, $k$) do
19         $\bar{E}[i,j] = E[i,j] - \frac{e[i]+e[j]}{2}$;
20     end
21 end
22 $D = \frac{D - \min(D)}{\max(D) - \min(D)}$;
23 $\bar{E} = \frac{\bar{E} - \min(\bar{E})}{\max(\bar{E}) - \min(\bar{E})}$;
24 for $i$ in range(1, $k$) do
25     for $j$ in range(1, $k$) do
26         $S[i,j] = \max(D[i,j], \bar{E}[i,j])$;
27     end
28 end
29 [rowID, columnID] = argmin($S$);
```

Algorithm 3: Merging Routine for Agglomerative Hierarchical Clustering

```
Input  : The two clusters to be merged as returned by the Selection Routine of the form $\langle \mathbf{X}_i, \mathbf{y}_i \rangle$ and $\langle \mathbf{X}_j, \mathbf{y}_j \rangle$
Output: The merged cluster of the form $\langle \mathbf{X}, \mathbf{y} \rangle$, their hyperplane $\boldsymbol{\theta}$ and centroid $\boldsymbol{\mu}$
1  $\mathbf{X} = [\mathbf{X}_i\ ;\ \mathbf{X}_j]$;
2  $\mathbf{y} = [\mathbf{y}_i\ ;\ \mathbf{y}_j]$;
3  $\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$;
4  $\boldsymbol{\mu}$ =sum($\mathbf{X}$, 'columns') / size($\mathbf{X}$, 'rows');
```

Algorithm 4: Pseudocode for Agglomerative Hierarchical Clustering

```
Input  : Dataset instance matrix $\mathbf{X}$, Output values vector $\mathbf{y}$, number of clusters $k_{\max}$, maximum number of iterations $I$
Output: Dendrogram representation of the $k_{\max}$ clustering solutions
1  $\{C_1, \dots, C_{k_{\max}}\}$ = partitional($\mathbf{X}$, $\mathbf{y}$, $k_{\max}$, $I$);          // Alg. 1
2  dendrogram[$k_{\max}$] = $\{C_1, \dots, C_{k_{\max}}\}$;
3  for $k$ in range($k_{\max} - 1$, 1) do
4      $[i, j]$ = selection(dendrogram[$k+1$]);          // Alg. 2
5      $C_i$ = dendrogram[$k+1$][$i$];          // cluster i of layer k+1
6      $C_j$ = dendrogram[$k+1$][$j$];
7      $C_{\text{new}}$ = merging($C_i$, $C_j$);          // Alg. 3
8      dendrogram[$k$] = $C_{\text{new}} \cup \{$dendrogram[$k+1$][$h$]$\}_{h=1,\dots,k+1}^{h\neq i\ \&\ h\neq j}$;
9  end
```

Algorithm 5: Selection Routine for Divisive Hierarchical Clustering

```
Input  : A set of $k$ clusters $\mathscr{S}_i|_{i=1}^{k}$ of the form $\langle \mathbf{X}_i, \mathbf{y}_i \rangle|_{i=1}^{k}$, their centroids $\boldsymbol{\mu}^{(i)}|_{i=1}^{k}$ and hyperplanes $\boldsymbol{\theta}^{(i)}|_{i=1}^{k}$, tradeoff weight $\varepsilon$
Output: ID of the cluster selected for splitting
1  for $i$ in range(1, $k$) do
2      $e[i] = \sum_{j=1}^{|C_i|} \varepsilon d(\mathbf{x}^{(j)}, \boldsymbol{\mu}^{(i)}) + (1-\varepsilon)(y^{(j)} - (\boldsymbol{\theta}^{(i)T}\mathbf{x}^{(j)} - \theta_0^{(i)}))^2$;
3  end
4  ID = argmax($e$);
```

Algorithm 6: Split Routine for Divisive Hierarchical Clustering

```
Input  : The cluster to split of the form $\langle \mathbf{X}, \mathbf{y} \rangle$, the maximum number of iterations $I$
Output: Two offspring clusters $C_1$ and $C_2$
1  $\{C_1, C_2\}$ = partitional($\mathbf{X}$, $\mathbf{y}$, 2, $I$);          // Alg. 1
```

Algorithm 7: Pseudocode for Divisive Hierarchical Clustering

```
Input  : Dataset instance matrix $\mathbf{X}$, Output values vector $\mathbf{y}$, number of clusters $k_{\max}$, maximum number of iterations $I$
Output: Dendrogram representation of the $k_{\max}$ clustering solutions
1  dendrogram[1] = $\langle \mathbf{X}, \mathbf{y} \rangle$;
2  for $k$ in range(2, $k_{\max}$) do
3      $i$ = selection(dendrogram[$k-1$]);          // Alg. 5
4      $C_i$ = dendrogram[$k-1$][$i$];
5      $[C_{\text{new}_1}, C_{\text{new}_2}]$ = split($C_i$);          // Alg. 6
6      dendrogram[$k$]
         = $C_{\text{new}_1} \cup C_{\text{new}_2} \cup \{$dendrogram[$k-1$][$h$]$\}_{h=1,\dots,k+1}^{h\neq i}$;
7  end
```

it lies by taking into account the min and max coordinates of the hyperboxes and the coordinates of the pattern itself. After the reassignment operation is complete, each hyperbox will be populated by a finite set of training patterns, making possible the re-estimation of the prototype vector and covariance matrix of the MF.

Two Adaptive Resolution Min–Max Classifiers [39] are considered in order to avoid large MF superpositions, namely Pruning Adaptive Resolution Min–Max Classifier (PARC) and Generalized Pruning Adaptive Resolution Min–Max Classifier (GPARC). The aim of PARC is to cover the entire training set with hyperboxes, each of which is associated to a given problem-related class label. PARC considers two lists, $\mathscr{L}_H$ and $\mathscr{L}_P$: the former contains 'hybrid' hyperboxes (namely, hyperboxes containing patterns belonging to different classes), the latter contains 'pure' hyperboxes (in which all patterns share the same class label). In an initial stage, the entire dataset is covered by one (hybrid) hyperbox. Iteratively, hybrid hyperboxes are properly cut until there are no more hybrid hyperboxes ($\mathscr{L}_H$ is empty, $\mathscr{L}_P$ contains a suitable number of 'pure' hyperboxes). Intuitively, this can lead to a huge number of hyperboxes, especially in case of highly non-linear decision boundaries. According to basic learning theory, PARC aims at minimizing the following cost function

$$F = (1 - \lambda)\pi + \lambda\kappa \tag{4}$$

where π takes into account the model performances (*i.e.*, ratio of misclassified pattern), κ considers the model complexity (*i.e.*, number of hyperboxes with respect to the training set size) and $\lambda \in [0, 1]$ weights the two contributions. PARC and GPARC share the same training procedure, with the major difference that PARC designs hyperboxes which are parallel to the coordinate axes of the input space, whereas GPARC is able to rotate the hyperboxes by considering a different (local) reference system thanks to the eigendecomposition of the set of patterns belonging to a candidate hyperbox [47].

Recalling the toy problem of Fig. 3, Fig. 3-(b) shows the refinement procedure thanks to the PARC classifier. Clearly, PARC has been able to detect the former orange cluster as an in-between, despite the first and third clusters shared the same (blue) label. The refinement procedure returned four separate clusters, with no ambiguities in the corresponding MFs.

Aiming at making more evident the benefits due to the Min–Max classifier, Figs. 4–5 show an explicit comparison between the MFs synthesized on the SPIRAL dataset[2] by the application of k-means only, k-means with PARC and k-means with GPARC. Patterns belonging to the same spiral arm have been associated with the same class label, casting this way the original data into a 3-classes classification problem instance. As shown in Fig. 4, for a satisfactory ANFIS synthesis, the plain k-means algorithm needs $k = 24$ clusters, upon which 24 MFs are built. Conversely, in Fig. 5 GPARC and PARC are able to achieve satisfactory results by setting $k = 6$ MFs only, and by exploiting the subsequent refinement performed by the classifier itself. Therefore, in this case study, the Min–Max allows to automatically set the resolution level of the MFs. Indeed, PARC returns 20 hyperboxes (hence 20 MFs), whereas GPARC returns 16 hyperboxes (16 MFs). Moreover, the GPARC ability to rotate the hyperboxes results in a more optimized coverage of the input space, as thoroughly discussed in [47]. As a drawback, the GPARC training procedure is computationally expensive due to the extensive use of singular value decompositions in order to find, for each hyperbox, the local reference system spanned by the maximum variance.

We conclude the section by focusing on the model interpretability of the proposed ANFIS design:

- the multidimensional bell-shaped MFs and the automatic clustering-based synthesis procedures, together with MFs refining step by Min–Max classifiers, well manages the generation of a limited number of rules, ensuring at the same time the coverage of the whole input space;
- hyperplane and joint input-hyperplane clustering procedures allow a more effective piece-wise linear approximation of the unknown function to be modelled;
- once trained, adopted fuzzy rules can be easily explained and interpreted by field experts.

4. Experimental settings

4.1. ANFIS synthesis and validation

We considered a set of tests for ANFIS synthesis by changing the following building blocks:

- dissimilarity measure $\mathscr{D}$: squared Euclidean distance (SQE), Mahalanobis distance (MAH);
- clustering algorithm $\mathscr{C}$: partitional (PRT), (hierarchical) divisive (DIV), (hierarchical) agglomerative (AGG);
- space in which the clustering procedure shall be performed (see Eq. (3)): input space ($\varepsilon = 1$) and almost pure hyperplane space ($\varepsilon = 0.05$).
- classification algorithm $\mathscr{P}$: without classification (-), PARC classifier (PAR), GPARC classifier (GPA).

As concerns the ε parameter, it weights the hyperplane space and input space contributions in the overall dissimilarity measure used for performing the clustering procedure (see Eq. (3)). Preliminary tests showed that the compromise value 0.5 does not provide satisfactory results. Moreover, choosing $\varepsilon = 0.05$ rather than $\varepsilon = 0$ allows to slightly consider the clusters compactness in the input space. In fact, our tests showed that in the former case the clustering procedure is faster, with no difference in the final performances, since it helps to keep low the number of clusters during training. Hence, our analyses will only focus on the two aforementioned values of $\varepsilon = 0.05$ and $\varepsilon = 1$. The λ value in Eq. (4) has been set equal to 0, hence privileging the classification accuracy.

In order to study how the ANFISs performances change as function of $\mathscr{C}$, $\mathscr{D}$, $\mathscr{P}$ and ε, every combination of the above parameters has been individually tested.

Regarding the ANFIS synthesis procedure, let us consider a labelled dataset split in Training Set (TR), Validation Set (VL) and Test Set (TS): the clustering algorithm is executed on the TR in order to generate $k_{\max}$ solutions. In case of partitional clustering, the procedure is independently executed $k_{\max}$ times by changing the number of clusters k from 1 to $k_{\max}$. For hierarchical clustering, each layer (*i.e.*, the set of leaf nodes after k merging – or splitting – operations) is considered as a prospective clustering solution.

After the clustering execution, if no downstream classification algorithm is set, the ANFIS MFs (1) and hyperplanes (2) are directly evaluated for each solution, as illustrated in Fig. 4. Otherwise, the TR patterns will be labelled with their respective cluster membership and then fed to the classifier. As shown in Fig. 5, the classifier would split every cluster in hyperboxes in order to refine the generation of the ANFIS MFs (and rule hyperplanes) increasing their own granularity and compactness: each resulting hyperbox can be read as a cluster, hence it can be used to synthesize the ANFIS MFs according to Eqs. (1)–(2).

Since this strategy foresees the generation of several ANFISs (namely one ANFIS for each clustering solution), the one which shows the minimum Root Mean Squared Error (RMSE) on the VL is selected, whereas the others are discarded. The ANFIS selected is then performed on the TS and its goodness is evaluated by considering its structural complexity (number of rules) and its performances (coefficient of determination, also known as R^2). RMSE and R^2 are respectively defined as:

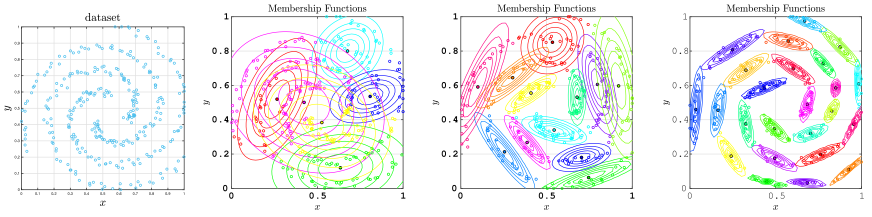
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5}$$

$$R^2 = \frac{\left(n\sum_{i=1}^{n} y_i\hat{y}_i - \left(\sum_{i=1}^{n}\hat{y}_i\right)\left(\sum_{i=1}^{n} y_i\right)\right)^2}{\left(n\sum_{i=1}^{n}\hat{y}_i^2 - \left(\sum_{i=1}^{n}\hat{y}_i\right)^2\right)\cdot\left(n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2\right)} \tag{6}$$

where n is the number of samples, y and $\hat{y}$ indicate the true and predicted output values, respectively.

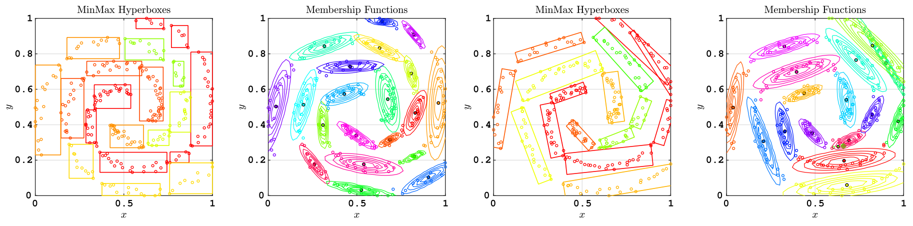
In Fig. 6, the block scheme of the overall ANFIS synthesis procedure is shown by considering the downstream support of a classifier: for the sake of simplicity, the scheme is focused on the synthesis of $ANFIS_{\bar{k}}$, where $\bar{k}$ is the number of clusters generated by the clustering block. It coincides with the ANFIS number of MFs only in case the classifier is not used or does not split any cluster.

2 http://cs.uef.fi/sipu/datasets/



(a) SPIRAL dataset (b) $K = 6$ clusters (c) $K = 12$ clusters (d) $K = 24$ clusters

Fig. 4. Application of plain k-means on SPIRAL dataset, considering 6, 12 and 24 clusters for the ANFIS MFs generation.



(a) $K = 6$ clusters + PARC (b) $K = 6$ clusters + GPARC

Fig. 5. Study of the k-means supported by PARC and GPARC classifiers for the generation considering the SPIRAL dataset.

4.2. Benchmark models

The investigated ANFIS synthesis techniques are benchmarked against three affirmed models for function approximation:

K-NN: Given a test pattern $\mathbf{x}$, performing regression via nearest neighbours consists in finding its K closest samples according to the Euclidean distance. Each neighbour is weighted by considering the reciprocal of the Euclidean distance. The predicted output $\hat{y}$ is given by $\hat{y} = \sum_{i=1}^{K} w_i y_i / \sum_{i=1}^{K} w_i$, where w_i and y_i are the weight and the ground-truth output for the ith neighbour.

The number of neighbours K is an hyper-parameter which needs to be properly tuned: to this end, different K-NN models have been synthesized by bruteforcing the number of neighbours $K = \{1, 2, \ldots, 21\}$: the best K is the one that minimizes the RMSE on the VL.

MLP: In the late 80's/early 90's the Cybenko theorem proved the universal approximation capabilities of feedforward ANNs [48,49], which have been widely used for solving non-linear function approximation problems. The second competitor is a feedforward MLP with the following hyper-parameters to be tuned: number of layers in $\{1, 2, 3\}$, number of neurons per layer in $\{2, 3, \ldots, 100\}$, learning rate in $[0, 0.9]$, hidden layers activation function to be chosen between six suitable candidates[3] and training algorithm to be chosen between standard backpropagation and adaptive backpropagation.

Conversely to the K-NN case, where the only parameter to be tuned is fairly bounded and integer-valued, a GA has been used in order to find a suitable MLP architecture. Each individual exploits the configuration written in its genetic code in order to generate a candidate MLP which is trained on TR and its performances are evaluated on the VL. As the evolution ends, the best individual is retained and tested on TS. The GA has been configured as follows: the initial population has been randomly generated considering a population of 50 individuals; the maximum number of generation has been set to 25 and in case the fitness function does not decrease of $\sim 10^{-5}$ for at least 4 generations, the optimization procedure halts. Standard crossover, mutation, elitism and selection operators take care of moving towards the next generation.

SVR: SVMs aim at finding the maximal margin hyperplane in order to separate two classes. For function approximation tasks, the hyperplane is found in such a way that the training data is approximated within a given precision. Specifically, a kernelized ν-SVR equipped with the radial basis function kernel is considered. The penalty term $\nu \in (0, 1]$ and the kernel shape parameter $\gamma \in (0, 100]$ are the two hyper-parameters to be tuned.

As per the MLP case, a GA has been used in order to find suitable values for ν and γ. The optimization procedure does not change with respect to the former case.

5. Computational results

5.1. Datasets

Different datasets available in the literature have been used for studying the efficiency and effectiveness of the algorithms discussed in this paper. These datasets are detailed in Table 1,[4] by remarking the dimensionality of the input space (number of variables) and the range of candidate clusters considered for each ANFIS synthesis test. Each dataset has been split in TR, VL and TS: in most cases, the training, validation, and test split has been performed by keeping the proportions of 50%, 25% and 25% for TR, VL and TS, respectively. For NASA, as instead, the dataset partitioning has been kept as in [50].

5.2. Results

The ANFIS tests have been distinguished per dataset, dissimilarity function $\mathscr{D}$, clustering algorithm $\mathscr{C}$, and classification algorithm $\mathscr{P}$.

[3] Logarithmic sigmoid, ReLU, linear, radial basis, soft-max, symmetric sigmoid.

[4] *MGFC* dataset is not yet available since the reference paper is under submission.

Table 1
Details of the datasets considered for testing.

Dataset	Ref.	TR set size	VL set size	TS set size	#vars	Clusters range
NASA	[51]	18e3	6e3	21e3	4	1–20
abalone	[52]	2e3	1e3	1e3	8	1–20
building	[53]	2.1e3	1e3	1e3	14	1–20
heart	[53]	4.5e2	2.3e2	2.3e2	35	1-11
mgdata	[52]	6.9e2	3.5e2	3.5e2	6	1–20
MGFC		2.8e3	1.3e3	1.3e3	6	1–20
mpgdata	[52]	1.9e2	9.7e1	9.6e1	7	1–16
space ga	[54]	1.5e3	7.7e2	7.7e2	6	1–20
PCN	[55]	2.4e3	1.2e3	1.2e3	348	1–6

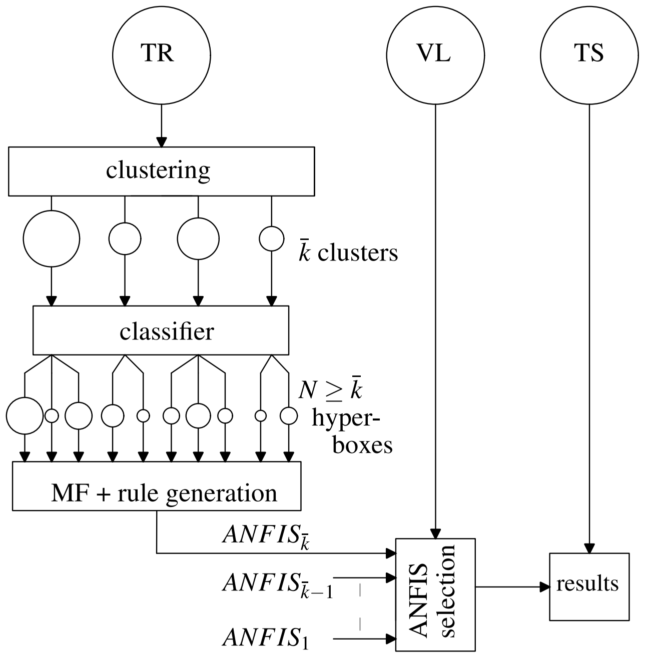


Fig. 6. Scheme of the ANFIS synthesis procedure.

The software has been implemented in MATLAB® R2019a on a workstation equipped with two 6-cores Intel® Xeon® E5-2630 CPUs at 2.60 GHz and 64 GB RAM.

The overall results achieved by the ANFIS configurations are shown in Table 2, Table 3, and Table 4, where the resulting ε value, R^2 on the TS and number of rules are listed, respectively. For each configuration and dataset, the tables report the results of the best synthesized ANFIS according to the procedure described in Fig. 6, *i.e.* the one that achieves the best performances in VL. For each dataset, the configuration achieving the best R^2 on the TS is highlighted in bold red. A dagger (†) indicates that the corresponding tests have failed to return a suitable ANFIS model due to out-of-memory error on the considered hardware. A double-dagger (‡) indicates that the corresponding tests have failed to return a suitable ANFIS model within a 24-h deadline on the considered hardware. The former case regards NASA, by far the largest dataset in terms of number of patterns, and usually happens when dealing with partitional or agglomerative clustering techniques. Indeed, these techniques exploit multi-threaded parallelism in order to perform in parallel different replicates of the (partitional) clustering procedure with different initial seeds, which might trigger out-of-memory errors when dealing with large datasets. The latter case regards PCN, by far the dataset with greatest number of features, especially when the GPARC classifier is involved: as already mentioned in Section 3.3, GPARC extensively leverages on singular value decompositions for properly rotating the hyperboxes and by considering the computational complexity for solving the eigencomposition problem (*i.e.*, $\Omega(nm^2+mn^2)$ for an $n\times m$ set), this can lead to non-negligible training times.

Concerning the clustering space, it can be seen in Table 2 that there is not a predominant configuration for the ε value. Indeed, $\varepsilon = 1$ emerged as the most suitable value 89 times over 162, whereas $\varepsilon = 0.05$ appears 73 times. This parameter seems to be highly dependent on the specific dataset, suggesting that for ABALONE, BUILDING, MGDATA, MPGDATA and SPACEGA the input space only is preferable, whereas in NASA, MGFC, PCN $\varepsilon = 0.05$ is preferred.

Concerning the clustering algorithms, the divisive method is the most performing, appearing among the best configurations in 5 datasets over 9 (NASA, ABALONE, BUILDING, HEART and MPGDATA). The agglomerative variant appears as the most performing clustering strategy only for SPACEGA, whereas the partitional clustering is the best for PCN, MGFC and MGDATA. Following the same reasoning, the most performing dissimilarity measure is SQE, achieving the best performances in 6 datasets (NASA, ABALONE, HEART, MGFC, SPACEGA and PCN). Finally, concerning the Min–Max classifier, for 4 datasets over 9 the best results have been achieved without the Min–Max postprocessing. It is interesting to note that where the Min–Max has been beneficial, in the vast majority of the cases, the parameter ε was set to 1, according to the fact that the Min–Max classifier helps at refining the solutions found by a pure input space clustering. All in all, from the above discussion it appears that the best configuration employs the divisive algorithm, with the classification post-processing.

In order to verify the effectiveness of the proposed ANFIS synthesis techniques, Table 5 shows the comparison among the best synthesized ANFISs and the benchmark models (SVR, MLP, and K-NN) in terms of model performance (R^2 and RMSE on the TS), the running times for both training and testing phases and the model complexity. The latter is defined as the number of neurons per layer for MLP, the number of patterns elected as support vectors for SVR, and the number of synthesized rules for ANFIS. It is important to note that, despite the number of neighbours K is shown in Table 5, the proper complexity of K-NN is always related to the number of training data, since no training phase (data compression) is performed: indeed, regardless of the number of neighbours, K-NN performs a lazy evaluation of all pairwise distances between the test pattern and the training data. This intrinsically results in K-NN being the worst technique in terms of model complexity.

The worst ANFIS training times are referred to the Heart dataset (the best ANFIS has been generated by using GPARC) and the NASA dataset, which is featured by the bigger TR. With the exception of these two cases, the ANFIS training procedure is approximately restricted up to few tens of seconds, whereas the MLP and SVR models are mostly featured by training time of hundreds or even thousands of seconds, respectively.

For a better comparison among the selected ANFIS and the benchmark solutions, a box-plot of the R^2 results of Table 3 is

Table 2
Best ε parameter between the two tested values $\varepsilon \in \{0.05, 1\}$ for each ANFIS configuration. In bold red, for each dataset, the value corresponding to the configuration that returned the highest R^2 (see Table 3).

NASA		†	0.05	0.05	0.05	†	0.05	†	0.05	0.05	**0.05**	†	0.05	†	1	0.05	0.05	†	0.05
abalone		1	1	1	1	0.05	1	1	1	0.05	**1**	1	1	1	1	1	1	1	1
building		0.05	1	1	1	0.05	1	1	1	**1**	1	0.05	1	0.05	1	0.05	1	1	1
heart		0.05	0.05	1	**1**	0.05	1	0.05	0.05	1	1	1	1	1	1	0.05	0.05	0.05	0.05
mgdata		1	1	1	1	0.05	1	0.05	1	1	1	0.05	1	1	1	1	1	1	**1**
MGFC		0.05	0.05	1	0.05	0.05	0.05	0.05	0.05	0.05	1	0.05	1	0.05	1	0.05	0.05	1	**1**
mpgdata		1	0.05	1	1	0.05	0.05	1	1	**1**	0.05	1	0.05	1	1	1	1	0.05	0.05
spacega		1	1	1	1	0.05	1	1	1	1	1	1	1	1	**1**	1	1	0.05	1
PCN		‡	‡	‡	‡	‡	‡	0.05	0.05	1	0.05	0.05	0.05	1	0.05	0.05	0.05	1	**0.05**
Dataset test	$\mathscr{C}$	AGG	AGG	DIV	DIV	PRT	PRT	AGG	AGG	DIV	DIV	PRT	PRT	AGG	AGG	DIV	DIV	PRT	PRT
	$\mathscr{D}$	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE
	$\mathscr{P}$	GPA	GPA	GPA	GPA	GPA	GPA	PAR	PAR	PAR	PAR	PAR	PAR	–	–	–	–	–	–

Table 3
ANFIS R^2 values considering each test. In bold red, the best (highest) result for each dataset.

NASA		†	1.00	0.98	0.99	†	0.99	†	1.00	1.00	**1.00**	†	1.00	†	0.99	1.00	1.00	†	0.99
abalone		0.46	0.46	0.53	0.54	0.52	0.54	0.50	0.50	0.50	**0.54**	0.51	0.50	0.53	0.53	0.53	0.54	0.54	0.52
building		0.29	0.77	0.80	0.55	0.63	0.54	0.86	0.82	**0.86**	0.75	0.78	0.80	0.81	0.78	0.10	0.49	0.75	0.78
heart		0.50	0.47	0.49	**0.52**	0.29	0.51	0.27	0.28	0.26	0.41	0.47	0.49	0.49	0.50	0.49	0.49	0.49	0.49
mgdata		0.57	0.57	0.62	0.63	0.60	0.60	0.64	0.57	0.64	0.65	0.65	0.66	0.66	0.66	0.63	0.65	0.62	**0.70**
MGFC		0.77	0.78	0.77	0.79	0.79	0.81	0.81	0.82	0.81	0.80	0.82	0.81	0.82	0.81	0.82	0.82	0.82	**0.82**
mpgdata		0.84	0.88	0.88	0.88	0.84	0.88	0.87	0.87	**0.89**	0.87	0.85	0.86	0.82	0.85	0.88	0.87	0.88	0.88
spacega		0.67	0.66	0.67	0.69	0.63	0.69	0.71	0.68	0.66	0.67	0.68	0.69	0.65	**0.71**	0.64	0.63	0.69	0.69
PCN		‡	‡	‡	‡	‡	‡	0.00	0.38	0.00	0.01	0.04	0.35	0.08	0.37	0.24	0.24	0.30	**0.44**
Dataset test	$\mathscr{C}$	AGG	AGG	DIV	DIV	PRT	PRT	AGG	AGG	DIV	DIV	PRT	PRT	AGG	AGG	DIV	DIV	PRT	PRT
	$\mathscr{D}$	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE
	$\mathscr{P}$	GPA	GPA	GPA	GPA	GPA	GPA	PAR	PAR	PAR	PAR	PAR	PAR	–	–	–	–	–	–

Table 4
ANFIS numbers of rules. In bold red, for each dataset, the value corresponding to the configuration that returned the highest R^2 (see Table 3).

NASA		†	66	159	98	†	95	†	46	108	**26**	†	55	†	19	2	2	†	20
abalone		35	45	10	10	8	12	26	36	57	**9**	8	39	10	5	2	2	15	9
building		6	3	5	2	6	3	13	12	**11**	4	4	5	2	3	2	2	4	5
heart		2	4	2	**2**	2	2	28	36	11	11	3	8	2	2	1	1	1	1
mgdata		27	16	10	20	19	14	14	14	9	20	21	22	10	18	3	4	11	**11**
MGFC		22	55	19	57	24	49	15	32	25	23	21	11	3	13	2	1	6	**9**
mpgdata		5	2	3	2	4	3	8	2	**3**	2	6	3	7	5	2	2	3	4
spacega		24	32	24	20	37	20	49	75	56	68	75	75	7	**14**	2	2	9	14
PCN		‡	‡	‡	‡	‡	‡	202	16	431	148	173	19	3	2	1	1	2	**3**
Dataset test	$\mathscr{C}$	AGG	AGG	DIV	DIV	PRT	PRT	AGG	AGG	DIV	DIV	PRT	PRT	AGG	AGG	DIV	DIV	PRT	PRT
	$\mathscr{D}$	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE	MAH	SQE
	$\mathscr{P}$	GPA	GPA	GPA	GPA	GPA	GPA	PAR	PAR	PAR	PAR	PAR	PAR	–	–	–	–	–	–

shown in Fig. 7 together with benchmark results of Table 5. The box-plot captures the distribution of the performances obtained on a given dataset across all tested configurations (ε, $\mathscr{D}$, $\mathscr{C}$, $\mathscr{P}$). In a similar way, Fig. 8 depicts the complexity: the number of rules (ANFIS), alongside the number of neurons (MLP) and number of support vectors (SVR). By jointly considering Table 5 and Figs. 7–8, it is safe to summarize our experiments by saying that the (best) ANFIS models outperform the three competitors in terms of R^2 for the vast majority of the considered datasets, while at the same time keeping a low computational complexity.

6. Conclusions

In industrial applications relying on control systems or decision support systems, not only effectiveness and robustness, but also complexity and interpretability by human operators are key features that must be addressed when designing suitable data-driven solutions to engineering problems.

In literature, ANFISs are often mentioned due to their potentials in solving non-linear supervised modelling problems, underlining their rule-based structure which can contribute to a better comprehension of how the model takes its decisions, peculiarity which rarely holds in other machine learning models.

In this paper, we investigated advanced synthesis procedures based on k-clustering algorithms for training ANFISs models for function approximation, equipped with bell-shaped multidimensional MFs. Starting from a straightforward synthesis procedure based on a plain partitional k-means clustering which explores a range of partitions obtained by increasing number of clusters, different variants have been implemented in order to stress the problem under analysis and to observe possible upgrades of such kind of paradigm. Indeed the plain k-means is performed to discover clusters (*i.e.* rule antecedents, and thus MFs) considering distance measures (such as the Euclidean one) defined in the plain input space, not considering explicitly the output information associated to each training pattern. In order to overcome this limitation, we have carried out an exhaustive investigation exploring:

(i) some hierarchical clustering variants;
(ii) the use of the Mahalanobis distance, in order to improve non-spherical clusters modelling;
(iii) hyperplane space clustering procedures, in order to improve the overall piece-wise linear approximation of resulting ANFIS models;
(iv) the downstream support of neuro-fuzzy Min–Max classifiers for MFs refinement.

These variants have been tested on different datasets available in the literature and then compared with three benchmark solutions: a K-NN approximator and GA-optimized SVR and MLP.

Table 5
MLP, K-NN, SVR and best ANFIS results reported in terms of R^2 and the RMSE on the TS, the model complexity, the TS operational time and the model training time. Concerning the model complexity, #Nrs, #SVs and #Rules indicate the number of neurons (MLP), the number of support vectors (SVR) and the number of rules (ANFIS), respectively. For K-NN, we also show the number of neighbours, albeit it should not be interpreted as a proper measure of structural complexity. In bold, for each dataset, the most performing method according to the R^2.

SVR Dataset	RMSE	R^2	Operational Time [s]	Training Time [s]	Complexity #SVs
NASA	0.04	0.98	0.06	79924.7	96
abalone	0.15	0.52	0.04	232.2	84
building	0.14	0.73	0.01	596.4	101
heart	0.27	0.51	0.01	37.2	181
mgdata	0.16	0.59	0.00	30.1	29
MGFC	0.09	0.81	0.01	1163.7	78
mpgdata	0.08	0.88	0.01	4.9	41
spacega	0.11	0.57	0.00	109.0	30
PCN	0.31	0.03	0.04	926.9	246
MLP	RMSE	R^2	TS Op. Time [s]	TR Time [s]	#NRs/Layer
NASA	0.02	0.99	0.04	4371.0	14 [13 1 0]
abalone	0.12	0.53	0.02	1396.0	24 [23 1 0]
building	0.22	0.15	0.02	2520.0	67 [66 1 0]
heart	0.28	0.47	0.02	1829.0	104 [60 43 1]
mgdata	0.15	0.65	0.02	8976.0	26 [25 1 0]
MGFC	0.05	**0.83**	0.02	7856.0	34 [33 1 0]
mpgdata	0.08	0.88	0.01	6416.0	35 [34 1 0]
spacega	0.10	0.66	0.01	4928.0	46 [45 1 0]
PCN	0.27	0.27	0.04	3582.0	100 [77 22 1]
K-**NN**	RMSE	R^2	TS Op. Time [s]	TR Time [s]	K
NASA	0.05	0.97	14.56	–	4
abalone	0.13	0.50	0.23	–	14
building	0.10	**0.89**	0.08	–	21
heart	0.26	0.51	0.01	–	18
mgdata	0.14	**0.70**	0.01	–	21
MGFC	0.05	0.82	0.11	–	9
mpgdata	0.09	0.82	0.02	–	3
spacega	0.10	0.63	0.04	–	4
PCN	0.18	**0.66**	0.21	–	5
ANFIS	RMSE	R^2	TS Op. Time [s]	TR Time [s]	#Rules
NASA	0.01	**1**	25.14	396.0	26
abalone	0.12	**0.54**	0.46	73.0	9
building	0.13	0.86	0.62	49.5	11
heart	0.19	**0.52**	0.01	901.0	2
mgdata	0.14	**0.70**	0.18	3.4	11
MGFC	0.05	0.82	0.59	14.4	9
mpgdata	0.07	**0.89**	0.01	1.8	3
spacega	0.09	**0.71**	0.52	7.8	14
PCN	0.23	0.44	5.85	33.2	3

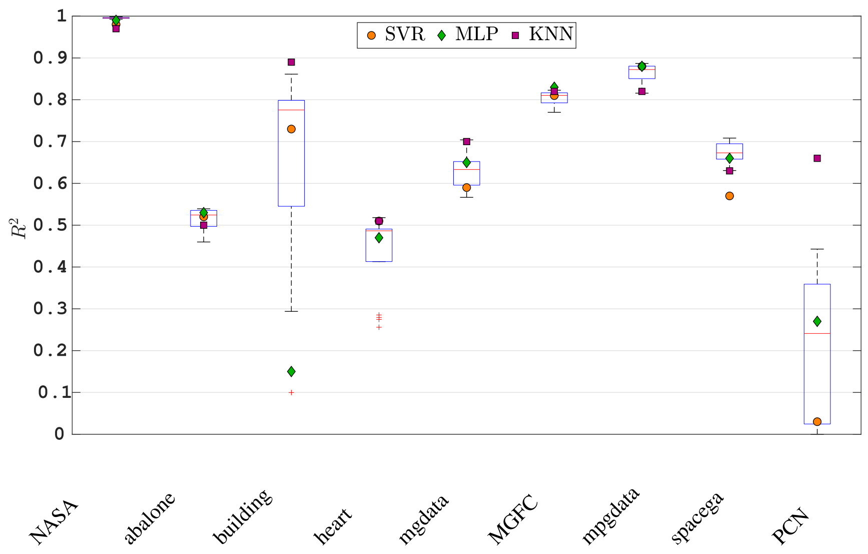


Fig. 7. R^2 results. The boxplot shows the ANFIS results from Table 3, whereas competitors results refer to Table 5.

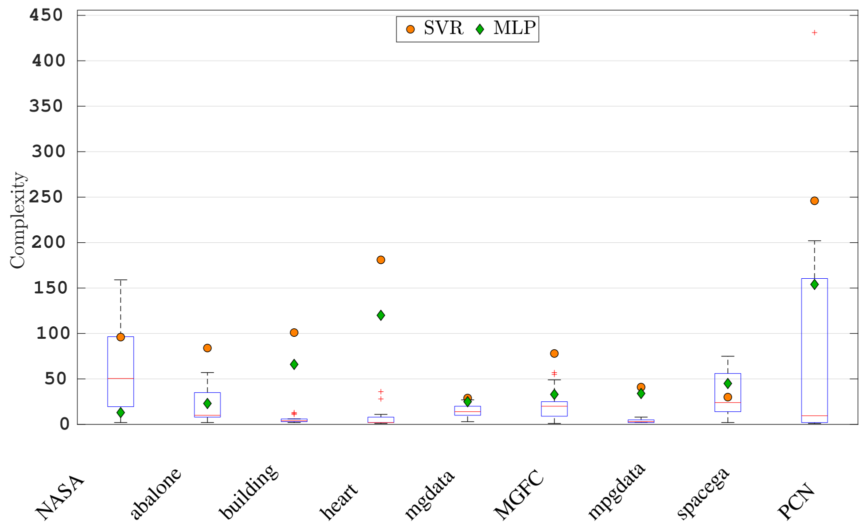


Fig. 8. Structural complexity results. The boxplot shows the ANFIS results from Table 4, whereas competitors results refer to Table 5.

Computational results show that ANFIS model outperform on average the considered competitors, showing at the same time a remarkably lower model complexity and training time.

The ANFIS synthesis procedure herein proposed has to be considered as a general framework and can easily be personalized according to the data and problem at hand. In fact, the choice behind *k*-means-like clustering procedures, possibly in an hyperplane clustering configuration, and the use of neuro-fuzzy classifiers for MFs refining should be considered just as first effective components, as suggested in previous works (*e.g.*, [37,38,46]): nonetheless, virtually any clustering algorithm capable of being equipped with ad hoc dissimilarity measures can be placed in the clustering block and any classifier able to perform a tessellation of the input space can be placed in lieu of the Min–Max classifier in the classification block (cf. Fig. 6).

A complete software suite implementing the proposed framework is freely available on *GitLab* under an open-source licence. The software is a general tool for training ANFIS function approximation models by derivative free clustering techniques. The library is written in MATLAB® by following an Object-Oriented Programming paradigm. It allows to easily customize most of training parameters for ANFIS synthesis. Specifically, it is possible to chose the clustering algorithm, the dissimilarity measure, the cluster representative and to define the clustering space ranging from the plain input space to the pure hyperplane space. The tool is also equipped with different Min–Max classifiers, implemented in *C++* and available as pre-compiled *.exe* files, for possible running, downstream the clustering phase, a refining procedure of ANFIS Membership Functions.

Future research efforts will focus in investigating different strategies in this remark. Moreover, future works foresee the application of the proposed ANFIS synthesis approach in real problems by also considering training procedures able to track time-variant systems.

CRediT authorship contribution statement

Stefano Leonori: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Visualization. **Alessio Martino:** Methodology, Software, Validation, Formal analysis, Writing - original draft, Writing - review & editing. **Massimiliano Luzi:** Methodology, Software, Formal analysis, Writing - original draft, Writing - review & editing. **Fabio Massimo Frattale Mascioli:** Investigation, Supervision. **Antonello Rizzi:** Conceptualization, Investigation, Writing - review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] C.A. Peña-Reyes, Evolutionary fuzzy modeling human diagnostic decisions, Ann. New York Acad. Sci. 1020 (2004) 190—211, http://dx.doi.org/10.1196/annals.1310.017.

[2] A. Asgharnia, R. Shahnazi, A. Jamali, Performance and robustness of optimal fractional fuzzy PID controllers for pitch control of a wind turbine using chaotic optimization algorithms, ISA Trans. 79 (2018) 27–44, http://dx.doi.org/10.1016/j.isatra.2018.04.016, URL http://www.sciencedirect.com/science/article/pii/S0019057818301629.

[3] A. Youssef, M. El-Telbany, A. Zekry, The role of artificial intelligence in photo-voltaic systems design and control: A review, Renew. Sustain. Energy Rev. 78 (2017) 72–79, http://dx.doi.org/10.1016/j.rser.2017.04.046, URL http://www.sciencedirect.com/science/article/pii/S1364032117305555.

[4] G.N. Pillai, J. Pushpak, M.G. Nisha, Extreme learning ANFIS for control applications, in: 2014 IEEE Symposium on Computational Intelligence in Control and Automation, CICA, 2014, pp. 1–8, http://dx.doi.org/10.1109/CICA.2014.7013226.

[5] H. Yang, Y. Fu, D. Wang, Multi-ANFIS model based synchronous tracking control of high-speed electric multiple unit, IEEE Trans. Fuzzy Syst. 26 (3) (2018) 1472–1484, http://dx.doi.org/10.1109/TFUZZ.2017.2725819.

[6] D. Arcos-Aviles, F. Guinjoan, J. Pascual, L. Marroyo, P. Sanchis, R. Gordillo, P. Ayala, M.P. Marietta, A review of fuzzy-based residential grid-connected microgrid energy management strategies for grid power profile smoothing, in: Energy Sustainability in Built and Urban Environments, Springer, 2019, pp. 165–199.

[7] D. Espí n Sarzosa, R. Palma-Behnke, O. Núñez Mata, Energy management systems for microgrids: Main existing trends in centralized control architectures, Energies 13 (3) (2020) 547, http://dx.doi.org/10.3390/en13030547.

[8] S. Leonori, M. Paschero, F.M. Frattale Mascioli, A. Rizzi, Optimization strategies for Microgrid energy management systems by Genetic Algorithms, Appl. Soft Comput. (2019) 105903, http://dx.doi.org/10.1016/j.asoc.2019.105903.

[9] S. Leonori, A. Rizzi, M. Paschero, F. Mascioli, Microgrid energy management by anfis supported by an esn based prediction algorithm, in: 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1–8.

[10] S. Leonori, A. Martino, F.M. Frattale Mascioli, A. Rizzi, Microgrid energy management systems design by computational intelligence techniques, Applied Energy 277 (2020) 115524, http://dx.doi.org/10.1016/j.apenergy.2020.115524.

[11] J. Soto, P. Melin, O. Castillo, A new approach for time series prediction using ensembles of IT2FNN models with optimization of fuzzy integrators, Int. J. Fuzzy Syst. 20 (3) (2018) 701–728, http://dx.doi.org/10.1007/s40815-017-0443-6.

[12] P. Melin, J. Soto, O. Castillo, J. Soria, A new approach for time series prediction using ensembles of ANFIS models, Expert Syst. Appl. 39 (3) (2012) 3494–3506, http://dx.doi.org/10.1016/j.eswa.2011.09.040, URL http://www.sciencedirect.com/science/article/pii/S0957417411013492.

[13] A. Mardani, Y. Van Fan, M. Nilashi, R.E. Hooker, S. Ozkul, D. Streimikiene, N. Loganathan, A two-stage methodology based on ensemble adaptive neuro-fuzzy inference system to predict carbon dioxide emissions, Journal of Cleaner Production 231 (2019) 446–461, http://dx.doi.org/10.1016/j.jclepro.2019.05.153.

[14] G. Montavon, W. Samek, K.-R. Müller, Methods for interpreting and understanding deep neural networks, Digit. Signal Process. 73 (2018) 1–15, http://dx.doi.org/10.1016/j.dsp.2017.10.011, URL http://www.sciencedirect.com/science/article/pii/S1051200417302385.

[15] S. Guillaume, Designing fuzzy inference systems from data: An interpretability-oriented review, IEEE Trans. Fuzzy Syst. 9 (3) (2001) 426–443, http://dx.doi.org/10.1109/91.928739.

[16] T.R. Razak, J.M. Garibaldi, C. Wagner, A. Pourabdollah, D. Soria, Interpretability indices for hierarchical fuzzy systems, in: 2017 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE, 2017, pp. 1–6.

[17] J. Casillas, O. Cordon, F. Herrera, L. Magdalena, Accuracy improvements to find the balance interpretability-accuracy in linguistic fuzzy modeling: An overview, in: Interpretability Improvements to Find the Balance Interpretability-Accuracy in Fuzzy Modeling: An Overview, 2008, http://dx.doi.org/10.1007/978-3-540-37057-4_1.

[18] B. Bede, Fuzzy systems with sigmoid-based membership functions as interpretable neural networks, in: R.B. Kearfott, I. Batyrshin, M. Reformat, M. Ceberio, V. Kreinovich (Eds.), Fuzzy Techniques: Theory and Applications, Springer International Publishing, Cham, 2019, pp. 157–166.

[19] N. Walia, H. Singh, A. Sharma, ANFIS: Adaptive neuro-fuzzy inference system-a survey, Int. J. Comput. Appl. 123 (13) (2015).

[20] J.-S.R. Jang, C.-T. Sun, Neuro-fuzzy modeling and control, Proc. IEEE 83 (3) (1995) 378–406.

[21] J.-S.R. Jang, ANFIS: adaptive-network-based fuzzy inference system, IEEE Trans. Syst. Man Cybern. 23 (3) (1993) 665–685.

[22] E. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, Int. J. Man-Mach. Stud. 7 (1) (1975) 1–13, http://dx.doi.org/10.1016/S0020-7373(75)80002-2.

[23] M. Gacto, R. Alcalá, F. Herrera, Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures, in: Special Issue on Interpretable Fuzzy Systems, Inform. Sci. 181 (20) (2011) 4340–4360, http://dx.doi.org/10.1016/j.ins.2011.02.021, URL http://www.sciencedirect.com/science/article/pii/S0020025511001034.

[24] L. Olatomiwa, S. Mekhilef, M. Ismail, M. Moghavvemi, Energy management strategies in hybrid renewable energy systems: A review, Renew. Sustain. Energy Rev. 62 (2016) 821–835, http://dx.doi.org/10.1016/j.rser.2016.05.040, URL http://www.sciencedirect.com/science/article/pii/S1364032116301502.

[25] L. Meng, E.R. Sanseverino, A. Luna, T. Dragicevic, J.C. Vasquez, J.M. Guerrero, Microgrid supervisory controllers and energy management systems: A literature review, Renew. Sustain. Energy Rev. 60 (2016) 1263–1273, http://dx.doi.org/10.1016/j.rser.2016.03.003, URL http://www.sciencedirect.com/science/article/pii/S1364032116002380.

[26] I. Pan, S. Das, Fractional order fuzzy control of hybrid power system with renewable generation using chaotic PSO, in: SI: Control of Renewable Energy Systems, ISA Trans. 62 (2016) 19–29, http://dx.doi.org/10.1016/j.isatra.2015.03.003, URL http://www.sciencedirect.com/science/article/pii/S001905781500083X.

[27] J. Soto, P. Melin, O. Castillo, Particle swarm optimization of the fuzzy integrators for time series prediction using ensemble of IT2FNN architectures, 667, ISBN: 978-3-319-47053-5, 2017, pp. 141–158, http://dx.doi.org/10.1007/978-3-319-47054-2_9.

[28] E. Lughofer, Evolving fuzzy systems: Fundamentals, reliability, interpretability, useability and applications, in: 2015 7th International Joint Conference on Computational Intelligence, Vol. 1, IJCCI, 2015, p. 11.

[29] E. Lughofer, Evolving fuzzy systems—fundamentals, reliability, interpretability, useability, applications, in: Handbook on Computational Intelligence, pp. 67–135, http://dx.doi.org/10.1142/9789814675017_0003, URL https://www.worldscientific.com/doi/abs/10.1142/9789814675017_0003.

[30] E. Lughofer, M. Sayed-Mouchaweh, Autonomous data stream clustering implementing split-and-merge concepts – Towards a plug-and-play approach, Inform. Sci. 304 (2015) 54–79, http://dx.doi.org/10.1016/j.ins.2015.01.010, URL http://www.sciencedirect.com/science/article/pii/S0020025515000328.

[31] M. Luzi, M. Paschero, A. Rizzi, F.M. Frattale Mascioli, An ANFIS based system identification procedure for modeling electrochemical cells, in: 2018 International Joint Conference on Neural Networks, IJCNN, 2018, pp. 1–8, http://dx.doi.org/10.1109/IJCNN.2018.8489250.

[32] G. Heydari, A. Gharaveisi, M. Vali, New formulation for representing higher order TSK fuzzy systems, IEEE Trans. Fuzzy Syst. 24 (4) (2016) 854–864, http://dx.doi.org/10.1109/TFUZZ.2015.2486813.

[33] S.D. Nguyen, S. Choi, T. Seo, Recurrent mechanism and impulse noise filter for establishing ANFIS, IEEE Trans. Fuzzy Syst. 26 (2) (2018) 985–997, http://dx.doi.org/10.1109/TFUZZ.2017.2701313.

[34] P. Jagtap, G.N. Pillai, Comparison of extreme-ANFIS and ANFIS networks for regression problems, in: 2014 IEEE International Advance Computing Conference, IACC, 2014, pp. 1190–1194, http://dx.doi.org/10.1109/IAdCC.2014.6779496.

[35] J.d.J. Rubio, SOFMLS: Online self-organizing fuzzy modified least-squares network, IEEE Trans. Fuzzy Syst. 17 (6) (2009) 1296–1309, http://dx.doi.org/10.1109/TFUZZ.2009.2029569.

[36] J.d.J. Rubio, D. Cruz, I. Elias Barrón, G. Ochoa, R. Balcazarand, A. Aguilar, ANFIS system for classification of brain signals, J. Intell. Fuzzy Systems 37 (2019) 4033–4041, http://dx.doi.org/10.3233/JIFS-190207.

[37] S. Leonori, A. Martino, F.M. Frattale Mascioli, A. Rizzi, ANFIS microgrid energy management system synthesis by hyperplane clustering supported by neurofuzzy min–max classifier, IEEE Trans. Emerg. Top. Comput. Intell. 3 (3) (2019) 193–204, http://dx.doi.org/10.1109/TETCI.2018.2880815.

[38] M. Panella, A. Rizzi, F.M. Frattale Mascioli, G. Martinelli, ANFIS synthesis by hyperplane clustering, in: Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference, Vol. 1, 2001, pp. 340–345, http://dx.doi.org/10.1109/NAFIPS.2001.944275.

[39] A. Rizzi, M. Panella, F.M. Frattale Mascioli, Adaptive resolution min-max classifiers, IEEE Trans. Neural Netw. 13 (2002) 402–414, http://dx.doi.org/10.1109/72.991426.

[40] B. Schölkopf, A.J. Smola, R.C. Williamson, P.L. Bartlett, New support vector algorithms, Neural Comput. 12 (5) (2000) 1207–1245, http://dx.doi.org/10.1162/089976600300015565.

[41] N.S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, Amer. Statist. 46 (3) (1992) 175–185, http://dx.doi.org/10.1080/00031305.1992.10475879.

[42] S. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (2) (1982) 129–137.

[43] A. Martino, A. Rizzi, F.M. Frattale Mascioli, Efficient approaches for solving the large-scale k-medoids problem: towards structured data, in: C. Sabourin, J.J. Merelo, K. Madani, K. Warwick (Eds.), Computational Intelligence: 9th International Joint Conference, IJCCI 2017 Funchal-Madeira, Portugal, November 1-3, 2017 Revised Selected Papers, Springer International Publishing, Cham, 2019, pp. 199–219, http://dx.doi.org/10.1007/978-3-030-16469-0_11.

[44] A. Martino, A. Giuliani, A. Rizzi, Granular computing techniques for bioinformatics pattern recognition problems in non-metric spaces, in: W. Pedrycz, S.-M. Chen (Eds.), Computational Intelligence for Pattern Recognition, in: Studies in Computational Intelligence, vol. 777, Springer International Publishing, Cham, 2018, pp. 53–81.

[45] A. Martino, A. Rizzi, F.M. Frattale Mascioli, Efficient approaches for solving the large-scale k-medoids problem, in: Proceedings of the 9th International Joint Conference on Computational Intelligence, Vol. 1, IJCCI, SciTePress, INSTICC, 2017, pp. 338–347, http://dx.doi.org/10.5220/0006515003380347.

[46] S. Leonori, A. Martino, A. Rizzi, F.M. Frattale Mascioli, ANFIS synthesis by clustering for microgrids EMS design, in: Proceedings of the 9th International Joint Conference on Computational Intelligence, Vol. 1, IJCCI, SciTePress, INSTICC, 2017, pp. 328–337, http://dx.doi.org/10.5220/0006514903280337.

[47] A. Rizzi, Automatic training of min-max classifiers, in: H. Bunke, A. Kandel (Eds.), Neuro-Fuzzy Pattern Recognition, 2000, pp. 101–124, http://dx.doi.org/10.1142/9789812792204_0005.

[48] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Systems 2 (4) (1989) 303–314, http://dx.doi.org/10.1007/BF02551274.

[49] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (2) (1991) 251–257, http://dx.doi.org/10.1016/0893-6080(91)90009-T.

[50] M. Luzi, M. Paschero, A. Rizzi, E. Maiorino, F.M. Frattale Mascioli, A novel neural networks ensemble approach for modeling electrochemical cells, IEEE Trans. Neural Netw. Learn. Syst. 30 (2) (2019) 343–354, http://dx.doi.org/10.1109/TNNLS.2018.2827307.

[51] B. Bole, C. Kulkarni, M. Daigle, Randomized Battery Usage Data Set, Tech. Rep., NASA Ames Prognostics Data Repository, CA, NASA Ames Research Center, Moffett Field, 2014, URL.

[52] D. Dua, C. Graff, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, URL http://archive.ics.uci.edu/ml.

[53] L. Prechelt, PROBEN 1 - A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms, Tech. Rep., Fakultät für Informatik, Universität Karlsruhe, 1994.

[54] R.K. Pace, R. Barry, Quick computation of spatial autoregressive estimators, Geogr. Anal. 29 (3) (1997) 232–247, http://dx.doi.org/10.1111/j.1538-4632.1997.tb00959.x.

[55] A. Martino, A. Giuliani, A. Rizzi, (Hyper)Graph embedding and classification via simplicial complexes, Algorithms 12 (11) (2019) http://dx.doi.org/10.3390/a12110223.