SK-MOEFS: A Library in Python for Designing Accurate and Explainable Fuzzy Models

Gionatan Gallo, Vincenzo Ferrari, Francesco Marcelloni, and Pietro Ducange(✉)

Department of Information Engineering, University of Pisa, Largo Lucio Lazzarino 1, Pisa, Italy
{gionatan.gallo,vincenzo.ferrari,francesco.marcelloni, pietro.ducange}@unipi.it

Abstract. Recently, the explainability of Artificial Intelligence (AI) models and algorithms is becoming an important requirement in real-world applications. Indeed, although AI allows us to address and solve very difficult and complicated problems, AI-based tools act as a black box and, usually, do not explain how/why/when a specific decision has been taken. Among AI models, Fuzzy Rule-Based Systems (FRBSs) are recognized world-wide as transparent and interpretable tools: they can provide explanations in terms of linguistic rules. Moreover, FRBSs may achieve accuracy comparable to those achieved by less transparent models, such as neural networks and statistical models. In this work, we introduce SK-MOEFS (acronym of SciKit-Multi Objective Evolutionary Fuzzy System), a new Python library that allows the user to easily and quickly design FRBSs, employing Multi-Objective Evolutionary Algorithms. Indeed, a set of FRBSs, characterized by different trade-offs between their accuracy and their explainability, can be generated by SK-MOEFS. The user, then, will be able to select the most suitable model for his/her specific application.

Keywords: Explainable Artificial Intelligence · Multi-objective Evolutionary Algorithms · Fuzzy Rule-Based Systems · Python · Scikit-Learn

1 Introduction

The proliferation of Artificial Intelligence (AI) has a significant impact on society [1]. Indeed, AI has already become ubiquitous in personal life and the modern industry. As regards the latter, we are experiencing the "Industry 4.0 Era", and Machine Learning (ML) and AI play a crucial role among its enabling technologies

This work was supported by the Italian Ministry of Education and Research (MIUR), in the framework of the CrossLab project (Departments of Excellence).

[12]. Models based on ML and AI are learnt from the input data and are generally very accurate. However, in most cases, they are highly non-transparent, *i.e.*, it is not clear which information in the input data causes the generated output. In the context of Industry 4.0, making decisions has a crucial impact, so modern approaches are shifting towards AI models with understandable outcomes.

Recently, a new trend is gaining importance within AI, namely, eXplainable Artificial Intelligence (XAI). XAI methodologies and algorithms aim to make AI-based models and methods more transparent while maintaining high-performance levels of accuracy and precision [5]. Fuzzy Rule-Based Systems (FRBSs) are a category of models strongly oriented towards explainability. FRBSs are highly interpretable and transparent because of the linguistic definitions of fuzzy rules and fuzzy sets, which represent the knowledge base of these models. Moreover, the simplicity of the reasoning method, adopted for providing a decision based on input facts, ensures also a high explainability level of FRBSs [10].

In the last decade, Multi-Objective Evolutionary Algorithms (MOEAs) have been successfully adopted for designing FRBSs from data, leading to the so-called Multi-Objective Evolutionary Fuzzy Systems (MOEFSs) [8,9]. MOEFSs are designed to concurrently optimize the accuracy and explainability of FRBSs, which are two conflicting objectives. Indeed, in general, very accurate models are characterized by low explainability and vice-versa.

Regarding software tools to generate and evaluate XAI models, there are not many options. For example, GUAJE [13] and ExpliClas [2] are examples of tools for designing interpretable models. They also handle FRBSs, but without the boost of MOEAs for optimizing their accuracy and explainability.

In this paper, we propose and discuss SK-MOEFS, a new Python library that helps data scientists to define, build, evaluate, and use MOEFSs, under the Scikit-Learn environment [14]. The latter is an Open Source toolbox that provides state-of-the-art implementations of many well-known ML algorithms. We designed SK-MOEFS according to Scikit-Learn's design principles. Indeed, we exploited the available data structures and methods in the Scikit-Learn library. As a result, the user is allowed, under the same framework, to easily and quickly design, evaluate, and use several ML models, including MOEFSs. The current version of SK-MOEFS includes an implementation of a specific MOEFS, namely PAES-RCS, introduced in [3]. PAES-RCS selects a reduced number of rules and conditions, from an initial set of rules, during the multi-objective evolutionary learning process. Precisely, we implemented PAES-RCS-FDT, which adopts a fuzzy decision tree (FDT) for generating the initial set of rules [7]. We also highlight that SK-MOEFS is an extendable framework that allows easy integration of different types of MOEFSs.

The paper is organized as follows. Section 2 introduces FRBSs and the general multi-objective evolutionary learning scheme for designing them. Afterward, Sect. 3 illustrates the design of SK-MOEFS, focusing on the functionalities provided. Then, we describe in detail the implementation of a specific MOEFS for classification problems in Sect. 4. Section 5 is devoted to show an example of building and evaluating a MOEFS tested with a real-world dataset. Finally, Sect. 6 draws some conclusions.

2 Multi-objective Evolutionary Fuzzy Systems

2.1 Fuzzy Rule-Based Systems

A Fuzzy Rule-Based System (FRBS) is characterized by two main components, namely the Knowledge Base (KB) and the fuzzy inference engine. The KB is composed by a set of linguistic rules and by a set of parameters which describe the fuzzy sets on which the rules are defined. The fuzzy inference engine is in charge of generating a prediction, given a new input pattern, based on the content of the KB.

Let $X = \{X_1, \dots, X_F\}$ be the set of input attributes and X_{F+1} be the output attribute. Let U_f, with $f = 1, ..., F+1$, be the universe of the f^{th} attribute X_f. Let $P_f = \{A_{f,1}, \dots, A_{f,T_f}\}$ be a fuzzy partition of T_f fuzzy sets on attribute X_f. Finally, we define the training set $\{(\mathbf{x}_1, x_{F+1,1}), \dots, (\mathbf{x}_N, x_{F+1,N})\}$ as a collection of N input-output pairs, with $\mathbf{x}_t = [x_{t,1} \dots, x_{t,F}] \in \Re^F$, $t = 1, \dots, N$.

In regression problems, X_{F+1} is a continuous attribute and, therefore, $\forall t \in [0..N]$, $x_{F+1,t} \in \Re$. With the aim of estimating the output value corresponding to a given input vector, we can adopt a Fuzzy Rule-Based Regressor (FRBR) with a rule base (RB) composed of M linguistic fuzzy rules expressed as:

$$\begin{aligned} R_m : \textbf{IF } X_1 \textbf{ is } A_{1,j_{m,1}} \textbf{ AND} \dots \textbf{AND } X_f \textbf{ is } A_{f,j_{m,f}} \textbf{ AND} \\ \dots \textbf{AND } X_F \textbf{ is } A_{F,j_{m,F}} \textbf{ THEN } X_{F+1} \textbf{ is } A_{F+1,j_{m,F+1}} \end{aligned} \tag{1}$$

where $j_{m,f} \in [1, T_f]$, $f = 1, ..., F+1$, identifies the index of the fuzzy set (among the T_f linguistic terms of partition P_f), which has been selected for X_f in rule R_m.

In classification problems, X_{F+1} is categorical and $x_{F+1,t} \in C$, where $C = \{C_1, \dots, C_K\}$ is the set of K possible classes. With the aim of determining the class of a given input vector, we can adopt a Fuzzy Rule-Based Classifier (FRBC) with an RB composed of M rules expressed as:

$$\begin{aligned} R_m : \textbf{IF } X_1 \textbf{ is } A_{1,j_{m,1}} \textbf{ AND} \dots \textbf{AND } X_f \textbf{ is } A_{f,j_{m,f}} \textbf{ AND} \\ \dots \textbf{AND } X_F \textbf{ is } A_{F,j_{m,F}} \textbf{ THEN } X_{F+1} \textbf{ is } C_{j_m} \textit{ with } RW_m \end{aligned} \tag{2}$$

where C_{j_m} is the class label associated with the m^{th} rule, and RW_m is the rule weight, i.e., a certainty degree of the classification in the class C_{j_m} for a pattern belonging to the fuzzy subspace delimited by the antecedent of the rule R_m. Different definitions of the rule weight RW_m are commonly found in the literature [4]:

Given a new input pattern $\hat{\mathbf{x}} \in \Re^F$, the estimated output value or class label is provided by the FRBR or by the FRBC, respectively, adopting a specific inference engine. In both cases, the output depends on the strength of activation of each rule with the input. Details on the different inference engines can be found in [4].

In the current version of SK-MOEFS, we adopt strong triangular fuzzy partitions. As shown in Fig. 1, each partition is made up of triangular fuzzy sets $A_{f,j}$, whose membership function can be represented by the tuples $(a_{f,j},\ b_{f,j},\ c_{f,j})$, where $a_{f,j}$ and $c_{f,j}$ correspond to the left and right extremes of the support of $A_{f,j}$, and $b_{f,j}$ to its core. Other typologies of FRBSs, such as TSK-FRBSs, FRBSs with DNF rules and FRBSs based on multiple granularities, have also been considered in the MOEFS specialized literature [9]. For the sake of brevity, in this Section we have described only the two types of FRBSs which have been mostly discussed and experimented in the last years, mainly due to their high explainability level. However, the SK-MOEFS toolbox has been designed for allowing the programmer to easily implement multi-objective evolutionary learning schemes for any kind of FRBS, both for regression and classification problems.

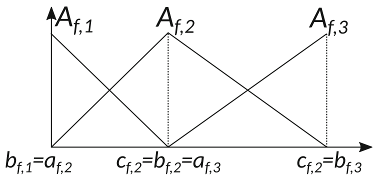


Fig. 1. An example of a strong triangular fuzzy partition with three fuzzy sets.

2.2 Multi-objective Evolutionary Learning Schemes

The FRBS design process aims: i) to determine the optimal set of rules for managing regression or classification problems, and ii) to find the appropriate number of fuzzy sets for each attribute and their parameters. The objective of the design process is to concurrently maximize the system accuracy and, possibly, the model explainability. The accuracy of an FRBR is usually maximized by means of a minimization process of the estimation error of the output values. On the other hand, the accuracy of an FRBC is usually calculated in terms of percentage of correctly classified patterns. As regards the explainability, when dealing with FRBS we usually talk about their *intepretability*, namely the capability of explaining how predictions have been done, using terms understandable to humans. Thus, the simplicity of the fuzzy inference engine, adopted to deduce conclusions from facts and rules, assumes a special importance. Moreover, the intepretability is strictly related to the *transparency* of the model, namely to the capability of understanding the structure of the model itself. FRBSs can be characterized by a high transparency level, whenever the linguistic RB is composed of a reduced number of rules and conditions and the fuzzy partitions have a good *integrity*. The integrity of fuzzy partitions depends on some properties, such as order, coverage, distinguishability and normality [4]. The work in [11] discusses several measures for evaluating the interpretability of an FRBS, taking into consideration semantic and complexity aspects of both the RB and of the fuzzy partitions.

As stated in the Introduction, in the last decade, MOEAs have been successfully adopted for designing FRBSs by concurrently optimizing both their accuracy and explainability, leading to the so-called MOEFSs [9]. Indeed, MOEAs allow us to approach an optimization process in which two or more conflicting objectives should be optimized at the same time, such as accuracy and explainability of FRBSs. MOEAs return a set of non-dominated solutions, characterized by different trade-offs between the objectives, which represents an approximation of Pareto front [6]. Adopting a Multi-Objective Learning Scheme (MOEL) it is possible to learn the structure of FRBSs using different strategies, such as learning only the RB considering pre-defined fuzzy partitions, optimizing only the fuzzy set parameters, selecting rules and conditions, from an initial set of rules, and learning/selecting rules concurrently with the optimization of the fuzzy set parameters. A complete taxonomy of MOEFSs can be found in [8].

In general, an MOEL scheme includes a chromosome coding, which is related to the type of FRBS and to the specific learning strategy, and a set of mating operators, namely mutation and crossover, appropriately defined for acting on the chromosome and generating offsprings. Obviously, an MOEL scheme must use a specific MOEA for handling the multi-objective evolutionary optimization process. During this process, a candidate solution is evaluated decoding its chromosome for building the actual FRBS. Specifically, its accuracy is calculated adopting a training set provided as an input. The explainability is evaluated on the basis of a pre-defined measure, such as the number of rules or the total number of conditions in the RB (also called Total Rule Length (TRL)). At the end of the optimization, a set of FRBSs, characterized by different trade-off between accuracy and intepretability, is returned. In the following sections, we show how to design and implement an MOEL scheme in our SK-MOEFS toolbox.

3 SK-MOEFS Design

Previously, we argued about the importance of MOEFSs in the context of XAI. In this Section, we discuss the design of SK-MOEFS, a Python library for generating explainable FRBSs. SK-MOEFS extends the functionalities of Scikit-Learn[1], a popular Open Source tool for predictive data analysis [14]. Data scientists and researchers deeply adopt Scikit-Learn due to its *ease-of-use*. Moreover, it is highly efficient both in terms of memory occupancy and computational costs. Indeed, Scikit-Learn takes advantage of other Python libraries, such as NumPy, SciPy, and MatplotLib, largely employed in the data analysis field.

Similarly to Scikit-Learn, SK-MOEFS allows us also to adopt the generated models for making predictions and evaluating the models in terms of different metrics. However, since SK-MOEFS creates a collection of different FRBSs, we had to appropriately design data structures and methods for handling more than one model. Indeed, classically, Scikit-Learn algorithms allow the user to define, train, evaluate, and use just one model. Finally, we have also designed the methods for extracting explainability metrics, such as TRL, number of rules, and partition integrity indices [11].

[1] https://scikit-learn.org/.

3.1 Class Hierarchy

To design and implement SK-MOEFS, we followed the official Scikit-Learn guidelines for developers[2].

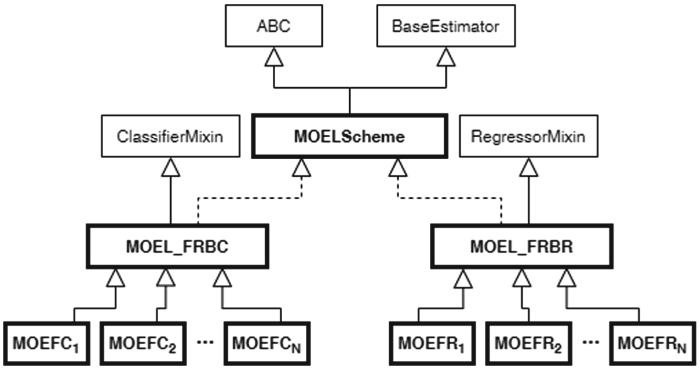


Fig. 2. UML class diagram describing the class hierarchy of SK-MOEFS.

As depicted in Fig. 2, the principal abstract class of SK-MOEFS, that we labeled as MOELScheme, derives from the BaseEstimator class of Scikit-Learn library. Moreover, to define the infrastructure of an abstract class, MOELScheme must extend the ABC class. A MOELScheme represents a general multi-objective evolutionary learning scheme for generating a set of FRBSs characterized by different trade-offs between accuracy and explainability. We recall that the chromosome coding and the mating operators depend on the selected learning scheme. As regards the fitness functions, the accuracy measure depends on the type of problems to be approached (classification or regression), and the explainability measure can be defined in several ways, as discussed in the previous section.

In general, a classifier or a regressor is an instance of a specific class derived by the BaseEstimator and by a ClassifierMixin or RegressorMixin classes: it is an object that fits a model based on some training data and is capable of making predictions on new data.

Since we aim to provide a general scheme for approaching both classification and regression problems by using MOEFSs, we derive two abstract classes from the MOELScheme one, namely MOEL_FRBC and MOEL_FRBR. They define, respectively, the MOEL scheme for Fuzzy Rule-based Classifiers (FRBCs) and the one for Fuzzy Rule-based Regressors (FRBRs). The former includes methods from the ClassifierMixin class and the latter from RegressorMixin class.

Finally, from the MOEL_FRBC and the MOEL_FRBR classes, actual MOEL schemes (we labeled them as Multi-objective Evolutionary Fuzzy Classifier (MOEFC) or Regressor (MOEFR)) can be derived, such as the PAES-RCS that has been implemented and experimented, as discussed in the following sections.

[2] https://scikit-learn.org/stable/developers.

3.2 Description of the Main Methods

Each MOEL scheme must provide the typical Scikit-Learn methods, for both classifiers and regressors adapted explicitly for handling multiple models. In Fig. 3 we show another UML class diagram that describes the main features of MOELScheme, MOEL_FRBC, and MOEL_FRBS classes.

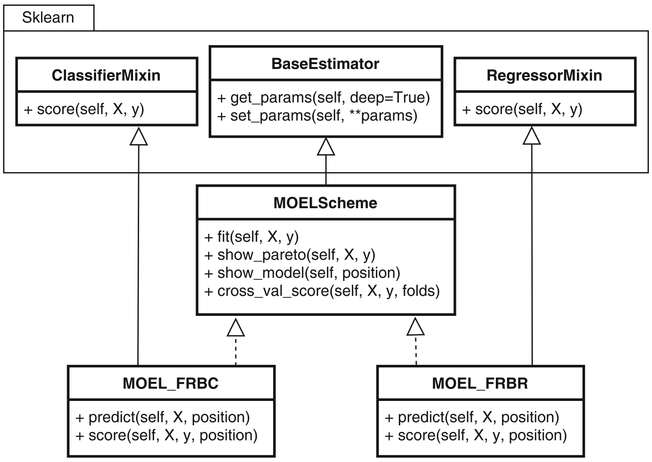

Sklearn
ClassifierMixin
+ score(self, X, y)
BaseEstimator
+ get_params(self, deep=True)
+ set_params(self, **params)
RegressorMixin
+ score(self, X, y)
MOELScheme
+ fit(self, X, y)
+ show_pareto(self, X, y)
+ show_model(self, position)
+ cross_val_score(self, X, y, folds)
MOEL_FRBC
+ predict(self, X, position)
+ score(self, X, y, position)
MOEL_FRBR
+ predict(self, X, position)
+ score(self, X, y, position)

Fig. 3. UML class diagrams describing the main methods of SK-MOEFS

In Scikit-Learn, the methods *fit*, *predict*, and *score* are typically implemented on each classifier or regressor. They allow, respectively, creating a model, using the model for making predictions, and extracting some metrics for evaluating the model. In the following, we describe these and other specific methods that must be implemented for each new MOEL scheme:

– *fit*: this method estimates the model parameters, namely the RB and the fuzzy partitions, exploiting the provided training set. We recall that in Scikit-Learn a training set must be provided in terms of an $N \times F$ NumPy matrix $\boldsymbol{X}$, describing the input patterns in terms of F features, and a vector $\boldsymbol{y}$ with N elements representing the actual label or value associated with a specific input pattern. In the beginning, the method initializes a MOEL scheme according to a specific learning strategy and to the type of problem to be handled, namely classification or regression. Then, an MOEA is in charge of carrying out the learning process, which stops when a specific condition is reached (for example, when the algorithm reaches the maximum number of fitness function evaluations). Finally, it returns an approximated Pareto front of FRBSs, which are sorted by an ascending order per accuracy. The first model, labeled as the FIRST solution, is the one characterized by the highest accuracy and by the lowest explainability. On the contrary, we marked the model with the highest explainability but the lowest accuracy as the LAST solution.

Finally, the MEDIAN model is the middle ground between the two. Indeed, its accuracy is the median among the solutions.
- *predict*: this method is in charge of predicting the class labels or the values associated with a new set of input patterns. It returns a vector of estimated labels or values. Since the MOEL scheme generates multiple models, the method takes as input also an index for selecting the model into the Pareto front. By default, the function adopts the most accurate model (FIRST) for making predictions. Notice that all the learning schemes that extend from MOEL_FRBC or MOEL_FRBR, must implement the predict method to define different and specific behaviors.
- *score*: this method takes as inputs a matrix $\boldsymbol{X}$, which contains data described in the feature space of F values, and the vector $\boldsymbol{y}$ of the r labels or values associated with each input. Moreover, it takes the position of a model belonging to the Pareto front, and it generates the values of the accuracy and explainability measures for that selected model. Also in this case, the FIRST solution is selected by default.
- *show_pareto*: this method extracts and plots the values of the accuracy and the explainability. By default, for each model of the Pareto front generated by an MOEL scheme, it runs the *fit* method on the training set. SK-MOEFS allows the user to provide also a test set; in this case, show_pareto calculates the accuracies considering the additional data. As a result, it returns a plot of the approximated Pareto front, both on the training and the test sets.
- *show_model*: given the position of a model in the Pareto front, this method shows the set of fuzzy linguistic rules and the fuzzy partitions associated with each linguistic attribute. The predefined model of choice is, as always, the FIRST solution.

Finally, since Scikit-Learn provides methods for performing a *k-fold cross-validation* analysis, we re-designed these methods for handling the fact that a MOEL scheme generates a set of solutions. Specifically, we redefined the *cross_val_score* which usually returns an array of k scores, one for each fold. Here, the method returns a $k \times 6$ matrix, where each row contains the accuracy, calculated on the test set, and the explainability of the FIRST, MEDIAN, and LAST solutions. Moreover, when performing cross-validation with MOEFSs [4], we decided to act as follows: first, we compute the mean values of the accuracy and the explainability of the FIRST, MEDIAN and LAST solutions, then we plot them on a graph.

4 An Example of an MOEL Scheme Implementation: PAES-RCS-FDT for Classification Problems

In this Section, we describe the actual implementation of an MOEL scheme for classification problems in SK-MOEFS, namely PAES-RCS-FDT [7]. The implemented algorithm adopts the rule and condition selection (RCS) learning scheme [3] for classification problems. The multi-objective evolutionary learning scheme is based on the (2 + 2)M-PAES, which is an MOEA successfully employed in the context of MOEFSs during the last years. The algorithm concurrently optimizes two objectives: the first objective considers the TRL as explainability measure; the second objective takes into account the accuracy, assessed in terms of classification rate.

In the learning scheme, an initial set of candidate rules must be generated through a heuristic or provided by an expert. In our implementation, the set of candidate rules is generated exploiting the fuzzy multi-way decision trees (FMDT) [15]: each path from the root to a leaf node translates into a rule. Before learning the FMDT, we need to define an initial strong fuzzy partition for each attribute. The adopted FMDT algorithm embeds a discretization algorithm that is in charge of generating such partitions.

During the evolutionary process, the most relevant rules and their conditions are selected. Moreover, each triangular strong fuzzy partition P_f is concurrently tuned, by adapting the positions of the cores $b_{f,j}$.

In PAES-RCS, a chromosome C codifies a solution for the problem. The former is composed of two parts (C_R, C_T), which define, respectively, the RB and the positions of the representatives of the fuzzy sets, namely the cores.

Let J_{DT} be the initial set of M_{DT} rules generated from the decision tree. Compact and interpretable RBs are desirable, so we allow that the RB of a solution contains at most M_{max} rules. The C_R part, which codifies the RB, is a vector of M_{max} pairs $\mathbf{p}_m = (k_m, \mathbf{v}_m)$, where $k_m \in [0, M_{DT}]$ identifies the selected rule of J_{DT} and $\mathbf{v}_m = [v_{m,1}, \ldots, v_{m,F}]$ is a binary vector which indicates, for each attribute X_f, if the condition is present or not in the selected rule. In particular, if $k_m = 0$ the m^{th} rule is not included in the RB. Thus, we can generate RBs with a lower number of rules than M_{max}. Further if $v_{m,f} = 0$ the f^{th} condition of the m^{th} rule can be replaced by a *"don't care"* condition.

C_T is a vector containing F vectors of $T_f - 2$ real numbers: the f^{th} vector $\left[b_{f,2}, \ldots, b_{f,T_f-1}\right]$ determines the positions of the cores of the partition P_f. We recall that using strong fuzzy partitions ensures the partition integrity. Indeed, order, coverage, distinguishability and normality are always ensured. In order to increase the integrity level, we can define constrains on the intervals on which cores can assume valid values. For more details check [3].

In order to generate the offspring populations, we exploit both crossover and mutation. We apply separately the one-point crossover to C_R and the BLX-α-crossover, with $\alpha = 0.5$, to C_T. As regards the mutation, we apply two distinct operators for C_R and an operator for C_T. More details regarding the mating operators and the steps of PAES-RCS can be found in [3]. In the next Section, we will briefly introduce the main parameters that must be set for running PAES-RCS-FDT.

In Fig. 4, we show a detailed UML class diagram describing the main classes and methods that we implemented. First of all, we have derived from the MOEL_FRBC the class MPAES_RCS, which is in charge of handling the rule and condition selection multi-objective learning scheme, by means of (2 + 2)M-PAES algorithm. This class needs the RCSProblem, which is a class derived

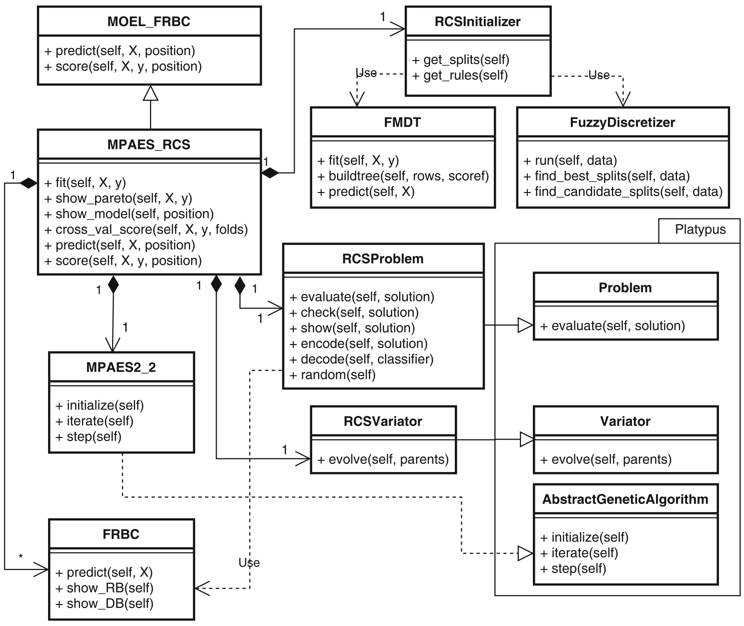


Fig. 4. UML class diagram of PAES-RCS-FDT in SK-MOEFS.

from the Problem class of the Package *Platypus*, a Python framework for multi-objective evolutionary optimization. It defines operations on a possible solution (a chromosome) such as its encoding, feasibility checks, evaluation of objectives, and generation of a random solution. Moreover, the MPAES_RCS class adopts an RCS_Variator, a particular implementation of the Platypus Variator, which includes all the mating operators that we discussed before. Two additional classes are adopted as compositions by the MPAES_RCS class, namely the MPAES2_2 and the RCSInitializer. Specifically, the MPAES2_2 class extends the Abstract-GeneticAlgorithm class of Platypus and implements (2 + 2)M-PAES. Finally, the RCSInitializer implements the methods for the definition of the initial strong fuzzy partitions and for generating the initial set of rules. To this aim, this class uses the fuzzy discretizer (implemented by the FuzzyDiscretizer class) and the Multi-way Fuzzy Decision Tree (implemented by the MFDT class), respectively. Both the discretizer and the algorithm for generating the fuzzy decision tree are described in detail in [15]. More information on the organization of the Platypus framework can be found in the official guide[3] and github repository[4]. The code of SK-MOEFS, including the implementation of PAES-RCS-FDT as a standard Python program, is available on a GitHub repository[5], along with a detailed documentation describing all the classes and its methods.

3 https://platypus.readthedocs.io/en/latest/.
4 https://github.com/Project-Platypus/Platypus.
5 https://github.com/GionatanG/skmoefs.

5 Examples of Usage of SK-MOEFS

In this Section, we show some examples of usage of SK-MOEFS. Specifically, we show how to generate a set of FRBCs, characterized by different trade-offs between accuracy and explainability, using our PAES-RCS-FDT implementation. To this aim, we have selected the NewThyroid dataset[6]: the classification task associated with this dataset is to detect if a given patient is normal (Class 1), suffers from hyperthyroidism (Class 2) or hypothyroidism (Class 3). We recall that the objective of this work is not to assess the goodness of the PAES-RCS-FDT. Indeed, as stated before, PAES-RCS was introduced in 2014 in [3], where a wide experimental analysis was conducted, adopting its original implementation in C++ (the initial set of rules was generated using the C4.5 algorithm). Moreover, additional experimentation, carried out utilizing the FDT for generating the initial set of rules, has also been discussed in [7]. However, we have verified that the results obtained using PAES-RCS-FDT implemented in SK-MOEFS are in line with those discussed in [3] and in [7].

Table 1 shows the parameters of (2 + 2)M-PAES-FDT used in the examples. These values have been set also as default parameters of our PAES-RCS-FDT implementation.

Table 1. Values of the parameters used in the examples

N_{val}	Total number of fitness evaluations	30000
AS	(2 + 2)M-PAES archive size	32
M_{max}	Maximum number of rules for each RB	50
P_{C_R}	Probability of applying crossover operator to C_R	0.1
P_{C_T}	Probability of applying crossover operator to C_T	0.5
P_{MRB_1}	Probability of applying first mutation operator to C_R	0.1
P_{MRB_2}	Probability of applying second mutation operator to C_R	0.7
P_{M_T}	Probability of applying mutation operator to C_T	0.2
T_{max}	Maximum number of fuzzy sets for each attribute	5

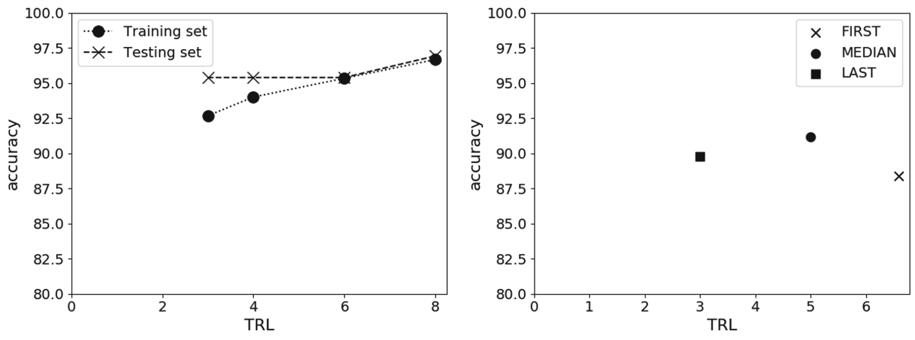
```
from skmoefs.rcs import MPAES_RCS, RCSInitializer, RCSVariator
from sklearn.model_selection import train_test_split

X, y = load_dataset('newthyroid')
Xtr, ytr, Xte, yte = train_test_split(X, y, test_size=0.3)
my_moefs = MPAES_RCS(variator=RCSVariator(), initializer=
    RCSInitializer())
my_moefs.fit(Xtr, ytr, max_evals=30000)
my_moefs.show_pareto(Xte, yte)
my_moefs.show_model('median')
```

Listing 1.1: Example for generating and plotting a Pareto front approximation of FRBCs

[6] https://sci2s.ugr.es/keel/dataset.php?cod=66.

In code Listing 1.1, we show an example of usage, in which we first load a dataset from a file and then we divide it into training and test sets. Second, we instantiate an MPAES-RCS object passing to its constructor the RCSVariator and an RCSInitializer. The latter will partition each input attributes into a pre-defined number of fuzzy sets and will generate the matrix J_{DT}. Afterward, we call the fit method, which returns the fitted model having now a list of the FRBCs characterized by different trade-offs between accuracy and explainability. Then, we call the method for showing the Pareto front approximation (in Fig. 5(a)), both on the training and test sets. Finally, we show the RB and the fuzzy partitions (in Fig. 6) of the MEDIAN solution in the Pareto Front approximation. In this example, we labeled the five fuzzy sets of each partition as Very Low (VL), Low (L), Medium (M), High (H), and Very High (VH). As we can see, the set of linguistic rules allows the user to understand the motivation of a decision: simply speaking, based on the levels of each input attribute describing a new patient, a specific class is associated with him/her. As regards the fuzzy partitions, it is worth noting, especially for the last two, that they moved from the initial uniform shape. However, they are still strong fuzzy partitions, thus ensuring a good integrity level, in terms of order, distinguishability, coverage and normality.



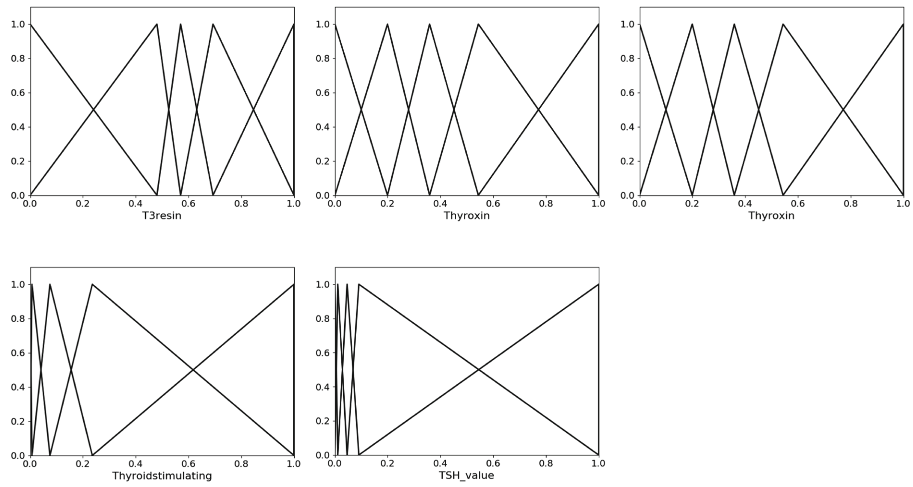
Pareto front approximation Cross-validation scores

Fig. 5. Two examples of plots

```
from skmoefs.rcs import MPAES_RCS, RCSInitializer, RCSVariator

X, y = load_dataset('newthyroid')
my_moefs = MPAES_RCS(variator=RCSVariator(), initializer=
    RCSInitializer())
my_moefs.cross_val_score(X, y, folds=5)
```

Listing 1.2: Example for performing the cross validation



```
RULE BASE
1:  IF Thyroxin is VL THEN Class is 3
2:  IF Thyroxin is M  AND Thyroidstimulating is L THEN Class is 1
3:  IF Thyroxin is VH THEN Class is 2
```

Fig. 6. Fuzzy partitions and rule base of the MEDIAN solution

Finally, in the code Listing 1.2, we also show how to perform a 5-fold cross-validation. As a result, we draw a graph with the average values of accuracy and explainability of the FIRST, MEDIAN, and LAST solutions on the test set, as shown in Fig. 5(b).

6 Conclusions

In this paper, we have introduced a new Python library for generating, evaluating and using both accurate, and explainable AI-based models, namely fuzzy rule-based systems (FRBSs). The library, called SK-MOEFS, allows the users to adopt multi-objective evolutionary learning (MOEL) schemes for identifying, from data, the structure of a set of FRBSs, characterized by different trade-offs between accuracy and explainability. Specifically, we designed the overall software infrastructure, i.e. all the class hierarchy, for handling a generic multi-objective evolutionary learning scheme. Moreover, we show an example of an actual implementation of a well known MOEL scheme, namely PAES-RCS-FDT. This scheme, during the evolutionary process, selects rules and conditions from an initial set of candidate classification rules, generated using a fuzzy decision tree. Additionally, the parameters of the fuzzy partitions can be learned concurrently with the set of rules. Finally, we have shown a simple example on how our SK-MOEFS can be used in Python for generating and evaluating a set of fuzzy classifiers on a benchmark dataset.

References

1. Adadi, A., Berrada, M.: Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE Access **6**, 52138–52160 (2018)
2. Alonso, J.M., Bugarín, A.: ExpliClas: automatic generation of explanations in natural language for weka classifiers. In: 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–6. IEEE (2019)
3. Antonelli, M., Ducange, P., Marcelloni, F.: A fast and efficient multi-objective evolutionary learning scheme for fuzzy rule-based classifiers. Inf. Sci. **283**, 36–54 (2014)
4. Antonelli, M., Ducange, P., Marcelloni, F.: Multi-objective evolutionary design of fuzzy rule-based systems. In: Handbook on Computational Intelligence: Volume 2: Evolutionary Computation, Hybrid Systems, and Applications, pp. 635–670. World Scientific (2016)
5. Carletti, M., Masiero, C., Beghi, A., Susto, G.A.: Explainable machine learning in industry 4.0: evaluating feature importance in anomaly detection to enable root cause analysis. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 21–26. IEEE (2019)
6. Coello, C.A.C., Lamont, G.B.: Applications of Multi-objective Evolutionary Algorithms, vol. 1. World Scientific, Singapore (2004)
7. Ducange, P., Mannarà, G., Marcelloni, F.: Multi-objective evolutionary granular rule-based classifiers: an experimental comparison. In: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–6. IEEE (2017)
8. Ducange, P., Marcelloni, F.: Multi-objective evolutionary fuzzy systems. In: Fanelli, A.M., Pedrycz, W., Petrosino, A. (eds.) WILF 2011. LNCS (LNAI), vol. 6857, pp. 83–90. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23713-3_11
9. Fazzolari, M., Alcala, R., Nojima, Y., Ishibuchi, H., Herrera, F.: A review of the application of multiobjective evolutionary fuzzy systems: current status and further directions. IEEE Trans. Fuzzy Syst. **21**(1), 45–65 (2012)
10. Fernandez, A., Herrera, F., Cordon, O., del Jesus, M.J., Marcelloni, F.: Evolutionary fuzzy systems for explainable artificial intelligence: why, when, what for, and where to? IEEE Comput. Intell. Mag. **14**(1), 69–81 (2019)
11. Gacto, M.J., Alcalá, R., Herrera, F.: Interpretability of linguistic fuzzy rule-based systems: an overview of interpretability measures. Inf. Sci. **181**(20), 4340–4360 (2011)
12. Lu, Y.: Industry 4.0: a survey on technologies, applications and open research issues. J. Ind. Inf. Integr. **6**, 1–10 (2017)
13. Pancho, D.P., Alonso, J.M., Magdalena, L.: Quest for interpretability-accuracy trade-off supported by fingrams into the fuzzy modeling tool GUAJE. Int. J. Comput. Intell. Syst. **6**(sup1), 46–60 (2013)
14. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
15. Segatori, A., Marcelloni, F., Pedrycz, W.: On distributed fuzzy decision trees for big data. IEEE Trans. Fuzzy Syst. **26**(1), 174–192 (2018)