pyFUME: a Python Package for Fuzzy Model Estimation

Caro Fuchs[*], Simone Spolaor[†‡], Marco S. Nobile[*‡], Uzay Kaymak[*]
[*]School of Industrial Engineering
Eindhoven University of Technology, Eindhoven, The Netherlands
Email: {c.e.m.fuchs, m.s.nobile, u.kaymak}@tue.nl
[†]Department of Informatics, Systems and Communication
University of Milano-Bicocca, Milan, Italy
Email: simone.spolaor@disco.unimib.it
[‡]SYSBIO.IT Centre for Systems Biology, Milan, Italy

***Abstract*—Living in the era of "data deluge" demands for an increase in the application and development of machine learning methods, both in basic and applied research. Among these methods, in the last decades fuzzy inference systems carved out their own niche as (light) grey box models, which are considered more interpretable and transparent than other commonly employed methods, such as artificial neural networks. Although commercially distributed alternatives are available, software able to assist practitioners and researchers in each step of the estimation of a fuzzy model from data are still limited in scope and applicability. This is especially true when looking at software developed in Python, a programming language that quickly gained popularity among data scientists and it is often considered their language of choice. To fill this gap, we introduce pyFUME, a Python library for automatically estimating fuzzy models from data. pyFUME contains a set of classes and methods to estimate the antecedent sets and the consequent parameters of a Takagi-Sugeno fuzzy model from data, and then create an executable fuzzy model exploiting the Simpful library. pyFUME can be beneficial to practitioners, thanks to its pre-implemented and user-friendly pipelines, but also to researchers that want to fine-tune each step of the estimation process.**

***Index Terms*—fuzzy logic, Takagi-Sugeno fuzzy model, data-driven, open-source software, Python**

I. Introduction

With the rapidly growing amount of gathered data [1] and the increasing computer performance [2], Machine Learning (ML) has gained in popularity. In the past decade, ML led to many advancement, including self-driving cars [3], effective image analysis [4] (in particular in the clinical domain [5]), high quality speech recognition [6], an increased understanding of the human genome [7], and so forth.

In 2018, Kaggle, an online community of data scientists and ML practitioners, conducted an industry wide survey to find out who is working with data and what is happening with ML in different industries [8]. One of the findings of this survey is that the most used programming language for data professionals is Python, which was used on a regular basis by 83% of the nearly 24,000 respondents. The survey also showed that 78% of the respondents recommends aspiring data scientist to learn how to program in Python as a first language.

Fuzzy inference systems (FIS) [9] are a popular type of models whose parameters can be optimized using machine learning optimization methods. FIS are an interpretable and transparent type of models that reveal the underlying relations in the data in terms of fuzzy rules. Since these fuzzy rules are comprehensible to human beings, FIS function as (light) grey box models, in contrast to, for example, (deep) artificial neural networks. This property makes it possible to study these models' underlying reasoning mechanism and to gain a deeper understanding of patterns in the data. Hence, FIS are considered transparent and interpretable to a certain degree.

One of the most well-known and used software tools to create data-driven fuzzy models is the MATLAB Fuzzy Logic Toolbox [10], which provides MATLAB functions and a GUI for analyzing, designing, and simulating systems based on fuzzy logic, which can be both created from knowledge or data. However, being a closed source, commercial product, MATLAB is often unavailable for practitioners. This could explain why only 14% of the respondents of the Kaggle survey indicated to use MATLAB on a regular basis.

Given the popularity of Python it is remarkable that, to the best of our knowledge, Python libraries providing methods to automatically estimate fuzzy models from data are scarce and none of them gives the user complete control over the estimation process. Existing examples include libraries that provide support for specific data-driven applications, i.e., fuzzy clustering and creation of Adaptive Network Fuzzy Inference Systems [11], [12]. In this paper we present pyFUME, a Python package that provides a set of classes to define pipelines for the automatic estimation of fuzzy models from data. In pyFUME the fuzzy reasoning is handled by Simpful [13], a Python library that provides a set of classes and methods to intuitively define and handle fuzzy sets, fuzzy rules and perform fuzzy inference. pyFUME contains functions to estimate the antecedent sets and the consequent parameters of a Takagi-Sugeno fuzzy model directly from data. This information is then used to create an executable fuzzy model using the Simpful library. pyFUME also provides facilities for the evaluation of performance from a statistical standpoint.

This paper is structured as follows. In Section II, we first provide some background on the structure and generation of Takagi-Sugeno fuzzy models. After this, we give some information about the implementation details of the Simpful library, which is the fuzzy reasoner used by pyFUME. Then, in Section III, we describe in detail the functions implemented in pyFUME. In Section IV we show how pyFUME can be used to generate a fuzzy model from data using a publicly available data set. Lastly, in Section V we draw final remarks and provide insights on future developments of pyFUME.

II. Background

A. The Takagi-Sugeno fuzzy model

FIS are "expert systems" [14], in which knowledge is formalized and codified by means of membership functions and fuzzy rules. The process of fuzzy inference leverages fuzzy logic for mapping a given input to an output. This mapping can serve as a basis from which decisions or predictions can be made, or it can reveal patterns in the data.

In this context, a Takagi-Sugeno (TS) FIS [15] consists of a set of fuzzy rules, where each rule describes a local relation between input and output variables. When first-order TS fuzzy systems are used, each model consists of rules in the form:

$$\begin{aligned} \mathbf{R}_j : \textbf{IF } & x_1 \text{ is } A_{j1} \text{ and } \dots \text{ and } x_N \text{ is } A_{jN} \\ & \textbf{THEN } y_j = \mathbf{a}_j^T \mathbf{x} + b_j \end{aligned} \tag{1}$$

where, $j = 1, \dots J$ denotes the rule number; $\mathbf{x} = (x_1, \dots x_N)$ denotes the input vector; N is the number of input features; A_{jn} is the fuzzy set for rule R_j and n^{th} feature, while y_j is the consequent function of rule R_j, that is, a linear combination of the elements of $\mathbf{x}$, with coefficients $\mathbf{a}_j$ and a constant b_j.

We can define the degree of fulfillment of a rule j as:

$$\beta_j = \min(\mu_{A_{j1}}(\mathbf{x}), \dots, \mu_{A_{jN}}(\mathbf{x})), \text{ for } j = 1, \dots, N. \tag{2}$$

The overall output y^* of the system for a given input vector $\mathbf{x}$ is a weighted average of the individual rule outputs:

$$y^* = \frac{\sum_{j=1}^{J} \beta_j y_j}{\sum_{j=1}^{J} \beta_j}. \tag{3}$$

The development of a Takagi-Sugeno FIS generally consists of two phases [16], [17]:

- during *structure identification* phase, a suitable number of rules and a proper partition of the feature space must be determined. This can be done, e.g., by means of grid partitioning or k-means clustering [18], fuzzy c-means [19] or subtractive [20] clustering;
- in a second phase, corresponding to the *parameter identification*, the FIS parameters (e.g., the membership functions or the linear coefficients) are adjusted. For this purpose, least-square or derivative-based optimization techniques can be used (for more information see [16]).

B. Simpful

Simpful is a user-friendly Python library, which provides the users with a lightweight application programming interface (API) to define FIS for any purpose [13]. Compared to other existing Python software (e.g, [11]), Simpful's API was designed to be as close as possible to natural language, in order to facilitate the users in the definition of fuzzy sets, linguistic variables, fuzzy rules and perform fuzzy inference. Notably, this was achieved without dependencies from any Python libraries besides numpy and scipy, and the use of virtual machines (a solution adopted, for example, in [21]).

One of the distinguishing features of Simpful is that fuzzy rules are defined through well-formed strings of text, written in natural language, thus simplifying the definition of the rule base. Such encoding of the rules in a human readable format facilitates the inspection of the model and the interpretation of the results, with respect to other competitor libraries.

Once a collection of linguistic variables, fuzzy rules, and consequent outputs is defined and added to Simpful's fuzzy system main class, this performs a recursive tokening and parsing of the antecedents of each rule (exploiting the parentheses as delimiters), in order to identify atomic clauses and functional logic operators. This allows Simpful to build executable representations of the antecedents of the rules in the form of derivation trees, which are exploited to perform Sugeno inference and return final output values, once the input values of the antecedents are provided.

The latest version of Simpful (v2.0.0) supports: *(i)* polygonal and functional (*e.g.*, sigmoidal, gaussian or custom shaped) fuzzy sets; *(ii)* the definition of fuzzy rules with an arbitrary degree of complexity, employing common logic operators (e.g., AND, OR, NOT), by means of strings of text written in natural language; *(iii)* the Sugeno inference method, with support for consequent functions of any order. Moreover, Simpful supports the definition of FIS with multiple inputs and outputs, and the definition of fuzzy networks [22] with arbitrary topologies. The source code of Simpful is available, under GPL license, on GitHub at the following URL: https://github.com/aresio/simpful. Simpful can also be installed by using the PyPI facility: `pip install simpful`.

III. Implementation of pyFUME

pyFUME was implemented in the Python 3 programming language [23] and depends on numpy [24], scipy [25] and Simpful [13]. pyFUME can be downloaded from GITHUB at the following address: https://github.com/CaroFuchs/pyFUME. A PyPI package for pyFUME is also under development. Currently, pyFUME supports:

1) Loading of the input data.
2) Division of the data in a training and test data set.
3) Clustering of the data using Fuzzy C-Means (FCM) [19] or an approach based on Fuzzy Self-Tuning Particle Swarm Optimization (FST-PSO [26], [27]) in the input-output space.
4) Estimation of the antecedent set parameters of the fuzzy model, using the method described in [28].
5) Estimation of the consequent parameters of the first order Takagi-Sugeno fuzzy model, implementing the functionalities described in [29].
6) The generation, using the estimated antecedents and consequents, of an executable fuzzy model based on

Simpful, possibly exporting the source code as a separate, executable file.
7) Testing of the estimated fuzzy model, by measuring the Root Mean Squared Error (RMSE), Mean Squared Error (MSE) or Mean Absolute Error (MAE).
8) Creation of visual summaries of the fuzzy sets derived by pyFUME.

The primary goal of pyFUME is to facilitate the practitioners in the creation of a first order Takagi-Sugeno fuzzy model from data. Therefore, a fuzzy model using default settings can be created using the `pyFUME()` class, for which the user only has to specify the path to the data he would like to use, the number of clusters/rules that should be identified in the data and the type of fuzzy model the user wishes to be created (default: `model_type='Takagi-Sugeno'`). `pyFUME()` then functions as a wrapper and calls pyFUME's classes as shown in Figure 1 for fitting the Takagi-Sugeno fuzzy model. If the user wishes to change the default settings, he can do so by providing key–value pair as input arguments for `pyFUME()`. For example, providing the key value pair `normalize=True`, will result in normalization of the input data. As output, the function provides the user with the Simpful fuzzy model, executable Simpful code and the RMSE of the created model.

Users that wish to be more in control of the modeling process can produce their own pipeline, exploiting the methods exposed by the underlying classes. These classes and their methods are described in detail in what follows.

A. *DataLoader*

The `DataLoader` class takes the path to the data as input. The `DataLoader` accepts data in a comma separated format (*.csv*). Optionally, the user can specify the names of the N variables as a list, using the `variable_names` argument. If the user does not provide variable names, these names are read from the first row of the data set.

In the `DataLoader` class, the data is divided in input data `dataX` (the first N columns) and the labels `dataY` (the $N+1$-th column). If the user prefers to use normalized data, he can set the input argument `normalize` to True, after which the data will be normalized using min-max normalization. By default, `normalize` is set to False and no normalization is performed.

B. *DataSplitter*

The `DataSplitter` class takes as inputs `dataX` and the labels in `dataY`. Its method `holdout()` splits the data into a training and a test data set. By default, the training data set contains 75% of all samples. This proportion can be changed by specifying the optional input argument `percentage_training`. As output the `holdout()` method provides the user with four subsets of the data: `x_train`, `y_train`, `x_test` and `y_test`.

C. *Clusterer*

The `Clusterer` class clusters the (training) data in the input-output feature space. The current version of pyFUME supports FCM [19] and a fuzzy clustering approach based on FST-PSO, presented in [27]. The latter method can have better performance, at the cost of an increased computational effort, when dealing with clustering problems that display multiple, locally optimal solutions. The implementation of additional clustering methods is planned for future releases.

`Clusterer` object is initialized by passing as arguments the data and the number of clusters. This class is equipped with a `cluster()` method, which accepts as arguments the clustering method to be employed and its additional settings (if needed). This method effectively behaves like a wrapper for the pre-implemented clustering algorithm available in pyFUME, namely FCM and the one based on FST-PSO, returning the cluster centers and the associated partition matrix. Experienced users who might want to employ their own clustering algorithm can easily override this method with their own one. The private methods implementing the FCM and FST-PSO based algorithm accept the following additional arguments: in `_fcm()`, the user can specify the values of the fuzzy clustering coefficient `m` (default `m` = 2), maximum number of iterations `max_iter` (default `max_iter` = 1000), and the minimum improvement of the objective function to be reached as stopping criterion `error` (default `error` = 0.005); in `_fstpso()`, the user can specify the values of the fuzzy clustering coefficient `m` (default `m` = 2), maximum number of iterations `max_iter` (default `max_iter` = 100), number of particles `n_particles` (default automatically calculates the number of particles, according to a heuristic as explained in [26]).

D. *AntecedentEstimator*

The `AntecedentEstimator` estimates the parameters of the membership functions that are used as antecedents of the fuzzy rules. To do this, the `AntecedentEstimator` class uses the same algorithm described in [28]. As input, the class' `determineMF()` method requires the `partition_matrix` and `x_train`. Optionally, the user can choose the shape of the fitted fuzzy sets. pyFUME supports Gaussian (`shape=gauss`), double Gaussian (`shape=gauss2`) and sigmoid (`shape=sigmf`) membership functions. Unless specified otherwise, pyFUME fits Gaussian membership functions to the data. The `determineMF()` method returns the `antecedent_parameters`. In the rules, the identified antecedent sets will be connected with the AND operator, which is specified as the minimum in Simpful.

E. *ConsequentEstimator*

The `ConsequentEstimator` requires the `partition_matrix` and `x_train` to estimate the parameters of the linear consequent function of the fuzzy model using least squares fitting. This is done by means of the `suglms()` method, which returns the `consequent_parameters` Additionally, the user can opt for global (default, `global_fit=True`) or local fitting (`global_fit=False`) of the linear function.

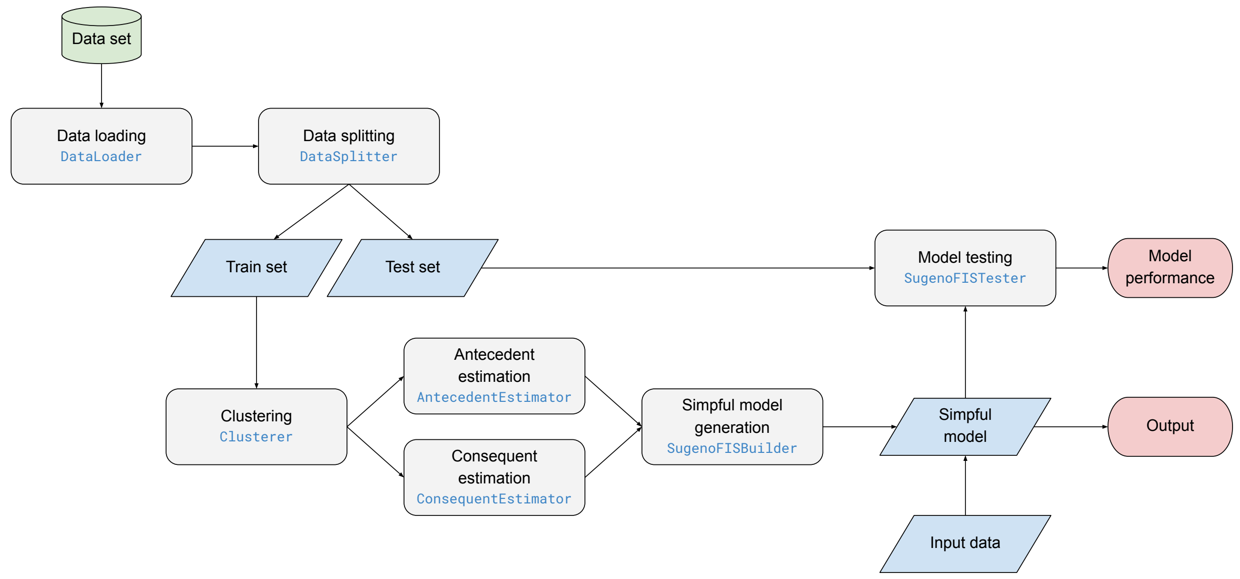

Fig. 1: Graphical representation of the pyFUME pipeline for the estimation of a Takagi-Sugeno fuzzy model. Blue text highlights the pyFUME classes employed in each step.

F. `SugenoFISBuilder`

The `SugenoFISBuilder` class takes as input arguments the `antecedent_parameters`, the `consequent_parameters` and the `variable_names` and uses these to build the Simpful model. The Simpful model is a first order Takagi-Sugeno fuzzy model in which the antecedents are connected using the AND-operator and in which the consequent exists out of a linear combination of the input variables and a constant. The `SugenoFISBuilder` class generates an object with the executable Simpful model. In addition, it can generate and save the Simpful code belonging to that model as a standalone file. The fuzzy sets of the generated model can be exported as figures (in any format supported by the Matplotlib library) using the `produce_figure()` method.

G. `SugenoFISTester`

The `SugenoFISTester` class requires as input the Simpful model and the data subsets `x_test` and `y_test`. When the `calculate_RMSE()`, `calculate_MSE()` or `calculate_MAE()` method is called, the samples provided in `x_test` are evaluated using the Simpful model and the accuracy of the model is assessed by calculating the RMSE, MSE or MAE respectively. The error value is then returned to the user.

IV. Test Case

In this section, we illustrate how pyFUME can be used to create a fuzzy model from data and we show how the generated model can be used and inspected by the user.

A. Data set

As an example, we will generate a model using the concrete data set [30]. The concrete data set is a publicly available data set, containing 1030 samples. It is gathered to analyze the compressive strength (in MPa) of concrete samples. Compressive strength is well-known to be a highly nonlinear function of the age of the concrete and the ingredients employed in its mixing. The concrete data set contains both the age and the amounts of the ingredients used in the concrete mix: cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. In what follows, we use this data set to estimate a FIS that describes the relationship between the concrete's compressive strength, its age and its exact composition.

A subset of the concrete data set is shown in Table I. As it can be observed, the data is in a comma separated format (i.e., *.csv*): the first row contains the variable names, each successive row contains the data of one data instance.

B. Fuzzy model generation and results

Both the code in Listing 1 and Listing 2 can be used to estimate a fuzzy model from the concrete data using pyFUME.

The code in Listing 1 is simple and easy to use, making it ideal to use for practitioners who wish to use the default settings or only wish to use few non-default settings. In this, it is similar to MATLAB's `genfis`. Users that wish to deviate from the default settings can use the code shown in Listing 2. Options for deviations are discussed in Section III. An example of a deviation can be seen in Listing 2, line 14, where normalization of the input data is activated.

Both the code shown in Listing 1 and Listing 2 generate Simpful code and an executable model. Line 17 to 28 of Listing 1 show how the user can use the `set_variable()` and `Sugeno_inference()` methods of the model object to predict the compressive strength of a new concrete sample.

Besides the executable model, pyFUME gives Simpful code as its output. An example of automatically generated Simpful code for the concrete data set can be found in Listing 3. This

TABLE I: Sample of the concrete data set.

Cement,	**BlastFurnaceSlag**,	**FlyAsh**,	**Water**,	**Superplasticizer**,	**CoarseAggregate**,	**FineAggregate**,	**Age**,	**CompressiveStrength**,
540.0,	0.0,	0.0,	162.0,	2.5,	1040.0,	676.0,	28,	79.99,
540.0,	0.0,	0.0,	162.0,	2.5,	1055.0,	676.0,	28,	61.89,
...								
159.1,	186.7,	0.0,	175.6,	11.3,	989.6,	788.9,	28,	32.77,
260.9,	100.5,	78.3,	200.6,	8.6,	864.5,	761.5,	28,	32.40,

Simpful code is not only an easy way to store the model for later usage. It also gives insights in the inner workings of the model since the code is readable.

In line 4 of Listing 3, the empty fuzzy system is created. Line 5 to 8 shows how the rules are defined. As can be seen in these rules, when rules are automatically identified from data only the AND-operator is used to connect the rules' antecedent sets. In line 10 to 12 the consequents with their coefficients are shown. Since we fit a first-order Takagi-Sugeno system, the consequent are a linear combinations of the input variables, plus a constant. From line 14 to 60, the fuzzy sets and membership functions of the input variables are defined. These membership functions are visualized in Figure 2. In line 10 and 11 of Listing 1 and line 40 and 41 of Listing 2, the accuracy of the model is calculated. Since the `calculate_error()` method is not given an additional argument, by default the RMSE is used. The RMSE is then printed in the console. In this example, the RMSE of this system is 14.65.

This example shows how pyFUME can be used to generate first-order Takagi-Sugeno models from data, both by using the `pyFUME` class and by using its underlying functions.

V. Conclusion

In this paper we presented pyFUME, a python package for estimating fuzzy models from data. For practitioners, pyFUME offers a user-friendly API, which is able to create a model out of data with limited interaction required by the user, owing to its pre-implemented, commonly employed methods. For more experienced users and researchers, the classes defined in pyFUME expose methods that can be easily overridden, in order to tweak the pre-implemented methods or define custom ones. This feature allows for the generation of a personalized pipeline for estimating the parameters of the fuzzy model.

In the near future, we plan to further extend pyFUME's support, by adding additional methods to its set of already implemented classes. In particular, we will add other data splitting methods (e.g., cross-validation), additional clustering methods (e.g., Gustafson-Kessel [31] and possibilistic fuzzy c-means [32] clustering), and more evaluation metrics to test the performance of the generated model. We will also add support for other types of fuzzy models, such as zero-order Takagi-Sugeno, Mamdani [33], Tsukamoto [34], AnYa [35], and probabilistic fuzzy models [36]–[38]. We will also implement support for other commonly used t-norms.

The approach and the software presented in this paper will be adapted to develop a framework for the automatic definition of dynamic fuzzy models [39], [40]. These models are represented by fuzzy inference networks [22], whose evolution over time can be simulated. Since their definition is time consuming and relies on domain-experts' knowledge, we envision that pyFUME can help the modelers in the definition of accurate models more efficiently, based on data.

Finally, future releases of Simpful [13], the fuzzy reasoner behind pyFUME, will support the Fuzzy Markup Language (FML) format, defined in the IEEE Std 1855-2016 [41]. The use of the FML, possibly achieved by leveraging existing libraries (i.e., JFML and Py4JFML [21], [42]), will facilitate the import, export and sharing of the FIS defined with pyFUME.

References

[1] C. V. N. Index, "The zettabyte era–trends and analysis," *Cisco white paper*, 2013.

[2] P. J. Denning and T. G. Lewis, "Exponential laws of computing growth," *Communications of the ACM*, vol. 60, no. 1, pp. 54–65, 2016.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] L. Rundo, C. Han, Y. Nagano, J. Zhang, R. Hataya, C. Militello, A. Tangherloni, M. S. Nobile, C. Ferretti, D. Besozzi *et al.*, "USE-Net: Incorporating Squeeze-and-Excitation blocks into U-Net for prostate zonal segmentation of multi-institutional MRI datasets," *Neurocomputing*, vol. 365, pp. 31–43, 2019.

[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[7] M. Kircher, U. Stenzel, and J. Kelso, "Improved base calling for the illumina genome analyzer using machine learning strategies," *Genome biology*, vol. 10, no. 8, p. R83, 2009.

[8] Kaggle, "2018 Kaggle ML & DS Survey," 2018. [Online]. Available: https://www.kaggle.com/kaggle/kaggle-survey-2018

[9] V. Cherkassky, "Fuzzy inference systems: a critical review," in *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*. Springer, 1998, pp. 177–197.

[10] MathWorks, "Fuzzy logic toolbox - r2019b," 2019. [Online]. Available: https://www.mathworks.com/products/fuzzy-logic.html

[11] "SciKit-Fuzzy," 2019. [Online]. Available: https://pythonhosted.org/scikit-fuzzy/

[12] "ANFIS in pyTorch," 2019. [Online]. Available: https://github.com/jfpower/anfis-pytorch

[13] S. Spolaor, C. Fuchs, P. Cazzaniga, U. Kaymak, D. Besozzi, and M. S. Nobile, "Simpful: Fuzzy logic made simple," *[under review]*, 2020.

[14] P. Jackson, *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[15] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 1, pp. 116–132, 1985.

[16] J. S. R. Jang, C. T. Sun, and E. Mizutani, *Neuro–Fuzzy and Soft Computing*. Upper Saddle River: Prentice–Hall, 1997.

[17] J. M. da Costa Sousa and U. Kaymak, *Fuzzy Decision Making in Modeling and Control*, ser. World Scientific Series in Robotics and Intelligent Systems. New Jersey: World Scientific, 2002, vol. 27.

[18] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

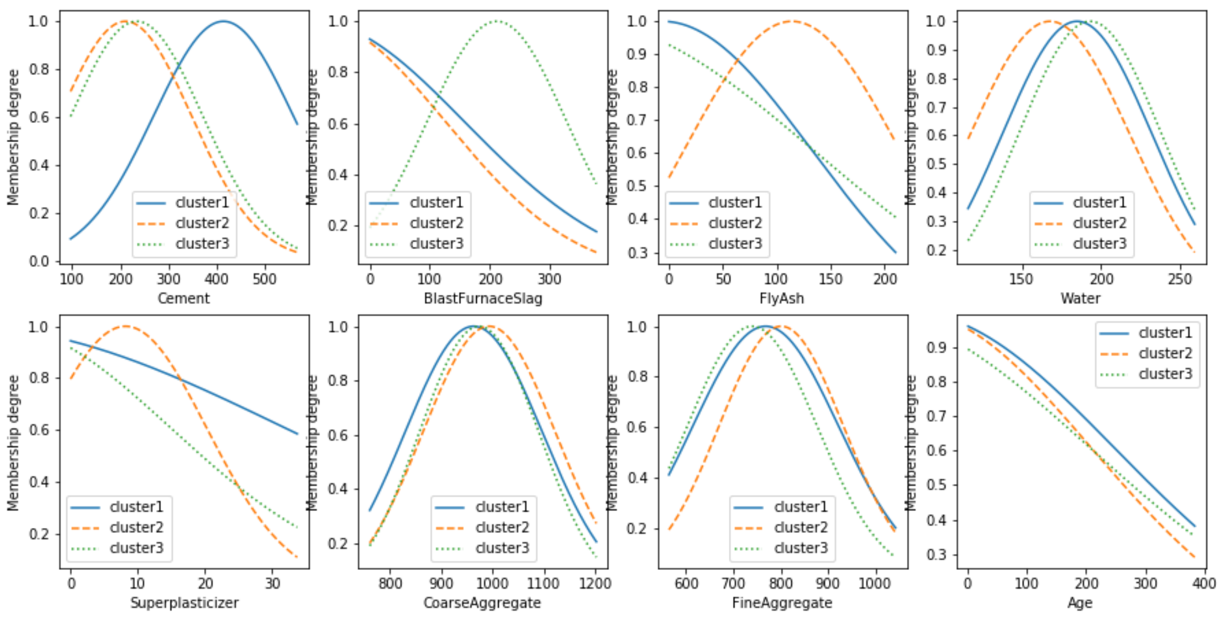

Fig. 2: The membership functions generated by pyFUME for the concrete data set.

Listing 1: The simple method to fit a fuzzy model to data using pyFUME.

```
from pyfume import *

# Set the path to the data and choose the number of clusters
path='./Concrete_data.csv'
nc=3

# Generate the Takagi-Sugeno FIS
FIS = pyFUME(datapath=path, nr_clus=nc)

# Calculate and print the accuracy of the generated model
print ("The calculated error is:", FIS.calculate_error())

## Use the FIS to predict the compressive strength of a new concrete sample
# Extract the model from the FIS object
model=FIS.get_model()

# Set the values for each variable
model.set_variable('Cement', 300.0)
model.set_variable('BlastFurnaceSlag', 50.0)
model.set_variable('FlyAsh', 0.0)
model.set_variable('Water', 175.0)
model.set_variable('Superplasticizer',0.7)
model.set_variable('CoarseAggregate', 900.0)
model.set_variable('FineAggregate', 600.0)
model.set_variable('Age', 45.0)

# Perform inference and print predicted value
print(model.Sugeno_inference(['OUTPUT']))
```

[19] J. C. Bezdek, “Models for pattern recognition,” in *Pattern Recognition with Fuzzy Objective Function Algorithms*. Springer, 1981, pp. 1–13.

[20] S. L. Chiu, “Fuzzy model identification based on cluster estimation,” *Journal of Intelligent & Fuzzy Systems*, vol. 2, no. 3, pp. 267–278, 1994.

[21] J. Alcalá-Fdez, J. M. Alonso, C. Castiello, C. Mencar, and J. M. Soto-Hidalgo, “Py4JFML: A Python wrapper for using the IEEE Std 1855-2016 through JFML,” in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019, pp. 1–6.

[22] A. Gegov, *Fuzzy Networks for Complex Systems*. Springer, 2010.

[23] T. E. Oliphant, “Python for scientific computing,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[24] ——, “Python for scientific computing,” *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 10–20, 2007.

[25] E. Jones, T. Oliphant, P. Peterson *et al.*, “Scipy: Open source scientific tools for python,” 2001. [Online]. Available: https://www.scipy.org

[26] M. S. Nobile, P. Cazzaniga, D. Besozzi, R. Colombo, G. Mauri, and G. Pasi, “Fuzzy Self-Tuning PSO: A settings-free algorithm for global optimization,” *Swarm and Evolutionary Computation*, vol. 39, pp. 70–85, 2018.

Listing 2: Flexible method to fit a fuzzy model to data using pyFUME.

```
from LoadData import DataLoader
from Splitter import DataSplitter
from ModelBuilder import SugenoFISBuilder
from Clustering import Clusterer
from EstimateAntecendentSet import AntecedentEstimator
from EstimateConsequentParameters import ConsequentEstimator
from Tester import SugenoFISTester

# Set the path to the data and choose the number of clusters
path='./Concrete_data.csv'
nr_clus=3

# Load and normalize the data
dl=DataLoader(path,normalize=1)
variable_names=dl.variable_names
dataX=dl.dataX
dataY=dl.dataY

# Split the data using the hold-out method in a training (default: 75%)
# and test set (default: 25%).
ds = DataSplitter(dl.dataX,dl.dataY)
x_train, y_train, x_test, y_test = ds.holdout(dataX, dataY)

# Cluster the training data (in input-output space) using FCM with default settings
cl = Clusterer(x_train, y_train, nr_clus)
cluster_centers, partition_matrix, _ = cl.cluster(method="fcm")

# Estimate the membership funtions of the system (default: mf_shape = gaussian)
ae = AntecedentEstimator(x_train, partition_matrix)
antecedent_parameters = ae.determineMF(x_train, partition_matrix)

# Estimate the parameters of the consequent (default: global fitting = True)
ce = ConsequentEstimator(x_train, y_train, partition_matrix)
consequent_parameters = ce.suglms(x_train, y_train, partition_matrix)

# Build a first-order Takagi-Sugeno model using Simpful
simpbuilder = SugenoFISBuilder(antecedent_parameters, consequent_parameters, variable_names)
model = simpbuilder.get_model()

# Calculate the mean squared error of the model using the test data set
print ("The calculated error is:", model.calculate_error())
```

[27] C. Fuchs, S. Spolaor, M. S. Nobile, and U. Kaymak, “A swarm intelligence approach to avoid local optima in fuzzy c-means clustering,” in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019, pp. 1–6.

[28] C. Fuchs, A. Wilbik, and U. Kaymak, “Towards more specific estimation of membership functions for data-driven fuzzy inference systems,” in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2018, pp. 1–8.

[29] R. Babuška, “Fuzzy modelling and identification toolbox,” *Control Engineering Laboratory, Faculty of Information Technology and Systems, Delft University of Technology, Delft, The Netherlands, version*, vol. 3, 2000.

[30] I.-C. Yeh, “Modeling of strength of high-performance concrete using artificial neural networks,” *Cement and Concrete research*, vol. 28, no. 12, pp. 1797–1808, 1998.

[31] D. E. Gustafson and W. C. Kessel, “Fuzzy clustering with a fuzzy covariance matrix,” in *Decision and Control including the 17th Symposium on Adaptive Processes, 1978 IEEE Conference on*. IEEE, 1979, pp. 761–766.

[32] N. R. Pal, K. Pal, J. M. Keller, and J. C. Bezdek, “A possibilistic fuzzy c-means clustering algorithm,” *IEEE transactions on fuzzy systems*, vol. 13, no. 4, pp. 517–530, 2005.

[33] E. H. Mamdani and S. Assilian, “An experiment in linguistic synthesis with a fuzzy logic controller,” *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[34] Y. Tsukamoto, “An approach to fuzzy reasoning method,” *Advances in Fuzzy Set Theory and Applications*, 1979.

[35] P. Angelov and R. Yager, “Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density,” in *2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*. IEEE, 2011, pp. 62–69.

[36] J. van den Berg, U. Kaymak, and W.-M. van den Bergh, “Fuzzy classification using probability-based rule weighting,” in *Proceedings of 2002 IEEE International Conference on Fuzzy Systems*, Honolulu, Hawaii, May 2002, pp. 991–996.

[37] ——, “Probabilistic reasoning in fuzzy rule-based systems,” in *Soft Methods in Probability, Statistics and Data Analysis*, ser. Advances in Soft Computing, P. Grzegorzewski, O. Hryniewicz, and M. A. Gil, Eds. Heidelberg: Physica Verlag, 2002, pp. 189–196.

[38] J. van Berg, U. Kaymak, and R. J. Almeida, “Conditional density estimation using probabilistic fuzzy systems,” *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 5, pp. 869–882, 2013.

[39] M. S. Nobile, G. Votta, R. Palorini, S. Spolaor, H. De Vitto, P. Cazzaniga, F. Ricciardiello, G. Mauri, L. Alberghina, F. Chiaradonna, and D. Besozzi, “Fuzzy modeling and global optimization to predict novel therapeutic targets in cancer cells,” *Bioinformatics, btz868*, 2019.

[40] S. Spolaor, M. S. Nobile, G. Mauri, P. Cazzaniga, and D. Besozzi, “Coupling mechanistic approaches and fuzzy logic to model and simulate complex systems,” *IEEE Transactions on Fuzzy Systems*, 2019.

[41] I. S. Association, “IEEE standard for fuzzy markup language,” 2016. [Online]. Available: https://standards.ieee.org/

[42] J. M. Soto-Hidalgo, J. M. Alonso, G. Acampora, and J. Alcalá-Fdez, “JFML: a java library to design fuzzy logic systems according to the IEEE std 1855-2016,” *IEEE Access*, vol. 6, pp. 54 952–54 964, 2018.

Listing 3: pyFUME's automatically generated code for the Simpful model for the concrete data set.

```
# WARNING: this source code was automatically generated by Simpfulifier.
from simpful import *

FS = FuzzySystem()
RULE1 = "IF (Cement IS cluster1) AND (BlastFurnaceSlag IS cluster1) AND (FlyAsh IS cluster1) AND
    (Water IS cluster1) AND (Superplasticizer IS cluster1) AND (CoarseAggregate IS cluster1) AND
    (FineAggregate IS cluster1) AND (Age IS cluster1) THEN (OUTPUT IS fun1)"
RULE2 = "IF (Cement IS cluster2) AND (BlastFurnaceSlag IS cluster2) AND (FlyAsh IS cluster2) AND
    (Water IS cluster2) AND (Superplasticizer IS cluster2) AND (CoarseAggregate IS cluster2) AND
    (FineAggregate IS cluster2) AND (Age IS cluster2) THEN (OUTPUT IS fun2)"
RULE3 = "IF (Cement IS cluster3) AND (BlastFurnaceSlag IS cluster3) AND (FlyAsh IS cluster3) AND
    (Water IS cluster3) AND (Superplasticizer IS cluster3) AND (CoarseAggregate IS cluster3) AND
    (FineAggregate IS cluster3) AND (Age IS cluster3) THEN (OUTPUT IS fun3)"
FS.add_rules([RULE1, RULE2, RULE3])

FS.set_output_function('fun1', '5.93e-02*Cement + -1.18e-02*BlastFurnaceSlag + -1.96e-02*FlyAsh +
    -2.38e-02*Water + 1.35e+00*Superplasticizer + -5.22e-02*CoarseAggregate + 2.28e-02*FineAggregate
    + 8.86e-02*Age + 4.97e+01')
FS.set_output_function('fun2', '3.44e-01*Cement + 3.00e-01*BlastFurnaceSlag + 3.58e-01*FlyAsh +
    1.97e-01*Water + 5.91e-01*Superplasticizer + 1.83e-01*CoarseAggregate + 2.19e-01*FineAggregate +
    2.96e-01*Age + -5.04e+02')
FS.set_output_function('fun3', '-1.15e-02*Cement + 8.05e-02*BlastFurnaceSlag + -7.30e-02*FlyAsh +
    -5.25e-01*Water + -7.51e-01*Superplasticizer + -5.63e-02*CoarseAggregate +
    -1.45e-01*FineAggregate + -3.76e-05*Age + 3.05e+02')

FS_1 = FuzzySet(function=Gaussian_MF(243.387029, 143.193521), term='cluster1')
FS_2 = FuzzySet(function=Gaussian_MF(207.987023, 125.660130), term='cluster2')
FS_3 = FuzzySet(function=Gaussian_MF(408.002063, 142.006995), term='cluster3')
MF_Cement = LinguisticVariable([FS_1, FS_2, FS_3], concept='Cement')
FS.add_linguistic_variable('Cement', MF_Cement)

FS_4 = FuzzySet(function=Gaussian_MF(206.107403, 122.491976), term='cluster1')
FS_5 = FuzzySet(function=Gaussian_MF(-83.389683, 211.605492), term='cluster2')
FS_6 = FuzzySet(function=Gaussian_MF(-119.439592, 267.033342), term='cluster3')
MF_BlastFurnaceSlag = LinguisticVariable([FS_4, FS_5, FS_6], concept='BlastFurnaceSlag')
FS.add_linguistic_variable('BlastFurnaceSlag', MF_BlastFurnaceSlag)

FS_7 = FuzzySet(function=Gaussian_MF(-97.019491, 237.333653), term='cluster1')
FS_8 = FuzzySet(function=Gaussian_MF(119.906630, 95.818603), term='cluster2')
FS_9 = FuzzySet(function=Gaussian_MF(-11.090035, 140.207001), term='cluster3')
MF_FlyAsh = LinguisticVariable([FS_7, FS_8, FS_9], concept='FlyAsh')
FS.add_linguistic_variable('FlyAsh', MF_FlyAsh)

FS_10 = FuzzySet(function=Gaussian_MF(193.106922, 48.895664), term='cluster1')
FS_11 = FuzzySet(function=Gaussian_MF(166.586342, 49.653055), term='cluster2')
FS_12 = FuzzySet(function=Gaussian_MF(188.404605, 46.939142), term='cluster3')
MF_Water = LinguisticVariable([FS_10, FS_11, FS_12], concept='Water')
FS.add_linguistic_variable('Water', MF_Water)

FS_13 = FuzzySet(function=Gaussian_MF(-5.405363, 22.805987), term='cluster1')
FS_14 = FuzzySet(function=Gaussian_MF(7.964341, 10.870129), term='cluster2')
FS_15 = FuzzySet(function=Gaussian_MF(-13.938112, 42.977322), term='cluster3')
MF_Superplasticizer = LinguisticVariable([FS_13, FS_14, FS_15], concept='Superplasticizer')
FS.add_linguistic_variable('Superplasticizer', MF_Superplasticizer)

FS_16 = FuzzySet(function=Gaussian_MF(968.153605, 123.938034), term='cluster1')
FS_17 = FuzzySet(function=Gaussian_MF(1004.065932, 130.560237), term='cluster2')
FS_18 = FuzzySet(function=Gaussian_MF(964.782349, 137.077242), term='cluster3')
MF_CoarseAggregate = LinguisticVariable([FS_16, FS_17, FS_18], concept='CoarseAggregate')
FS.add_linguistic_variable('CoarseAggregate', MF_CoarseAggregate)

FS_19 = FuzzySet(function=Gaussian_MF(742.820604, 144.626316), term='cluster1')
FS_20 = FuzzySet(function=Gaussian_MF(797.656516, 126.027296), term='cluster2')
FS_21 = FuzzySet(function=Gaussian_MF(768.521770, 149.111213), term='cluster3')
MF_FineAggregate = LinguisticVariable([FS_19, FS_20, FS_21], concept='FineAggregate')
FS.add_linguistic_variable('FineAggregate', MF_FineAggregate)

FS_22 = FuzzySet(function=Gaussian_MF(-99.208188, 339.230593), term='cluster1')
FS_23 = FuzzySet(function=Gaussian_MF(-85.161565, 273.964216), term='cluster2')
FS_24 = FuzzySet(function=Gaussian_MF(-93.942323, 336.506917), term='cluster3')
MF_Age = LinguisticVariable([FS_22, FS_23, FS_24], concept='Age')
FS.add_linguistic_variable('Age', MF_Age)



# end of automatically generated code #
```