http://dx.doi.org/10.1109/FUZZ-IEEE.2019.8858811

Py4JFML: A Python wrapper for using the IEEE Std 1855-2016 through JFML

Jesús Alcalá-Fdez*, Jose M. Alonso†, Ciro Castiello‡, Corrado Mencar‡ and José M. Soto-Hidalgo§
*Department of Computer Science and Artificial Intelligence, University of Granada, Spain
Email: jalcala@decsai.ugr.es
†Centro Singular de Investigación en Tecnoloxías da Información, University of Santiago de Compostela, Spain
Email: josemaria.alonso.moral@usc.es
‡Department of Informatics, University of Bari Aldo Moro, Italy
Email: ciro.castiello@uniba.it, corrado.mencar@uniba.it
§Department of Electrical & Computer Engineering, University of Cordoba, Spain
Email: jmsoto@uco.es

***Abstract*—JFML is an open source Java library aimed at facilitating interoperability of fuzzy systems by implementing the IEEE Std 1855-2016 – the IEEE Standard for Fuzzy Markup Language (FML) that is sponsored by the IEEE Computational Intelligence Society. We developed a Python wrapper for JFML that enables to use all the functionalities of JFML through a Python 3.x module. The bridge between Python and Java is accomplished through the use of the Py4J framework. As a result, the possibility of using the IEEE standard for representing fuzzy systems is enlarged to a wider community of developers and knowledge engineers, with minimal code redundancy. Experiments show full interoperability between Python programs and JFML without any tangible overhead. We illustrate the use of Py4JFML in a beer style classification case study.**

I. Introduction

Fuzzy Inference Systems (FIS) are rule-based expert systems based on fuzzy set theory to represent the semantics of rules and to process inference when input data are provided [1]. The success of FIS is widely recognized in the fields of control, decision making, system identification, and other fields of Computer Science and Engineering. In consequence of this success, a number of tools have been developed, both with commercial and public licenses [2]. Some of the most widely known tools are Matlab Fuzzy Logic Toolbox[1], FisPro[2] [3], GUAJE[3] [4], FuzzyLite[4] or JuzzyOnline[5], among others. The scientific and industrial interest towards FIS—and fuzzy systems in general—led the IEEE Computational Intelligence Society (IEEE-CIS) to constitute a Fuzzy Systems Technical Committee which includes the Task Force on Fuzzy Systems Software[6] with the goal of promoting the research, development, education, and understanding of Fuzzy Systems Software, regarding both theory and applications.

In 2016, the IEEE Std 1855-2016 [5] was sponsored by IEEE-CIS [6] and defined to represent FIS in the Fuzzy Markup Language, which is based on XML [7]. This standard is intended to supersede a plethora of proprietary formats (often incompatible) with a recognized and extensible format based on a widely used language. On the basis of this standard, the JFML library has been recently developed [8]. JFML is a Java library that implements all the features defined by the IEEE Std 1855-2016. Accordingly, it is capable of handling the four different types of FIS (namely Mamdani, Takagi-Sugeno-Kang, Tsukamoto and AnYa) that are enclosed in the W3C XML Schema definition of the standard. Additionally, JFML supports legacy formats such as the Predictive Model Markup Language (PMML) [9] and the Fuzzy Control Language (FCL) [10]. On the overall, JFML is an enabling technology for fostering FIS into widespread real-world usage.

JFML is written in Java, a very popular general-purpose programming language and developing ecosystem. On the other hand, in scientific computing there are other languages—like R and Python—that are also widely used because of their syntactical simplicity, dynamic typing and fast prototyping. In particular, Python gained a lot of attention in the last few years because of the availability of state-of-the-art libraries for scientific computing, such as SciPy[7], NumPy[8] for fast numerical computations; matplotlib[9] for easy-to-use and high-quality plotting; Jupyter[10] for interactive computing; pandas[11] and scikit-learn[12] for data analysis and data mining. As a matter of fact, a number of scientific research groups (often not directly related to Computer Science) are increasingly adopting Python for developing software prototypes, running experiments and analyzing data [11].

Unfortunately, only a few Python packages are related to

[1]https://www.mathworks.com/products/fuzzy-logic.html
[2]https://www.fispro.org/en/
[3]https://sourceforge.net/projects/guajefuzzy/
[4]https://fuzzylite.com/
[5]http://ritweb.cloudapp.net:8080/JuzzyOnline/juzzy
[6]https://sci2s.ugr.es/TF-FSS
[7]http://www.scipy.org
[8]http://www.numpy.org/
[9]https://matplotlib.org/
[10]http://jupyter.org/
[11]https://pandas.pydata.org/
[12]https://scikit-learn.org

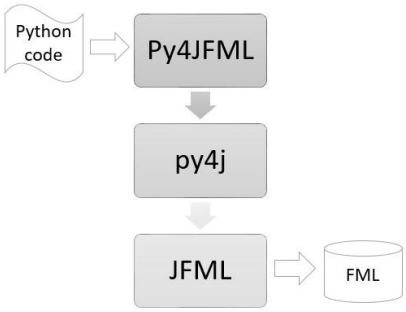

Figure 1. General architecture of Py4JFML.

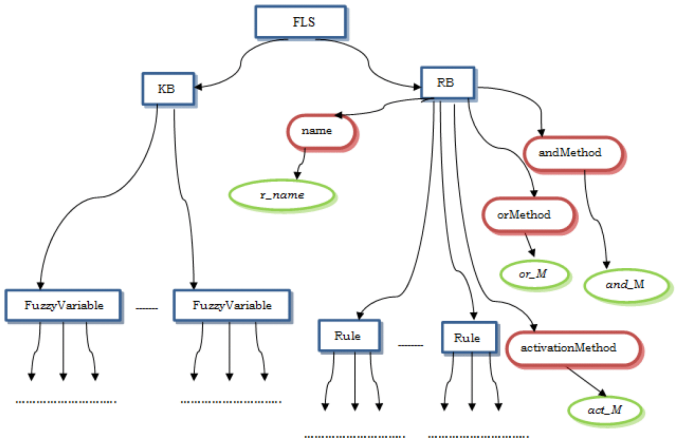

Figure 2. Top hierarchy of FML (Source: IEEE [5]).

FIS (e.g., pyfuzzy[13], fuzzycreator[14] or SyFSeL[15] [12]). Based on this consideration, the possibility of using the IEEE Std 1855-2016 in Python programs is beneficial for two main reasons: (i) to enlarge the community of people who can use FIS technology with the stable support of a standard; (ii) to allow interaction of FIS with state-of-the-art numerical and machine-learning algorithms available in Python libraries such as SciPy and scikit-learn.

In this paper we describe Py4JFML, which is a Python wrapper for JFML. In essence, Py4JFML is a library that exposes the same functionalities of JFML but in the Python language. As a consequence, it is possible to write Python programs to read, evaluate, modify and store FIS according to the IEEE Std 1855-2016 as well as to compare and interact such systems with other Python objects. Py4JFML uses the Java bytecode of JFML through the wrapper library Py4J[16], therefore there is no need for code duplication; on the contrary, the evolution of Py4JFML follows the evolution of JFML effortlessly. The general architecture of Py4JFML is sketched in Fig. 1. The software is freely available under GPLv3 licence at http://www.uco.es/JFML/software.

In Section II, the technologies that are involved in Py4JFML are briefly overviewed, while in Section III the architecture and main design features of Py4JFML are reported. Section IV describes the integration of Py4JFML and scikit-learn in an illustrative use case related to beer style classification. The paper ends with some final remarks about future developments of Py4JFML.

II. Background

A. IEEE Std 1855-2016

This standard defines a specification language, named Fuzzy Markup Language (FML), which is based on the XML meta-language to represent FIS in a human-readable and hardware independent way [5]. FML includes an extensible schema that defines the basic components of different types of FIS —including Mamdani, Tsukamoto and Takagi-Sugeno-Kang (TSK) [13]; but also the most recent AnYa [14]. The standard focuses on interoperability, i.e., the exchange of XML-based instances of FIS between various systems without taking care of hardware specifications. Hardware-dependent drivers can be designed through XSLT documents [15]. Thus, overcoming some limitations of embedded systems [16].

FML is aimed at improving the IEC 61131-7 FCL [10] as well as to replace a number of *de facto* standards still in use, such as the FIS Matlab file format, which is widely used but has limited capabilities when compared to FML (e.g. lack of interoperability due to the use of a proprietary format, or lack of a distribution model).

A FIS described in FML is characterized by five main components: (1) fuzzy knowledge base; (2) fuzzy rule base; (3) inference engine; (4) fuzzification subsystem; and (5) defuzzification subsystem. Furthermore, each component is described by a hierarchy of sub-components in order to fully specify a FIS (see Fig. 2). All such components are grouped into XML tags that also specify the IP address of devices that compute them, thus enabling networked interoperability among devices.

B. JFML

JFML is an open source Java library that allows to design FIS according to the IEEE Std 1855-2016 [8]. This library offers a complete implementation of all FIS types enclosed in the standard; additionally JFML includes a module to import/export FIS in accordance with FCL and PMML documents, but also the proprietary '*.fis' format used by the Matlab Fuzzy Logic Toolbox.

JFML has been designed following the hierarchical structure of FML. It follows an object-oriented approach and a modular design based on the same labeled tree structure that FML uses to represent FIS. The library relies on Java Architecture for XML Binding (JAXB) to bind FML documents to Java representations, providing an efficient way for I/O operations. Accordingly, classes are organized in several interdependent packages, which implement specific features of FML following JAXB requirements. In addition, JFML includes the

[13]http://pyfuzzy.sourceforge.net/
[14]https://bitbucket.org/JosieMcCulloch/fuzzycreator
[15]https://bitbucket.org/JosieMcCulloch/syfsel
[16]https://www.py4j.org/

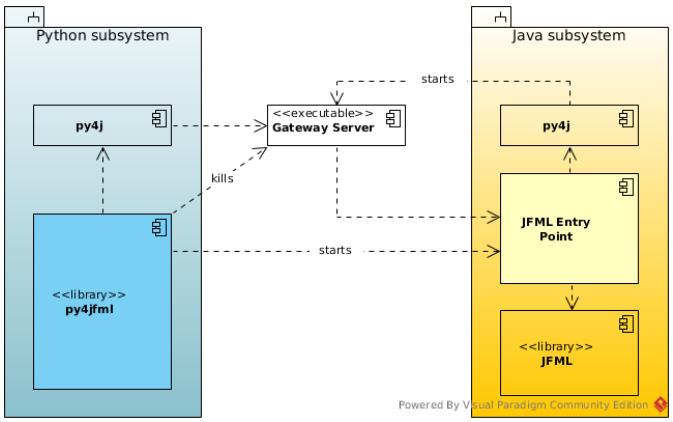

Figure 3. Component diagram of Py4JFML.

extension methods considered in the standard, thus facilitating the integration of changes in future releases of the standard.

C. Py4J

Py4J[17] is an open source software that works as a bridge between a Python interpreter and the Java Virtual Machine (JVM). By exploiting the similarities between the two languages, it is possible to access all objects residing in a JVM as if they were Python objects, thus interoperability between Java and Python programs is maximized.

There are some minor technical issues regarding specific Python data structures that are not implemented in Java and vice-versa; however they can be easily overcome. Communication between Python and Java programs is made possible through a *Gateway Server*, i.e., a server that is started by the Java program and is listening a specific local TCP port; the Python side of Py4J transmits all messages to Java objects through the Gateway Server, which in turn invokes the JVM and returns back the results (including possible exceptions) to the Python interpreter. This mechanism requires some extra-code in Java so as to make JFML classes accessible to Python through the Gateway Server. Also, the functionality of killing the server within the Python program is required in order to avoid memory leakage due to the server still running after the end of the Python program.

III. Py4JFML

Py4JFML is a Python 3.x library designed for mirroring the functionality of JFML in Python. For this purpose, the architecture of Py4JFML has two main components: (1) the Py4JFML module containing all bridge classes mirroring the corresponding JFML classes; and (2) the JFML entry-point in Java. These two components belong to the Pyhton and Java subsystems as depicted in Fig. 3.

In essence, the Java subsystem is deployed as a runnable Java Archive (JAR) file which is started from within Py4JFML in order to start up the Gateway Server. It is worth noting that this is done transparently to the user as soon as the library is included in a program thanks to Python's module management system. Once the server is up, the Py4JFML library can connect to the JFML library through Py4J as follows:

1) A Py4JFML object sends a message to Py4J in order to connect to a JFML object.
2) Py4J sends the request to the JVM through the Gateway Server.
3) The Gateway Server is able to connect to the JFML Entry Point, which is a Java object that is visible to the server.
4) The JFML Entry Point receives the message and dispatches it to the right JFML object.
5) The result is back-propagated to the original Python object.

All these operations are transparent to the user. In this way, it is possible to mirror all the classes available in JFML in Py4JFML without rewriting code.

As an example, in the case an actor wants to load a FML file containing a FIS, he/she could use the library function `load(fml_file)`. The sequence of operations that are executed are sketched in Fig. 4. A call to the Py4JFML module `load` (1) causes a message (1.1) to Py4J in order to get the reference to the entry point object, which is a Python object that is able to communicate with the Gateway Server. The entry point is then asked to get a JFML Factory (1.2), hence it sends a message (1.2.1) to the Gateway Server. The Gateway Server contacts the JFML entry point (2), which contains as many methods as the number of JFML packages, each of them returning an instance of a factory object (in this example, the JFML entry point is requested to return a reference to a JFML Factory). The JFML Factory is instantiated (2.1) and its reference returned to the Gateway Server (2.2), then to the Python entry point (3). This will serve to instantiate a "proxy" JFML Factory in Python (1.2.2), i.e., a Python object that exposes the same methods of the specific JFML Factory: whenever a method is called to the proxy, it launches a call to the corresponding method in the Java object through the mediation of the Gateway Server.

Once the proxy is available, Py4JFML asks for an instance of the JFML object (1.3); this is translated into a message to the Gateway Server (1.3.1) and, in turn, to a call to the JFML Factory (4), which instantiates a JFML object (4.1) and returns its reference (4.2, 5). As a result, a new proxy is created in Python, now referring to the specific JFML object in Java. The JFML Java instance is responsible of some I/O operations, such as loading and writing FML files. Therefore, Py4JFML asks the proxy to load a FML file (1.5); again, this request is translated into a message to the Gateway Server (1.5.1) and to the JFML object (6). Then, JFML loads the FML file and creates a *FuzzyInferenceSystem* instance (not showed in the picture for the sake of space); it is then returned to Py4JFML with the proxy mechanism (6.1, 7, 1.6) and finally returned to the actor (1.7).

A similar mechanism has been implemented for all classes belonging to the JFML library. In total, 112 classes belonging to the JFML library have been mapped into Python.

[17]https://www.py4j.org/

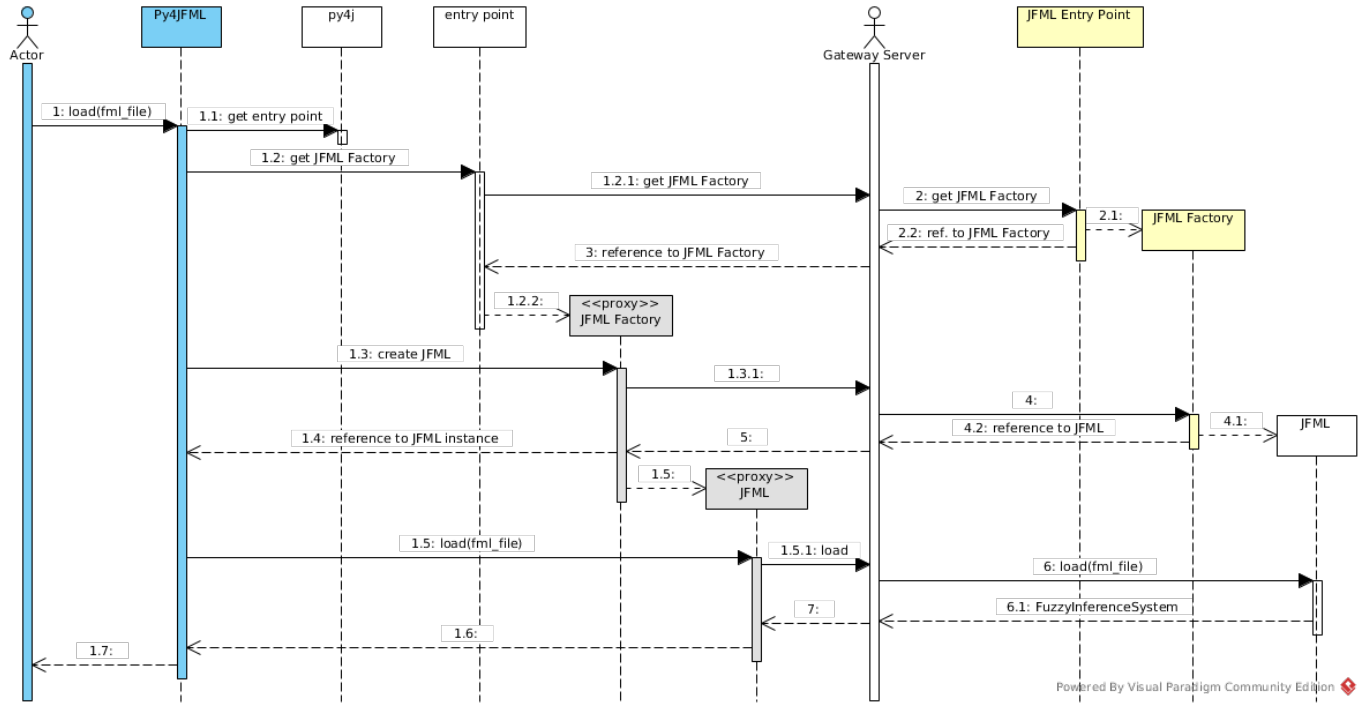

Figure 4. Sequence diagram for the `load` function. Blue elements are Python components implemented in Py4JFML; yellow elements are Java components implemented in Py4JFML; grey elements are auto-generated Python objects; white components are external to Py4JFML.

```
iris = FuzzyInferenceSystem("iris")
kb = KnowledgeBaseType()
iris.setKnowledgeBase(kb)
pw = FuzzyVariableType(name="PetalWidth",
        domainLeft=0.1, domainRight=2.5)
pw_low = FuzzyTermType(name="low",
        type_java=FuzzyTermType.TYPE_trapezoidShape,
        param=[0.1, 0.1, 0.244, 1.087])
pw.addFuzzyTerm(pw_low)
...
kb.addVariable(pw)
...
rb =MamdaniRuleBaseType("rulebase-iris")
r1 =FuzzyRuleType(name="rule1", connector="and",
        connectorMethod="MIN", weight=1.0)
...
rb.addRule(r1)
...
iris.addRuleBase(rb)
...
Py4jfml.writeFSTtoXML(iris, "iris.fml")
Py4jfml.kill()
```

Listing 1: A Python excerpt to create a Mamdani FIS.

```
FuzzyInferenceSystem iris = new FuzzyInferenceSystem(
    "iris");
KnowledgeBaseType kb = new KnowledgeBaseType();
iris.setKnowledgeBase(kb);
FuzzyVariableType pw = new FuzzyVariableType(
    "PetalWidth", 0.1f, 2.5f);
FuzzyTermType pw_low = new FuzzyTermType("low",
    FuzzyTermType.TYPE_trapezoidShape,
    (new float[] { 0.1f, 0.1f, 0.244f, 1.087f }));
pw.addFuzzyTerm(pw_low);
...
kb.addVariable(pw);
...
MamdaniRuleBaseType rb = new MamdaniRuleBaseType(
    "rulebase-iris");
FuzzyRuleType r1 = new FuzzyRuleType("rule1", "and",
    "MIN", 1.0f);
...
rb.addRule(r1);
...
iris.addRuleBase(rb);
...
File irisXMLFile = new File("iris.fml");
JFML.writeFSTtoXML(iris, irisXMLFile);
```

Listing 2: A Java excerpt to create a Mamdani FIS.

Listings 1 and 2 show two excerpts of Python and Java programs, respectively, for creating a Mamdani FIS to be written in a FML file. The strong similarity of the two program codes can be appreciated; with the only exception of the extra line in Python to kill the Gateway Server at the end of the program.

The use of Py4J to interface Py4JFML to JFML requires an overhead due to the communication between the Python interpreter and the JVM that is mediated by the Gateway Server. Empirical tests evidenced an overload between 20ms and 60ms on standard personal computers[18], which is acceptable for research in most applications without real-time requirements.

IV. Use Case

In order to show the interoperability achieved by Py4JFML, we implemented a typical machine learning use case in which some models are compared on a dataset through 10-fold cross

[18]CPU Intel I5 / AMD A6-5350M, RAM 4-8GB, Linux OS.

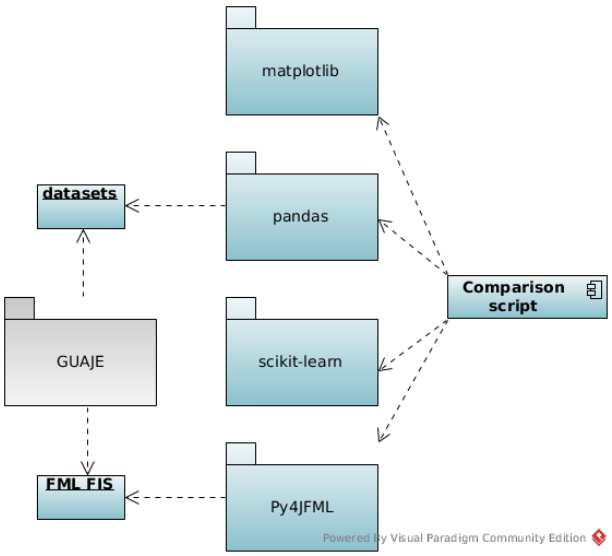

Figure 5. Component diagram in the use case.

validation. We used the BEER dataset, which is made up of 400 instances [17] classified in eight beer styles (Blanche, Lager, Pilsner, IPA, Stout, Barleywine, Porter, and Belgian Strong Ale) in terms of three attributes (color, bitterness and strength). The implemented workflow is the following (all steps are repeated for each fold):

1) Three Mamdani-type FIS have been designed through GUAJE. In particular, the FIS have been initialized with 3, 6 and 9 linguistic terms for each fuzzy partition; then, the fuzzy decision tree method has been applied to generate interpretable fuzzy rules. Finally, the resulting FIS have been simplified following the HILK methodology [18]. The final FIS have been exported as FML files for further processing.
2) The FIS generated in the previous step have been imported in Python through Py4JFML, then they have been evaluated on the test set. Notice that the Pandas package has been used to manage data in Python.
3) Three machine learning models (ML) have been designed by using Python's scikit-learn, namely a Decision Tree Classifier, a Random Forest Classifier and a Support Vector Classifier.[19] Like in the previous step, the resulting models have been evaluated on the test set.
4) Confusion matrices and average classification accuracy have been computed for all models, namely FIS and ML; results have been visualized by using pyplot, a subset of the matplotlib library that provides a MATLAB-like plotting framework.

Fig. 5 shows the diagram of the main components involved in the use case. It is possible to notice that GUAJE is decoupled from all other components; communication takes place through standard format files (Comma Separated Values, CSV, for datasets and FML for FIS).

[19]Default parameters have been used for all the models.

Table I
COMPARATIVE RESULTS

Model	Avg. accuracy
FIS (3 fuzzy sets per feature)	67.5%
FIS (6 fuzzy sets per feature)	93.5%
FIS (9 fuzzy sets per feature)	94.5%
ML (Decision Tree Classifier)	95.0%
ML (Random Forest Classifier)	**95.7%**
ML (Support Vector Classifier)	87.3%

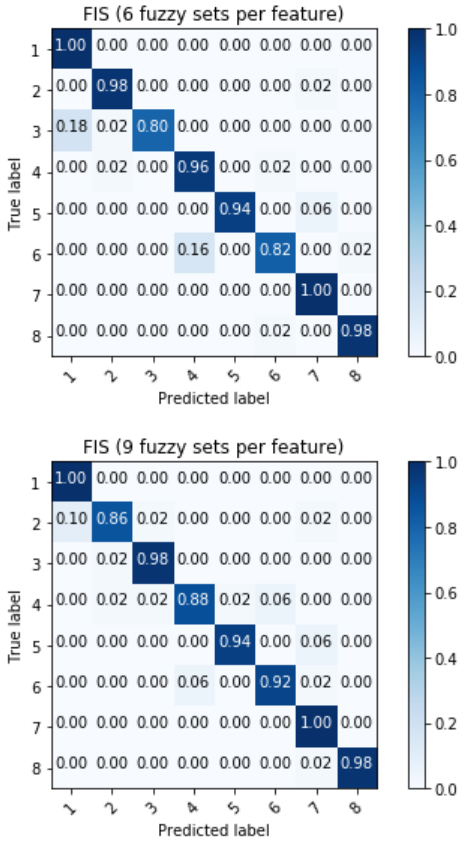

Figure 6. Confusion matrices for FIS.

Results are collected by using standard Python commands and reported in Table I. The `confusion_matrix` function available in scikit-learn has been used to compute confusion matrices; then a simple procedure based on pyplot has been written to draw them (e.g., Figs. 6 and 7 depict the confusion matrices for models achieving average accuracy higher than 90% in Table I).

V. CONCLUDING REMARKS

The design of Py4JFML allows full interoperability of FIS models with Python libraries, especially those concerned with machine learning like scikit-learn. In particular, the use case showed that FIS designed by stand-alone tools like GUAJE can be seemlessly used in Python scripts, thanks to the use of FML to store FIS in files.

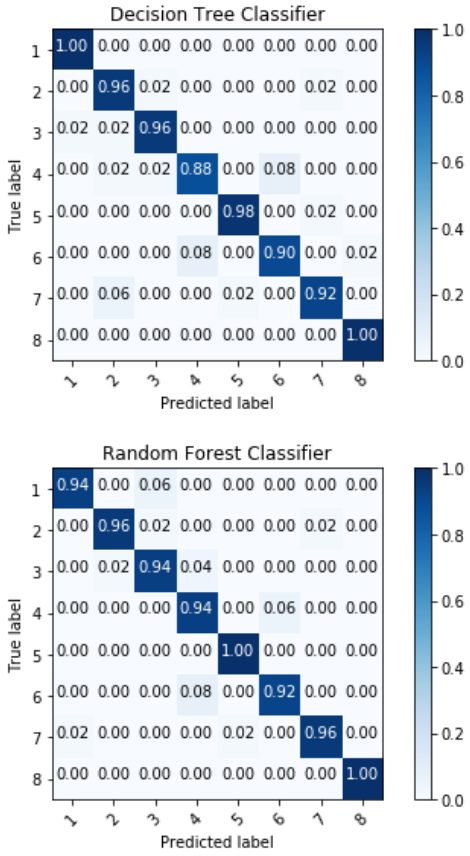

Figure 7. Confusion matrices for ML models.

Py4JFML follows the line of development of its companion library, JFML. In particular, a future development of JFML is to serve as a universal conversion tool for different formats currently used to store FIS [8]. Also, the capability of distributing computations in a network of systems is part of the future development of JFML. Py4JFML naturally fits with this development line. Additionally, an adapter module is under consideration, so as to make Py4JFML available with an interface that is homogeneous with scikit-learn; in this way, FIS could be used without effort by users that are accustomed with scikit-learn interface. As future work, we will enhance Py4JFML with learning and visualization modules for membership functions, rules and fuzzy inference; supported by those packages already existing in Python.

Py4JFML source code (along with documentation; including the scripts needed to run the use case in Section IV) is freely available under GPLv3 licence at

http://www.uco.es/JFML/software

https://github.com/cmencar/py4jfml/

ACKNOWLEDGMENTS

Jose M. Alonso is *Ramón y Cajal* Researcher (RYC-2016-19802). This research was also funded by the Spanish Ministry of Science, Innovation and Universities (grants TIN2017-84796-C2-1-R, TIN2017-90773-REDT, and TIN2017-89517-P) and the Galician Ministry of Education, University and Professional Training (grants ED431F 2018/02, ED431C 2018/29 and "accreditation 2016-2019, ED431G/08"). Financial support from the European Union (European Regional Development Fund - ERDF), is gratefully acknowledged. The authors wish also to thank the graduate students M. G. Addati and N. Bruno for their support in implementing Py4JFML.

REFERENCES

[1] S. Guillaume, "Designing fuzzy inference systems from data: An interpretability-oriented review," *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 3, pp. 426–443, 6 2001.

[2] J. Alcala-Fdez and J. M. Alonso, "A Survey of Fuzzy Systems Software: Taxonomy, Current Research Trends, and Prospects," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 1, pp. 40–56, 2 2016.

[3] S. Guillaume and B. Charnomordic, "Learning interpretable fuzzy inference systems with FisPro," *Information Sciences*, vol. 181, no. 20, pp. 4409–4427, 10 2011.

[4] J. M. Alonso and L. Magdalena, "Generating Understandable and Accurate Fuzzy Rule-Based Systems in a Java Environment," in *Fuzzy Logic and Applications (WILF)*, ser. Lecture Notes in Computer Science, A. M. Fanelli, W. Pedrycz, and A. Petrosino, Eds., vol. 6857. Springer, Berlin, Heidelberg, 2011, pp. 212–219.

[5] IEEE-SA Standards Board, "IEEE Standard for Fuzzy Markup Language," *IEEE Std 1855-2016*, pp. 1–89, 2016, https://standards.ieee.org/findstds/standard/1855-2016.html.

[6] G. Acampora, B. di Stefano, and A. Vitiello, "IEEE 1855^{TM}: The first IEEE standard sponsored by IEEE Computational Intelligence Society," *IEEE Computational Intelligence Magazine*, vol. 11, pp. 4–7, 2016.

[7] G. Acampora, "Fuzzy Markup Language: A XML Based Language for Enabling Full Interoperability in Fuzzy Systems Design," in *On the Power of Fuzzy Markup Language*. Springer Berlin Heidelberg, 2013, pp. 17–31.

[8] J. M. Soto-Hidalgo, J. M. Alonso, G. Acampora, and J. Alcala-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.

[9] A. Guazzelli, M. Zeller, W.-C. Lin, G. Williams, and others, "PMML: An open standard for sharing models," *The R Journal*, vol. 1, no. 1, pp. 60–65, 2009.

[10] *International Electrotechnical Commission technical committee industrial process measurement and control. IEC 61131 - Programmable Controllers - Part 7: Fuzzy control programming*. IEC, 2000.

[11] J. M. Perkel, "Why Jupyter is data scientists' computational notebook of choice," *Nature*, vol. 563, no. 7729, pp. 145–146, 10 2018.

[12] J. McCulloch, "SyFSeL: generating synthetic fuzzy sets made simple," in *IEEE International Conference on Fuzzy Systems. FUZZ-IEEE 2018*. Rio de Janeiro, Brazil: IEEE, 2018, pp. 1–6.

[13] T. J. Ross, *Fuzzy logic with engineering applications*, 4th ed. John Wiley & Sons, 2016.

[14] P. Angelov and R. Yager, "Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density," in *IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*, 4 2011, pp. 62–69.

[15] G. Acampora and A. Vitiello, "Extending IEEE Std 1855 for designing Arduino™-based fuzzy systems," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 7 2017, pp. 1–6.

[16] J. M. Soto.Hidalgo, A. Vitiello, J. M. Alonso, G. Acampora, and J. Alcala-Fdez, "Design of fuzzy controllers for embedded systems with JFML," *International Journal of Computational Intelligence Systems*, pp. 204–214, 2019.

[17] G. Castellano, C. Castiello, and A. M. Fanelli, "The FISDeT software: Application to beer style classification," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Naples, Italy, 2017, pp. 1–6.

[18] J. M. Alonso and L. Magdalena, "HILK++: an interpretability-guided fuzzy modeling methodology for learning readable and comprehensible fuzzy rule-based classifiers," *Soft Computing*, vol. 15, no. 10, pp. 1959–1980, 2011.