A Universal Representation Framework for Fuzzy Rule-Based Systems Based on PMML

Lala Septem Riza[a,*], Christoph Bergmeir[b],
Francisco Herrera[a], José Manuel Benítez[a]

[a] *Department of Computer Science and Artificial Intelligence, CITIC-UGR, IMUDS, University of Granada*
[b] *Clayton School of Information Technology, Faculty of Information Technology, Monash University, Melbourne*

Abstract

Fuzzy rule-based systems (FRBSs) have been implemented and deployed by researchers and practitioners in many different application contexts to deal with complex real-world problems. However, a challenge that still remains mainly unresolved is the lack of a general representation framework for FRBSs that allows interoperability among platforms and applications. Therefore, this paper proposes a universal framework for representing FRBS models, called frbsPMML, which is a format adopted from the Predictive Model Markup Language (PMML). Three models, which can be used for handling regression and classification tasks, are specified by the proposed representations: Mamdani, Takagi Sugeno Kang, and fuzzy rule-based classification systems. A key advantage of FRBS model specification in frbsPMML is that high degrees of transparency and interpretability can be achieved. Moreover, an easier deployment and integration of FRBSs with other tools for modelling and data analysis becomes possible, as well as easier reproducibility of research. In this paper we also present two implementations of the proposed standard model format: the R package "frbs" as an frbsPMML producer and consumer, and a Java implementation of an frbsPMML consumer, named "frbsJpmml." A comparison with other representations and examples to show schemata of the frbsPMML format are provided.

Keywords: Fuzzy inference systems, Fuzzy system models, R language, Interpretability fuzzy models, System interoperability, Reproducible research

1. Introduction

Fuzzy rule-based systems (FRBSs) are models based on fuzzy sets proposed by Zadeh [1] that express knowledge in a set of fuzzy rules to address complex real-world problems. The concepts are popular because FRBSs allow to cope with uncertainty, imprecision, and non-linearity. Another reason is their interpretability so that the FRBS model generated from data can be interpreted, verified, and modified by human experts. Furthermore, this allows to combine a model obtained by learning from data with human expert knowledge, called the grey box modelling [2]. In a real-world scenario, a priori knowledge is required when available data are not sufficient —because of, e.g., outlier data and a limited number of data— to construct a reliable and sophisticated model.

*Corresponding Author, email: lala.s.riza@decsai.ugr.es

In an FRBS model, knowledge is represented as a combination of a database and a rulebase. Fuzzy set definitions (i.e., linguistic values) and parameters of membership functions are included in the database whereas the rulebase refers to a set of rules. There are membership functions available that can be used to calculate a degree of membership such as, e.g., Gaussian, triangular, and trapezoid functions. Regarding the rulebase, we can construct a set of rules according to the following models. The Mamdani model constructs the antecedent and consequent parts by involving linguistic values [3, 4]. Therefore, the model can be interpreted easily and is more flexible to change than a classical model. Another model, the Takagi Sugeno Kang (TSK) model, replaces the consequent part of the Mamdani model by a linear combination of input variables [5, 6]. It offers different benefits compared to the Mamdani model, being the main benefit an improved accuracy of prediction. In classification tasks, fuzzy rule-based classification systems (FRBCS) are models based on the Mamdani model that involve categorical values on the consequent parts [7–9].

FRBSs can be constructed by using knowledge from human experts or by learning from data. In some cases, it may not be feasible to extract knowledge from human experts, for instance, when experts are not available or the problem at hand is too large and complex to be handled. Therefore, approaches generating knowledge from available data are developed and implemented as software systems.

Nowadays, many such software systems are available for both academic and industry purposes. For example, "Xfuzzy" is an open-source framework based on fuzzy inference-based systems [10]. To represent an FRBS model, it uses a formal language called the Xfuzzy 3.0 specification language (XFL3). XFL3 contains declarations about membership functions, a set of rules, and other parameters. In the MATLAB environment, there is the Fuzzy Logic Toolbox [11]. It is developed by utilizing Simulink, which is a graphical user interface (GUI) used for data flow, and a command-line mode to build an FRBS model saved in the so-called *.fis* file format. In the R environment [12, 13], there is the "frbs" package that includes over 10 learning methods for regression and classification [14]. Additionally, apart from these most relevant software systems, there are others, e.g., "FisPro" [15], "GUAJE" [16], and "KEEL" [17, 18]. Though available software libraries provide many useful features for tackling real-world problems, we note that there is not a standard interface that connects between them, so that it is difficult to exchange models between the different software systems. As interoperability is an important issue not only in industry cases but also for academic purposes, this is a shortcoming that we address with our work.

For academic purposes, when proposing a new algorithm, it is important to perform an experimental study and then provide a comparison with other related approaches, to analyze the behaviour and performance of the new technique. One of the critical issues regarding this process is that it typically requires to understand and analyze different formats of models produced by various software libraries. Naturally, it is difficult to make a comprehensive comparison, e.g., according to the interpretability perspective. And even further processing steps involving the models, such as assembling and aggregation, are almost impossible. Therefore, we see that a universal representation framework is urgent to be designed and implemented, especially for the academic research community. Another advantage of the universal representation is that it promotes reproducible research [19] as research results can be

archived, distributed, replicated, and reproduced easily in a standard format. In other words, multiple research groups using different platforms can share and analyze models.

In industry, interoperability is often very important and required, as users dedicated to model construction may often be located in another department as the users of the models, also using different computer programs in their workflows. For example, an insurance company may have a department to generate models of a risk level. Then, there may be another department that is in charge of applying the model for prediction of the risk level of somebody according to given profiles. Therefore, in this case the obtained models would be distributed to many places. Furthermore, it is desirable for the resulting models to be easily understood and communicated. Again, from an industry perspective, a universal representation framework that satisfies these requirements is desirable.

There exists an open standard dedicated to the representation of data mining models and sharing of different applications, which is the Predictive Model Markup Language (PMML) [20]. The standard currently includes several models, such as association rules, cluster models, neural networks, etc. Furthermore, members from companies such as IBM, SAS, and Microsoft are part of the consortium developing PMML, so that it can be considered as the standard framework for data mining model interchange.

A main contribution of PMML is to provide interoperable schemata of predictive models. Using PMML, we can easily perform these tasks as our models are documented in an XML-based language. Human experts can also update and modify the model on the files directly. Furthermore, the study of [21] shows that PMML has been deployed in cloud computing [22–24]. Therefore, we can apply our models anywhere without worrying about details of applications and resources. However, FRBSs are not yet among the models supported by PMML.

In this work, we contribute to overcome this shortcoming by designing and implementing a proposal for a universal representation framework of FRBSs based on PMML, called frbsPMML. As mentioned before, a universal representation framework naturally offers advantages for: interoperability and reproducible research. Moreover, two essential aspects considered for measuring the performance in FRBSs are accuracy and interpretability. Using FRBSs in the frbsPMML format, we gain a benefit of high levels of interpretability. Due to the XML-based language, an FRBS model becomes readable both by humans and machines. Therefore, human experts can easily check, verify, and modify the model. Additionally, from the FRBS point of view, interpretability mainly refers to the capability of the fuzzy model to express the behaviour of the system in an understandable way, which depends on several aspects: the model structure, the number of input variables, the number of fuzzy rules, the number of linguistic terms, and the shape of the fuzzy sets [25]. FRBSs in the frbsPMML format allow to represent a model in accordance with these criteria, as a database and a rulebase are specified by the XML-based language in a flexible way.

Additionally, in this paper we present functionalities of the "frbs" package [14] to produce and consume an FRBS model in frbsPMML format. Another implementation, written in Java, is presented as well. It is called "frbsJpmml," and can be used to deploy frbsPMML models and perform predictions on new, unknown data.

The remainder of this paper is structured as follows. Section 2 presents an introduction to PMML, along with

its basic components and their implementations. A universal representation of FRBSs is proposed in Section 3. Then implementations are discussed in Section 4, where we present functions in "frbs" and "frbsJpmml" to support importing, exporting, and deploying FRBS models in the proposed format. Section 5 depicts some advantages of the new representation, together with a comparison to other representations of FRBS models. Some examples showing how to use the packages for regression and classification and how to interpret the obtained models in the frbsPMML format are presented in Section 6. Finally, Section 7 concludes the paper.

2. Predictive Model Markup Language

This section explores PMML whose current version is 4.2 as of this writing. Firstly, an introduction to PMML and its benefits is presented. After that, we briefly illustrate its basic schema and components. Some applications allowing to produce and consume PMML are depicted to show that PMML is widely used in data-science areas.

2.1. Introduction to PMML

Due to the complexity of problems faced today, researchers and practitioners deal with them by proposing and using a wide variety of methods implemented in various software libraries. This phenomenon leads us to a situation where so many kinds of software are available for use. Next, when using various packages having different specification of input and output data, another problem arises, which is interoperability. The Institute of Electrical and Electronics Engineers (IEEE) defines interoperability as follows [26]: "The ability of two or more systems or components to exchange information and to use the information that has been exchanged." In other words, interoperability attempts to minimize any role of human to intervene of the models e.g., re-write, re-format, and transform. Thus, the efficiency can be achieved by researchers and practitioners. However, a challenge that should be addressed is that how we establish a common representation of models. The universal representation should be independent of programming languages and environment/platforms. Additionally, it has to have solid definitions and constraints in representing models so that ambiguities can be avoided.

PMML is a universal representation framework specified in an XML-based language that aims to provide interoperability of models produced by data mining and machine learning algorithms [20]. It is developed by the Data Mining Group (DMG, `http://www.dmg.org`), and based on the Extensible Markup Language (XML).

Figure 1 shows a PMML workflow, together with some advantages of PMML in data analysis processes. The workflow generally involves a modelling, expert intervention, and deploying phase. In the modelling phase, the final result is a model produced by learning methods according to given data. It may also involve data pre-processing and model validation. After the modelling, the model is exported to the PMML format, which is XML-based and human-readable. So, even though interpretability of the model mainly depends on the type of learning methods used, PMML helps at this end with readability and transparency. Therefore, human experts can relatively easy read, understand, and even modify the model and adapt it better to real-world conditions. In the final phase, we can also

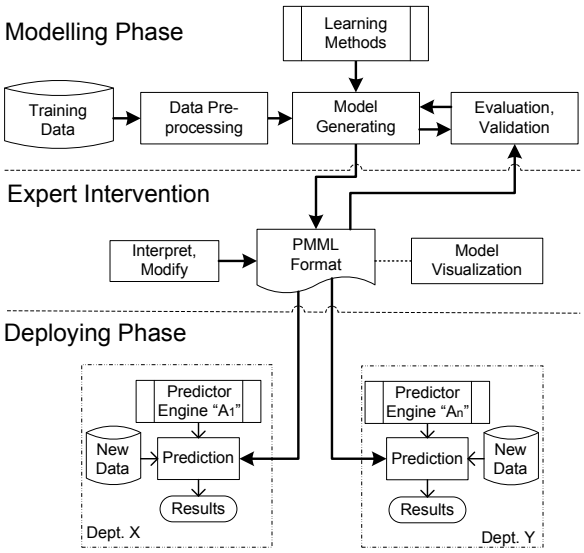

Figure 1: Workflow using PMML.

see several advantages. The model can be used in various predictor engines compliant with the PMML format to predict new data. In other words, with PMML it is easy to move the obtained models between various applications and platforms, so that it is easy to share them, e.g., across different departments. In addition, we note that prediction with new data in this phase is usually performed and repeated more frequently than the modelling in the first phase. Here, PMML helps to achieve a reproducible concept [19] since PMML provides a standard format that can be used anytime to predict new data by any compliant application.

There are some reasons why PMML is defined in the XML schema. First, XML is a standard language defined in the XML 1.0 specification by the World Wide Web Consortium (W3C) [27]. It provides a format that is both human- and machine-readable. There are many applications that use XML as their standard format, such as Microsoft Office and LibreOffice. To write a document based on XML, we need to consider definitions determined by a given schema, e.g., based on XML Schema [28]. It contains a basic grammar explaining the structure, content, and constraints of documents. Moreover, any new extensions made in the document have to be defined by using the XML schema.

In PMML, currently, there are available 16 models as follows:

- Association rules: representing rules showing relations between attributes;
- Baseline models: specifying change detection models;
- Cluster models: representing a set of clusters;
- General regression: allowing a multitude of regression models;
- k-nearest neighbors: representing a model of instance-based learning algorithms;
- Naïve Bayes: representing a model based on simple probabilistic classifiers according to Bayes' theorem;

- Neural networks: describing models based on artificial neural networks;
- Regression model: determining the relationship between dependent and independent attributes;
- Ruleset models: representing rules based on decision tree models;
- Scorecard models: describing models that map a set of inputs to predict a target value;
- Sequence rules: containing a set of rules for various items;
- Text models: providing a model used for text operations, such as frequency of terms;
- Time series models: providing time series analysis, such as forecasting;
- Tree models: providing a model represented by a tree for classification;
- Support vector machine (SVM): representing SVM models for classification and regression.

Additionally, PMML also provides schemata for data pre- and post-processing. For example, PMML defines normalization, discretization, value mapping, aggregation, etc.

2.2. The PMML Components and Schema

Since PMML is an XML-based language, the specification is defined by the XML Schema as recommended by the World Wide Web Consortium (W3C) [28]. The general schema and components of PMML can be seen in Listing 1. The PMML format is specified by the main tag *PMML* that contains some components. In the following, we describe the main components:

- *Header*: It contains general information about the PMML document, such as copyright information for the model, its description, application, and timestamp of generation.
- *DataDictionary*: It contains information related to fields or variables, such as number, names, types, and value ranges of variables.
- *MODEL-ELEMENT*: It is a main part of the PMML document that consists of models supported by PMML. In each model, there are several components embedded in the element, such as *MiningSchema* and *Output*. *MiningSchema* specifies outlier treatment, a missing value replacement policy, and missing value treatment, whereas *Output* shows a description of the output variable. For example, in a clustering model, we define a schema representing the cluster centers that are included in the *ClusteringModel* element.

Besides these components, there are some optional elements, such as *MiningBuildTask*, *TransformationDictionary*, and *Extension*. More detailed information about PMML can be found in [29].

```
<xs:element name="PMML">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Header"/>
      <xs:element ref="MiningBuildTask" minOccurs="0"/>
      <xs:element ref="DataDictionary"/>
      <xs:element ref="TransformationDictionary" minOccurs="0"/>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="MODEL-ELEMENT"/>
      </xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string"
        use="required"/>
  </xs:complexType>
</xs:element>

<xs:group name="MODEL-ELEMENT">
  <xs:choice>
    <xs:element ref="AssociationModel"/>
    ...
    <xs:element ref="TreeModel"/>
  </xs:choice>
</xs:group>
```

Listing 1: XML Schema for general components of PMML.

2.3. Implementations of PMML

In this section, we briefly review some applications implementing PMML. According to its functionalities, applications can be classified into two groups:

- PMML producer: It refers to a software that produces models, and exports/writes them to PMML format.
- PMML consumer: It refers to a software used for importing/reading and deploying PMML models to predict new data. In this software, there are procedures for validating and verifying the PMML format.

Nowadays, the PMML framework is implemented in several platforms. In the R environment, we can find the PMML-producer application "pmml" [20]. In order to generate models, the package executes several other packages available in R, such as "arules" for mining association rules, and "nnet" for neural networks. Next, the Konstanz Information Miner (KNIME), which is a platform for data integration, processing, analysis, and exploration [30], can be used both as a PMML producer and consumer [31]. SPSS provides a feature to import and export from/to PMML format [32]. The Waikato Environment for Knowledge Analysis (WEKA) allows to import PMML models based on regression, general regression, artificial neural networks, tree models, rule set models, and SVM models [33]. In order to provide further interoperability in delivering software solutions, PMML has been deployed in cloud computing using the Software-as-a-Service (SaaS) license model [21]. For example, it is embedded in the ADAPA scoring engine on the Amazon Web Services (AWS). A detailed table showing all software systems that implement the PMML standard can be found at `http://www.dmg.org/products.html`.

3. frbsPMML as a Universal Framework for Representing FRBSs

Just like the techniques and models mentioned in Section 2.1, FRBSs are frequently used in data analysis, modeling, and data mining. To further facilitate and promote their usage, we propose an extension of PMML for FRBSs, named frbsPMML. In this section, we describe its basic elements and the XML schemata for specifying an FRBS model in frbsPMML format.

Firstly, the main tag of frbsPMML is defined by *frbsPMML*. Then, the other extensions of PMML are made in the *MODEL-ELEMENT* part, while other components are still based on the existing PMML schema. Therefore, we

only discuss on the new components of the extension. The general schema specifying an FRBS model is described in Listing 2.

```
<xs:element name="FrbsModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:element ref="MiningSchema"/>
      <xs:element ref="Output" minOccurs="0"/>
      <xs:element ref="InferenceSchema"/>
      <xs:element ref="Database"/>
      <xs:element ref="Rulebase"/>
      <xs:element ref="ModelVerification" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="modelName" type="xs:string"
        use="required"/>
    <xs:attribute name="functionName" type="MINING-FUNCTION"
        use="optional"/>
    <xs:attribute name="algorithmName" type="xs:string"
        use="optional"/>
    <xs:attribute name="targetFieldName" type="xs:string"
        default="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 2: XML Schema for FRBS models.

It can be seen that the *FrbsModel* tag is required for representing an FRBS model. In the *FrbsModel*, there are two types of components: attribute and element. We define four attributes: *modelName*, *functionName*, *algorithmName*, and *targetFieldName*, where only *modelName* is required to be set. The *modelName* attribute refers to the type of FRBS model, i.e., MAMDANI, TSK, and FRBCS, for representing the Mamdani, Takagi Sugeno Kang, and fuzzy rule-based system model, respectively. In the elements, three components are important and emphasized, as follows: *InferenceSchema*, *Database*, and *Rulebase*.

InferenceSchema is a schema representing essential parameters in an FRBS model for inference/reasoning: conjunction, disjunction, implication, and aggregation operators. The XML schema of the tag *InferenceSchema* and its optional values can be seen in Listing 3. For example, the conjunction operators can be any of the following functions: *MIN*, *PRODUCT*, *HAMACHER*, *YAGER*, and *BOUNDED*. It should be noted that the parameters are defined as an optional components depending on the models. For instance, we need to set the *AggregationOperator* value if we use the Mamdani model.

```
<xs:element name="InferenceSchema">
  <xs:complexType>
    <xs:element ref="ConjunctionOperator" use="optional"/>
    <xs:element ref="DisjunctionOperator" use="optional"/>
    <xs:element ref="ImplicationOperator" use="optional"/>
    <xs:element ref="AggregationOperator" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="ConjunctionOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="MIN"/>
        <xs:enumeration value="PRODUCT"/>
        <xs:enumeration value="HAMACHER"/>
        <xs:enumeration value="YAGER"/>
        <xs:enumeration value="BOUNDED"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>

<xs:element name="DisjunctionOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="MAX"/>
        <xs:enumeration value="SUM"/>
        <xs:enumeration value="HAMACHER"/>
        <xs:enumeration value="YAGER"/>
        <xs:enumeration value="BOUNDED"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>

<xs:element name="ImplicationOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="DIENES_RESHER"/>
```

```
        <xs:enumeration value="LUKASIEWICZ"/>
        <xs:enumeration value="ZADEH"/>
        <xs:enumeration value="GOGUEN"/>
        <xs:enumeration value="SHARP"/>
        <xs:enumeration value="MIZUMOTO"/>
        <xs:enumeration value="DUBOIS_PRADE"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>

<xs:element name="AggregationOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="WAM"/>
        <xs:enumeration value="FIRST.MAX"/>
        <xs:enumeration value="LAST.MAX"/>
        <xs:enumeration value="MEAN.MAX"/>
        <xs:enumeration value="COG"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Listing 3: XML Schema for the *InferenceSchema* component.

The database is represented by the *Database* element, and it contains the following information:

- names of variables including the number of their linguistic values,
- types of membership functions, such as Gaussian, trapezoid, and triangular memberships,
- parameters of membership functions. For example, in the Gaussian membership function, the parameters are mean and variance.

The XML Schema of the *Database* is described in Listing 4. Basically, the *Database* contains *MembershipFunction* for each variable (i.e., inputs and outputs). The *MembershipFunction* consists of the element *FuzzyTerm* and two attributes: *name* and *numberOfLabels*. While *FuzzyTerm* represents databases containing linguistic values and their parameters, the *name* and *numberOfLabels* attributes express the variable name and the number of linguistic terms corresponding to each variable.

```
<xs:element name="Database">
  <xs:complexType>
    <xs:element ref="MembershipFunction"/>
  </xs:complexType>
</xs:element>

<xs:element name="MembershipFunction">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="FuzzyTerm"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="numberOfLabels" type="INT-NUMBER"
        use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="FuzzyTerm">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="type" use="required">
      <xs:complexType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="GAUSSIAN">
            <xs:element name="Parameters">
              <xs:element name="Mean" type="REAL-NUMBER"/>
              <xs:element name="Variance" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="TRAPEZOID">
            <xs:element name="Parameters">
              <xs:element name="Left" type="REAL-NUMBER"/>
              <xs:element name="LeftMiddle"
                  type="REAL-NUMBER"/>
              <xs:element name="RightMiddle"
                  type="REAL-NUMBER"/>
              <xs:element name="Right" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="TRIANGLE">
            <xs:element name="Parameters">
              <xs:element name="Left" type="REAL-NUMBER"/>
              <xs:element name="Middle" type="REAL-NUMBER"/>
              <xs:element name="Right" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="SIGMOID">
            <xs:element name="Parameters">
              <xs:element name="Gamma" type="REAL-NUMBER"/>
              <xs:element name="Distance" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="BELL">
            <xs:element name="Parameters">
```

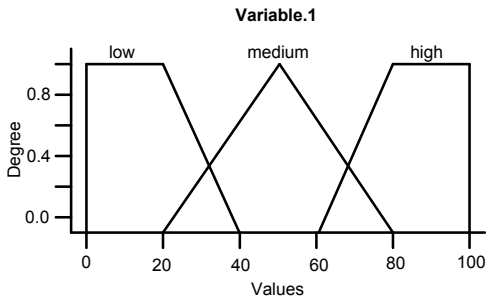

Figure 2: The membership functions of "Variable.1".

```
      <xs:element name="Width" type="REAL-NUMBER"/>
      <xs:element name="Power" type="REAL-NUMBER"/>
      <xs:element name="Center" type="REAL-NUMBER"/>
    </xs:element>
  </xs:enumeration>
</xs:restriction>
    </xs:complexType>
  </xs:attribute>
</xs:complexType>
</xs:element>
```

Listing 4: XML Schema for the *Database* component.

According to the *FuzzyTerm* schema, we provide five types of membership functions:

1. *GAUSSIAN*: In this case, we need to define two elements: *Mean* and *Variance* representing mean and variance of the Gaussian function.
2. *TRAPEZOID*: We supply four components *Left*, *LeftMiddle*, *RightMiddle*, and *Right* for representing the corner points.
3. *TRIANGLE*: It has three parameters: *Left*, *Middle*, and *Right* that represent the corner points.
4. *SIGMOID*: There are two parameters: *Gamma* and *Distance*, representing steepness of the function, and distance from the origin, respectively.
5. *BELL*: Three parameters need to be defined in *BELL*: *Width*, *Power*, and *Center*, which determine the width of the curve, a positive number for the power, and the center of the curve.

Furthermore, it is possible to define different membership functions and numbers of linguistic values for the variables. We can also assign different numbers of linguistic values for other variables. For instance, "Variable.1" has 3 linguistic values which are "low," "medium," and "high." To determine the degree we define that "medium" has *TRIANGLE* and the rest have *TRAPEZOID* memberships as in Figure 2. This example can be specified in frbsPMML format as in Listing 5.

```
<Database>
<MembershipFunction name="Variable.1"
  numberOfLabels="3">
 <FuzzyTerm name="low" type="TRAPEZOID">
  <Parameters>
   <Left>0</Left>
   <LeftMiddle>0</LeftMiddle>
   <RightMiddle>20</RightMiddle>
   <Right>40</Right>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="medium" type="TRIANGLE">
  <Parameters>
   <Left>20</Left>
   <Middle>50</Middle>
   <Right>80</Right>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="high" type="TRAPEZOID">
  <Parameters>
   <Left>60</Left>
   <LeftMiddle>80</LeftMiddle>
```

```
      <RightMiddle>100</RightMiddle>
      <Right>100</Right>
    </Parameters>
  </FuzzyTerm>
</MembershipFunction>
</Database>
```

Listing 5: The *Database* schema of "Variable.1".

Finally, Listing 6 describes the XML Schema of the *Rulebase* consisting of the element *Rule* and the attribute *numberOfRules*. *Rule* specifies a set of rules whereas *numberOfRules* shows the number of rules used for validation.

```
<xs:element name="Rulebase">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Rule" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="numberOfRules" type="INT-NUMBER" use="required"/>
  </xs:complexType>
</xs:element>
```

Listing 6: XML Schema for the *Rulebase* component.

As mentioned before, FRBS models can be classified into two popular models: Mamdani and TSK, used for dealing with regression problems. Additionally, FRBCS is suitable for classification tasks. Since the difference of models is determined by the representations of rules, we explain the components of the *Rulebase* in the following.

3.1. The Mamdani Model

This model was introduced by Mamdani in [3, 4]. It is built by linguistic variables in both the antecedent and consequent parts of the rules. So, considering multi-input and single-output (MISO) systems, fuzzy IF-THEN rules are of the following form:

$$\textbf{IF } X_1 \text{ is } A_1 \text{ and } \ldots \text{ and } X_n \text{ is } A_n \textbf{ THEN } Y \text{ is } B \tag{1}$$

Here, X_i and Y are input and output linguistic variables, respectively, while A_i and B are linguistic values, e.g., "hot," "medium," and "cold."

Generally, a rule represented by Equation 1 can be specified by the XML Schema as in Listing 7. It contains two elements: *If* and *Then*, used for expressing the antecedent and consequence parts.

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:element name="If">
      <xs:complexType>
        <xs:element ref="CompoundPredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Then">
      <xs:complexType>
        <xs:element ref="SimplePredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="id" type="INT-NUMBER"
        use="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 7: XML Schema for the *Rule* component based on the Mamdani model.

The *If* part includes the *CompoundPredicate* component, whose XML Schema is shown in Listing 8. Basically, *CompoundPredicate* consists of *SimplePredicate* together with the attribute *booleanOperator* to construct the antecedent part recursively. The *SimplePredicate* element is built from two components: *field* and *value*. The *field*

attribute expresses a variable name whereas *value* is a linguistic value. The attribute *booleanOperator* expresses the logic operators (i.e., *and* and *or*). Furthermore, since we assume that the model is MISO, the *Then* part contains a single *SimplePredicate*.

```
<xs:element name="CompoundPredicate">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="SimplePredicate"/>
      <xs:element ref="CompoundPredicate"/>
    </xs:sequence>
    <xs:attribute name="booleanOperator" use="required">
      <xs:simpleType>
        <xs:enumeration value="and"/>
        <xs:enumeration value="or"/>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="SimplePredicate">
  <xs:simpleType>
    <xs:attribute> name="field" type="xs:string" use="required"/>
    <xs:attribute> name="value" type="xs:string" use="required"/>
  </xs:simpleType>
</xs:element>
```

Listing 8: XML Schema for the *CompoundPredicate* and *SimplePredicate* components.

For example, the rule 2 is documented in frbsPMML as in Listing 9.

$$\textbf{IF } X1 \text{ is } \textit{normal} \text{ and } X2 \text{ is } \textit{tall} \text{ and } X3 \text{ is } \textit{small} \textbf{ THEN } Y \text{ is } \textit{good} \tag{2}$$

```
<Rulebase numberOfRules="1">
  <Rule id="1">
    <If>
      <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="X1" value="normal"/>
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="X2" value="tall"/>
          <SimplePredicate field="X3" value="small"/>
        </CompoundPredicate>
      </CompoundPredicate>
    </If>
    <Then>
      <SimplePredicate field="Y" value="good"/>
    </Then>
  </Rule>
</Rulebase>
```

Listing 9: The *Rulebase* schema of the example based on the Mamdani model.

A main benefit of the schema is that it is rather flexible. For example, we can define different values of *booleanOperator* for each predicate. Additionally, this schema allows us to put the negation operator (i.e., *not*) and linguistic hedges (e.g., *very*, *somewhat*, etc.) in the *value* attribute. Furthermore, it is not necessary to involve all input variables for the construction of each rule. In other words, the length of *SimplePredicate* in each rule can be different. We can also set the *dont_care* value which represents a variable whose degree of membership is 1 for all conditions.

3.2. The TSK Model

The difference of the TSK model from the Mamdani model is on the consequent part. TSK uses rules whose consequent parts are represented by a function of input variables instead of using linguistic variables [5, 6]. The most commonly used function is a linear combination of the input variables: $Y = f(X_1, \ldots, X_n)$ where X_i and Y are the input and output variables, respectively. Therefore, we can express it as $Y = p_1 \cdot X_1 + \cdots + p_n \cdot X_n + p_0$ with a vector of real parameters $p = (p_0, p_1, \ldots, p_n)$.

We define the XML Schema of *Rule* based on TSK as in Listing 10.

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:element name="If">
      <xs:complexType>
        <xs:element ref="CompoundPredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Then">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Coefficient" minOccurs="0"/>
        </xs:sequence>
        <xs:element ref="Constant" minOccurs="1"/>
        <xs:attribute name="type" default="LinearFunction">
          <xs:simpleType>
            <xs:enumeration value="LinearFunction"/>
            <xs:enumeration value="NonLinearFunction"/>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="id" type="INT-NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 10: XML Schema for the *Rule* component based on the TSK model.

It can be seen that on the antecedent part (i.e., the *If* element) we have the same schema as in the Mamdani model, but in the *Then* block, we define two components: *Coefficient* and *Constant*. The XML Schema of both components is described in Listing 11. In order to construct a linear function, the attribute *Coefficient* represents coefficient values of each variable while *Constant* is the constant value of the equation. Using the specification, we allow to define first- and zero-order TSK.

```
<xs:element name="Coefficient">
  <xs:simpleType>
    <xs:attribute name="field" type="xs:string" use="required"/>
    <xs:attribute name="value" type="REAL-NUMBER" use="required"/>
  </xs.simpleType>
</xs:element>

<xs:element name="Constant">
  <xs:simpleType>
    <xs:attribute name="value" type="REAL-NUMBER"
        use="required">
  </xs:simpleType>
</xs:element>
```

Listing 11: XML Schema for the *Coefficient* and *Constant* components.

For example, a rule as in 3 can be specified in Listing 12.

$$\textbf{IF } X1 \text{ is } normal \text{ and } X2 \text{ is } tall \text{ and } X3 \text{ is } small \textbf{ THEN } Y = 0.2 \cdot X_1 + 0.1 \cdot X_2 - 0.2 \cdot X_3 + 0.9 \tag{3}$$

```
<Rulebase numberOfRules="1">
  <Rule id="1">
    <If>
      <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="X1" value="normal"/>
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="X2" value="tall"/>
          <SimplePredicate field="X3" value="small"/>
        </CompoundPredicate>
      </CompoundPredicate>
    </If>
    <Then type="LinearFunction">
      <Coefficient field="X1" value="0.2"/>
      <Coefficient field="X2" value="0.1"/>
      <Coefficient field="X3" value="-0.2"/>
      <Constant value="0.9"/>
    </Then>
  </Rule>
</Rulebase>
```

Listing 12: The *Rulebase* schema of the example based on the TSK model.

3.3. The FRBCS Models

A main characteristic of classification is that the outputs are categorical data. Therefore, in this model type we preserve the antecedent part of linguistic variables, and change the consequent part to be a class C_j from a prespecified class set $C = \{C_1, \ldots, C_M\}$. Generally, there are three structures for representing FRBCS. First, the

simplest form introduced by [7] is constructed with a class in the consequent part. Then, the FRBCS model with a certainty degree (called weight) in the consequent part is discussed in [8]. In [9], every fuzzy rule has with a certainty degree for all classes in the consequent part. In other words, instead of considering one class, this model provides prespecified classes with their respective weights for each rule. In this paper, we consider the second type.

Listing 13 shows the schema of *Rule* for FRBCS. We note that it is quite similar to the Mamdani model in Listing 7, but in the *Then* part we have categorical values instead of linguistic ones. Additionally, there is a component *Grade* representing a degree of the certainty of each rule that has a value between 0 and 1.

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:element name="If">
      <xs:complexType>
        <xs:element ref="CompoundPredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Then">
      <xs:complexType>
        <xs:element ref="SimplePredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Grade" value="REAL-NUMBER"/>
    <xs:attribute name="id" type="INT-NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 13: XML Schema for the *Rule* component based on the FRBCS model.

For example, we define a rule as in 4, where w is its grade. In the frbsPMML format, it can be specified as in Listing 14.

$$\textbf{IF } X1 \text{ is } \textit{normal} \text{ and } X2 \text{ is } \textit{tall} \text{ and } X3 \text{ is } \textit{small } \textbf{THEN } \textit{class} \text{ is } 1 \text{ with } w = 0.1. \tag{4}$$

```
<Rulebase numberOfRules="1">
  <Rule id="1">
    <If>
      <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="X1" value="normal"/>
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="X2" value="tall"/>
          <SimplePredicate field="X3" value="small"/>
        </CompoundPredicate>
      </CompoundPredicate>
    </If>
    <Then>
      <SimplePredicate field="Class" value="1"/>
    </Then>
    <Grade>0.1</Grade>
  </Rule>
</Rulebase>
```

Listing 14: The *Rulebase* schema of the example based on the FRBCS model.

4. Implementation of frbsPMML

The frbsPMML format described above is a complete specification for representing the most commonly used model types. As most XML representations, it is designed to be complete and exhaustive, and usually considered for manual editing. Thus, we present two libraries of software for managing the frbsPMML representation. They are published under an open-source license, hence available freely, for use, adaption, and extension.

The two libraries represented in the following are called "frbs" and "frbsJpmml." They can be used to export an FRBS model to the frbsPMML format and vice versa. The general workflow of the applications to generate an FRBS model and perform prediction for new data can be seen in Figure 3.

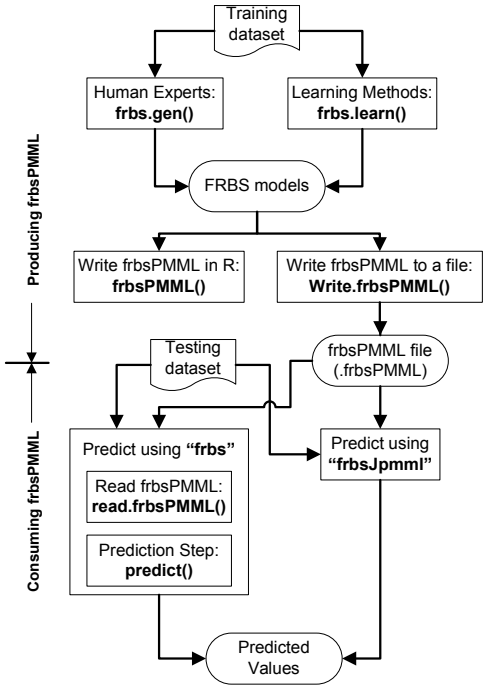

Figure 3: Workflow and interactions between "frbs" and "frbsJpmml".

4.1. The "frbs" Package

The "frbs" software is an open-source R package that provides prominent FRBS models and implements widely used learning procedures in FRBSs [14]. It can be considered the standard package for frbs in R and included in the Comprehensive R Archive Network (CRAN) Task View for machine learning. It is available at `http://cran.r-project.org/package=frbs`.

R is a widely used analysis environment for scientific computing and visualization, Statistics, Data Mining, Bioinformatics, and Machine Learning [12, 13]. Currently, over 6000 packages are included in CRAN. It is the standard internet repository of packages written for R, and every package submitted into the repository is checked to meet certain quality standards.

Regarding learning approaches to construct FRBS models from data, "frbs" classifies them into five groups: space partition, clustering, gradient descent, genetic algorithms, and neural networks. For example, in the group of space partition we consider Wang and Mendel's technique (WM) [34] and the FRBCS using Chi's method (FRBCS.CHI) [7]. Based on neural networks, there are the adaptive-network-based fuzzy inference system (ANFIS) [35] and the hybrid neural fuzzy inference system (HYFIS) [36]. Combination of FRBSs and genetic algorithms, called the genetic fuzzy systems (GFS) [37, 38], are implemented in the package in form of the following algorithms: the genetic fuzzy systems based on Thrift's method (GFS.Thrift) [39], the genetic fuzzy systems for fuzzy rule learning based on the MOGUL methodology (GFS.FR.MOGUL) [40], Ishibuchi's method based on the genetic cooperative competitive learning (GFS.GCCL) [41], Ishibuchi's method based on the hybridization of genetic cooperative competitive learning (GCCL) and Pittsburgh (FH.GBML) [42]. Moreover, the following algorithms based on FRBSs employing clustering

methods are included in the package: the subtractive clustering (SBC) [43] and the dynamic evolving neural fuzzy inference system (DENFIS) [44]. Finally, the package contains two simple algorithms using gradient descent to optimize parameters: the fuzzy inference rules with descent method (FIR.DM) [45] and the FRBS using heuristics and the gradient descent method (FS.HGD) [46].

Besides providing many learning methods, the "frbs" package offers many other functionalities for constructing an FRBS models. First, the package implements various choices for triangular norm (t-norm), s-norm, implicator functions, defuzzification methods, and membership functions. For example, we consider minimum, Hamacher, Yager, product, and bounded product as t-norm operators. For performing fuzzification, which is a process for determining a degree of membership, we consider triangular, trapezoid, Gaussian, sigmoid, and general bell membership functions. Moreover, even though we focus on constructing FRBS models by learning from data, we facilitate building FRBS models manually from human expert knowledge. Also, to obtain a representative model, experts can define linguistic hedges. Moreover, the "don't care" value can be assigned to be a lingustic value representing a value that always has the degree of 1 so that we can minimize the complexity of the rules.

Table 1 shows the main functions in the package, where the last three are functions designated for managing the frbsPMML format. First, there are two functions that are used for constructing models: *frbs.learn()* and *frbs.gen()*. Then, the two functions *frbsPMML()* and *write.frbsPMML()* are used for converting FRBS models to frbsPMML. Finally, to obtain prediction for new data, there is the function *predict()*. Two additional functions: *summary()* and *plot.MF()*, are used to display an FRBS model in the R environment and plot membership functions, respectively.

Table 1: The main functions of the "frbs" package.

Functions	Description
frbs.learn()	It is a main function used to construct an FRBS model automatically from data.
predict()	It performs fuzzy reasoning to obtain predicted values for new data, using a given FRBS model.
frbs.gen()	It is used to construct an FRBS model manually from expert knowledge.
summary()	It is used to show a summary of an FRBS model.
plotMF()	It is used to plot the membership functions.
frbsPMML()	It is a main function used to convert a model to the frbsPMML format.
read.frbsPMML()	It is used to read and convert a model in frbsPMML format to an R object.
write.frbsPMML()	It is used to write and save a model in frbsPMML format to a file.

The following are signatures of the functions related to frbsPMML:

- *frbsPMML()*: Though the function has several arguments, usually only the *model* parameter needs to be supplied which refers to the FRBS model.

```
frbsPMML(model, model.name = "frbs_model", app.name = "frbs",
  description = NULL, copyright = NULL, algorithm.name = model$method.type, ...)
```

- *read.frbsPMML()*: The function has as its only parameter the name of the file to read.

```
read.frbsPMML(fileName)
```

- *write.frbsPMML()*: There are two required parameters: *object* and *fileName*. *object* represents the FRBS model in R format whereas *fileName* is the name of the file where the model will be written to.

```
write.frbsPMML(object, fileName = NULL)
```

For signatures of the other functions from the package we refer to [14].

So, as illustrated in Figure 3, we can use "frbs" both as an frbsPMML producer and a consumer with the following steps:

1. Construct an FRBS model: this can be done by executing *frbs.learn()* or *frbs.gen()*.
2. Export the model to frbsPMML format: we call *write.frbsPMML()* to save the model to a file or *frbsPMML()* to store the model in frbsPMML format in an R object. Obviously, after obtaining the model in frbsPMML format, we can also modify directly the file.
3. Import the FRBS model in frbsPMML format to an R object: we execute *read.frbsPMML()*.
4. Perform prediction for new data with *predict()*.

4.2. The "frbsJpmml" Package

This frbsPMML consumer application is implemented in Java and can be used to make predictions from FRBS models that are available in the frbsPMML format. It is designed in compliance with "frbs," and provides the standard functionalities for constructing an FRBS model.

Basically, "frbsJpmml" consists of four parts, as follows:

- *DataReader*: It is a package containing classes for reading new data and saving results into files.
- *FRBSEngine*: It consists of classes representing the FRBS models. There are three child classes of the *frbsModel* class representing the models: *MamdaniModel*, *TSKModel*, and *FRBCSModel*. Additionally, in the parent class *frbsModel* we include the *fuzzifier* and *Inference* which are methods used for fuzzifying data and reasoning, respectively. *predict*, an abstract method for prediction, is included in this part as well.
- *PMMLreader*: It is a package used for reading/importing FRBS models in the frbsPMML format to Java objects. A verification procedure of the obtained model is also included in this part.

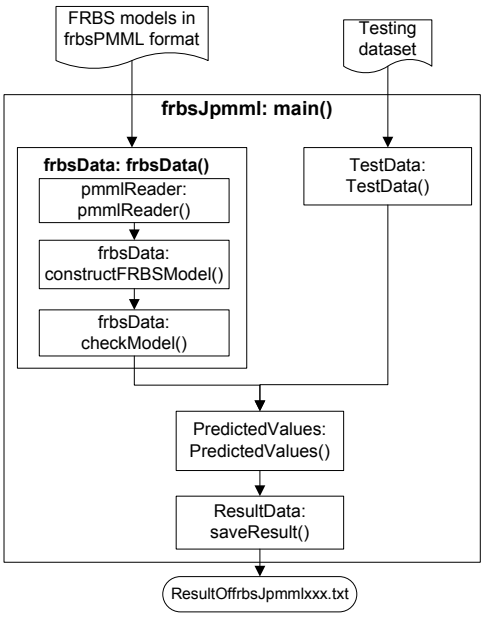

Figure 4: Classes and their methods involved to predict new data.

- *MainIOfrbs*: It is the main package that includes the class *frbsJpmml* and has the *main* method. So, it is a user interface to work with the package.

A global description explaining classes and their methods involved to predict new data can be seen in Figure 4.

Two arguments have to be supplied: an FRBS model in frbsPMML format saved into a file with the extension *frbsPMML* and a text file containing testing data. To use "frbsJpmml," the following needs to be executed on the command line:

```
java -jar frbsJpmml.jar <pmmlFileName> <tstData>
```

Here, *pmmlFileName* and *tstData* are file names containing the model in frbsPMML format and the testing dataset, respectively. Then, the output is a *txt* file that has the prefix "Result." "frbsJpmml" also produces a log file, namely *pmmlReaderLog.txt*. The code, executable file, and its detailed description explaining how to use the package can be found at `http://sci2s.ugr.es/dicits/software/frbsJpmml`.

5. Features of the New Representation

This section aims to recapitulate all features of the presented framework and their benefits of implementations for researchers and practitioners. Moreover, a short comparison with representations from other applications is presented.

We discuss features included in the new representation from the following perspectives: completeness of FRBS models and expressiveness of the language.

Regarding FRBS models included in the representation we consider three models: Mamdani, TSK, and FRBCS, with the following detailed specifications:

- In the *InferenceSchema* part, we provide complete parameters, such as conjunction, disjunction, aggregation, and implication operators. Furthermore, each operator has several options representing different approaches.
- The *Database* component supports five membership functions: Gaussian, triangle, trapezoid, sigmoid, and generalized bell. Besides parameters of the membership functions being defined in an easy way, the main benefit is that we allow to set different numbers of labels and membership functions in a particular variable.
- The *Rulebase* used to represent rule-based knowledge has several useful features. Firstly, we can mix boolean operators, i.e., *and* and *or*, together in one rule. It is not necessary to involve all variables in each rule. In other words, the interpretability of rules is favored by the representation. Linguistic hedges can be also included together with fuzzy terms in the *value* element. Furthermore, for the TSK model, the representation allows to use first- and zero-order TSK. For dealing with classification tasks, we provide the FRBCS representation that involves a degree component for each rule.
- The XML Schema frbsPMML is compatible with PMML that is an established industry standard. Furthermore, as included in PMML, several facilitations for data analysis are available as well, such as data transformation, missing values completion, etc. Additionally, descriptions of data are included in the *DataDictionary* element containing information about names, ranges, and types of variables.

From a language point of view, the new representation offers several advantages as follows:

- In the *Rulebase*, each rule is constructed in a recursive way, and it contains two components: *SimplePredicate* and *CompoundPredicate*. This means it has a sophisticated structure which makes rule reduction and extension a straightforward task. In fact, the expression represents a mathematical formulation of rules.
- The XML schema used to specify the representation provides transparency and readability of documents. Therefore, it is easy for users to read, understand, and modify the documents. Furthermore, since we develop an open standard, other researchers can constribute.
- An FRBS model represented in frbsPMML has a text-based representation. So, human experts can read it without problems. It can also be easily archived and transferred to other platforms. Moreover, further deployment is possible, e.g., for cloud computing applications.
- The representation provides some validity and verification components, such as *numberOfRules* and *numberOfLabels* are used to validate the number of rules and the number of linguistic values.

From the interpretability perspective, at least there are two important studies in [25] and [47]. While the paper [25] defines that the interpretability depends on the model structure, the number of input variables, the number of fuzzy rules, the number of linguistic terms, and the shape of the fuzzy sets; the second research provides two categories: low-level interpretability (i.e., by optimizing membership functions and fuzzy set level) and high-level

interpretability (i.e., a compact and consistent rulebase). So, it can be seen that frbsPMML helps the interpretability of FRBS models by providing a readable standard format.

Table 2 shows a comparison of the proposed format with others. We consider four formats: XFL3 [10], *.fis* (MATLAB) [11], XFSML [48], and "FisPro" [15].

XFL3 is a formal language representing fuzzy systems that is implemented by "Xfuzzy". It consists of two parts: the logical definition of the system structure and the mathematical definition of the fuzzy functions. Basically, an FRBS model is specified in a function-based format. Therefore, for common users a complex model in this format can be difficult to read and understand. Additionally, there exist several membership functions, operators, hedges, and defuzzification methods.

Next, XFSML is an XML-based language for modelling fuzzy systems. It contains four components: domains, partitions, relations, and modules. One main drawback of this representation is that rulesets are expressed in a relatively complicated manner. Furthermore, though it attempts to be a standard modeling language in the fuzzy community, to the best of our knowledge it is not implemented by any applications, and it is not documented in a formal schema of XML.

In the MATLAB environment, the Fuzzy Logic Toolbox has the proprietary *.fis* format. Since the format is not open, it does not facilitate interoperability. The same holds for "FisPro."

Another work that is similar to and improved by our research is the Fuzzy Markup Language (FML). It is an emerging XML-based markup language used for designing and implementing fuzzy controllers (FLC) [49, 50]. Two models are supported in this representation: Mamdani and TSK. An interface connecting to the Matlab Fuzzy Logic Toolbox is provided, as well as Extensible Stylesheet Language Transformations (XSLTs) that are used to convert the FML fuzzy controller to a representation in a general purpose computer language. Even though FML is quite similar to the proposed representation in this paper, we improve and extend some aspects. For example, FML is not designed to accommodate a rule containing mixed operators (i.e., "and" and "or"). This issue is resolved by frbsPMML since it constructs a fuzzy rule in a recursive way. Secondly, since FML is used for representing FCL, it does not provide other typical components included in data mining, such as data pre-processing, missing value handling, etc. Since frbsPMML adopts the schema of PMML, we have the same capabilities as PMML for dealing with data mining processes. Another drawback of FML that attempts to be refined by frbsPMML is that no formal definition of an XML Schema is provided. So, it is relatively difficult to extend the format.

In addition to a standard representation, the study in [51] proposes a representation based on the unified modeling language (UML), called the evolutionary computing modeling language (ECML). It focuses on representing the concepts of the meta evolutionary computation domain. There is a significant drawback of the representation, which is that the graphical schema can be difficult to be understood and processed when ECML expresses a big model. In [52] rule-based representations based on the Resource Description Framework and Ontology Representation Languages (RDFS and OWL) have been proposed. The format is designed so that it easily allows to supply it to database

management systems, such as the Oracle RDBMS. Since these studies are not related to fuzzy sets, we do not include them in Table 2.

Table 2: Comparison with other representations

Components	frbsPMML	XFL3	.fis (MATLAB)	XFSML	FisPro	FML
General						
Open standard	Yes	No	No	Yes	No	No
Implementations	"frbs", "frbsJpmml"	"Xfuzzy"	"Fuzzy Logic Toolbox"	-	"FisPro", "GUAJE"	"Fuzzy Logic Toolbox"
Other features	Data mining methods (e.g., neural networks, association rules, etc.), Data preprocessing (e.g., transformations, missing value completion)	Support for Java, C, C++, VHDL, and SysGen	-	-	-	-
Completeness of FRBS models						
Models	Mamdani, TSK, FRBCS	Mamdani, TSK	Mamdani, TSK	Mamdani, Fuzzy decision trees,	Mamdani	Mamdani, TSK
Inference parameters	Many options	Many options	Many options	Not specified	Many Options	Not specified
Membership functions	Many options	Many options	Many options	Not specified	Many Options	Not specified
Hedges	Supported	Supported	Supported	-	Supported	Supported
Interpretable rules	Supported	Supported	-	-	Supported	Supported
Operators: AND, OR, NOT	Supported	Supported	Supported	-	Supported	Supported (not mixed)
Expressiveness of languages						
Format base	Text (XML)	Function, GUI	Object, GUI	Text (XML)	GUI	Text (XML)
Interoperability	High	Medium	Low	High	Low	High
Validity and verification components	Provided	-	-	-	-	-
Readability	High	Medium	High	Medium	High	High
Ease of extension	High	High	Low	High	Low	Medium

Therefore, according to the features and their benefits, it can be seen that the new representation should be considered as an open standard for representing FRBS models by researchers and practitioners. Since it is an open standard based on XML, other developers and researchers in the fuzzy community can adopt it in any applications and can propose enhancements, e.g., in form of definitions of XML schemata to accommodate complicated models.

6. Usage Examples

In this section, we consider two examples for handling regression and classification tasks. The examples demonstrate how to construct an FRBS model and export/import FRBS models to/from the proposed format using "frbs" and "frbsJpmml". The examples only briefly discuss model construction. For more detailed explanations, the reader may refer to [14]. We note that the following examples are executed in the R environment. The complete scripts are, together with other material, available at `http://sci2s.ugr.es/dicits/software/frbsJpmml`.

6.1. Regression

In this section, we describe how to use the "frbs" package to predict real-valued output based on the input variables expressed by a continuous function called the *four hill* function:

$$f(x, y) = \frac{1}{x^4 + y^4 - 2x^2 - 2y^2 + 3}$$

It involves two input variables $x \in [-2, 2]$ and $y \in [-2, 2]$.

In this example, we build an FRBS model based on the Mamdani model. Let us assume we have a dataset split into a training and a test set available in R, in data frames called *data.tra* and *data.tst*.

To construct an FRBS model from the training data, we need to assign values to available parameters, e.g., as follows:

```
R> method.type <- "WM"
R> control <- list(num.labels = 5, type.mf = "GAUSSIAN", type.defuz = "WAM",
+   type.tnorm = "MIN", type.implication.func = "LUKASIEWICZ", name = "fourhill")
```

Then, we can execute the following function for constructing the model (where *range.data* represents the range of the data):

```
R> mod.reg <- frbs.learn(data.tra, range.data, method.type, control)
```

The FRBS model that we obtain in this way, called *mod.reg*, contains matrices with the database, rulebase, and the method parameters.

6.1.1. Exporting to frbsPMML format using "frbs"

After obtaining the model, we can construct and save the model in frbsPMML format to a file with extension *frbsPMML* using "frbs" as follows:

```
R> write.frbsPMML(frbsPMML(mod.reg), "modRegress")

[1] "modRegress.frbsPMML"
```

We can also export to frbsPMML format but with storing it within R in the memory or directly displaying it as follows:

```
R> frbsPMML(mod.reg)
```

Listing 15 shows the standard components of frbsPMML: the *Header* and *DataDictionary*. The *Header* contains metadata such as copyright, description of the simulation, application, and timestamp. In the *DataDictionary*, there are the following parameters:

- *numberOfFields*: It refers to the number of variables/fields. In this case, we have 3 variables.
- *DataField*: It shows a description of each variable such as names of variables (*name*), data types (*dataType*), and their intervals (*Interval*).

```
<frbsPMML version="1.0" ...>
 <Header copyright="Copyright (c) 2014" description="The fourhill was
     resulted by WM method based on MAMDANI model">
  <Extension name="user" value="Lala" extender="frbs"/>
  <Application name="frbs" version="3.0-0"/>
  <Timestamp>2014-07-04 17:45:16</Timestamp>
 </Header>
 <DataDictionary numberOfFields="3">
```

```
<DataField name="X" optype="continuous" dataType="double">
 <Interval closure="closedClosed" leftMargin="-2"
     rightMargin="1.92"/>
</DataField>
<DataField name="Y" optype="continuous" dataType="double">
 <Interval closure="closedClosed" leftMargin="-2"
     rightMargin="1.92"/>
</DataField>
<DataField name="Z" optype="continuous" dataType="double">
 <Interval closure="closedClosed" leftMargin="0.05263157894"
     rightMargin="0.9967463011"/>
</DataField>
</DataDictionary>
```

Listing 15: XML Schema for *Header* and *DataDictionary* of the regression example.

The main components of the model can be seen in Listing 16 i.e., the *InferenceSchema*, the *Database*, and the *Rulebase*. The *InferenceSchema* specifies the user-assigned parameters for inference/reasoning. In the *Database* tag, in the case of variable "X", it has the following components:

- *numberOfLabels*: It refers to the number of linguistic values, which is 5.
- *FuzzyTerm*: It refers to a description regarding each linguistic value and its membership function. For example, the linguistic value "very.small" has Gaussian for its membership function.
- *Parameters*: It refers to parameters related to the membership function of each linguistic value. Since the linguistic value "very.small" has a Gaussian membership function, here we have the parameters *Mean* and *Variance*.

Finally, regarding the *Rulebase*, it can be seen that 47 rules have been generated. Due to the limited space we do not show the full model here. It is available from our webpage.

```
<FrbsModel modelName="MAMDANI" functionName="regression"
    algorithmName="WM" targetFieldName="Z">
 <MiningSchema>
  <MiningField name="X" usageType="active"/>
  <MiningField name="Y" usageType="active"/>
  <MiningField name="Z" usageType="predicted"/>
 </MiningSchema>
 <Output>
  <OutputField name="Predicted_Z" optype="continuous"
      dataType="double" feature="predictedValue"/>
 </Output>
 <InferenceSchema>
  <ConjunctionOperator value="MIN"/>
  <DisjunctionOperator value="MAX"/>
  <ImplicationOperator value="LUKASIEWICZ"/>
  <AggregationOperator value="WAM"/>
 </InferenceSchema>
 <Database>
  <MembershipFunction name="X" numberOfLabels="5">
   <FuzzyTerm name="very.small" type="GAUSSIAN">
    <Parameters>
     <Mean>0</Mean>
     <Variance>0.0875</Variance>
    </Parameters>
   </FuzzyTerm>
   ...
   <FuzzyTerm name="very.large" type="GAUSSIAN">
    <Parameters>
     <Mean>1</Mean>
     <Variance>0.0875</Variance>
    </Parameters>
   </FuzzyTerm>
  </MembershipFunction>
 </Database>
 <Rulebase numberOfRules="47">
  <Rule id="1">
   <If>
    <CompoundPredicate booleanOperator="and">
     <SimplePredicate field="X" value="very.small"/>
     <SimplePredicate field="Y" value="very.small"/>
    </CompoundPredicate>
   </If>
   <Then>
    <SimplePredicate field="Z" value="very.small"/>
   </Then>
  </Rule>
  ...
  <Rule id="47">
   <If>
    <CompoundPredicate booleanOperator="and">
     <SimplePredicate field="X" value="medium"/>
     <SimplePredicate field="Y" value="large"/>
    </CompoundPredicate>
   </If>
   <Then>
    <SimplePredicate field="Z" value="small"/>
   </Then>
  </Rule>
 </Rulebase>
</FrbsModel>
```

</frbsPMML>

Listing 16: XML Schema for the *FrbsModel* component of the regression case.

6.1.2. Importing from the proposed format using "frbs" and "frbsJpmml"

In this example, firstly we use "frbs" to import and apply the model from the file *modRegress.frbsPMML* by the following command:

```
R> objReg <- read.frbsPMML("modRegress.frbsPMML")
R> res.test <- predict(objReg, data.tst)

[1] "note: Some of your new data are out of the previously specified range"
[1] "note: Some of your new data are out of the previously specified range"
```

We note that in this case *data.tst* is new data in the R environment. The predicted values are saved as a matrix. They can be compared with the actual values using, e.g., the mean squared error (MSE) by

```
R> err.MSE <- mean((real.val - res.test)^2)
R> print(err.MSE)

[1] 0.009978245
```

Secondly, we perform prediction for new data according to the FRBS model "modRegress.frbsPMML" using "frbsJpmml" by the following command:

```
java -jar frbsJpmml.jar "modRegress.frbsPMML" "newdataReg.txt"
```

It should be noted that in this case the new data for testing are saved in the file *newdataReg.txt*.

Furthermore, other examples of regression problems are available on our web page.

6.2. Classification

In this example, we consider a classification problem. The *iris* dataset is well-known in the pattern recognition literature. It contains 3 classes of 50 instances each, where each class refers to a type of iris plant. We are to build an FRBCS to solve it. Let us assume the data are already available in the variable *iris*. They are then divided into two datasets: *tra.iris* and *tst.iris*, used for training and testing. The detailed script can be found on our web page.

As in the regression example, we need to define some parameters concerning the used method and its *control* parameter. For example:

```
R> method.type <- "GFS.GCCL"
R> control <- list(popu.size = 30, num.class = 3, num.labels = 3,
+   persen_cross = 0.9, max.gen = 200, persen_mutant = 0.3, name="sim-Iris")
```

We generate an FRBS model through the following command:

```
R> mod.class <- frbs.learn(tra.iris, range.data.input, method.type, control)
```

It should be noted that *range.data.input* is a matrix containing intervals of each input variables.

6.2.1. Exporting to frbsPMML format using "frbs"

After obtaining the model, we can save it in the proposed format to the file *modClass.frbsPMML* as follows:

```
R> write.frbsPMML(frbsPMML(mod.class), "modClass")

[1] "modClass.frbsPMML"
```

We can also write to frbsPMML format, and then display it in the R environment by the following command:

```
R> frbsPMML(mod.class)
```

The header and main components are shown in Listing 17. Basically, the schemata of the header are similar to the previous example. However, since we have categorical values in the output variable "Species", the *DataType* is specified as "categorical" with the string values: "1", "2", and "3". In this case, three parameters (i.e., *Left*, *Middle*, and *Right*) representing the corner points of the triangular function are defined for each linguistic value in *Database*. Furthermore, regarding the *Rulebase*, we obtain 5 rules corresponding to the *Grade* which represents the degrees of certainty. It should be noted that the FRBS model generated by the GFS.GCCL method may contain the "dont_care" value representing a degree of 1.

```
<frbsPMML version="1.0" ...>
 <Header copyright="Copyright (c) 2014" description="The sim-0 was
      resulted by FRBCS.W method based on FRBCS model">
  <Extension name="user" value="Lala" extender="frbs"/>
  <Application name="frbs" version="3.0-0"/>
  <Timestamp>2014-07-04 17:51:02</Timestamp>
 </Header>
 <DataDictionary numberOfFields="5">
  <DataField name="Sepal.Length" optype="continuous" dataType="double">
   <Interval closure="closedClosed" leftMargin="4.3" rightMargin="7.9"/>
  </DataField>
  ...
  <DataField name="Species" optype="categorical" dataType="string">
   <Value value="1"/>
   <Value value="2"/>
   <Value value="3"/>
  </DataField>
 </DataDictionary>
<FrbsModel modelName="FRBCS" functionName="classification"
     algorithmName="GFS.GCCL" targetFieldName="Species">
  ...
  <InferenceSchema>
   <ConjunctionOperator value="PRODUCT"/>
   <DisjunctionOperator value="MAX"/>
   <ImplicationOperator value="ZADEH"/>
  </InferenceSchema>
 <Database>
  <MembershipFunction name="Sepal.Length" numberOfLabels="3">
   <FuzzyTerm name="small" type="TRIANGLE">
    <Parameters>
     <Left>0</Left>
     <Middle>0</Middle>
     <Right>0.5</Right>
    </Parameters>
   </FuzzyTerm>
   ...
   <FuzzyTerm name="large" type="TRIANGLE">
    <Parameters>
     <Left>0.5</Left>
     <Middle>1</Middle>
     <Right>1</Right>
    </Parameters>
   </FuzzyTerm>
  </MembershipFunction>
 </Database>
 <Rulebase numberOfRules="5">
  <Rule id="1">
   <If>
    <CompoundPredicate booleanOperator="and">
     <SimplePredicate field="Sepal.Length" value="dont_care"/>
     <CompoundPredicate booleanOperator="and">
      <SimplePredicate field="Sepal.Width" value="dont_care"/>
      <CompoundPredicate booleanOperator="and">
       <SimplePredicate field="Petal.Length" value="small"/>
```

```
      <SimplePredicate field="Petal.Width" value="dont_care"/>
     </CompoundPredicate>
    </CompoundPredicate>
   </CompoundPredicate>
  </If>
  <Then>
   <SimplePredicate field="Species" value="1"/>
  </Then>
  <Grade>0.9352</Grade>
 </Rule>
 ...
 <Rule id="5">
  <If>
   <CompoundPredicate booleanOperator="and">
    <SimplePredicate field="Sepal.Length" value="dont_care"/>
    <CompoundPredicate booleanOperator="and">
     <SimplePredicate field="Sepal.Width" value="medium"/>
     <CompoundPredicate booleanOperator="and">
      <SimplePredicate field="Petal.Length" value="medium"/>
      <SimplePredicate field="Petal.Width" value="medium"/>
     </CompoundPredicate>
    </CompoundPredicate>
   </CompoundPredicate>
  </If>
  <Then>
   <SimplePredicate field="Species" value="2"/>
  </Then>
  <Grade>0.687581670394897</Grade>
 </Rule>
</Rulebase>
</FrbsModel>
</frbsPMML>
```

Listing 17: XML Schema of the classification example.

6.2.2. Importing from the proposed format using "frbs" and "frbsJpmml"

As in the previous example, we perform prediction for new data using "frbs" as follows:

```
R> objectClass <- read.frbsPMML("modClass.frbsPMML")
R> res.test <- predict(objectClass, tst.iris)
```

Then, we can check the result, e.g., by calculating the percentage error:

```
R> err = 100 * sum(real.iris != res.test)/nrow(real.iris)
R> print(err)

[1] 4.444444
```

We can also perform prediction for the new data (in a file *newdataClass.txt*) using "frbsJpmml" by the following command:

```
java -jar frbsJpmml.jar "modClass.frbsPMML" "newdataClass.txt"
```

Furthermore, other examples of classification problems are available from our web page.

7. Conclusions and Future Work

The main contributions and results of this paper can be summarized as follows:

1. frbsPMML, which is a universal representation framework for FRBSs based on the PMML standard, has been presented. The specifications of the Mamdani, TSK, and FRBCS models, which contain the database, rulebase, and inference parameters, are provided in a flexible way. According to the available features and a comparison with other representations, it can be seen that the representation offers the following advantages: interoperability, reproducibility, transparency, interpretability, and flexibility.

2. "frbs," a standard package for constructing FRBS models in the R environment, allows to represents models in the frbsPMML format. Additionally, it offers several options in terms of FRBS models, learning methods, and other parameters for the reasoning and aggregation.
3. The software "frbsJpmml," written in Java, can be used to import or consume FRBS models in the frbsPMML format and for prediction on new data.
4. Usage examples of both software libraries have been illustrated in the paper.

As future work, we plan to implement the framework and its applications for Cloud Computing and some other programming languages, like C++ and Python. In addition, to increase the adoption of the proposed standard format, we are going to design and implement interfaces connecting existing software libraries with the frbsPMML format.

Acknowledgment

This work was partially supported by the Spanish Ministry of Economy and Competitiveness under Projects TIN2013-47210-P and TIN2014-57251-P, the Andalusian Research Plan P10-TIC-6858, P11-TIC-7765, and P11-TIC-9704, and Regional Project P12-TIC-2958. Lala Septem Riza would like to express his gratitude to the Dept. of Computer Science, Universitas Pendidikan Indonesia, for supporting him to pursue the Ph.D. program, and to the Directorate General of Higher Education of Indonesia, for providing a Ph.D. scholarship.

References

[1] L. A. Zadeh, Fuzzy sets, Information and control 8 (3) (1965) 338–353.

[2] T. Bohlin, Special issue on grey box modelling, International journal of adaptive control and signal processing 9 (6) (1995) 461–464.

[3] E. H. Mamdani, Application of fuzzy algorithms for control of simple dynamic plant, in: Proceedings of the Institution of Electrical Engineers, Vol. 121 of 12, IET, 1974, pp. 1585–1588.

[4] E. H. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, International journal of man-machine studies 7 (1) (1975) 1–13.

[5] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Transactions on Systems, Man and Cybernetics SMC-15 (1) (1985) 116–132.

[6] M. Sugeno, G. T. Kang, Structure identification of fuzzy model, Fuzzy sets and systems 28 (1) (1988) 15–33.

[7] Z. Chi, H. Yan, T. Pham, Fuzzy algorithms: With applications to image processing and pattern recognition, Vol. 10, World Scientific, 1996.

[8] H. Ishibuchi, K. Nozaki, H. Tanaka, Distributed representation of fuzzy rules and its application to pattern classification, Fuzzy sets and systems 52 (1) (1992) 21–32.

[9] D. P. Mandal, C. A. Murthy, S. K. Pal, Formulation of a multivalued recognition system, IEEE Transactions on Systems, Man and Cybernetics 22 (4) (1992) 607–620.

[10] I. Baturone, F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, P. Brox, A. A. Gersnoviez, M. Brox, Using Xfuzzy environment for the whole design of fuzzy systems, in: Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International, IEEE, 2007, pp. 1–6.

[11] The MathWorks, Inc., The fuzzy logic toolbox for use with MATLAB version 2 (2002).

[12] R. Ihaka, R. Gentleman, R: A language for data analysis and graphics, Journal of computational and graphical statistics 5 (3) (1996) 299–314.

[13] R Core Team, R: A language and environment for statistical computing. R foundation for statistical computing, vienna, austria, 2012 (2012).

[14] L. S. Riza, C. Bergmeir, F. Herrera, J. M. Benítez, frbs: Fuzzy rule-based systems for classification and regression in R, Journal of Statistical Software 65 (6) (2015) 1–30.

[15] S. Guillaume, B. Charnomordic, Learning interpretable fuzzy inference systems with FisPro, Information Sciences 181 (20) (2011) 4409–4427.

[16] J. M. Alonso, L. Magdalena, Generating understandable and accurate fuzzy rule-based systems in a java environment, in: Fuzzy Logic and Applications, Springer, 2011, pp. 212–219.

[17] J. Alcalá-Fdez, L. Sanchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, F. Herrera, KEEL: A software tool to assess evolutionary algorithms for data mining problems, Soft Computing 13 (3) (2009) 307–318.

[18] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework, Journal of Multiple-Valued Logic and Soft Computing 17 (255-287) (2010) 11.

[19] R. D. Peng, Reproducible research in computational science, Science (New York, Ny) 334 (6060) (2011) 1226.

[20] A. Guazzelli, M. Zeller, W. C. Lin, G. Williams, PMML: An open standard for sharing models, The R Journal 1 (1) (2009) 60–65.

[21] A. Guazzelli, K. Stathatos, M. Zeller, Efficient deployment of predictive analytics through open standards and cloud computing, ACM SIGKDD Explorations Newsletter 11 (1) (2009) 32–38.

[22] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, Above the clouds: A Berkeley view of cloud computing, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS 28 (2009) 13.

[23] R. Buyya, C. S. Yeo, S. Venugopal, Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities, in: High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, Ieee, 2008, pp. 5–13.

[24] R. Buyya, Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility, in: ChinaGrid Annual Conference, 2009. ChinaGrid'09. Fourth, IEEE, 2009, pp. xii–xv.

[25] J. Casillas, O. Cordón, F. Herrera, L. Magdalena, Interpretability issues in fuzzy modeling, Vol. 128, Springer, 2013.

[26] Institute of Electrical & Electronics Engineers (IEEE), IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries: 610 (1991).

[27] World Wide Web Consortium (W3C), XML 1.0 specification, `http://www.w3.org/TR/xml/` (2014).

[28] P. Walmsley, Definitive XML schema, Prentice Hall, 2012.

[29] Data Mining Group, PMML 4.2, `http://www.dmg.org/v4-2-1/GeneralStructure.html` (2014).

[30] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel, KNIME: The Konstanz information miner, Springer, 2008.

[31] D. Morent, K. Stathatos, W. C. Lin, M. R. Berthold, Comprehensive PMML preprocessing in KNIME, in: Proceedings of the 2011 workshop on Predictive markup language modeling, ACM, 2011, pp. 28–31.

[32] A. Guazzelli, Representing predictive solutions in PMML: From raw data to predictions, IBM developerWorks website (2010).

[33] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The WEKA data mining software: An update, ACM SIGKDD explorations newsletter 11 (1) (2009) 10–18.

[34] L. X. Wang, J. M. Mendel, Generating fuzzy rules by learning from examples, IEEE Transactions on Systems, Man and Cybernetics 22 (6) (1992) 1414–1427.

[35] J. S. R. Jang, ANFIS: adaptive-network-based fuzzy inference system, IEEE Transactions on Systems, Man and Cybernetics 23 (3)

(1993) 665–685.

[36] J. Kim, N. Kasabov, HyFIS: Adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems, Neural Networks 12 (9) (1999) 1301–1319.

[37] O. Cordón, F. Herrera, F. Hoffmann, L. Magdalena, Genetic fuzzy systems: Evolutionary tuning and learning of fuzzy knowledge bases, Vol. 19, World Scientific Publishing Company Singapore, 2001.

[38] F. Herrera, Genetic fuzzy systems: Taxonomy, current research trends and prospects, Evolutionary Intelligence 1 (1) (2008) 27–46.

[39] P. Thrift, Fuzzy logic synthesis with genetic algorithms., in: ICGA, 1991, pp. 509–513.

[40] F. Herrera, M. Lozano, J. Verdegay, A learning process for fuzzy control rules using genetic algorithms, Fuzzy sets and systems 100 (1) (1998) 143–158.

[41] H. Ishibuchi, T. Nakashima, T. Murata, Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 29 (5) (1999) 601–618.

[42] H. Ishibuchi, T. Yamamoto, T. Nakashima, Hybridization of fuzzy gbml approaches for pattern classification problems, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 35 (2) (2005) 359–365.

[43] S. Chiu, Method and software for extracting fuzzy classification rules by subtractive clustering, in: Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American, IEEE, 1996, pp. 461–465.

[44] N. Kasabov, Q. Song, DENFIS: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction, Fuzzy Systems, IEEE Transactions on 10 (2) (2002) 144–154.

[45] H. Nomura, I. Hayashi, N. Wakami, A learning method of fuzzy inference rules by descent method, in: Fuzzy Systems, 1992., IEEE International Conference on, IEEE, 1992, pp. 203–210.

[46] H. Ishibuchi, K. Nozaki, H. Tanaka, Empirical study on learning in fuzzy systems by rice taste analysis, Fuzzy Sets and Systems 64 (2) (1994) 129–144.

[47] S. M. Zhou, J. Q. Gan, Low-level interpretability and high-level interpretability: a unified view of data-driven interpretable fuzzy system modelling, Fuzzy Sets and Systems 159 (23) (2008) 3091–3131.

[48] F. J. Moreno-Velo, A. Barriga, S. Sanchez-Solano, I. Baturone, XFSML: An XML-based modeling language for fuzzy systems, in: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2012, IEEE, 2012, pp. 1–8.

[49] G. Acampora, V. Loia, Fuzzy control interoperability and scalability for adaptive domotic framework, IEEE Transactions on Industrial Informatics 1 (2) (2005) 97–111.

[50] G. Acampora, V. Loia, C. S. Lee, M. H. Wang, On the power of fuzzy markup language, Springer, 2013.

[51] H. Aydt, S. J. Turner, W. Cai, M. Y. H. Low, Y. S. Ong, R. Ayani, Toward an evolutionary computing modeling language, IEEE Transactions on Evolutionary Computation 15 (2) (2011) 230–247.

[52] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, J. Srinivasan, Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle, in: IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008, IEEE, 2008, pp. 1239–1248.