The PMML Path towards True Interoperability in Data Mining

Alex Guazzelli
Zementis, Inc.*
Alex.Guazzelli@
zementis.com

Tridivesh Jena
Zementis, Inc.
Tridivesh.Jena@
zementis.com

Wen-Ching Lin
Zementis, Inc.
Wenching.Lin@
zementis.com

Michael Zeller
Zementis, Inc.
Michael.Zeller@
zementis.com

ABSTRACT

As the de facto standard for data mining models, the Predictive Model Markup Language (PMML) provides tremendous benefits for business, IT, and the data mining industry in general, since it allows for predictive models to be easily moved between applications. Due to the cross-platform and vendor-independent nature of such an open-standard, auto-generated PMML code is often represented in different versions of PMML. A tool may export PMML 2.1 and another import PMML 4.0. This problem raises the issue of conversion. For true interoperability, PMML needs to be easily converted from one version to another.

In this paper, we describe the capabilities associated with the "PMML Converter". This application represents a great step in the PMML path towards true interoperability in data mining. Besides converting older versions of PMML to its latest, the PMML converter checks PMML files for syntax issues and, if issues are encountered, automatically corrects them.

This paper also describes the capabilities associated with an interactive PMML-based application, the "Transformations Generator." Auto-generated PMML code can omit important data pre-processing steps which are an integral part of a predictive solution. The Transformations Generator aims to bridge this gap by providing a graphical interface for the development and expression of data pre-processing steps in PMML.

Categories and Subject Descriptors

H.2.8 [Database Management]: Data Mining; G.3 [Probability and Statistics]: Statistical Computing, Statistical Software; I.5.1 [Models]: Statistical Models, Neural Nets.

General Terms

Management, Performance, Standardization, Languages.

Keywords

Open Standards, Predictive Analytics, Data Mining, PMML, Predictive Model Markup Language, Preprocessing, Data Transformations.

---

* Zementis, Inc. (http://www.zementis.com) is located at 6125 Cornerstone Court East, Suite 250, San Diego, CA.

1. INTRODUCTION

PMML (Predictive Model Markup Language) is the de facto standard used to represent and share predictive analytic solutions between applications [1][2][3]. It enables data mining scientists and users alike to easily build, visualize, and deploy their solutions using different platforms and systems. PMML is the brain child of the DMG (Data Mining Group), an independent, vendor-led consortium that develops data mining standards. Given that PMML is an XML-based language, the DMG is responsible for publishing a PMML Schema (.XSD file) which is specific on how all language elements comprising a PMML file should be used.

As an open-standard for data mining, PMML first made its debut in 1997 with version 0.7. As the language became more refined, it was updated throughout the years, which resulted in the release of many versions, culminating in version 4.0, its latest, released in June 2009 [4]. Today, PMML is a mature and refined language supported by all major statistical and data mining applications.

In the ideal scenario, all applications and tools would support the latest version of PMML at all times. In reality though, due to commercial product release cycles, PMML producers and consumers have a tendency to lag behind when it comes to updating importers and exporters to accompany the latest PMML release. In addition, some applications that produce PMML do not adhere 100% to the schema published by the DMG.

Therefore, an application that can convert older versions of PMML to its latest, and also validate PMML files against its schema is highly desirable. Zementis, with the support of the DMG, has released such an application. Called the "PMML Converter", it is available for use free of charge by the community at large via the DMG website (http://www.dmg.org) and the PMML resources page in the Zementis website (http://www.zementis.com). The PMML Converter is further described in Section 2.

Besides developing PMML, another DMG goal is to make learning material and PMML related tools available to the data mining community. As a member of the DMG, we supported these goals by publishing the first book about PMML in 2010. Our book, entitled "PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics" (see [1]), is now available for purchase on Amazon.com.

Furthermore, in line with the DMG goals, Zementis introduced a PMML application providing an interactive, hands-on approach to the generation of data transformations. These transformations are key ingredients in the execution of a predictive solution, since they are used to represent the preprocessing steps necessary to transform the raw data into feature detectors which are in turn used by the predictive algorithms themselves [5]. With this application, called the "Transformations Generator", the user

graphically designs a transformation and immediately has access to the resulting PMML code.

PMML provides a variety of data transformations, including value mapping, normalization, and discretization. It also offers several built-in functions as well as arithmetic and logical operators which can be combined to represent complex pre-processing steps. In Section 3, we illustrate the use of these operations in the interactive PMML tool.

2. PMML CONVERTER

The PMML Converter encompasses three key functions. These are:

- Conversion of PMML files to version 4.0
- Validation of PMML files against the different PMML versions
- Automatic correction of certain issues

First and foremost, the PMML Converter is responsible for converting older versions of PMML to version 4.0, its latest. This can only be done though once the provided PMML file is validated against the language schema. If a predictive solution is represented in PMML 2.1, the PMML file needs to be successfully validated against the PMML schema for version 2.1 before it is converted to version 4.0.

The PMML Converter will automatically correct known issues with PMML code from several sources/vendors. The aim is to successfully validate code in older versions of PMML and convert them to the latest PMML version. Files in the latest PMML version can also be passed through the converter so that they can be corrected and validated against the schema.

If the PMML code cannot be converted, it usually means that it could not be automatically corrected. In that case, the PMML Converter embeds comments into the PMML code pinpointing any problems so that they can be fixed manually before being submitted again for conversion. This is done via a hyper-link in the converter which allows for the PMML file to be downloaded after a failed conversion.

3. Using the PMML Converter

To start a conversion, the user selects and uploads into the converter a PMML file of any of the following versions: 2.0, 2.1, 3.0, 3.1, 3.2 or 4.0. Once uploading is completed, the file is converted automatically.

If the model is converted successfully, it will be available for download. One can then click on the "Download" button in the converter interface to save the file locally. The user may also register for a free trial of the Zementis ADAPA Scoring Engine [6] by clicking on "Try your model for free in ADAPA". Since ADAPA incorporates the PMML Converter, the user could upload the original PMML file directly in ADAPA for scoring.

Note that the PMML Converter expects valid PMML files. Auto-generated PMML code may sometimes contain elements that are not valid PMML. In that case, the converter will display a message stating that the file does not conform to the PMML specification. One is able to download the original PMML file with converter-generated comments by clicking on the "Details" button. A comment summarizing the problems encountered during conversion is automatically placed on the very top of the file. Specific comments are also embedded into the file exactly where a problem was found. One can then use this information as feedback to manually obtain a valid PMML file before attempting to convert the file again.

3.1 The PMML Converter in Action

The examples shown here illustrate the convenience provided by the PMML Converter by 1) automatically converting PMML code to its latest version; 2) correcting schema errors; and 3) pointing the location of more serious errors. Typically, the entire PMML file is uploaded into the converter for conversion to take place. However, due to space limitations and sake of clarity, only part of the PMML code highlighting the fragment being converted is shown here. Note that the examples described in this article showcase a small subset of all the capabilities of the PMML Converter.

As aforementioned, for conversion to take place, PMML code needs to be successfully validated against the schema. Figure 1 shows an example of a "DataDictionary" element. This element lists all input variables used in the model along with their data type and operational type. Note that in this example, each variable is defined as of type "double" but the operational type ("optype"), a required attribute, was omitted.

```
<DataDictionary numberOfFields="3">
    <DataField name="var1" dataType="double"/>
    <DataField name="var2" dataType="double"/>
    <DataField name="var3" dataType="double"/>
</DataDictionary>
```

Figure 1. PMML code before conversion. "DataField" elements are missing required attribute "optype".

Figure 2 shows the same PMML fragment after being corrected by the PMML Converter. The converter safely assumes that double valued variables are continuous by default and automatically adds the operational type ("continuous") to all variables.

```
<DataDictionary numberOfFields="3">
    <DataField name="var1"
        dataType="double" optype="continuous"/>
    <DataField name="var2"
        dataType="double" optype="continuous"/>
    <DataField name="var3"
        dataType="double" optype="continuous"/>
</DataDictionary>
```

Figure 2. PMML code with schema errors corrected by the converter. Each "DataField" element now contains all required attributes.

As with any evolving standard, the PMML language and its schema have been updated and refined throughout the years. As part of this process, certain language elements which were valid in an older version of the standard may no longer be valid in a newer one. For example, the "CenterFields" element shown in Figure 3 was used in earlier versions of PMML (e.g., 2.1, 3.0, and 3.1) to represent variables used to calculate the distances in a Clustering Model. Note that in this case, two variables are defined: "clst0" and "clst1". These derived variables are used to calculate the distances to clusters "0" and "1".

This representation was changed in subsequent PMML versions, where all derived fields are now represented in one central location, the "LocalTransformations" element. Therefore, the converter moves all derived fields defined under the element "CenterFields" to the "LocalTransformations" element.

```xml
<ComparisonMeasure kind="distance">
<Euclidean/>
</ComparisonMeasure>
<ClusteringField field="var1" |
    compareFunction="absDiff"/>
<ClusteringField field="var2"
    compareFunction="absDiff"/>
<CenterFields>
<DerivedField name="clst0"
    dataType="double" optype="continuous">
    ...
</DerivedField>
<DerivedField name="clst1"
    dataType="double" optype="continuous">
    ...
</DerivedField>
</CenterFields>
```

Figure 3. PMML code incompatible with the latest schema. "CenterFields" element contains "DerivedField" elements.

Figure 4 shows the resulting PMML code which conforms to the latest PMML version. As dictated by the language schema, the converter placed the derived fields before element "ComparisonMeasure".

```xml
<LocalTransformations>
<DerivedField name="clst0"
    dataType="double" optype="continuous">
    ...
</DerivedField>
<DerivedField name="clst1"
    dataType="double" optype="continuous">
    ...
</DerivedField>
</LocalTransformations>
<ComparisonMeasure kind="distance">
  <Euclidean>
    <!--(Comment generated by ADAPA) Schema Error:
    Expected elements 'Extension euclidean
    squaredEuclidean chebychev cityBlock
    minkowski simpleMatching jaccard tanimoto
    binarySimilarity' instead of 'Euclidean'
    here in element ComparisonMeasure-->
  </Euclidean>
</ComparisonMeasure>
```

Figure 4. Converted PMML code. Although the "DerivedField" elements are now part of "LocalTransformatiosn", the converter has found a syntax error in the element "ComparisonMeasure".

Figure 4 also shows a syntactic schema violation which was encountered by the converter when manipulating the submitted PMML file. The error message here indicates that "Euclidean" is not recognized in element "ComparisonMeasure". The message provides help to the user by listing all the possible element names it expected instead of the unrecognized term. Either by looking at the schema or by going through the provided list, one can immediately realize that "Euclidean" needs to be replaced by "euclidean" for the PMML code to be valid.

Once a complete PMML file which describes the entire modeling process is generated, the PMML converter can be used to check the code syntactically and provide proper corrections. A corresponding updated PMML file in the latest version is then returned to the user. It should be noted that while the converter can check a PMML file syntactically by comparing its elements and attributes with the language schema, it does not provide a semantic check of the file. This is because semantic checks imply model execution or, ultimately, the process of preparing a model for scoring. This additional functionality is provided as part of the ADAPA Scoring Engine. Since ADAPA incorporates the PMML Converter, it is capable of performing syntactic and semantic validation of PMML files, including pre- and post-processing elements as well as a number of predictive techniques.

4. TRANSFORMATIONS GENERATOR

In the spirit of creating PMML applications that help disseminate the use and understanding of the standard, Zementis has also made available an application for generating data transformations in PMML. Although PMML is supported by all the major data mining applications, export of PMML models often lacks preprocessing steps which are an essential part of a predictive solution [5]. This scenario is changing rapidly though since the advent of PMML 4.0 in 2009. PMML version 4.0 added several built-in functions to the standard which allow for the representation of rich and complex preprocessing steps. Worth noting is KNIME, an open-source analytical platform, that has been in the forefront of incorporating rich PMML preprocessing support [7].

From early on (as in versions 1.1, 2.0, and 2.1), PMML already defined several kinds of data transformation elements. These are:

- Normalization: Map values to numbers, the input can be continuous or discrete.
- Discretization: Map continuous values to discrete values.
- Value Mapping: Map discrete values to discrete values.
- Functions: Derive a value by applying a function to one or more parameters.

PMML also defines a comprehensive list of built-in functions which perform text, arithmetic, and logical manipulations.

Below we will see how these PMML elements can be used to represent a wide variety of preprocessing operations.

4.1 Continuous Normalization

Continuous normalization in PMML is represented by the element "NormContinous". This element provides a general method to normalize the data of a continuous variable from a given range to another designated range. For example, given that the data for input variable "var" ranges from 500 to 1,000, we would like to linearly normalize it to values from 0 to 1. This implies the need for two "LinearNorm" elements in PMML, one mapping 500 to 0, and the other mapping 1,000 to 1. To generate the PMML code for the above transformation, we can simply utilize the Transformations Generator.

As shown in Figure 5, we entered the names of the original input variable "var" as well as the desired derived variable "derivedVar" (to be obtained as a result of the normalization) in the Transformations Generator. The third item "Map Missing Values To" lets a user specify what the value of variable "derivedVar" should be if variable "var" is missing. Then the normalization formula should be provided by supplying the "Normalization Elements". Once all required fields are filled, one can click the button "Generate PMML Code" to obtain the respective PMML code (shown in Figure 6).

Figure 5. Continuous normalization example as represented in the Transformations Generator graphical interface.

```
<DerivedField name="derivedVar"
    optype="continous" dataType="double">
    <NormContinous field="var"
        mapMissingTo="0.5" outliers="asIs">
        <LinearNorm orig="500" norm="0"/>
        <LinearNorm orig="1000" norm="1"/>
    </NormContinous>
</DerivedField>
```

Figure 6. PMML code generated for the continuous normalization transformation shown in Figure 5.

4.2 Discrete Normalization

Discrete normalization in PMML is represented by the element "NormDiscrete". This element is used to transform string values to numeric values. Many models encode string values into numeric values in order to perform mathematical functions, such as regression and neural network models. In this case, split categorical variables are used to create multiple derived "dummy" variables.

For example, say a categorical variable "colorVar" contains data with possible values red, blue, and green. When applied to "colorVar", the "NormDiscrete" element creates three new derived variables for each categorical value which we could name: "colorRed", "colorBlue", and "colorGreen". When the value of "colorVar" is red, variable "colorRed" would be 1 and "colorBlue" and "colorGreen" would both be 0. The same rationale applies to colors blue and green and variables "colorBlue" and "colorGreen". By entering this information in the respective graphical elements in the Transformations Generator, as shown in Figure 7, and clicking on the button "Generate PMML Code", one obtains the corresponding code for this "NormDiscrete" transformation as shown in Figure 8.

Figure 7. Discrete normalization example as represented in the Transformations Generator graphical interface.

```
<DerivedField name="colorRed"
    optype="continous" dataType="double">
    <NormDiscrete field="colorVar"
    value="red" mapMissingTo="1">
</DerivedField>
<DerivedField name="colorBlue"
    optype="continous" dataType="double">
    <NormDiscrete field="colorVar"
    value="blue" mapMissingTo="1">
</DerivedField>
<DerivedField name="colorGreen"
    optype="continous" dataType="double">
    <NormDiscrete field="colorVar"
    value="green" mapMissingTo="1">
</DerivedField>
```

Figure 8. PMML code generated for the discrete normalization transformation shown in Figure 7.

4.3 Discretization

Discretization in PMML is represented by the element "Discretize". This element is used for the mapping of continuous values to discrete values. For a continuous variable we define a set of intervals. If its input value falls within one of the intervals, we assign it a new value as defined by that interval. For example, consider a continuous input variable "var" which needs to be discretized into three bins: "low", "medium" and "high". Each bin corresponds to the following three intervals: (-infinity, 7], (7, 10], and (10, 30), respectively. As shown in Figure 9, we entered this information in the Transformations Generator, including closure information associated with each interval. Note that in the generated PMML code, shown in Figure 10, each interval is associated with a "DiscretizeBin" element.

Figure 9. Discretization example as represented in the Transformations Generator graphical interface.

```
<DerivedField name="derivedVar"
    optype="categorical" dataType="string">
  <Discretize field="var"
   mapMissingTo="low" defaultValue="medium">
      <DiscretizeBin binValue="low">
          <Interval closure="openClosed"
              rightMargin="7"/>
      </DiscretizeBin>
      <DiscretizeBin binValue="medium">
          <Interval closure="openClosed"
              leftMargin="7" rightMargin="10"/>
      </DiscretizeBin>
      <DiscretizeBin binValue="high">
          <Interval closure="openOpen"
              leftMargin="10" rightMargin="30"/>
      </DiscretizeBin>
  </Discretize>
</DerivedField>
```

Figure 10. PMML code generated for the discretization transformation shown in Figure 9.

The mapping from a continuous to a discrete variable can be many-to-one but not one-to-many. In other words, two intervals can be mapped to the same string value but the same interval may not have more than one string value. This implies that the intervals defined must be disjoint. There can be no overlap between two defined intervals.

One can establish what should happen if the input is not in any of the defined intervals. The attribute "defaultValue" in the "Discretize" element establishes that if the input does not lie in any of the intervals (as for example, if "var" = 31), then by default it is assigned the value "medium".

4.4 Value Mapping

Value mapping in PMML is represented by the element "MapValues". This element is used to map discrete values to discrete values. This is done by using a table which lists the input values and the mapped output values. Each row in the table refers to a possible value for the input variable(s). Each column in the table has a name which is used to refer to that column.

Table 1. Mapping table with one input column and one output column.

input	output
sunny	play
cloudy	read
rainy	sleep

As an example, consider the table shown in Table 1. The column "input" corresponds to input field "weather" and column "output" is associated with the derived variable "activity". Figure 11 shows this information as entered in the Transformation Generator.

Please, enter information for Value Mapping

Original Field Name(s):	weather + -
Derived Field Name:	activity
Derived Data Type:	string
Derived Operational Type:	categorical
Map Missing To:	play
Default Value:	read

Mapping Elements:

	Input Column(s):	input	Output Column:	output
1	Value(s):	sunny	Maps to value:	play
2	Value(s):	cloudy	Maps to value:	read
3	Value(s):	rainy	Maps to value:	sleep

+ -

Generate PMML Code

Figure 11. Value mapping example as represented in the Transformations Generator graphical interface.

In the generated PMML code, shown in Figure 12, the attribute "outputColumn" in element "MapValues" establishes that the mapped output value is to be found in the column named "output". The element "FieldColumnPair" uses the attribute "field" to define the input for mapping, which consists of one or more input variables. In this example, we used variable "weather" as input. We find the input values in the "input" column for "weather" which are mapped to the output value in the "output" column.

```
<DerivedField name="activity"
    optype="categorical" dataType="string">
  <MapValues outputColumn="output"
   mapMissingTo="play" defaultValue="read">
      <FieldColumnPair
          field="weather" column="input"/>
      <InlineTable>
          <row><input>sunny<input>
              <output>play<output></row>
          <row><input>cloudy<input>
              <output>read<output></row>
          <row><input>rainy<input>
              <output>sleep<output></row>
      </InlineTable>
  </MapValues>
</DerivedField>
```

Figure 12. PMML code generate for the value mapping transformation shown in Figure 11.

As in the Discretize element, the "MapValues" element can have a "defaultValue" attribute which specifies what to map the input to if it does not have a matching input value (or set of).

4.5 Functions

If a certain transformation is to be applied to input data many times and to multiple fields, it makes sense to encapsulate the transformation inside a function and just use it as many times as necessary. This reduces the complexity of the PMML model and greatly simplifies its application. PMML provides a number of built-in functions as well as the capability for the user to define a function. User-defined functions are not part of the Transformations Generator and so we focus here solely on built-in functions.

4.6 Built-in Functions

Figure 13 shows the Transformations Generator interface for the building of a "Generic Operation". PMML supports a variety of arithmetic and logic functions. These can be used together to create complex operations which allow for the representation of

generic data preprocessing. Not shown in Figure 13 are the instructions for building generic operations using the graphical interface provided by the Transformations Generator. These can be found in the same area used to show the generated PMML code.

In the following example, we show how to use the Transformations Generator to generate a piece of PMML code that calculates a derived field "derivedVar" by adding variables "var1" and "var2". We first enter the desired output variable name "derivedVar" in the "Derived Field Name" field in the Transformation Generator, and choose the operation "Arithmetic: @+@". We then click the "@" under the "Expression Tree" which turns it red. Figure 13 shows part of the Transformations Generator which depicts the information and actions taken up to this point.

Figure 13. Designing a simple arithmetic operation.

Once the icon "@" under "Expression Tree" turns red, we can click the "Add" button to add the arithmetic component "@+@". One can then expand the blue plus icon in front of the arithmetic sign we just added to continue building the expression tree. Similar to the previous procedure, we can now click either one of the "@" in the expression tree until it turns red, then add in the required input fields: "var1" and "var2". The resulting expression tree is shown in Figure 14.
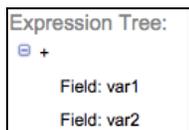

Figure 14. Expression tree for adding two variables.

After both "var1" and "var2" are added into the expression tree, one can then click the "Generate PMML Code" button to generate the PMML code as shown in Figure 15.

Figure 15. PMML code generated for the expression tree shown in Figure 14.

PMML defines many functions that support Boolean operations. These are used to compare parameters which are required to be of identical type (e.g., strings or dates) or of compatible type for numeric variables (e.g., double vs. integer). The result is of Boolean type: "true" or "false", which is evaluated by functions such as "if" and "not".

Below, we give an example that shows how to utilize the Transformations Generator to generate the PMML code for logical operations. Assume we want to inspect input variable "var": if it equals "green", we assign output variable "derivedVar" to value "go"; otherwise, we assign the output variable to value "stop".

In the Transformations Generator, by selecting the "if @ then @ else @" element and adding it to the expression tree, we end up with three "@" icons in the Expression Tree. We use the first one to add the logical operation "@ equal @". This operation is responsible for checking if the input variable "var" ("@ = Field Name: var") equals "green" ("@ = Constant Value: green"). If the result is "true", the output field "derived_var" is assigned the value specified in the second "@" icon ("@ = Constant Value: go"). Otherwise, the output field is assigned the value in the third "@" icon ("@ = Constant Value: stop"). The resulting expression tree is shown in Figure 16 and its corresponding PMML code is shown in Figure 17.
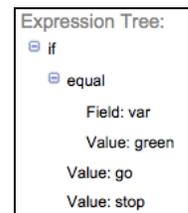

Figure 16. Expression tree representing the logical built-in function "if @ then @ else @". In this example, if the variable "var" equals "green", then the derived variable "derivedVar" is assigned the string "go"; otherwise, "derivedVar" is assigned the string "stop".

Figure 17. PMML code generated for the expression tree shown in Figure 16.

To summarize, the Transformations Generator can be used to generate a myriad of operations which can then be copied to a PMML file. It also serves as a great learning tool for PMML, since it allows users to interactively explore all its components and immediately see the respective PMML code.

Like the PMML Converter, the Transformations Generator can also be accessed directly from the DMG website or from the Zementis PMML resources web page.

5. CONCLUSION

The main goal of the DMG is the development of PMML, which is now the de facto standard to represent data mining models. More than 10 years in the making, PMML is a mature and refined standard. The path to maturity though left a legacy of different versions which are supported by different analytical applications. In Part I of this article, we described the PMML Converter, an application that converts older versions of PMML, starting with version 2.0, to its latest, version 4.0. The PMML Converter is also used for the syntactic validation of PMML code. By comparing uploaded files against the PMML schema, the converter is able to pinpoint any encountered issues and, in many cases, automatically correct them. The PMML Converter has become an indispensable application under the PMML tool belt.

Part II of this article describes a second important PMML application, the Transformations Generator. This application allows users to interactively design data preprocessing steps in PMML. Through its graphical interface, it can be used to generate PMML code for a myriad of transformations, including not only normalization and value mapping, but also generic operations containing arithmetic and logical functions. Finally, the Transformations Generator demonstrates how easy it is to represent data transformations in PMML.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] A. Guazzelli, W. Lin, T. Jena (2010). *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics.* CreativeSpace (available on Amazon.com).

[2] A. Guazzelli, M. Zeller, W. Lin, G. Williams. PMML: An Open Standard for Sharing Models. The R Journal, Volume 1/1, May 2009.

[3] A. Guazzelli (2010). What is PMML? Explore the power of predictive analytics and open standards. IBM developerWorks.

[4] R. Pechter. What's PMML and What's New in PMML 4.0? ACM SIGKDD Explorations Newsletter, July 2009.

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

[6] A. Guazzelli, K. Stathatos, M. Zeller. Efficient Deployment of Predictive Analytics through Open Standards and Cloud Computing. ACM SIGKDD Explorations Newsletter, July 2009.

[7] D. Morent, K. Stathatos, W. Lin, M. Berthold. Comprehensive PMML Preprocessing in KNIME. To appear in the *Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.