The R *pmmlTransformations* Package

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

Alex Guazzelli
Zementis, Inc.
Alex.Guazzelli@
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. Due to the cross-platform and vendor-independent nature of such an open-standard, it allows for predictive models to be easily moved between applications.

Although PMML has offered support for common data transformations for quite some time now, the release of PMML 4.0 in 2009 brought the support for data pre-processing steps to a new level. As a consequence, several data mining tools and model building platforms have been adding more and more support for data pre-processing into the PMML code they export. It is no surprise then that the same is true for R.

R has become a popular statistical platform for all things analytics. The R Project allows for a myriad of specialized packages to be installed and utilized by its users as needed. These include packages and functions for predictive analytics and model building. A package for exporting PMML out of several model types is also available. Called the *pmml* package, it allows for a few data pre-processing steps to be exported together with the modeling technique itself. However, a package to enable data transformations in a generic way was still missing.

This paper describes a package which intends to close this gap. The *pmmlTransformations* package provides R users with functions that greatly enhance the available data mining capabilities and PMML support by allowing transformations to be performed on the data before it is used for modeling. The *pmmlTransformations* package works in tandem with the *pmml* package so that data pre-processing can be represented together with the model in the resulting PMML code.

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, The R Project, pmmlTransformations.

PMML'13. Aug 10, 2013, Chicago, IL, USA

* Zementis, Inc. (http://www.zementis.com) is located at 3570 Carmel Mountain Road, 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 brainchild 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. Among all these elements, one can find several that are specific to different modeling techniques. In addition, the language also offers a cadre of elements to represent data transformations [4]. In this way, PMML can be used to implement not only the model itself, but also data pre- and post-processing.

The R Project is a platform for data mining and statistical analysis. It is widely used among data miners and statisticians for data analysis and model building. Recent surveys have shown that its popularity has increased significantly in recent years. A major attraction of R is its extensibility. Users can write their own functions and analysis tools, which can then be uploaded into a common repository, the Comprehensive R Archive Network (CRAN), and used by others. These are provided in the form of R packages which, once tested, are officially made available by the R maintainers for all users.

A package for exporting predictive models built in R into PMML has existed since 2009 [5]. With this package, a data scientist can mine data and construct a model to suit her needs. She can then use the *pmml* package to output her model as a PMML file, which can be deployed and put to use immediately in a variety of platforms, independently of R. These include the Zementis ADAPA Scoring Engine, which is available for on-site deployment as well as on the Amazon and IBM cloud infrastructures [6] and the Zementis Universal PMML Plug-in (UPPI), which offers big data scoring for Hadoop and in-database [7].

Although the R *pmml* package offers support for a myriad of model objects, it offers limited functionality for data transformations. In order to be exported as PMML, these need to be part of the resulting model object. As data frequently has to be transformed before it can be used for modeling, a package that allows for generic data transformations and their expression in PMML was needed. The *pmmlTransformations* package is such a package. It works in tandem with the *pmml* package to be able to represent data transformations and model in a single PMML file. Other tools already provide similar functionality, including KNIME [8] and IBM SPSS Statistics [4].

While Section 2 of this article focuses on the *pmml* package and its support for different data mining algorithms in R, Section 3

focuses on the *pmmlTransformations* package. We describe its functionality in detail through the use of examples, which include not only the R sequence of commands necessary to implement data transformations, but also the resulting PMML code. Finally, Section 4 offers the conclusion.

2. Exporting PMML from R

To export PMML from R, users have traditionally relied on a single package, the *pmml* package. This package allows for a variety of different modeling techniques to be exported into a ready to be deployed PMML file. As of the time of this article, the *pmml* package offers support for the following data mining algorithms:

- *ksvm* (*kernlab*): Support Vector Machines
- *nnet*: Neural Networks
- *rpart*: C&RT Decision Trees
- *lm* & *glm* (*stats*): Linear and Binary Logistic Regression Models as well as Generalized Linear Models for classification and regression with a wide variety of link functions
- *multinom* (*nnet*) Multinomial Logistic Regression Models
- *arules*: Association Rules
- *kmeans* and *hclust*: Clustering Models
- *coxph* (*survival*): Cox Regression Models to calculate survival and stratified cumulative hazards
- *randomForest*: Random Forest Models for classification and regression
- *naiveBayes* (*e1071*): Naïve Bayes Classifiers
- *glmnet*: Linear ElasticNet Regression Models

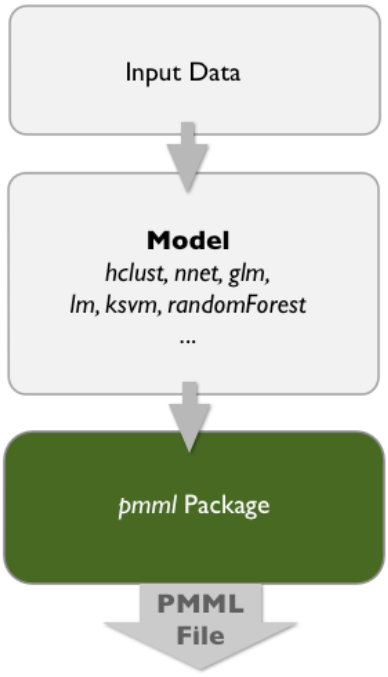

Figure 1. Diagram illustrating the data flow in R for the creation of a PMML file.

If, for example, the user builds a Random Forest Model using the *randomForest* package, it can be exported in PMML by simply invoking the *pmml* package with the resulting model object. The sequence of R commands below shows such an example.

```
library(randomForest);
library(pmml);
data(airquality);
ozone.out <- randomForest(Ozone ~ Wind+Temp+Month,
  data=na.omit(airquality), ntree=200);
saveXML(pmml(ozone.out, data=airquality),
  "airquality_rf.pmml");
```

Note that the *pmml* function is invoked with two parameters, the first being the model object *ozone.out*. Note also that the *saveXML* package is used to save the resulting PMML code into file *airquality_rf.pmml*.

As depicted in Figure 1, the data is presented as is to the modeling package and by consequence, the resulting PMML file will only contain the computations specified in the resulting model object.

For certain modeling packages, a few pre-processing steps are performed along with the modeling task. These steps represent data transformation directives performed by the model package itself. For example, *ksvm* automatically transforms categorical input fields into numerical fields before presenting them to the support vector machine. Given that such directives are part of the resulting model object, they are also part of the generated PMML file. Although data pre-processing functionality is limited to a few steps for certain packages, it is non-existent for others. This picture has changed considerably with the introduction of the *pmmlTransformations* package.

3. The *pmmlTransformations* Package

PMML defines several kinds of data transformation elements. These are:

- Normalization (PMML elements *NormContinuous* and *NormDiscrete*): Map values to numbers, the input can be continuous or discrete.
- Discretization (element *Discretize*): Map continuous values to discrete values.
- Value Mapping (element *MapValues*): Map discrete values to discrete values.
- Functions (invoked via element *Apply*): 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 operations.

The *pmmlTransformations* package provides the ability to perform a variety of data transformations prior to modeling, which are directly representable in PMML through the language's transformation elements. The *pmml* package, which has been updated to work in tandem with the *pmmlTransformations* package, can then represent the entire modeling process; data pre-processing as well as modeling.

In R, this process includes three steps:

1. With the use of the *pmmlTransformations* package, transform the raw input data as appropriate
2. Use transformed and raw data as inputs to the modeling function/package
3. Output the entire solution (data pre-processing + model) in PMML using the *pmml* package

Figure 2 shows the diagram depicting such a process flow.

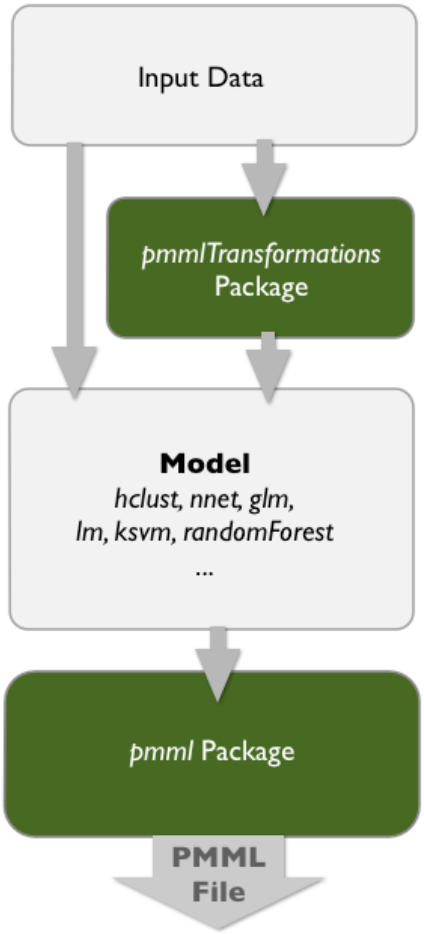

Figure 2. Diagram illustrating how several R packages can work together to create a single PMML file.

The process of choosing a specific modeling package depends on the kind of model the analyst wishes to build. Examples include packages *nnet* for building Neural Network Models and *randomForest* for building Random Forest Models. The data pre-processing and model are output in PMML by simply passing the transformations object obtained from the *pmmlTransformations* package along with the model object obtained from the modeling package as inputs to the *pmml* package.

The sequence of R commands depicted below shows an example of how input data can be transformed with the use of the *pmmlTransformations* package. Note that in this example, we are using the Iris dataset [9] to create a linear regression model for which the predicted field is "Sepal.Length" (not the typically used field "Species" - remember, this is an example). As it can be seen, the information about the data transformation, in this case, a continuous normalization operation, is stored in an object called *irisBox*. This transformation object together with the linear regression object *irisLR* are then passed as arguments to the *pmml* package, which is ultimately responsible for generating the resulting PMML code.

```
library(pmmlTransformations);
library(pmml);
data(iris);
irisBox <- WrapData(iris);
irisBox <- MinMaxXform(irisBox, xformInfo="column2");
dataset <- irisBox$data[c(1:4,6)];
irisLR <- lm(Sepal.Length~., data=dataset);
pmml(irisLr, transform=irisBox);
```

The standard procedure to use any of the functions provided by the *pmmlTransformations* package is to read in the raw data and first initialize the transformations property by wrapping the data in a transformations object using the *WrapData* function. All subsequent transformation operations are applied to the resulting *WrapData* object (named *irisBox* in this particular example). This object contains all the information about the subsequent transformation operations as well as the original and transformed data. In this example, we showed how to access the data from the R object using the *data* attribute.

The PMML code output by the *pmml* package for the R commands listed above is shown in Figure 3 (*Header* and *DataDictionary* elements are omitted). As expected, it contains the *RegressionModel* element which describes the linear regression model as well as the *NormContinuous* element which describes the continuous normalization operation performed on the second data column or input field "Sepal.Width". This operation is described next.

```
<RegressionModel algorithmName="least squares"
  functionName="regression"
  modelName="Linear_Regression_Model">
  <MiningSchema>
    <MiningField name="Sepal.Length" usageType="predicted"/>
    <MiningField name="Sepal.Width" usageType="active"/>
    <MiningField name="Petal.Length" usageType="active"/>
    <MiningField name="Petal.Width" usageType="active"/>
  </MiningSchema>
  <Output>
    <OutputField feature="predictedValue"
      name="Predicted_Sepal.Length"
      dataType="double" optype="continuous"/>
  </Output>
  <LocalTransformations>
    <DerivedField dataType="double"
      name="derived_Sepal.Width" optype="continuous">
      <NormContinuous field="Sepal.Width">
        <LinearNorm norm="0" orig="2"/>
        <LinearNorm norm="1" orig="4.4"/>
      </NormContinuous>
    </DerivedField>
  </LocalTransformations>
  <RegressionTable intercept="1.85599749291755">
    <NumericPredictor coefficient="0.650837159313218"
      exponent="1" name="Sepal.Width"/>
    <NumericPredictor coefficient="0.709131959136729"
      exponent="1" name="Petal.Length"/>
    <NumericPredictor coefficient="-0.556482660167024"
      exponent="1" name="Petal.Width"/>
    <NumericPredictor coefficient="0.012"
      exponent="1" name="derived_Sepal.Width"/>
  </RegressionTable>
</RegressionModel>
```

Figure 3. PMML code output by the *pmml* package for transformation and model objects.

Note that the transformations object is an optional argument for the *pmml* function. Without it, the *pmml* function assumes that no transformations were applied to the data before modeling.

3.1 Continuous Normalization

Continuous normalization in PMML is represented by the element *NormContinous*. This element provides a general method for normalizing the input data for a continuous field from a given range to another designated range. For example, given that the data for input field *Input1* 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 *MinMaxXform* function of the *pmmlTransformations* package.

Given input field "InputField", the name of the derived field "OutputField", the desired minimum value that the derived field may have "lowLimit", the desired maximum value that the derived field may have "highLimit", and the desired value of the derived field if the input field value is missing "missVal", the *MinMaxXform* command is described using the format:

MinMaxXform(boxdata, xformInfo, mapMissingTo="missVal")

where *boxData* is the wrapped object and

xformInfo = "InputField->OutputField[lowLimit, highLimit]"

There are two ways input data fields can be referred to. The first is to use its column number; given the *data* attribute of the *boxdata* object, this would be the order in which the fields appear in the input data. This can be indicated in the format "column#". The second way is to simply refer to the desired input data field by its name.

The name of the transformed field is optional; if not provided, the derived field is given the name:

derived_ + {original_field_name}

Similarly, the low and high limit values are optional; they have the default values of "0" and "1" respectively. The value for *missingValue* is an optional parameter as well. If provided, it is used as the resulting value of the derived field if the value of the input field is missing.

If no input field names are provided, by default all numeric fields are transformed. Note that, in this case, a replacement value for missing input values cannot be specified; the same applies to the *lowLimit* and *highLimit* parameters.

We used the *MinMaxXform* function as part of the sequence of R commands described above. In it, the well-known Iris dataset is used as input and a linear transformation is performed on the input field "Sepal.Width". The example uses an operation where this field is transformed into a field named "derived_Sepal.Width". Note that this is the default name for the derived field. Also note that it assumes the default values for high and low limits, as implemented in the resulting PMML code shown in Figure 4.

Another common continuous normalization supported by the *pmmlTransformations* package is the z-score transformation. This is implemented by function *ZScoreXform*. Its follows the same format described here for function *MinMaxXform*.

3.2 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 support vector machines and neural network models. For this operation, the *pmmlTransformations* package provides the *NormDiscreteXform* function. It defines a new derived field for each possible value of a categorical field.

Given a categorical input field *InputField,* and *missVal*, the desired value of the derived field if the value of the input field is missing, the *NormDiscreteXform* command including all optional parameters is described using the format:

NormDiscreteXform(boxData, "InputField", mapMissingTo="missVal")

where *boxData* is the wrapped object.

As with other functions, the input variable can be referred to by name or by its column number. The output of the *NormDiscreteXform* function is a set of derived fields, one for each possible value of the input field. For example, given the values "val1", "val2", … for an input field named "InputField", the resulting derived fields are by default named "InputField_val1", "InputField_val2", …

The sequence of R commands below shows an example of how this function can be used in practice. Once again, we use the well known Iris dataset. In particular, we use field "Species" as input to the discrete normalization operation.

```
library(pmmlTransformations);
library(pmml);
data(iris);
irisBox <- WrapData(iris);
irisBox <- NormDiscreteXform(irisBox,inputVar="Species");
dataset <- irisBox$data[,-5];
irisLR <- lm(Sepal.Length~., data=dataset);
pmml(irisLr, transform=irisBox);
```

```
<RegressionModel modelName="Linear_Regression_Model"
functionName="regression" algorithmName="least squares">
 <MiningSchema>
  <MiningField name="Sepal.Length" usageType="predicted"/>
  <MiningField name="Sepal.Width" usageType="active"/>
  <MiningField name="Petal.Length" usageType="active"/>
  <MiningField name="Petal.Width" usageType="active"/>
  <MiningField name="Species" usageType="active"/>
 </MiningSchema>
 <Output>
  <OutputField name="Predicted_Sepal.Length"
    feature="predictedValue"/>
 </Output>
 <LocalTransformations>
  <DerivedField name="Species_setosa"
    dataType="double" optype="continuous">
   <NormDiscrete field="Species" value="setosa"/>
  </DerivedField>
  <DerivedField name="Species_versicolor"
    dataType="double" optype="continuous">
   <NormDiscrete field="Species" value="versicolor"/>
  </DerivedField>
  <DerivedField name="Species_virginica"
    dataType="double" optype="continuous">
   <NormDiscrete field="Species" value="virginica"/>
  </DerivedField>
 </LocalTransformations>
 <RegressionTable intercept="2.17126629215507">
  <NumericPredictor name="Sepal.Width"
    exponent="1" coefficient="0.495888938388551"/>
  <NumericPredictor name="Petal.Length"
    exponent="1" coefficient="0.829243912234806"/>
  <NumericPredictor name="Petal.Width"
    exponent="1" coefficient="-0.315155173326474"/>
  <NumericPredictor name="Species_setosa"
    exponent="1" coefficient="-0.723561957780729"/>
  <NumericPredictor name="Species_versicolor"
    exponent="1" coefficient="-1.02349781449083"/>
  <NumericPredictor name="Species_virginica"
    exponent="1" coefficient="0"/>
 </RegressionTable>
</RegressionModel>
```

Figure 4. PMML code output by the *pmml* package containing elements *NormDiscrete* and *RegressionModel*.

As a result, three derived fields will be created based on the values of the "Species" field. The new derived fields are:

1. "Species_setosa" such that it is "1" if input field "Species" equals "setosa", else "0"

2. "Species_versicolor" such that it is "1" if input field "Species" equals "versicolor", else 0
3. "Species_virginica" such that it is "1" if input field "Species" equals "virginica", else "0"

These are appended to the original dataset when the transformation is completed.

The PMML code output by the *pmml* function for the R commands listed above is shown in Figure 4 (as before, *Header* and *DataDictionary* elements are omitted). As expected, it contains the *RegressionModel* element, which describes the linear regression model as well as the *NormDiscrete* element, which describes the discrete normalization operation performed on the input field "Species".

3.3 Discretization

Discretization or binning in PMML is represented by the element *Discretize*. For a continuous field, a set of intervals is defined and if the input value falls within one of the intervals, a new value is assigned to the resulting derived field as defined for that interval. This mapping from a continuous to a discrete field can be many-to-one but not one-to-many. In other words, two intervals can be mapped to the same value but the same interval may not have more than one value. This implies that the intervals defined must be disjoint. There can be no overlap between two defined intervals. To generate the PMML code for such a transformation, we can simply use the *DiscretizeXform* function of the *pmmlTransformations* package.

Given a continuous input field "InField", which is to be discretized to various levels of a derived field "OutField", where the fields have the data types "InType" and "OutType", the *DiscretizeXform* command is described using the format:

DiscretizeXform (boxdata, xformInfo)

where *boxData* is the wrapped object and

xformInfo="[InField->OutField][InType>OutType]",
table="CSVFileName",
defaultValue="defVal",mapMissingTo="missingVal"

The data types of the input and output fields are optional. "CSVFileName" is the name of the CSV (Comma-Separated Value) file where the ranges of the continuous input field and resulting discrete values are defined. "defVal" and "missingVal" are the default and missing values respectively. The default value will be used if the value of "InField" does not lie in any of the ranges specified in the CSV file and the missing value will be used if the input value is missing.

The CSV file containing the input table is required and should not contain any row and column identifiers. Intervals are given by their left and right limits separated by the ":" character; if one is missing, the limit is considered to be infinite. The intervals themselves may be open or closed, as indicated by brackets or parenthesis respectively.

To illustrate the usage of the *DiscretizeXform* function, we created a CSV file named *Intervals.csv* containing the following information:

```
5],val1
(5:6],22
(6,val2
```

And use it as an argument for function *DiscretizeXform* in the following sequence of R commands:

```
library(pmmlTransformations);
library(pmml);
data(iris);
irisBox <- WrapData(iris);
irisBox <- DiscretizeXform(irisBox,
  xformInfo="[Sepal.Length -> derived_Sepal.Length]
  [double -> string]",
  table = "Intervals.csv", missingValue="0");
```

In this example, we can see that the first line of the CSV file establishes that if the value of the continuous input field "Sepal.Length" is less than or equal to 5, the derived field "derived_Sepal.Length" should have value "val1". Similarly, the second line indicates that if the input value is greater than 5 and less than or equal to 6, then the derived field should have value "22". Finally, the third line indicates that if the input value is greater than 6, then the derived field should have value "val2".

The PMML code output by the *pmml* function for the R commands listed above is shown in Figure 5 (as before, *Header* and *DataDictionary* elements are omitted). As expected, it contains the *RegressionModel* element, which describes the linear regression model as well as the *Discretize* element, which describes the discretization operation performed on input field "Sepal.Length".

```
<RegressionModel
  functionName="regression" algorithmName="least squares">
 <MiningSchema>
  <MiningField name="Sepal.Width" usageType="predicted"/>
  <MiningField name="Petal.Length" usageType="active"/>
  <MiningField name="Petal.Width" usageType="active"/>
  <MiningField name="Sepal.Length" usageType="active"/>
 </MiningSchema>
 <LocalTransformations>
  <DerivedField name="derived_Sepal.Length"
    dataType="string" optype="categorical">
   <Discretize field="Sepal.Length" mapMissingTo="0">
    <DiscretizeBin binValue="val1">
     <Interval closure="openClosed" rightMargin="5"/>
    </DiscretizeBin>
    <DiscretizeBin binValue="22">
     <Interval closure="openClosed"
       leftMargin="5" rightMargin="6"/>
    </DiscretizeBin>
    <DiscretizeBin binValue="val2">
     <Interval closure="openOpen" leftMargin="6"/>
    </DiscretizeBin>
   </Discretize>
  </DerivedField>
 </LocalTransformations>
 <RegressionTable intercept="4.00623161917765">
  <NumericPredictor name="Petal.Length"
    exponent="1" coefficient="-0.387348571527146"/>
  <NumericPredictor name="Petal.Width"
    exponent="1" coefficient="0.364309221201285"/>
  <CategoricalPredictor name="derived_Sepal.Length"
    value="22" coefficient="0"/>
  <CategoricalPredictor name="derived_Sepal.Length"
    value="val1" coefficient="-0.379996411767168"/>
  <CategoricalPredictor name="derived_Sepal.Length"
    value="val2" coefficient="0.371054150420373"/>
 </RegressionTable>
</RegressionModel>
```

Figure 5. PMML code output by the *pmml* package containing elements *Discretize* and *RegressionModel*.

3.4 Value Mapping

Value mapping in PMML is represented by the element *MapValues*. For the *pmmlTransformations* package, the map is defined in a CSV file, which serves as an argument to function *MapXform*.

Given a map from the combination of fields "InField1", "InField2", ... to the derived field "OutField", where the fields have the data types "InType1", "InType2", ... and "OutType", the *MapXform* command is described using the format:

MapXform(boxdata, xformInfo)

where *boxData* is the wrapped object and

xformInfo="[InField1, InField2, ... -> OutField][InType1, InType2, ... -> OutType]", table="CSVFileName", defaultValue="defVal", mapMissingTo="missingVal"

where *CSVFileName* is the name of the CSV file containing the map itself (a N to 1 map where N is greater or equal to 1) and the data types of the variables can be any of the ones defined in PMML including integer, double or string. "defVal" is the default value of the derived field and if any of the map input values are missing, "missingVal" is the resulting value assigned to the derived field.

While the CSV file is required, the input data type arguments as well as *defaultValue* and *mapMissingTo* are optional. The CSV file containing the map should not have any row and column identifiers, and the values given must be in the same order as in *xformInfo*. If the data types of the input fields are not given, they are derived from the transformation object itself. If this is not possible, the data type of the input fields is assumed to be of type "string".

The example below shows the sequence of R commands to apply the *MapXform* transformation on the Iris dataset:

```
library(pmmlTransformations);
library(pmml);
data(iris);
irisBox <- WrapData(iris);
irisBox <- MapXform(irisBox,
  xformInfo="[species ->
  derived_Species][string -> integer]",
  table = "MapSpecies.csv", missingValue="0");
```

As aforementioned, "Species" has three possible values: "setosa", "versicolor" and "virginica". For this example, this information, together with the values they map to is defined in a file named "MapSpecies.csv", as follows:

```
setosa,2
versicolor,4
virginica,1
```

As it indicates, we want to derive a value such that:

- If input field "Species" equals "setosa", then derived field "derived_Species" is set to "2"
- If input field "Species" equals "versicolor", then derived field "derived_Species" is set to "4"
- If input field "Species" equals "virginica", then derived field "derived_Species" is set to "1"
- If input field "Species" is missing, then derived field "derived_Species" is set to "0"

The PMML code output by the *pmml* function for the R commands listed above is shown in Figure 6 (as before, *Header* and *DataDictionary* elements are omitted). As expected, it contains the *RegressionModel* element, which describes the linear regression model as well as the *MapValues* element, which describes the mapping operation performed on input field "Species".

```
<RegressionModel
  functionName="regression" algorithmName="least squares">
 <MiningSchema>
  <MiningField name="Sepal.Length" usageType="predicted"/>
  <MiningField name="Sepal.Width" usageType="active"/>
  <MiningField name="Petal.Length" usageType="active"/>
  <MiningField name="Petal.Width" usageType="active"/>
  <MiningField name="Species" usageType="active"/>
 </MiningSchema>
 <LocalTransformations>
  <DerivedField name="derived_Species"
    dataType="double" optype="continuous">
   <MapValues mapMissingTo="0" outputColumn="output">
    <FieldColumnPair field="Species" column="input1"/>
    <InlineTable>
     <row>
      <input1>setosa</input1>
      <output>2</output>
     </row>
     <row>
      <input1>versicolor</input1>
      <output>4</output>
     </row>
     <row>
      <input1>virginica</input1>
      <output>1</output>
     </row>
    </InlineTable>
   </MapValues>
  </DerivedField>
 </LocalTransformations>
 <RegressionTable intercept="1.85337484653521">
  <NumericPredictor name="Sepal.Width"
    exponent="1" coefficient="0.651325180019908"/>
  <NumericPredictor name="Petal.Length"
    exponent="1" coefficient="0.708934446391052"/>
  <NumericPredictor name="Petal.Width"
    exponent="1" coefficient="-0.555791838494623"/>
  <NumericPredictor name="derived_Species"
    exponent="1" coefficient="0.000447570791813313"/>
 </RegressionTable>
</RegressionModel>
```

Figure 6. PMML code output by the *pmml* package containing elements *MapValues* and *RegressionModel*.

It should be noted that the transformations performed via the *pmmlTransformations* package are independent of the *pmml* package. That is, the transformed data can be accessed by any other function using the "data" attribute of the wrapped R object.

4. CONCLUSION

R is a popular and powerful tool for statistical data analysis and model building. Part of its power comes from the fact that users can contribute data mining capabilities to it in the form of external libraries or packages, which can then be used by anyone. Its vast number of community developers resulted in a myriad of packages capable of handling a wide variety of data mining techniques. The *pmml* package transforms obtained R model objects into dynamic assets that can deployed and put to use in any platform by representing them in PMML, de facto standard to represent data mining models.

This article describes another R package that supports PMML, the *pmmlTransformations* package. Statistical analysis of raw data frequently requires pre-processing of input fields before these can be used for the building of a predictive model. Given that data

transformations are not part of most modeling packages. The *pmmlTransformations* package fills this void by offering a variety of functions, which provide the user the capability to manipulate data in different ways prior to modeling.

The *pmmlTransformations* package works in tandem with the *pmml* package to provide a PMML representation of the model describing not just the modeling technique itself but also the data pre-processing steps. This adds to the power of R as a development tool since it makes the entire predictive solution available for deployment and execution in different production platforms, independently of R.

5. ACKNOWLEDGMENTS

We would like to thank our colleagues at Zementis for the support and feedback. We also would like to thank all the other DMG members for their commitment to making PMML a great language and standard. Finally, we would like to thank Graham Williams at Togaware for his invaluable support and feedback during the development of the *pmmlTransformations* package and consequently, all the required changes in the *pmml* package.

6. REFERENCES

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

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

[3] A. Guazzelli (2012). Predicting the Future, Part 4: Put a Predictive Solution to Work. IBM developerWorks website.

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

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

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

[7] K. K. Das, E. Fratkin, A. Gorajek, K. Stathatos, M. Gajjar (2011). Massively Parallel In-Database Predictions using PMML. In *Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[8] D. Morent, K. Stathatos, W. Lin, M. Berthold (2011). Comprehensive PMML Preprocessing in KNIME. In *Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[9] K. Bache & M. Lichman (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science