SESUG Paper SD-17-2017

PMML & SAS®: An Introduction to PMML and PROC PSCORE

Andrew Gannon, The Financial Risk Group

ABSTRACT

The Predictive Model Markup Language (PMML) is becoming the standard write-up language for modeling. Based in XML, it can easily be used by many different applications including R, SAS®, and SPSS - among others. SAS can integrate PMML by invoking the `PROC PSCORE` process, and can be exported through Enterprise Miner. This document serves to introduce the PMML language and its terminology - as well as an introduction to the `PROC PSCORE` method in SAS. This paper should serve only as an advertisement to PMML and an introduction to using PMML with SAS.

INTRODUCTION

The Predictive Model Markup Language, PMML, was developed by the Data Mining Group (DMG), which is a consortium of companies and software service providers. PMML is the premier model language because many software suites can import and/or export PMML code. From the standard coding languages of R and SAS to more business oriented application such as SAP, PMML is becoming the new model standard. Developing model code in PMML makes it far more efficient when several applications may use that model. Instead of developing the model in both SAS and Oracle Data Mining code, you can do so with PMML and then easily invoke that code in both the Oracle Data Mining and SAS environment. PMML follows the same structure as XML, and has its own tags for the development of models that are held in schema files located on the DMG's website. PMML can be used to construct tree, cluster, regression, and scorecard models, among others.

THE PMML DOCUMENT

To understand PMML it is important to know Extensible Markup Language or XML. It is a markup-language that is easily read in its raw form. It consists of tags and elements. Elements typically begin and end with a tag. Tags begin and end with angle-brackets to enclose a text. Since tags surround elements, then they come in pairs, typically. Closing tags being with '/' inside the brackets. Additional tags can be opened within a tag. Below is an example of an element enclosed in tags.

```
<Timestamp> 01/01/2017 15:06:46 </Timestamp>
```

STARTING A PMML DOCUMENT - HEADER

To begin the PMML document, you start a PMML tag and version (currently 4.2),

```
<PMML version="4.2" xmlns="http://www.dmg.org/PMML-4_2">
```

The location corresponding to the 'xmlns' is that of the schema file that defines the tags and PMML definitions that can be used. Viewing that document is a great way to learn the in's and out's PMML. It is a good practice to close this tag now, and not forget to at the end. All of the code that is included in the model will be included within the PMML tag. The first tag group after the opening PMML tag is the Header tag. It contains a *copyright*= and *description*= inputs as seen below. These two fields can contain any set of characters and values.

```
<Header copyright="(c) FRG, Cary, NC, USA" description="Regression"/>
```

Notice that the tag is ended with a backslash /. This is another way to end a tag if it is all contained within the opening and there are no other tags subset within it. This method saves time and space within the model. Within the header tier there are several other options to include like the *Timestamp, Application, and Extension tags*. Although they are not required for the program to work properly, they are recommended for documentation. Below is a full example of a PMML header.

```
<Header copyright="(c) FRG, Cary, NC, USA" description="Regression">
      <Extension name="user" value="AGannon" extender="PMML"/>
```

```
      <Application name="PMML" version="1.6.2"/>
      <Timestamp>01JAN2017</Timestamp>
</Header>
```

The space before the tags in the header clause are for readability. They are not necessary for the code to execute properly, but make it easier to read and to work with. Below is how to write commented-out text to your program. This text is in the program but will not be evaluated during the execution of the program.

```
<!-- Text enclosed with this structure is a comment and is not evaluated by
the program.  Notice the exclamation point and dashes (req'd).  -->
```

DATADICTIONARY CLAUSE

In order to use a variable within PMML, the data must be read in. For PMML, this is done within the *DataDictionary* clause. Character, numeric, and categorical data are available options within this clause. Only read-in data and predicted data needs to be within the DataDictionary, data that is created from these variables do not need to be stated (see *ratio* in LocalTransformation section below). Within the DataDictionary tab are *DataField* tags, each representing a variable that is being read in. The DataField clause has three (3) main components: *name=, optype=, & dataType=*. The optype= option states what the data variable type is. The dataType= option states whether it is string or numeric data (numeric data is represented by "double").

Categorical variables require extra data for each possible choice, these are tagged with the *Value* clause. Any variables that are to be used, that are not created within the PMML document must be expressed in the DataDictionary. Below is an example of a DataDictionary with numeric variables and a character categorical variable.

```
<DataDictionary>
   <DataField name="passing_yds" optype="continuous" dataType="double"/>
   <DataField name="rushing_yds" optype="continuous" dataType="double"/>
   <DataField name="touchdowns" optype="continuous" dataType="double"/>
   <DataField name="qb_class" optype="categorical" dataType="string">
      <Value value="single_threat"/>
      <Value value="dual_threat"/>
   </DataField>
   <DataField name="interceptions" optype="continuous" dataType="double"/>
</DataDictionary>
```

The variable "qb_class" can thus take two options, either *single_threat* or *dual_threat* - both character strings. Again, indenting tiered text makes it easier to work with and read and is highly recommended in PMML.

REGRESSIONMODEL CLAUSE

Once the header and DataDictionary have been defined the model can be developed. For our purposes, we will work with the PMML Regression model in this document. Each model has its own respective tag, for regression it is RegressionModel with three main components: *modelName=, functionName=, & algorithmName=.* Like the DataField's from before, these statements are inside of the tag. Below is an example of the opening tag for a regression model:

```
<RegressionModel modelName="Reg" functionName="regression" algorithmName="linearRegression">
```

With this, the regression model can begin. One PMML model can contain many models. Each model will use some (or all) of the variables in the DataDictionary. In order for a model to utilize variables from the DataDictionary they must be read into the individual model with a *MiningSchema* statement. This tag will contain tags analogous to the DataField tags called *MiningField*'s. Any outside variable that is not created within the model must be read into the model through the MiningSchema tag for the model to run

successfully. An invalid value statement can also be added to replace the value if it is missing, null, or equal to a particular value. Below is an example of a MiningSchema tag with data being read in:

```
<MiningSchema>
   <MiningField name="passing_yds" usageType="active"/>
   <MiningField name="rushing_yds" usageType="active"/>
   <MiningField name="touchdowns" usageType="active"/>
   <MiningField name="qb_class" usageType="active"/>
   <MiningField name="interceptions" usageType="predicted"/>
</MiningSchema>
```

The data is now available for use within the model. If you manipulate or create new data for the model then the *LocalTransformations* tag can be used. LocalTransformations are not required. If all the data that is read in is refined and ready for the regression model, then no LocalTransformations are necessary. With this tag addition, multiplication, division, if-then statements, and min-max options are available for use to create new data. There are several components to the LocalTransformation tag. First, you define the new variable with a *DerivedField* followed by the transformation. Below is an example of the code.

```
<LocalTransformation>
   <DerivedField dataType="double" name="ratio" optype="continuous">
      <Apply function="/">
         <FieldRef field="touchdowns"/>
         <FieldRef field="passing_yds"/>
      </Apply>
   </DerivedField>
</LocalTransformations>
```

The above code creates a new variable "ratio" that is assigned a value of *touchdowns* divided by *passing_yds*. Notice the Apply function and how the statements that follow will be used with the function. The apply/function option can be used for many other basic functions such as min, max, addition, subtraction, sum, average, etc... Ratio is now available to use later in the regression model. Finally, the regression model can be completed with the *RegressionTable* tag, which has a required option for the intercept of the regression. It will contain tiered tags for variables of the regression equation. Each of these tags will contain the exponent and coefficient value. If it is a categorical variable there will be an option for each possible value of the variable. As seen in the example, the created value can be used in the RegressionTable portion of the program. Below is an example of a RegressionTable:

```
<RegressionTable intercept=".15">
   <NumericPredictor name="passing_yds" exponent="1" coefficient=".0035"/>
   <NumericPredictor name="rushing_yds" exponent="1" coefficient="0.001"/>
   <NumericPredictor name="touchdowns" exponent="1" coefficient="-.02"/>
   <NumericPredictor name="ratio" exponent="1" coefficient="1"/>
   <CategoricalPredictor name="class" value="single_threat" coefficient="1.2"/>
   <CategoricalPredictor name="class" value="dual_threat" coefficient=".8"/>
</RegressionTable>

<!-- Above RegressionTable is equivalent to the equation below -->
```

f(x) = passing_yds(.0035) + rushing_yds*(.001) + touchdowns*(-.02) + ratio*(1) + (1.2 or .8) + (.15)*

With this, the regression model is complete upon closing the RegressionModel tag. If the entire program is complete then simply close the PMML tag that was opened at the very beginning. If there is still more, continue to add RegressionModel tags.

RUNNING PMML IN SAS®

There are two components to running the PMML code in other programs. First, the PMML needs to be read into the respective programming language and converted to the proper syntax. SAS, this is done internally with the *PROC PSCORE* option (It can also simply be uploaded with SAS Enterprise Miner). During the PROC PSCORE procedure, the PMML is read in, converted to SAS code, and saved as a SAS file to the location that is specified. Secondly, the code that has been converted and stored must then be called in and initialized. This is done using as simple *%Include* statement linking to the saved file with the SAS converted PMML code. Below is example of both parts of reading in the PMML code in SAS:

```
PROC PSCORE
   PMML File = "C:\PMML\QB_Stats.XML"
   DS File = "C:\PMML\QB_Stats.SAS"
   ;
RUN;


DATA Work.Regression_QB_Stats;
   SET work.QB_Data;
   %INCLUDE "C:\PMML\QB_Stats.SAS";
RUN;
```

Upon running the PSCORE invocation, the log does not output much about what is happening. If there is an error with either the syntax in SAS or with the PMML being passed it will error, as expected. Otherwise it will only output timing information if successful. See a successful log of the proc pscore invocation below in figure 1.

```
14   /* Run SAS's PSCORE to convert PMML to SAS code */
15   PROC PSCORE
16      PMML File = "C:\PMML\QB_Stats.XML"
17      DS File = "C:\PMML\QB_Stats.SAS";
18   RUN;

NOTE: PROCEDURE PSCORE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Figure 1.

Notice that the fileref location in the DATA step is the same location in the PROC PSCORE step. With the proper dataset and this code, SAS - as well as SAP, SPSS, and other programs with corresponding code - can run the model. The output of this particular code, given a valid dataset is a new dataset with an additional column data called *PSCR_T_0_interceptions* with data filled in based on the regression. The 'Ratio' object that was created in the LocalTransformation portion of the PMML is dropped, as it is intended to be used in our regression equation only and not output.

The new variables that are added to the newly created dataset are attached at the end of the current variables. You can calculate their values by hand, using the same formula that was developed in the PMML document to verify that the SAS system has correctly done its job. You can also verify by checking on the new SAS file that the proc pscore created. Inside this file is the fully converted PMML code, now in SAS form.

The PROC PSCORE takes in the PMML and converts it to pure SAS code in the location that is specified. This code cannot run on its own because it does not contain any data or proc steps. It is simply the equations that are specified. Below is an example of the output SAS code post-PMML conversion:

```
...
"ratio"n = "touchdowns"n / " passing_yds"n;
if( missing( "ratio"n )) then do;
   PSCR_WARN = 1 ;
end;
...
```

Using the %INCLUDE allows us to nest these equations within the data step to be executed with the given dataset. (View the entire output in Appendix D). Notice that if the 'ratio' variable does not get set, then the SAS code sets a warning flag in PSCR_WARN = 1. It computes this warning after reading in each variable, and therefore can be set at a number of locations within the code. Just before SAS begins to execute it uses GOTO to exit if PSCR_WARN is not 0 (thus exiting making no changes if any of the variables are missing). Like the 'ratio' variable, the warning variables also are dropped before final output. To view full output of the proc pscore invocation with the PMML file, see Appendix D.

CONCLUSION

PMML is growing in use and application including use with: SAS, SPSS, R, Netezza, SAP, Hadoop, MS SQL Server, KNIME, Apache Spark, Teradata, etc… For more information on PMML the Data Mining Group's website, http://dmg.org/pmml/v4-3/GeneralStructure.html, is the best resource. There is also a very helpful book on PMML, *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics* by Guazzelli et al.

What follows after the references section in Appendix A and Appendix B are the PMML code and SAS code that can be directly copied and used. The dataset that is used for the SAS code is included within the SAS code itself. Appendix C is a how-to for using the example code as well as the output dataset. This is a compilation of all the example code from this document and is for linear regression model. Finally, Appendix D is the output of the DS File created by the PROC PSCORE invocation.

REFERENCES

Guazzelli, Alex et al. Jan 31, 2012. *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics*.

In-Database Analytics Developer's Guide. IBM Netezza Analytics Release 3.2.0.0. Nov. 20, 2014. Accessed on Jan 31, 2017. IBM.

Weiss, Albrecht. Jan 31, 2017. *Integrating Real-Time Predictive Analytics into SAP Applications.* Dec. 30, 2009. SAP Community Network – SAP AG.

Guazzelli, Alex et al. May 1, 2009. "PMML: An Open Standard for Sharing Models." *R-Journal.* Vol. 1:60-65

“PMML Standard” Data Mining Group. Jan 15, 2017. Available at http://dmg.org/pmml/v4-3/GeneralStructure.html.

"Model Types Supporting PMML" IBM Knowledge Center. Jan 15, 2017. Available at https://www.ibm.com/support/knowledgecenter/SS3RA7_15.0.0/com.ibm.spss.modeler.help/models_pmml_modeltypes.htm

RECOMMENDED READING

- *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics*.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Andrew Gannon
The Financial Risk Group
(336) 684-4031
andrew.gannon@frgrisk.com
www.frgrisk.com

APPENDIX A – EXAMPLE PMML CODE

```
<PMML version="4.2" xmlns="http://www.dmg.org/PMML-4 2">

<Header copyright="(c) FRG, Cary, NC, USA" description="Regression">
   <Extension name="user" value="AGannon" extender="PMML"/>
   <Application name="PMML" version="1.6.2"/>
   <Timestamp>01JAN2016</Timestamp>
</Header>

<DataDictionary numberOfFields="5">
   <DataField name="passing_yds" optype="continuous" dataType="double"/>
   <DataField name="rushing_yds" optype="continuous" dataType="double"/>
   <DataField name="touchdowns" optype="continuous" dataType="double"/>
   <DataField name="qb_class" optype="categorical" dataType="string"/>
      Value value="single_threat"/>
      Value value="dual_threat"/>
   </DataField>
   <DataField name="interceptions" optype="continuous" dataType="integer"/>
</DataDictionary>

<RegressionModel modelName="Reg" functionName="regression" algorithmName="linearRegression">

   <MiningSchema>
      <MiningField name="passing_yds" usageType="active"/>
      <MiningField name="rushing_yds" usageType="active"/>
      <MiningField name="touchdowns" usageType="active"/>
      <MiningField name="qb_class" usageType="active"/>
      <MiningField name="interceptions" usageType="predicted"/>
   </MiningSchema>

   <LocalTransformation>
      <DerivedField dataType="double" name="ratio" optype="continuous">
         <Apply function="/">
            <FieldRef field="touchdowns"/>
            <FieldRef field="passing_yds"/>
         </Apply>
      </DerivedField>
   </LocalTransformations>

   <RegressionTable intercept=".15">
      <NumericPredictor name="passing_yds" exponent="1" coefficient="0.0035"/>
      <NumericPredictor name="rushing_yds" exponent="1" coefficient="0.001"/>
      <NumericPredictor name="touchdowns" exponent="1" coefficient="-0.02"/>
      <NumericPredictor name="ratio" exponent="1" coefficient="1"/>
      <CategoricalPredictor name="qb_class" value="single_threat" coefficient="1.2"/>
      <CategoricalPredictor name="qb_class" value="dual_threat" coefficient="0.8"/>
   </RegressionTable>

</RegressionModel>

</PMML>
```

APPENDIX B – EXAMPLE SAS CODE

```
/* Create mock dataset */
DATA work.qb_stats;
   length class $13;
   input passing_yds rushing_yds
         touchdowns class $;
DATALINES;
212 85 3 dual_threat
423 -3 4 single_threat
326 46 3 dual_threat
390 3 2 single_threat
462 -1 5 single_threat
;

/* Run SAS's PSCORE to convert PMML to SAS code */
PROC PSCORE
   PMML File = "C:\PMML\QB_Stats.XML"
   DS File = "C:\PMML\QB_Stats.SAS";
RUN;

/* Use the generated SAS with mock data */
DATA WORK.regression_qb_stats;
   SET work.qb_stats;
   %INCLUDE "C:\PMML\QB_Stats.SAS";
RUN;

/* View the output */
PROC PRINT DATA=WORK.regression_qb_stats;
RUN;
```

APPENDIX C – USING THE SAMPLE CODE

To use the example code in Appendix A and Appendix B, follow these steps:

1. Copy the PMML code (appendix a) into a text editor and save it to a new folder, 'PMML' (in the c-drive) as **QB_Stats.xml**.
2. Open a SAS session and copy the SAS code (appendix b) into the session. This code generates an example dataset, pulls in the PMML (.xml) file, generates a new file with SAS code, and runs the code with the created dataset. The end result is a new dataset.

The dataset that is created at the beginning of the SAS session from datalines:

Obs	qb_class	passing_yds	rushing_yds	touchdowns
1	dual_threat	212	85	3
2	single_threat	423	-3	4
3	dual_threat	326	46	3
4	single_threat	390	3	2
5	single_threat	462	-1	5

The dataset that is created after the %include with the PMML is:

Obs	qb_class	passing_yds	rushing_yds	touchdowns	PSCR_T_0_interceptions
1	dual_threat	212	85	3	1.73115
2	single_threat	423	-3	4	2.75696
3	dual_threat	326	46	3	2.08620
4	single_threat	390	3	2	2.68313
5	single_threat	462	-1	5	2.87682

APPENDIX D – SAS FILE CREATED BY PROC PSCORE

When PROC PSCORE executes it creates a new SAS file in the specified directory. It contains a header with various runtime information. Below is the output from the examples PROC PSCORE procedure, minus some of the header information:

```
/*******************************************/;
* PSCORE TIMESTAMP: 2017-15-9 11:11:40.38 ;
* SAS VERSION: 9.04.01M3P062415 ;
* SAS HOSTNAME: AndrewGannon ;
* SAS ENCODING: wlatin1 ;
* SAS USER: agannon ;
* SAS LOCALE: EN_US ;
* PMML Path: C:\PMML\QB_Stats.XML ;
* MODEL TYPE: RegressionModel ;
* MODEL FUNCTION NAME: Regression ;
/*******************************************/;
if missing("passing_yds"n) then do;
    PSCR_WARN = 1;
end;
if missing("rushing_yds"n) then do;
    PSCR_WARN = 1;
end;
if missing("touchdowns"n) then do;
    PSCR_WARN = 1;
end;
if missing("qb_class"n) then do;
    PSCR_WARN = 1;
end;
else do;
    "PSCR_AP0"n = "qb_class"n;
    if ( "PSCR_AP0"n not in ( 'single_threat', 'dual_threat'  )  ) then do;

        PSCR_WARN = 1;
    end;
end;

label "PSCR_T_0_interceptions"n = 'PSCR_T_0_interceptions';
"ratio"n = "touchdowns"n / "passing_yds"n;
if( missing( "ratio"n )) then do;
    PSCR_WARN = 1;
end;
if (PSCR_WARN) then do;
    goto PSCR_EXIT ;
end ;

"PSCR_T_0_interceptions"n = 0 ;
"PSCR_T_0_interceptions"n = "PSCR_T_0_interceptions"n + ( 0.0035 ) * "passing_yds"n ;
"PSCR_T_0_interceptions"n = "PSCR_T_0_interceptions"n + ( 0.001 ) * "rushing_yds"n ;
"PSCR_T_0_interceptions"n = "PSCR_T_0_interceptions"n + ( -0.02 ) * "touchdowns"n ;
"PSCR_T_0_interceptions"n = "PSCR_T_0_interceptions"n + "ratio"n ;
if ("PSCR_AP0"n = 'single_threat') then "PSCR_T_0_interceptions"n =
   "PSCR_T_0_interceptions"n + ( 1.2 );
else if ("PSCR_AP0"n = 'dual_threat') then "PSCR_T_0_interceptions"n =
   "PSCR_T_0_interceptions"n + ( 0.8 );
"PSCR_T_0_interceptions"n = "PSCR_T_0_interceptions"n + 0.15 ;
PSCR_EXIT :

drop
 "ratio"n "PSCR_AP0"n PSCR_WARN;
```