Article from

Predictive Analytics & Futurism News

August 2018
Issue 18

Introduction to PMML in R

By Jeff Heaton

Predictive Model Markup Language (PMML)[1] is an Extensible Markup Language (XML)–based predictive model interchange format originally introduced by Dr. Robert Lee Grossman, who was at that time the director of the National Center for Data Mining at the University of Illinois at Chicago. Models produced in R, Python and other platforms can be exported to PMML. Once in PMML, these models can be executed on a variety of other platforms to produce scores.

A platform's PMML capabilities are described as being a consumer or a producer. Some platforms are both producers and consumers; however, most only support one side. For example, the R programming language can function as a producer, but not a consumer. This means that a random forest that was trained in R can be exported to PMML. However, the PMML saved by R cannot be loaded by R. Because of this, PMML is not a sort of general purpose file format. The primary purpose of PMML is to allow a trained model to be exported from a development language, such as R, Python or another language to be executed on a production language such as Java or Scala. In this way, PMML is more of an export format for deployment. The PMML website contains a list of what platforms are producers and consumers.[2]

> The primary purpose of PMML is to allow a trained model to be exported from a development language.

PMML CAPABILITIES

Once a model has been properly trained, it is desirable to save the state of that model. If the model is not saved, then it will be necessary to retrain each time the model is needed. Such retraining is undesirable on several levels. Firstly, it might have taken many hours of computer runtime to have trained that model. Secondly, there is a stochastic element to the training of many models. Saving the model's internal state is often the only way to reproduce the results of a particular model. Modeling frameworks provide a means of saving the state of your model. The model's state is whatever the model needs to produce a score. For a GLM the state includes the coefficients, intercept and choice of link function. For a random forest, the state would include the tree structure and any values used to calculate the score. Programming languages, such as R and Python provide a means of storing this model state. R stores these models to RData files and Python uses the Pickle file format.

It might be tempting to think of PMML as another file format to store your model in, similar to RData or Pickle. However, this is not exactly the case. I do not suggest that you use

PMML as a replacement for Pickle or RData. One obvious problem is that R only has the ability to write PMML, not read it—at least with the most popular free PMML libraries. Because the conversion to PMML might be a one-way trip for many programming languages, PMML is not a desirable alternative to that language's native format. Another difference between PMML and RData/Pickle is that preprocessing and ensemble information is encoded into PMML.

Your data will likely require some preprocessing before they are sent to the model. Continuous values might be normalized to a z-score. Categorical values might be encoded as dummy variables. When a model is stored as a RData/Pickle file, this encoding is not saved as part of the file. A PMML file attempts to encode the entire pipeline of data processing for your model. This includes common preprocessing steps, such as normalization, dummy variables, and dealing with missing values. PMML can also store the pipeline used to ensemble multiple models together. Because PMML focuses on encoding the entire pipeline, PMML is primarily a storage format for deployment. Once your model's pipeline is encoded into PMML, it can be deployed with a number of different open source and commercial products.

One popular open source deployment package for PMML is OpenScoring.[3] The OpenScoring framework can deploy PMML files as restful web services. These web services can be accessed by other programs, even those that are outside of your company. All communication with your deployed web service occurs using the JavaScript Object Notation (JSON). This allows other applications, written in nearly any programming language, to interact with your model. The programming language that you originally produced the PMML from is not important.

Because PMML is a standard, it requires that each model type that you seek to use to be covered by the PMML standard. Because of this, you might not have access to the latest models or new features from existing model types. However, unless you are producing models using bleeding edge technology, this is often not a problem. For example, the list of supported models for the R programming languages include kNN, Mining Models, Regression Models, General Regression Models (including Cox), Neural Networks, Decision Trees, Clustering Models, Association Rules, Support Vector Machines, Multinomial Logistic Regression, Random Forest, Random Survival Forest, and Naïve Bayes Classifiers.[4]

EXAMPLE: EXPORTING A RANDOM FOREST IN R

In this section, a sample R script to produce PMML is examined. This code, along with the resulting PMML, can be found at the author's GitHub page.[5] This example creates a random forest and trains it against Fisher's Iris Dataset.[6] The example code is provided here:

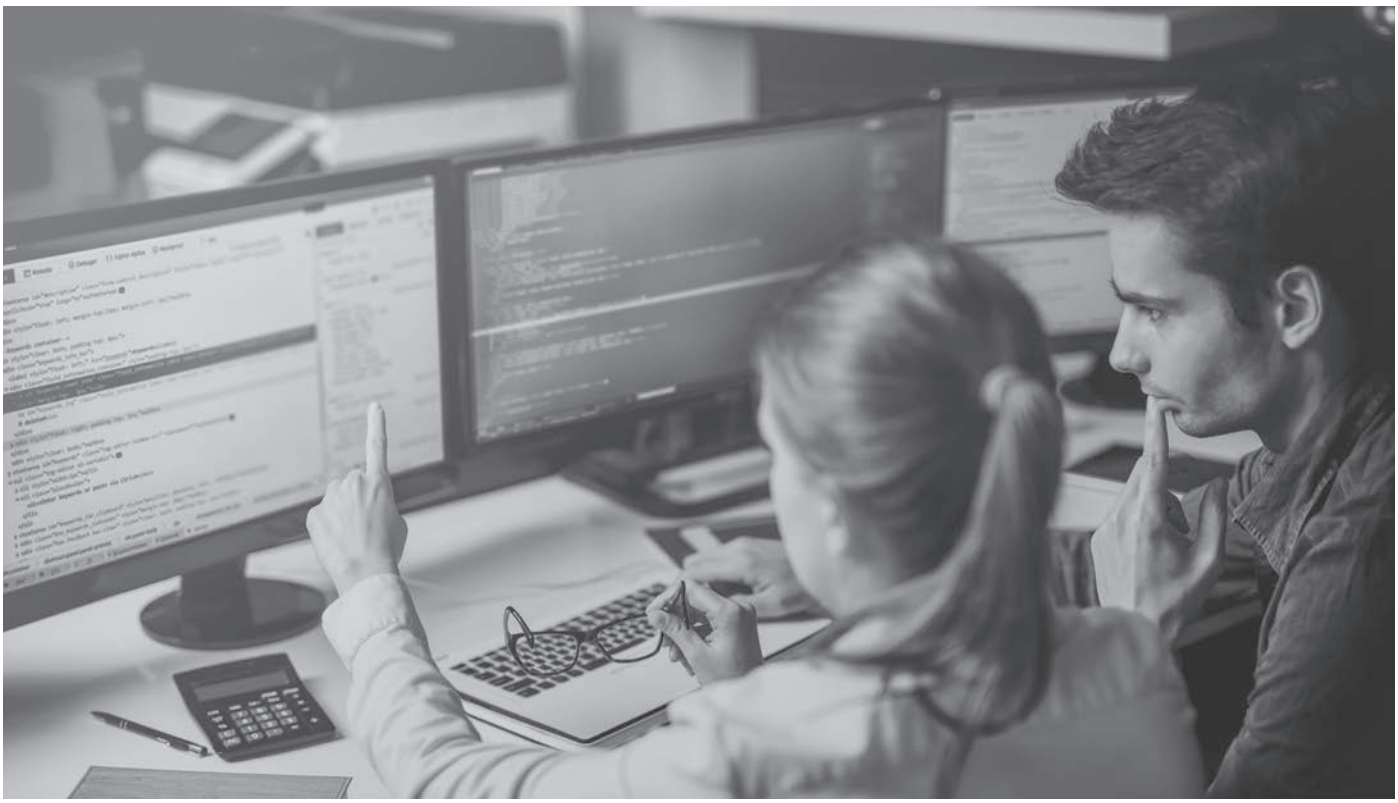
```
# Libraries
library(randomForest)
library(XML)
library(pmml)

# data to build model on
data(iris)

# train a model on a 75-25 split between
training and validation
z <- sample(2,nrow(iris),replace=TRUE,
prob=c(0.75,0.25))
trainData <- iris[z==1,]
testData <- iris[z==2,]

# train model
rf <- randomForest(Species~.,data=trainData,
                   ntree=100,proximity=TRUE)
table(predict(rf),trainData$Species)

# convertto pmml
pmml <- pmml(iris_rf,name="Iris Random
Forest",data=iris_rf)

# save PMML XML
saveXML(iris_rf.pmml,"iris.pmml")
```

The random forest is created with the common R library named simply RandomForest. Once the random forest is trained, it is encoded to PMML using the R PMML library that can be obtained through the Comprehensive R Archive Network (CRAN). Once the model has been encoded to PMML, it can be saved to a file with the R XML library.

The entire PMML file is verbose and lengthy. While the file is not reproduced here, it can be viewed at the author's GitHub repository.

Now that the random forest has been saved to a PMML file, it can be deployed as a restful web service with a PMML server, such as OpenScoring. This allows other applications to send JSON, such as the following, to receive an iris prediction.

```
{
"petal-width": 1.1,
"petal-length": 2.2,
"sepal-width": 3.3,
"sepal-length": 4.4
}
```



A result from the model might be as follows:

```
{
“prediction”: “Iris-versicolor”
“confidence”: .83
}
```

For a complete guide to setting up a restful web service to score PMML refer to the URL previously given for OpenScoring.

CONCLUSION

PMML is a standard file format that is typically used to encode models for deployment. The standard format of PMML allows model deployment platforms to be designed without consideration to the original language that the data scientist chose to implement the model in. If you seek to deploy a model with PMML it is important to ensure that the model type that you make use of is supported by the PMML client that you will ultimately deploy your model on. ■

Jeff Heaton, Ph.D. is lead data scientist, RGA Reinsurance Company, in Chesterfield, Mo. He can be reached at *jheaton@rgare.com.*

ENDNOTES

1 *http://dmg.org/pmml/v4-3/GeneralStructure.html*
2 *http://dmg.org/pmml/products.html*
3 *https://github.com/openscoring/openscoring*
4 *http://dmg.org/pmml/products.html*
5 *https://github.com/jeffheaton/present/tree/master/SOA/paf-newsletter/2018/pmml*
6 *https://en.wikipedia.org/wiki/Iris_flower_data_set*