Extending the Naïve Bayes Model Element in PMML: Adding Support for Continuous Input Variables

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

The Predictive Model Markup Language (PMML) is the de facto standard to represent data mining and predictive analytic models. With PMML, one can easily share a predictive solution among PMML-compliant applications and systems.

PMML as a standard has evolved significantly over the years. PMML 4.1, the language's latest version represents a major leap forward in terms of its ability to represent data post-processing and multiple models. It also provides entirely new model elements for supporting Scorecards and K-Nearest Neighbors. The same is no exception for PMML 4.2, currently being worked on by the Data Mining Group (DMG), the body responsible for maintaining and advancing the PMML standard. PMML 4.2 is bound to offer new elements and increased capabilities. This article describes one of such improvement. In particular, it proposes extending the existing model element for Naïve Bayes Classifiers to support continuous input fields.

The R Project is a popular choice for data miners to analyze and build predictive models. Naïve Bayes is just one of a myriad of model types supported by R. The R *e1071* package provides a *naiveBayes* function to build Naïve Bayes Models using categorical as well as continuous fields. The R *pmml* package has been recently extended to allow for the export of PMML code for objects built with the *naiveBayes* function. For now, it includes a PMML *Extension* element for continuous fields, but with the release of PMML 4.2, the support will be standardized. This article describes this process in view of our proposal to extend the current model element for Naïve Bayes Models.

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, Naïve Bayes Models, The R Project.

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 for data processing and model. For example, the schema for a Neural Network Model specifies elements and attributes for defining neural layers, neurons and connection weights. The same is true for all other model elements, including the element for Naïve Bayes Models, which specifies all the constructs necessary for representing a classifier using categorical input fields. Currently, however, support for continuous input fields is lacking.

PMML is a very mature and refined language; with the initial version 0.7 released back in 1997 and the latest, version 4.1, in December 2011. For each new version, new elements are added to support new functionality. For example, version 4.1 added support for K-Nearest Neighbors, Baseline Models and Scorecards [4]. At the same time, existing elements are revised to include new use-cases such as extending the Support Vector Machine element to allow for multi-class classification. It is in this context that we propose here an extension to the Naïve Bayes Model element to support continuous input fields.

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. A major attraction of R is its extensibility. Users can write their own packages containing functions and analysis tools, which can then be uploaded into a common repository, the Comprehensive R Archive Network (CRAN), and used by others. Available in CRAN is the *pmml* package, which is used for exporting predictive models built in R into PMML [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].

Also available in CRAN is the *e1071* package. It provides a function called *naiveBayes* to build Naïve Bayes Models using categorical as well as continuous input fields. The R *pmml* package has been recently extended to allow for the export of

PMML code for objects built with the *naiveBayes* function. In this way, besides proposing an extension to the PMML Naïve Bayes Model element, this paper also describes how the R *pmml* package outputs PMML for Naïve Bayes model objects.

We start by presenting the theory behind Naïve Bayes Models in Section 2. It shows the formulation necessary to deal with categorical as well as continuous input fields. Section 3 describes the Naïve Bayes element in PMML 4.1, the latest version of the standard. The proposed extensions to this element for representing continuous fields are described in Section 4. Section 5 shows how R offers support for building Naïve Bayes Models with the *e1071* package and to export them in PMML with the *pmml* package. Section 6 describes the scoring procedure once categorical and continuous input fields are used as predictors. Finally, Section 7 offers a conclusion.

2. Naïve Bayes

The Bayes theorem gives an expression for the inversion of the conditional probability. In other words, it relates the probability of an event given certain conditions to the probability of the conditions given the event. This may be expressed mathematically as shown below. Given an event E and a set of conditions $C_1,C_2,...,C_N$, the probability of the event taking place given a set of conditions is:

$$P(E \mid C_1,...,C_N) = P(E)\, P(C_1,...,C_N \mid E) / P(C_1,...,C_N)$$

If we "naïvely" assume that the conditions are independent of each other, then

$$P(C_1,...,C_N \mid E) = P(C_1 \mid E) * P(C_2 \mid E) *...* P(C_N \mid E)$$

Therefore if an event E has possible categorical values $E_1,...,E_M$, the naïve Bayes theorem states that the probability of the event E_i taking place given that the set of conditions $C_1,...,C_N$ are true is:

$$P(E_i \mid C_1,...,C_N) = P(E_i)\, \Pi_{j=1,...,N} P(C_j \mid E_i) / P(C_1,...,C_N)$$

$$= L_i / \Sigma_{k=1,...,M} L_k$$

where the likelihood $L_i = P(E_i)\, \Pi_{j=1,...,N} P(C_j \mid E_i)$

Note that the common factor $P(C_1,...,C_N)$ was cancelled out in the normalization. To apply this model, we must now work out how to calculate the probabilities required given a training data set. The events are a set of discrete values so that the probability of a particular event taking place can be calculated by the number of times each possible event category appears in the training dataset. If we denote the number of times a particular event category E_i takes place in the training dataset by $Count[E_i]$, then

$$P(E_i) = Count[E_i] / (\Sigma_{j=1,...,M} Count[E_j])$$

The second and last probability value to be calculated is the probability of a condition having had taken place given that an event occurred. If the condition is a set of discrete values as well, the probability may be calculated in a similar way. If the number of data points in the training data where both the condition C_j and event E_i occurred is denoted by $Count[C_jE_i]$, then

$$P(C_j \mid E_i) = Count[C_jE_i] / \ Count[E_i]$$

On the other hand, if the condition is a continuous field, we do not have the counts of the number of times that condition occurred in the training dataset; we only have a range of values. In this case, the probability of a condition existing given that an event has occurred can be represented by a continuous probability distribution. If this probability function is known, we can simply use it to calculate the desired probability for a given input value. While in principle many different probability functions can be used, the *e1071* package implements the Gaussian probability distribution; the computation of which requires knowledge of the mean and variance of a particular field. We can use the training dataset to calculate the desired mean μ and the variance σ^2.

To summarize then, the probability of a possible event E_i occurring given a set of conditions $C_j=1...M$ can be represented as follows:

$$P(E_i) = L_i / \Sigma_{k=1,...,M} L_k$$

Given A categorical and B continuous conditional fields, we have

$$L_i = P(E_i)\, \Pi_{j=1,...,A} P(C_j \mid E_i)\, \Pi_{j=1,...,B} P(C_j \mid E_i)$$

For the A categorical fields, $P(C_j \mid E_i)$ can be computed from the counts data as described earlier. For the B continuous fields, the probability is the probability distribution of the continuous field which, in case of a Gaussian distribution, is:

$$P(C_j \mid E_i) = Exp[-(x_j-\mu)^2 / 2\sigma^2] / Sqrt[2\pi\sigma^2]$$

where x_j is the value of the continuous conditional field.

A special case to be considered is a probability value of zero. If certain count data for a categorical field is zero or if the mean and variance of a continuous field is such that the probability calculated is extremely small, the resulting total probability will be zero or very close to it. Since this may be biased due to the limited size of the training data, it is customary to replace all probability values smaller than a given value by a small constant. All probabilities therefore have a lower limit, which is represented in PMML by the *threshold* attribute. All values smaller than this threshold are replaced by the threshold value. The threshold is therefore a critical part of a Naïve Bayes Model and is included in the R as well as the PMML representation of the model.

3. Naïve Bayes in PMML

A Naïve Bayes Model in PMML uses specific elements to represent the computations outlined in the previous section. Besides the *threshold* attribute aforementioned, it contains specific elements for defining predictors or the conditional input fields. Element *BayesInputs* is used as an envelope for all inputs which are each described by their own *BayesInput* element. Each *BayesInput* element may contain a sequence of *PairCounts* elements, one for each category. Figure 1 shows a fragment of the example used by the DMG to illustrate how all the PMML elements come together to represent a Naïve Bayes Model. Note that categorical input field "gender" is represented by its own *BayesInput* element, which contains two *PairCounts* elements, one used for discrete input value "male" and the other for "female". A different *BayesInput* element is used to represent input field "domicile". Note also that element *TargetValueCounts* within *PairCounts* lists, for each value of the target field, the count of the joint occurrences of that target value with a particular discrete input value. Finally, element *BayesOutput* is used to

represent the counts associated with the discrete values ("100", "500", "1000", "5000", "10000") of the target field "amount of claims".

```
<NaiveBayesModel modelName="NaiveBayes Insurance"
    functionName="classification" threshold="0.001">
  <MiningSchema>
    <MiningField name="gender"/>
    <MiningField name="domicile"/>
    ...
    <MiningField name="amount of claims" usageType="predicted"/>
  </MiningSchema>
  <BayesInputs>
    <BayesInput fieldName="gender">
      <PairCounts value="male">
        <TargetValueCounts>
          <TargetValueCount value=  "100" count="4273"/>
          <TargetValueCount value=  "500" count="1321"/>
          <TargetValueCount value= "1000" count= "780"/>
          <TargetValueCount value= "5000" count= "405"/>
          <TargetValueCount value="10000" count=  "42"/>
        </TargetValueCounts>
      </PairCounts>
      <PairCounts value="female">
        <TargetValueCounts>
          <TargetValueCount value=  "100" count="4325"/>
          <TargetValueCount value=  "500" count="1212"/>
          <TargetValueCount value= "1000" count= "742"/>
          <TargetValueCount value= "5000" count= "292"/>
          <TargetValueCount value="10000" count=  "48"/>
        </TargetValueCounts>
      </PairCounts>
    </BayesInput>
    <BayesInput fieldName="domicile">
      <PairCounts value="suburban">
        <TargetValueCounts>
          <TargetValueCount value=  "100" count="2536"/>
          <TargetValueCount value=  "500" count= "165"/>
          <TargetValueCount value= "1000" count= "516"/>
          <TargetValueCount value= "5000" count= "290"/>
          <TargetValueCount value="10000" count=  "42"/>
        </TargetValueCounts>
      </PairCounts>
      ...
    </BayesInput>
    ...
  </BayesInputs>
  <BayesOutput fieldName="amount of claims">
    <TargetValueCounts>
      <TargetValueCount value=  "100" count="8723"/>
      <TargetValueCount value=  "500" count="2557"/>
      <TargetValueCount value= "1000" count="1530"/>
      <TargetValueCount value= "5000" count= "709"/>
      <TargetValueCount value="10000" count= "100"/>
    </TargetValueCounts>
  </BayesOutput>
</NaiveBayesModel>
```

Figure 1. PMML code fragment used to represent a typical Naïve Bayes Model in PMML 4.1.

4. Adding Support for Continuous Input Fields in PMML

As shown above, the current *NaiveBayesModel* element does not offer support for continuous input fields. Supported is limited to categorical input fields. A continuous field can be used, but only if it is binned or discretized first. Figure 2 shows that by using as an example the continuous input field "Sepal.Length", which is part of the well-known Iris dataset [8]. Note that the transformations element *Discretize* is used to bin this field into different discrete values, "tall" and "short", before it can actually be used. Also note that field "Sepal.Length" is being represented by its own *BayesInput* element.

In PMML 4.1, the language's current version, the *BayesInput* element may include a *DerivedField* element, which can be solely used to discretize continuous fields, and one or more *PairCounts* elements depending on the number of discrete values present in the particular categorical input field being represented. To be able to provide support for continuous input fields innately, however, it needs to be changed. We propose extending the *BayesInput* element in the next version of PMML to include a new element we call *TargetValueStats*. This element is described next.

```
<BayesInput fieldName="Sepal.Length">
  <DerivedField optype="categorical" dataType="string">
    <Discretize field="Sepal.Length">
      <DiscretizeBin binValue="short">
        <Interval closure="closedOpen"
          leftMargin="4.3" rightMargin="5.5"/>
      </DiscretizeBin>
      <DiscretizeBin binValue="tall">
        <Interval closure="closedOpen"
          leftMargin="5.5" rightMargin="8"/>
      </DiscretizeBin>
    </Discretize>
  </DerivedField>
  <PairCounts value="short">
    <TargetValueCounts>
      <TargetValueCount value="setosa" count="45"/>
      <TargetValueCount value="versicolor" count="6"/>
      <TargetValueCount value="virginica" count="1"/>
    </TargetValueCounts>
  </PairCounts>
  <PairCounts value="tall">
    <TargetValueCounts>
      <TargetValueCount value="setosa" count="5"/>
      <TargetValueCount value="versicolor" count="44"/>
      <TargetValueCount value="virginica" count="49"/>
    </TargetValueCounts>
  </PairCounts>
</BayesInput>
```

Figure 2. PMML code fragment showing the *BayesInput* element and the discretization of a continuous input field in PMML 4.1.

4.1 Element *TargetValueStats*

Figure 3 shows our proposed changes in the XSD schema for the *BayesInput* element. Note that besides elements *DerivedField* and *PairCounts*, it has been extended to include element *TargetValueStats*. This new element is responsible for listing the distributions obtained for a given continuous input field with respect to each of the values of the target field.

```
<xs:element name="BayesInput">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="DerivedField" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="TargetValueStats" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="PairCounts" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="fieldName" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

Figure 3. Extended XSD Schema for the *BayesInput* element.

With this change, the *BayesInput* element may encompass either one *TargetValueStats* element or one or more *PairCounts* elements. Element *DerivedField* can only be used in conjunction with *PairCounts* whenever the discretization of a given continuous field is desired.

We also propose that *TargetValueStats* serves as the envelope for yet another new element called *TargetValueStat*. Ultimately, this is the element that is going to be used by a continuous input field I_i to define the obtained statistical measures associated with each value of the target field. Figure 4 shows the XSD schema for elements *TargetValueStats* and *TargetValueStat*.

```
<xs:element name="TargetValueStats">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="TargetValueStat" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="TargetValueStat">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
      <xs:group ref="CONTINUOUS-DISTRIBUTION-TYPES" minOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 4. Proposed XSD schema for new elements *TargetValueStats* and *TargetValueStat*.

Note that element *TargetValueStat* uses the *CONTINUOUS-DISTRIBUTION-TYPES* construct from Baseline Models. This construct defines the different distribution types that can be used to represent the required statistical measures. Note that the *GaussianDistribution* element is part of these measures. This is the distribution used by the *naiveBayes* function in the R *e1071* package to build Naïve Bayes Models. Figures 5 and 6 show the respective XSD schemas for *CONTINOUS-DISTRIBUTION-TYPES* and *GaussianDistribution* as depicted in the DMG website (dmg.org – Baseline Models).

```
<xs:group name="CONTINUOUS-DISTRIBUTION-TYPES">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="AnyDistribution"/>
      <xs:element ref="GaussianDistribution"/>
      <xs:element ref="PoissonDistribution"/>
      <xs:element ref="UniformDistribution"/>
    </xs:choice>
    <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:group>
```

Figure 5. XSD schema for *CONTINUOUS-DISTRIBUTION-TYPES*.

```
<xs:element name="GaussianDistribution">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="mean" type="REAL-NUMBER" use="required" />
    <xs:attribute name="variance" type="REAL-NUMBER" use="required" />
  </xs:complexType>
</xs:element>
```

Figure 6. XSD schema for element *GaussianDistribution*.

Once the extensions proposed here are approved and incorporated into the PMML standard, continuous input field *Sepal.Length* can then be represented directly in PMML using element *TargetValueStats* as shown in Figure 7.

5. R Support for PMML

R supports PMML via two packages, the *pmml* package [5] and the *pmmlTransformations* package [9]. The latter allows for a series of data transformations to be applied to the input data before it is used for modeling. The *pmmlTransformations* package works in tandem with the *pmml* package for the exporting of model as well as data transformations into a single PMML file. The *pmml* package provides PMML export functionality for a variety of modeling techniques. As of the time of this article, it 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

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 *saveXML* package is being used to save the resulting PMML code into file *airquality_rf.pmml*.

```
<BayesInput fieldName="Sepal.Length">
  <TargetValueStats>
    <TargetValueStat value="setosa">
      <GaussianDistribution mean="5.006"
        variance="0.124248979591837"/>
    </TargetValueStat>
    <TargetValueStat value="versicolor">
      <GaussianDistribution mean="5.936"
        variance="0.266432653061224"/>
    </TargetValueStat>
    <TargetValueStat value="virginica">
      <GaussianDistribution mean="6.588"
        variance="0.404342857142857"/>
    </TargetValueStat>
  </TargetValueStats>
</BayesInput>
```

Figure 7. PMML code fragment showing the *BayesInput* element for a continuous input field in the next version of PMML (assuming changes are approved by the DMG).

5.1 Exporting Naïve Bayes Models from R in PMML 4.1

As mentioned above, the R *e1071* package implements a function called *naiveBayes* that allows for the building of Naïve Bayes Models. The *pmml* package has recently been updated to support PMML export for objects created by this function. The sequence of R commands below shows how one can use the *naiveBayes* function to build a classification model for the Iris dataset. It also shows how this model can be exported in PMML with the use of the *pmml* function.

```
Library(e1071);
library(pmml);
library(pmmlTransformations);
# Use built in Iris dataset
irisBox <- WrapData(iris);
# Discretize all 4 continuous input fields
irisBox <- DiscretizeXform(irisBox,xformInfo="[Sepal.Length ->
  dSL][double -> string]",table="DiscretizeIrisSL.csv");
irisBox <- DiscretizeXform(irisBox,xformInfo="[Sepal.Width ->
  dSW][double -> string]",table="DiscretizeIrisSW.csv");
irisBox <- DiscretizeXform(irisBox,xformInfo="[Petal.Length ->
  dPL][double -> string]",table="DiscretizeIrisPL.csv");
irisBox <- DiscretizeXform(irisBox,xformInfo="[Petal.Width ->
  dPW][double -> string]",table="DiscretizeIrisPW.csv");
# Build model without using the continuous fields
model <- naiveBayes(Species~., data=irisBox$data[,c(-1:-4)],
```

```
  threshold=0.003);
# Export PMML
pmml(model,dataset=irisBox$data,transforms=irisBox);
```

Note that since PMML 4.1 does not offer support for continuous input fields, we use the *pmmlTransformations* package to discretize the inputs fields before modeling. For example, when discretized, input field "Sepal.Length" is assigned to derived field "dSL". Figure 8 shows the resulting PMML code.

```
<NaiveBayesModel
  functionName="classification" threshold="0.003">
 <MiningSchema>
  <MiningField name="Species" usageType="predicted"/>
  <MiningField name="Sepal.Length" usageType="active"/>
  <MiningField name="Sepal.Width" usageType="active"/>
  <MiningField name="Petal.Length" usageType="active"/>
  <MiningField name="Petal.Width" usageType="active"/>
 </MiningSchema>
 <LocalTransformations>
  <DerivedField name="dSL"
    dataType="string" optype="categorical">
   <Discretize field="Sepal.Length">
    <DiscretizeBin binValue="SL1">
     <Interval closure="openClosed" rightMargin="5"/>
    </DiscretizeBin>
    <DiscretizeBin binValue="SL2">
     <Interval closure="openOpen" leftMargin="5"/>
    </DiscretizeBin>
   </Discretize>
  </DerivedField>
  ...
 </LocalTransformations>
 <BayesInputs>
  <BayesInput fieldName="dSL">
   <PairCounts value="SL1">
    <TargetValueCounts>
     <TargetValueCount value="setosa" count="28"/>
     <TargetValueCount value="versicolor" count="3"/>
     <TargetValueCount value="virginica" count="1"/>
    </TargetValueCounts>
   </PairCounts>
   <PairCounts value="SL2">
    <TargetValueCounts>
     <TargetValueCount value="setosa" count="22"/>
     <TargetValueCount value="versicolor" count="47"/>
     <TargetValueCount value="virginica" count="49"/>
    </TargetValueCounts>
   </PairCounts>
  </BayesInput>
  ...
 </BayesInputs>
 <BayesOutput fieldName="Species">
  <TargetValueCounts>
   <TargetValueCount value="setosa" count="50"/>
   <TargetValueCount value="versicolor" count="50"/>
   <TargetValueCount value="virginica" count="50"/>
  </TargetValueCounts>
 </BayesOutput>
</NaiveBayesModel>
```

Figure 8. PMML code fragment for the Naïve Bayes Model trained with the discretized Iris dataset.

Note that the *pmml* function exported the discretization step as part of the *LocalTransformations* element. This representation is equivalent to having the *Discretize* element inside a *DerivedFied* in element *BayesInput* as shown in Figure 2.

5.2 Exporting Naïve Bayes Models from R with Element *TargetValueStats*

As aforementioned, the *naiveBayes* function is able to build a Naïve Bayes Model with categorical and/or continuous input fields. The sequence of R commands below shows how a Naïve Bayes Model for the Iris dataset can be built and exported in PMML.

```
library(e1071);
library(pmml);
# Use built-in Iris dataset
# Build model defining a threshold value of 0.001
model<-naiveBayes(Species~.,data=iris,threshold=0.001);
# Output the PMML representation in the console
pmml(model,dataset=iris);
```

Note that no pre-processing was required for any of the Iris continuous input fields. As a consequence, the *pmml* function is called without the transformations object. Figure 9 shows the resulting PMML code.

```
<NaiveBayesModel
  functionName="classification" threshold="0.001">
  <MiningSchema>
    <MiningField name="Species" usageType="predicted"/>
    <MiningField name="Sepal.Length" usageType="active"/>
    <MiningField name="Sepal.Width" usageType="active"/>
    <MiningField name="Petal.Length" usageType="active"/>
    <MiningField name="Petal.Width" usageType="active"/>
  </MiningSchema>
  <BayesInputs>
    <Extension>
      <BayesInput fieldName="Sepal.Length">
        <TargetValueStats>
          <TargetValueStat value="setosa">
            <GaussianDistribution mean="5.006"
              variance="0.124248979591837"/>
          </TargetValueStat>
          <TargetValueStat value="versicolor">
            <GaussianDistribution mean="5.936"
              variance="0.266432653061224"/>
          </TargetValueStat>
          <TargetValueStat value="virginica">
            <GaussianDistribution mean="6.588"
             variance="0.404342857142857"/>
          </TargetValueStat>
        </TargetValueStats>
      </BayesInput>
    </Extension>
    <Extension>
     <BayesInputs>
       <BayesInput fieldName="Sepal.Width">
         ...
       </BayesInput>
    </Extension>
    ...
  </BayesInputs>
  <BayesOutput fieldName="Species">
    <TargetValueCounts>
      <TargetValueCount value="setosa" count="50"/>
      <TargetValueCount value="versicolor" count="50"/>
      <TargetValueCount value="virginica" count="50"/>
    </TargetValueCounts>
  </BayesOutput>
</NaiveBayesModel>
```

Figure 9. PMML code fragment featuring the *TargetValueStats* element.

In a typical situation, element *BayesInputs* servers as an envelope for one or more *BayesInput* elements. Note, however, that given that the Iris dataset is composed solely of continuous input fields and that these are not being discretized in advance of model building, the *pmml* package adds an *Extension* element around every single *BayesInput* element so that the resulting code is still compatible with PMML 4.1. This *Extension* element is temporary and will be used until the proposed changes to the *NaiveBayesModel* element are incorporated into the standard itself with the release of PMML version 4.2. In that case, element *TargetValueStats* will be used directly, without the surrounding *Extension* element.

6. Scoring Procedure

The scoring procedure for Naïve Bayes Models can be summarized as follows. Given an input vector such as (i_{12},i_{23},i_3), where i_{12},i_{23} are discrete input values (for categorical fields i_1 and i_2) and i_3 is a continuous input field, the probability for class t_1 is computed as

$$P(t_1 \mid i_{12},i_{23},i_{31}) = L_1 / (L_1 + L_2 + L_3)$$

where

$$L_1 = count[t_1] * count[i_{12},t_1]/count[t_1] * count[i_{23},t_1]/count[t_1] * exp(-(i_3 - mean[1,3])^2 / 2*variance[1,3]) / sqrt(2\pi*variance[1,3])$$

$$L_2 = count[t_2] * count[i_{12},t_2]/count[t_2] * count[i_{23},t_2]/count[t_2] * exp(-(i_3 - mean[2,3])^2 / 2*variance[2,3]) / sqrt(2\pi*variance[2,3])$$

$$L_3 = count[t_3] * count[i_{12},t_3]/count[t_3] * count[i_{23},t_3]/count[t_3] * exp(-(i_3 - mean[3,3])^2 / 2*variance[3,3]) / sqrt(2\pi*variance[3,3])$$

When comparing the formulation shown above with the original scoring procedure described in the DMG website for Naïve Bayes Models in PMML 4.1, we can see that the only difference is the ability to score continuous input fields, as represented by field i_3.

7. CONCLUSION

PMML has evolved throughout the years into a robust and refined language. Its power comes from being able to continue to evolve so that it is at par with the techniques being used today for data processing and model building. The current version of the Naïve Bayes Model element in PMML only allows for categorical input fields. Continuous input fields need to be discretized before they can be represented in PMML. As proposed in here, we extended this element to allow for continuous input fields to be innately represented as well. This is accomplished by a series of new elements including *TargetValueStats*, which is responsible for listing the distributions obtained for a given continuous input field with respect to each of the values of the target field.

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. The *pmml* package has been recently modified to allow for the export of Naïve Bayes Models built by the *naiveBayes* function of package *e1071*. This function allows for categorical and continuous input fields to be used for model building. As a consequence, the *pmml* package incorporates the changes to the Naïve Bayes Model element described here so that models built in R can be fully expressed in PMML. For now the new elements are represented inside an *Extension* element, but this will be dropped as soon as the proposed changes are approved and the next version of PMML is released.

We are excited to be part of the DMG and in the shaping of PMML, the de facto standard to represent data mining and predictive analytic models.

8. ACKNOWLEDGMENTS

We would like to thank our colleagues at Zementis for their 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.

9. REFERENCES

[1] J. M. Bernardo & A. F. Smith (1993). *Bayesian theory*. John Wiley & Sons.

[2] 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).

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

[4] A. Flint & A. Guazzelli (2011). Scorecard Element in PMML 4.1 Provides Rich, Accurate Exchange of Predictive Models for Improved Business Decisions. In *Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

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

[9] T. Jena, A. Guazzelli, W. Lin, M. Zeller (2013). The R *pmmlTransformations* Package. To appear in *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.