Exchanging Data Mining Models with the Predictive Modelling Markup Language

Dietrich Wettschereck and Stefan Müller

1 University of Applied Sciences Bonn-Rhein-Sieg, Department of Applied Computer Science, Grantham-Allee 20, 53757 Sankt Augustin, Germany, dietrich.wettschereck@dialogis.de
2 LIACC-University of Porto, Rua do Campo Alegre, 823, 4150 Porto, Portugal

Abstract. The aim of the Predictive Model Markup Language (PMML) is to support the exchange of data mining models between different applications and visualization tools. It is the result of a standardization effort by a group of vendors. PMML is an XML-based language (grammar) for describing data mining models. Despite its name, it is not limited to predictive models. The contribution of this paper is two-fold: an encouragement for researchers to base their work on PMML and a slightly enhanced PMML DTD for multi-relational rule models.

1 Introduction

This document introduces some of the concepts of PMML and highlights how PMML can be employed for the benefit of researchers and commercial entities. One such benefit is the use of PMML in knowledge discovery support engines (KDDSE) such as Dialogis’ D-Miner or IBM’s Intelligent Miner. In some sense, PMML is to the output of KDDSEs what JDBC or ODBC is to their input: a vendor independent means of separating data mining results from the tools that produced these results.

The primary advantages of using the PMML standard in (extensible) knowledge discovery support engines (KDDSE) and implementations of specific mining methods are (1) that no tool-specific parsers are required which decreases time and effort during integration and eliminates many potential errors during parsing and (2) that the knowledge discovered can be separated from the tool that was used to discover this knowledge.

The paper is organized as follows: Section 2 introduces PMML and gives a short overview of the architecture of D-Miner and how PMML can be integrated into such a KDDSE. Section 3 proposes a new PMML DTD for multi-relational rule models. The discussion in Section 4 concludes the paper.1

1 This work has been supported by SolEuNet [5] and Mining Mart [6].

1 Example PMML models for decision trees and their corresponding visualizations as produced by D-Miner are available upon request.
2 PMML as part of a KDDSE

2.1 PMML

The Predictive Model Markup Language (PMML) is a set of Document Type Descriptions (DTDs) specified in XML. The first version (1.0) was provided in July 1999 [1] by the Data Mining Group (DMG, http://www.dmg.org). The DMG is "an independent, vendor-led group which develops data mining standards" [1]. Currently, its seven core members are Angoss, IBM, Magnify, NCR, Oracle, SPSS, and the National Centre of Data Mining (University of Illinois, Chicago). Additionally, a number of associated members have been accepted to the group. Version 1.1 has been released in August 2000 to incorporate improvements based on the lessons learned from the first release as well as definitions for further data mining models. Version 1.2 is currently under development. The following new features are planned for the upcoming version:

- open to any Data Mining vendor
- utilize review by industry experts
- additional functionality (i.e. models)
- align with related standards (e.g. OLE DB)

The advantages of PMML can be summarized as follows.

1. It provides independence of the knowledge extracted from application, implementation, (hardware) platform, and operating system.
2. It simplifies the use of data mining models by other applications or people.
   For example, consultants or researchers (SolEuNet [5] members etc.) can function as producers of models and customers can import models into their own tools.
3. It is not concerned with the process of creating a model or the specific implementation of the algorithm.
4. DTDs support proprietary extensions to allow for enriched information storage for specialized tools.

Figure 1 indicates where PMML can be put to a productive use within the KDD process. After a model is generated by a data mining algorithm and stored in the PMML format, it can be used by other tools to visualize the model or to classify unseen values. Thanks to this publicly available standard format, tools from different vendors can be used at different stages of the KDD process or by different users. PMML models are typically produced by data mining algorithms. Converters can be employed to translate the proprietary format into PMML for implementations that do not directly support PMML (see Fig. 3).

The PMML 1.1 definition includes DTDs for the following types of models:

1. polynomial regression
2. general regression
3. decision trees
4. center based clusters
Fig. 1. The role of PMML in the KDD process.

Further information about the DMG and PMML can be obtained at the DMG web site at http://www.dmg.org.

2.2 D-Miner

D-Miner is a knowledge discovery suite that includes a variety of data pre-processing, data visualization, and data mining methods. It includes task management in order to repeat stored mining tasks or to export such tasks for deployment of data mining results. It also consists of several data management functions. Finally, its plug-in concept allows for easy integration of new algorithms into the system [9].

Figure 2 shows a simplified schema of the architecture of D-Miner. The main system with its data and process management functions is represented on the right side. The graphical user interface (GUI) is also part of the main system. The remainder of the figure shows the architecture of the plug-in concept of D-Miner. The plug-in API is based on the ideas of [8] and has been significantly overhauled to encompass recent advances in Java and XML technology [7]. The XML parser and the D-Miner side of the plug-in API will be explained below. The box on the left-hand side of the figure symbolizes a plug-in module. For example, this could be an algorithm such as C4.5 [2], but also an import filter or pre-processing operator. In order to integrate a plug-in module a developer has to provide two files. An XML file describes the meta data of the plug-in such
as its input and output parameters, author and usage information or command line parameters. A Java class file implements the module API and manages the data flow to and from the plug-in module. The XML file must comply to a DTD that is publicly available from the authors. Additionally, the Java class file must implement a Java interface that is available from the authors.

D-Miner reads module meta data at start-up time from all XML files that are stored in a certain subdirectory (tools directory) of the D-Miner installation and checks their syntax. If the syntax complies with the given DTD, then the plug-in will be added to one of the D-Miner menus and can be called by the user. To start the plug in module, users have to select the desired analysis (or import or pre-processing) method from the menu and change some of the parameter values, if the default values are not sufficient. Parameters are shown in D-Miner as graphical elements to make the inspection and modification of parameter settings as easy as possible. All information required to construct the graphical user interface is extracted from the corresponding XML file thus enabling non-programmers to change default settings or tool tips in order to adapt the system for their own use. D-Miner distinguishes between two parameter types: data parameters and tool parameters. Data parameters allow the user the selection of the data to be analyzed. This data may reside within D-Miner or in a database. Tool parameters control the behavior of the analysis method. D-Miner calls the appropriate methods of the module API to execute the plug in module after the parameters and data sources are selected. The module API is primarily responsible for three activities: (1) retrieving the data to be analyzed from the core system in order to transform it into the format required by the plug-in, (2) calling the plug-in within the same Java thread or as a separate executable, and (3) reading the analysis results back into the system and telling D-Miner to store these results permanently. Most implementations of analysis algorithms produce proprietary textual output that must be parsed by the plug-in API and can then be converted into D-Miner’s internal format or PMML.
2.3 D-Miner support for PMML

D-Miner supports the PMML standard in two ways. 1) Import and export facilities allow for the exchange of data mining models with other KDD tools. 2) A PMML API, as part of the plug-in API, provides immediate access to PMML models. The latter solution allows the direct exchange of PMML models between different plug-in modules and the D-Miner system and simplifies the integration of new plug-ins into D-Miner as long as they produce (or use) PMML models.

![Diagram of D-Miner support for PMML](image)

**Fig. 3.** The PMML import and export facility of D-Miner is also implemented as a plug-in.

Figure 4 is similar to Fig. 2 and shows how PMML formats can be used and supported by plug-in modules. The module converts the PMML model into a format known by the D-Miner system. Currently, D-Miner is able to exchange tree models (import and export) and rule models (only export to date) in the PMML format.

![Diagram of D-Miner support for plug-ins](image)

**Fig. 4.** D-Miner support for plug-ins that produce PMML models.
3 The DTD for the RuleModel

Rules are widely used in data mining tasks. To date nearly every implementation of a rule learning algorithm uses its own means for representing rules. A number of systems use Prolog syntax to present rules. However, Prolog is not widely known and requires a Prolog interpreter for rule evaluation. A common format for the exchange of rule models between tools would therefore improve the general utility of rules.

PMML already defines a DTD for rule models with a strong focus on association rules. This DTD is however restricted to relatively simple rule models, since only transactions with a single attribute can be represented. Furthermore it cannot express variables, negated literals, or multi-relational structures as they are usually required by rule learners based on Inductive Logic Programming techniques.

We have modified the PMML DTD for association rules so that D-Miner can store rule models that (optionally) use first order predicate logic in PMML. Although the rule model is not yet an official part of the PMML standard it is an important supplementation for rule based algorithms such as Midos [3] or FOIL [4]. Figure 5 shows the modified grammar for first order rules. The primary XML-element is RuleModel. Additions to the original PMML 1.1. DTD for association rules are shown in bold face. An example rule model produced from D-Miner is visualized in Fig. 6. The corresponding PMML representation is listed in Figs. 7 through 10. The intention of this model is to cover models that were produced by ILP (inductive logic programming) algorithms such as Midos, FOIL, TILDE, Progol, WARMR and others. Association rule models (propositional rule models) as for example APRIORI can be considered as subsets of the first order rule model. Therefore these models (namely AssociationModel) could be converted into the RuleModel format.

3.1 The RuleModel DTD

<!-- ============== Rule Model ======================= -->
<!ELEMENT RuleModel (Extension*, MiningSchema, ModelStats?, Term+, Literal+, Rule+)>
<!ATTLIST RuleModel modelName CDATA #IMPLIED negations (yes | no) "no" functions (yes | no) "no">
<!-- Subelements of Term are only allowed for type="function" -->
<!ELEMENT Term (Term*)>
<!ATTLIST Term id %ELEMENT-ID; #REQUIRED
  type (constant | variable | function) "constant"
  symbol CDATA #REQUIRED
  mappedSymbol CDATA #IMPLIED
  weight %REAL-NUMBER; #IMPLIED
  numberOfParams %INT-NUMBER; #IMPLIED>
The main structure of the DTDs AssociationModel and RuleModel is similar. For example, to avoid redundancy within the rule elements, Terms and Literals (AssocItem and AssocItemset in AssociationModel) are listed only once, then references are used for building rules. Differences between the standard PMML DTD AssocRules and the proposed RuleModel DTD are listed below.

- The Element AssocItem corresponds to Term, although Term is far more expressive. While AssocItem stands for an attribute value, i.e. a constant, Term can be a constant, a variable, or a function (with respect to first order rules).
- AssocItemset corresponds to Literal. While AssocItemsets are sets of constant values (AssocItem), Literals consist of terms in the sense of first order rules. Furthermore, references to literals (LiteralRef) allow to negate a literal in a given rule.
- Antecedent and Consequent correspond to rule body and rule head in first order logic terminology, respectively. In the AssocRule DTD they are attributes, in the RuleModel DTD Antecedent and Consequent are elements. This way it is possible to put more than one literal (Literal) into the rule.
body or rule head to support multi-relational structures. This is not possible with the AssocRule model.

– To support first order rules the RuleModel DTD is able to express functions, enabling the definition of recursive Terms. This may not be relevant for commercial applications, but can be valuable for research tools.

Fig. 6. D-Miner visualization of rule models.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<PMML>

<Header copyright="www.dialogis.de">
    <Application name="D-Miner" version="2.5"/>
</Header>

<DataDictionary numberOfFields="7">
    <DataField name="life insurance" opType="categorical"></DataField>
    <DataField name="income" opType="continuous"></DataField>
    <DataField name="region" opType="categorical"></DataField>
</DataDictionary>
```
<DataField name="social status" opType="categorical"></DataField>
<DataField name="age" opType="continuous"></DataField>
<DataField name="o__att24o" opType="categorical"></DataField>
<DataField name="home insurance" opType="categorical"></DataField>
</DataDictionary>

<RuleModel>
<MiningSchema>
<MiningField name="life insurance" usageType="active"></MiningField>
<MiningField name="income" usageType="active"></MiningField>
<MiningField name="region" usageType="active"></MiningField>
<MiningField name="social status" usageType="active"></MiningField>
<MiningField name="age" usageType="active"></MiningField>
<MiningField name="o__att24o" usageType="predicted"></MiningField>
<MiningField name="home insurance" usageType="active"></MiningField>
</MiningSchema>
</RuleModel>

/* DataDictionary and MiningSchema are same in most cases -->

Fig. 7. Example rule model in PMML format (meta information)

<Term id="1" type="variable" symbol="o__att24o" mappedSymbol=""/>
<Term id="2" type="constant" symbol="no" mappedSymbol=""/>
<Term id="3" type="variable" symbol="gender" mappedSymbol=""/>
<Term id="4" type="constant" symbol="female" mappedSymbol=""/>
<Term id="5" type="variable" symbol="social status" mappedSymbol=""/>
<Term id="6" type="constant" symbol="family" mappedSymbol=""/>
<Term id="7" type="variable" symbol="age" mappedSymbol=""/>
<Term id="8" type="constant" symbol="18" mappedSymbol=""/>
<Term id="9" type="variable" symbol="single" mappedSymbol=""/>
<Term id="10" type="constant" symbol="68" mappedSymbol=""/>
<Term id="11" type="constant" symbol="B8" mappedSymbol=""/>
<Term id="12..21" ... not shown due to space restrictions ... />

Fig. 8. Example rule model in PMML format (term definitions)

<List id="1" predicate="equal" numberOfParams="2">
<TermRef termRef="3" position="0"/>
<TermRef termRef="4" position="1"/>
</List>

<List id="2" predicate="equal" numberOfParams="2">
<TermRef termRef="5" position="0"/>
<TermRef termRef="6" position="1"/>
</List>
<Literal id="3" predicate="equal" numberOfParams="2">
  <TermRef termRef="1" position="0"/>
  <TermRef termRef="2" position="1"/>
</Literal>

<Literal id="4" predicate="lessOrEqual" numberOfParams="2">
  <TermRef termRef="7" position="0"/>
  <TermRef termRef="8" position="1"/>
</Literal>

<Literal id="5" predicate="equal" numberOfParams="2">
  <TermRef termRef="5" position="0"/>
  <TermRef termRef="9" position="1"/>
</Literal>

<Literal id="6" predicate="greaterThan" numberOfParams="2">
  <TermRef termRef="7" position="0"/>
  <TermRef termRef="10" position="1"/>
</Literal>

<Literal id="7" predicate="greaterThan" numberOfParams="2">
  <TermRef termRef="7" position="0"/>
  <TermRef termRef="11" position="1"/>
</Literal>

<Literal id="8..18" ... not shown due to space... </Literal>

<Literal id="19" predicate="lessOrEqual" numberOfParams="2">
  <TermRef termRef="7" position="0"/>
  <TermRef termRef="10" position="1"/>
</Literal>

Fig. 9. Example rule model in PMML format (literals)

<Rule support="0.0" confidence="0.0" ruleId="1">
  <Antecedent>
    <LiteralRef literalRef="1" position="1" negated="false"/>
    <LiteralRef literalRef="2" position="2" negated="false"/>
  </Antecedent>
  <Consequent>
    <LiteralRef literalRef="3" position="1" negated="false"/>
  </Consequent>
</Rule>

<Rule support="0.0" confidence="0.0" ruleId="2">
  <Antecedent>
    <LiteralRef literalRef="4" position="1" negated="false"/>
    <LiteralRef literalRef="5" position="2" negated="false"/>
  </Antecedent>
  <Rule/>
</Rule>
<Rule support="0.0" confidence="0.0" ruleId="3">
  <Antecedent>
    <LiteralRef literalRef="6" position="1" negated="false"/>
  </Antecedent>
  <Consequent>
    <LiteralRef literalRef="3" position="1" negated="false"/>
  </Consequent>
</Rule>

<Rule support="0.0" confidence="0.0" ruleId="4">
  <Antecedent>
    <LiteralRef literalRef="7" position="1" negated="false"/>
    <LiteralRef literalRef="2" position="2" negated="false"/>
  </Antecedent>
  <Consequent>
    <LiteralRef literalRef="3" position="1" negated="false"/>
  </Consequent>
</Rule>

<Rule support="0.0" confidence="0.0" ruleId="5">
  <Antecedent>
    <LiteralRef literalRef="1" position="1" negated="false"/>
    <LiteralRef literalRef="8" position="2" negated="false"/>
    <LiteralRef literalRef="9" position="3" negated="false"/>
    <LiteralRef literalRef="10" position="4" negated="false"/>
    <LiteralRef literalRef="11" position="5" negated="false"/>
    <LiteralRef literalRef="12" position="6" negated="false"/>
  </Antecedent>
  <Consequent>
    <LiteralRef literalRef="3" position="1" negated="false"/>
  </Consequent>
</Rule>

<Rule support="0.0" confidence="0.0" ruleId="6">
  <Antecedent>
    <LiteralRef literalRef="13" position="1" negated="false"/>
    <LiteralRef literalRef="14" position="2" negated="false"/>
    <LiteralRef literalRef="15" position="3" negated="false"/>
    <LiteralRef literalRef="5" position="4" negated="false"/>
  </Antecedent>
  <Consequent>
    <LiteralRef literalRef="3" position="1" negated="false"/>
  </Consequent>
</Rule>

<Rule support="0.0" confidence="0.0" ruleId="7,8">
  ... not shown ...
</Rule>

<Rule support="0.0" confidence="0.0" ruleId="9">
  <Antecedent>
  </Antecedent>
  <Consequent>
    <LiteralRef literalRef="17" position="1" negated="false"/>
  </Consequent>
</Rule>

</RuleModel> </PMML>

Fig. 10. Example rule model in PMML format (rules)
4 Discussion

The primary aim of this paper is to advertise and promote the use of PMML in data mining. The first step towards a wide acceptance of this new standard is already taken since a number of tool vendors have started to provide import and export filters for PMML. The second step however is that researchers directly use PMML when implementing new data mining methods. We are currently not aware of any implementations of data mining methods that directly support the PMML standard. We strongly encourage all researchers to comply to this standard so that the integration of their implementation into (commercial) tools will be made as simple as possible thus opening up their work to a wider public.

References

7. Stefan Mueller, "Design und Implementierung einer Plug-In-Schnittstelle zur Integration von Data-Mining-Algorithmen", diploma thesis (in German) at the Rheinische Friedrich-Wilhelms-Universität Bonn, Germany, 2000
9. Kepler and D-Miner are the result of joint work by GMD - German National Research Center for Information Technology (http://ais.gmd.de/KD) and Dialogis (http://www.dialogis.de), 1997 - 2001