

Augustus: The Design and Architecture of a PMML-Based Scoring Engine¹

John Chaves, Chris Curry, Robert L. Grossman²,
David Locke and Steve Vejcik

Open Data Group
River Forest, IL 60305, USA

1. Introduction

The Predictive Model Markup Language or PMML is an XML markup language for statistical and data mining models that has been developed over the past several years by the Data Mining Group, a consortium of data mining vendors [DMG:2006].

Expressing models in PMML provides several advantages when deploying and maintaining data mining applications:

1. **Scoring.** By using PMML, one application can produce a PMML model using training data, and an entirely different application, running on an entirely different system, can consume the PMML model and a stream of data to score. The latter is sometimes called a *scoring engine*.
2. **Model management.** When data mining applications are deployed, the statistical models used usually change over time. Also, a single application can sometimes require several, or indeed many, different statistical models. Managing multiple models is simplified by using a repository or database of PMML models. In this case, a scoring engine can simply query for the appropriate model and load it.
3. **Highly available applications.** Some data mining applications have the requirements that they are both highly available and yet can be updated from time to time. There are two basic approaches to defining standards for data mining applications: *declarative* approaches, such as PMML, that describe the inputs, outputs, parameters, and metadata associated with a model, and *process-oriented* approaches [Ferrucci:2004] that define a work flow graph whose nodes are data mining processes and whose edges are data flows. Declarative approaches are generally considered a safer approach when high

¹ This is a draft of a paper from the Fourth Workshop on Data Mining Standards, Services and Platforms (DM-SSP'06), Philadelphia, August 20, 2006,

² Robert Grossman is also a faculty member at the University of Illinois at Chicago.

availability is required since no new code is being used but rather data is just being read and processed.

4. **Compliance.** Data mining applications in regulated industries such as financial services, insurance, or health care must sometimes ensure that the statistical and data mining models used are compliant with various rules and regulations. Encapsulating a statistical or data mining model in PMML, checking that the model is compliant, using a scoring engine, and keeping database logs of which model was used for scoring provides a simple methodology for meeting these requirements.

In this paper, we describe the design and implementation of an open source PMML-based scoring engine called Augustus. We also describe two case studies that use it.

2. Requirements

In this section, we describe the main requirements that motivated the design of the Augustus system.

Self-describing models. The first requirement is to view statistical and data mining models as first class, self-describing objects. When this requirement is supported, it is straightforward to exchange models between applications, manage them using a database or repository, and audit them for compliance related purposes.

Standards-compliant. The second requirement is for Augustus to be standards-compliant when feasible.

Event-based data mining process model. The third requirement is for Augustus to support an event-based data mining process model. One such model is described in [Grossman:2005a]. An event-based data mining process model a) processes a stream of events, b) forms feature or state vectors by transforming and aggregating these events, c) scores the feature vectors using a model to produce scores, and then d) processes the scores using rules or other transformations to produce alerts and reports.

Interoperability. The fourth requirement is for Augustus to be interoperable with other data mining applications. This was interpreted as Augustus components should have clearly defined inputs, outputs and APIs, and have behavior that was completely determined by the inputs, and therefore should be able to interoperate with any other components that were similarly designed.

Encapsulate data preparation. The fifth requirement is for Augustus to isolate and encapsulate data preparation.

Encapsulate deployment. The sixth requirement is for Augustus to isolate and encapsulate deployment related issues in what the PMML Working Group sometimes

refers as a deployment package. A deployment package describes the data sources and data targets, post-processing of scores, various metadata required for operating the application, and related information.

3. Design

PMML components. Augustus uses the standard components of PMML to define the inputs, outputs, parameters, and transformations required by statistical and data mining models. In particular, Augustus supports the following PMML components:

- A data dictionary is used to define the fields that are potential inputs or outputs of models.
- A mining schema is used to identify the specific fields used by a model. The mining schema is a subset of the data dictionary that is specific for a given model, while the data dictionary contains information that does not vary from model to model. For example, the mining schema defines the dependent variable for a predictive model and specifies whether a given field should be used as input to a model.
- The transformation dictionary defines derived fields using one or more mining fields. A number of different types of transformations are supported, including normalization, discretization, value mapping and aggregation.
- The specific parameters required to define a model are defined using one or more PMML elements specific to that model.

Augustus components. The current version of Augustus consists of the following main components.

- A data management component called a Unitable is part of the Augustus kernel. A UniTable is broadly similar to a data frame in the R system [R:2006]. It contains data fields arranged in columns, that may be of different types, but all of which have the same number of rows.
- Components for shaping and transforming data. These components are also part of the Augustus kernel.
- Utilities for processing PMML files. These are collected together in the pmmlib directory in the distribution.

- Model specific components. Currently, Augustus supports baseline models and tree-based models [DMG:2006]. These are collected together in the modellib directory in the distribution.
- Run time support. Augustus provides a general mechanism for reading data into Augustus and writing data out of Augustus called anyreader and anywriter, respectively. For example, anyreader can read data from a stream, a file, or a database and put it into a common format for further processing by Augustus. These are collected together in the runlib directory in the distribution.
- Miscellaneous tools. The Augustus distribution also includes a number of auxiliary utilities, such as configuration utilities, utilities for working with date and time, etc. For example, there is a utility that will create a stub PMML data schema based upon an analysis of a data file.

The relationship of these components can be seen in Figure 1.

Segmentation. The case studies described below required that a large number of different models be used, one for each cell in a data cube. After some experimentation, we settled upon the following approach.

We introduced a segmentAssignment PMML-extension of the following form:

```
<segmentAssignment name= "segmentName">
  <segmentAssignmentField name= "miningFieldName">
    <segmentAssignmentType name= saType>
      <segmentParameters>
        </segmentParameters>
        etc.
      </segmentAssignment>
```

where *saType* is one of the following: regular-segmentation or explicit-segmentation.

Here *segmentName* is a unique identifier of the given segment assignment, *miningFieldName* is the name of some mining field described earlier in the mining schema which is the field that will be segmented using the given parameters and *etc* indicates that there can be any number of segmentParameters in order.

The segment parameters depend on which of the values for *saType* was chosen. For regular, the segment parameters take the following form:

```
<segmentParameters>
  <segmentParameter name= "left-endpoint"> value </segmentParameter>
  <segmentParameter name= "right-endpoint"> value </segmentParameter>
  <segmentParameter name= "number-partitions"> value </segmentParameter>
</segmentParameters>
```

Here *value* is any real number for each of the endpoints and is an integer for the number of partitions. This will form a given input parameter value for each segment endpoint.

For explicit, the segment parameters take the following form:

```
<segmentParameters>
  <segmentParameter value="someValue" dataType="someDataType" />
</segmentParameters>
```

Here *someValue* represents an explicit value of the given data type and *someDataType* represents a defined data type for explicit segmentation. Currently, the only defined data type for explicit segmentation is *string*. This will form a given input parameter value for the given value.

Augustus creates a separate model for all possible combinations of the given parameter values for each parameter as output for this process. For the below example, Augustus would produce 1,000,000 different segmented models.

```
<segmentAssignment name="timeSegmentation">
  <segmentAssignmentField name="time">
    <segmentAssignmentType name="regular-segmentation">
      <segmentParameters>
        <segmentParameter name="left-endpoint">0</segmentParameter>
        <segmentParameter name="right-endpoint">1</segmentParameter>
        <segmentParameter name="number-partitions">10000</segmentParameter>
      </segmentParameters>
    </segmentAssignmentType>
  </segmentAssignmentField>
</segmentAssignment>
<segmentAssignment name="incidentSegmentation">
  <segmentAssignmentField name="incidents">
    <segmentAssignmentType name="regular-segmentation">
      <segmentParameters>
        <segmentParameter name="left-endpoint">1.7</segmentParameter>
        <segmentParameter name="right-endpoint">2000.45</segmentParameter>
        <segmentParameter name="number-partitions">100</segmentParameter>
      </segmentParameters>
    </segmentAssignmentType>
  </segmentAssignmentField>
</segmentAssignment>
```

Data shaping in Augustus. Data preparation is usually one of the most time consuming components in a data mining project. Capturing and segregating all the code required to prepare data is not always easy. PMML supports a number of data preparation functions, including functions for normalization, discretization, value mapping, and aggregation, which can all be included in PMML's transformation dictionary [PMML:2006].

On the other hand, PMML code describing data preparation is designed to be written by a machine, not by a human. For this reason, Augustus also allows a user to use Unitable functions and Python to define operations and functions required when preparing data.

The following fragment is taken from the Mining Schema describing a model that requires computing aggregations with data restricted to a specific data segment. In this example, the raw data is comprised of the fields 'CountType', 'count', and 'Date'

```
<MiningField name="count"          usageType="supplementary"/>
<MiningField name="AvgContribution" usageType="active"/>
<MiningField name="DailyCount"     usageType="supplementary"/>
<MiningField name="ScaledDailyCount" usageType="supplementary"/>
<MiningField name="globalDailyCount" usageType="supplementary"/>
<MiningField name="CountType"      usageType="supplementary"/>
<MiningField name="Date"           usageType="supplementary"/>
```

Augustus shaping code instantiates these fields as columns in a UniTable. Shaping code also applies the necessary transformations (specified in the Transformation Dictionary) to create some additional columns (AvgContribution, DailyCount, ScaledDailyCount). For example, here is a PMML fragment that specifies how to create the field 'DailyCount' from the fields 'count', and 'Date'

```
<DerivedField name="DailyCount">
  <Aggregate field="count" function="sum" groupField="Date"/>
</DerivedField>
```

This example actually creates several new fields as the required function describes the calculation of the sum of field 'count' sorted according to groupings of field 'Date'. Depending on the actual values of 'Date', there may be zero, one, or many new fields.

4. Implementation

Augustus is implemented in Python but is designed to be interoperable with components designed in other languages.

The UniTable class is designed to function as a “Universal Table” that is used by most Augustus components. The UniTable data structure is analogous to an R frame: a table where the columns are vectors of equal length, but may be of different types. It is based on the Python numarray package and the Augustus application programming interface attempts to maintain consistency with the numarray APIs.

The design goal of UniTable was to create a very fast, efficient object for data shaping, model building, and scoring, both in a batch and real-time context. UniTable’s main features include:

- A file format that matches the native machine memory storage of the data. This allows for memory-mapped access to the data, eliminating the need for data parsing or sequential reading.
- Fast column-oriented vector operations.
- Support for demand driven, rule based calculations. Derived columns that support data shaping can be defined in terms of operations on other columns, including other derived columns, and are made available when referenced.
- The ability to support high volume event streams by automatically switching to vector mode when behind, and scalar mode when keeping up with individual input events.

5. Case Studies

Augustus was used to compute baselines for the application described in [Grossman:2005b]. The goal of this application is to detect changes in sensor data measuring traffic in the Chicago region. We emphasize that the goal of this application is to detect unexpected variations or changes in traffic, not traffic congestion per se, which is quite easy to identify. To detect changes, segmentation was used to build separate baseline models for each hour of the day (24 hours), for each day of the week (7 days), and for approximately 300 different locations (300 locations). This yielded 24 x 7 x 300 or 50,400 separate baseline models. Augustus was used both to build these baseline models and to score real time event streams of sensor measurements using these baseline models. The transformations and aggregations for this application were done using PMML-based transformations and UniTable-based data shaping functions.

Augustus was also used to compute baselines for the application described in [Bugajski:2005]. The goal of this application is to detect data quality and data interoperability issues for a large payments card processor. This application computed separate baselines for several payment fields, in each of several different regions, for large numbers of different banks and merchants. All together over 1,000,000 separate baselines models were computed. Again, the transformations and aggregations for this application were done using PMML-based transformations and UniTable-based data shaping functions.

6. Related Work

There are several open source data mining applications. One of the most widely used is the R system [R:2006], which has an extensive collection of statistical and data mining algorithms. Weka is a popular Java™ based data mining application that includes an easy to use graphical user interface [Weka:2006].

At this time, neither R nor Weka are PMML compliant and both are designed as general purpose statistical/data mining applications rather than PMML-based scoring engines.

As far as we are aware of, Augustus is one of the first open source general purpose scoring engines that is PMML compliant.

7. Status

We have used Augustus internally for projects for about two years. Augustus version 0.2.4 was the first general release. Version 0.2.4 was released on Source Forge in December 2005.

We expect to release version 0.3 of Augustus on Source Forge during the third quarter of 2006.

8. Summary and Conclusion

In this paper, we have described the design of an open source PMML-compliant scoring engine called Augustus. Some novel features of Augustus include its support for model segmentation and its use of Python to simplify data preparation. We have also described two case studies that used Augustus.

References

[Bugajski:2005] Joseph Bugajski, Robert Grossman, Eric Sumner, Tao Zhang, A Methodology for Establishing Information Quality Baselines for Complex, Distributed Systems, 10th International Conference on Information Quality (ICIQ), 2005.

[DMG:2006] The Data Mining Group, Predictive Model Markup Language (PPML), retrieved from www.dmg.org on June 20, 2006.

[Ferrucci:2004] D. Ferrucci and A. Lally, Building an example application with the Unstructured Information Management Architecture, IBM Systems Journal, Volume 43, 2004, pages 455-475.

[Grossman:2002] Robert Grossman, Mark Hornick, and Gregor Meyer, Data Mining Standards Initiatives, Communications of the ACM, Volume 45-8, 2002, pages 59-61

[Grossman:2004] Robert L. Grossman, editor, Proceedings of the Second Annual Workshop on Data Mining Standards, Services, and Platforms (DM-SSP 2004), ACM, 2004.

[Grossman:2005a] Robert L. Grossman, Alert Management Systems: A Quick Introduction, in Managing Cyber Threats: Issues, Approaches and Challenges, edited by Vipin Kumar, Jaideep Srivastava and Aleksandar Lazarevic, Springer Science+Business Media, Inc., New York, 2005, pages 281-291.

[Grossman:2005b] Robert L. Grossman, Michal Sabala, Javid Alimohideen, Anushka Aanand, John Chaves, John Dillenburg, Steve Eick, Jason Leigh, Peter Nelson, Mike Papka, Doug Rorem, Rick Stevens, Steve Vejcik, Leland Wilkinson, and Pei Zhang, Real Time Change Detection and Alerts from Highway Traffic Data, ACM/IEEE International Conference for High Performance Computing and Communications (SC '05).

[R:2006] The R Project, retrieved from www.r-project.org on June 10, 2006.

[Weka:2006] Weka 3 – Data Mining Software in Java, retrieved from www.cs.waikato.ac.nz/ml/weka on June 10, 2006.

Figure 1. This figure summarizes the basic architecture of Augustus version 0.2.3. The shaping functions and state management are based upon UniTable. One or more models may be either defined directly or else defined using the PMML-extension segmentation element. A project can then use one or more of the models.

