Pattern: PMML for Cascading and Hadoop

Paco Nathan
Concurrent, Inc.
pacoid@cs.stanford.edu

Girish Kathalagiri
AgilOne, Inc.
girish.kathalagiri@agilone.com

ABSTRACT

Pattern is an open source project based on Predictive Model Markup Language, which is focused on the convergence of predictive modeling, machine learning, cloud computing, distributed systems, Hadoop, etc. PMML as a language helps to ease this integration. The open source Cascading API is used as a foundation for constructing and optimizing workflows, which are highly parallel at scale. Pattern implements a *domain-specific language* (DSL) to translate PMML model descriptions into Cascading workflows, which can then be deployed on cloud computing platforms such as Amazon AWS ElasticMapReduce and Microsoft HDInsight.

Observed benefits include greatly reduced development costs and less licensing issues at scale, while leveraging the scalability of Apache Hadoop clusters, existing intellectual property in predictive models, and the core competencies of analytics staff. Analysts can train predictive models in popular analytics frameworks, such as SAS, Microstrategy, R, Weka, SQL Server, etc., then run those models at scale on Apache Hadoop with little or no coding required.

This paper explains the integration of PMML and Cascading, using a sample application based on the crime dataset from the City of Chicago Open Data. The sample application implements a predictive model for expected crime rates based on location, hour of day, and month. Multiple models are captured as PMML, then integrated via Pattern to implement the entire workflow as a single application.

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

Standardization, Language

PMML '13 Chicago, Illinois USA

Keywords

Abstraction Layer, Cascading, Data Mining, Distributed Systems, Domain Specific Language, Ensembles, Hadoop, Literate Programming, Open Data, Open Source Software, PMML, Pattern Language, Predictive Analytics, Predictive Model Markup Language, Workflows

1. INTRODUCTION

The convergence of Big Data and predictive analytics has opened up a wide range of potential application areas. Early work with Apache Hadoop tended to focus on ETL processes, marketing funnel, reporting – use cases which did not require much in terms of predictive models. Subsequent work on recommender systems, anti-fraud classifiers, and related needs in e-commerce and online advertising proved the case for large-scale analytics; however, these applications tended to be written directly in the Hadoop API, requiring large amounts of algorithm work and expert programming in a MapReduce[**?**] context. More recently, Hadoop applications in use cases such as climatology, genomics, remote sensing, and *Internet of Things* sensor arrays are demonstrating the demand to migrate predictive models from popular analytics platforms to run on Hadoop clusters. As a wider range of scientific applications become used on Hadoop, there will be even more need to find ways to migrate models and leverage domain expertise, without requiring a huge amount of expert programming.

Pattern is a machine learning library which extends the open source Cascading[1] API to provide support for the Predictive Model Markup Language (PMML). Predictive models which have been created on an analytics platform, such as SAS or R, can be exported as PMML. The Pattern library works by translating PMML – an established XML standard for predictive model markup[**?**] – into data pipelines and other functional programming constructs in the Cascading API in Java. Cascading provides a pattern language for constructing large-scale data workflows on Apache Hadoop – effectively describing business process in terms of functional programming. Since PMML describes the business process of workflows (e.g., model composition and segmentation), there exists a natural mapping into Cascading.

The open source code repository for Pattern on GitHub[2] provides its Java source code and unit tests, Gradle build scripts, PMML examples and accompanying sample data sets, R scripts for generating PMML examples, as well as

[1] http://www.cascading.org/
[2] http://github.com/Cascading/pattern

Python scripts for generating large-scale data sets to use in creating and scoring predictive models. PMML models can be run in Pattern using a pre-defined JAR file – with no coding required.

PMML can also be combined with other Cascading components based on ANSI SQL (Lingual), Scala (Scalding), Clojure (Cascalog), etc. Combing such components in Cascading helps to provide for seamless integration of data preparation and model scoring. It also integrates the process moving results into production use, which mitigates the operational risk of managing complex analytics applications. Moreover, instead of having the model run as a separate process, the compiler has visibility into the full workflow. This allows opportunities for flow optimization, such as *hoisting* code from adjoining steps into a single step, thus reducing the resources consumed during MapReduce operations.

PMML provides the capability to specify multiple models as a single workflow. This translates directly into Cascading workflows, which can be used to parallelize ensembles and otherwise perform optimization of the logic specified in PMML. The benefits of ensembles in predictive modeling was noted by Breiman[**?**][**?**], in what was called a *multiplicity of data models*. This has borne out in practice in the years since, particularly in the context of the Netflix Prize competitions. Ensemble methods[**?**] were leveraged by the leading competitors for the goal of improving accuracy. For example, the BellKor team blended over 100 individual models[**?**] for their final solution in the 2007 Progress Prize. Learnings from those competitions[**?**] indicate that while the process of combining models adds complexity – in particular, making it more difficult to anticipate or explain predictions – accuracy may be increased substantially. Support for ensemble methods is not generally available in the libraries which export PMML from SAS, R, etc.; however, the Cascading API provides a variety of features through the Pattern project which anticipate this need growing in future work.

This paper explains the integration of PMML and Cascading, using a example application based on the crime dataset from the City of Chicago Open Data. Section 2 introduces the use of Cascading and the Pattern project, based on an example with the Iris data set. It explains how a PMML model can be exported using R, and also how it can be included in Cascading then run on Apache Hadoop. Section 3 explains the process of data prepration for the Chicago crime data set, using a Cascading workflow. Then it shows an example of PMML `MultipleModel` and how this kind of ensemble workflow translates into the corresponding Cascading API calls.

2. CASCADING AND PATTERN

Cascading is an open source project based on Java for developing large-scale data workflows[**?**]. The API was originally introduced in late 2007, as an abstraction layer atop Apache Hadoop[**?**]. Experiences from many large-scale commercial production deployments have informed several iterations of the API over the past five years.

Moreover, several other open source projects have been built on top of Cascading. DSLs have been written for Clojure, Scala, Jython, etc. In general, JVM languages which include functional programming features tend to be efficient for specifying large, complex data pipelines. Cascading provides an integration layer for these – in other words, a kind of middleware – as an abstraction for calls into Hadoop, HBase, Cassandra, Memcached, and other distributed data frameworks. This is valuable as DSLs cut down the development time. Two of the most widely used Cascading DSLs are Cascalog[3] in Clojure and Scalding[4] in Scala. Both projects were developed by machine learning teams at Twitter. Another more recent open source project called Lingual[5] provides ANSI SQL as a DSL. Applications also allow for components in these different languages to be blended together.

Cascading represents a pattern language, based on the functional programming paradigm. According to the notion of pattern languages[**?**] as originally articulated by Christopher Alexander, et al., in the field of architectural design, the syntax of the language is constructed such that when its components fit together then specific best practices are guaranteed for the resulting application. A simple case of pattern language is encountered in the use of Lego®toy blocks. When the blocks snap together, some properties of stability and intended movement can be guaranteed as a toy building scales. This represents a means of conveying expertise and simplifying construction for novices. In the case of Cascading, when components of the API are fit together correctly in an application, there are some guarantees that the JVM compiler and the underlying technologies such as Apache Hadoop will be able to parallelize the workflow. When the components have not been fit together correctly, the compiler rejects the code. This occurs in a *fail fast* mode, long before consuming expensive resources on a large-scale Hadoop cluster. This is important as it enables development of large scale applications, where expertise in Apache Hadoop and other Big Data technologies is generally scarce.

One interesting aspect of Cascading is the use of *literate programming*[**?**] as articulated by Donald Knuth. The API serves as an abstraction layer, separating the business process which defines large-scale data workflows away from the flow planners which generate jobs on Hadoop and other distributed topologies. The flow planners generate graphical documents called flow diagrams to represent each query plan as a *directed acyclic graph* (DAG). Schema, runtime metrics, and other metadata may be used to annotate the DAG for a given application. Within the developer community, these DAGs are used to discuss applications and nuances of the API – oftentimes used rather than reviewing the actual source code. This provides a holistic view and hence is easier to debug.

PMML[6] similarly represents a formal specification for workflows. PMML is capable of capturing the parameters for a predictive model. It also captures metadata about input, transforms for input and output, and moreover can be used to specify ensembles[**?**] of multiple models, or model composition. This kind of workflow business process maps directly into Cascading API calls via the Pattern interface. Cascading can parse the PMML, generate the necessary API components to implement it, then merge those components into the application DAG and apply optimizations. In commercial practices, the data preparation required for a model is often handled in ANSI SQL. This too can be parsed by Cascading, which generates the necessary API components, then

[3]https://github.com/nathanmarz/cascalog/wiki
[4]https://github.com/twitter/scalding/wiki
[5]http://www.cascading.org/lingual/
[6]http://www.dmg.org/v4-1/GeneralStructure.html

merges those into a DAG. The resulting application from the compiler's perspective is one integrated DAG, which is one consistent space for applying optimizations (e.g., moving predicates), for troubleshooting error conditions, handling exceptions, instrumenting for notifications, analyzing utilization rates, etc. In this way the business logic represented by PMML and SQL components combine seamlessly into a single application, as a single JAR file.

2.1 Generating PMML from R

In this section, we use statistical language R to train a Random Forest[?] model based on the popular *Iris* data set and export a PMML file. An example from the Iris data shows the Iris flow species versus a few measured attributes, such as sepal length:

```
sepal_length sepal_width petal_length
petal_width species predict
5.1 3.5 1.4 0.2 setosa setosa
4.9 3.0 1.4 0.2 setosa setosa
5.6 2.5 3.9 1.1 versicolor versicolor
5.9 3.2 4.8 1.8 versicolor virginica
6.3 3.3 6.0 2.5 virginica virginica
4.9 2.5 4.5 1.7 virginica versicolor
```

The R script uses the measured attributes as independent variables, to classify the species as the dependent variable:

```
library(pmml)
library(randomForest)
require(graphics)

# split data into test and train sets
data(iris)
iris_full <- iris
colnames(iris_full) <- c("sepal_length",
   "sepal_width", "petal_length",
   "petal_width", "species")

idx <- sample(150, 100)
iris_train <- iris_full[idx,]
iris_test <- iris_full[-idx,]

# train a Random Forest model
f <- as.formula("as.factor(species) ~ .")
fit <- randomForest(f, data=iris_train,
    proximity=TRUE, ntree=50)
pred <- predict(fit, iris_test, type="class")

# export PMML
saveXML(pmml(fit), file="iris.rf.xml")
```

When run, the R script reports on the predictive power of the Random Forest classifier:

```
        OOB estimate of  error rate: 5%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         32          0         0  0.00000000
versicolor      0         26         2  0.07142857
virginica       0          3        37  0.07500000
```

The script generates trained model as PMML using the Rattle[7] open source library.

[7]http://rattle.togaware.com/

2.2 Model Scoring in Cascading

Given the PMML file, one can import the model into Cascading. A pre-defined application runs model scoring on Hadoop, using the PMML file as a command line argument:

```
hadoop jar build/libs/pattern-examples-*.jar
    data/iris.rf.tsv out/classify
    --pmml data/iris.rf.xml
```

That command line causes the Cascading application to parse the PMML, generate API components to implement the required workflow, then generate an Apache Hadoop job to run the model in parallel. A portion of the output from running this model on Hadoop looks like:

```
$ head out/classify/part-00000
sepal_length  sepal_width   petal_length
petal_width species predict score
5.1 3.5 1.4 0.2 setosa  setosa  setosa
4.9 3   1.4 0.2 setosa  setosa  setosa
4.7 3.2 1.3 0.2 setosa  setosa  setosa
4.6 3.1 1.5 0.2 setosa  setosa  setosa
```

The `predict` column represents the scores from the model. Source code required to implement a PMML workflow is quite succinct:

```
FlowDef flowDef = FlowDef.flowDef()
  .setName("classifier")
  .addSource("input", inputTap)
  .addSink("classify", classifyTap);

PMMLPlanner pmmlPlanner = new PMMLPlanner()
  .setPMMLInput(new File(pmmlPath))
  .retainOnlyActiveIncomingFields();

flowDef.addAssemblyPlanner(pmmlPlanner);

Flow classifyFlow = flowConnector.connect(flowDef);
 classifyFlow.writeDOT("dot/classify.dot");
 classifyFlow.complete();
```

Assuming that the PMML file location is given by a string `pmmlPath`, and that the input and output data sets are given by `inputTap` and `classifyTap` respectively, that Java code will run the model scoring in parallel on Hadoop.

Note that the Java code is relatively independent of the model type in PMML. Internally, there may be Random Forest, Logistic Regression, K-Means Clustering, etc., or ensembles or model chains. The portions exposed to the Cascading application are the `DataDictionary`[8] and `MiningSchema`[9] elements. These PMML elements correspond to tuple schema in Cascading, defining metadata for the fields in the data tuples which flow through assemblies of Cascading pipes.

Figure 1 shows a conceptual flow diagram for a similar Cascading application. In this case, a data sample dataset gets scored by a PMML model which implements a classifier and outputs the classification. Additional logic can be added to allow for optional regression testing and creating a confusion matrix.

Note that the PMML model gets parsed on the client side, before the application is submitted to the Hadoop scheduler. This allows the parser to detect errors and fail fast before consuming expensive compute resources on the cluster.

[8]http://www.dmg.org/v4-1/DataDictionary.html
[9]http://www.dmg.org/v4-1/MiningSchema.html

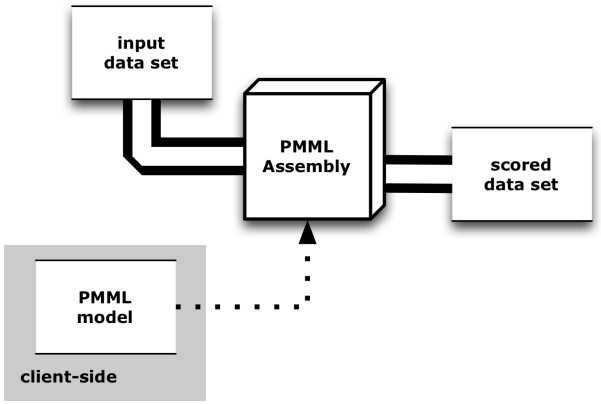

Figure 1: Conceptual flow diagram of a sample Cascading application using PMML.

3. CHICAGO CRIME DATA

In this section, we demonstrate how the Pattern project can be applied for use in a real problem. The City of Chicago has released much of its municipal data[10] to the public. Their stated intent is to promote access to government data and encourage the development of applications which engage and benefit their community. Over 200 data sets have been published and are available via an online portal web site. Most of these data sets are available as comma-separated values (CSV).

One of the data sets[11] lists the reported incidents of crime in Chicago since 2001. For each incident, the data set lists the date, location, case number, various crime classifications (FBI, etc.), plus an index to both a *ward* and a *community area*. There are less than 50 wards, and less than 100 community areas. The former represent political divisions, while the latter are used for collecting US Census data. Analysts working on Chicago demographics have noted that the ward numbers can be problematic given that ward boundaries change frequently based on political motivations[12]. The community areas also change; however, these are relatively more stable and therefore more useful for aggregating the crime data.

Other data sets are also indexed by these same community areas. In addition to the crime reports, data sets for census and predominant language spoken are joined based on community area index. These help enrich the crime data with additional signals to boost predictions.

Data preparation is performed in a Cascading application, which results in a data cube used to train classifiers in R. This code can be reused for data preparation in production. An open source code repository for this applicaton is avaialble on GitHub[13].

First the crime data is read as CSV, with an assertion used to remove rows with null values. Then a regular expression is used to extract the date components: month, hour, week. These values plus the year, crime category, and community area index comprise the dimensions for the data cube. A `GroupBy` aggregates on these dimensions, to obtain counts for each crime category. Those results get merged with defaults, so that any missing crime categories have counts set to zero. Then the census and predominant language spoken get joined, using a `HashJoin` for a replicated join. This provides an optimization when run in MapReduce, since the latter two data sets are relatively smaller and can be replicated in memory. The final data cube is then written to output, for training use in R.

A conceptual flow diagram in Figure 2 illustrates the business logic for this workflow. The actual flow diagram for the parallelized query generated by the Cascading flow planner is shown in Figure 3.

3.1 Ensemble Scoring

This section explains the modeling: how a complex model of ensembles is created for the Chicago crime data set, how multiple models are used, and the way in which it gets mapped into Cascading as a data workflow. Currently, Pattern supports Random Forest Ensemble. A inital version of the support for Ensemble of Ensembles is in the GitHub repo.[14]

First, data is prepared on Hadoop using a Cascading app to create the Chicago crime data cube. R is then used to build models for different crime categories based on that data cube. There are two Random Forest Regression models trained in R and exported as PMML files. These are composed into an ensemble in PMML[**?**] using the `selectFirst` method for segmentation. A simple predicate divides the data into individual crime buckets and applies the RF models for each.

```
<Segmentation multipleModelMethod="selectFirst">
  <Segment id="1">
   <SimplePredicate field="crime_id"
   operator="equal" value="PROSTITUTION" />
   <MiningModel modelName="randomForest_Model"
   functionName="regression">
. . .
   </MiningModel>
</Segment>
<Segment id="2">
   <SimplePredicate field="crime_id"
   operator="equal" value="THEFT" />
   <MiningModel modelName="randomForest_Model"
   functionName="regression">
. . .
   </MiningModel>
</Segment>
</Segmentation>
```

As shown in Figure 4, the sample app for Pattern reads the ensemble PMML file and creates a `SubAssembly` in Cascading. A branch in the workflow is created for each predicate, using the `Filter` operation in Cascading. The first branch has a simple predicate comparing *crime id* and 'PROSTITUTION', and the second branch compares *crime id* and 'THEFT'. Cascading applies the corresponding RF models in the ensemble for each branch. The resulting scored data

[10]http://www.cityofchicago.org/city/en/narr/foia/CityData.html
[11]https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
[12]http://www.robparal.com/ChicagoCommunityAreaData.html
[13]https://github.com/ceteri/ChicagoCrime
[14]https://github.com/girish-a1/pattern-2/tree/wip-1.0-multi

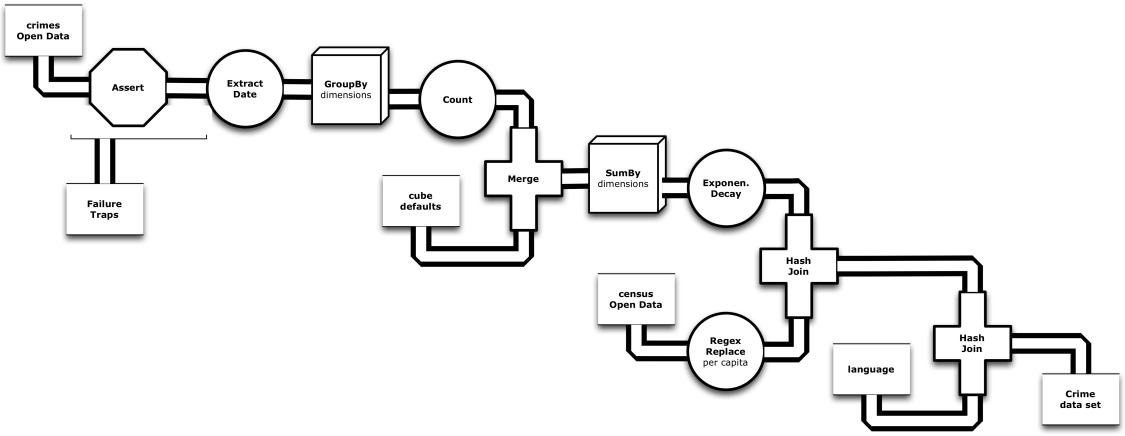

Figure 2: Conceptual flow diagram of the data preparation.

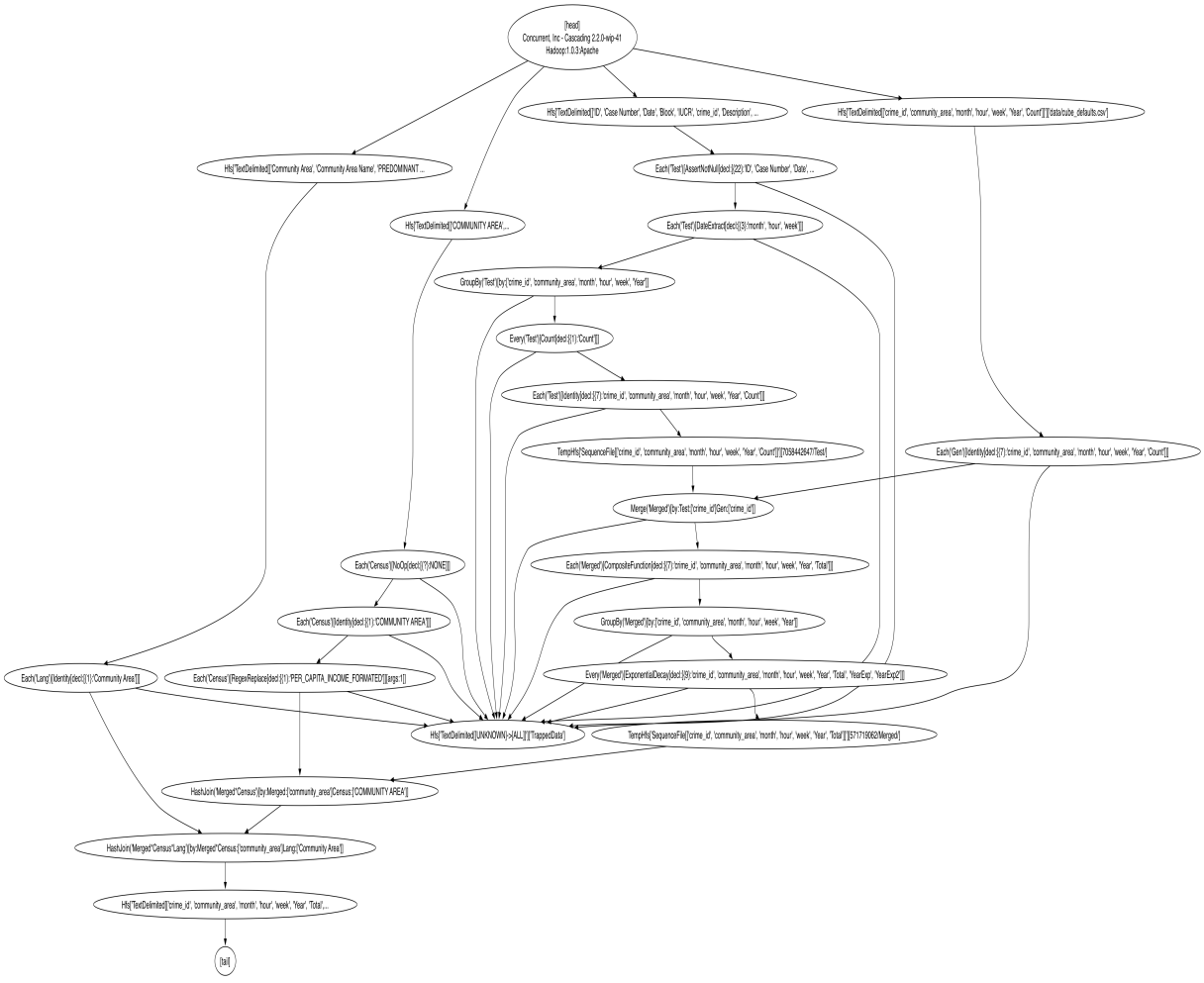

Figure 3: Flow diagram generated by the Cascading flow planner.

from each branch gets combined using a `Merge` or `GroupBy` with and aggregation function in Cascading to produce the final output. The selection depends on the value specified by `multipleModelMethod` in PMML. This approach shows how `MultipleModel` and `Ensemble` can be interpreted by the Cascading API.

4. CONCLUSIONS AND FUTURE WORK

PMML makes the process of exporting and migration predictive models onto other platforms quite simple and cost-effective. Without any coding required, the PMML exported from R may be run on Apache Hadoop clusters at scale via Cascading. Moreover, there are significant cost benefits for commercial deployments as a result of using open source.

There are also interesting points of overlap between PMML and Cascading, since both provide a means for formal specification of data workflows. PMML is specific to predictive models; however, many industry use cases for Cascading involve predictive models at scale. Cascading tends to differentiate from other "Big Data" technologies in that it is often used for integrating Hadoop with legacy environments. Therefore PMML provides a vector for migrating workloads off of SAS, SPSS, etc., onto Hadoop clusters for more cost-effective scaling.

In terms of work with ensembles, there are many cases where this approach can be used for improving predictive models in industry. Customer segmentation in e-commerce is one clear example. PMML provides excellent features for representing ensembles. With the addition of the Pattern library, the Cascading API can readily translate PMML into parallelized, scalable apps running on Hadoop. The project described in this paper shows that Cascading can integrate new types of PMML workflows, even if there are not many ways to generate ensembles in PMML currently. Other than Random Forest, which is by definition an ensemble, there are not many analytics platforms which support creating ensembles of different models. In this project, it was necessary to compose the `MultipleModel`[15] in PMML manually. It is hoped that analytics platforms such as SAS, R, etc., will begin to provide support for creating ensembles, as well as their export as PMML.

One platform which is experiencing much growth in adoption for analytics is Python, and the related packages such as NumPy, SciPy, and Pandas[16]. Given the ease and flexibility of working with XML in Python, it may become a good means of experimenting with support for creating ensembles and their PMML export.

One of the driving motivations for the Cascading developers to incorporate PMML was to collect user data about machine learning use cases in industry, to try to learn ways in which the Cascading API could be extended or modified to support this area of application development better. Clearly these initial attempts with ensemble models in Pattern have opened up new kinds of workflows which could become useful in industry, and expand the range of use cases for Cascading. It may be the case that ensemble model creation in PMML would be handled most directly within extensions to the Pattern project.

4.1 Future Work

There are several organizations collaborating on the Pattern project, extending it in a few directions. Support for scoring models based on additional algorithms is currently being developed. In particular, the library is being extended to handle support vector machine (SVM) and neural networks (NN). Along with the addition of other algorithms, the project is also being extended to support model creation at scale, based on Hadoop. For example, there are many use cases for training regression models based on very large scale data, and this becomes cost-effective on Hadoop.

Cascading has another kind of flow planner for *local mode*, which allows for workflows planned without the Hadoop APIs underneath. In other words, the workflow can run in local memory on a single server. This allows for integration into near real-time services, such as web services. There are projects in progress to utilize Cascading, Pattern, and PMML to deploy the same predictive models for use cases at scale in batch (Hadoop) as well as real-time web services.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge their employers, Concurrent and AgilOne respectively, for the opportunity to collaborate on this open source project, and also to the City of Chicago for providing the input data sets through their open data portal. The authors also acknowledge assistance from Rob Paral in interpreting the Chicago data sets, and also acknowledge Daragh Sibley for feedback and suggestions.

This work is based on Cascading, and much credit is due to the API author, Chris Wensel. PMML export from R was based on the Rattle package, by Togaware.

6. REFERENCES

[1] Christopher Alexander, Sara Ishikawa, Murray Silverstein. *A Pattern Language: Towns, Buildings, Constructions* New York, NY: Oxford University Press (1977).

[2] Robert M. Bell, Yehuda Koren, Chris Volinsky. 'The BellKor solution to the Netflix prize.' *KorBell Team's Report to Netflix* (2007).

[3] Leo Breiman. 'Random forests.' *Machine Learning*, 45, 5-32 (2001).

[4] Leo Breiman. 'Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author).' *Statistical Science*, v16 n3 (2001), 199-231.

[5] Jeffrey Dean, Sanjay Ghemawat. 'MapReduce: simplified data processing on large clusters.' *Communications of the ACM*, 51.1 (2008): 107-113.

[6] R Grossman, S Bailey, A Ramu, B Malhi, P Hallstrom. 'The management and mining of multiple predictive models using the predictive modeling markup language.' *Information and Software Technology*, v41 n9 (1999).

[7] Alex Guazzelli, Wen-Ching Lin, Tridivesh Jena. *PMML in Action: unleashing the power of open standards for data mining and predictive analytics.* Seattle, WA: CreateSpace (2010).

[8] Trevor Hastie, Robert Tibshirani, J H Friedman. *The Elements of Statistical Learning: data mining, inference, and prediction.* New York, NY: Springer (2001).

[15] http://www.dmg.org/v4-1/MultipleModels.html
[16] http://pandas.pydata.org/

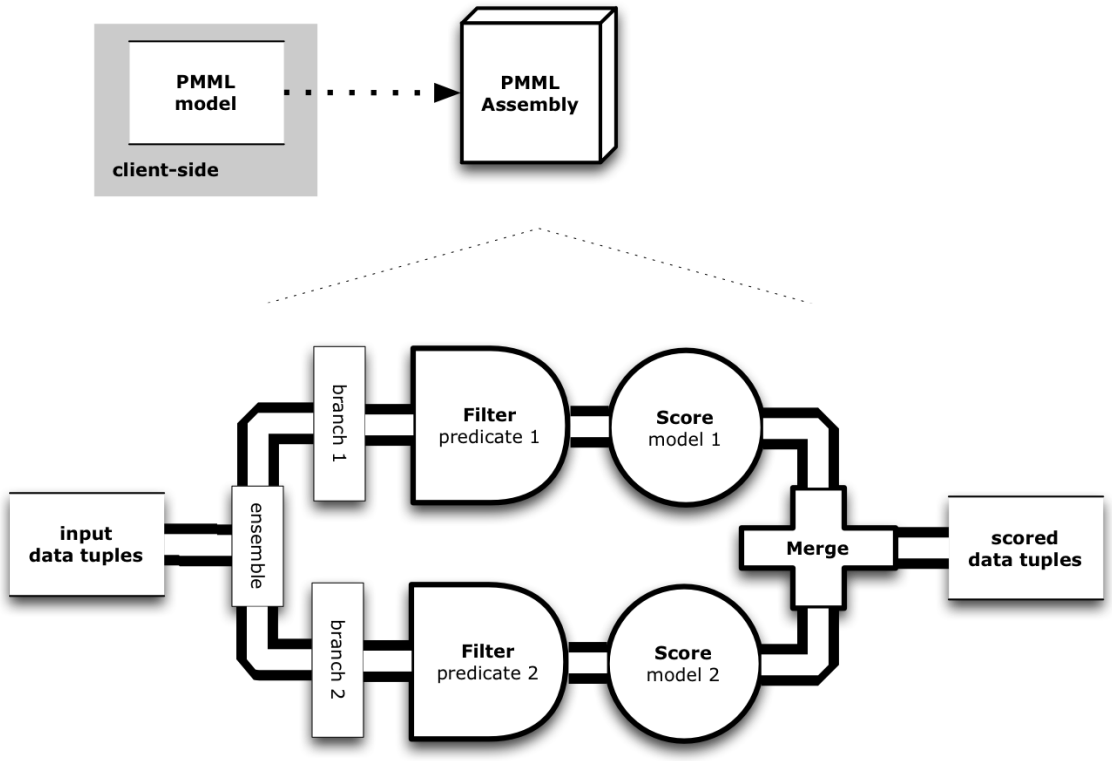

Figure 4: Conceptual flow diagram for the Cascading assembly which implements the PMML ensemble.

[9] Todd Holloway. 'Ensemble Learning: Better Predictions Through Diversity.' *ETech* (2008) [17]

[10] Donald E. Knuth. *Literate Programming* Stanford, CA: Stanford University Center for the Study of Language and Information (1992).

[11] Lester Mackey. 'The Story of the Netflix Prize: An Ensemblers Tale.' *National Academies Seminar* Washington, DC (2011) [18]

[12] Paco Nathan. *Enterprise Data Workflows with Cascading* Sebastopol, CA: O'Reilly (2013).

[13] Tom White. *Hadoop: the definitive guide* Sebastopol, CA: O'Reilly (2012).

[17] http://abeautifulwww.com/EnsembleLearningETech.pdf
[18] http://www.stanford.edu/~lmackey/papers/