Towards the Realization of a DSML for Machine Learning: A Baseball Analytics Use Case

Kaan Koseler and Matthew Stephan

Department of Computer Science and Software Engineering
Miami University
Oxford, OH, USA
Email: {koselekt , stephamd}@miamioh.edu

Abstract. Using machine learning (ML) for big data is challenging, requiring specialized knowledge of the domain, learning algorithms, and software engineering. To demonstrate the viability of model-driven engineering in the ML domain we consider an ML use case of baseball analytics by extending and applying an existing, but untested, ML domain specific modeling language (DSML). Additionally, we aim to make ML software development more accessible and formalized, and help facilitate future research in this area. This paper describes our plan, initial work, and anticipated contributions in extending, testing, and validating this DSML, and implementing a code generation scheme that is targeted at a binary classification baseball problem.

Keywords: Model driven engineering · Domain specific modeling languages · Machine Learning · Analytics · Baseball

1 Introduction

The growth of big data analytics and machine learning (ML) has introduced several new challenges for software engineers. Due to the esoteric nature of the field, it is difficult to find software engineers that can develop and maintain applications incorporating machine learning techniques to analyze data [2] who also have the requisite domain knowledge.

One approach proposed to address some of these challenges is to use domain specific models for big data analysis using machine learning, such as the domain specific modeling language (DSML) proposed by Brueker [1]. This language is a proof of concept only, and yet to be realized in practice or tested. To that end, and for the purpose of demonstrating the practicality and potential of applying model driven engineering (MDE) to ML, we plan on updating that DSML, implementing its code generation, and applying it to a baseball analytics use case. We choose baseball due to the discrete nature of its data, its popularity and relatability, and the wealth of available data compiled over its long history. Also, ML techniques are increasingly being incorporated into baseball to the point that each team now has analytics departments and millions of dollars invested [4]. This paper summarizes our approach, including preliminary work, and our expected deliverables.

2 Background Information

We provide background information, omitting background on model driven engineering and domain specific modeling languages, assuming the reader has this knowledge.

2.1 Machine Learning - Binary Classification

While there are many classes of ML problems, we focus on the one relevant to our use case, Binary Classification. Binary classification is perhaps the best understood problem in machine learning [6]. Given a set of observations in a domain, X, and their target values, Y, as training data, determine the values Y on the test data, where Y is a binary value that classifies the observations. In general, the values of Y are referred to as either positive or negative.

2.2 Baseball Analytics - Pitch Prediction

We consider the important binary classification baseball domain problem of classifying pitches into fastball and non-fastball categories. This problem was first explored by Ganeshapillai and Guttag [3]. They considered a large feature vector of pitcher-batter match up data and game state, and used that to predict whether the next pitch thrown would be a fastball. They achieved an 18% improvement in prediction accuracy over a basic naive classifier. Their training data was drawn from the 2008 MLB season and their test data was drawn from the 2009 MLB season.

3 Related Work

A common tool for representing a ML model is a probabilistic graphical model (PGM). These are visual representations comprised primarily of random variables, with factor graphs explicitly designating joint distributions. These are helpful from a design perspective, but it is difficult to establish a one-to-one correspondence between a PGM and its resulting code implementation. One of the primary advantages of employing a model-driven approach is making this correspondence more explicit.

The only DSML we discovered designed for modeling machine learning was first proposed by Breuker [1]. They propose a DSML, which we omit for space, to build ML models graphically rather than through code. There is no code generation scheme, but Breuker does lay out a skeleton consisting of three methods within a single C# class using the Infer.NET C# library [7]. This library builds ML models by having users define variables and connect them with factors, allowing execution of a given inference algorithm.

4 Proposed Approach

We will update and validate Breuker's DSML, and apply it to a real-world problem for the first time, including creation of a code generation scheme and model instances. Our use case involves classifying baseball pitches into fastball and

non-fastball categories, which has been done in the past manually, achieving an 18% improvement over a naive classifier [3].

Figure 1 presents our simplified example model instance for classifying pitches. It uses the "count" to predict the pitch. The "count" consists of the current number of strikes and balls that have been accrued in the match up. The factor node represents the relationship between the variables, and we are interested in $P(Fastball|Balls\ \&\ Strikes)$. This factor, when using the Bayes Point Machine method, produces an array of weights that will be used for inferring the fastball distribution. In reality, we will include as many as 37 factors [3] in our model instances.

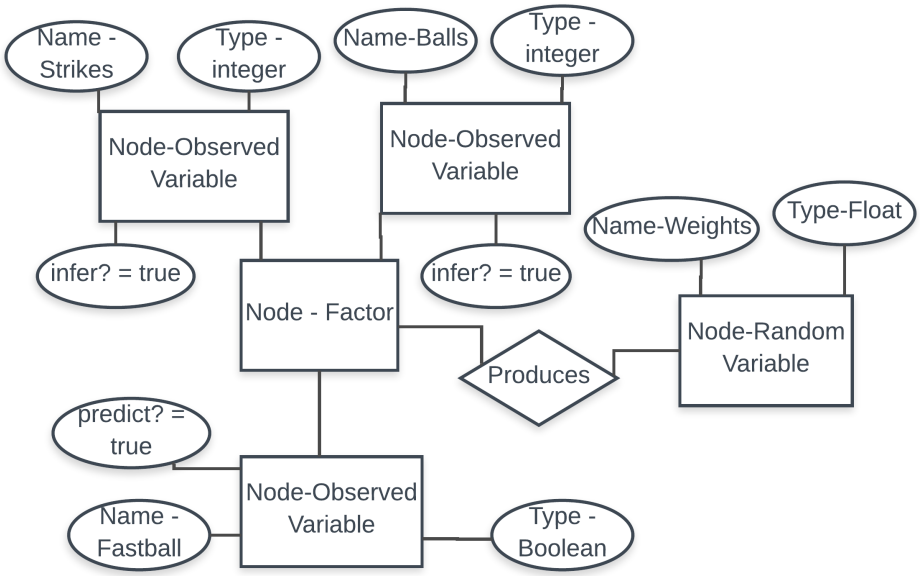


Fig. 1. Basic example model instance conforming to Breuker's DSML

Our example generated setup code from this model instance, shown underneath the labeled training data in Figure 2, begins by initialization of the variables containing the training data into arrays. Different model instances will have different numbers of variables and different respective data types. In this example, we have balls, strikes, and a boolean fastball variable. We initialize the weight vector with an identity matrix that will later be used for inference. This is followed by a call to the BayesPointMachine method, which uses the training data to calculate and update the inference weights. Both our example and an actual generated system would make calls to a generated inference engine for posterior weight distribution and predictions, such as the one shown in the latter half of Figure 2.

We present an example of potentially generated code for our BayesPointMachine method in Figure 3. It collects training data in one vector, $xVector$, and

calculates the dot product of the weights w and $xVector$ for each data point, storing it in the target vector, y.

```
// The labeled training data
double[] strikes = { 2, 1, 2, 0, 1, 0 };
double[] balls = { 3, 3, 0, 2, 1, 0 };
bool[] fastball = { true, false, true, true, false, false };

//create target vector and weight vector
VariableArray<bool> y = Variable.Observed(fastball).Named("y");
Variable<Vector> w = Variable.Random(new VectorGaussian(Vector.Zero(2),
    PositiveDefiniteMatrix.Identity(2))).Named("w");

//Call the bayes point machine method
BayesPointMachine(strikes, balls, w, y);

//Create inference engine object and infer
//posterior distribution of weights
InferenceEngine engine = new InferenceEngine();
VectorGaussian wPosterior = engine.Infer<VectorGaussian>(w);
Console.WriteLine("Dist over w=\n" + wPosterior);

//Make predictions on test data
double[] strikesTest = { 2, 1, 2 };
double[] ballsTest = { 3, 2, 0 };
VariableArray<bool> ytest = Variable.Array<bool>(new Range(strikesTest.Length)).Named("ytest");
BayesPointMachine(strikesTest, ballsTest, Variable.Random(wPosterior).Named("w"), ytest);
Console.WriteLine("output=\n" + engine.Infer(ytest));
```

Fig. 2. Example Setup and Call to the BayesPointMachine Method, and Inference

```
public void BayesPointMachine(double[] strikes, double[] balls, Variable<Vector> w, VariableArray<bool> y)
{
    // Create training data vector
    Range j = y.Range;
    Vector[] xVector = new Vector[balls.Length];
    for (int i = 0; i < xVector.Length; i++)
        xVector[i] = Vector.FromArray(strikes[i], balls[i]);
    VariableArray<Vector> x = Variable.Observed(xVector, j).Named("x");

    // Bayes Point Machine, dot product of weights and feature vector
    y[j] = Variable.GaussianFromMeanAndVariance(Variable.InnerProduct(w, x[j]).Named("dotProduct")) > 0;
}
```

Fig. 3. Projected Generated Code for our Bayes Point Machine

4.1 Scope

Our work will not include a defined code generation scheme for the entirety of Breuker's DSML. The scope of our code generation is limited strictly to modeling this binary classification problem and using a Bayes Point Machine created in Infer.NET to make predictions. There also exists the possibility that Breuker's DSML syntax contains heretofore unknown errors or deficiencies. In that case, a critical component of our approach will be to find and correct any of these errors.

4.2 Evaluation and Anticipated Results

To evaluate our work and validate the DSML, we will use the same 37 features, training data and test data that Ganeshapillai and Guttag used when developing their solution through source code. We will then compare our results to theirs by evaluating the percentage improvement over a naive classifier. We anticipate comparable results to their 18% improvement. Any improvement less than 18%, while not ideal, can be viewed as a trade off for gaining MDE benefits, including automation and formalism [5]. Additionally, we will demonstrate the flexibility afforded in an ML MDE approach by developing multiple model instances, generating systems that consider different combinations and subsets of the 37 features.

5 Project Deliverables

We anticipate making a number of contributions in this project:

1. Extended, and now validated, ML DSML
2. Instances of the DSML with different combinations of the 37 factors [3]
3. Code generation scheme for this binary classification problem using the Infer.NET library
4. Executable systems generated from our model instances
5. Results from running generated systems on training and test data, with an evaluation of prediction accuracy

Acknowledgments

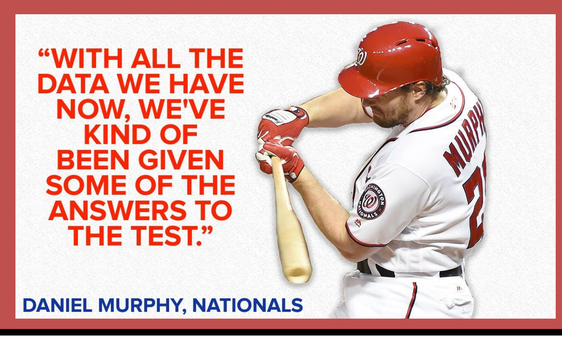
This work is supported by the Miami University Senate Committee on Faculty Research (CFR) Faculty Research Grants Program.

References

1. Breuker, D.: Towards model-driven engineering for big data analytics–an exploratory analysis of domain-specific languages for machine learning. In: Hawaii International Conference on System Sciences. pp. 758–767 (2014)
2. DeLine, R.: Research opportunities for the big data era of software engineering. In: International Workshop on Big Data Software Engineering. pp. 26–29 (2015)
3. Ganeshapillai, G., Guttag, J.: Predicting the next pitch. In: Sloan Sports Analytics Conference (2012)
4. Sawchik, T.: Big Data Baseball: Math, Miracles, and the End of a 20-Year Losing Streak. Macmillan (2015)
5. Schmidt, D.C.: Model-driven engineering. Computer 39(2), 25 (2006)
6. Smola, A., Vishwanathan, S.: Introduction to Machine Learning. Cambridge University Press (2008)
7. Wang, S.S., Wand, M.P.: Using Infer.NET for statistical analyses. The American Statistician 65(2), 115–126 (2011)

Towards the Realization of a DSML for Machine Learning: A Baseball Analytics Use Case

Kaan Koseler & Matthew Stephan
Miami University
{koselekt, stephamd} @miamioh.edu



Motivation

- Building quality Machine Learning (ML) software is challenging
  - Requires domain, ML, and software engineering knowledge
  - Difficult for organizations to find individuals possessing requisite knowledge
  - MDE holds promise in facilitating development of ML software
- Our goal: Demonstrate feasibility of applying MDE in ML domain through a baseball analytics use case

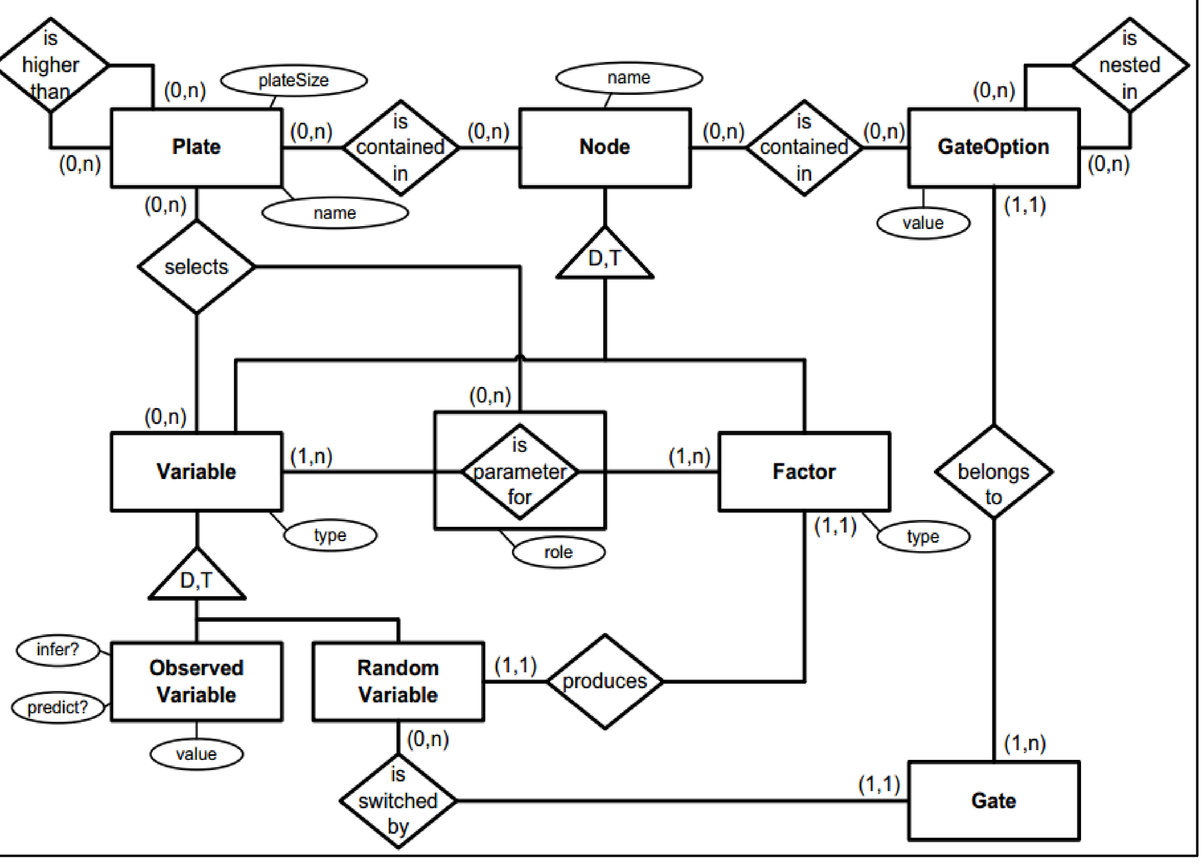
Details of Approach

- Plan on extending and applying proposed, but untested, DSML (Breuker, 2014) built on the Infer.NET C# library
- Implement code generation
- Apply it to a specific Baseball Analytics use case and compare against existing work
  - Binary classification problem: Predicting the next pitch in a baseball game (Fastball or Non-Fastball) (Ganeshapillai & Guttag, 2012)
- Construct several different model instances with 37 different features (Home/Away game, Handedness, etc.)

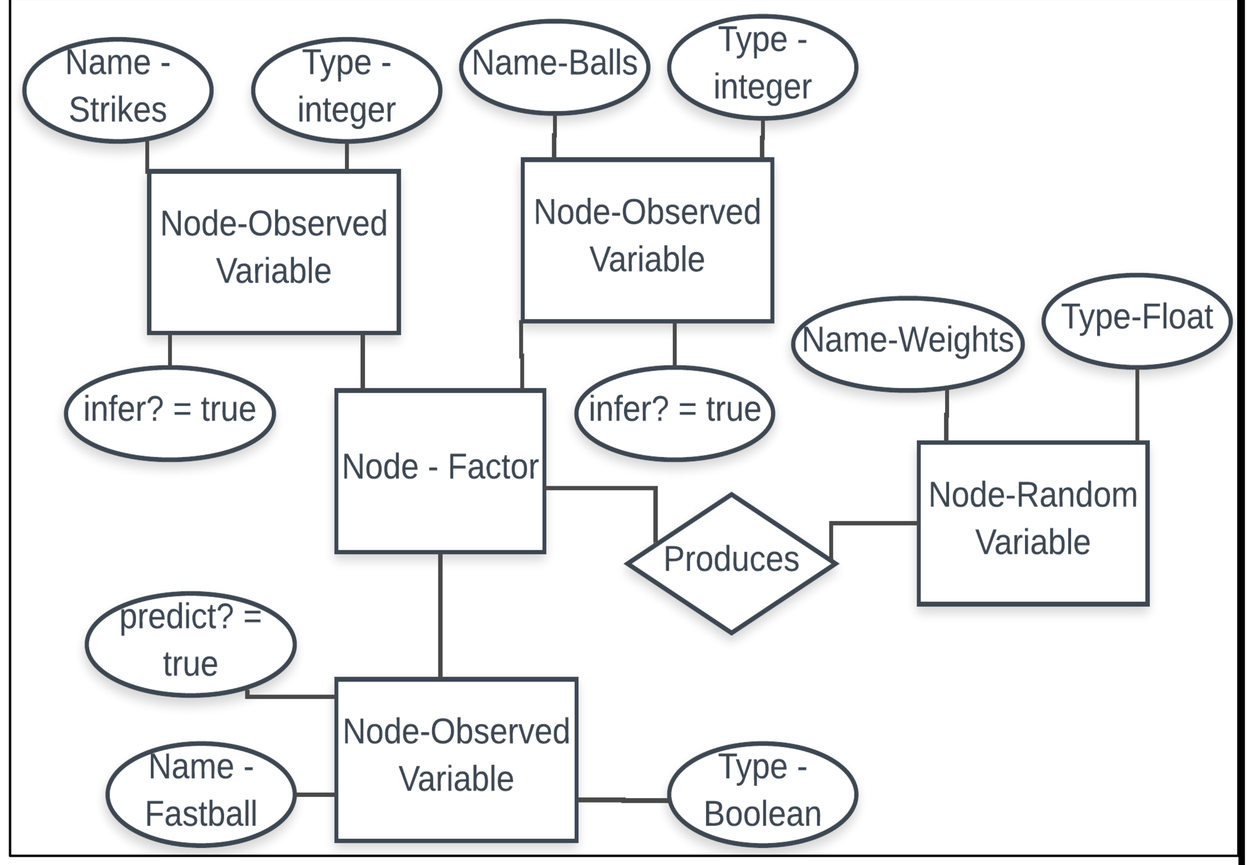
Initial Example

- Model instance uses count (# of balls and strikes) as a feature to predict the next pitch
- Contains three observed variables with a factor representing the relationship between them
- This relationship ultimately produces weights to be used for inference and prediction
- This DSM models 2 features, whereas our target DSM will model 37

Proposed DSML (Breuker 2014)



Sample Model Instance



Sample Training Code

```
// The labeled training data
double[] strikes = { 2, 1, 2, 0, 1, 0 };
double[] balls = { 3, 3, 0, 2, 1, 0 };
bool[] fastball = { true, false, true, true, false, false };

//create target vector and weight vector
VariableArray<bool> y = Variable.Observed(fastball).Named("y");
Variable<Vector> w = Variable.Random(new VectorGaussian(Vector.Zero(2),
    PositiveDefiniteMatrix.Identity(2))).Named("w");

    //Call the bayes point machine method
    BayesPointMachine(strikes, balls, w, y);
}

public void BayesPointMachine(double[] strikes, double[] balls, Variable<Vector> w, VariableArray<bool> y)
{
    // Create training data vector
    Range j = y.Range;
    Vector[] xVector = new Vector[balls.Length];
    for (int i = 0; i < xVector.Length; i++)
        xVector[i] = Vector.FromArray(strikes[i], balls[i]);
    VariableArray<Vector> x = Variable.Observed(xVector, j).Named("x");

    // Bayes Point Machine, dot product of weights and feature vector
    y[j] = Variable.GaussianFromMeanAndVariance(Variable.InnerProduct(w, x[j]).Named("dotProduct")) > 0;
}
```

Sample Inference Code

```
//Create inference engine object and infer
//posterior distribution of weights
InferenceEngine engine = new InferenceEngine();
VectorGaussian wPosterior = engine.Infer<VectorGaussian>(w);
Console.WriteLine("Dist over w=\n" + wPosterior);

//Make predictions on test data
double[] strikesTest = { 2, 1, 2 };
double[] ballsTest = { 3, 2, 0 };
VariableArray<bool> ytest = Variable.Array<bool>(new Range(strikesTest.Length)).Named("ytest");
BayesPointMachine(strikesTest, ballsTest, Variable.Random(wPosterior).Named("w"), ytest);
Console.WriteLine("output=\n" + engine.Infer(ytest));
```

References

•Breuker, D.: Towards model-driven engineering for big data analytics–an exploratory analysis of domain-specific languages for machine learning. In: Hawaii International Conference on System Sciences. pp. 758–767 (2014)
•Ganeshapillai, G., Guttag, J.: Predicting the next pitch. In: Sloan Sports Analytics Conference (2012)

Acknowledgements

This work is supported by the Miami University Senate Committee on Faculty Research (CFR) Faculty Research Grants Program