Probabilistic Programming with Infer.NET

John Bronskill

University of Cambridge

19th October 2017
The Promise of Probabilistic Programming

“We want to do for machine learning what the advent of high-level program languages 50 years ago did for the software development community as a whole.”

Kathleen Fisher, DARPA program manager.
Goals of Probabilistic Programming

• Reduce the level of expertise necessary to build machine learning applications

• Compact model code
  • faster to write and easier to understand

• Reduce development time & cost
  • encourage experimentation
Probabilistic Programming Languages

Church
HBC
BLOG
Turing
R2
Anglican
FACTORIE
PyMC
Figaro
ProbLog
BUGS
Hansei
Infer.NET
Dimple
PRISM
Stan
Venture

http://probabilistic-programming.org
A Great Resource

http://www.mbmlbook.com/
Infer.NET
Probabilistic Programming Framework
Background

• Primary authors: Tom Minka and John Winn
  • They were tired of writing inference code by hand
  • Under development for 12+ years
• Design Goals:
  • Fast inference
    • Emphasis on approximate inference (Expectation Propagation)
  • Industrial strength for production environments
  • Write the forward model, everything else automatic
    • Inference code generated by compiler
  • Not a solution to all ML problems
    • e.g. Focused on fully factored models
How Infer.NET Works
Two Coins Example

The Model

Variable<bool> firstCoin = Variable.Bernoulli(0.5).Named("firstCoin");
Variable<bool> secondCoin = Variable.Bernoulli(0.5).Named("secondCoin");
Variable<bool> bothHeads = (firstCoin & secondCoin).Named("bothHeads");
Two Coins Example (con’t)

Inference Query 1 – What is the probability that both coins are heads?

InferenceEngine ie = new InferenceEngine();
Bernoulli query1 = (Bernoulli)ie.Infer(bothHeads);

Probability both coins are heads: Bernoulli(0.25)
Inference Query 2

What is the probability distribution over the first coin given that we observe that bothHeads = false?

\[
\text{bothHeads.ObservedValue = false;}
\]

\[
\text{Bernoulli query2 = (Bernoulli)ie.Infer(firstCoin));}
\]

Probability distribution over firstCoin: Bernoulli(0.3333)
Two Coins Demo
public interface IGeneratedAlgorithm // Generated Inference Code
{
    // Get/Set Observed Values
    void SetObservedValue(string variableName, object value);
    object GetObservedValue(string variableName);

    // Do work
    void Execute(int numberOfIterations);
    void Update(int additionalIterations);
    void Reset();

    // Get computed marginals/posteriors/messages
    object Marginal(string variableName);
    T Marginal<T>(string variableName);
    object Marginal(string variableName, string query);
    T Marginal<T>(string variableName, string query);
    object GetOutputMessage(string variableName);

    // Monitor progress
    int NumberOfIterationsDone { get; }
    event EventHandler<ProgressChangedEventArgs> ProgressChanged;
}
Generated Code Demo
Basic Steps for Using Infer.NET

• Create Model
• Set priors and observe random variables
• Infer posterior(s)
• Use the posterior(s) to make predictions, etc.
Dataset

This data set gives average weights for humans as a function of their height in the population of American women of age 30–39. (Wikipedia)
Linear Regression Model

\[ Y[n] = M * X[n] + B + N[n] \]

- \( M \): slope
- \( B \): \( y \)-intercept
- \( N \): noise

\( M, B, N \): random variables whose uncertainty is represented with a Gaussian distribution.

Gaussian(0, 1)
M = Variable<double>.Random(MPrior);

B = Variable<double>.Random(BPrior);

DataLength = Variable.New<int>();
n = new Range(DataLength);

N = Variable.Array<double>(n);

X = Variable.Array<double>(n);

Y = Variable.Array<double>(n);

Variable.ForEach(n)
{
    N[n] = Variable.GaussianFromMeanAndVariance(0, 1);
    Y[n] = M * X[n] + B + N[n];
}
Linear Regression Training

// set observed values
DataLength.ObservedValue = instanceCount;
X.ObservedValue = xTrainingData;
Y.ObservedValue = yTrainingData;
MPrior.ObservedValue =
    Gaussian.FromMeanAndVariance(0, 1000);
BPrior.ObservedValue =
    Gaussian.FromMeanAndVariance(0, 1000);

// infer slope M and intercept B
var Engine = new InferenceEngine();
MPosterior = Engine.Infer<Gaussian>(M);
BPosterior = Engine.Infer<Gaussian>(B);
Linear Regression Prediction

// set observed values
DataLength.ObservedValue = instanceCount;
X.ObservedValue = xTestData;
MPrior.ObservedValue = MPosterior;
BPrior.ObservedValue = BPosterior;

// infer Y
YPosterior = Engine.Infer<Gaussian[]>(Y);
Infer.NET Regression Demo
TrueSkill Model

TrueSkill Model Code

```
GameCount = Variable.Observed(default(int))
var game = new Range(this.GameCount);

PlayerCount = Variable.Observed(default(int));
var player = new Range(this.PlayerCount);

Skills = Variable.Array<double>(player);
Variable.ForEach(player)
{
    Skills[player] = Variable.
        GaussianFromMeanAndVariance(25.0, 70.0);
}

Winner = Variable.Observed(default(int[]), game);
Loser = Variable.Observed(default(int[]), game);

Variable.ForEach(game)
{
    // Define the player performance as a noisy version of their skill
    var winnerPerformance = Variable.GaussianFromMeanAndVariance(Skills[Winner[game]], 17.0);
    var loserPerformance = Variable.GaussianFromMeanAndVariance(Skills[Loser[game]], 17.0);

    // The winner's performance must be higher than the loser's in a given game
    Variable.ConstrainTrue(winnerPerformance > loserPerformance);
}
```
public static Gaussian[] InferPlayerSkills(int playerCount, int[] winners, int[] losers, double performanceNoiseVariance)
{
    // Instantiate a model
    TrueSkillModel model = new TrueSkillModel(performanceNoiseVariance);

    // Set observed data
    model.GameCount.ObservedValue = winners.Length;
    model.PlayerCount.ObservedValue = playerCount;
    model.Winner.ObservedValue = winners;
    model.Loser.ObservedValue = losers;

    // Infer results
    var engine = new InferenceEngine();
    return engine.Infer<Gaussian[]>(model.PlayerSkills);
}
TrueSkill Demo
Infer.NET Architecture

Factors/Constraints
- Represent relationships between variables, e.g., Boolean, comparison, arithmetic, linear algebra, etc.

Inference Algorithms
- Expectation Propagation (EP)
- Variational Message Passing (VMP)
- Block Gibbs sampling

Distributions
- Used for messages and marginals
- Continuous: Beta, Gaussian, Gamma
- Discrete: Bernoulli, Binomial, Poisson
- Multivariate: Dirichlet, Wishart

Message operators
- The atomic operations of an inference algorithm (100s)

Infer.NET components
- Factors
- Algorithms
- Distributions
- Message operators

Infer.NET framework

.NET 4.0 or higher
Modeling Features

• A model is essentially a factor graph represented in code using Infer.NET APIs
  • Random Variables are represented by .NET variables
  • Factors are represented as .NET methods/functions.
• Sparse and dense arrays are supported
• Probabilistic control flow (loop, conditional, switch) to facilitate plates, gates, and mixtures
• Chain and grid models (i.e. Markov chain) are supported
• Inference on strings
• Extensible: Can write your own factors and distributions
• List of Factors:
  http://infernet.azurewebsites.net/docs/list%20of%20factors%20and%20constraints.aspx
Inference on Strings
Hello uncertain world in Infer.NET

```csharp
// Create two uncertain strings
var a = Variable.UniformOver(Strings.All);
var b = Variable.UniformOver(Strings.All);

// Format the two strings together
var c = Variable.Format("{0} {1}!!", a, b);

// Observe the result
c.ObservedValue = "Hello Uncertain World!!";
```

What is the distribution over a?

Uniform("Hello Uncertain","Hello")

And b?

Uniform("Uncertain World","World")
Learning templates

```javascript
var a = Variable.UniformOver(WordStrings.All);
var b = Variable.UniformOver(WordStrings.All);
// Uncertain template
var template = Variable.UniformOver(Strings.All);
// Format the two strings together
var c = Variable.Format(template, a, b);
// Observe the inputs and outputs
a.ObservedValue = "Hello";
c.ObservedValue = "Hello Uncertain World!!";
```

What is the distribution over `template`?

```
Uniform("{0} Uncertain {1}!!", "{0} {1}!!" , "{0} {1} World!!")
```
Fact Finding

Seed Fact:  
*Date-Of-Birth*
[Barack Obama, August 4, 1961]

Learned Template:  
“{Entity Name} was born on (Entity Birthdate)”

New Fact:  
*Date-Of-Birth*
[George Bush, July 6, 1946]

Search corpus for seed text

Infer Template

Search corpus for text that matches template.

Infer Fact

Observed Text:  
“Barack Obama was born on August 4, 1961.”

Observed Text:  
“George Bush was born on July 6, 1946”
Examples Browser Demo

FirstExample
A simple first example showing the basics of Infer.NET.

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using MicrosoftResearch.Infer.Models;
using MicrosoftResearch.Infer;

namespace MicrosoftResearch.Infer.Tutorials
{
    public class FirstExample
    {
        public void Run()
        {
            Variable<bool> firstCoin = Variable.Bernoulli(0.5).Named("firstCoin");
            Variable<bool> secondCoin = Variable.Bernoulli(0.5).Named("secondCoin");
            Variable<bool> bothHeads = (firstCoin & secondCoin).Named("bothHeads");
            InferenceEngine ie = new InferenceEngine();
            if (!ie.Algorithm.IsVariationalMessagePassing)
            {
                Console.WriteLine("Probability both coins are heads: \+ie.Infer(bothHeads)\);
                bothHeads.ObservedValue=false;
                Console.WriteLine("Probability distribution over firstCoin: \+ie.Infer(firstCoin)\);
            }
            else
                Console.WriteLine("This example does not run with Variational Message Passing");
        }
    }
}
High Scale Applications

Gears of War

Azure Machine Learning

Office 365
Office 365 E-Mail Clutter

Extract Features
- Sender
- Recipients
- To Distribution List?
- Is Meeting Request?
- Has attachment?
...

Classifier
- Clutter
- Not Clutter
Community Training:
1. Compute e-mail features $x$ for many pieces of e-mail for many users.
2. Observe $y$.
3. Infer $w_m$ and $w_p$ for each feature. These are the priors for the weights $w$.

Classification:
1. Compute features $x$ for each e-mail.
2. Use weights $w$ learned for each user to date.
3. Infer $y$ for each e-mail.

Daily Online Training:
1. Train on new user e-mails older than a fixed period of time (e.g. 1 week).
2. Compute e-mail features $x$.
3. Observe $y$ by looking at action the user actually took.
4. Infer new weights $w$ for each feature using current weights as priors.

$w_m$: weight mean
$w_p$: weight precision
$w$: weight
$t$: score
$x$: feature data
$y$: action label
### Recommendation

<table>
<thead>
<tr>
<th>User 10 Million</th>
<th>Item 10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="User 1" /></td>
<td><img src="image2.png" alt="Item 1" /></td>
</tr>
<tr>
<td><img src="image3.png" alt="User 2" /></td>
<td><img src="image4.png" alt="Item 2" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="User 3" /></td>
<td><img src="image6.png" alt="Item 3" /></td>
</tr>
<tr>
<td><img src="image7.png" alt="User 4" /></td>
<td><img src="image8.png" alt="Item 4" /></td>
</tr>
<tr>
<td><img src="image9.png" alt="User 5" /></td>
<td><img src="image10.png" alt="Item 5" /></td>
</tr>
</tbody>
</table>

- **User 10 Million**
  - Rating: 1
  - Rating: 3
  - Rating: ?
  - Rating: ?
  - Rating: ?
  - Rating: ?
  - Rating: ?

- **Item 10,000**
  - Rating: 3
  - Rating: ?
  - Rating: ?
  - Rating: ?
  - Rating: 3
  - Rating: 3
  - Rating: ?

Microsoft Confidential
Recommender Generative Model

**User Traits**
- Prefer escapist
- Prefer serious
- Prefer serious drama
- Prefer action oriented

**Movie Traits**
- Escapist
- Serious drama
- Serious
- Action oriented

**User-Movie Affinity**
- -2.0
Recommendation Model

Matchbox: Large Scale Online Bayesian Recommendations.
Why .NET/C#?

• Pros
  • Industrial strength, high-scale production ready
  • Easily connect to large databases
  • Threading, parallel, asynchronous processing
  • High quality profilers and debuggers

• Cons
  • Not as nice for rapid prototyping as Python or MatLab
  • Weaker on visualisation/plotting
Issues

• Not really a language
  • Set of APIs for doing Bayesian inference on graphical models

• Inference
  • Inference (e.g. EP) not always possible for all models.
  • Can fail in mysterious ways (model symmetry, lack of convergence, etc.).
  • Hard to know whether inference actually worked.
  • Tricky to debug – no inference debugger.

• Performance
  • Compiler only generates single-threaded code
  • Can manually “parallelise”, but this can be difficult
Current Status

• Actively worked on at Microsoft Research Cambridge
  • New features aimed at supporting specific research interests or products

• License
  • Currently: academic use only
    • No commercial license
Thanks and Questions

• Infer.NET

http://research.microsoft.com/infernet