Infer.NET and CSoft
A framework and language for Machine Learning

John Winn
Machine Learning and Perception Group

NIPS, December 2008
How machine learning is applied

Current approach

1. Define probabilistic model
2. Choose inference method
3. Derive algorithm by hand
4. Implement algorithm (e.g., Matlab)
5. Revise model/method
6. Re-implement algorithm (e.g., C++/C#)
How machine learning is applied

**Current approach**

1. Define probabilistic model
2. Choose inference method
3. Derive algorithm by hand
4. Implement algorithm (e.g. Matlab)
5. Revise model/method
6. Re-implement algorithm (e.g. C++/C#)

**New approach**

1. Define probabilistic model
2. Write model in modelling language
3. Apply inference engine
4. Revise model/engine settings
Example: BossPredictor

- $P(C=1) = 0.6$
- $P(R=1) = 0.8$

Coffee $\Rightarrow$ Good mood $\Rightarrow$ Will approve my trip

$P(G=1 \mid C,R) = \begin{cases} 0.9 & \text{if } C \mid \sim R \\ 0.2 & \text{otherwise} \end{cases}$

$P(L=1 \mid G) = \begin{cases} 0.9 & \text{if } G=1 \\ 0.4 & \text{otherwise} \end{cases}$
Hand-coded solution (Matlab)

% Perform variable elimination on BossPredictor model
% Model specification
PCoffee = [0.4 0.6];
PRaining = [0.2 0.8];

PGoodMood(2,:,:,:) = [0.9 0.2;0.9 0.9];
PGoodMood(1,:,:,:) = 1 - PGoodMood(2,:,:,:);

PLikesIdea(2,:) = [0.4 0.9];
PLikesIdea(1,:) = 1 - PLikesIdea(2,:);

% Add observation
PRaining = [0 1];

%% Perform variable elimination
% Eliminate coffee
PGoodMood2 = zeros(2,2);
for coffee=1:2
    PGoodMood2 = PGoodMood2 + squeeze(PGoodMood(:,coffee,:)*PCoffee(coffee));
end

% Eliminate raining
PGoodMood3 = zeros(2,1);
for raining = 1:2
    PGoodMood3 = PGoodMood3 + PGoodMood2(:,raining)*PRaining(raining);
end

% Eliminate good mood
PLikesIdea2 = zeros(2,1);
for goodMood = 1:2
    PLikesIdea2 = PLikesIdea2 + PLikesIdea(:,goodMood)*PGoodMood3(goodMood);
end
PLikesIdea2(2)
Example: BossPredictor

Coffee → Good mood → Approved my trip=1
Raining=1 → Good mood → Approved my trip=1

Will approve my friend’s trip
BossPredictor with Inference Engine

P(C=1) P(R=1)

Coffee Raining=1

P(G=1 | C,R)

Good mood

P(L=1 | G)

Approved my trip=1

Will approve my friend’s trip

P(approves friend’s trip | approved my trip) = 79%

Inference runs automatically in new model:
Existing modelling methods

- Graphical editors/factor graphs
  - Easy to use
  - Hard to develop and maintain large models
  - Hard to integrate with other code
  - Limited scope

- XML
  - Awkward syntax
  - Hard to integrate with other code
  - Limited toolset
**Csoft modelling language**

- A representation language for probabilistic models.

- Takes C# and adds support for:
  - random variables
  - constraints on variables
  - inference

- Can be embedded in ordinary C# to allow integration of deterministic + stochastic code
Csoft – random variables

- Normal variables have a fixed single value.
  e.g. `int length=6, bool visible=true`.

- Random variables have uncertain value specified by a probability distribution.
  e.g. `int length = random(Uniform(0,10))`  
    `bool visible = random(Bernoulli(0.8))`.

- Introduce random operator which means ‘is distributed as’. 
Csoft—constraints

- We can define constraints on random variables, e.g.
  \[
  \text{constrain}(\text{visible}==\text{true})
  \]
  \[
  \text{constrain}(\text{length}==4)
  \]
  \[
  \text{constrain}(\text{length}>0)
  \]
  \[
  \text{constrain}(i==j)
  \]

- The \text{constrain(b)} operator means ‘we constrain b to be true’.
**Csoft — inference**

- The **infer()** operator gives the posterior distribution of one or more random variables.

- Example:
  ```
  int i = random(Uniform(1,10));
  bool b = (i*i>50);
  Dist bdist = infer(b);//Bernoulli(0.3)
  ```

- Output of infer() is always *deterministic* even when input is *random.*
Model definition

```c
bool coffee = random(Bernoulli(0.6));
bool raining = random(Bernoulli(0.8));
bool goodMood =
    random(Bernoulli((coffee|!raining)?0.9:0.2));
bool approvesTrip = random(Bernoulli(goodMood?0.9:0.4));
```

Constraints and inference

```c
constrain(raining==true);
return infer(approvesTrip);
```
TrueSkill model (without draws)

```csharp
double[] skill = new double[nPlayers];
double[] performance = new double[nPlayers];
for (int j = 0; j < nPlayers; j++) {
    skill[j] = random(Gaussian(mu[j], sigma[j]));
    double x = random(Gaussian(0, beta));
    performance[j] = skill[j] + x;
    if (j > 0) constrain(performance[j - 1] > performance[j]);
}
return infer(skill);
```
**Csoft for analysing existing code**

```csharp
int i = random(Uniform(-100, 100));
bool b = false;
try {
    Read(i);
} catch (Exception ex) {
    b = true;
}
return infer(b);
```

Existing code called with *random* parameter

```csharp
public byte[] Read(int numBytes) {
    if (numBytes < 0) throw new ArgumentOutOfRangeException();
    ...
}
```
Random objects

- **CSoft** is **object-oriented**. Random objects are objects whose members are random variables.

- Useful for domain-specific inference: can provide a library of **random objects** relevant to a domain e.g. Image, ImageOperation, Texture.

- For example: machine vision models can be specified as a series of graphics operations which generate an image.
Implementing **Csoft**

**Csoft** is implemented as a .NET library. The operators appear as **static methods** e.g. `Csoft.random()`.

**Non-deterministic**
- **Csoft**
  - Compilation
  - Model specification
  - Execution
  - Inference engine

**Deterministic**
- **C#, VB, F#...**
  - Compilation
  - MSIL (Microsoft intermediate language)
  - Execution
  - CLR (Common Language Runtime)
Inference engine requirements

To support CSoft, engine must be:

- **Flexible**: capable of handling very broad range of model specifications
- **Accurate**: must give appropriately accurate inference results so must use an appropriate inference algorithm
- **Efficient**: must scale to run on large models with large data sets
**Infer.NET version 1**

- **Flexible**: Yes - general purpose architecture for discrete/continuous variables and a large variety of factors
- **Accurate**: Yes – supported multiple inference algorithms: VMP/EP/Gibbs
- **Efficient**: No – constructed in-memory factor graphs and traversed them during inference, introducing considerable overhead. Also, made the code very difficult to maintain as more features were added.
Infer.NET version 2

- Version 2 *compiles* modelling code into inference code.
- No in-memory factor graphs = no overhead
- Consists of a chain of code transformations:
  
  ![Diagram](image)
  
  - Each intermediate program is a valid C# program.
Infer.NET inference engine

- **Coffee**
- **Raining=1**
- **Good mood**
- **Will approve my trip**

C_Soft program

T1 --> T2 --> T3 --> Inference program
Infer.NET compiler

Channel transform -> Message transform -> T3

C_SOFT program -> Channel transform -> Message transform -> Inference program

- Coffee
- Raining
- Good mood
- Will approve my trip

Will approve my trip
Infer.NET compiler

C_{SOFT} program → Channel transform → Message transform → Scheduler → Inference program

- Coffee
- Raining
- Good mood
- Will approve my trip
- Schedule
Infer.NET Demo
Advantages of CSOFT + Infer.NET

- Rich and compact modelling language
  Wide range of complex models can be represented succinctly.

- Powerful inference framework
  Supports multiple inference algorithms, highly customisable.

- Efficient inference
  Compilation means almost no overhead.

- Easy integration
  Inference can be invoked from .NET with minimal effort.

- Easy to learn
  Just the three new operators added to the language
Thanks!

http://research.microsoft.com/infernet