

PRIMO

PRobabilistic Inference MOdules

Manuel Baum, Denis John,
Lukas Kettenbach, Maximilian Koch

Bielefeld University

13. Oktober 2013

Introduction

Idea

- probabilistic inference modules for Python
- library which offers well known probabilistic (graphical) models like Bayesian or temporal networks
- variety of inference algorithms

Download/Documentation/Installation Guide

- `github.com/mbaumBielefeld/PRIMO`
- `github.com/mbaumBielefeld/PRIMO/wiki`

Structure

PRIMO/

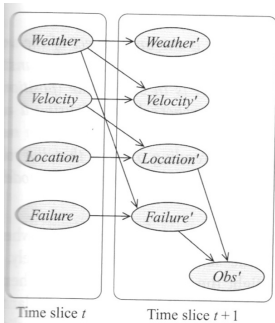
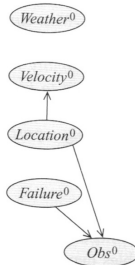
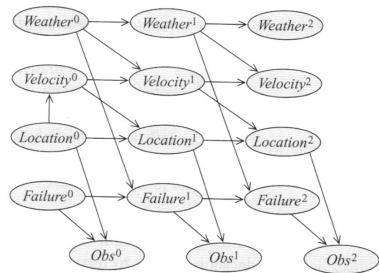
- doc/
- examples/
- primo/
 - core/ → BayesNet.py, Node.py, DynamicBayesNet.py, ...
 - decision/ → DecisionNode.py, UtilityNode.py, ...
 - reasoning/ → DiscreteNode.py, density/, MCMC.py, ...
 - tests/
 - utils/ → XMLBIF.py
- setup.py

Definition

A DBN is a pair (B_0, B_{\rightarrow}) , where B_0 is a Bayesian network over $\chi^{(0)}$ representing the initial distribution, and B_{\rightarrow} is a 2-TBN for the process. For any desired time span $T \geq 0$, the distribution over $\chi^{(0:T)}$ is defined as a unrolled Bayesian network, where, for any $i = 1, \dots, n$:

- the structure and CPDs of $X_i^{(0)}$ are the same as those for X_i in B_0 ,
- the structure and CPDs of $X_i^{(t)}$ for $t \geq 0$ are the same as those for X_i' in B_{\rightarrow} .

Example

(a) $\mathcal{B}_{\rightarrow}$ (b) \mathcal{B}_0 

(c) DBN unrolled over 3 steps

Exact Inference

- We can use standard inference algorithms (e.g. variable elimination)
- Problem I: run inference on larger and larger networks over time
- Problem II: maintain our entire history of observations indefinitely
- Solution/workaround: use approximate inference

Approximate Inference

- We can use some kind of Likelihood Weighting
- Two modifications:
 - 1 run all samples together through the DBN, one slice at a time
 - 2 focus the set of samples on the high-probability regions of the state space
- Particle Filter:
 - 1 Each sample is propagated forward by sampling the next state value x_{t+1} given the current value x_t for the sample
 - 2 Each sample is weighted by the likelihood it assigns to the new evidence $P(e_{t+1} | x_{t+1})$
 - 3 The population is *resampled* to generate a new population of N samples. Each new sample is selected from the current population; the probability that a particular sample is selected is proportional to its weight.

Algorithm

Algorithm 12.2 Likelihood-weighted particle generation

Procedure LW-Sample (

\mathcal{B} , // Bayesian network over \mathcal{X}

$Z = z$ // Event in the network

)

1 Let X_1, \dots, X_n be a topological ordering of \mathcal{X}

2 $w \leftarrow 1$

3 **for** $i = 1, \dots, n$

4 $\mathbf{u}_i \leftarrow \mathbf{x} \langle \text{Pa}_{X_i} \rangle$ // Assignment to Pa_{X_i} in x_1, \dots, x_{i-1}

5 **if** $X_i \notin Z$ **then**

6 Sample x_i from $P(X_i \mid \mathbf{u}_i)$

7 **else**

8 $x_i \leftarrow z \langle X_i \rangle$ // Assignment to X_i in z

9 $w \leftarrow w \cdot P(x_i \mid \mathbf{u}_i)$ // Multiply weight by probability of desired value

10 **return** $(x_1, \dots, x_n), w$

Algorithm

Algorithm 15.2 Likelihood-weighted particle generation for a 2-TBN

Procedure LW-2TBN (

$\mathcal{B}_{\rightarrow}$ // 2-TBN

ξ // Instantiation to time $t - 1$ variables

$\mathbf{O}^{(t)} = \mathbf{o}^{(t)}$ // time t evidence

)

1 Let X'_1, \dots, X'_n be a topological ordering of \mathcal{X}' in $\mathcal{B}_{\rightarrow}$

2 $w \leftarrow 1$

3 **for** $i = 1, \dots, n$

4 $\mathbf{u}_i \leftarrow (\xi, \mathbf{x}') \langle \text{Pa}_{X'_i} \rangle$

5 // Assignment to $\text{Pa}_{X'_i}$ in $x_1, \dots, x_n, x'_1, \dots, x'_{i-1}$

6 **if** $X'_i \notin \mathbf{O}^{(t)}$ **then**

7 Sample x'_i from $P(X'_i \mid \mathbf{u}_i)$

8 **else**

9 $x'_i \leftarrow \mathbf{o}^{(t)} \langle X'_i \rangle$ // Assignment to X'_i in $\mathbf{o}^{(t)}$

10 $w \leftarrow w \cdot P(x'_i \mid \mathbf{u}_i)$ // Multiply weight by probability of desired val

11 **return** $(x'_1, \dots, x'_n), w$

Algorithm

Algorithm 15.4 Particle filtering for DBNs

Procedure Particle-Filter-DBN (

$\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle, \quad // \text{ DBN}$

$M \quad // \text{ Number of samples}$

$\mathbf{o}^{(1)}, \mathbf{o}^{(2)}, \dots \quad // \text{ Observation sequence}$

)

1 **for** $m = 1, \dots, M$

2 Sample $\bar{\mathbf{x}}^{(0)}[m]$ from \mathcal{B}_0

3 $w^{(0)}[m] \leftarrow 1/M$

4 **for** $t = 1, 2, \dots$

5 **for** $m = 1, \dots, M$

6 Sample $\bar{\mathbf{x}}^{(0:t-1)}$ from the distribution $\hat{P}_{\mathcal{D}^{(t-1)}}$.

7 // Select sample for propagation

8 $(\bar{\mathbf{x}}^{(t)}[m], w^{(t)}[m]) \leftarrow \text{LW-2TBN}(\mathcal{B}_{\rightarrow}, \bar{\mathbf{x}}^{(t-1)}, \mathbf{o}^{(t)})$

9 // Generate time t sample and weight from selected sample
 $\bar{\mathbf{x}}^{(t-1)}$

10 $\mathcal{D}^{(t)} \leftarrow \{(\bar{\mathbf{x}}^{(0:t)}[m], w^{(t)}[m]) : m = 1, \dots, M\}$

11 $\hat{\sigma}^{(t)}(\mathbf{x}) \leftarrow \hat{P}_{\mathcal{D}^{(t)}}$

Algorithm

Algorithm 15.4 Particle filtering for DBNs

```

Procedure Particle-Filter-DBN ( ParticleFilterDBN.py
     $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$ , // DBN
     $M$  // Number of samples
     $\mathbf{o}^{(1)}, \mathbf{o}^{(2)}, \dots$  // Observation sequence
)
1  for  $m = 1, \dots, M$ 
2      Sample  $\bar{\mathbf{x}}^{(0)}[m]$  from  $\mathcal{B}_0$  sample_from_initail_distribution()
3       $w^{(0)}[m] \leftarrow 1/M$ 
4  for  $t = 1, 2, \dots$  sample_one_time_slice()
5      for  $m = 1, \dots, M$  createTimeslice()
6          Sample  $\bar{\mathbf{x}}^{(0:t-1)}$  from the distribution  $\hat{P}_{\mathcal{D}^{(t-1)}}$ .
7          // Select sample for propagation weighted_sample()
8           $(\bar{\mathbf{x}}^{(t)}[m], w^{(t)}[m]) \leftarrow \text{LW-2TBN}(\mathcal{B}_{\rightarrow}, \bar{\mathbf{x}}^{(t-1)}, \mathbf{o}^{(t)})$ 
9          // Generate time  $t$  sample and weight from selected sampl
              $\bar{\mathbf{x}}^{(t-1)}$  wighted_sample_with_replacement()
10          $\mathcal{D}^{(t)} \leftarrow \{(\bar{\mathbf{x}}^{(0:t)}[m], w^{(t)}[m]) : m = 1, \dots, M\}$ 
11          $\hat{\sigma}^{(t)}(\mathbf{x}) \leftarrow \hat{P}_{\mathcal{D}^{(t)}}$ 

```

PRIMO/primo/

- core/
 - DynamicBayesNet.py
 - TowTBN.py (create_timeslice())
- reasoning/particlebased/
 - ParticleFilterDBN.py (sample_from_initial_distribution(), wighted_sample_with_replacement(), ...)
- tests/
 - DynamicBayesNet_test.py
 - XMLBIF_test.py
- utils/
 - XMLBIF.py

- STUART RUSSELL AND PETER NORVIG
Artificial Intelligence: A Modern Approach
- DAPHNE KOLLER AND NIR FRIEDMAN
Probabilistic Graphical Models: Principles and Techniques

Task Description

- Exact inference
- Use elimination trees
- Prior Marginal, Posterior Marginal & PoE

Algorithm 10 FE2($\mathcal{N}, \mathbf{Q}, (\mathcal{T}, \phi), r$)

input:

\mathcal{N} : Bayesian network
 \mathbf{Q} : some variables in network \mathcal{N}
 (\mathcal{T}, ϕ) : elimination tree for the CPTs of network \mathcal{N}
 r : a node in tree \mathcal{T} where $\mathbf{Q} \subseteq \text{vars}(r)$

output: the prior marginal $\text{Pr}(\mathbf{Q})$

main:

```

1: while tree  $\mathcal{T}$  has more than one node do
2:   remove a node  $i \neq r$  having a single neighbor  $j$  from tree  $\mathcal{T}$ 
3:    $\mathbf{V} \leftarrow$  variables appearing in  $\phi_i$  but not in remaining tree  $\mathcal{T}$ 
4:    $\phi_j \leftarrow \phi_j \sum_{\mathbf{V}} \phi_i$ 
5: end while
6: return project( $\phi_r, \mathbf{Q}$ )

```

Literature

- Modeling and Reasoning with Bayesian Networks , Adnan Darwiche

Thank you for your attention!

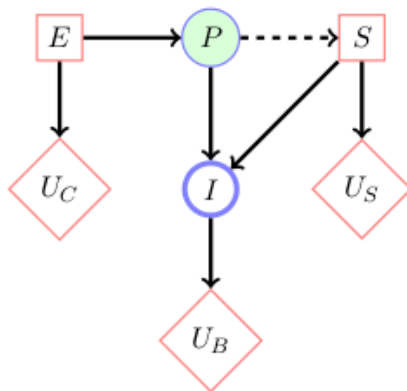
What is a BDN?

- Bayesian Networks with additional decision nodes and utility nodes
 - Decision nodes hold values for different actions
 - Utility nodes are deterministic functions of their parents
- BDN shows which information is required in order to make each decision
- and the order in which these decisions are to be made

Causal consistency

- BDN is consistent when a current decision cannot affect the past
- Descendants of a decision node must come later in the partial order
- A valid BDN has a directed path connecting all decisions

Example

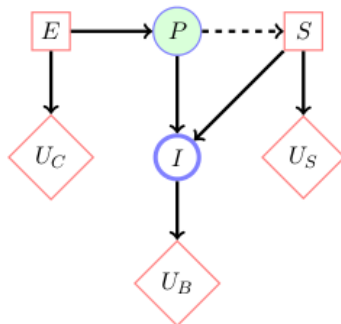


Partial ordering:

$$E^* \prec P \prec S^* \prec I$$

Solving a BDN

- Optimal EU is given by summing over unrevealed variables and optimising over future decisions



$$U(E) = \sum_P \max_s \sum_I p(I|S, P) p(P|E) [U_S(S) + U_C(E) + U_B(I)] \quad (1)$$

Algorithm

- For each possible value of a decision node:
 - Set decision node to that value
 - Calculate the posterior probability of the parent nodes of the utility node, using BN inference
 - Calculate the resulting (expected) utility for action
- Return the action with the highest utility

PRIMO/primo/

- core/
 - BayesianDecisionNetwork.py
- decision/
 - DecisionNode.py
 - UtilityNode.py
 - UtilityTable.py
- decision/make_decision/
 - Make_Decision.py

- BARBER, DAVID
Bayesian Reasoning and Machine Learning