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/

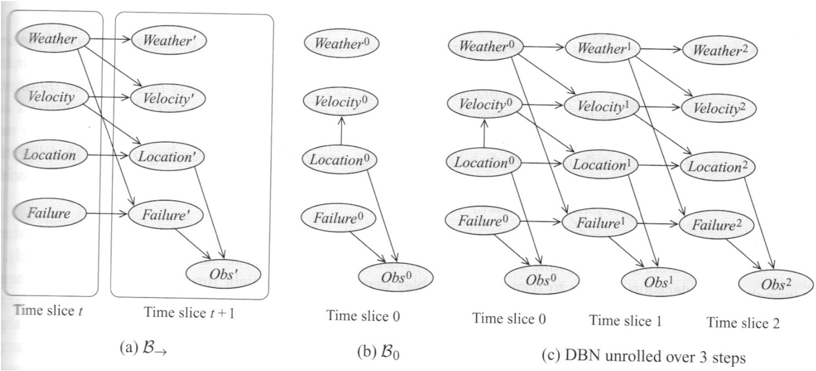
- doc/
- examples/
- primo/
  - core/ → BayesNet.py, Node.py, DynamicBayesNet.py, ...
  - decision/ → DecisionNode.py, UtilityNode.py, ...
  - reasoning/ → DiscreteNode.py, density/, MCMC.py, ...
  - tests/
  - utils/ → XMLBIF.py
- setup.py

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 $i = 1, ..., n$:

- 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


Weather
Velocity
Location
Failure
Weather'
Velocity'
Location'
Failure'
Obs'
Time slice t
Time slice t+1
(a) $\mathcal{B}_\to$
Weather^0
Velocity^0
Location^0
Failure^0
Obs^0
Time slice 0
(b) $\mathcal{B}_0$
Weather^1
Weather^2
Velocity^1
Velocity^2
Location^1
Location^2
Failure^1
Failure^2
Obs^1
Obs^2
Time slice 1
Time slice 2
(c) DBN unrolled over 3 steps

Exact Inference

- We can use standard inference algorithms (e.g. variable elimination)
- Problem I: run inference on larger an 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 $\mathcal{X}$
  $\boldsymbol{Z} = \boldsymbol{z}$ // Event in the network
)

Let $X_1, \ldots, X_n$ be a topological ordering of $\mathcal{X}$
$w \leftarrow 1$
for $i = 1, \ldots, n$
  $\boldsymbol{u}_i \leftarrow \boldsymbol{x}\langle \mathrm{Pa}_{X_i} \rangle$ // Assignment to Pa_{X_i} in $x_1, \ldots, x_{i-1}$
  if $X_i \notin \boldsymbol{Z}$ **then**
    Sample x_i from $P(X_i \mid \boldsymbol{u}_i)$
  else
    $x_i \leftarrow \boldsymbol{z}\langle X_i \rangle$ // Assignment to X_i in $\boldsymbol{z}$
    $w \leftarrow w \cdot P(x_i \mid \boldsymbol{u}_i)$ // Multiply weight by probability of desired value
return $(x_1, \ldots, x_n), w$

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

Let $X'_1, \ldots, X'_n$ be a topological ordering of $\mathcal{X}'$ in $\mathcal{B}_\rightarrow$
$w \leftarrow 1$
for $i = 1, \ldots, n$
  $\boldsymbol{u}_i \leftarrow (\xi, \boldsymbol{x}')\langle \mathrm{Pa}_{X'_i} \rangle$
    // Assignment to $\mathrm{Pa}_{X'_i}$ in $x_1, \ldots, x_n, x'_1, \ldots, x'_{i-1}$
  if $X'_i \notin \boldsymbol{O}^{(t)}$ **then**
    Sample x'_i from $P(X'_i \mid \boldsymbol{u}_i)$
  else
    $x'_i \leftarrow \boldsymbol{o}^{(t)}\langle X'_i \rangle$ // Assignment to X'_i in $\boldsymbol{o}^{(t)}$
    $w \leftarrow w \cdot P(x'_i \mid \boldsymbol{u}_i)$ // Multiply weight by probability of desired valu
return $(x'_1, \ldots, x'_n), w$

Algorithm

Algorithm 15.4 Particle filtering for DBNs

Procedure Particle-Filter-DBN (
$\langle \mathcal{B}_0, \mathcal{B}_\rightarrow \rangle$, // DBN
M // Number of samples
$\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)}, \ldots$ // Observation sequence
)

for $m = 1, \ldots, M$
  Sample $\bar{\boldsymbol{x}}^{(0)}[m]$ from $\mathcal{B}_0$
  $w^{(0)}[m] \leftarrow 1/M$
for $t = 1, 2, \ldots$
  for $m = 1, \ldots, M$
    Sample $\bar{\boldsymbol{x}}^{(0:t-1)}$ from the distribution $\hat{P}_{\mathcal{D}^{(t-1)}}$.
      // Select sample for propagation
    $(\bar{\boldsymbol{x}}^{(t)}[m], w^{(t)}[m]) \leftarrow$ LW-2TBN$(\mathcal{B}_\rightarrow, \bar{\boldsymbol{x}}^{(t-1)}, \boldsymbol{o}^{(t)})$
      // Generate time t sample and weight from selected sampl $\bar{\boldsymbol{x}}^{(t-1)}$
  $\mathcal{D}^{(t)} \leftarrow \{(\bar{\boldsymbol{x}}^{(0:t)}[m], w^{(t)}[m]) \ : m = 1, \ldots, M\}$
  $\hat{\sigma}^{(t)}(\boldsymbol{x}) \leftarrow \hat{P}_{\mathcal{D}^{(t)}}$

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

$\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)}, \ldots$ // Observation sequence

)

for $m = 1, \ldots, M$

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

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

for $t = 1, 2, \ldots$ sample_one_time_slice()

for $m = 1, \ldots, M$

createTimeslice()

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

// Select sample for propagation weighted_sample()

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

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

wighted_sample_with_replacement()

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

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

Files

PRIMO/primo/

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

Literature

- 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($\mathbf{N}$, $\mathbf{Q}$, $(\mathcal{T}, \phi)$, r)

input:

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

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

main:

while tree $\mathcal{T}$ has more than one node **do**
  remove a node $i \neq r$ having a single neighbor j from tree $\mathcal{T}$
  $\mathbf{V} \leftarrow$ variables appearing in ϕ_i but not in remaining tree $\mathcal{T}$
  $\phi_j \leftarrow \phi_j \sum_{\mathbf{V}} \phi_i$
end while
return project(ϕ_r, $\mathbf{Q}$)

Literature

- Modeling and Reasoning with Bayesian Networks , Adnan Darwiche

Thank you for your attention!

Why use approximate Inference?

- Possible to include non-linear dependencies of/on real-valued variables
- Only local computations $\rightarrow$ size of net rather unimportant
- Possible to trade off computation time for accuracy

Whats the stucture?

There are two central classes for computation:

- MCMC - Interface for usage, final computations
- MarkovChainSampler - Constructs markov chains

Other classes:

- Evidence
- ContinuousNode (and densities: Gauss, Exponential, Beta)

MarkovChainSampler generates a markov-chain given some evidence when chain has converged

Parameters

- transition_model: Gibbs or MetropolisHastings
- convergence_test: Test for convergence
- time_steps: Maximum length of chain
- evidence: Different kinds of evidence for a subset of nodes

MCMC-class This encapsules MarkovChainSampler

Possible requests:

- def calculate_PriorMarginal(self, variables, AssumedDensity):
- def calculate_MAP(self, variables, evidence, AssumedDensity):
- def calculate_PosteriorMarginal(self, variables, evidence, AssumedDensity):
- def calculate_PoE(self, evidence):

Real-valued variables

Which densities are supported?

- Gaussian
- Beta
- Exponential

Literature

DAPHNE KOLLER AND NIR FRIEDMAN

Probabilistic Graphical Models: Principles and Techniques

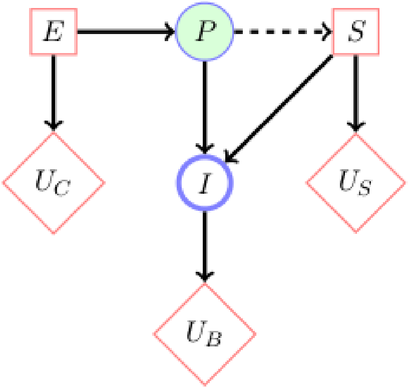
What is a BDN?

- 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 vaild BDN has a directed path connecting all decisions

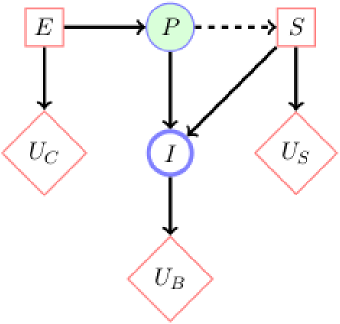
Example



Partial ordering:

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

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



$$U(E) = \sum \max_s \sum p(I|S,P)p(P|E)[U_S(S)+U_C(E)+$$

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

Files

PRIMO/primo/

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

Literature

- BARBER, DAVID
  Bayesian Reasoning and Machine Learning