UPPSALA UNIVERSITET

Probabilistic Machine Learning

Lecture 9 – Inference on graphical models, and probabilistic programming

Lawrence Murray, Uppsala University

2018-05-22

Outline – Lecture 9

Aim: To understand how to perform inference with graphical models, and how they are extended to universal probabilistic programs.

1. Summary of lecture 8
2. Inference on graphical models
   - On chains
   - On trees
   - On directed acyclic graphs
3. Probabilistic programs
   - Extending graphical models to universal probabilistic programs
   - Birch: a universal probabilistic programming language

Summary of lecture 8

We looked at three types of graphical models:

- **Directed models** encode the joint distribution of a probabilistic model as a product of conditional distributions; conditional independencies are determined using the rules of **d-separation**.
- **Undirected models** encode the joint distribution as a product of potential functions; conditional independencies are determined simply by paths on the graph.
- **Factor graphs** are a representation to which either of the above can be converted, and which can be useful for inference and learning.

Summary of lecture 8

A reminder of the rules of d-separation. Two nodes in a graph are d-separated if, on all possible paths between them:

- the **tail-to-tail** nodes are observed,
- the **head-to-tail** nodes are observed, and
- the **head-to-head** nodes are not observed and nor are any of their descendants.

Inference on Graphical Models

Inference in graphical models

Inference in graphical models amounts to computing the posterior distribution of one or more of the nodes that are not observed. We:

- **condition** on the nodes that are observed, and
- **marginalize** out the nodes that we don't care about, using the
- **structure** of the graphical model to do so efficiently.

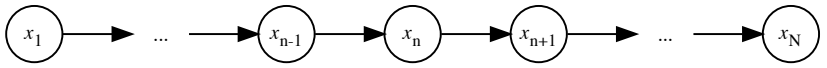
Inference in graphical models

We will consider two inference questions. Let A denote the set of unobserved nodes and B the set of observed nodes.

- What are the **posterior marginal** distributions of all the unobserved nodes given all the observed nodes? i.e. $p(x_n \mid B)$ for all $x_n \in A$.
- What is the **posterior joint** distribution of all the unobserved nodes given all the observed nodes? i.e. $p(A \mid B)$.

We will consider the first of these to begin with, and assume that no nodes are observed just yet, i.e. $B = \emptyset$.

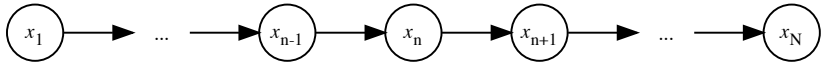
Posterior marginals in a chain



In a directed graphical model that is a **chain**, each node has at most one parent and at most one child.

$$
\begin{aligned}
p(x_n) &= \int_{\mathcal{X}_N} \cdots \int_{\mathcal{X}_{n+1}} \int_{\mathcal{X}_{n-1}} \cdots \int_{\mathcal{X}_1} p(x_{1:N}) \, \mathrm{d}x_1 \ldots \mathrm{d}x_{n-1} \, \mathrm{d}x_{n+1} \ldots \mathrm{d}x_N \\
&= \left(\int_{\mathcal{X}_{n-1}} p(x_n \mid x_{n-1}) \cdots \int_{\mathcal{X}_1} p(x_2 \mid x_1) p(x_1) \, \mathrm{d}x_1 \ldots \mathrm{d}x_{n-1} \right) \times \\
&\qquad \left(\int_{\mathcal{X}_{n+1}} p(x_{n+1} \mid x_n) \cdots \int_{\mathcal{X}_N} p(x_N \mid x_{N-1}) \, \mathrm{d}x_N \ldots \mathrm{d}x_{n+1} \right) \\
&= \underbrace{\mu_\alpha(x_n)}_{\text{forward message}} \times \underbrace{\mu_\beta(x_n)}_{\text{backward message}} .
\end{aligned}
$$

Posterior marginals in a chain

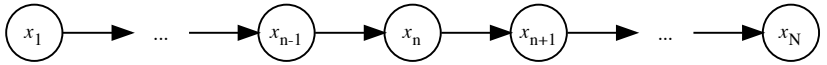


Note the recursive structure of these messages:

$$\mu_\alpha(x_n) = \int_{\mathcal{X}_{n-1}} p(x_n \mid x_{n-1}) \mu_\alpha(x_{n-1}) \, \mathrm{d}x_{n-1}$$
$$\mu_\beta(x_n) = \int_{\mathcal{X}_{n+1}} p(x_{n+1} \mid x_n) \mu_\beta(x_{n+1}) \, \mathrm{d}x_{n+1}.$$

So the structure of the chain suggests an efficient inference algorithm to compute **all** marginals.

Posterior marginals in a chain



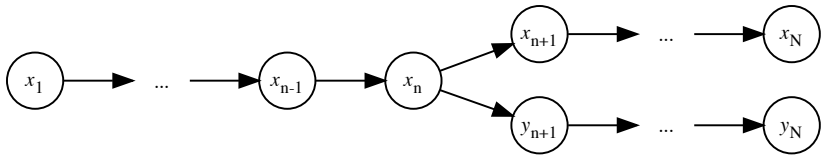
Inference proceeds as follows:

1. **Forward pass:** Begin at x_1, recursively compute the forward message for each of the nodes x_1 to x_N in turn.
2. **Backward pass:** Begin at x_N, recursively compute the backward message for each of the nodes x_N to x_1 in turn.

We then have, for each x_n, the distribution $p(x_n)$.

We have not introduced observations yet, and so all of the backward messages are equal to 1 at this stage, so that we have $p(x_n)$ after the forward pass only. But we do need the backward pass usually!

Posterior marginals in a tree



In a directed graphical model that is a **tree**, each node has at most one parent, but any number of children. Chains are also trees.

The derivations are much the same as for a chain, but for each node we end up with one forward message and **multiple backward messages**—one for each child of the node.

Posterior marginals in a tree

Inference proceeds by visiting each of the nodes in **depth-first order**, beginning with the root node:

1. Compute the forward message.
2. Visit the first child, and receive its backward message. Update own forward message by multiplying it with this backward message.
3. Visit each remaining child in turn, and receive its backward message. Each time, update own forward message by multiplying it with the backward message.
4. Compute own backward message, which is actually just the updated forward message at this point, and send it to the parent.

What do the messages actually look like?

Think of each node in the graph as having a set of hyperparameters that describe its distribution:

- For discrete-valued variables, these hyperparameters are the probabilities of it taking on each possible value.
- For continuous-valued variables, these are the hyperparameters of some parametric distribution, such as the mean and variance of a Gaussian distribution, or shape and scale of a Gamma distribution.

During inference, the forward and backward messages update these hyperparameters from those that describe the **prior** distribution to those that (eventually) describe the **posterior** distribution.

Handling observations with discrete variables

If some discrete-valued node x_n is observed to have a value $\widehat{x}_n$:

- Multiply the joint distribution by an indicator function $I(x_n, \widehat{x}_n)$ that takes the value 1 when $x_n = \widehat{x}_n$ and 0 otherwise.
- The integrals are just sums over all possible values of a variable.
- The sum over x_n will contain only one term for which $x_n = \widehat{x}_n$.

Handling observations with continuous variables

If some continuous-valued node x_n is observed to have a value $\widehat{x}_n$:

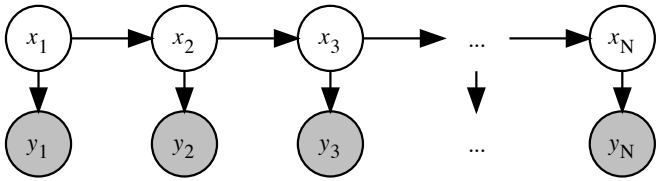
- Multiply the joint distribution by the Dirac measure $\delta_{\widehat{x}_n}(x_n)$.
- The integral over x_n then reduces to just one term for which $x_n = \widehat{x}_n$.

The solution of the integrals is analytically tractable only for particular choices of distributions that form analytically-tractable relationships, such as *conjugate prior* relationships. A common case is where all nodes form linear-Gaussian relationships.

Posterior marginals in a tree

- The algorithm is often called **belief propagation** or the **sum-product** algorithm.
- This is just its expression on directed graphical models, Bishop (2006) explains its expression on factor graphs.
- The algorithm that was given for chains is just a special case of it (recall that a chain is also a tree).

A particular tree: the state-space model



- The state-space model is also a tree.
- If all nodes have linear-Gaussian relationships, the inference algorithm that has been described is precisely the **Kalman smoother**.
- If all nodes have discrete values, the inference algorithm that has been described is the **forward-backward algorithm** for Hidden Markov Models (the more common name for the model in this case).

Simplifying the task when there are observations

Before proceeding with inference, we can simplify the graph according to the node(s) of interest:

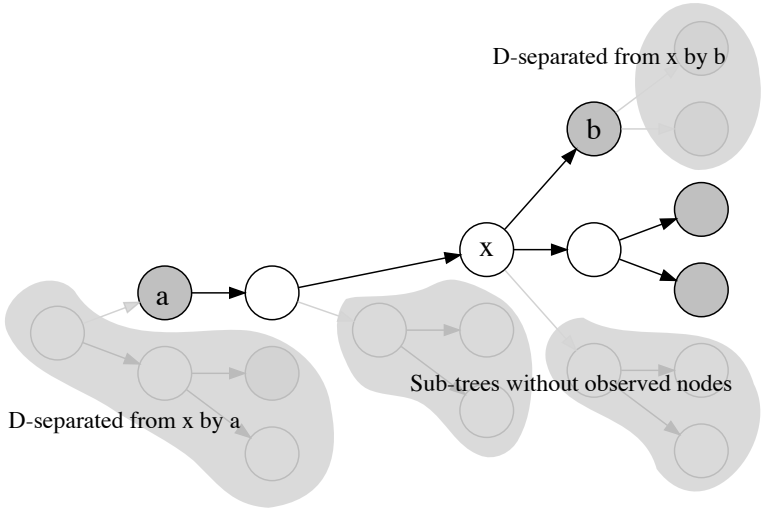
- Subgraphs that are d-separated from the node(s) of interest can be ignored.
- Subtrees that contain neither the node(s) of interest nor observed nodes can be ignored.

D-separation is easy for chains and trees:

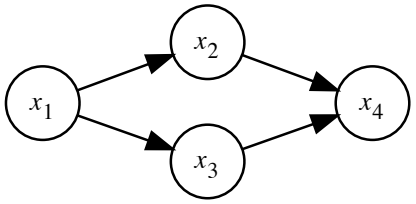
- In a **chain** all paths have only **head-to-tail** nodes.
- In a **tree** all paths have only **head-to-tail** and **tail-to-tail** nodes.

Simplifying the task when there are observations

Say we are interested in the posterior marginal of x.



Posterior marginals in a directed acyclic graph



Using the depth-first rule we get stuck: x_4 needs forward messages from both x_2 and x_3, but this is not possible.

The solution to this is an iteration scheme called **loopy belief propagation**, which is usually described as operating on **factor graphs**. But there are other ways to handle it as well, such as Monte Carlo methods.

Posterior joint inference

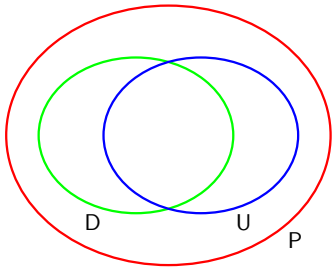
We can obtain a **sample** from the posterior joint distribution with a similar algorithm to that already described:

1. After computing the posterior marginal of some node x_n, sample from that posterior marginal to obtain some $\widehat{x}_n$.
2. Clamp the value of the node x_n to the value $\widehat{x}_n$ and adjust the backward message accordingly.
3. Repeat.

Once a node is sampled, it acts just like any other observed node. It is sometimes referred to as a **pseudo-observation** to distinguish it from a true observation.

Probabilistic Programming

Probabilistic programming



- The space of all probabilistic models is *P*.
- The directed (*D*) and undirected (*U*) classes intersect.
- Probabilistic programs attempt to fill the whole space *P*.

Universal probabilistic programs

Universal probabilistic programs are written in a Turing complete programming language. They are the most expressive class, and the most difficult for which to develop inference methods.

Universal probabilistic programs relate to graphical models in the following way:

- They may have an infinite number of nodes.
- The edges between those nodes (i.e. the structure of the model) may not be known *a priori*.
- Whether or not an edge exists may depend on the value of one or more nodes.

Probabilistic programming languages

A **probabilistic program** encodes a probabilistic model according to the semantics of a particular **probabilistic programming language (PPL)**.

A PPL may provide:

- A library of probability distributions with the ability to simulate, evaluate, and condition them.
- Specialized language features for writing probabilistic models, e.g. operators for random variables and distributions.
- Specialized language features for writing probabilistic inference methods, e.g. continuation passing style or coroutine support.

There are many PPLs

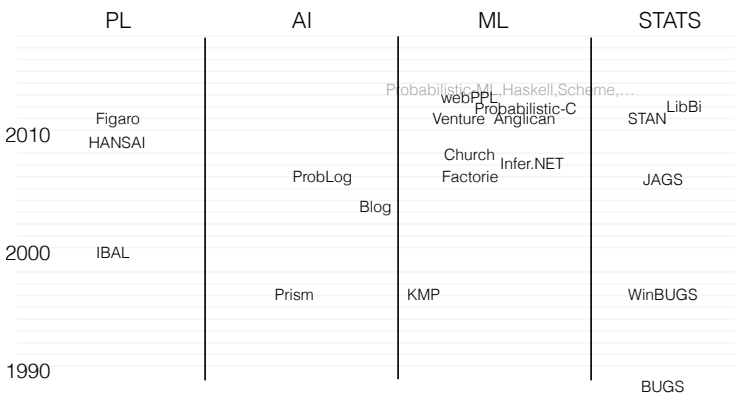


Figure courtesy of Frank Wood, Oxford University.

There are many PPLs

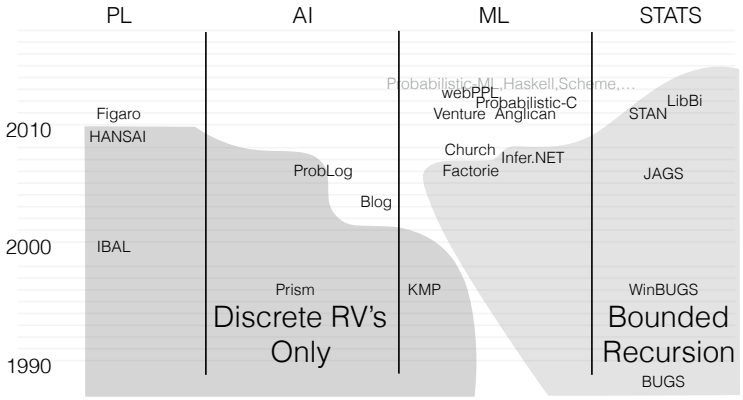


Figure courtesy of Frank Wood, Oxford University.

Birch

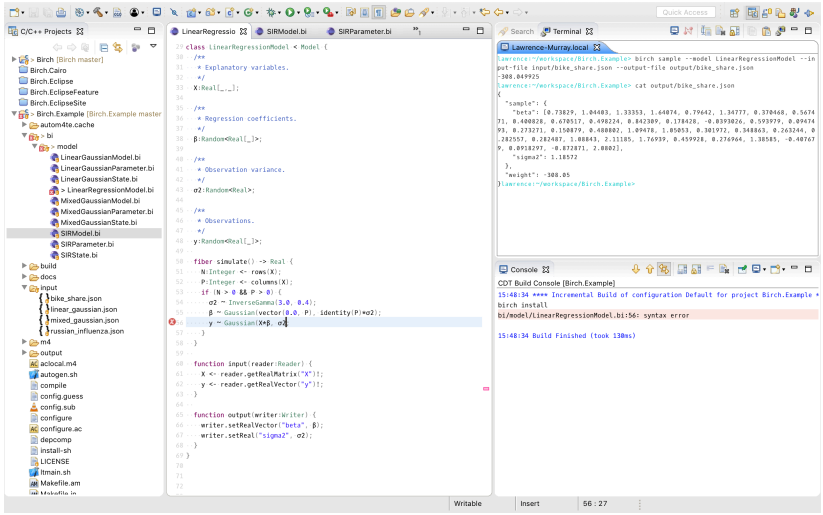
- Universal probabilistic programming language (PPL).
- Supports procedural, generic, object-oriented, and (of course) probabilistic programming paradigms.
- Both models and methods are written in the Birch language itself.
- Compiles to C++14.
- Free and open source, under the Apache 2.0 license.
- See `birch-lang.org`

Methods in Birch

Birch includes an algorithm called *delayed sampling*. This takes the sampling algorithm that was introduced for inference of the posterior joint distribution for graphical models, and extends it to universal probabilistic programs.

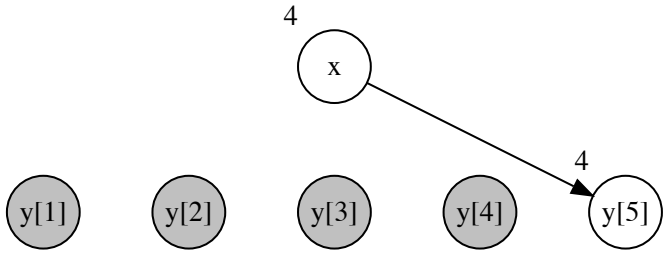
- Other inference algorithms include:
  - analytical solutions,
  - importance sampling,
  - bootstrap, alive, auxiliary and Rao–Blackwellized particle filters.
- Not far off are:
  - particle MCMC methods,
  - other MCMC methods.

Birch



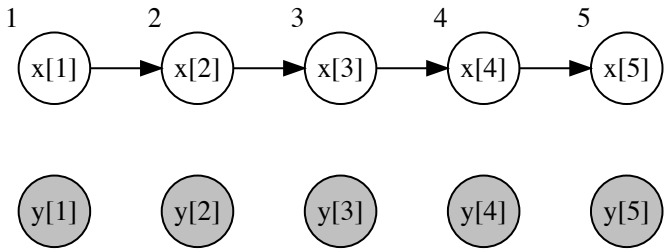
Example #1

Code	Checkpoint
`x ~ Gaussian(0.0, 1.0);`	assume x
`for (n in 1..N) {`	
`  y[n] ~ Gaussian(x, 1.0);`	observe y[n]
`}`	
`stdout.print(x);`	value x



Example #2

Code	Checkpoint
`x[1] ~ Gaussian(0.0, 1.0);`	assume x[1]
`y[1] ~ Gaussian(x[1], 1.0);`	observe y[1]
`for (t in 2..T) {`	
`  x[t] ~ Gaussian(a*x[t - 1], 1.0);`	assume x[t]
`  y[t] ~ Gaussian(x[t], 1.0);`	observe y[t]
`}`	
`stdout.print(x[1]);`	value x[1]



A few concepts to summarize lecture 9

Inference on chains and trees can be performed with forward and backward passes, often called **belief propagation** or the **sum-product** algorithm.

Inference on directed acyclic graphs requires an iterative algorithm such as **loopy belief propagation**, or can be done with **Monte Carlo** or **variational** algorithms.

Universal probabilistic programs extend graphical models into Turing complete programming languages, and are a current research concern.

Have a look at Birch to try some of these inference algorithms (`birch-lang.org`).