CSC 412/2506: Probabilistic Learning and Reasoning

Week 4 - 1/2: Message Passing

Michal Malyska

University of Toronto

Overview

- Trueskill latent variable model
- Message passing

Latent variables

- What to do when a variable z is unobserved?
- If we never condition on z when in the inference problem, then we can just integrate it out.
- However, in certain cases, we are interested in the latent variables themselves, e.g. the clustering problems.
- More on latent variables when we cover Gaussian mixtures.

The TrueSkill latent variable model

- TrueSkill model is a player ranking system for competitive games.
- The goal is to infer the skill of a set of players in a competitive game, based on observing who beats who.
- In the TrueSkill model, each player has a fixed level of skill, denoted z_i.
- We initially don't know anything about anyone's skill, but we assume everyone's skill is independent (e.g. an independent Gaussian prior).
- We never get to observe the players' skills directly, which makes this a latent variable model.

TrueSkill model

- Instead, we observe the outcome of a series of matches between different players.
- For each game, the probability that player i beats player j is given by

$$p(i \text{ beats } j) = \sigma(z_i - z_j)$$

where sigma is the logistic function: $\sigma(y) = \frac{1}{1+\exp(-y)}$.
- We can write the entire joint likelihood of a set of players and games as:

$$p(z_1, z_2, \ldots z_N, \text{game 1, game 2, .. game T})$$
$$= \left[\prod_{i=1}^{N} p(z_i)\right] \left[\prod_{\text{games}} p(\text{i beats j}|z_i, z_j)\right]$$

Posterior

- Given the outcome of some matches, the players' skills are no longer independent, even if they've never played each other.
- Computing the posterior over even two players' skills requires integrating over all the other players' skills:

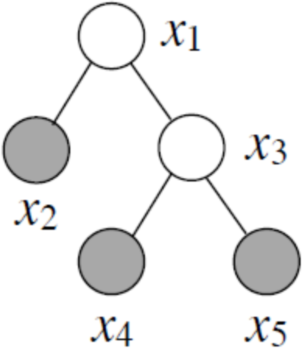
$$p(z_1, z_2|\text{game 1, game 2, ... game T}) \\ = \int \cdots \int p(z_1, z_2, z_3 \dots z_N|x) dz_3 \dots dz_N$$

- Message passing can be used to compute posteriors!
- More on this model in Assignment 2.

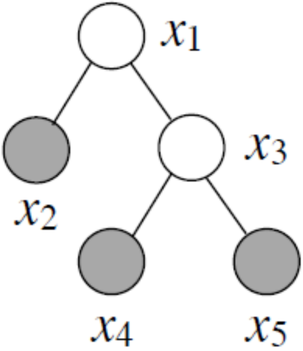
Variable Elimination Order and Trees

- **Last week**: we can do exact inference by variable elimination: I.e. to compute $p(A|C)$, we can marginalize $p(A, B|C)$ over every variable in B, one at a time.
- Computational cost is determined by the graph structure, and the elimination ordering.
- Determining the optimal elimination ordering is hard.
- Even if we do, the resulting marginalization might also be unreasonably costly.
- Fortunately, for trees, any elimination ordering that goes from the leaves inwards towards any root will be optimal.
- You can think of trees as just chains which sometimes branch.

Inference in Trees



Inference in Trees

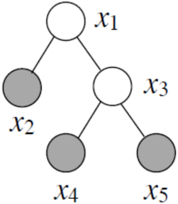


- A graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices (nodes) and $\mathcal{E}$ the set of edges
- For $i, j \in \mathcal{V}$, we have $(i, j) \in \mathcal{E}$ if there is an edge between the nodes i and j.
- For a node in graph $i \in \mathcal{V}$, $N(i)$ denotes the neighbors of i, i.e. $N(i) = \{j : (i, j) \in \mathcal{E}\}$.
- Shaded nodes are observed, and denoted by $\bar{x}_2, \bar{x}_4, \bar{x}_5$.

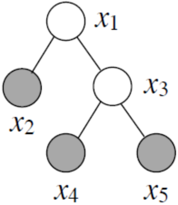
The joint distribution in the general case is

$$p(x_{1:n}) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \psi(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j).$$

Inference in Trees



Inference in Trees



- Joint distribution is

$$p(x_{1:n}) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \psi(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j).$$

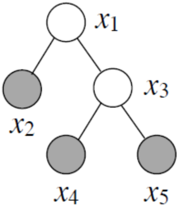
- Want to compute $p(x_3|\bar{x}_2, \bar{x}_4, \bar{x}_5)$.
- We have

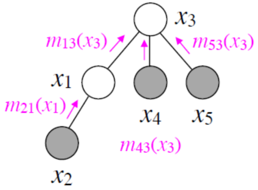
$$p(x_3|\bar{x}_2, \bar{x}_4, \bar{x}_5) \propto p(x_3, \bar{x}_2, \bar{x}_4, \bar{x}_5).$$

$$p(x_3 \mid \bar{x}_2, \bar{x}_4, \bar{x}_5) = \frac{1}{Z^E} \sum_{x_1} \psi_1(x_1)\psi_3(x_3)\psi_2(\bar{x}_2)\psi_4(\bar{x}_4)\psi_5(\bar{x}_5)\psi_{12}(\bar{x}_2, x_1)\psi_{34}(\bar{x}_4, x_3)\psi_{35}(\bar{x}_5, x_3)\psi_{13}(x_1, x_3)$$

- Let's write the variable elimination.

Inference in Trees





$$
\begin{aligned}
p(x_3 \mid \bar{x}_2, \bar{x}_4, \bar{x}_5) &= \frac{1}{Z^E} \sum_{x_1} \psi_1(x_1)\psi_3(x_3)\psi_2(\bar{x}_2)\psi_4(\bar{x}_4)\psi_5(\bar{x}_5)\psi_{12}(\bar{x}_2, x_1)\psi_{34}(\bar{x}_4, x_3)\psi_{35}(\bar{x}_5, x_3)\psi_{13}(x_1, x_3) \\
&= \frac{1}{Z^E} \underbrace{\psi_4(\bar{x}_4)\psi_{34}(\bar{x}_4, x_3)}_{m_{43}(x_3)} \underbrace{\psi_5(\bar{x}_5)\psi_{35}(\bar{x}_5, x_3)}_{m_{53}(x_3)} \psi_3(x_3) \sum_{x_1} \psi_1(x_1)\psi_{13}(x_1, x_3) \underbrace{\psi_2(\bar{x}_2)\psi_{12}(\bar{x}_2, x_1)}_{m_{21}(x_1)} \\
&= \frac{1}{Z^E} \psi_3(x_3) m_{43}(x_3) m_{53}(x_3) \underbrace{\sum_{x_1} \psi_1(x_1)\psi_{13}(x_1, x_3) m_{21}(x_1)}_{m_{13}(x_3)} \\
&= \frac{1}{Z^E} \psi_3(x_3) m_{43}(x_3) m_{53}(x_3) m_{13}(x_3) = \frac{\psi_3(x_3) m_{43}(x_3) m_{53}(x_3) m_{13}(x_3)}{\sum_{x_3} \psi_3(x_3) m_{43}(x_3) m_{53}(x_3) m_{13}(x_3)}
\end{aligned}
$$

Slide credit: S. Ermon

Message Passing on Trees

We perform variable elimination from leaves to root, which is the sum product algorithm to compute all marginals. Belief propagation is a message-passing between neighboring vertices of the graph.

- The message sent from variable j to $i \in N(j)$ is

$$m_{j \to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k \in N(j)/i} m_{k \to j}(x_j)$$

  - If x_j is observed, the message is

$$m_{j \to i}(x_i) = \psi_j(\bar{x}_j)\psi_{ij}(x_i, \bar{x}_j) \prod_{k \in N(j)/i} m_{k \to j}(\bar{x}_j)$$

- Once the message passing stage is complete, we can compute our beliefs as

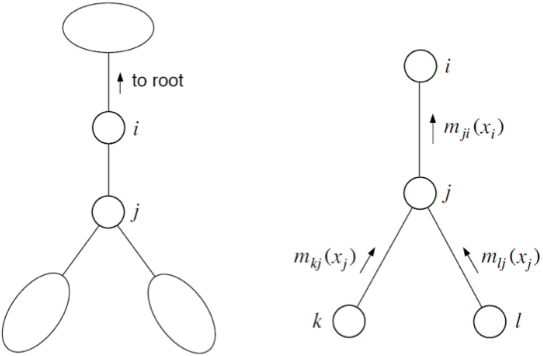
$$b(x_i) \propto \psi_i(x_i) \prod_{j \in N(i)} m_{j \to i}(x_i).$$

- Once normalized, beliefs are the marginals we want to compute!

Message Passing on Trees

The message sent from variable j to $i \in N(j)$ is

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)/i} m_{k\to j}(x_j)$$

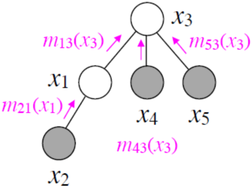


Each message $m_{j\to i}(x_i)$ is a vector with one value for each state of x_i.

Inference in Trees: Compute $p(x_1|\bar{x}_2, \bar{x}_4, \bar{x}_5)$

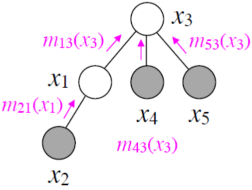
$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)/i} m_{k\to j}(x_j)$$

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i).$$



$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)/i} m_{k\to j}(x_j)$$

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i).$$



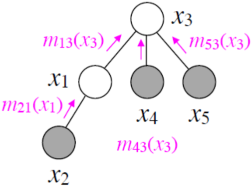
- $m_{5\to 3}(x_3) = \psi_5(\bar{x}_5)\psi_{35}(x_3, \bar{x}_5)$
- $m_{2\to 1}(x_1) = \psi_2(\bar{x}_2)\psi_{12}(x_1, \bar{x}_2)$
- $m_{4\to 3}(x_3) = \psi_4(\bar{x}_4)\psi_{34}(x_3, \bar{x}_4)$
- $m_{1\to 3}(x_3) = \sum_{x_1} \psi_1(x_1)\psi_{13}(x_1, x_3)m_{2\to 1}(x_1)$
- $b(x_3) \propto \psi_3(x_3)m_{1\to 3}(x_3)m_{4\to 3}(x_3)m_{5\to 3}(x_3)$

This is the same as variable elimination, so

$$p(x_3|\bar{x}_2, \bar{x}_4, \bar{x}_5) = b(x_3)$$

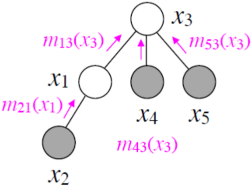
Belief Propagation on Trees

Belief Propagation Algorithm on Trees



Belief Propagation on Trees

Belief Propagation Algorithm on Trees



- Choose root r arbitrarily
- Pass messages from leafs to r
- Pass messages from r to leafs
- These two passes are sufficient on trees!
- Compute beliefs (marginals)

$$b(x_i) \propto \psi_i(x_i) \prod_{j \in \mathcal{N}(i)} m_{j \to i}(x_i), \ \forall_i$$

One can compute them in two steps:

- Compute unnormalized beliefs $\tilde{b}(x_i) =\propto= \psi_i(x_i) \prod_{j \in \mathcal{N}(i)} m_{j \to i}(x_i)$
- Normalize them $b(x_i) = \tilde{b}(x_i) / \sum_{x_i} \tilde{b}(x_i)$.

Loopy Belief Propagation

- What if the graph (MRF) we have is not a tree and have cycles?
- Keep passing messages until convergence.
- This is called **Loopy Belief Propagation**.
- This is like when someone starts a rumour and then hears the same rumour from someone else, making them more certain it's true.
- We won't get the exact marginals, but an approximation.
- But turns out it is still very useful!

Loopy Belief Propagation

Loopy BP:

- Initialize all messages uniformly:

$$m_{i\to j}(x_j) = [1/k, ..., 1/k]^\top$$

  where k is the number of states x_j can take.

- Keep running BP updates until it "converges":

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

  and normalize for stability.

- It will generally not converge, but that's generally ok.
- Compute beliefs

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in\mathcal{N}(i)} m_{j\to i}(x_i).$$

This algorithm is still very useful in practice, without any theoretical guarantee (other than trees).

Sum-product vs. Max-product

- The algorithm we learned is called **sum-product BP** and approximately computes the **marginals** at each node.
- For MAP inference, we maximize over x_j instead of summing over them. This is called **max-product BP**.
- BP updates take the form

$$m_{j\to i}(x_i) = \max_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

- After BP algorithm converges, the beliefs are **max-marginals**

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in\mathcal{N}(i)} m_{j\to i}(x_i).$$

- MAP inference:

$$\hat{x}_i = \arg\max_{x_i} b(x_i).$$

Summary

- This algorithm is still very useful in practice, without any theoretical guarantee (other than trees).
- Loopy BP multiplies the same potentials multiple times. It is often over-confident.
- BP can oscillate, but may be still useful.
- It often works better if we normalize messages, and use momentum.
- The algorithm we learned is called **sum-product BP**. If we are interested in MAP inference, we can maximize over x_j instead of summing over them. This is called **max-product BP**.