CS761 Spring 2013 Advanced Machine Learning

Graphical Models

Lecturer: Xiaojin Zhu *jerryzhu@cs.wisc.edu*

1 Directed, Undirected, Factor Graphs

In directed graphs, a directed cycle is a sequence $s_1 \to s_2 \to \ldots s_k \to s_1$. A directed acyclic graph (DAG) is a graph in which every edge is directed and without directed cycle. A *directed graphical model* (aka Bayesian Network) on a DAG is a *family* of probability distributions that *factorize* as follows:

$$p(x_1, \ldots, x_m) = \prod_{s \in V} p(x_s \mid x_{\pi(s)}), \tag{1}$$

where V is the vertex set of the DAG, and $\pi(s)$ is the parent set of s in the DAG. The conditionals $p(x_s \mid x_{\pi(s)})$ are also known as the conditional probability distributions (CPDs) or conditional probability tables (CPTs, for discrete random variables).

In undirected graphs, a clique C is a fully connected subset of V: $s, t \in C \Rightarrow (s, t) \in E$ where E is the edge set. Let ψ_C be a nonnegative potential function defined over C. An *undirected graphical model* (aka Markov Random Field) on the graph is a *family* of probability distributions that *factorize* as follows:

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \prod_{C} \psi_C(x_C), \tag{2}$$

where Z is the normalization factor and x_C is the set of nodes involved in C. A clique C is *maximal* if it is not contained in a larger clique.

Both directed and undirected graphical models can be unified as *factor graphs*. A factor denotes a multi-parents-single-child neighborhood in the former case, or a clique in the latter case. A vertex x connects to all the factor nodes it participates in.

One subtlety (and great success) of graphical models is the relationship between conditional independence and graph structure. For undirected graphical models, the family (2) has the following properties: $\forall$ disjoint $A, B, C \subset V$, x_A is conditionally independent of x_B given x_C if every path from A to B goes through C. For directed graphical models, a similar d-separation property holds.

So far, a graphical model is a *family* of distributions. However, we will also use the term to refer to a particular distribution when the CPDs or the potential functions are specified.

The study of graphical models is focused on solving the *inference* problems. Given a distribution p defined by a graphical model, inference problems include

- Computing the marginal distribution $p(x_A)$ over $A \subset V$
- Computing the conditional distribution $p(x_A \mid x_B)$, often $A \cup B \subset V$
- Computing the mode $\arg\max_x p(x)$.

2 Case Study: Latent Dirichlet Allocation

★ *Demo: lda.ppt*

Latent Dirichlet Allocation (LDA) assumes that each document is a mixture of multiple topics, and each document can have different topics weights. LDA is a full generative model and readily generalizes to unseen documents. The LDA generative process is the following.

1. Sample K multinomial distributions (each of size V) $\phi_{1:K}$ from a Dirichlet distribution $Dir(\beta)$, these are the topics. Note β is a parameter vector of length V.
2. For each document
   (a) Sample a topic multinomial (of size K) θ from a Dirichlet distribution $Dir(\alpha)$.
   (b) For each word position
      i. Sample topic index $z \sim \theta$
      ii. Sample a word from the topic $w \sim \phi_z$

The observation is the document collection $w_{1:n}$. The parameters are α and β. The other parameters z, ϕ and θ are hidden variables that will be marginalized out.

The probability of a topic multinomial ϕ drawn from $Dir(\beta)$ is

$$p(\phi|\beta) = \frac{\Gamma(\sum_{i=1}^{V} \beta_i)}{\prod_{i=1}^{V} \Gamma(\beta_i)} \prod_{i=1}^{V} \phi_i^{\beta_i - 1}. \tag{3}$$

The probability of drawing the K topic multinomials are

$$p(\phi_{1:K}|\beta) = \prod_{j=1}^{K} p(\phi_j|\beta). \tag{4}$$

Similarly,

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1}. \tag{5}$$

Let $p(z|\theta)$ and $p(w|z,\phi) = p(w|\phi_z)$ be the corresponding multinomial probabilities. The probability of generating the words $w_{1:N}$ for a single document, given θ, ϕ is

$$\prod_{n=1}^{N} \sum_{z_n=1}^{K} p(z_n|\theta) p(w_n|z_n,\phi), \tag{6}$$

where we marginalize out z for each word position. Putting things together, the probability of a single document, after marginalizing out all hidden variables, is

$$p(w|\alpha,\beta) = \int_{\phi_{1:K}} \int_{\theta} p(\phi_{1:K}|\beta) p(\theta|\alpha) \left(\prod_{n=1}^{N} \sum_{z_n=1}^{K} p(z_n|\theta) p(w_n|z_n,\phi) \right) d\theta d\phi_{1:K}. \tag{7}$$

Finally, the probability of a document collection $w^{(1)}, \ldots, w^{(M)}$ is

$$p(w^{(1)}, \ldots, w^{(M)}|\alpha,\beta) = \prod_{d=1}^{M} p(w^d|\alpha,\beta). \tag{8}$$

The typical inference problem for LDA is the following: Fixing the Dirichlet parameters α, β, given a document collection, compute the conditional distribution $p(\phi, \theta \mid \alpha, \beta, w)$, or find $\arg\max_{\phi,\theta,z} p(\phi, \theta, z \mid \alpha, \beta, w)$.

3 Belief Propagation: Sum-Product and Max-Sum Algorithms

3.1 The Sum-Product Algorithm

The sum-product algorithm is also known as belief propagation. It can compute the marginals of all nodes efficiently and exactly, if the factor graph is a tree (i.e., if there is only one path between any two nodes). The algorithm involves passing *messages* on the factor graph. A message is a vector of length K, where K is the number of possible states a node can take. It is an un-normalized 'belief'.

There are two types of messages:

1. A message from a factor node f to a variable node x, denoted as $\mu_{f\to x}$. Note it is a vector of length K, and we write the x-th element (a slight abuse of notation, $x = 1 \dots K$) as $\mu_{f\to x}(x)$.
2. A message from a variable node x to a factor node f, denoted as $\mu_{x\to f}$. It is also a vector of length K, with elements $\mu_{x\to f}(x)$.

The messages are defined recursively. In particular, consider a factor f_s that involves (connects to) a particular variable x. Denote the other variables involved in f_s by $x_{1:M}$. We have

$$\mu_{f_s\to x}(x) = \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m=1}^{M} \mu_{x_m\to f_s}(x_m), \tag{9}$$

and

$$\mu_{x_m\to f_s}(x_m) = \prod_{f\in ne(x_m)\backslash f_s} \mu_{f\to x_m}(x_m), \tag{10}$$

where $ne(x_m)\backslash f_s$ is the set of factors connected to x_m, excluding f_s.

The recursion is initialized as follows. Since we assumed the factor graph is a tree, we can pick an arbitrary node and call it the root. This defines all the leaf nodes, which we start all the messages. If a leaf is a variable node x, its message to a factor node f is

$$\mu_{x\to f}(x) = 1. \tag{11}$$

If a leaf is a factor node f, its message to a variable node x is

$$\mu_{f\to x}(x) = f(x). \tag{12}$$

A node (factor or variable) can send out a message if all the necessary incoming messages have arrived. This will eventually happen for tree structured factor graph.

Once all messages have been sent, one can compute the desired marginal probabilities as

$$p(x) \propto \prod_{f\in ne(x)} \mu_{f\to x}(x). \tag{13}$$

One can also compute the marginal of the *set of* variables x_s involved in a factor f_s

$$p(x_s) \propto f_s(x_s) \prod_{x\in ne(f)} \mu_{x\to f}(x). \tag{14}$$

If a variable x is observed $x = v$, it is a constant in all neighboring factors. Its message $\mu_{x\to f}(x)$ is set to zero for all $x \neq v$. Alternatively, we can eliminate observed nodes by absorbing them (with their observed constant values) into the corresponding factors. Let X_o be the set of observed variables. With this modification, we get the *joint* probability (NB. not the conditional $p(x|X_o)$) of a single node x and all the observed nodes when we multiply the incoming messages to x:

$$p(x, X_o) \propto \prod_{f\in ne(x)} \mu_{f\to x}(x). \tag{15}$$

The conditional is easily obtained by normalization afterward

$$p(x|X_o) = \frac{p(x, X_o)}{\sum_{x'} p(x', X_o)}. \tag{16}$$

When the factor graph contains loops (not a tree), there is no longer guarantee that the algorithm will even converge. However, people find in practice that it still works quite well. This way of applying the sum-product algorithm is known as loopy belief propagation (loopy BP).

3.2 The Max-Sum Algorithm

Sometimes it is important to know the 'best states' $z_{1:N}$ corresponding to the observation $x_{1:N}$. There are at least two senses of 'best':

1. With the sum-product algorithm we can compute the marginal $p(z_n|x_{1:N})$ for each node. We can define 'best' to be the state with the highest marginal probability

$$z_n^* = \arg\max_k p(z_n = k|x_{1:N}), \tag{17}$$

   and we will have a set of most likely states $z_{1:N}^*$. Each time step is the best individual, however $z_{1:N}^*$ as a whole may not be the most likely *state configuration*. In fact it can even be an invalid configuration with zero probability, depending on the model!

2. The alternative is to find

$$z_{1:N}^* = \arg\max_{z_{1:N}} p(z_{1:N}|x_{1:N}). \tag{18}$$

   It finds the most likely *state configuration* as a whole. The max-sum algorithm addresses this problem efficiently.

We first modify the sum-product algorithm to obtain the *max-product* algorithm. The idea is very simple: replace $\sum$ with max in the messages. In fact only factor-to-variable messages are affected:

$$\mu_{f_s \to x}(x) = \max_{x_1} \dots \max_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m=1}^{M} \mu_{x_m \to f_s}(x_m) \tag{19}$$

$$\mu_{x_m \to f_s}(x_m) = \prod_{f \in ne(x_m) \setminus f_s} \mu_{f \to x_m}(x_m) \tag{20}$$

$$\mu_{x_{\text{leaf}} \to f}(x) = 1 \tag{21}$$

$$\mu_{f_{\text{leaf}} \to x}(x) = f(x). \tag{22}$$

As before, we specify an arbitrary variable node x as the root, and pass messages from leaves until they reach the root. At the root, we multiply all incoming messages to obtain the maximum probability

$$p^{\max} = \max_x \left(\prod_{f \in ne(x)} \mu_{f \to x}(x) \right). \tag{23}$$

This is the probability of the most likely state configuration. But we have not specified how to identify the configuration itself. Note unlike the sum-product algorithm, we do not pass messages back from root to leaves. Instead, we keep *back pointers* whenever we perform the max operation. In particular, when we create the message

$$\mu_{f_s \to x}(x) = \max_{x_1} \dots \max_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m=1}^{M} \mu_{x_m \to f_s}(x_m), \tag{24}$$

for each x value, we separately create M pointers back to the values of $x_1, \ldots, x_M$ that achieve the maximum. When at the root, we back trace the pointers from the value x that achieve $p^{\max}$. This eventually gives us the complete most likely state configuration.

The *max-sum algorithm* is equivalent to the max-product algorithm, but work in log space, to avoid potential underflow problem. In particular, the messages are

$$\mu_{f_s \to x}(x) = \max_{x_1} \ldots \max_{x_M} \log f_s(x, x_1, \ldots, x_M) + \sum_{m=1}^{M} \mu_{x_m \to f_s}(x_m) \tag{25}$$

$$\mu_{x_m \to f_s}(x_m) = \sum_{f \in ne(x_m) \setminus f_s} \mu_{f \to x_m}(x_m) \tag{26}$$

$$\mu_{x_{\text{leaf}} \to f}(x) = 0 \tag{27}$$

$$\mu_{f_{\text{leaf}} \to x}(x) = \log f(x). \tag{28}$$

When at the root,

$$\log p^{\max} = \max_x \left(\sum_{f \in ne(x)} \mu_{f \to x}(x) \right). \tag{29}$$

The back pointers are the same. The max-product or max-sum algorithm, when applied to HMMs, is known as the Viterbi algorithm.

★ *BPexample.pdf*

4 The Mean Field Algorithm

For concreteness, consider the *Ising model* where the random variables x take values in $\{0, 1\}$. The density is

$$p_\theta(x) = \frac{1}{Z} \exp\left(\sum_{s \in V} \theta_s x_s + \sum_{(s,t) \in E} \theta_{st} x_s x_t \right) \tag{30}$$

Consider Gibbs sampling on this graphical model. Fixing values at all nodes except x_s, the Gibbs sampler assigns value 0 or 1 to x_s according to the conditional probability $p(x_s \mid x_{-s}) = p(x_s \mid x_{N(s)})$, where $N(s)$ is the set of neighboring nodes of s and the equation holds because of the Markov property. It is easy to see that the Gibbs sampling distribution is

$$p(x_s = 1 \mid x_{N(s)}) = \frac{1}{\exp(-(\theta_s + \sum_{t \in N(s)} \theta_{st} x_t)) + 1}. \tag{31}$$

In Gibbs sampling, one would draw a new binary value for x_s based on the above equation. Departing from Gibbs sampling, we look at a continuous variable μ_s (the estimated marginal $p(x_s = 1)$) with a similar update

$$\mu_s \leftarrow \frac{1}{\exp(-(\theta_s + \sum_{t \in N(s)} \theta_{st} \mu_t)) + 1}. \tag{32}$$

The μ's are updated iteratively, too. This is known as the *mean field* algorithm for the Ising model.

References

[1] Martin J Wainwright and Michael I Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc., Hanover, MA, USA, 2008.