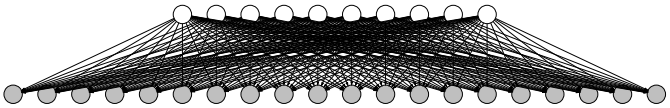
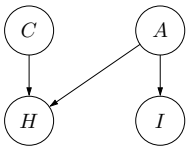
10.2: Bayesian networks IV



Review: Bayesian network



$$\mathbb{P}(C=c, A=a, H=h, I=i) = p(c)p(a)p(h \mid c,a)p(i \mid a)$$

Definition: Bayesian network

Let $X = (X_1, \ldots, X_n)$ be random variables.

A **Bayesian network** is a directed acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions, one for each node:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

- Recall that a Bayesian network allows us to define a **joint** probability distribution over many variables (e.g., $\mathbb{P}(C, A, H, I)$) by specifying **local** conditional distributions (e.g., $p(i \mid a)$).

Review: probabilistic inference

Bayesian network:

$$\mathbb{P}(X = x) = \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

Probabilistic inference:

$$\mathbb{P}(Q \mid E = e)$$

Algorithms:

- Variable elimination: general, exact
- Forward-backward: HMMs, exact
- Gibbs sampling, particle filtering: general, approximate

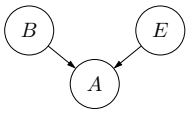
- In the last two lectures, we focused on algorithms for probabilistic inference: how do we efficiently compute queries of interest? We can do many things in closed form by leveraging the conditional independence structure of Bayesian networks.
- For HMMs, we could use the forward-backward algorithm to compute the queries efficiently.
- For general Bayesian networks / factor graphs, we must resort to an approximate algorithm such as Gibbs sampling or particle filtering.

Paradigm

Modeling

Inference **Learning**

Where do parameters come from?



b	$p(b)$
1	?
0	?

e	$p(e)$
1	?
0	?

b	e	a	$p(a \mid b, e)$
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

- Today's lecture focuses on the following question: where do all the local conditional distributions come from? These local conditional distributions are the parameters of the Bayesian network.

Roadmap

Supervised learning

Laplace smoothing

- The plan for today is tackle the supervised setting, where our training data consists of a set of **complete assignments**. The algorithms here are just counting and normalizing.
- Then we'll talk about smoothing, which is a mechanism for guarding against overfitting. This involves just a small tweak to existing algorithms.
- Finally, we'll consider the unsupervised or missing data setting, where our training data consists of a set of **partial assignments**.

Learning task

Training data

$\mathcal{D}_{\text{train}}$ (an example is an assignment to X)

Parameters

θ (local conditional probabilities)

- As with any learning algorithm, we start with the data. We will first consider the supervised setting, where each data point (example) is a complete assignment to all the variables in the Bayesian network.
- We will first develop the learning algorithm intuitively on some simple examples. Later, we will provide the algorithm for the general case and a formal justification based on maximum likelihood.

Example: one variable

Setup:

- One variable R representing the rating of a movie $\{1, 2, 3, 4, 5\}$

$$\mathbb{P}(R = r) = p(r)$$

Parameters:

$$\theta = (p(1), p(2), p(3), p(4), p(5))$$

Training data:

$$\mathcal{D}_{\text{train}} = \{1, 3, 4, 4, 4, 4, 4, 5, 5, 5\}$$

- Suppose you want to study how people rate movies. We will develop several Bayesian networks of increasing complexity, and show how to learn the parameters of these models. (Along the way, we'll also practice doing a bit of modeling.)
- Let's start with the world's simplest Bayesian network, which has just one variable representing the movie rating. Here, there are 5 parameters, each one representing the probability of a given rating.
- (Technically, there are only 4 parameters since the 5 numbers sum to 1 so knowing 4 of the 5 is enough. But we will call it 5 for simplicity.)

Example: one variable

Learning:

$$\mathcal{D}_{\text{train}} \quad \Rightarrow \quad \theta$$

Intuition: $p(r) \propto$ number of occurrences of r in $\mathcal{D}_{\text{train}}$

Example:

$$\mathcal{D}_{\text{train}} = \{1, 3, 4, 4, 4, 4, 4, 5, 5, 5\}$$

θ:

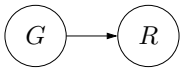
r	$p(r)$
1	0.1
2	0.0
3	0.1
4	0.5
5	0.3

- Given the data, which consists of a set of ratings (the order doesn't matter here), the natural thing to do is to set each parameter $p(r)$ to be the empirical fraction of times that r occurs in $\mathcal{D}_{\text{train}}$.

Example: two variables

Variables:

- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Rating $R \in \{1, 2, 3, 4, 5\}$



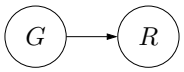
$$\mathbb{P}(G = g, R = r) = p_G(g) p_R(r \mid g)$$

$$\mathcal{D}_{\text{train}} = \{(\text{d}, 4), (\text{d}, 4), (\text{d}, 5), (\text{c}, 1), (\text{c}, 5)\}$$

Parameters: $\theta = (p_G, p_R)$

- Let's enrich the Bayesian network, since people don't rate movies completely randomly; the rating will depend on a number of factors, including the genre of the movie. This yields a two-variable Bayesian network.
- We now have two local conditional distributions, $p_G(g)$ and $p_R(r \mid g)$, each consisting of a set of probabilities, one for each setting of the values.
- Note that we are explicitly using the subscript G and R to uniquely identify the local conditional distribution inside the parameters. In this case, we could just infer it from context, but when we talk about parameter sharing later, specifying the precise local conditional distribution will be important.
- There should be $2 + 2 \cdot 5 = 12$ total parameters in this model. (Again technically there are $1 + 2 \cdot 4 = 9$.).

Example: two variables



$$\mathbb{P}(G = g, R = r) = p_G(g)p_R(r \mid g)$$

$$\mathcal{D}_{\text{train}} = \{(\text{d}, 4), (\text{d}, 4), (\text{d}, 5), (\text{c}, 1), (\text{c}, 5)\}$$

Intuitive strategy: Estimate each local conditional distribution (p_G and p_R) separately

θ:

g	$p_G(g)$
d	3/5
c	2/5

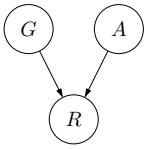
g	r	$p_R(r \mid g)$
d	4	2/3
d	5	1/3
c	1	1/2
c	5	1/2

- To learn the parameters of this model, we can handle each local conditional distribution separately (this will be justified later). This leverages the modular structure of Bayesian networks.
- To estimate $p_G(g)$, we look at the data but just ignore the value of r. To estimate $p_R(r \mid g)$, we go through each value of g and estimate the probability for each r.
- Operationally, it's convenient to first keep the integer count for each local assignment, and then just normalize by the total count for each assignment.

Example: v-structure

Variables:

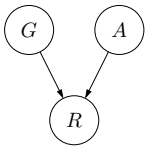
- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Won award $A \in \{0, 1\}$
- Rating $R \in \{1, 2, 3, 4, 5\}$



$$\mathbb{P}(G = g, A = a, R = r) = p_G(g)p_A(a)p_R(r \mid g, a)$$

- While probabilistic inference with V-structures was quite subtle, learning parameters for V-structures is really the same as any other Bayesian network structure. We just need to remember that the parameters include the conditional probabilities for each joint assignment to both parents.
- In this case, there are roughly $2+2+(2 \cdot 2 \cdot 5) = 24$ parameters to set (or 18 if you're more clever). Given the five data points though, most of these parameters will be zero (without smoothing, which we'll talk about later).

Example: v-structure



$$\mathcal{D}_{\text{train}} = \{(\text{d}, 0, 3), (\text{d}, 1, 5), (\text{c}, 0, 1), (\text{c}, 0, 5), (\text{c}, 1, 4)\}$$

Parameters: $\theta = (p_G, p_A, p_R)$

θ:

g	$p_G(g)$
d	3/5
c	2/5

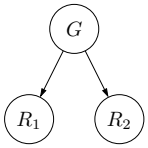
a	$p_A(a)$
0	3/5
1	2/5

g	a	r	$p_R(r \mid g, a)$
d	0	3	1
d	1	5	1
c	0	1	1/2
c	0	5	1/2
c	1	4	1

Example: inverted-v structure

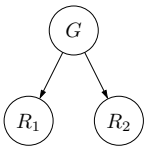
Variables:

- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Jim's rating $R_1 \in \{1, 2, 3, 4, 5\}$
- Martha's rating $R_2 \in \{1, 2, 3, 4, 5\}$



$$\mathbb{P}(G = g, R_1 = r_1, R_2 = r_2) = p_G(g)p_{R_1}(r_1 \mid g)p_{R_2}(r_2 \mid g)$$

Example: inverted-v structure



$\mathcal{D}_{\text{train}} = \{(\text{d}, 4, 5), (\text{d}, 4, 4), (\text{d}, 5, 3), (\text{c}, 1, 2), (\text{c}, 5, 4)\}$

Parameters: $\theta = (p_G, p_{R_1}, p_{R_2})$

θ:

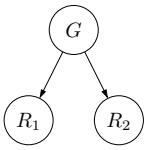
g	$p_G(g)$
d	3/5
c	2/5

g	r_1	$p_{R_1}(r \mid g)$
d	4	2/3
d	5	1/3
c	1	1/2
c	5	1/2

g	r_2	$p_{R_2}(r \mid g)$
d	3	1/3
d	4	1/3
d	5	1/3
c	2	1/2
c	4	1/2

- Let's suppose now that you're trying to model two people's ratings, those of Jim and Martha. We can define a three-node Bayesian network.
- Learning the parameters in this way works the same as before.

Example: inverted-v structure



$\mathcal{D}_{\text{train}} = \{(\text{d}, 4, 5), (\text{d}, 4, 4), (\text{d}, 5, 3), (\text{c}, 1, 2), (\text{c}, 5, 4)\}$

Parameters: $\theta = (p_G, p_R)$

θ:

g	$p_G(g)$
d	3/5
c	2/5

g	r	$p_R(r \mid g)$
d	3	1/6
d	4	3/6
d	5	2/6
c	1	1/4
c	2	1/4
c	4	1/4
c	5	1/4

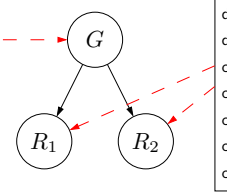
- Recall that currently, every local conditional distribution is estimated separately. But this is non-ideal if some variables behave similarly (e.g., if Jim and Martha have similar movie tastes).
- In this case, it would make more sense to have one local conditional distribution. To perform estimation in this setup, we simply go through each example (e.g., (d, 4, 5)) and each variable, and increment the counts on the appropriate local conditional distribution (e.g., 1 for $p_G(\text{d})$, 1 for $p_R(4 \mid \text{d})$, and 1 for $p_R(5 \mid \text{d})$). Finally, we normalize the counts to get local conditional distributions.

Parameter sharing

Key idea: parameter sharing

The local conditional distributions of different variables use the same parameters.

g	$p_G(g)$
c	2/5
d	3/5



g	r	$p_R(r \mid g)$
d	1	0/6
d	2	0/6
d	3	1/6
d	4	3/6
d	5	2/6
c	1	1/4
c	2	1/4
c	3	0/4
c	4	1/4
c	5	1/4

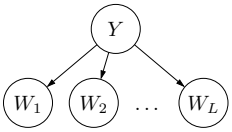
Result: more reliable estimates, less expressive

- This is the idea of **parameter sharing**. Think of each variable as being powered by a local conditional distribution (a table). Importantly, each table can drive multiple variables.
- Note that when we were talking about probabilistic inference, we didn't really care about where the conditional distributions came from, because we were just reading from them; it didn't matter whether $p(r_1 \mid g)$ and $p(r_2 \mid g)$ came from the same source. In learning, we have to write to those distributions, and where we write to matters. As an analogy, think of when passing by value and passing by reference yield the same answer.

Example: Naive Bayes

Variables:

- Genre $Y \in \{\text{comedy}, \text{drama}\}$
- Movie review (sequence of words): $W_1, \ldots, W_L$



$$\mathbb{P}(Y = y, W_1 = w_1, \ldots, W_L = w_L) = p_{\text{genre}}(y) \prod_{j=1}^{L} p_{\text{word}}(w_j \mid y)$$

Parameters: $\theta = (p_{\text{genre}}, p_{\text{word}})$

- As an extension of the previous example, consider the popular Naive Bayes model, which can be used to model the contents of documents (say, movie reviews about comedies versus dramas). The model is said to be "naive" because all the words are assumed to be conditionally independent given class variable Y.
- In this model, there is a lot of parameter sharing: each word W_j is generated from the same distribution p_{word}.

Question

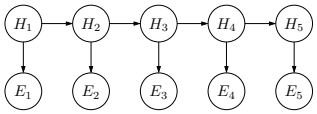
If Y can take on 2 values and each W_j can take on D values, how many parameters are there?

- There are $L + 1$ variables, but all but Y are powered by the same local conditional distribution. We have 2 parameters for p_{genre} and $2D$ for p_{word}, for a total of $2 + 2D = O(D)$. Importantly, due to parameter sharing, there is no dependence on L.

Example: HMMs

Variables:

- $H_1, \ldots, H_n$ (e.g., actual positions)
- $E_1, \ldots, E_n$ (e.g., sensor readings)



$$\mathbb{P}(H = h, E = e) = p_{\text{start}}(h_1) \prod_{i=2}^{n} p_{\text{trans}}(h_i \mid h_{i-1}) \prod_{i=1}^{n} p_{\text{emit}}(e_i \mid h_i)$$

Parameters: $\theta = (p_{\text{start}}, p_{\text{trans}}, p_{\text{emit}})$

$\mathcal{D}_{\text{train}}$ is a set of full assignments to (H, E)

- The HMM is another model, which we saw was useful for object tracking.
- With K possible hidden states (values that H_t can take on) and D possible observations, the HMM has K^2 transition parameters and KD emission parameters.

General case

Bayesian network: variables $X_1, \ldots, X_n$

Parameters: collection of distributions $\theta = \{p_d : d \in D\}$ (e.g., $D = \{\text{start}, \text{trans}, \text{emit}\}$)

Each variable X_i is generated from distribution p_{d_i}:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} p_{d_i}(x_i \mid x_{\text{Parents}(i)})$$

Parameter sharing: d_i could be same for multiple i

- Now let's consider how to learn the parameters of an arbitrary Bayesian network with arbitrary parameter sharing. You should already have the basic intuitions; the next few slides will just be expressing these intuitions in full generality.
- The parameters of a general Bayesian network include a set of local conditional distributions indexed by $d \in D$. Note that many variables can be powered by the same $d \in D$.

General case: learning algorithm

Input: training examples $\mathcal{D}_{\text{train}}$ of full assignments

Output: parameters $\theta = \{p_d : d \in D\}$

Algorithm: maximum likelihood for Bayesian networks

Count:
  For each $x \in \mathcal{D}_{\text{train}}$:
    For each variable x_i:
      Increment $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$

Normalize:
  For each d and local assignment $x_{\text{Parents}(i)}$:
    Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$

- Estimating the parameters is a straightforward generalization. For each distribution, we go over all the training data, keeping track of the number of times each local assignment occurs. These counts are then normalized to form the final parameter estimates.

Maximum likelihood

Maximum likelihood objective:

$$\max_{\theta} \prod_{x \in \mathcal{D}_{\text{train}}} \mathbb{P}(X = x; \theta)$$

Algorithm on previous slide exactly computes maximum likelihood parameters (closed form solution).

Solution: can take logs, use Lagrange multipliers, and solve for the best θ

- So far, we've presented the count-and-normalize algorithm, and hopefully this seems to you like a reasonable thing to do. But what's the underlying principle?
- It can be shown that the algorithm that we've been using is no more than a closed form solution to the **maximum likelihood** objective, which says we should try to find θ to maximize the probability of the training examples.

Maximum likelihood

$\mathcal{D}_{\text{train}} = \{(\text{d}, 4), (\text{d}, 5), (\text{c}, 5)\}$

$$\prod_{x \in \mathcal{D}_{\text{train}}} \mathbb{P}(X = x; \theta) = p_G(\text{d}) p_R(4 \mid \text{d}) p_G(\text{d}) p_R(5 \mid \text{d}) p_G(\text{c}) p_R(5 \mid \text{c})$$

$$\sum_{x \in \mathcal{D}_{\text{train}}} \log \mathbb{P}(X = x; \theta) = \log p_G(\text{d}) + \log p_R(4 \mid \text{d}) + \log p_G(\text{d}) + \log p_R(5 \mid \text{d}) + \log p_G(\text{c}) + \log p_R(5 \mid \text{c}))$$

- Key: decomposes into subproblems, one for each distribution d and assignment x_{Parents}
- For each subproblem, solve in closed form (Lagrange multipliers for sum-to-1 constraint)

Solution: can take logs, use Lagrange multipliers, and solve for the best θ

- We won't go through the math, but from the small example, it's clear we can switch the order of the factors.
- Notice that the problem decomposes into several independent pieces (one for each conditional probability distribution d and assignment to the parents).
- Each such subproblem can be solved easily (using the solution from the foundations homework).

Roadmap

Supervised learning

Laplace smoothing

Scenario 1

Setup:

- You have a coin with an unknown probability of heads $p(\text{H})$.
- You flip it 100 times, resulting in 23 heads, 77 tails.
- What is estimate of $p(\text{H})$?

Maximum likelihood estimate:

$$p(\text{H}) = 0.23 \quad p(\text{T}) = 0.77$$

- Having established the basic learning algorithm for maximum likelihood, let's try to stress test it a bit.
- Just to review, the maximum likelihood estimate in this case is what we would expect and seems quite reasonable.

Scenario 2

Setup:

- You flip a coin once and get heads.
- What is estimate of $p(\text{H})$?

Maximum likelihood estimate:

$$p(\text{H}) = 1 \quad p(\text{T}) = 0$$

Intuition: This is a bad estimate; real $p(\text{H})$ should be closer to half

When have less data, maximum likelihood overfits, want a more reasonable estimate...

- However, if we had just one data point, maximum likelihood places probability 1 on heads, which is a horrible idea. It's a very close-minded thing to do: just because we didn't see something doesn't mean it can't exist!
- This is an example of overfitting. If we had millions of parameters and only thousands of data points (which is not enough data), maximum likelihood would surely put 0 in many of the parameters.

Regularization: Laplace smoothing

Maximum likelihood:

$$p(\mathsf{H}) = \frac{1}{1} \quad p(\mathsf{T}) = \frac{0}{1}$$

Maximum likelihood with Laplace smoothing:

$$p(\mathsf{H}) = \frac{1+1}{1+2} = \frac{2}{3} \quad p(\mathsf{T}) = \frac{0+1}{1+2} = \frac{1}{3}$$

- There is a very simple fix to this called **Laplace smoothing**: just add 1 to the count for each possible value, regardless of whether it was observed or not.
- Here, both heads and tails get an extra count of 1.

Example: two variables

$\mathcal{D}_{\text{train}} = \{(\mathsf{d}, 4), (\mathsf{d}, 5), (\mathsf{c}, 5)\}$

Amount of smoothing: $\lambda = 1$

θ:

g	$p_G(g)$
d	3/5
c	2/5

g	r	$p_R(r \mid g)$
d	1	1/7
d	2	1/7
d	3	1/7
d	4	2/7
d	5	2/7
c	1	1/6
c	2	1/6
c	3	1/6
c	4	1/6
c	5	2/6

- As a concrete example, let's revisit the two-variable model from before.
- For example, d occurs 2 times, but ends up at 3 due to adding $\lambda = 1$. In particular, many values which were never observed in the data have positive probability as desired.

Regularization: Laplace smoothing

Key idea: Laplace smoothing

For each distribution d and partial assignment $(x_{\text{Parents}(i)}, x_i)$, add λ to $\text{count}_d(x_{\text{Parents}(i)}, x_i)$.

Then normalize to get probability estimates.

Interpretation: hallucinate λ occurrences of each local assignment

Larger $\lambda \Rightarrow$ more smoothing $\Rightarrow$ probabilities closer to uniform.

Data wins out in the end:

$$p(\mathsf{H}) = \frac{1+1}{1+2} = \frac{2}{3} \quad p(\mathsf{H}) = \frac{998+1}{998+2} = 0.999$$

- More generally, we can add a number $\lambda > 0$ (sometimes called a **pseudocount**) to the count for each distribution d and local assignment $(x_{\text{Parents}(i)}, x_i)$.
- By varying λ, we can control how much we are smoothing.
- No matter what the value of λ, as we get more and more data, the effect of λ will diminish. This is desirable, since if we have a lot of data, we should be able to trust our data more.

Summary

(Bayesian network without parameters) + training examples

↓

Learning: maximum likelihood (+Laplace smoothing)

↓

$$Q \mid E \Rightarrow \boxed{\begin{array}{c}\text{Parameters } \theta \\ \text{(of Bayesian network)}\end{array}} \Rightarrow \mathbb{P}(Q \mid E; \theta)$$

Next time: learning from data with **partial** assignments