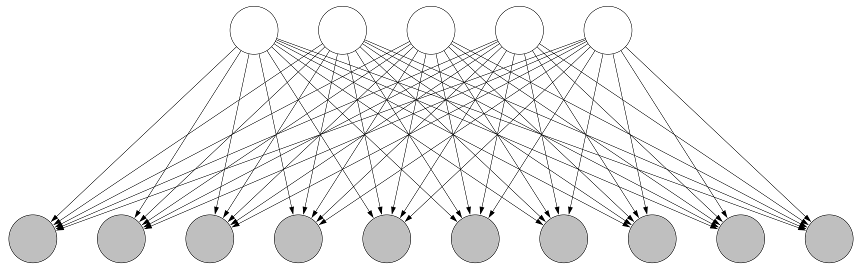
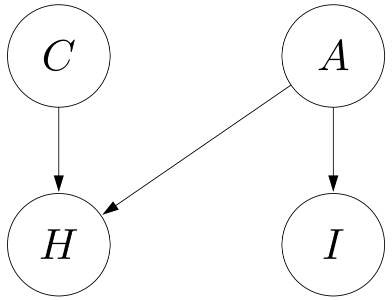
Bayesian networks: supervised learning



- So far, we have introduced Bayesian networks and talked about how to perform inference in them. In this module, we will turn to the question of how to learn them from data.

Review: Bayesian network



Random variables:

cold C, allergies A, cough H, itchy eyes I

Joint distribution:

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

Definition: Bayesian network

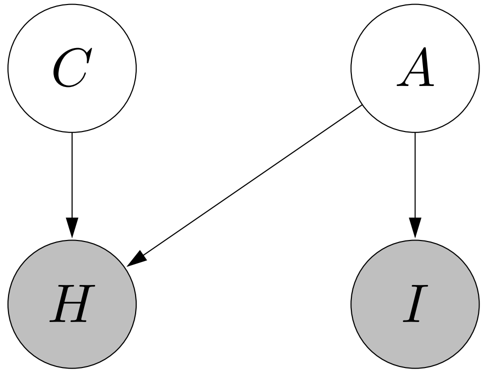
Let $X = (X_1, \dots, X_n)$ be random variables.

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:

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

- Recall that a Bayesian network is given by (i) a set of random variables, (ii) directed edges between those variables capturing qualitative dependencies, (iii) local conditional distributions of each variable given its parents which captures these dependencies quantitatively, and (iv) a joint distribution which is produced by multiplying all the local conditional distributions together.

Review: probabilistic inference



Question: $\mathbb{P}(C \mid H = 1, I = 1)$

Input

Bayesian network: $\mathbb{P}(X_1, \dots, X_n)$

Evidence: $E = e$ where $E \subseteq X$ is subset of variables

Query: $Q \subseteq X$ is subset of variables

$\Downarrow$

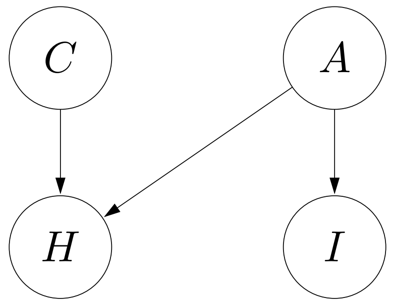
Output

$\mathbb{P}(Q \mid E = e) \longleftrightarrow \mathbb{P}(Q = q \mid E = e)$ for all values q

Algorithms: Gibbs sampling, forward-backward, particle filtering

- Given the joint distribution representing your probabilistic database, you can answer all sorts of questions on it using probabilistic inference.
- Given a set of evidence variables and values, a set of query variables, we want to compute the probability of the query variables given the evidence, marginalizing out all other variables.
- We have seen several algorithms including exhaustive enumeration, Gibbs sampling, forward-backward, and particle filtering for performing inference.

Where do parameters come from?



c	$p(c)$
1	?
0	?

a	$p(a)$
1	?
0	?

c	a	h	$p(h \mid c, a)$
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

a	i	$p(i \mid a)$
0	0	?
0	1	?
1	0	?
1	1	?

- Inference assumes that the local conditional distributions are known. But where do all the local conditional distributions come from? These local conditional distributions are the parameters of the Bayesian network.

Learning task

Training data

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

$$\big\downarrow$$

Parameters

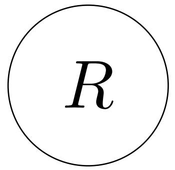
θ (local conditional probabilities)

- As with any learning algorithm, we start with the data. In this module, we'll focus on the fully-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.
- Probabilistic inference assumes you know the parameters, whereas learning does not, so one might think that inference should be easier. However, for Bayesian networks, somewhat surprisingly, it turns out that learning (at least in the fully-supervised setting) is easier.

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'll use this as a running example. We will develop several Bayesian networks of increasing complexity, and show how to learn the parameters of each 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.)
- Suppose you're giving this training data.

Example: one variable

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

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

$$\Downarrow$$

θ:

r	count(r)	$p(r)$
1	1	0.1
2	0	0.0
3	1	0.1
4	5	0.5
5	3	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 $\mathcal{D}_{\text{train}}$. Just count and normalize!

Example: two variables

Variables:

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

$$\begin{tikzcd} G \arrow[r] & R \end{tikzcd} \qquad \mathbb{P}(G = g, R = r) = p_G(g) p_R(r \mid g)$$

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

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 θ. 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.
- Here, 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

$$\begin{tikzcd} G \arrow[r] & R \end{tikzcd}$$

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

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

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

θ:

g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

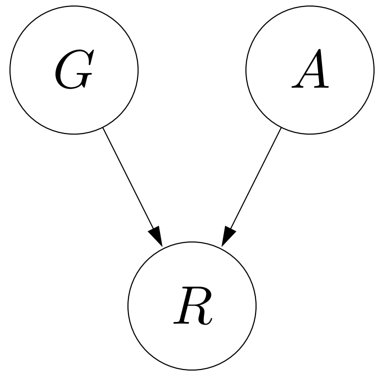
g	r	$\text{count}_R(g, r)$	$p_R(r \mid g)$
d	4	2	2/3
d	5	1	1/3
c	1	1	1/2
c	5	1	1/2

- 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. Count and normalize. To estimate $p_R(r \mid g)$, we go through each value of g and estimate the probability for each r. Count and normalize.

Example: v-structure

Variables:

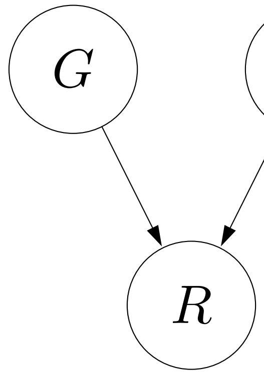
- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Won award $A \in \{0, 1\}$
- Rating $R \in \{1, 2, 3, 4, 5\}$


$$\begin{tikzcd} G \arrow[dr] & & A \arrow[dl] \\ & R & \end{tikzcd}$$

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

- Let us now consider three variables arranged in a v-structure, which remember was the special thing in Bayesian networks that gives rise to explaining away. But from the perspective of learning, there's nothing special here.

Example: v-structure



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

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

θ:

g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

a	$\text{count}_A(a)$	$p_A(a)$
0	3	3/5
1	2	2/5

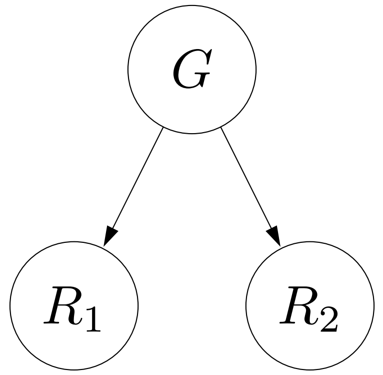
g	a	r	$\text{count}_R(g, a, r)$	$p_R(r \mid g, a)$
d	0	1	1	1/2
d	0	3	1	1/2
d	1	5	1	1
c	0	5	1	1
c	1	4	1	1

- 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: inverted-v structure

Variables:

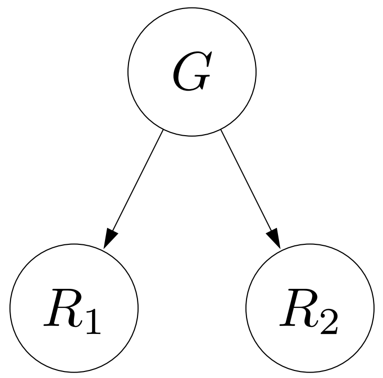
- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Jim's rating $R_1 \in \{1, 2, 3, 4, 5\}$
- Martha's rating $R_2 \in \{1, 2, 3, 4, 5\}$



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

- Let's suppose now that you're trying to model two people's ratings, those of Jim and Martha, which both depend on the genre of the movie. We can define a three-node Bayesian network.

Example: inverted-v structure



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

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

θ:

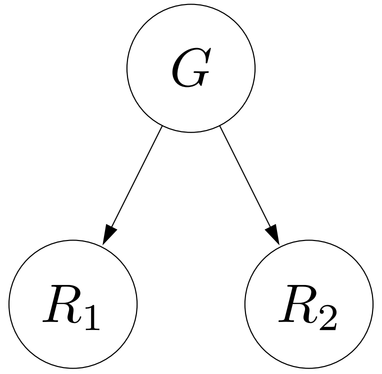
g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

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

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

- As expected, the parameters for p_{R_1} and p_{R_2} can be estimated separately (count and normalize).

Example: inverted-v structure



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

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

θ:

g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

g	r	$\text{count}_R(g, r)$	$p_R(r \mid g)$
d	3	1	1/6
d	4	3	3/6
d	5	2	2/6
c	1	1	1/4
c	2	1	1/4
c	4	1	1/4
c	5	1	1/4

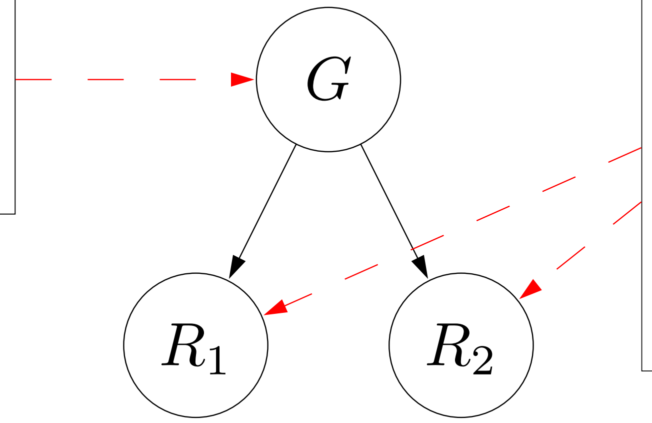
- But this is non-ideal if some variables behave similarly (e.g., if Jim and Martha have similar movie tastes).
- In this case, it would make more sense to have one local conditional distribution p_R. To perform estimation in this variant, we simply go through each example (e.g., (d, 4, 5)) and each 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 can share the same parameters.

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



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

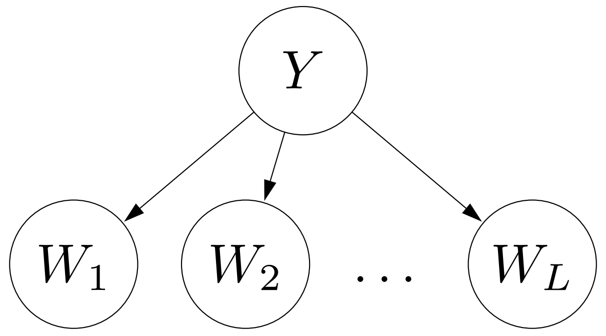
Impact: more reliable estimates, less expressive model

- 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, passing by value and passing by reference yield the same answer when you're reading, but not so when you're writing.
- When should you do parameter sharing? This is a modeling decision: you get more reliable estimates if you share parameters, but a less expressive model.

Example: Naive Bayes

Variables:

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



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

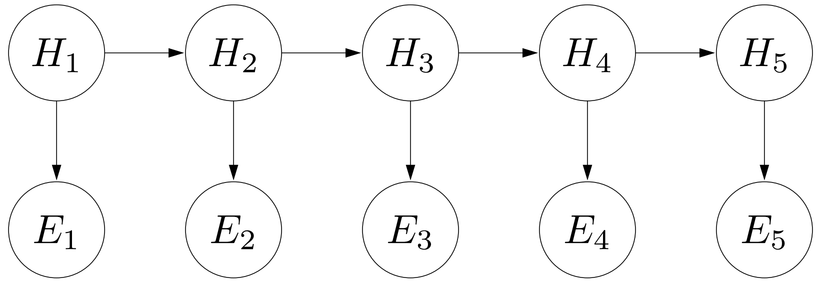
Parameters: $\theta = (p_{\text{genre}}, p_{\text{word}})$

- 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}.
- Suppose Y can take on 2 values and each W_j can take on D values. 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:

- $H_1, \dots, H_n$ (e.g., actual positions)
- $E_1, \dots, E_n$ (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.
- Here, we have three local conditional distributions which are shared across all the variables.
- 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. Again, there is no dependence on the length n.

General case

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

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}:

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

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: count and normalize

```
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: maximum likelihood

```
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 $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$
```

Closed form — no iterative optimization!

- 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

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

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

$$= \max_{p_G(\cdot)} (p_G(\mathsf{d}) p_G(\mathsf{d}) p_G(\mathsf{c})) \max_{p_R(\cdot \mid \mathsf{c})} p_R(5 \mid \mathsf{c}) \max_{p_R(\cdot \mid \mathsf{d})} (p_R(4 \mid \mathsf{d}) p_R(5 \mid \mathsf{d}))$$

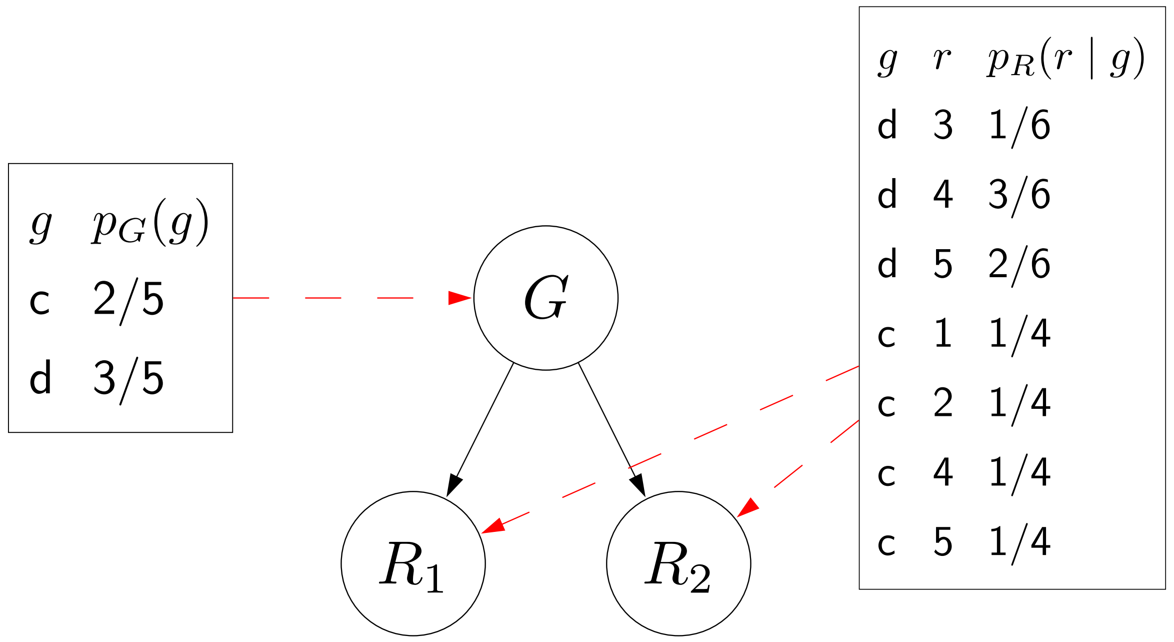
Solution:

$$p_G(\mathsf{d}) = \frac{2}{3}, p_G(\mathsf{c}) = \frac{1}{3}, p_R(5 \mid \mathsf{c}) = 1, p_R(4 \mid \mathsf{d}) = \frac{1}{2}, p_R(5 \mid \mathsf{d}) = \frac{1}{2}$$

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

- Why is this the case? We won't go through the math, but work out a 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).

Summary



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

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

- Parameter sharing: variables powered by parameters (passing by reference)
- Maximum likelihood = count and normalize

- In summary, we described learning in fully-supervised Bayesian networks.
- One important concept to remember is parameter sharing. Up until now, we just assumed each variable had some local conditional distribution without worrying about where it came from, because you just needed to read from it to do inference. But learning involves writing to it, and we need to think of the parameters as being something mutable that gets written to based on the data.
- Secondly, we've seen that performing maximum likelihood estimation in fully-supervised Bayesian networks (principled) boils down to counting and normalizing (simple and intuitive). This simplicity is what makes Bayesian networks (especially Naive Bayes) still practically useful.

Review: maximum likelihood

$$\begin{tikzcd} G \arrow[r] & R \end{tikzcd}$$

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

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

θ:

g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

g	r	$\text{count}_R(g, r)$	$p_R(r \mid g)$
d	4	2	2/3
d	5	1	1/3
c	1	1	1/2
c	5	1	1/2

Do we really believe that $p_R(r = 2 \mid g = \text{c}) = 0$?

Overfitting!

- Suppose we have a two-variable Bayesian network whose parameters (local conditional distributions) we don't know.
- Instead, we obtain training data, where each example includes a full assignment.
- Recall that maximum likelihood estimation in a Bayesian network is given by a simple count + normalize algorithm.
- But is this a reasonable thing to do? Consider the probability of a 2 rating given comedy? It's hard to believe that there is zero chance of this happening. That would be very closed-minded.
- This is a case where maximum likelihood has overfit to the training data!

Laplace smoothing example

Idea: just add $\lambda = 1$ to each count

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

θ:

g	$\text{count}_G(g)$	$p_G(g)$
d	1+3	4/7
c	1+2	3/7

g	r	$\text{count}_R(g, r)$	$p_R(g, r)$
d	1	1	1/8
d	2	1	1/8
d	3	1	1/8
d	4	1+2	3/8
d	5	1+1	2/8
c	1	1+1	2/7
c	2	1	1/7
c	3	1	1/7
c	4	1	1/7
c	5	1+1	2/7

Now $p_R(r = 2 \mid g = \text{c}) = \frac{1}{7} > 0$

- There is a very simple patch to this form of overfitting called **Laplace smoothing**: just add some small constant λ (called a **pseudocount** or virtual count) for each possible value, regardless of whether it was observed or not.
- As a concrete example, let's revisit the two-variable model from before.
- We preload all the counts (now we have to write down all the possible assignments to g and r) with λ. Then we add the counts from the training data and normalize all the counts.
- Note that many values which were never observed in the data have positive probability as desired.

Laplace smoothing

Key idea: maximum likelihood with Laplace smoothing

For each distribution d and partial assignment $(x_{\text{Parents}(i)}, x_i)$:

Add λ to $\text{count}_d(x_{\text{Parents}(i)}, x_i)$.

Further increment counts $\{\text{count}_d\}$ based on $\mathcal{D}_{\text{train}}$.

Hallucinate λ occurrences of each local assignment

- More formally, when we do maximum likelihood with Laplace smoothing with smoothing parameter $\lambda > 0$, we add λ to the count for each distribution d and local assignment $(x_{\text{Parents}(i)}, x_i)$. Then we increment the counts based on the training data $\mathcal{D}_{\text{train}}$.
- Advanced: Laplace smoothing can be interpreted as using a Dirichlet prior over probabilities and doing maximum a posteriori (MAP) estimation.

Interplay between smoothing and data

Larger λ $\Rightarrow$ more smoothing $\Rightarrow$ probabilities closer to uniform

g	count$_G(g)$	$p_G(g)$
d	1/2+1	3/4
c	1/2	1/4

g	count$_G(g)$	$p_G(g)$
d	1+1	2/3
c	1	1/3

Data wins out in the end (suppose only see $g =$ d):

g	count$_G(g)$	$p_G(g)$
d	1+1	2/3
c	1	1/3

g	count$_G(g)$	$p_G(g)$
d	1+998	0.999
c	1	0.001

- By varying λ, we can control how much we are smoothing. The larger the λ, the stronger the smoothing, and the closer the resulting probability estimates become to the uniform distribution.
- However, no matter what the value of λ is, 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 and more.

Summary

g	$\text{count}_G(g)$	$p_G(g)$
d	$\lambda + 1$	$\frac{1+\lambda}{1+2\lambda}$
c	λ	$\frac{\lambda}{1+2\lambda}$

- Pull distribution closer to uniform distribution

- Smoothing gets washed out with more data

- In conclusion, Laplace smoothing provides a simple way to avoid overfitting by adding a smoothing parameter λ to all the counts, pulling the final probability estimates away from any zeros and towards the uniform distribution.
- But with more amounts of data, then the effect of smoothing wanes.