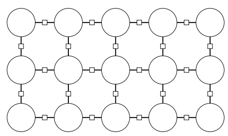
Bayesian Networks III



Lecture: Bayesian networks

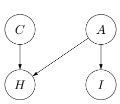
Learning: Supervised learning

Learning: Smoothing

Learning: EM Algorithm

- 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

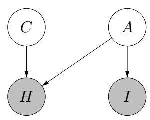
Let $X = (X_1, \dots, X_n)$ be random variables.

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

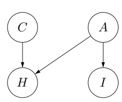
Output

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

$\downarrow$

Parameters

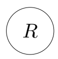
θ (local conditional probabilities)

- 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 $\mathcal{D}_{\text{train}}$

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

θ:

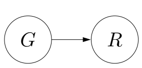
r	count(r)	$p(r)$
1	1	0.1
2	0	0.0
3	1	0.1
4	5	0.5
5	3	0.3

- 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 $R \in \{1, 2, 3, 4, 5\}$



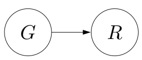
$$\mathbb{P}(G = g, R = r) = p_G(g) p_R(r \mid g)$$

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

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



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

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

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

θ:

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

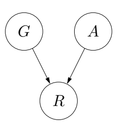
g	r	count$_R(g, r)$	$p_R(r \mid g)$
d	4	2	2/3
d	5	1	1/3
c	1	1	1/2
c	5	1	1/2

- 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:

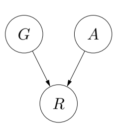
- $G \in \{\text{drama}, \text{comedy}\}$ (genre)
- $A \in \{0, 1\}$ (award)
- $R \in \{1, 2, 3, 4, 5\}$ (rating)



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

- 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	count$_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

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

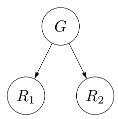
g	a	r	count$_R(g, a, r)$	$p_R(r \mid g, a)$
d	0	1	1	1/2
d	0	3	1	1/2
d	1	5	1	1
c	0	5	1	1
c	1	4	1	1

- 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 (can fix with smoothing, i.e., adding pseudocounts to avoid zeros).

Example: inverted-v structure

Variables:

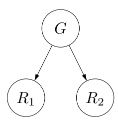
- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Jim's rating $R_1 \in \{1, 2, 3, 4, 5\}$
- Martha's rating $R_2 \in \{1, 2, 3, 4, 5\}$



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

- 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



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

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

θ:

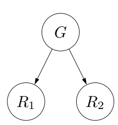
g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

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



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

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

θ:

g	$\text{count}_G(g)$	$p_G(g)$
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

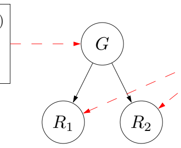
- But this is non-ideal if some variables behave similarly (e.g., if Jim and Martha have similar movie tastes).
- In this case, it would make more sense to have one local conditional distribution p_R. To perform estimation in this variant, we simply go through each example (e.g., (d, 4, 5)) and each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

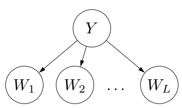
Impact: more reliable estimates, less expressive model

- 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): $W_1, \dots, W_L$



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

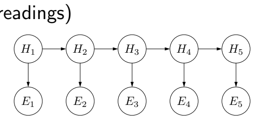
Parameters: $\theta = (p_{\text{genre}}, p_{\text{word}})$

- 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:

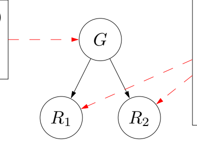
$$p_G(\mathsf{d}) = \frac{2}{3}, p_G(\mathsf{c}) = \frac{1}{3}, p_R(5 \mid \mathsf{c}) = 1, p_R(4 \mid \mathsf{d}) = \frac{1}{2}, p_R(5 \mid \mathsf{d}) = \frac{1}{2}$$

- 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.

Lecture: Bayesian networks

Learning: Supervised learning

Learning: Smoothing

Learning: EM Algorithm

- In this module, I'll talk about how Laplace smoothing for guarding against overfitting.

Review: maximum likelihood

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

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

θ:

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

g	r	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	count$_G(g)$	$p_G(g)$
d	1+3	4/7
c	1+2	3/7

g	r	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 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 $\lambda \Rightarrow$ more smoothing $\Rightarrow$ probabilities closer to uniform

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

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

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

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

g	$\text{count}_G(g)$	$p_G(g)$
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.

Lecture: Bayesian networks

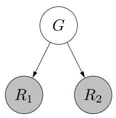
Learning: Supervised learning

Learning: Smoothing

Learning: EM Algorithm

- In this module, I'll introduce the EM algorithm for learning Bayesian networks when we have unobserved variables in our training data.

Motivation



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

If observe all the variables: maximum likelihood = count and normalize

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

What if we **don't observe** some of the variables?

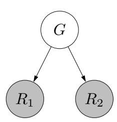
$$\mathcal{D}_{\text{train}} = \{(?, 4, 5), (?, 4, 4), (?, 5, 3), (?, 1, 2), (?, 5, 4)\}$$

- Let's start with our familiar movie rating example, where we have genre G, Jim's rating R_1, and Martha's rating R_2.
- If we observe all the variables in each training example, then we saw how we can do maximum likelihood estimation (a.k.a. count + normalize).
- Data collection is hard, and often we don't observe the value of every single variable. Maybe we only see the ratings (R_1, R_2), but not the genre G. Can we learn in this setting, which is clearly more difficult?
- Intuitively, it might seem hopeless. After all, how can we ever learn anything about the relationship between G and R_1 if we never observe G at all?
- The magic of EM (or unsupervised learning in general) is that you can in many (but certainly not all) cases.

Maximum marginal likelihood

Variables: H is hidden, $E = e$ is observed

Example:



$H = G \qquad E = (R_1, R_2) \qquad e = (1, 2)$
$\theta = (p_G, p_R)$

Maximum marginal likelihood objective:

$$\max_\theta \prod_{e \in \mathcal{D}_{\text{train}}} \mathbb{P}(E = e; \theta)$$
$$= \max_\theta \prod_{e \in \mathcal{D}_{\text{train}}} \sum_h \mathbb{P}(H = h, E = e; \theta)$$

- Let's try to solve this problem top-down — what do we want, mathematically?
- Formally we have a set of hidden variables H, observed variables E, and parameters θ, which define all the local conditional distributions. We observe $E = e$, but we don't know H or θ.
- If there were no hidden variables, then we would just use maximum likelihood: $\max_\theta \prod_{(h,e) \in \mathcal{D}_{\text{train}}} \mathbb{P}(H = h, E = e; \theta)$. But since H is unobserved, we can simply replace the joint probability $\mathbb{P}(H = h, E = e; \theta)$ with the marginal probability $\mathbb{P}(E = e; \theta)$, which is just a sum over values h that the hidden variables H could take on.

Expectation Maximization (EM)

Intuition: generalization of the K-means algorithm

cluster centroids = parameters θ cluster assignments = hidden variables H

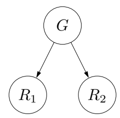
Variables: H is hidden, $E = e$ is observed

Algorithm: Expectation Maximization (EM)

Initialize θ randomly
Repeat until convergence:
  E-step:
    Compute $q(h) = \mathbb{P}(H = h \mid E = e; \theta)$ for each h (probabilistic inference)
    Create fully-observed weighted examples: (h, e) with weight $q(h)$
  M-step:
    Maximum likelihood (count and normalize) on weighted examples to get θ

- Expectation Maximization (EM), which was developed in statistics in the 1970s, is an algorithm that attempts to maximize the marginal likelihood, although special cases had been developed earlier (e.g., for HMMs).
- To get intuition for EM, consider K-means, which turns out to be a special case of EM (for Gaussian mixture models with variance tending to 0). In K-means, we had to somehow estimate the cluster centers, but we didn't know which points were assigned to which clusters. And in that setting, we took an alternating optimization approach: find the best cluster assignment given the current cluster centers, find the best cluster centers given the assignments, etc.
- The EM algorithm works analogously. EM consists of alternating between two steps, the E-step and the M-step. In the E-step, we don't know what the hidden variables are, so we compute the posterior distribution over them given our current parameters ($\mathbb{P}(H \mid E = e; \theta)$). This can be done using any probabilistic inference algorithm. If H takes on a few values, then we can enumerate over all of them. If $\mathbb{P}(H, E)$ is defined by an HMM, we can use the forward-backward algorithm. These posterior distributions provide a weight $q(h)$ (which is a temporary variable in the EM algorithm) to every value h that H could take on. Conceptually, the E-step then generates a set of weighted full assignments (h, e) with weight $q(h)$. (In implementation, we don't need to create the data points explicitly, since we can just add counts directly.)
- In the M-step, we take in our set of full assignments (h, e) with weights, and we just do maximum likelihood estimation, which can be done in closed form — just counting and normalizing (perhaps with smoothing if you want)!
- If we repeat the E-step and the M-step over and over again, we are guaranteed to converge to a **local optima**. Just like the K-means algorithm, we might need to run the algorithm from different random initializations of θ and take the best one.

Example: one iteration of EM



$\mathcal{D}_{\text{train}} = \{(?, 2, 2), (?, 1, 2)\}$

θ:

g	$p_G(g)$
c	0.5
d	0.5

g	r	$p_R(r \mid g)$
c	1	0.4
c	2	0.6
d	1	0.6
d	2	0.4

E-step

(r_1, r_2)	g	$\mathbb{P}(G = g, R_1 = r_1, R_2 = r_2)$	$q(g)$
(2, 2)	c	$0.5 \cdot 0.6 \cdot 0.6 = 0.18$	$\frac{0.18}{0.18+0.08} = 0.69$
(2, 2)	d	$0.5 \cdot 0.4 \cdot 0.4 = 0.08$	$\frac{0.08}{0.18+0.08} = 0.31$
(1, 2)	c	$0.5 \cdot 0.4 \cdot 0.6 = 0.12$	$\frac{0.12}{0.12+0.12} = 0.5$
(1, 2)	d	$0.5 \cdot 0.6 \cdot 0.4 = 0.12$	$\frac{0.12}{0.12+0.12} = 0.5$

M-step

θ:

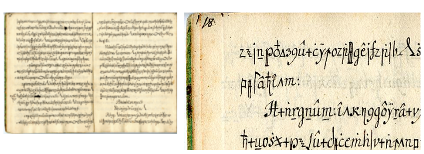
g	count	$p_G(g)$
c	0.69 + 0.5	0.59
d	0.31 + 0.5	0.41

g	r	count	$p_R(r \mid g)$
c	1	0.5	0.21
c	2	0.5 + 0.69 + 0.69	0.79
d	1	0.5	0.31
d	2	0.5 + 0.31 + 0.31	0.69

- In the E-step, we are presented with the current set of parameters θ. We go through all the examples (in this case $(2, 2)$ and $(1, 2)$). For each example (r_1, r_2), we will consider all possible values of g (c or d), and compute the posterior distribution $q(g) = \mathbb{P}(G = g \mid R_1 = r_1, R_2 = r_2)$.
- The easiest way to do this is to write down the joint probability $\mathbb{P}(G = g, R_1 = r_1, R_2 = r_2)$ because this is just simply a product of the parameters. For example, the first line is the product of $p_G(\text{c}) = 0.5$, $p_R(2 \mid \text{c}) = 0.6$ for $r_1 = 2$, and $p_R(2 \mid \text{c}) = 0.6$ for $r_2 = 2$. For each example (r_1, r_2), we normalize these joint probability to get $q(g)$.
- Now each row consists of a fictitious data point with g filled in, but appropriately weighted according to the corresponding $q(g)$, which is based on what we currently believe about g.
- In the M-step, for each of the parameters (e.g., $p_G(\text{c})$), we simply add up the weighted number of times that parameter was used in the data (e.g., 0.69 for (c, 2, 2) and 0.5 for (c, 1, 2)). Then we normalize these counts to get probabilities.
- If we compare the old parameters and new parameters after one round of EM, you'll notice that parameters tend to sharpen (though not always): probabilities tend to move towards 0 or 1.

Application: decipherment

Copiale cipher (105-page encrypted volume from 1730s):



Cracked in 2011 with the help of EM!

- Let's now look at an interesting application of EM (or Bayesian networks in general): decipherment. Given a ciphertext (a string), how can we decipher it?
- The Copiale cipher was deciphered in 2011 (it turned out to be the handbook of a German secret society), largely with the help of Kevin Knight, an NLP researcher.
- Real ciphers are a bit too complex, so we will focus on the simple case of substitution ciphers.

Substitution ciphers

Letter substitution table (unknown):

Plain:	`abcdefghijklmnopqrstuvwxyz`
Cipher:	`plokmijnuhbygvtfcrdxeszaqw`

Plaintext (unknown): `hello world`

Ciphertext (known): `nmyyt ztryk`

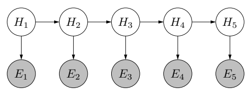
Challenge: Give ciphertext, recover the plaintext

- The input to decipherment is a ciphertext. Let's put our modeling hats on and think about how this ciphertext came to be.
- In a simple substitution cipher, someone comes up with a permutation of the letters (e.g., "a" maps to "p"). You can think about these as the unknown parameters of the model.
- Then they think of something to say — the plaintext (e.g., "hello world"). Finally, they apply the substitution table to generate the ciphertext (deterministically).

Application: decipherment as an HMM

Variables:

- $H_1, \dots, H_n$ (e.g., characters of plaintext)
- $E_1, \dots, E_n$ (e.g., characters of ciphertext)

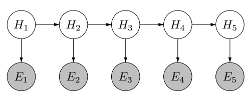


$$\mathbb{P}(H = h, E = e) = p_{\text{start}}(h_1) \prod_{i=2}^{n} p_{\text{trans}}(h_i \mid h_{i-1}) \prod_{i=1}^{n} p_{\text{emit}}(e_i \mid h_i)$$

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

- We can formalize this process as an HMM as follows. The hidden variables are the plaintext and the observations are the ciphertext. Each character of the plaintext is related to the corresponding character in the ciphertext based on the cipher, and the transitions encode the fact that the characters in English are highly dependent on each other. For simplicity, we use a character-level bigram model (though n-gram models would yield better results).

Application: decipherment as an HMM



Strategy:

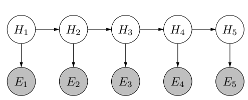
- p_{start}: set to uniform
- p_{trans}: estimate on tons of English text
- p_{emit}: **substitution table**, estimated from EM

Intuitions:

- p_{trans} to favor plaintexts h that look like English
- p_{emit} favors consistent characters substitutions

- We need to specify how we estimate the starting probabilities p_{start} the transition probabilities p_{trans}, and the emission probabilities p_{emit}.
- The **starting probabilities** we won't care about so much and just set to a uniform distribution.
- The **transition probabilities** specify how someone might have generated the plaintext. We can estimate p_{trans} on a large corpora of English text. Note we need not use the same data to estimate all the parameters of the model. Indeed, there is generally much more English plaintext lying around than ciphertext. This is one of the other nice things about Bayesian networks, is that estimation can sometimes be done in a modular way.
- The **emission probabilities** encode the substitution table. Here, we know that the substitution table is deterministic, but we let the parameters be general distributions, which can certainly encode deterministic functions (e.g., $p_{\text{emit}}(\mathsf{p} \mid \mathsf{a}) = 1$). We use EM to only estimate the emission probabilities.
- We emphasize that the principal difficulty here is that we neither know the plaintext nor the parameters! But why might this work? The intuition is that the transitions p_{trans} (which are known, so it's a bit easier than standard EM) will favor plaintexts h that look like English (e.g., $h_{i-1} = \mathsf{t}$ to $h_i = \mathsf{a}$ rather than to $h_i = \mathsf{b}$). The emissions p_{emit} will favor character substitutions that are consistent (so all occurrences of a should be mapped to the same character).

Application: decipherment as an HMM



E-step: forward-backward computes for each position i and character h

$$q_i(h) \stackrel{\text{def}}{=} \mathbb{P}(H_i = h \mid E_1 = e_1, \dots E_n = e_n)$$

M-step: count (fractional) and normalize for all characters e, h

$$\text{count}_{\text{emit}}(h, e) = \textstyle\sum_{i : e_i = e} q_i(h)$$

$$p_{\text{emit}}(e \mid h) \propto \text{count}_{\text{emit}}(h, e)$$

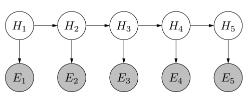
- Let's focus on the EM algorithm for estimating the emission probabilities. In the E-step, we can use the forward-backward algorithm to compute the posterior distribution over hidden assignments $\mathbb{P}(H \mid E = e)$. More precisely, the algorithm returns $q_i(h) \stackrel{\text{def}}{=} \mathbb{P}(H_i = h \mid E = e)$ for each position $i = 1, \dots, n$ and possible hidden state h.
- We can use $q_i(h)$ as fractional counts of each H_i. To compute the counts $\text{count}_{\text{emit}}(h, e)$, we loop over all the positions algorithm i where $E_i = e$ and add the fractional count $q_i(h)$.

Decipherment in Python

[code]

- In the code, we first estimate the Markov model p_{trans} on some plain text. Then we run EM to estimate the p_{emit}, where we leave p_{start} and p_{trans} alone.
- As you can see from the demo, the result isn't perfect, but not bad given the difficulty of the problem and the simplicity of the approach.

Summary



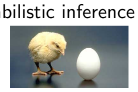
Maximum marginal likelihood:

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

EM algorithm:

$\Leftarrow$ probabilistic inference (E-step)

hidden variables $q(h)$



parameters θ

count and normalize (M-step) $\Rightarrow$

Applications: decipherment, phylogenetic reconstruction, crowdsourcing

- In summary, we introduced the EM algorithm for estimating the parameters of a Bayesian network when there are unobserved variables.
- The principle we follow is maximum marginal likelihood. The algorithm that optimizes this is the EM algorithm, which is very intuitive. Ultimately, like in k-means, we have a chicken-and-egg problem, where we don't know the hidden variables and we also don't know the parameters.
- But we can update each conditioned on the other: In the E-step, we use probabilistic inference to compute a distribution over hidden variables conditioned on the evidence. In the M-step, we have a weighted set of fully-observable examples, and we simply count and normalize. This procedure is guaranteed to converge to a local optimum of the marginal likelihood objective.
- Finally, after you have learned the parameters of your Bayesian network, you can go off and perform inference to answer all sorts of questions, which could be on the unobserved variables on new test examples or completely other variables. This highlights the flexibility of Bayesian networks in dealing with heterogenous data between training and test time.
- There are many applications of the EM algorithm. We looked at a simple form of decipherment, where we try to infer the plaintext form the ciphertext. EM can also be used to reconstruct the phylogenetic tree given the DNA of modern organisms. It can also be used to infer the unknown label of a data point, where the observations are the possibly noisy labels provided by crowdworkers.
- EM is the most canonical version of a broader class of variational inference approaches, which include things like variational autoencoders (VAEs), where the q distribution (encoder) is given by a neural network, and the Bayesian network is the decoder. I'd encourage you to go explore this connection in more detail.

Course plan

Search problems | Constraint satisfaction problems
Markov decision processes | Markov networks
Adversarial games | Bayesian networks

Reflex **States** **Variables** **Logic**

Low-level → High-level

Machine learning

- A reminder of where we are in the course. We just completed our discussion of Bayesian Networks, which included an overall description, and a discussion of inference and learning,
- More generally Variable-based models included a discussion of constraint satisfaction, Markov models, and Bayesian networks This concludes our discussion of variable-based models Next, we will move on to the last module on logic and higher-level reasoning.