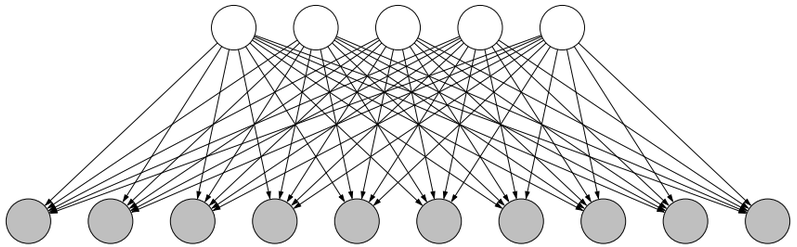
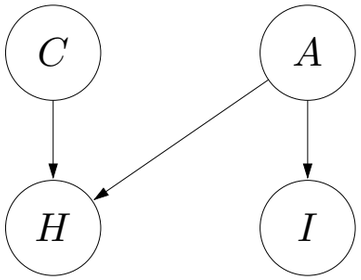
Bayesian networks: supervised learning



Learn Bayesian Networks from data

Review: Bayesian network

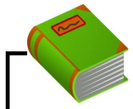


Random variables:

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

Joint distribution:

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



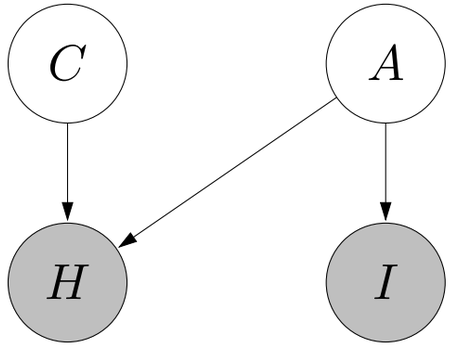
Definition: Bayesian network

Let $X = (X_1, \ldots, X_n)$ 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:

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

Review: probabilistic inference



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

Input

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

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

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

distribution over query conditioned on evidence

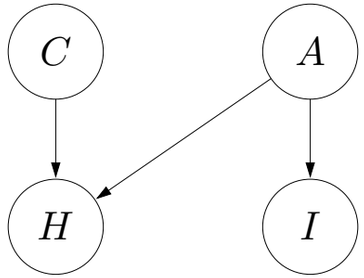
Output

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

Algorithms: Gibbs sampling, forward-backward, particle filtering

Where do parameters come from?

the local conditional distributions



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	?

Learning task

given data

Training data

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

find the parameters

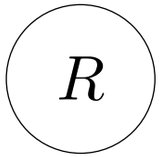
Parameters

θ (local conditional probabilities)

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:

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

prob of rating for each movie

Training data:

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

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:

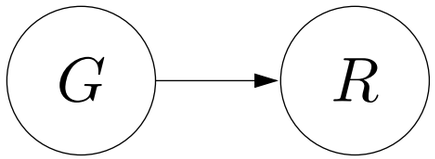
r	count(r)	$p(r)$
1	1	0.1
2	0	0.0
3	1	0.1
4	5	0.5
5	3	0.3

— prob of ruby 1 is 0.1

Example: two variables

Variables: — 2 variables

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



2 params 5 × 2 params

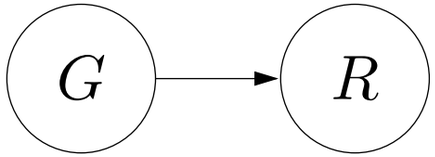
$$\mathbb{P}(G = g, R = r) = p_G(g)p_R(r \mid g)$$

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

12 total params

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

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

count & normalize

θ:

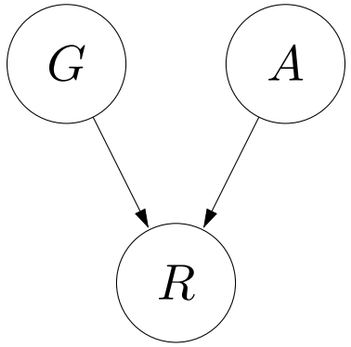
g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

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

Example: v-structure

Variables:

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

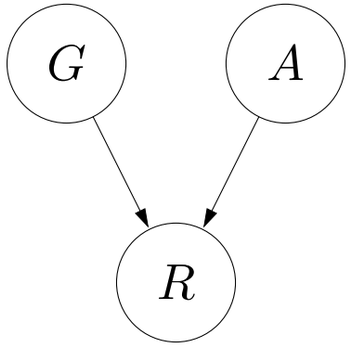


2 params 2 params

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

5 × 2 × 2 params

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

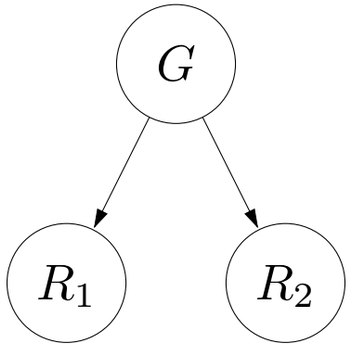
g	a	r	$\text{count}_R(g, a, r)$	$p_R(r \mid g, a)$
d	0	1	1	1/2
d	0	3	1	1/2
d	1	5	1	1
c	0	5	1	1
c	1	4	1	1

normalize on based condition

Example: inverted-v structure

Variables:

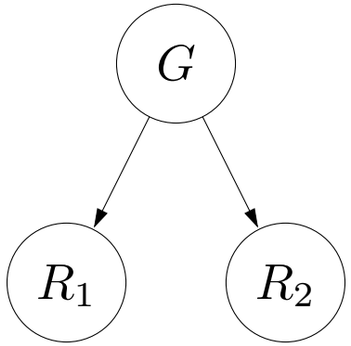
- Genre $G \in \{\text{drama}, \text{comedy}\}$
- Jim's rating $R_1 \in \{1, 2, 3, 4, 5\}$
- Martha's rating $R_2 \in \{1, 2, 3, 4, 5\}$



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

2 5×2 5×2

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

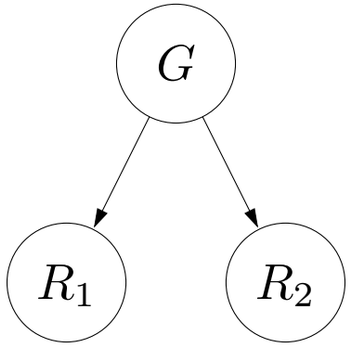
θ:

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

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

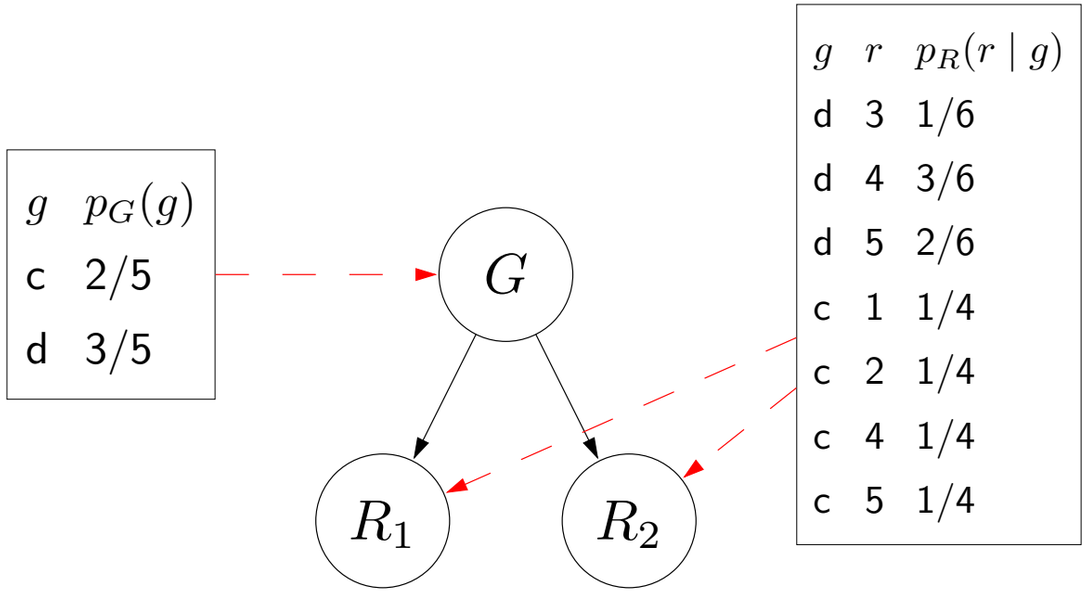
g	r	$\text{count}_R(g, r)$	$p_R(r \mid g)$
d	3	1	1/6
d	4	3	3/6
d	5	2	2/6
c	1	1	1/4
c	2	1	1/4
c	4	1	1/4
c	5	1	1/4

- 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.



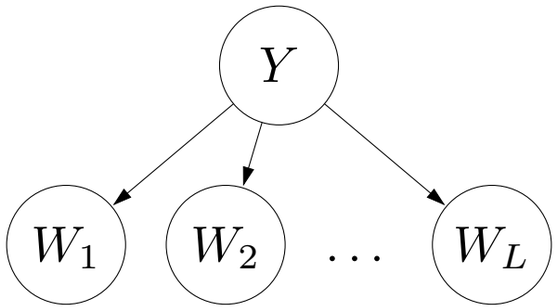
Impact: more reliable estimates, less expressive model

- 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): $W_1, \ldots, W_L$



each word is independent

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

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

2 params

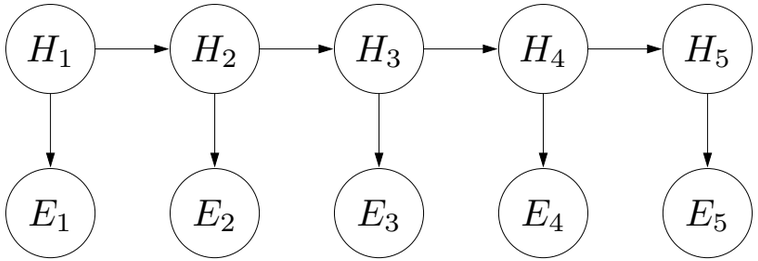
2 × domain of W_i

- 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:

- $H_1, \ldots, H_n$ (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.
- 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 $X_1, \ldots, X_n$

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}:

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

Every variable X_i has a distribution d_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 $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$

increment counter for distribution d_i behind variable X_i

Normalize:

For each d and local assignment $x_{\text{Parents}(i)}$:

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

Maximum likelihood

Maximum likelihood objective:

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

— training data

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!

Maximum likelihood

$p_G(d)\, p_R(4 \mid d)$

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

θ is joint PMF, which is equal to product of local

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

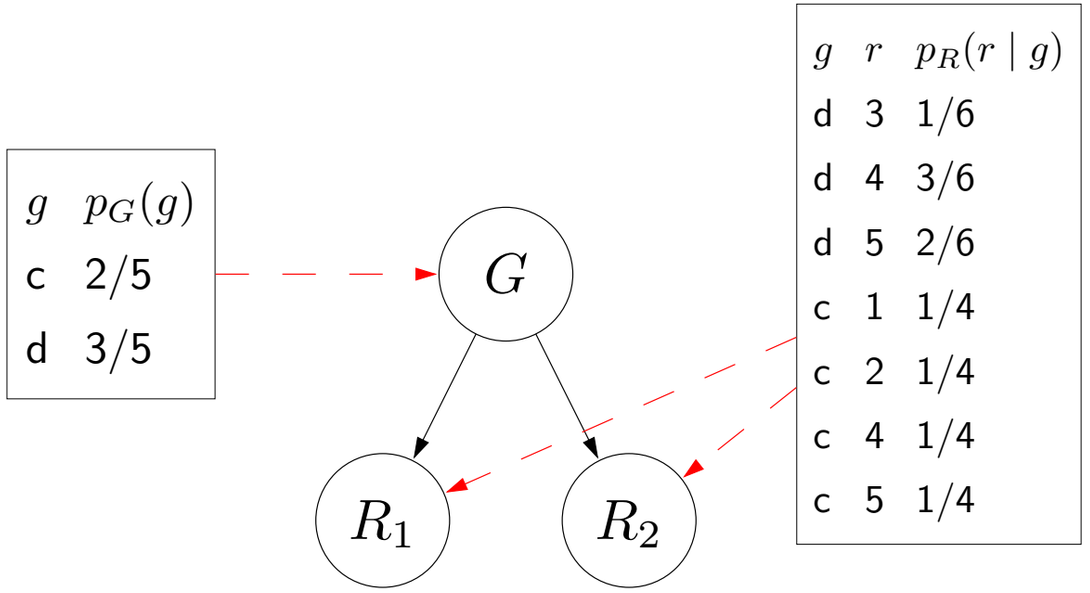
$$= \max_{p_G(\cdot)} (p_G(\text{d}) p_G(\text{d}) p_G(\text{c})) \max_{p_R(\cdot \mid \text{c})} p_R(5 \mid \text{c}) \max_{p_R(\cdot \mid \text{d})} (p_R(4 \mid \text{d}) p_R(5 \mid \text{d}))$$

Solution:

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

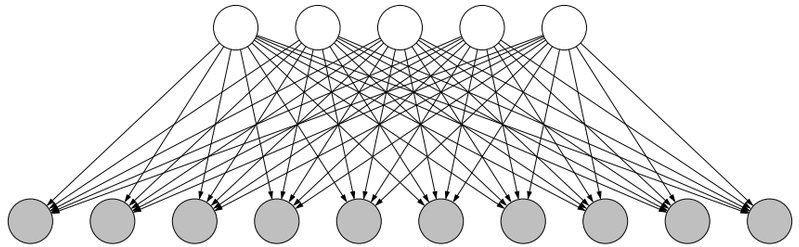
- Decomposes into subproblems, one for each distribution d and assignment to parents x_{Parents}
- For each subproblem, solve in closed form (Lagrange multipliers for sum-to-1 constraint)

Summary



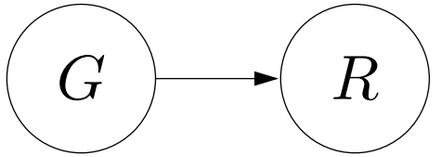
- Parameter sharing: variables powered by parameters (passing by reference)
- Maximum likelihood = count and normalize

Bayesian networks: smoothing



Laplace smoothing to prevent overfitting

Review: maximum likelihood



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

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

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

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 $g = \text{d}$):

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.

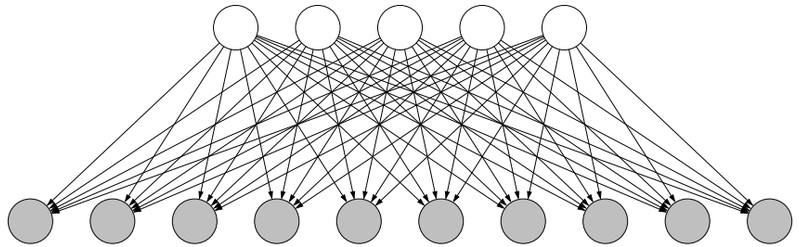
Summary

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

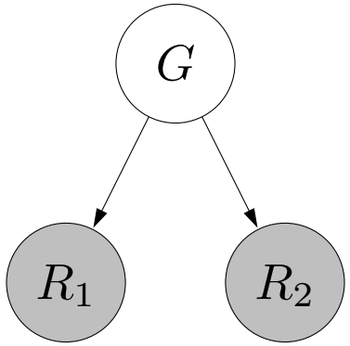
- Smoothing gets washed out with more data

Bayesian networks: 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?

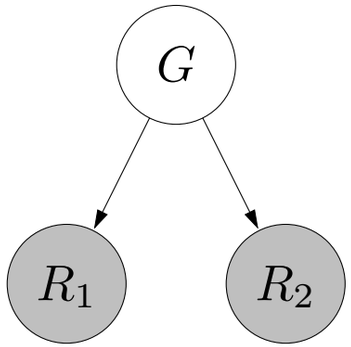
$$\mathcal{D}_{\text{train}} = \{(?, 4, 5), (?, 4, 4), (?, 5, 3), (?, 1, 2), (?, 5, 4)\}$$

- 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:

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

marginalization

- 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:

guess hidden variables and parameters so far

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 $q(h)$

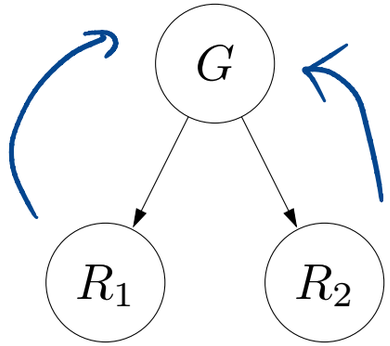
M-step:

Maximum likelihood (count and normalize) on weighted examples to get θ

* guaranteed to converge to local optimum

- 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:

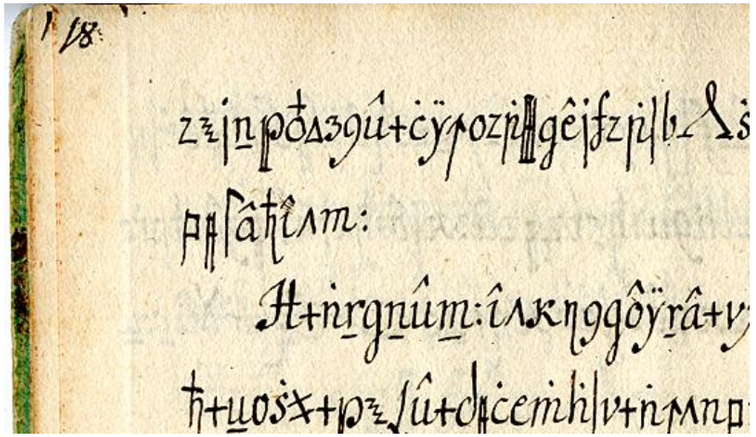
g	count	$p_G(g)$
c	0.69 + 0.5	0.59
d	0.31 + 0.5	0.41

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):

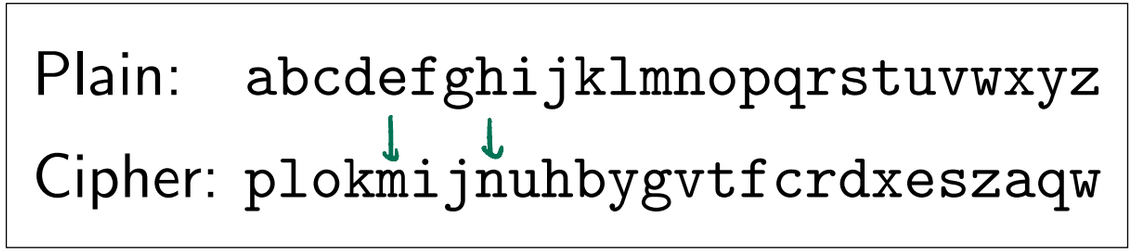


Cracked in 2011 with the help of EM!

- 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):



Plaintext (unknown): `hello world`

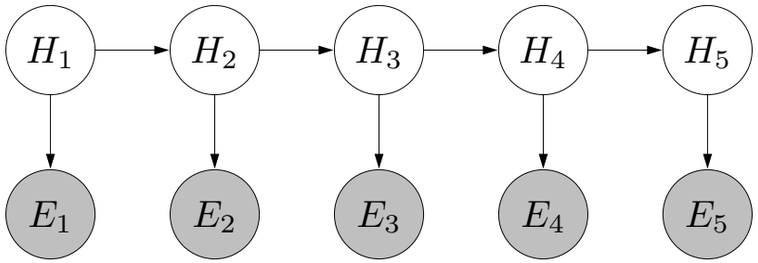
Ciphertext (known): `nmyyt ztryk`

Challenge: Give ciphertext, recover the plaintext

Application: decipherment as an HMM

Variables:

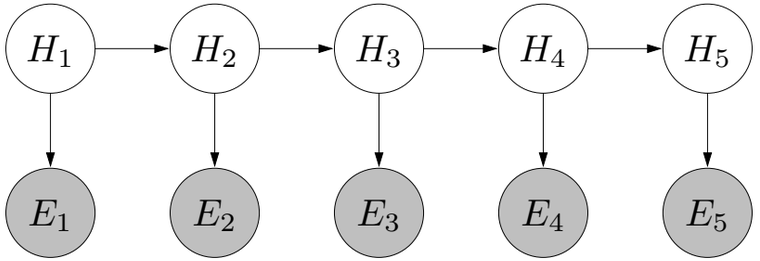
- $H_1, \ldots, H_n$ (e.g., characters of plaintext)
- $E_1, \ldots, E_n$ (e.g., characters of ciphertext)



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

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

Application: decipherment as an HMM



Strategy:

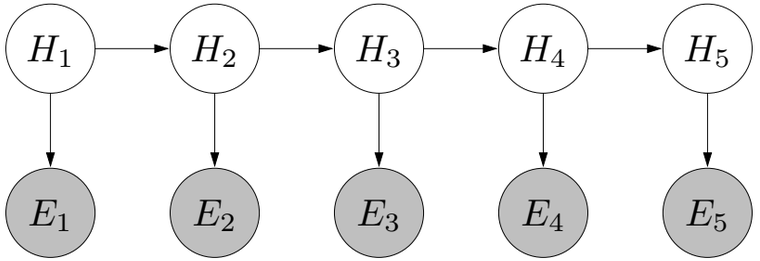
- p_{start}: set to uniform
- p_{trans}: estimate on tons of English text
- p_{emit}: **substitution table**, estimated from EM

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., $p_{\text{emit}}(\text{p} \mid \text{a}) = 1$). 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., $h_{i-1} = \text{t}$ to $h_i = \text{a}$ rather than to $h_i = \text{b}$). 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

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

plain cipher

— final prob of plaintext letter h given the observed ciphertext

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

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

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

- 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 $i = 1, \ldots, n$ 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

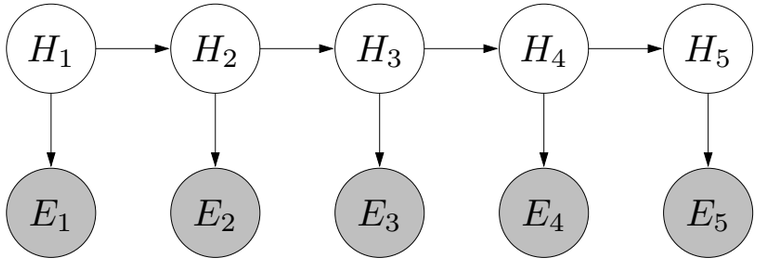
```python
K = 26 + 1  # Number of characters (lowercase letters + space)

### Initialize HMM

# startProbs[h] = p_start(h)
# Uniform distribution (DONE)
startProbs = [1.0 / K for h in range(K)]

# transitionProbs[h1][h2] = p_trans(h2 | h1)
# Estimate this from plaintext (DONE)
rawText = util.toIntSeq(util.readText('lm.train'))
transitionCounts = [[0 for h2 in range(K)] for h1 in range(K)]
for i in range(1, len(rawText)):
    h1 = rawText[i - 1]
    h2 = rawText[i]
    transitionCounts[h1][h2] += 1
transitionProbs = [util.normalize(transitionCounts[h1]) for h1 in range(K)]

# emissionProbs[h][e] = p_emit(e | h)
# Initialize it to uniform distribution
emissionProbs = [[1.0 / K for e in range(K)] for h in range(K)]

### Run EM

observations = util.toIntSeq(util.readText('ciphertext'))
n = len(observations)

for t in range(200):
    # E-step: q[i][h] = q_i(h)
    q = util.forwardBackward(observations, startProbs, transitionProbs, emissionProbs)

    # Print out best guess
    print(util.toStrSeq([util.argmax(q[i]) for i in range(n)]))
    print('')

    # M-step: count and normalize
    emissionCounts = [[0 for e in range(K)] for h in range(K)]
    for i in range(n):
        for h in range(K):
            emissionCounts[h][observations[i]] += q[i][h]
    emissionProbs = [util.normalize(emissionCounts[h]) for h in range(K)]
```

$p(H_i = h \mid E)$

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

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

Summary



Maximum marginal likelihood:

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

find params to maximize likelihood of observed variables

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.