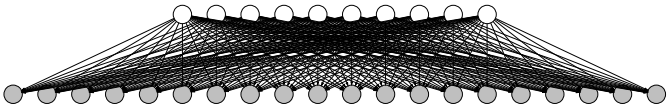
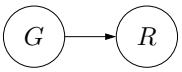
11.2: Bayesian networks V



Review: Learning Bayesian networks



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

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

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

θ:

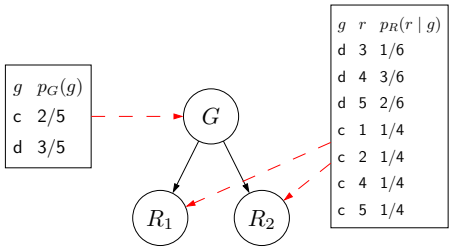
g	$p_G(g)$
d	3/5
c	2/5

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

Review: Parameter sharing

Key idea: parameter sharing

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

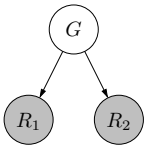


Roadmap

Expectation Maximization (EM)

HMM Demo

Motivation



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

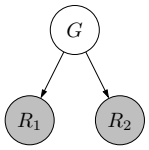
$$\mathcal{D}_{\text{train}} = \{(?, 4, 5), (?, 4, 4), (?, 5, 3), (?, 1, 2), (?, 5, 4)\}$$

- Data collection is hard, and often we don't observe the value of every single variable. In this example, 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?
- Indeed, if we don't observe enough of the variables, we won't be able to learn anything. But in many cases, we can.

Maximum marginal likelihood

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

Example:



$H = G \quad E = (R_1, R_2) \quad e = (4, 5)$

$\theta = (p_G, p_R)$

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 and 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)

Inspiration: K-means

Variables: H is hidden, E is observed (to be e)

Algorithm: Expectation Maximization (EM)

E-step:

- Compute $q(h) = \mathbb{P}(H = h \mid E = e; \theta)$ for each h (use any probabilistic inference algorithm)
- Create weighted points: (h, e) with weight $q(h)$

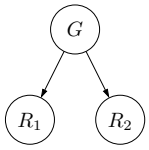
M-step:

- Compute maximum likelihood (just count and normalize) to get θ

Repeat until convergence.

- One of the most remarkable algorithms, Expectation Maximization (EM), tries to maximize the marginal likelihood.
- 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.)
- 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	$0.5 \cdot 0.6 \cdot 0.6$	$\frac{0.18}{0.18+0.08} = 0.69$
(2, 2)	d	$0.5 \cdot 0.4 \cdot 0.4$	$\frac{0.08}{0.18+0.08} = 0.31$
(1, 2)	c	$0.5 \cdot 0.4 \cdot 0.6$	$\frac{0.12}{0.12+0.12} = 0.5$
(1, 2)	d	$0.5 \cdot 0.6 \cdot 0.4$	$\frac{0.12}{0.12+0.12} = 0.5$

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.

EM properties

- EM increases objective function at each iteration

- EM converges to local optima, need to restart multiple times

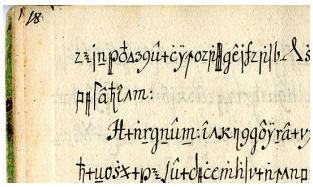
Roadmap

Expectation Maximization (EM)

HMM Demo

Application: decipherment

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



- Let's now look at an interesting potential 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. On a related note, he has been quite interested in using probabilistic models (learned with variants of EM) for decipherment. 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

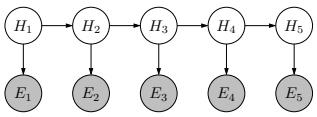
Ciphertext (known): nmyyt ztryk

- The input to decipherment is a ciphertext. Let's put on our modeling hats and think about how this ciphertext came to be.
- Most ciphers are quite complicated, so let's consider 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:

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

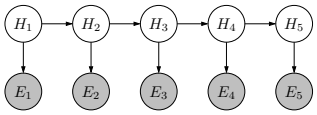


$$\mathbb{P}(H = h, E = e) = p_{\text{start}}(h_1) \prod_{i=2}^{n} p_{\text{trans}}(h_i \mid h_{i-1}) \prod_{i=1}^{n} p_{\text{emit}}(e_i \mid h_i)$$

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

- 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



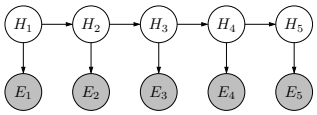
Strategy:

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

Intuition: rely on language model (p_{trans}) to favor plaintexts h that look like English

- 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.
- 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!

Application: decipherment as an HMM



E-step: forward-backward algorithm computes

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

M-step: count (fractional) and normalize

$$\text{count}_{\text{emit}}(h, e) = \textstyle\sum_{i=1}^{n} q_i(h) \cdot [e_i = e]$$

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

[live solution]

- 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 i where $E_i = e$ and add the fractional count $q_i(h)$.
- 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

(Bayesian network without parameters) + **partial** training data

Learning: EM (maximum marginal likelihood)

$Q \mid E \Rightarrow$ Parameters θ (of Bayesian network) $\Rightarrow \mathbb{P}(Q \mid E; \theta)$