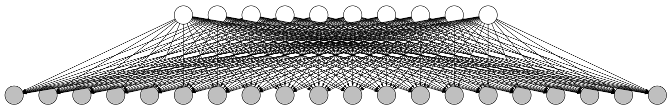
Lecture 15: Bayesian networks III



Announcements

- **car** is due tomorrow
- **p-progress** is due Thursday
- **exam** is next Tuesday
- covers material up to and including today's lecture
- reviews in Section this Thursday AND Friday
- all alternative exams have been scheduled
- problem solving, practice with old exam problems

Review: Bayesian network

$$\begin{tikzcd} C \arrow[d] & A \arrow[dl] \arrow[d] \\ H & I \end{tikzcd}$$

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

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:

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

- A Bayesian network allows us to define a **joint** probability distribution over many variables (e.g., $\mathbb{P}(C, A, H, I)$) by specifying **local** conditional distributions (e.g., $p(i \mid a)$). Two lectures ago, we talked about modeling: how can we use Bayesian networks to represent real-world problems.

Review: probabilistic inference

Bayesian network:

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

Probabilistic inference:

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

Algorithms:

- Forward-backward: HMMs, exact
- Particle filtering: HMMs, approximate
- Gibbs sampling: general, approximate

- Last lecture, we focused on algorithms for probabilistic inference: how do we efficiently compute queries of interest? We can do many things in closed form by leveraging the conditional independence structure of Bayesian networks.
- For HMMs, we could use the forward-backward algorithm to perform smoothing and filtering efficiently and exactly.
- For extremely large domains, particle filtering allows you to perform filtering efficiently but only approximately.
- For general Bayesian networks and factor graphs, we can apply Gibbs sampling, which is also approximate.

Paradigm

Modeling

Inference **Learning**

Where do parameters come from?

$$\begin{tikzcd}
B \arrow[dr] & & E \arrow[dl] \\
& A &
\end{tikzcd}$$

b	$p(b)$
1	?
0	?

e	$p(e)$
1	?
0	?

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

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

Roadmap

Supervised learning

Laplace smoothing

Unsupervised learning with EM

- The plan for today is to start with the supervised setting, where our training data consists of a set of **complete assignments**. The algorithms here are just counting and normalizing.
- Then we'll talk about smoothing, which is a mechanism for guarding against overfitting. This involves just a small tweak to existing algorithms.
- Finally, we'll consider the unsupervised or missing data setting, where our training data consists of a set of **partial assignments**. Here, the workhorse algorithm is the EM algorithm which breaks up the problem into probabilistic inference + supervised algorithms.

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

cs221.stanford.edu/q

Question

Which is computationally more expensive for Bayesian networks?

probabilistic inference given the parameters

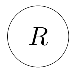
learning the parameters given fully labeled data

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

Example: one variable

Learning:

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

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

Example:

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

$$\Big\downarrow$$

θ:

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

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

Example: two variables

Variables:

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

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

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

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

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

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

Example: two variables

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

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

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

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

θ:

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

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

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

Example: v-structure

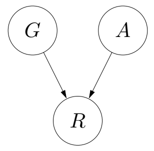
Variables:

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

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

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

Example: v-structure



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

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

θ:

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

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

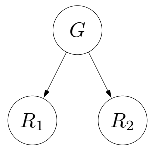
g	a	r	$p_R(r \mid g, a)$
d	0	1	1/2
d	0	3	1/2
d	1	5	1
c	0	5	1
c	1	4	1

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

Example: inverted-v structure

Variables:

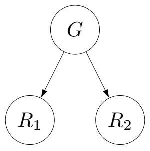
- Genre $G \in \{\mathsf{drama}, \mathsf{comedy}\}$
- Jim's rating $R_1 \in \{1, 2, 3, 4, 5\}$
- Martha's rating $R_2 \in \{1, 2, 3, 4, 5\}$



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

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

Example: inverted-v structure



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

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

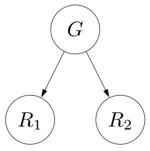
θ:

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

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

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

Example: inverted-v structure



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

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

θ:

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

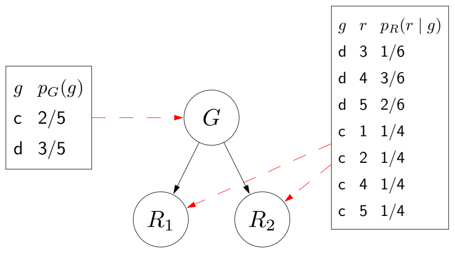
g	r	$p_R(r \mid g)$
d	3	1/6
d	4	3/6
d	5	2/6
c	1	1/4
c	2	1/4
c	4	1/4
c	5	1/4

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

Parameter sharing

Key idea: parameter sharing

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



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

g	r	$p_R(r \mid g)$
d	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

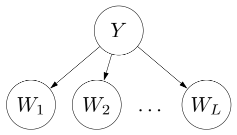
Impact: more reliable estimates, less expressive model

- 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, think of when passing by value and passing by reference yield the same answer.

Example: Naive Bayes

Variables:

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



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

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}.

cs221.stanford.edu/q

Question

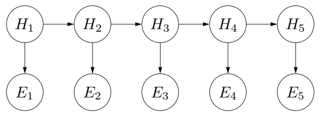
If Y can take on 2 values and each W_j can take on D values, how many parameters are there?

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

Example: HMMs

Variables:

- $H_1, \dots, H_n$ (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.
- With K possible hidden states (values that H_t can take on) and D possible observations, the HMM has K^2 transition parameters and KD emission parameters.

General case

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

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: maximum likelihood for Bayesian networks

```
Count:
  For each $x \in \mathcal{D}_{\text{train}}$:
    For each variable $x_i$:
      Increment $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$
Normalize:
  For each $d$ and local assignment $x_{\text{Parents}(i)}$:
    Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$
```

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

Maximum likelihood

Maximum likelihood objective:

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

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

- 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

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

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

Solution:

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

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

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

Roadmap

Supervised learning

Laplace smoothing

Unsupervised learning with EM

Scenario 1

Setup:

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

Maximum likelihood estimate:

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

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

Scenario 2

Setup:

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

Maximum likelihood estimate:

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

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

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

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

Regularization: Laplace smoothing

Maximum likelihood:

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

Maximum likelihood with Laplace smoothing:

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

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

Example: two variables

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

Amount of smoothing: $\lambda = 1$

θ:

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

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

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

Regularization: Laplace smoothing

Key idea: Laplace smoothing

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

Then normalize to get probability estimates.

Interpretation: hallucinate λ occurrences of each local assignment

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

Data wins out in the end:

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

- More generally, we can add a number $\lambda > 0$ (sometimes called a **pseudocount**) to the count for each distribution d and local assignment $(x_{\text{Parents}(i)}, x_i)$.
- By varying λ, we can control how much we are smoothing.
- No matter what the value of λ 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.

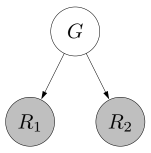
Roadmap

Supervised learning

Laplace smoothing

Unsupervised learning with EM

Motivation


$$\begin{tikzcd} & G \arrow[dl] \arrow[dr] & \\ R_1 & & R_2 \end{tikzcd}$$

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

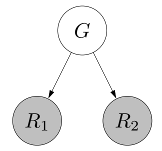
$\mathcal{D}_{\text{train}} = \{(?, 4, 5), (?, 4, 4), (?, 5, 3), (?, 1, 2), (?, 5, 4)\}$

- 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:



$H = G \quad E = (R_1, R_2) \quad e = (1, 2)$

$\theta = (p_G, p_R)$

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

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

Algorithm: Expectation Maximization (EM)

Initialize θ

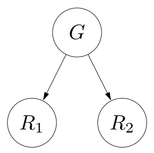
E-step:
- Compute $q(h) = \mathbb{P}(H = h \mid E = e; \theta)$ for each h (use any probabilistic inference algorithm)
- Create weighted points: (h, e) with weight $q(h)$

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 $\Rightarrow$

(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 $\Rightarrow$ θ:

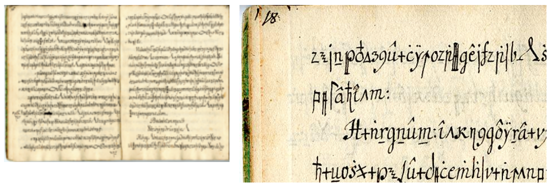
g	count	$p_G(g)$
c	0.69 + 0.5	0.59
d	0.31 + 0.5	0.41

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 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`

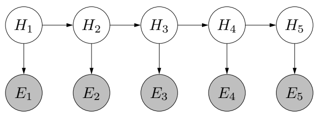
Ciphertext (known): `nmyyt ztryk`

- 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:

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

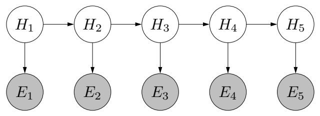


$$\mathbb{P}(H = h, E = e) = p_{\text{start}}(h_1) \prod_{i=2}^{n} p_{\text{trans}}(h_i \mid h_{i-1}) \prod_{i=1}^{n} p_{\text{emit}}(e_i \mid h_i)$$

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



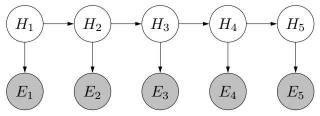
Strategy:

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

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

M-step: count (fractional) and normalize

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

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

[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 $q_i(h) \overset{\text{def}}{=} \mathbb{P}(H_i = h \mid E = e)$ 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 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

$\boxed{\text{(Bayesian network without parameters)} + \text{training examples}}$

$\Downarrow$

Learning: maximum likelihood (+Laplace smoothing, +EM)

$\Downarrow$

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