Lecture 11: Bayesian Networks – More on Inference, Maybe Learning
• Homework 2 due NOW!

• Homework 3 out this evening
  • Due MONDAY, Oct 12th

(inspect HW3)
Lecture 11: Bayesian Networks – More on Inference, Maybe Learning
Bayesian Networks

Bayesian networks are directed acyclic graphs with nodes representing random variables and edges representing dependency assumptions.

Conditional Probability Tables (CPTs)

P(Li | Lo) = 0.4
P(Li | ¬Lo) = 0.7

P(S | Lo) = 0.6
P(S | ¬Lo) = 0.2

Conditional Dependencies
Bayesian Networks

Joint distribution (factorization):
\[ P(Lo, Li, S) = P(Lo) \, P(Li \mid Lo) \, P(S) \]

P(Lo) = 0.5

Conditional Probability Tables (CPTs)

P(Li \mid Lo) = 0.4
P(Li \mid \neg Lo) = 0.7

Conditional Dependencies

P(S \mid Lo) = 0.6
P(S \mid \neg Lo) = 0.2

Long?

Liked?

Slept?
Bayesian Networks

Conditional independence:
\( P(\text{Li} | \text{Lo}, S) \)?

- \( P(\text{Lo}) = 0.5 \)
- \( P(\text{Li} | \text{Lo}) = 0.4 \)
- \( P(\text{Li} | \neg \text{Lo}) = 0.7 \)

Conditional Probability Tables (CPTs)

Conditional Dependencies

- \( P(S | \text{Lo}) = 0.6 \)
- \( P(S | \neg \text{Lo}) = 0.2 \)

Long?

Liked?

Slept?
Bayesian Networks

Conditional independence:
\[ P(Li \mid Lo, S) = \frac{P(Li \mid Lo, S)}{P(Lo)} = \frac{P(Li \mid Lo) P(Lo) P(S)}{P(Lo) P(S \mid Lo)} = P(Li \mid Lo) \]

\[ Li \perp S \mid Lo \]

---

**Conditional Probability Tables (CPTs)**

- \( P(Li \mid Lo) = 0.4 \)
- \( P(Li \mid \neg Lo) = 0.7 \)

**Conditional Dependencies**

- \( P(Lo) = 0.5 \)
- \( P(Li \mid Lo) = 0.4 \)
- \( P(Li \mid \neg Lo) = 0.7 \)
- \( P(S \mid Lo) = 0.6 \)
- \( P(S \mid \neg Lo) = 0.2 \)
Bayesian networks: Inference

• Algorithms for inferring the values of unobserved variables.

• Last time: Sampling
Stochastic Inference

We can easily sample the joint distribution to obtain possible instances:
1. Sample the free variables
2. For every other variable: If all parents have been sampled, sample based on conditional distribution

We end up with a new set of assignments for B, E, A, J and M which are a random sample from the joint

$$P(B) = .05 \quad P(E) = .1$$

$$P(A|B, E) = .95$$
$$P(A|B, \neg E) = .85$$
$$P(A|\neg B, E) = .5$$
$$P(A|\neg B, \neg E) = .05$$

$$P(J|A) = .7$$
$$P(J|\neg A) = .05$$

$$P(M|A) = .8$$
$$P(M|\neg A) = .15$$
Weighted Sampling for Computing

Problem: What if the condition rarely happens?
We would need lots and lots of samples, and most would be wasted

• Set $N_B, N_c = 0$
• Repeat:
  – Sample the joint setting the values for $J$ and $M$, compute the weight, $w$, of this sample
  – $N_c = N_c + w$
  – If $B = 1$, $N_B = N_B + w$

• After many iterations, set:
  \[ P(B \mid J, \neg M) = \frac{N_B}{N_c} \]
Bayesian networks: Inference

• Algorithms for inferring the values of unobserved variables.

• Last time: Sampling
  – fast, (often) approximate

• Last time: Exact inference
Inference

We are interested in queries of the form: $P(B \mid J, \neg M)$

This can also be written as a joint:

$$P(B \mid J, \neg M) = \frac{P(B, J, \neg M)}{P(B, J, \neg M) + P(\neg B, J, \neg M)}$$

How do we compute the new joint?

$$P(B, J, \neg M) = \sum_{a} \sum_{e} P(B, J, \neg M, a, e)$$

Sum all probabilities with these settings $(B, J, \neg M)$: the sum is over the possible assignments to the other two variables, $E$ and $A$.
Computing: $P(B, J, \lnot M)$

\[
P(B, J, \lnot M) = \sum_{a} \sum_{e} P(B, J, \lnot M, a, e) = P(B) + P(B, J, \lnot M, A, E) + P(B, J, \lnot M, \lnot A, E) + 0.0007 + 0.00001 + 0.005 + 0.0003 = 0.00601
\]
Instead of computing the value for every value of $e$
Computing: \( P(B, J, \neg M) \)

\[
P(B) \sum_e P(e) \sum_a P(a | B, e) P(\neg M | a) P(J | a)
\]

“factors”

\[
f() \quad f_E(e)
\]

“variable elimination”
Computing: $P(B, J, \neg M)$

$$P(B) \sum_e P(e) \sum_a P(a | B, e) P(\neg M | a) P(J | a)$$

“variable elimination”
Computing: $P(B, J, \neg M)$

$$P(B) \sum_e P(e) \sum_a P(a | B, e) P(\neg M | a) P(J | a)$$

“variable elimination”
Computing: $P(B, J, \neg M)$

$$P(B) \sum_e P(e) \sum_a P(a \mid B, e) P(\neg M \mid a) P(J \mid a)$$

“variable elimination”
Computing: $P(B, J, \neg M)$

$$P(B) \sum_e P(e) \sum_a P(a \mid B, e) P(\neg M \mid a) P(J \mid a)$$

$f_E(e)$

“variable elimination”
Computing: $P(B, J, \neg M)$

$$P(B) \sum_e P(e) \sum_a P(a \mid B, e) f_{E}(e) \mid a) P(J \mid a)$$

“variable elimination”
Computing: $P(B, J, \neg M)$

\[
P(B) \sum_{e} P(e) \sum_{a} P(a | B, e) f_{E}(e) | a) P(J | a)
\]

“variable elimination”
Computing: \( P(B, J, \neg M) \)

\[
P(B) \sum_e \sum_a P(a | B, f()) P(M | a) P(J | a)
\]

“variable elimination”
Actually Computing $f_E(e)$

$$f_A(a) = \left( \begin{array}{c}
P(\neg M|A) \ P(J|A) \\
P(\neg M|\neg A) \ P(J|\neg A)
\end{array} \right)$$

$$P(B) \sum_e P(e) \sum_a P(a | B,e) P(\neg M | a) P(J | a)$$

$$f_E(e) = \left( \begin{array}{c}
P(A|B,E) \ f_A(A) \\
+ P(\neg A|B,E) \ f_A(\neg A) \\
P(A|B,\neg E) \ f_A(A) \\
+ P(\neg A|B,\neg E) \ f_A(A)
\end{array} \right)$$

Reuse, don’t recompute!
btw, we computed $P(B,J,\neg M)$, but wanted $P(B|J,\neg M)$

$$P(B | J, \neg M) = \frac{P(B,J,\neg M)}{P(B,J,\neg M) + P(\neg B,J,\neg M)}$$

"normalization"

Also need to compute, but can reuse some computation again!
Actually Computing $f_E(e)$

$$f_A(a) = \begin{pmatrix} P(\neg M | A) \cdot P(J | A) \\ P(\neg M | \neg A) \cdot P(J | \neg A) \end{pmatrix}$$

$$f_E(e) = \left( P(A | B, E) \cdot f_A(A) + P(\neg A | B, E) \cdot f_A(\neg A) \right)$$

$P(B) \sum_e P(e) \sum_a P(a | B, e) P(\neg M | a) P(J | a)$

Reusable, don’t recompute!
Algorithm

- e - evidence (the variables that are observed)
- vars - the conditional probabilities derived from the network in reverse order (bottom up)

- For each var in vars
  - factors <- make_factor (var,e)
  - if var is a hidden variable then create a new factor by summing out var
- Compute the product of all factors
- Normalize
Computational Complexity

• We can reuse computations to reduce the running time
• However, there are still cases in which this algorithm will lead to exponential running time.
  – Exact Bayesian Inference is NP-Hard
• Consider the case of $f_x(y_1 \ldots y_n)$. When factoring $x$ out we would need to account for all possible values of the $y$'s.
• e.g. binary:
  $$f_x(y_1 \ldots y_n) = \begin{pmatrix} f_x(0, \ldots, 0) \\ \vdots \\ f_x(1, \ldots, 1) \end{pmatrix}$$
  $2^n$ values
Computational Complexity

• We can reuse computations to reduce the running time
• However, there are still cases in which this algorithm will lead to exponential running time.
  – Exact Bayesian Inference is NP-Hard
• Easy on trees:

\[ \sum_B P(B|A) \rightarrow f_1(A) \]
\[ \sum_C P(C|A) \rightarrow f_2(A) \]

→ never get functions (factors) with more than 1 argument (size 2)
Bayesian networks: Inference

- Algorithms for inferring the values of unobserved variables.
- Last time: Sampling
  - fast, (often) approximate
- Last time: Exact inference
  - variable elimination
- Also: “belief propagation”, “variational inference”

BP on trees = variable elimination
General DAGs need to be
Inference: compute probabilities from CPTs

But where do we get them?

Density estimation
“learning” parameters

P(B) = .05
P(E) = .1

P(A|B,E) = .95
P(A|B, ¬E) = .85
P(A| ¬B,E) = .5
P(A| ¬B, ¬E) = .05

P(J|A) = .7
P(J| ¬A) = .05

P(M|A) = .8
P(M| ¬A) = .15
Density Estimation

- A Density Estimator learns a mapping from a set of variables to a Probability, e.g. CPTs

Input data: “examples”

Density Estimator

Probability

P(B) = .05
P(E) = .1
P(A|B,E) = .95
P(A|B,¬E) = .85
P(A|¬B,E) = .5
P(A|¬B,¬E) = .05

P(J|A) = .7
P(J|¬A) = .05
P(M|A) = .8
P(M|¬A) = .15
Density estimation

- Binary and discrete variables:
  
  Easy: Just count!

- Continuous variables:
  
  Harder (but just a bit): Fit a model
Learning a density estimator

\[ \hat{P}(y_i = u) = \frac{\text{# examples in which } y_i = u}{\text{total number of examples}} \]

A trivial learning algorithm!
Learning a density estimator

\[ \hat{P}(y_i = u) = \frac{\# \text{examples in which } y_i = u}{\text{total number of examples}} \]

P(B) = ...
P(E) = ...
P(A|B,E) = ...
...
Learning a density estimator

\[ \hat{P}(y_i = u) = \frac{\# \text{examples in which } y_i = u}{\text{total number of examples}} \]
Maximum Likelihood Principle

\[ \hat{P}(\text{dataset}|M) = \hat{P}(x_1 \wedge x_2 \cdots \wedge x_R|M) = \prod_{k=1}^{R} \hat{P}(x_k|M) \]

- Fit models by maximizing the probability of generating the observed samples:
  \[ L(x_1, \ldots, x_n | \theta) = p(x_1 | \theta) \cdots p(x_n | \theta) \]
  e.g. "joint probability" from a CPT
- The examples are assumed to be independent
- For a binary random variable A with \( P(A=1) = q \)
  \[ \text{argmax}_q \text{ Likelihood} = \#(A=1)/\#\text{examples} \]
- Why?
Maximum Likelihood Principle

• For a binary random variable A with P(A=1)=q
  \[ \text{argmax}_q \text{ Likelihood} = \frac{\#(A=1)}{\#\text{examples}} \]
• Why?

Data likelihood: \[ P(D \mid q) = q^{n_1} (1-q)^{n_2} \]

We would like to find: \[ \text{arg max}_q q^{n_1} (1-q)^{n_2} \]

How?
Maximum Likelihood Principle

Data likelihood: \( P(D|q) = q^n (1-q)^s \)

We would like to find: \( \arg \max_q q^n (1-q)^s \)

\[
\frac{\partial}{\partial q} q^n (1-q)^s = n q^{n-1} (1-q)^s - q^n s (1-q)^{s-1} = 0
\]
\[
\frac{\partial}{\partial q} = 0 \Rightarrow
\]
\[
n q^{n-1} (1-q)^s - q^n s (1-q)^{s-1} = 0 \Rightarrow
\]
\[
q^{n-1} (1-q)^{s-1} (n (1-q) - q n_s) = 0 \Rightarrow
\]
\[
n (1-q) - q n_s = 0 \Rightarrow
\]
\[
n = n q + n_s q \Rightarrow
\]
\[
q = \frac{n}{n + n_s}
\]
Log Probabilities

When working with products, probabilities of entire datasets often get too small. A possible solution is to use the log of probabilities, often termed ‘log likelihood’

\[
\log \hat{P}(\text{dataset} \mid M) = \log \prod_{k=1}^{R} \hat{P}(x_k \mid M) = \sum_{k=1}^{R} \log \hat{P}(x_k \mid M)
\]

Log values between 0 and 1

Maximize that!
Density estimation

- Binary and discrete variables:
  
  Easy: Just count!

- Continuous variables:
  
  Harder (but just a bit): Fit a model

But what if we only have very few samples?
The danger of joint density estimation

\[ P(\text{summer} \& \text{size} \geq 20 \& \text{evaluation} = 3) = 0 \]
- No such example in our dataset

Now let's assume we are given a new ("test") dataset. If this dataset contains:

<table>
<thead>
<tr>
<th>Summer</th>
<th>Size</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>49</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>55</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>

Then the probability we would assign to the entire dataset is 0
Naïve Density Estimation

The problem with the Joint Estimator is that it just mirrors the training data.
We need something which generalizes more usefully.

The naïve model generalizes strongly:
Assume that each attribute is distributed independently of any of the other attributes.
Joint estimation, revisited

Assuming independence we can compute each probability independently

\[ P(\text{Summer}) = \frac{1}{2} = 0.5 \]
\[ P(\text{Evaluation} = 1) = \frac{1}{3} = 0.33 \]
\[ P(\text{Size} \geq 20) = \frac{2}{3} = 0.66 \]

How do we do on the joint?

\[ P(\text{Summer} \& \text{Evaluation} = 1) = \frac{1}{6} \]
\[ P(\text{Summer})P(\text{Evaluation} = 1) = \frac{1}{2} \times \frac{1}{3} = \frac{1}{6} \]

\[ P(\text{size} \geq 20 \& \text{Evaluation} = 1) = \frac{1}{3} = 0.33 \]
\[ P(\text{size} \geq 20)P(\text{Evaluation} = 1) = \frac{2}{3} \times \frac{1}{3} = 0.22 \]
Joint estimation, revisited

Assuming independence we can compute each probability independently

\[ P(\text{Summer}) = \frac{1}{2} = 0.5 \]
\[ P(\text{Evaluation} = 1) = \frac{1}{3} = 0.33 \]
\[ P(\text{Size} \geq 20) = \frac{2}{3} = 0.66 \]

How do we do on the joint?

\[ P(\text{Summer} \& \text{Size} \geq 20) = \frac{1}{6} = 0.16667 \]
\[ P(\text{Summer})P(\text{Size} \geq 20) = \frac{1}{2} \times \frac{2}{3} = \frac{1}{3} = 0.333 \]

We must be careful when using the Naïve density estimator.
<table>
<thead>
<tr>
<th><strong>Contrast</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Joint DE</strong></td>
<td><strong>Naïve DE</strong></td>
</tr>
<tr>
<td>Can model anything</td>
<td>Can model only very boring distributions</td>
</tr>
<tr>
<td>No problem to model “C is a noisy copy of A”</td>
<td>Outside Naïve’s scope</td>
</tr>
<tr>
<td>Given 100 records and more than 6 Boolean attributes will screw up badly</td>
<td>Given 100 records and 10,000 multivalued attributes will be fine</td>
</tr>
</tbody>
</table>
Naïve Density Estimation

The problem with the Joint Estimator is that it just mirrors the training data.

We need something which generalizes more usefully.

Joint estimator: $2^n - 1$ parameters

Naïve estimator: $n$ parameters

The naïve model generalizes strongly:

Assume that each attribute is distributed independently of any of the other attributes.
another way to deal with small datasets

- We just discussed one possibility: Naïve estimation

- Assume we want to compute the probability of heads in a coin flip (50/50)
  - What if we can only observe 3 flips?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- 25% of the times a maximum likelihood estimator will assign probability of 1 to either the heads or tails
Pseudo counts

- Use prior belief about the ‘fairness’ of most coins to influence the resulting model.
- We assume that we have “observed” 10 flips with 5 tails and 5 heads
- Thus \( P(\text{heads}) = (\#\text{heads}+5) / (\#\text{flips}+10) \)
- Advantages: 1. Never assign a probability of 0 to an event
   2. As more data accumulates we can get very close to the real distribution (the impact of the pseudo counts will diminish rapidly)
Pseudo counts

- Use *prior belief* about the ‘fairness’ of most coins to influence the results and change them toward 5 tails and 5 heads.
- Thus following our claims, we can say that...
- Advantage of this method:
  1. We can even justify this by incorporating a *real* distribution into your model!
  2. As mentioned, we should always use a *real* distribution (not the one that changes rapidly)
Let’s go back to Naïve vs. full model

What should I use?
This can be determined based on:
- Training data size
- Cross validation
- Likelihood ratio test

Cross validation is one of the most useful tricks in model fitting

Statistically valid!

Divide up data set into m parts, train on m-1, test on the 1 (do m times)
→ Which model does better?
Important points

- Showing conditional independence
- Inference: sampling & exact (variable elimination)

- Maximum likelihood estimation (MLE)
- Pseudo counts
- Cross-validation