What Probabilistic Models Should We Use?

- Full joint distribution
- Completely expressive
- Hugely data-hungry
- Exponential computational complexity
- Naive Bayes (full conditional independence)
  - Relatively concise
    - Need data $\sim (\#\text{hypotheses}) \times (\#\text{features}) \times (\#\text{feature-vals})$
    - Fast $\sim (\#\text{features})$
  - Cannot express dependencies among features or among hypotheses
  - Cannot consider possibility of multiple hypotheses co-occurring

Bayesian Networks (aka Belief Networks)

- Graphical representation of dependencies among a set of random variables
- Nodes: variables
- Directed links to a node from its parents: direct probabilistic dependencies
- Each $X_i$ has a conditional probability distribution, $P(X_i|\text{Parents}(X_i))$, showing the effects of the parents on the node.
- The graph is directed (DAG); hence, no cycles.
- This is a language that can express dependencies between Naive Bayes and the full joint distribution, more concisely
  - Given some evidence, how does this affect the probability of some other node(s)? $P(X|E)$ — belief propagation/updating
  - Given some evidence, what are the most likely values of other variables? $\arg\max_X P(X|E)$ — MAP explanation

Burglary Network (due to J. Pearl)

<table>
<thead>
<tr>
<th>$P(B)$</th>
<th>0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burglary</td>
<td>Earthquake</td>
</tr>
<tr>
<td>Alarm</td>
<td>JohnCalls</td>
</tr>
</tbody>
</table>

| $B$ | $E$ | $P(A|B,E)$ |
|-----|-----|------------|
| t   | t   | 0.95       |
| c   | f   | 0.94       |
| f   | t   | 0.29       |
| f   | f   | 0.001      |

If everything depends on everything

| $A$ | $P(J|A)$ |
|-----|----------|
| t   | 0.90     |
| f   | 0.05     |

| $A$ | $P(M|A)$ |
|-----|----------|
| t   | 0.70     |
| f   | 0.01     |

• This model requires just as many parameters as the full joint distribution!
Computing the Joint Distribution from a Bayes Network

• As usual, we abuse notation:
  \[ P(X_1 = x_1 \land \ldots \land X_n = x_n) \text{ is written as } P(x_1, \ldots, x_n) \]

• \[ P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{Par}(X_i)) \]

• E.g., what’s the probability that an alarm has sounded, there was neither an earthquake nor a burglary, and both John and Mary called?
  \[ P(j \land m \land a \land \neg b \land \neg e) = P(J|a)P(M|a)P(a|\neg b \land \neg e)P(\neg e)P(\neg b) = 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062 \]

Requirements for Constructing a BN

• Recall that the definition of the conditional probability was
  \[ P(x|y) = \frac{P(x \land y)}{P(y)} \]

• and thus we get the chain rule,
  \[ P(x \land y) = P(x|y)P(y) \]

• Generalizing to \( n \) variables,
  \[ P(x_1, \ldots, x_n) = P(x_n|x_{n-1}, \ldots, x_1)P(x_{n-1}, \ldots, x_1) \]

• and repeatedly applying this idea,
  \[ P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|\text{Par}(x_i)) \]

• This “works” just in case we can define a partial order so that \( \text{Par}(X_i) \subseteq \{X_{i-1}, \ldots, X_1\} \)

Topological Interpretations

A node, \( X \), is conditionally independent of its non-descendants, \( Z_i \), given its parents, \( U_i \).

A node, \( X \), is conditionally independent of all other nodes in the network given its Markov blanket: its parents, \( U_i \), children, \( Y_i \), and children’s parents, \( Z_i \).

BN’s can be Compact

• For a network of 40 binary variables, the full joint distribution has \( 2^{40} \) entries (\( > 1,000,000,000,000 \))

• If \( |\text{Par}(x_i)| \leq 5 \), however, then the 40 (conditional) probability tables each have \( \leq 32 \) entries, so the total number of parameters \( \leq 1,280 \)

• Largest medical BN I know (Pathfinder) had 109 variables! \( 2^{109} \approx 10^{16} \)
How Not to Build BN’s

- With the wrong ordering of nodes, the network becomes more complicated, and requires more (and more difficult) conditional probability assessments

Simplifying Conditional Probability Tables

- Do we know any structure in the way that Par(x) “cause” x?
- If each destroyer can sink the ship with probability $P(s|d_i)$, what is the probability that the ship will sink if it’s attacked by both? $\left(1 - P(s|d_1, d_2)\right) = \left(1 - P(s|d_1)\right)\left(1 - P(s|d_2)\right) \left(1 - l\right)$
- For $|\text{Par}(x)| = n$, this requires $O(n)$ parameters, not $O(k^n)$

Exact Solution of BN’s (Burglary example)

- Recall the two basic inference problems: Belief propagation & MAP explanation
- Trivially, we can enumerate all “matching” rows of the joint probability distribution
- For poly-trees (not even undirected loops—i.e., only one connection between any pair of nodes; like our Burglary example), there are efficient linear algorithms, similar to constraint propagation
- For arbitrary BN’s, all inference is NP-hard!
  - Exact solutions
  - Approximation

Notes:
- Sum over all “don’t care” variables
- Factor common terms out of summation
- Calculation becomes a sum of products of sums of products ...
Poly-trees are easy

- Singly-connected structures allow propagation of observations via single paths
- “Down” is just use of conditional probability
- “Up” is just Bayes rule
- Formulated as message propagation rules
- Linear time (network diameter)
- Fails on general networks!

Exact Solution of BN’s (non-poly-trees)

\[
P(A, B, C, D, E) = P(A)P(B|A)P(C|A)P(D|B, C)P(E|C)
\]

- What is the probability of a specific state, say \( A=t, B=f, C=t, D=t, E=t \)?
  \[
p(a, b, c, d, e) = p(a)p(b|a)p(c|a)p(d|b, c)p(e|c)
\]
- What is the probability that \( E=t \) given \( B=t \)?
  \[
p(e|b) = p(e|b)/p(b)
\]
- Consider the term \( P(e,b) \)

\[
P(e, b) = \sum_{A,C,D} P(A, b, C, D, e)
\]

\[
= \sum_{A,C,D} P(A)P(b|A)P(C|A)P(D|b, C)P(e|C)
\]

\[
= \sum_{C} P(e|C) \left( \sum_{A} P(A)P(C|A)P(b|A) \right) \left( \sum_{D} P(D|b, C) \right)
\]

Alas, optimal factoring is NP-hard

Other Exact Methods

- Join-tree: Merge variables into (small!) sets of variables to make graph into a poly-tree. Most commonly-used; aka Clustering, Junction-tree, Potential
- Cutset-conditioning: Instantiate a (small!) set of variables, then solve each residual problem, and add solutions weighted by probabilities of the instantiated variables having those values
- ...
- All these methods are essentially equivalent; with some time-space tradeoffs.

Approximate Inference in BN’s

- Direct Sampling—samples joint distribution
- Rejection Sampling—computes \( P(X|e) \), uses ancestor evidence nodes in sampling
- Likelihood Weighting—like Rejection Sampling, but weights by probability of descendant evidence nodes
- Markov chain Monte Carlo
- Gibbs and other similar sampling methods
Direct Sampling

- From a large number of samples, we can estimate all joint probabilities.
- The probability of an event is the fraction of all complete events generated by PS that match the partially specified event.
- Hence we can compute all conditionals, etc.

```
function Prior-Sample(bn) returns an event sampled from bn
inputs: bn, a Bayes net specifying the joint distribution P(X₁, ..., Xₙ)
x := an event with n elements
for i = 1 to n do
  xᵢ := a random sample from P(Xᵢ|Par(Xᵢ))
return x

lim_{n→∞} \frac{N_{PS}(x₁, ..., xₙ)}{N} = P(x₁, ..., xₙ)
P(x₁, ..., xₙ) \approx \frac{N_{PS}(x₁, ..., xₙ)}{N}
```

Likelihood Weighting

- In trying to compute P(X|e), where e is the evidence (variables with known, observed values).
- Sample only the variables other than those in e.
- Weight each sample by how well it predicts e.

```
S_{WS}(z, e)w(z, e) = \prod_{i=1}^{l} P(zᵢ|Par(Zᵢ)) \prod_{i=1}^{m} P(eᵢ|Par(Eᵢ))
= P(z, e)
```

Rejection Sampling

- Uses PriorSample to estimate the proportion of times each value of X appears in samples that are consistent with e.
- But, most samples may be irrelevant to a specific query, so this is quite inefficient.

```
function Rejection-Sample(X, e, bn, N) returns an estimate of P(X|e)
inputs: bn, a Bayes net
X, the query variable
e, evidence specified as an event
N, the number of samples to be generated
local: K, a vector of counts over values of X, initially 0
for j = 1 to N do
  y := PriorSample(bn)
  if y is consistent with e then
    K[v] := K[v]+1 where v is the value of X in y
return Normalize(K[X])
```

Likelihood Weighting

```
function Likelihood-Weighting(X, e, bn, N) returns an estimate of P(X|e)
inputs: bn, a Bayes net
X, the query variable
e, evidence specified as an event
N, the number of samples to be generated
local: W, a vector of weighted counts over values of X, initially 0
for j = 1 to N do
  y, w := WeightedSample(bn,e)
  if y is consistent with e then
    W[v] := W[v]+w where v is the value of X in y
return Normalize(W[X])
```

```
function Weighted-Sample(bn,e) returns an event and a weight
x := an event with n elements; w := 1
for i = 1 to n do
  if Xᵢ has a value xᵢ in e
    then w := w * P(Xᵢ = xᵢ | Par(Xᵢ))
  else xᵢ := a random sample from P(Xᵢ | Par(Xᵢ))
return x,w
```
Markov chain Monte Carlo

function MCMC(X, e, bn, N) returns an estimate of P(X|e)
local: K[X], a vector of counts over values of X, initially 0
    Z, the non-evidence variables in bn (includes X)
    x, the current state of the network, initially a copy of e
initialize x with random values for the vars in Z
for j = 1 to N do
    for each Zi in Z do
        sample the value of Zi in x from P(Zi|mb(Zi)), given the values of mb(Zi) in x
        K[v] := K[v]+1 where v is the value of X in x
    return Normalize(K[X])

- Wander incrementally from the last state sampled, instead of re-generating a completely new sample
- For every unobserved variable, choose a new value according to its probability given the values of vars in it Markov blanket (remember, it’s independent of all other vars)
- After each change, tally the sample for its value of X; this will only change sometimes
- Problem: “narrow passages”

Most Probable Explanation

- So far, we have been solving for P(X|e), which yields a distribution over all possible values of the x’s
- What it we want the best explanation of a set of evidence, i.e., the highest-probability set of values for the x’s, given e?
- Just maximize over the “don’t care” variables rather than summing
- This is not necessarily the same as just choosing the value of each x with the highest probability

Rules and Probabilities

- Many have wanted to put a probability on assertions and on rules, and compute with likelihoods
- E.g., Mycin’s certainty factor framework
  - A (p=.3) & B (p=.7) => p=.8 => C (p=?)
- Problems:
  - How to combine uncertainties of preconditions and of rule
  - How to combine evidence from multiple rules
- Theorem: There is NO such algebra that works when rules are considered independently.
- Need BN for a consistent model of probabilistic inference

Learning Structure

- In general, we are trying to determine not only parameters for a known structure but in fact which structure is best
- (or the probability of each structure, so we can average over them to make a prediction)
Structure Learning

- Recall that a Bayes Network is fully specified by
  - a DAG $G$ that gives the (in)dependencies among variables
  - the collection of parameters $\theta$ that define the conditional probability tables for each of the $P(x_i | \text{Par}(X_i))$
- Then $P(G|D) = \frac{P(D|G)P(G)}{P(D)} \propto P(D|G)P(G)$
- We define the Bayesian score as $\log P(D|G) + \log P(G)$
- But $P(D|G) = \int_{\theta} P(D|\theta, G)P(\theta|G)P(G)d\theta$
  - First term: usual marginal likelihood calculation
  - Second term: parameter priors
  - Third term: “penalty” for complexity of graph
- Define a search problem over all possible graphs & parameters

Searching for Models

- How many possible DAGs are there for $n$ variables?
  - $< 3^n \approx$ all possible directed graphs on $n$ vars
  - Not all are DAGs
- To get a closer estimate, imagine that we order the variables so that the parents of each var come before it in the ordering. Then
  - there are $n!$ possible ordering, and
  - the $j$-th var can have any of the previous vars as a parent
  - $n! \prod_{i=1}^{n} 2^{!-1} = n! \cdot 2^{\sum_{i=1}^{n} (i-1)} = O(n! \cdot 2^n)$
- If we can choose a particular ordering, say based on prior knowledge, then we need consider “merely” $O(2^n)$ models
- If we restrict $|\text{Par}(X)|$ to no more than $k$, consider $\sum_{i=1}^{n} \binom{n}{k}$ models; this is actually practical
- Search actions: add, delete, reverse an arc
- Hill-climb on $P(D|G)$ or on $P(G|D)$
  - All “usual” tricks in search: simulated annealing, random restart, ...

Caution about Hidden Variables

- Suppose you are given a dataset containing data on patients’ smoking, diet, exercise, chest pain, fatigue, and shortness of breath
- You would probably learn a model like the one below left
- If you can hypothesize a “hidden” variable (not in the data set), e.g., heart disease, the learned network might be much simpler, such as the one below right
- But, there are potentially infinitely many such variables

Re-Learning the ALARM Network from 10,000 Samples

(a) Original Network

(b) Starting Network

(c) Sampled Data

(d) Learned Network