Markov Networks

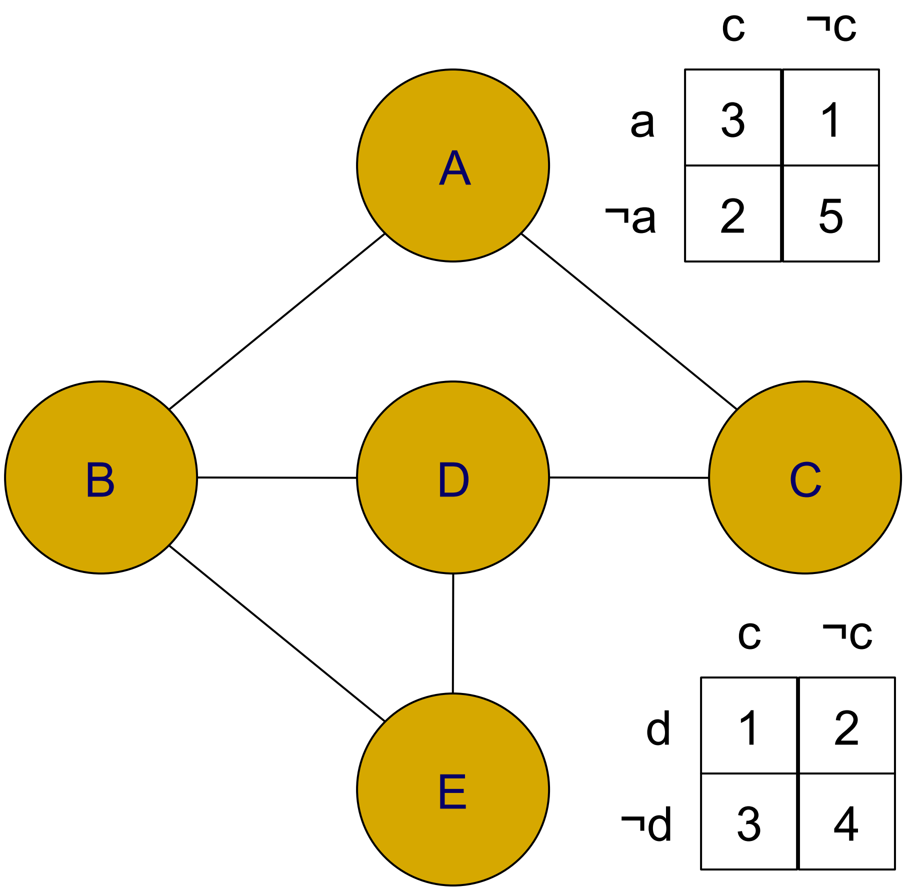
- Like Bayes Nets
  - Graphical model that describes joint probability distribution using tables (AKA potentials)
  - Nodes are random variables
  - Labels are outcomes over the variables

Markov Networks

- Unlike Bayes Nets
    - Undirected graph
    - No requirement that tables need not be are conditional distributions
    - Table distributed over complete subgraph

More on Potentials

- Values are typically non-negative
- Values need not be probabilities
- Generally, one table associated with each clique



	c	¬c
a	3	1
¬a	2	5

	c	¬c
d	1	2
¬d	3	4

Calculating the Full Joint Probability Density

- Full Joint Probability Density is the normalized product of the event probabilities

$$P\left(\vec{V}\right)=\frac{1}{Z}\prod_{k}\phi_k\left(\vec{V}\right)$$

Normalization constant

One potential

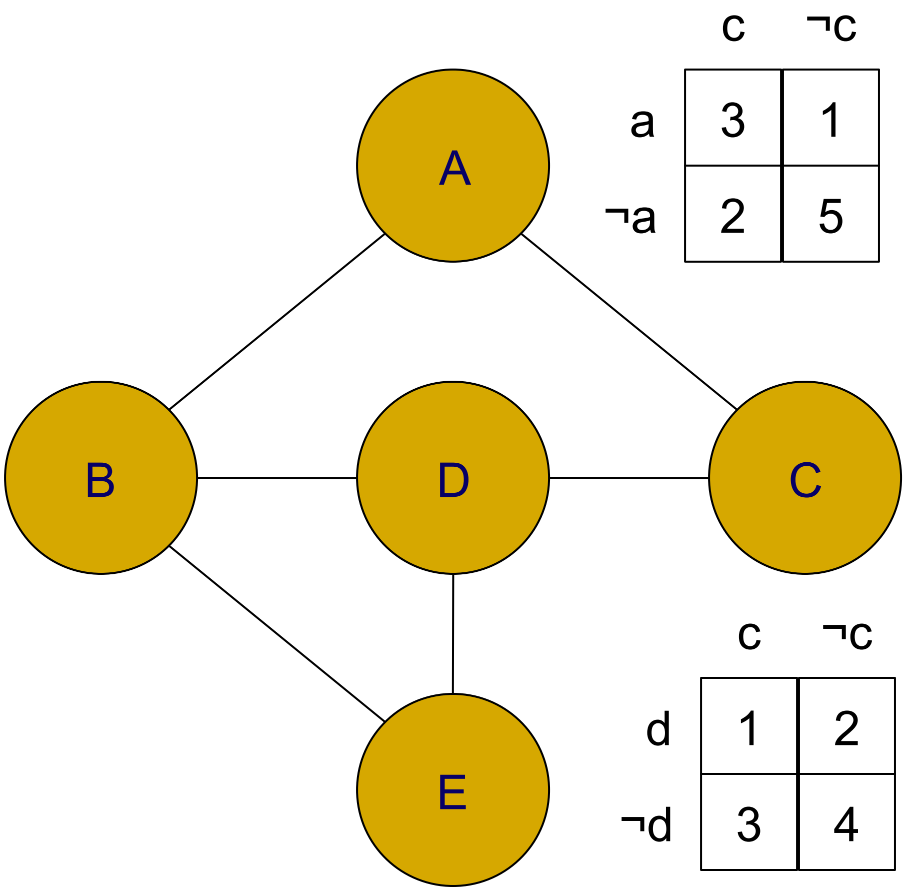
Feature vector (i.e. $\langle A,B,C,D,E\rangle$

Calculating the Normalization Constant Z

$$Z = \sum_{\vec{v} \in \vec{V}} \prod_{k} \phi_k(\vec{v})$$

Using $P(\vec{V}) = \frac{1}{Z} \prod_k \phi_k(\vec{V})$

- Get probability of A=1, B=0, C=1, D=0, E=0
- Only need potentials
- Multiply entries consistent with this setting (3 x 3 = 9)



	c	¬c
a	3	1
¬a	2	5

	c	¬c
d	1	2
¬d	3	4

Hammersley-Clifford Theorem

If Distribution is strictly positive $(P(x) > 0)$

And Graph encodes conditional independences

Then Distribution is product of potentials over cliques of graph

Inverse is also true.

("Markov network = Gibbs distribution")

Markov Nets versus Bayes Nets

- Disadvantages of Markov Nets
  - Computationally intensive to compute probability of any complete setting of variables with Markov Net (NP-hard), easy for Bayes Net
  - Hard to learn Markov Net parameters in a straightforward way
    - Can’t just use marginal frequencies from data as for Bayes nets
    - Gradient ascent requires inference (hard)

Markov Nets versus Bayes Nets

- Advantages of Markov Nets
  - Easier to reason about conditional independence
    - Markov nets are neighbors
    - d-separation: conditional independence achieved iff all paths cut off by evidence
  - No need to select an arbitrary, potentially misleading direction for a dependency in cases where the direction is unclear

Markov Nets vs. Bayes Nets

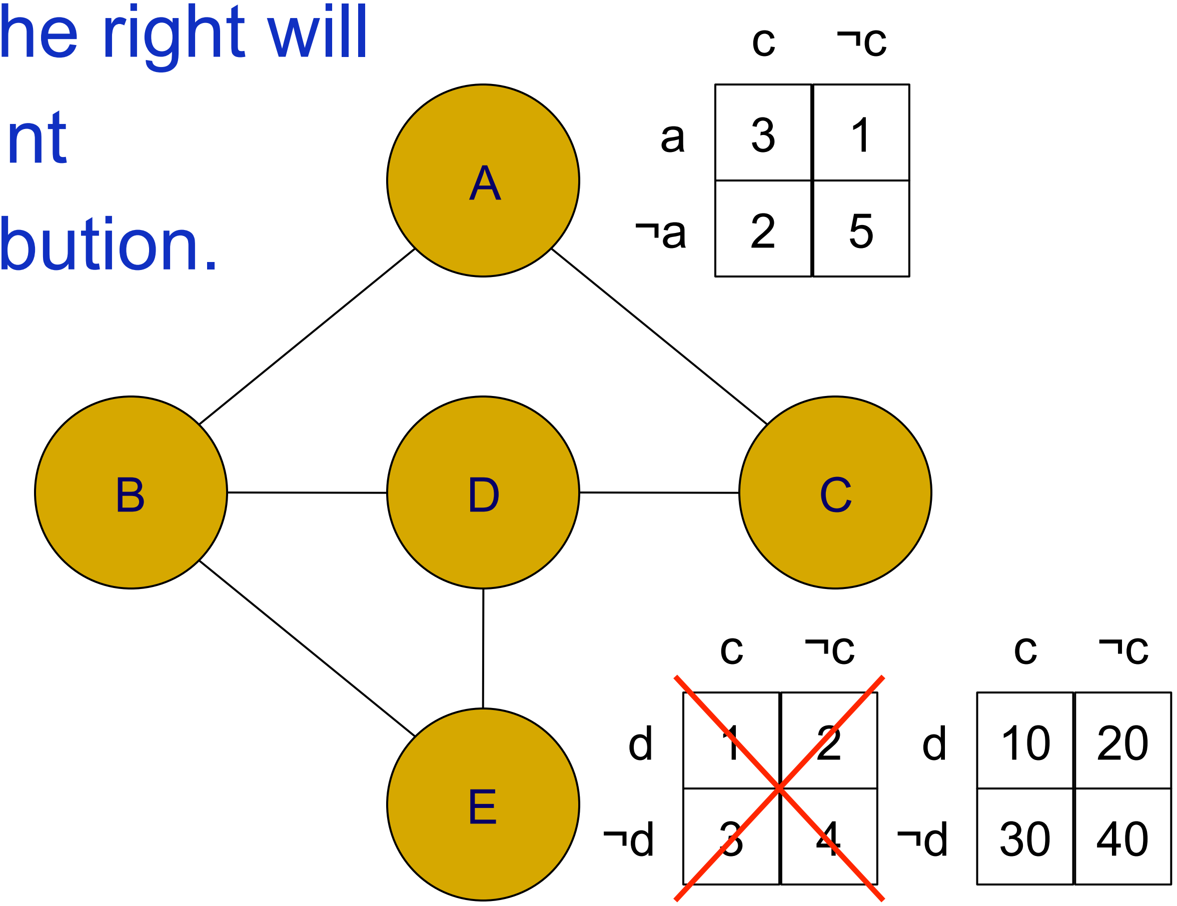
Property	Markov Nets	Bayes Nets
Form	Prod. potentials	Prod. potentials
Potentials	Arbitrary	Cond. probabilities
Cycles	Allowed	Forbidden
Partition func.	Z = ?	Z = 1
Indep. check	Graph separation	D-separation
Indep. props.	Some	Some
Inference	MCMC, BP, etc.	Convert to Markov

Constructing Markov Nets

- Just as in Bayes Nets, the decision of which tables to represent is based on background knowledge
- Although the model can be built from the data, it is often easier for people to leverage domain knowledge
- Although the model is undirected, it can still be helpful to think of directionality when constructing the Markov Net

Scale Invariance

The change at the right will not effect the joint probability distribution.



	c	¬c
a	3	1
¬a	2	5

	c	¬c
d	~~1~~	~~2~~
¬d	~~3~~	~~4~~

	c	¬c
d	10	20
¬d	30	40

Inference

- Almost the same as in Bayes Nets (this is somewhat surprising considering all the other differences!)
- Possible approaches:
    - Gibbs sampling
    - Variable elimination
    - Belief propagation

Inference in Markov Networks

- **Goal**: Compute marginals & conditionals of

$$P(X) = \frac{1}{Z}\exp\left(\sum_i w_i f_i(X)\right) \qquad Z = \sum_X \exp\left(\sum_i w_i f_i(X)\right)$$

- Conditioning on Markov blanket of a proposition *x* is easy, because you only have to consider cliques (formulas) that involve *x* :

$$P(x \mid MB(x)) = \frac{\exp\left(\sum_i w_i f_i(x)\right)}{\exp\left(\sum_i w_i f_i(x=0)\right) + \exp\left(\sum_i w_i f_i(x=1)\right)}$$

- Gibbs sampling exploits this

Markov Chain Monte Carlo

- General algorithm: **Metropolis-Hastings**
  - Sample next state given current one according to transition probability
  - Reject new state with some probability to maintain *detailed balance*
- Simplest (and most popular) algorithm: **Gibbs sampling**
  - Sample one variable at a time given the rest

$$P(x \mid MB(x)) = \frac{\exp\left(\sum_i w_i f_i(x)\right)}{\exp\left(\sum_i w_i f_i(x=0)\right) + \exp\left(\sum_i w_i f_i(x=1)\right)}$$

MCMC: Gibbs Sampling

```
state ← random truth assignment
for i ← 1 to num-samples do
    for each variable x
        sample x according to $P(x|neighbors(x))$
        state ← state with new value of x
$P(F)$ ← fraction of states in which F is true
```

Learning: Recall the Bayes Net approach

- In Bayes Nets, we go through each variable one at a time, row by row in the CPT adjusting weights
- One way to think of this approach is that we look at the prior setting and ask what the probability of this setting is based on what we see in the data, then adjust the CPT to be consistent with the data

Can we use this approach on Markov Nets?

- No! Consider changing a single table value.
  - This changes the partition function, Z.
  - Thus, a local change to one table effects other tables; local changes have global effects!

Markov Net Learning

- We want to get the derivative of the maximum likelihood function. We can then incrementally move each weight in direction of the gradient based on a learning parameter η
- The above approach amounts to differencing the expectation of priors and observed occurrences, computed as on the next slide

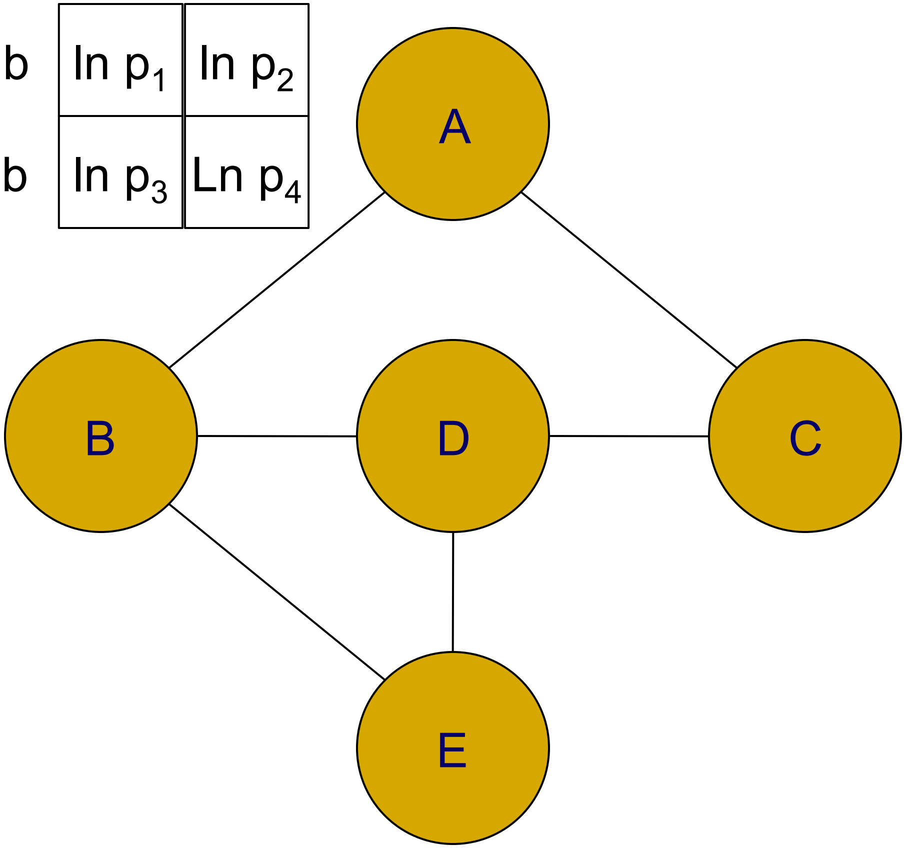
Markov Net Learning, continued

- Assume that the dataset is composed of M datapoints. Consider the task of computing the expectation of priors and observed occurrences for A ∧ B
  - Expectation of priors: M·Pr(A ∧ B)
  - Observed occurrences: Number of datapoints for which A and B hold
- Using this approach, it can be shown that gradient ascent converges

Log Linear Models

- Equivalent to Markov Nets (though they look very different)
- Take the natural log of each parameter

	a	¬a
b	ln p_1	ln p_2
¬b	ln p_3	Ln p_4



Log Linear Models

- This change allows us to write the probability density function as:

exp(X) = e^X

$$\Pr\left(\vec{V}\right) = \frac{1}{Z} \exp \sum_i w_i f_i\left(\vec{V}\right)$$

ln potential values

Logical statements, either 1 or 0
Also known as indicator functions
For example,
$f_1 = a \wedge b$
$f_2 = \neg a \wedge b$

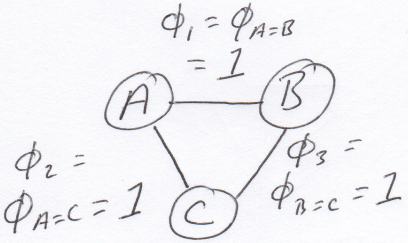
Weight Learning

- Maximize likelihood or posterior probability
- Numerical optimization (gradient or 2nd order)
- No local maxima

$$\frac{\partial}{\partial w_i} \log P_w(x) = \underbrace{\boxed{n_i(x)}}_{\text{No. of times feature } i \text{ is true in data}} - \underbrace{\boxed{E_w\left[n_i(x)\right]}}_{\text{Expected no. times feature } i \text{ is true according to model}}$$

- Requires inference at each step (slow!)

Example: Ising Model



Data

A	B	C
1	1	0
0	1	1
1	1	1
0	0	0
1	0	1

$$Z = \sum_{\vec{v}} e^{\sum_i \phi_i(\vec{v})}$$

$$\simeq 56.2$$

	$\vec{v}$			
A	B	C	$e^{\sum_i \phi_i(\vec{v})}$	$P_{\vec{\phi}}(\vec{v})$
0	0	0	$e^3 \approx 20$	.35
0	0	1	$e \approx 2.7$	.05
0	1	0	.	.
0	1	1	.	.
1	0	0	.	.
1	0	1	.	.
1	1	0	.	.
1	1	1	$e^3 \approx 20$	.35

$\#_{A=B} = 3$

$\#_{B=C} = 3$

$\#_{A=C} = 3$

$$E_{\vec{\Phi}, |D|=5}[\#_{A=B}] = .8(5) = 4$$

same for $\#_{A=C}$, $\#_{B=C}$

Gradient: $\langle -1, -1, -1 \rangle$

New $\vec{\Phi}$: $\langle 1, 1, 1 \rangle + \underset{\text{(e.g., .05)}}{\eta} \langle -1, -1, -1 \rangle$

$= \langle .95, .95, .95 \rangle$

Analyzing

$$\Pr\left(\vec{V}\right) = \frac{1}{Z} \exp \sum_i w_i f_i\left(\vec{V}\right)$$

- In this formulation, the w's are just weights and the f's are just features
- As such, we can throw the graph out if we want – we have everything we need in the w_is and f_is
- In this view, parameter learning is just weight learning

Statistical Relational Learning (SRL)

- For the most part, up until now, we have assumed feature vectors as our data representation
- In many cases, a database model is more likely
- Limitations of ILP
    - ILP that learned rules was somewhat robust to noise, but still used a closed world model
    - There is little that is unconditionally true in the real world
    - SRL addresses these limitations

Markov Logic

- Allows one to make statements that are *usually* true
- Example:

weight	
∞	$\forall x \, \text{smokes}(x) \rightarrow \text{cancer}(x)$
0	$\forall x \forall y \, \text{friends}(x, y) \wedge \text{smokes}(x) \rightarrow \text{smokes}(y)$

Attach weights to each rule. The probability of a setting that violates a rule drops off exponentially with the weight of a rule

Markov Logic, continued

- All variables, need not be universally quantified, but we assume so for here to ease notation
- Rules are mapped into a Markov Network
  - Syntactically we are dealing with predicate calculus (in our example, constants are people)
  - Semantically, we are dealing with a joint probability distribution over all facts (ground atomic formulas)
  - A world is a truth assignment: we have probabilities for each world based on weight

Translating Markov Logic to a Markov Net

- Create ground instances through substitution
- Create a node for each fact
- Create an edge between nodes if both appear in a ground instance

Example translated to Markov Network

- Facts:

$\text{smokes}(\text{Mary})$

$\text{smokes}(\text{Joe})$

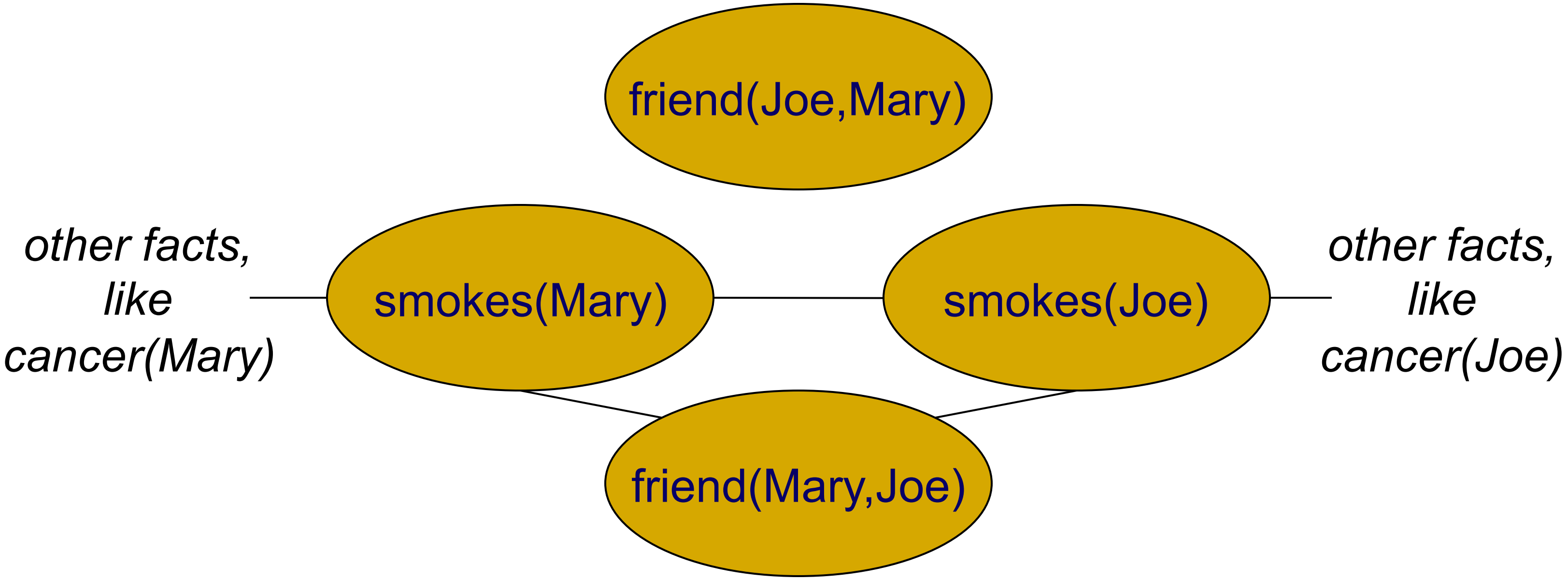
$\text{friend}(\text{Joe}, \text{Mary})$

- Rules:

Friend(x,y) ∧ Smokes(x) -> Smokes(y)

$\forall x \ \text{smokes}(x) \rightarrow \text{cancer}(x)$

$\forall x \forall y \ \text{friends}(x, y) \wedge \text{smokes}(x) \rightarrow \text{smokes}(y)$



Computing weights

- Consider the effect of this rule, called ◊ for convenience:

<u>weight</u>

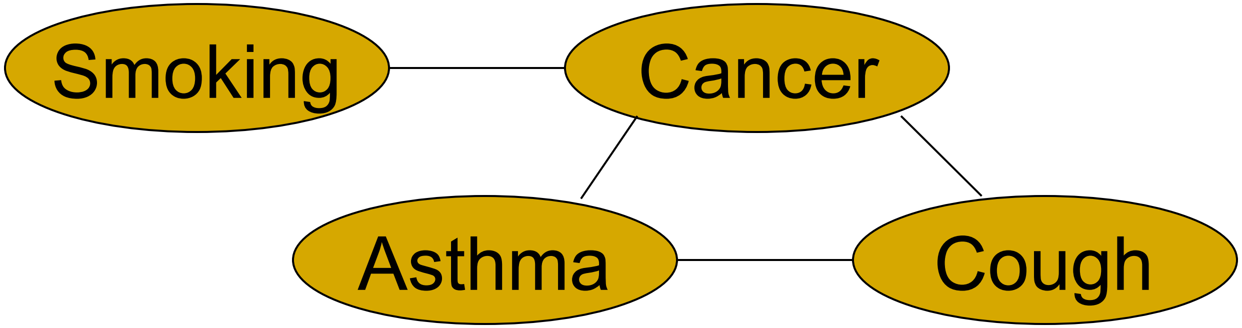
1.1 $\forall x \forall y \; \text{friends}(x, y) \wedge \text{smokes}(x) \rightarrow \text{smokes}(y)$

	smokes(Mary)		¬smokes(Mary)	
	smokes(Joe)	¬smokes(Joe)	smokes(Joe)	¬smokes(Joe)
friends(Mary,Joe)	$e^{1.1}$	$e^{1.1}$	$e^{1.1}$	$e^{1.1}$
¬friends(Mary,Joe)	1	$e^{1.1}$	$e^{1.1}$	$e^{1.1}$

This is the only predicate that does not satisfy ◊
Thus, it is given value 1, while the others are
Given value exp(weight(◊))

Markov Networks

- **Undirected** graphical models



- Potential functions defined over cliques

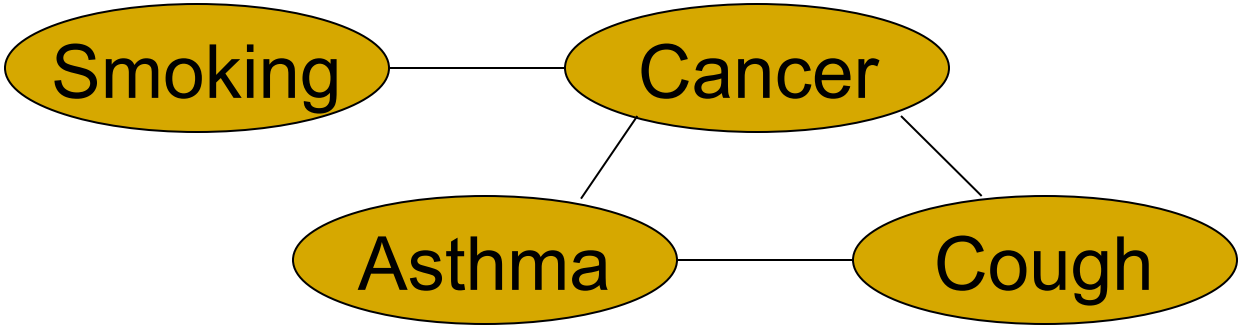
$$P(x) = \frac{1}{Z} \prod_c \Phi_c(x_c)$$

$$Z = \sum_x \prod_c \Phi_c(x_c)$$

Smoking	**Cancer**	**Φ(S,C)**
False	False	4.5
False	True	4.5
True	False	2.7
True	True	4.5

Markov Networks

- **Undirected** graphical models



- Log-linear model:

$$P(x) = \frac{1}{Z} \exp\left(\sum_i \underbrace{w_i}_{\text{Weight of Feature } i} \underbrace{f_i(x)}_{\text{Feature } i} \right)$$

$$f_1(\text{Smoking, Cancer}) = \begin{cases} 1 & \text{if } \neg \text{ Smoking } \vee \text{ Cancer} \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1.5$$

Pseudo-Likelihood

$$PL(x) \equiv \prod_i P(x_i \mid neighbors(x_i))$$

- Likelihood of each variable given its neighbors in the data
- Does not require inference at each step
- Consistent estimator
- Widely used in vision, spatial statistics, etc.
- But PL parameters may not work well for long inference chains

Structure Learning

- Start with atomic features
- Greedily conjoin features to improve score
- Problem: Need to reestimate weights for each new candidate
- Approximation: Keep weights of previous features constant

Generative Weight Learning

- Maximize likelihood or posterior probability
- Numerical optimization (gradient or 2nd order)
- No local maxima

$$\frac{\partial}{\partial w_i}\log P_w(x) = \boxed{n_i(x)} - \boxed{E_w\left[n_i(x)\right]}$$

No. of times feature *i* is true in data

Expected no. times feature *i* is true according to model

- Requires inference at each step (slow!)

Pseudo-Likelihood

$$PL(x) \equiv \prod_i P(x_i \mid neighbors(x_i))$$

- Likelihood of each variable given its neighbors in the data
- Does not require inference at each step
- Consistent estimator
- Widely used in vision, spatial statistics, etc.
- But PL parameters may not work well for long inference chains

Discriminative Weight Learning

- Maximize conditional likelihood of query (y) given evidence (x)

$$\frac{\partial}{\partial w_i} \log P_w(y \mid x) = \boxed{n_i(x,y)} - \boxed{E_w\left[n_i(x,y)\right]}$$

No. of true groundings of clause *i* in data

Expected no. true groundings according to model

- Approximate expected counts by counts in MAP state of y given x

Rule Induction

- **Given:** Set of positive and negative examples of some concept
  - **Example:** $(x_1, x_2, \dots, x_n, y)$
  - y: **concept** (Boolean)
  - $x_1, x_2, \dots, x_n$: **attributes** (assume Boolean)
- **Goal:** Induce a set of rules that cover all positive examples and no negative ones
  - **Rule:** $x_a \wedge x_b \wedge \dots$ **->** y (x_a: Literal, i.e., x_i or its negation)
  - Same as **Definite clause**: *Body* ⇒ *Head*
  - Rule *r* **covers** example *x* iff *x* satisfies body of *r*
- ***Eval(r):*** Accuracy, info. gain, coverage, support, etc.

Learning a Single Rule

```
$head \leftarrow y$
$body \leftarrow Ø$
repeat
    for each literal $x$
        $r_x \leftarrow r$ with $x$ added to $body$
        $Eval(r_x)$
    $body \leftarrow body$ ^ best $x$
until no $x$ improves $Eval(r)$
return $r$
```

Learning a Set of Rules

```
$R \leftarrow Ø$
$S \leftarrow examples$
**repeat**
    learn a single rule $r$
     $R \leftarrow R$ U $\{ r \}$
    $S \leftarrow$ S − positive examples covered by r
**until** S contains no positive examples
**return** $R$
```

First-Order Rule Induction

- *y* and x_i are now predicates with arguments
  E.g.: *y* is Ancestor(x,y), x_i is Parent(x,y)
- Literals to add are predicates or their negations
- Literal to add must include at least one variable already appearing in rule
- Adding a literal changes # groundings of rule
  E.g.: Ancestor(x,z) ^ Parent(z,y) **->** Ancestor(x,y)
- *Eval(r)* must take this into account
  E.g.: Multiply by # positive groundings of rule
  still covered after adding literal