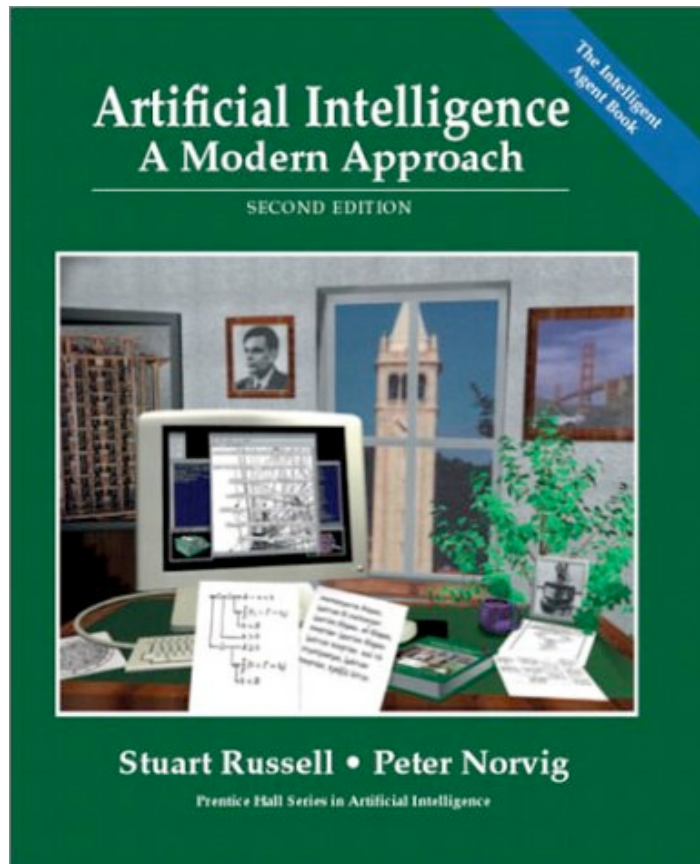


# Bayesian Learning and Learning Bayesian Networks



Chapter 20  
Slides by  
Cristina Conati

# Overview

- Full Bayesian Learning
- MAP learning
- Maximum Likelihood Learning
- Learning Bayesian Networks
  - Fully observable
  - With hidden (unobservable) variables

# Full Bayesian Learning

- In the learning methods we have seen so far, the idea was always to find the best model that could explain some observations
- In contrast, full Bayesian learning sees learning as Bayesian updating of a probability distribution over the hypothesis space, given data
  - $H$  is the hypothesis variable
  - Possible hypotheses (values of  $H$ )  $h_1, \dots, h_n$
  - $P(H)$  = prior probability distribution over hypothesis space
- $j_{th}$  observation  $d_j$  gives the outcome of random variable  $D_j$ 
  - training data  $\mathbf{d} = d_1, \dots, d_k$

# Full Bayesian Learning

➤ Given the data so far, each hypothesis  $h_i$  has a posterior probability:

- $P(h_i | \mathbf{d}) = \alpha P(\mathbf{d} | h_i) P(h_i)$  (**Bayes theorem**)
- where  $P(\mathbf{d} | h_i)$  is called the *likelihood* of the data under each hypothesis

➤ Predictions over a new entity  $X$  are a weighted average over the prediction of each hypothesis:

- $P(X | \mathbf{d}) =$   
 $= \sum_i P(X, h_i | \mathbf{d})$   
 $= \sum_i P(X | h_i, \mathbf{d}) P(h_i | \mathbf{d})$   
 $= \sum_i P(X | h_i) P(h_i | \mathbf{d})$   
 $\sim \sum_i P(X | h_i) P(\mathbf{d} | h_i) P(h_i)$

The data does not add anything to a prediction given an  $hp$

- The weights are given by the data likelihood and prior of each  $h$

➤ No need to pick one best-guess hypothesis!

# Example

Suppose there are five kinds of bags of candies:

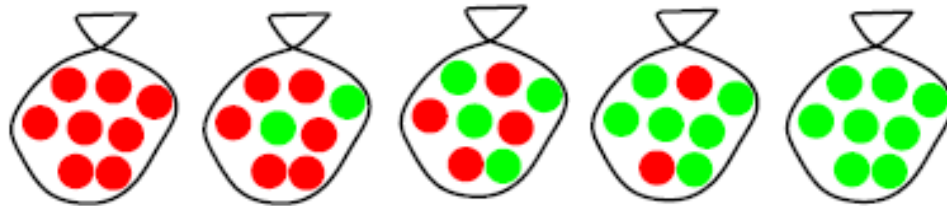
10% are  $h_{100}$  100% cherry candies

20% are  $h_{75}$  75% cherry candies + 25% lime candies

40% are  $h_{50}$  50% cherry candies + 50% lime candies

20% are  $h_{25}$  25% cherry candies + 75% lime candies

10% are  $h_0$  100% lime candies



Then we observe candies drawn from some bag: ● ● ● ● ● ● ● ● ● ●

- Let's call  $\theta$  the parameter that defines the fraction of cherry candy in a bag, and  $h_\theta$  the corresponding hypothesis
- Which one of the five kinds of bag has generated my 10 observations?  $P(h_\theta | \mathbf{d})$ .
- What flavour will the next candy be? Prediction

# Example

➤ If we re-wrap each candy and return it to the bag, our 10 observations are independent and identically distributed, *i.i.d.*, so

- $P(\mathbf{d} | h_\theta) = \prod_j P(d_j | h_\theta)$  for  $j=1, \dots, 10$

➤ For a given  $h_\theta$ , the value of  $P(d_j | h_\theta)$  is

- $P(d_j = \text{cherry} | h_\theta) = \theta$ ;  $P(d_j = \text{lime} | h_\theta) = (1 - \theta)$

➤ And given  $N$  observations, of which  $c$  are cherry and  $l = N - c$  lime

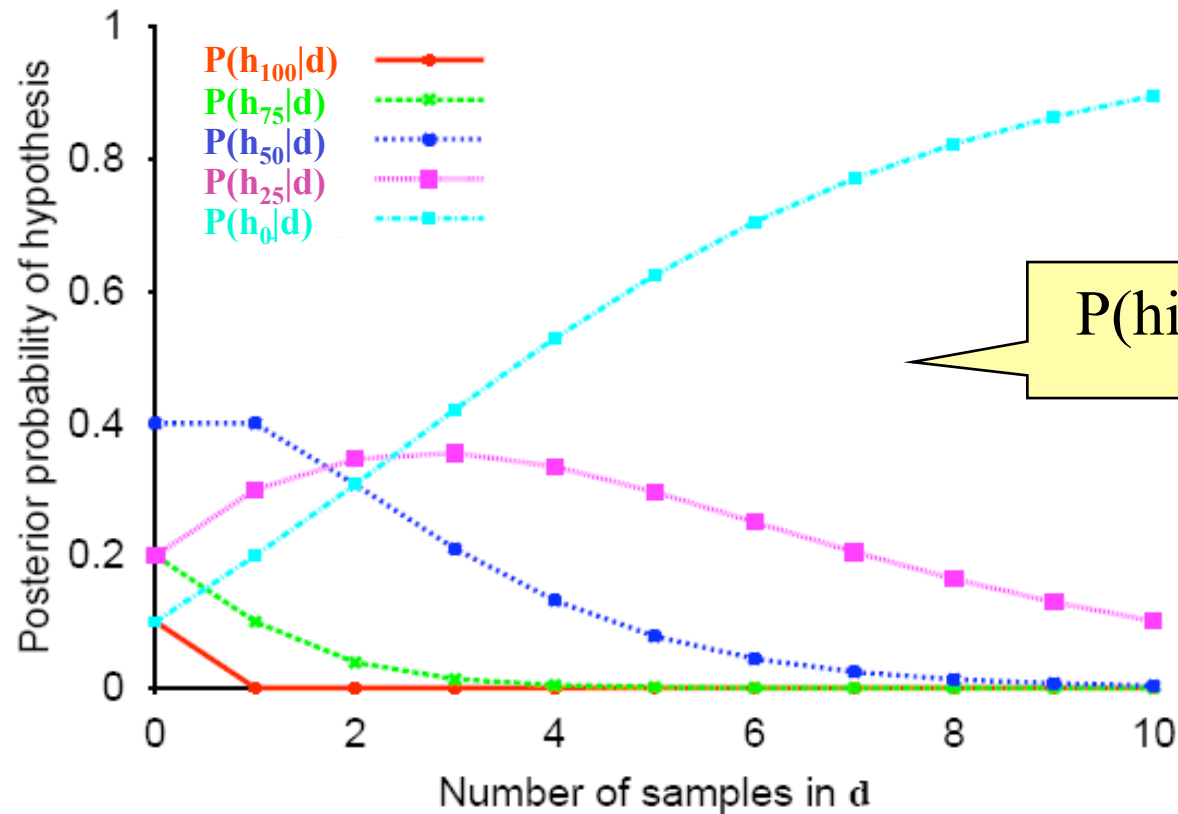
$$P(\mathbf{d} | h_\theta) = \prod_{j=1}^c \theta \prod_{j=1}^l (1 - \theta) = \theta^c (1 - \theta)^l$$

- **Binomial distribution**: probability of # of successes in a sequence of  $N$  independent trials with binary outcome, each of which yields success with probability  $\theta$ .

➤ For instance, after observing 3 lime candies in a row:

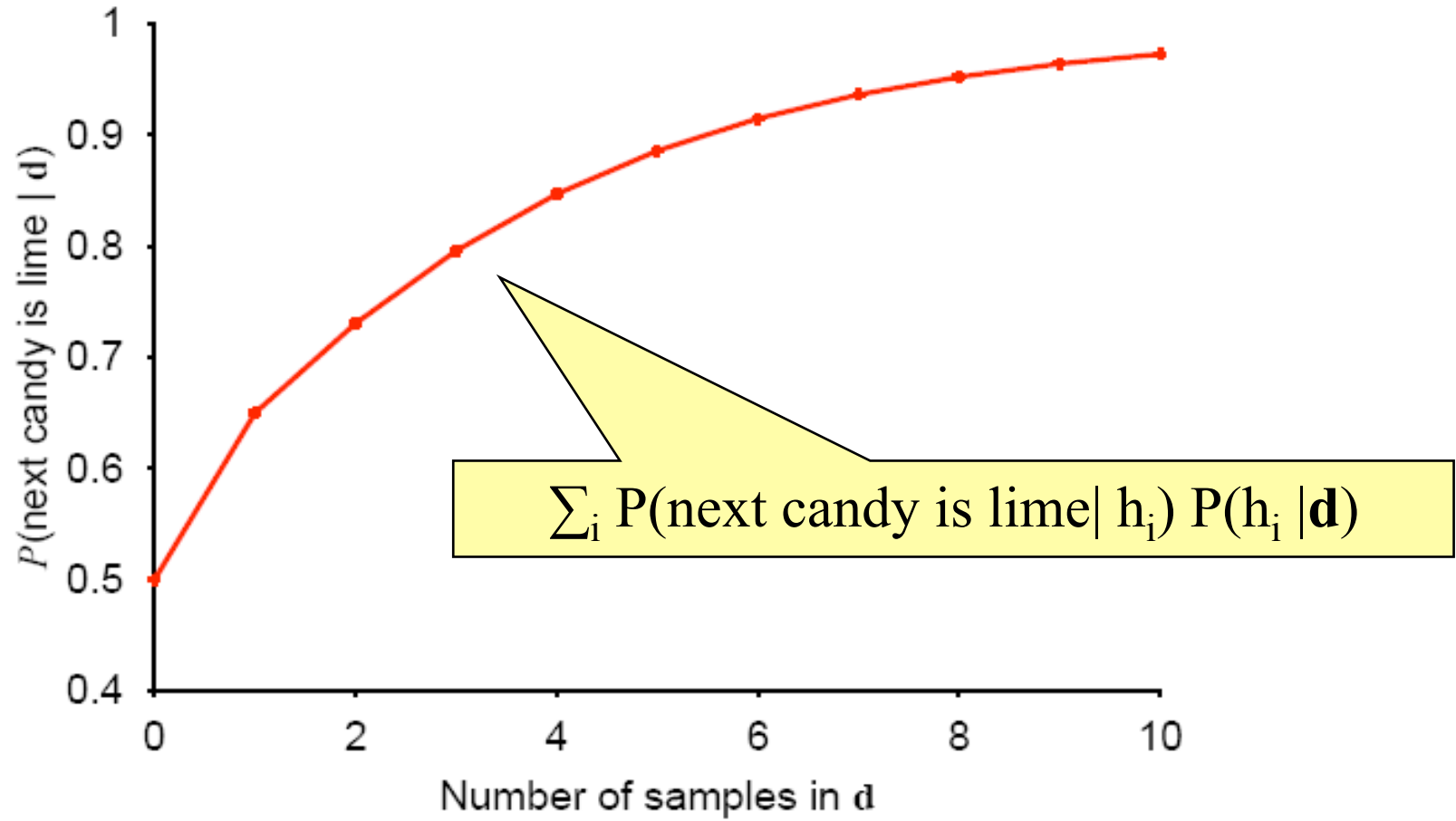
- $P([\text{lime}, \text{lime}, \text{lime}] | h_{.5}) = 0.5^3$  because the probability of seeing lime for each observation is 0.5 under this hypotheses

# Posterior Probability of H



- Initially, the  $h_p$  with higher priors dominate ( $h_{50}$  with prior = 0.4)
- As data comes in, the true hypothesis ( $h_0$ ) starts dominating, as the probability of seeing this data given the other hypotheses gets increasingly smaller
  - After seeing three lime candies in a row, the probability that the bag is the all-lime one starts taking off

# Prediction Probability



- The probability that the next candy is lime increases with the probability that the bag is an all-lime one

# Overview

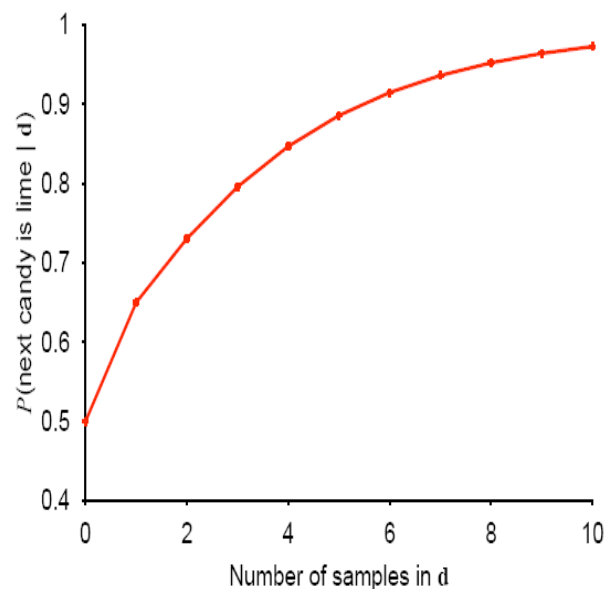
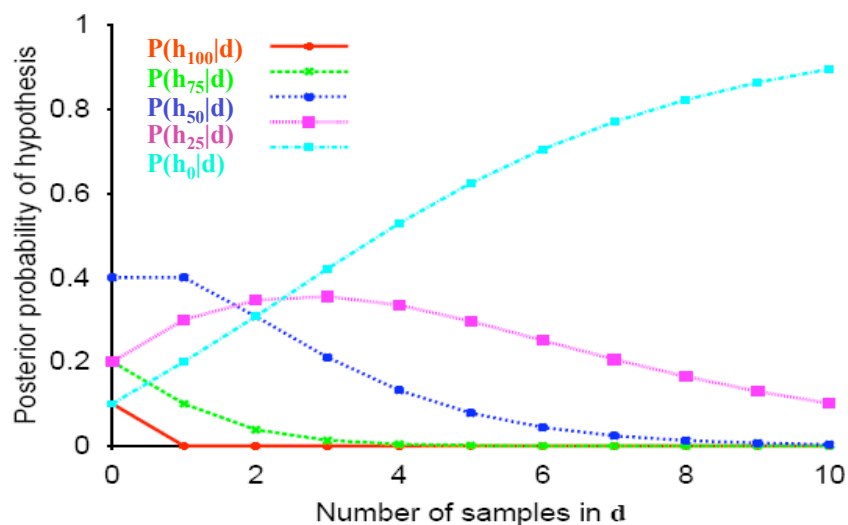
- Full Bayesian Learning
- MAP learning
- Maximum Likelihood Learning
- Learning Bayesian Networks
  - Fully observable
  - With hidden (unobservable) variables

# MAP approximation

- Full Bayesian learning seems like a very safe bet, but unfortunately it does not work well in practice
  - Summing over the hypothesis space is often intractable (e.g., 18,446,744,073,709,551,616 Boolean functions of 6 attributes)
- Very common approximation: Maximum a posteriori (MAP) learning:
  - Instead of doing prediction by considering all possible hp, as in
    - ✓  $P(X|\mathbf{d}) = \sum_i P(X|h_i) P(h_i|\mathbf{d})$
  - choose  $h_{MAP}$  that maximises  $P(h_i|\mathbf{d})$ 
    - ✓ I.e., maximize  $P(\mathbf{d}|h_i) P(h_i)$  or  $\log P(\mathbf{d}|h_i) + \log P(h_i)$

# MAP approximation

- Map is a good approximation when  $P(X | \mathbf{d}) \approx P(X | h_{\text{MAP}})$ 
  - In our example,  $h_{\text{MAP}}$  is the all-lime bag after only 3 candies, predicting that the next candy will be lime with  $p = 1$
  - the bayesian learner gave a prediction of 0.8, safer after seeing only 3 candies



# Bias

- As more data arrive, MAP and Bayesian prediction become closer, as MAP's competing hypotheses become less likely
- Often easier to find MAP (optimization problem) than deal with a large summation problem
- $P(H)$  plays an important role in both MAP and Full Bayesian Learning
  - Defines the *learning bias*, i.e. which hypotheses are favoured
- Used to define a tradeoff between model complexity and its ability to fit the data
  - More complex models can explain the data better => higher  $P(\mathbf{d} | h_i)$  **danger of overfitting**
  - But they are less likely a priori because there are more of them than simpler model => lower  $P(h_i)$
  - I.e. common learning bias is to penalize complexity

# Overview

- Full Bayesian Learning
- MAP learning
- Maximun Likelihood Learning
- Learning Bayesian Networks
  - Fully observable
  - With hidden (unobservable) variables

# Maximum Likelihood (ML) Learning

- Further simplification over Full Bayesian and MAP learning
  - Assume uniform priors over the space of hypotheses
  - MAP learning (maximize  $P(\mathbf{d} | h_i) P(h_i)$ ) reduces to maximize  $P(\mathbf{d} | h_i)$
- When is ML appropriate?

# Maximum Likelihood (ML) Learning

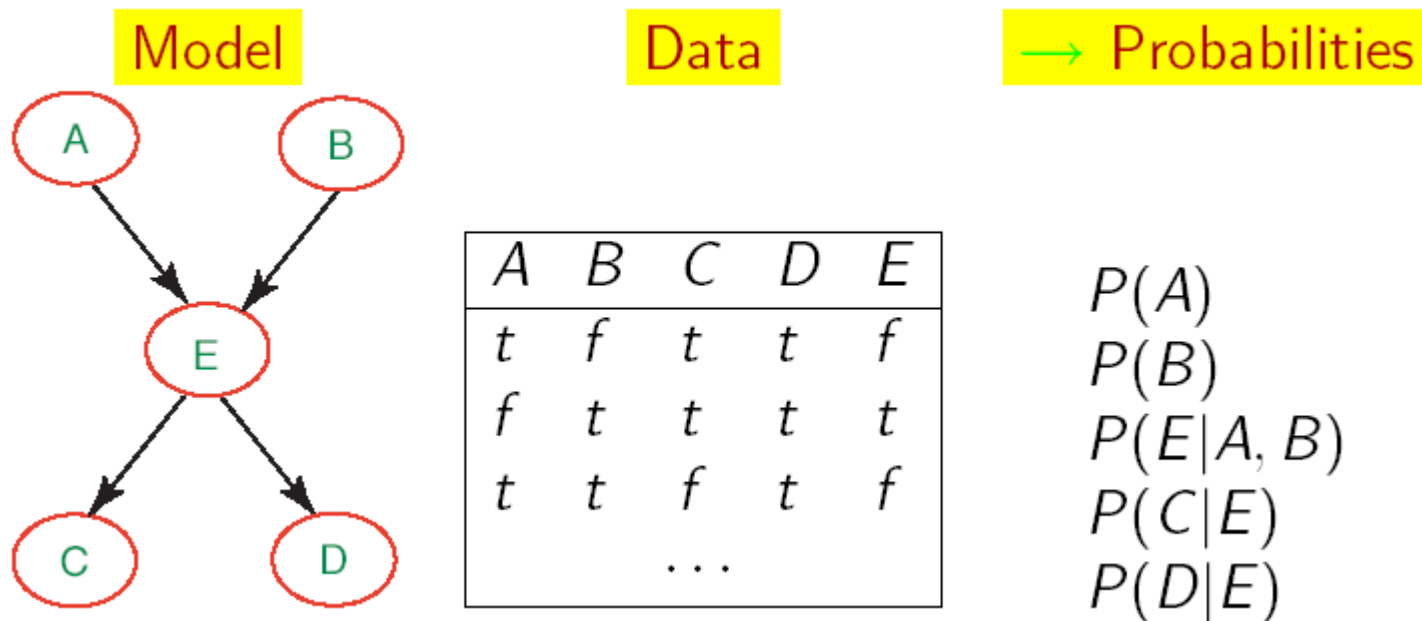
- Further simplification over Full Bayesian and MAP learning
  - Assume uniform prior over the space of hypotheses
  - MAP learning (maximize  $P(\mathbf{d} | h_i) P(h_i)$ ) reduces to maximize  $P(\mathbf{d} | h_i)$
- When is ML appropriate?
  - Used in statistics as the standard (non-bayesian) statistical learning method by those distrust subjective nature of hypotheses priors
  - When the competing hypotheses are indeed equally likely (e.g. have same complexity)
  - With very large datasets, for which  $P(\mathbf{d} | h_i)$  tends to overcome the influence of  $P(h_i)$

# Overview

- Full Bayesian Learning
- MAP learning
- Maximum Likelihood Learning
- Learning Bayesian Networks
  - Fully observable (complete data)
  - With hidden (unobservable) variables

# Learning BNets: Complete Data

- We will start by applying ML to the simplest type of BNets learning:
  - known structure
  - Data containing observations for all variables
    - ✓ All variables are observable, no missing data
- The only thing that I need to learn are the network's parameters



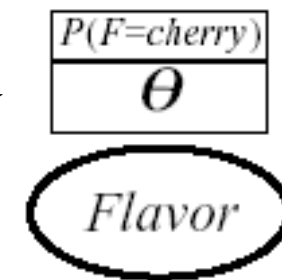
# ML learning: example

## ➤ Back to the candy example:

- New candy manufacturer that does not provide data on the probability of different types of bags, i.e. fraction  $\theta$  of cherry candy
- Any  $\theta$  is possible: continuum of hypotheses  $h_\theta$
- Reasonable to assume that all  $\theta$  are equally likely (we have no evidence of the contrary): uniform distribution  $P(h_\theta)$
- $\theta$  is a parameter for this simple family of models, that we need to learn

## ➤ Simple network to represent this problem

- *Flavor* represents the event of drawing a cherry vs. lime candy from the bag
- $P(F=cherry)$ , or  $P(cherry)$  for brevity, is equivalent to the fraction  $\theta$  of cherry candies in the bag



## ➤ We want to infer $\theta$ by unwrapping N candies from the bag

## ML learning: example (cont'd)

- Unwrap  $N$  candies,  $c$  cherries and  $l = N - c$  lime (and return each candy in the bag after observing flavor)
- As we saw earlier, this is described by a binomial distribution
  - $P(\mathbf{d} | h_\theta) = \prod_j P(d_j | h_\theta) = \theta^c (1 - \theta)^l$
- With ML we want to find  $\theta$  that maximizes this expression, or equivalently its *log likelihood* ( $L$ )
  - $L(P(\mathbf{d} | h_\theta))$ 
    - $= \log(\prod_j P(d_j | h_\theta))$
    - $= \log(\theta^c (1 - \theta)^l)$
    - $= c \log \theta + l \log(1 - \theta)$

## ML learning: example (cont'd)

- To maximise, we differentiate  $L(P(\mathbf{d} | h_\theta))$  with respect to  $\theta$  and set the result to 0

$$\begin{aligned} & \frac{\partial(c \log \theta + \ell \log(1 - \theta))}{\partial \theta} \\ &= \frac{c}{\theta} - \frac{\ell}{1 - \theta} \\ &= \frac{c}{\theta} - \frac{N - c}{1 - \theta} = 0 \end{aligned}$$

- Doing the math gives

$$\theta = \frac{c}{N}$$

# Frequencies as Priors

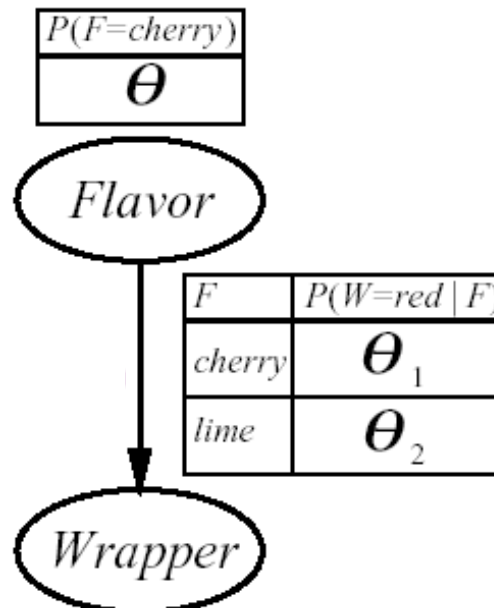
- So this says that the proportion of cherries in the bag is equal to the proportion (frequency) of in cherries in the data
- Now we have justified why this approach provides a reasonable estimate of node priors

# General ML procedure

- Express the likelihood of the data as a function of the parameters to be learned
- Take the derivative of the log likelihood with respect of each parameter
- Find the parameter value that makes the derivative equal to 0
- The last step can be computationally very expensive in real-world learning tasks

## Another example

- Let's apply the procedure to learn the conditional probabilities in a network
- The manufacturer chooses the wrapper for each candy based on flavor, following on an unknown distribution
- The network for this problem includes 3 parameters to be learned
  - $\theta \theta_1 \theta_2$



## Another example (cont'd)

➤  $P(W=\text{green}, F = \text{cherry} | h_{\theta_1\theta_2}) = (*)$

WHY?

$$= P(W=\text{green} | F = \text{cherry}, h_{\theta_1\theta_2}) P(F = \text{cherry} | h_{\theta_1\theta_2})$$

$$= \theta (1 - \theta_1)$$

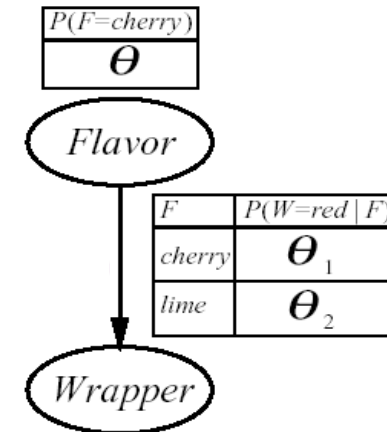
➤ We unwrap  $N$  candies

- $c$  are cherry and  $l$  are lime
- $r^c$  cherry with red wrapper,  $g^c$  cherry with green wrapper
- $r^l$  lime with red wrapper,  $g^l$  lime with green wrapper
- every trial is a combination of wrapper and candy flavor similar to event (\*) above, so

➤  $P(\mathbf{d} | h_{\theta_1\theta_2})$

$$= \prod_j P(d_j | h_{\theta_1\theta_2})$$

$$= \theta^c (1 - \theta)^l (\theta_1)^{r^c} (1 - \theta_1)^{g^c} (\theta_2)^{r^l} (1 - \theta_2)^{g^l}$$



## Another example (cont'd)

➤ I want to maximize the log of this expression

- $c \log \theta + l \log(1 - \theta) + r^c \log \theta_1 + g^c \log(1 - \theta_1) + r^l \log \theta_2 + g^l \log(1 - \theta_2)$

➤ Take derivative with respect of each of  $\theta, \theta_1, \theta_2$

- The terms not containing the derivation variable disappear

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{l}{1 - \theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c + l}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1 - \theta_1} = 0 \quad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c + g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_l}{\theta_2} - \frac{g_l}{1 - \theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{r_l}{r_l + g_l}$$

# ML parameter learning in Bayes nets

- Frequencies again!
- This process generalizes to every fully observable Bnet.
- With complete data and ML approach:
  - Parameters learning decomposes into a separate learning problem for each parameter (CPT), because of the log likelihood step
  - Each parameter is given by the frequency of the desired child value given the relevant parents values

# Problem with ML parameter learning

- With small datasets, some of the frequencies may be 0 just because I have not observed the relevant data
- Generates very strong incorrect predictions:
  - Common fix: initialize the count of every relevant event to 1 before counting the observations

# Probability from Experts

- As we mentioned in previous lectures, an alternative to learning probabilities from data is to get them from experts
- Problems
  - Experts may be reluctant to commit to specific probabilities that cannot be refined
  - How to represent the confidence in a given estimate
  - Getting the experts and their time in the first place
- One promising approach is to *leverage both sources* when they are available
  - Get initial estimates from experts
  - Refine them with data

# Combining Experts and Data

- Get the expert to express her belief on event  $A$  as the pair

$\langle n, m \rangle$

i.e. how many observations of  $A$  they have seen (or expect to see) in  $m$  trials

- Combine the pair with actual data

- If  $A$  is observed, increment both  $n$  and  $m$
- If  $\neg A$  is observed, increment  $m$  alone

- The absolute values in the pair can be used to express the expert's level of confidence in her estimate

- Small values (e.g.,  $\langle 2, 3 \rangle$ ) represent low confidence
- The larger the values, the higher the confidence
- Pair that represents complete ignorance?



WHY?

# Combining Experts and Data

- Get the expert to express her belief on event  $A$  as the pair

$$\langle n, m \rangle$$

i.e. how many observations of  $A$  they have seen (or expect to see) in  $m$  trials

- Combine the pair with actual data

- If  $A$  is observed, increment both  $n$  and  $m$
- If  $\neg A$  is observed, increment  $m$  alone

- The absolute values in the pair can be used to express the expert's level of confidence in her estimate

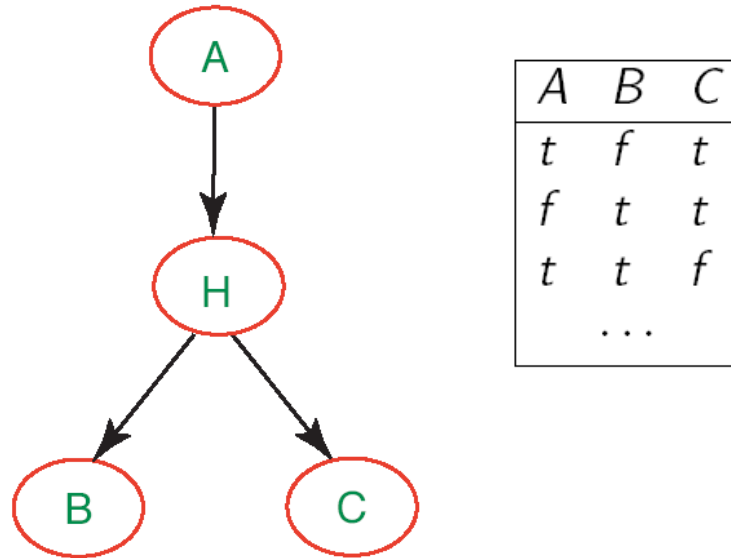
- Small values (e.g.,  $\langle 2, 3 \rangle$ ) represent low confidence, as they are quickly dominated by data
- The larger the values, the higher the confidence as it takes more and more data to dominate the initial estimate (e.g.  $\langle 2000, 3000 \rangle$ )
- Pair that represents complete ignorance?

# Overview

- Full Bayesian Learning
- MAP learning
- Maximum Likelihood Learning
- Learning Bayesian Networks
  - Fully observable (complete data)
  - With hidden (unobservable) variables

# Learning Parameters with Hidden Variables

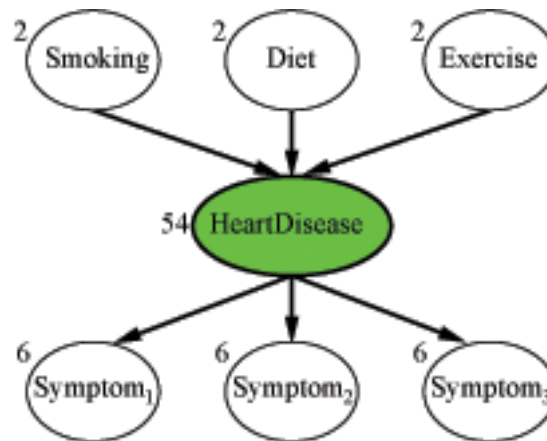
- So far we have assumed that I can collect data on all variables in the network
- What if this is not true, i.e. the network has *hidden variables*?



- Clearly I can't use the frequency approach, because I am missing all the counts involving H

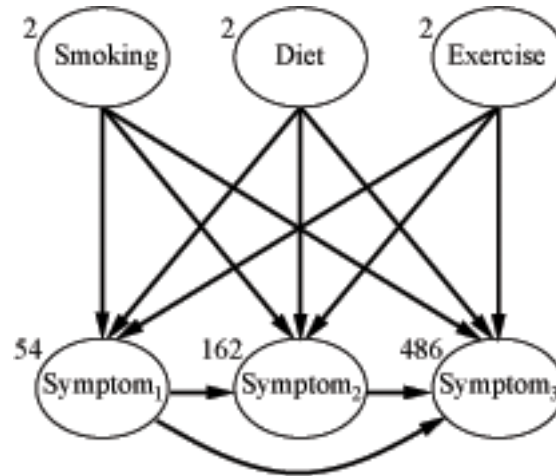
# Quick Fix

- Get rid of the hidden variables.
- It may work in the simple network given earlier, but what about the following one?



- Each variable has 3 values (low, moderate, high)
- the numbers by the nodes represent how many parameters need to be specified for the CPT of that node
- 78 probabilities to be specified overall

# Not Necessarily a Good Fix

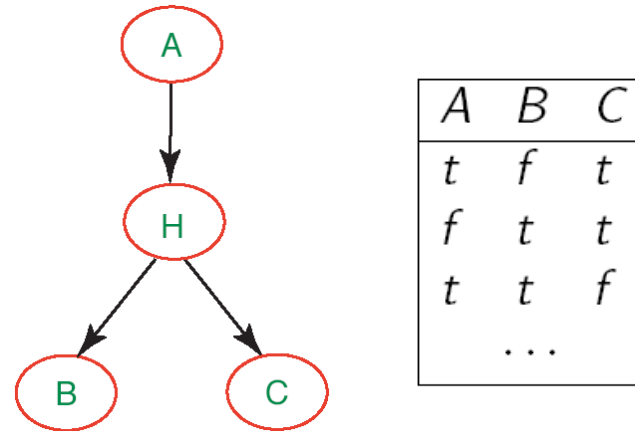


- The symptom variables are no longer conditionally independent given their parents
  - Many more links, and many more probabilities to be specified: 708 overall
  - Need much more data to properly learn the network

# Expectation-Maximization (EM)

- If we keep the hidden variables, and want to learn the network parameters from data, we have a form of *unsupervised learning*
  - the data do not include information on the true nature of each data point
- Expectation-Maximization
  - general algorithm for learning model parameters from incomplete data
  - We'll see how it works on learning parameters for Bnets with discrete, continuous variables

# EM: general idea



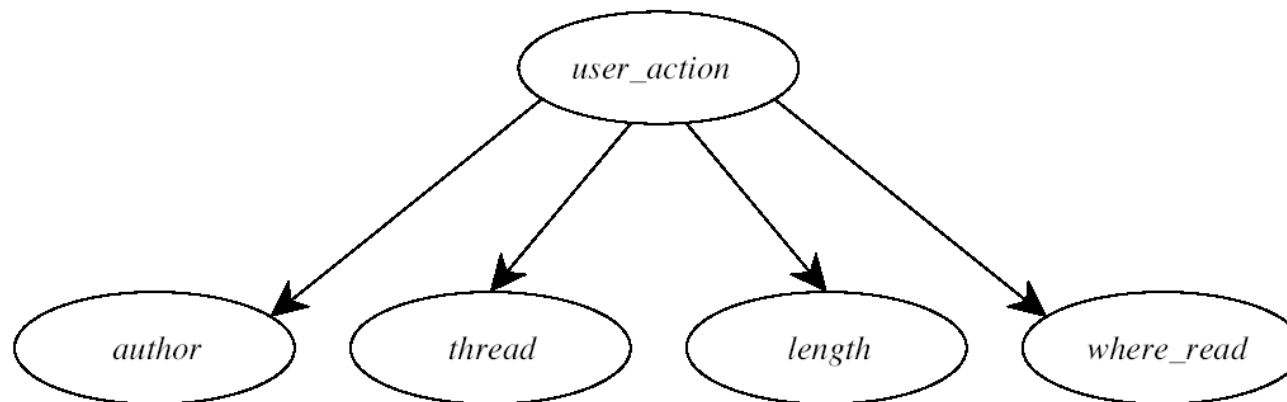
- If we had data for all the variables in the network, we could learn the parameters by using ML (or MAP) models
  - e.g. frequencies of the relevant events as we saw in previous examples
- If we had the parameters in the network, we could estimate the posterior probability of any event, including the hidden variables
  - e.g.,  $P(H|A,B,C)$

# EM: General Idea

- The algorithm starts from “invented” (e.g., randomly generated) information to solve the learning problem, e.g.,
  - the network parameters *or augmented* data to derive them
- It then refines this initial guess by cycling through two basic steps
  - Expectation (E): update the data with predictions generated via the current model
  - Maximization (M): given the updated data, infer the Maximum Likelihood (or MAP model)
    - ✓ This is the same step that we described when learning parameters for fully observable networks
- It can be shown that EM increases the log likelihood of the data at any iteration, and often achieves a local maximum

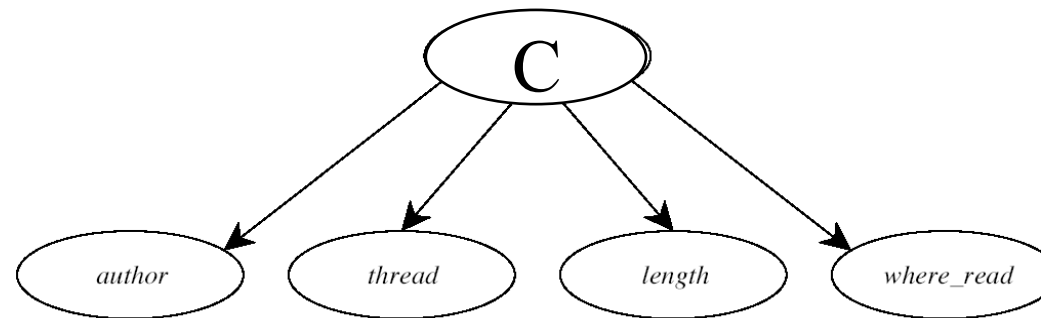
# Example 1: Bayes Classifier

- Very simple Bayesian Network that allows to classify entities in a set of classes  $C$ , given a set of attributes
  - The value of each attribute depends on the classification
  - The attributes are independent of each other given the classification
- Naïve Classifier for the newsgroup reading example



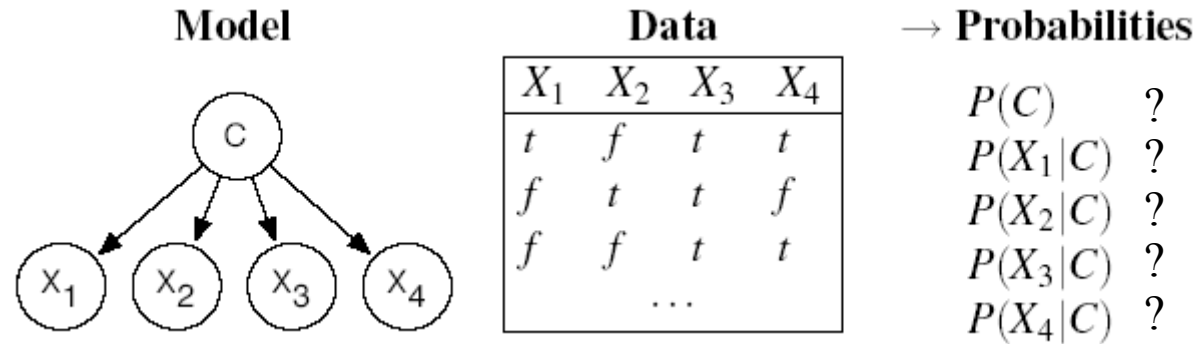
# Example 1: Bayes Classifier

- In the newsgroup example the data included the classification
  - I can use the frequencies to learn the necessary CPTs
- But I may want to find groups of users that behave similarly when reading news, without knowing the categories a priori
  - Type of unsupervised learning known as *clustering*
- I can do that by assuming  $k$  user categories and making them the values of a hidden variable  $C$  in the Naive classifier
  - Need EM to compute the parameters



# EM: How it Works on Naive Bayes

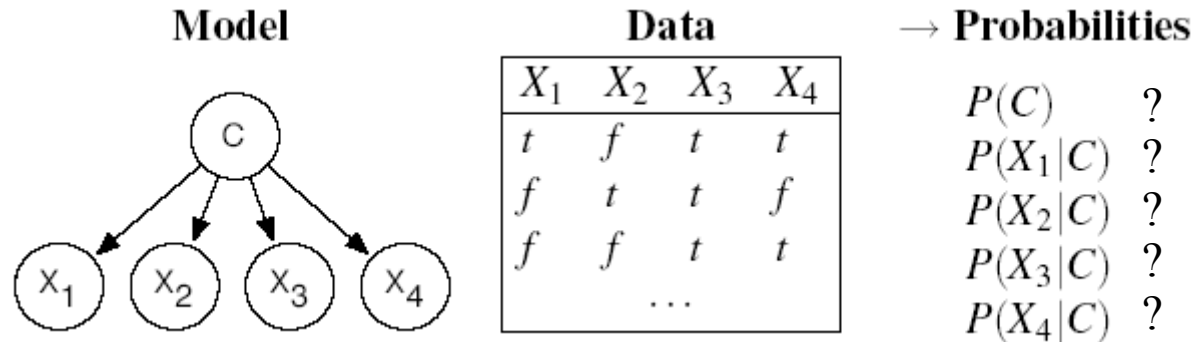
- We start from our network with hidden variable and incomplete data



- We then make up the missing data.
  - Suppose that we want  $C$  to have three values,  $[1,2,3]$  (categories)
- What would we need to learn the network parameters?

# EM: How it Works on Naive Bayes

- We start from our network with hidden variable and incomplete data



- We then make up the missing data.
  - Suppose that we want  $C$  to have three values,  $[1,2,3]$  (categories)
- What would we need to learn the network parameters?
  - for  $P(C) = \text{Count}(\text{datapoints with } C=i) / \text{Count}(\text{all datapoints}) \quad i=1,2,3$
  - for  $P(X_j|C) = \text{Count}(\text{datapoints with } X_j = \text{val}_k \text{ and } C=i) / \text{Counts}(\text{data with } C=i)$   
for all values of  $X_j$  and  $i=1,2,3$

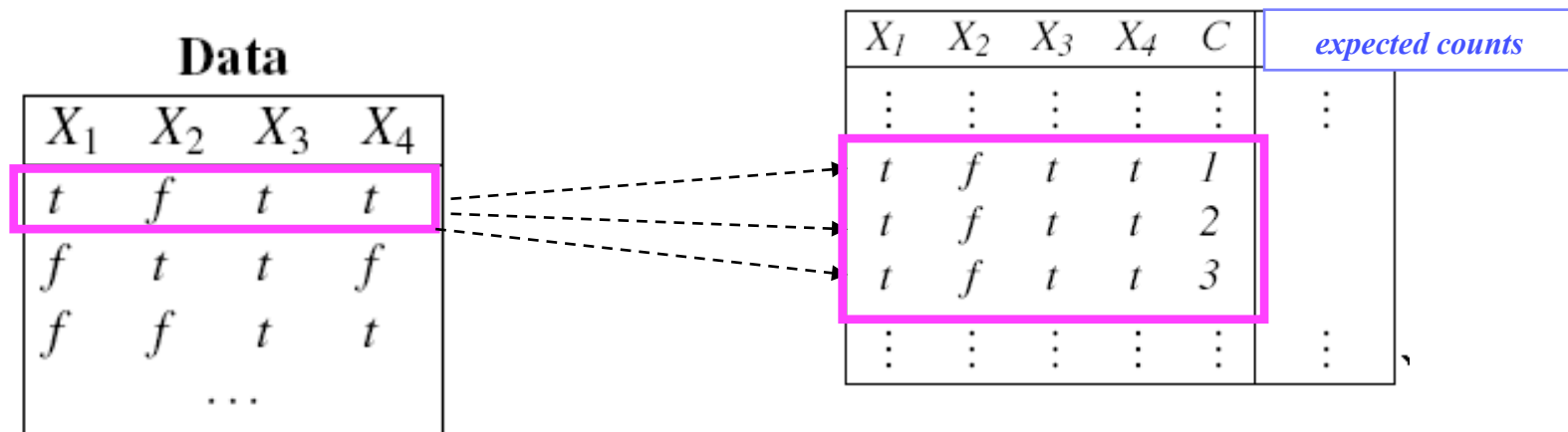
# EM: Augmenting the Data

- We don't have those counts, but we can approximate them with *expected* counts derived from the model
  - or from initial guesses if I start EM from augmented data
- For instance the expected count  $\hat{N}(C = i)$  is the sum, over all  $M$  examples in my dataset, of the probability that each example belongs to category  $i$

$$\hat{N}(C = i) = \sum_{j=1}^M P(C = i \mid \text{attributes of example } e_j)$$

# EM: Augmenting the Data

- Here is how to include expected counts in the original data
- For each tuple in the original data (i.e. [t,f,t,t]) below,



- duplicate it as many times as are the values of  $C$ , and add to each new tuple one of the values of  $C$
- Give an *expected count* for each new tuple (or get it from the model).
  - ✓ Probability that the original datapoint belongs to one of the possible categories  $\Rightarrow$  must make sure that counts associated with each set of augmented tuples sum to 1,

# EM: Augmenting the Data

From now on we will shorten expected counts and *counts*, like the book does, but remember that they are expected counts

**Data**

$X_1$	$X_2$	$X_3$	$X_4$
$t$	$f$	$t$	$t$
$f$	$t$	$t$	$f$
$f$	$f$	$t$	$t$
...			

$X_1$	$X_2$	$X_3$	$X_4$	$C$	<i>count</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t$	$f$	$t$	$t$	1	
$t$	$f$	$t$	$t$	2	
$t$	$f$	$t$	$t$	3	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

# EM: General Idea applied to Naive Class.

- Starts by giving EM the initial, guessed information, to solve the learning problem, that is either
  - the probability that each datapoint belongs to any of the available categories
  - the missing parameters in the network.
- Refines this initial guess by cycling through two basic steps
  - Expectation (E): given the current model, update the expected count for each data point with the expected count of the class given the attributes of that datapoint
    - ✓ I need to infer from the network  $P(C|\text{attr}_1, \text{attr}_2, \dots, \text{attr}_n)$
  - Maximization: given the updated data, infer the Maximum Likelihood (or MAP model)
    - ✓ This is the same step that we described when learning parameters for fully observable networks,

# M-Step

- Infer the maximum likelihood (or maximum a posterior) model parameters from the augmented data.
  - frequencies in the data as estimates of the relevant probabilities
- In the Bayes classifier example, let's suppose to have
  - $M$  examples in the original data
  - $E'$  augmented examples.
  - $val(e, attr)$ , the value of attribute  $attr$  for example  $e$

$$P(C = i) = \frac{\sum_{\{e \in E': val(e, C) = i\}} val(e, count)}{\sum_{\{e \in E'\}} val(e, count)}$$

equals  $M$  because the sum of counts is 1 for each set of augmented examples, and there are  $M$  of these sets

## M-Step

➤ Similarly

$$P(X_j = v_j \mid C = i) = \frac{\sum_{\{e \in E' : val(e, C) = i \wedge val(e, X_j) = v_j\}} val(e, count)}{\sum_{\{e \in E' : val(e, C) = i\}} val(e, count)}$$

## E-Step

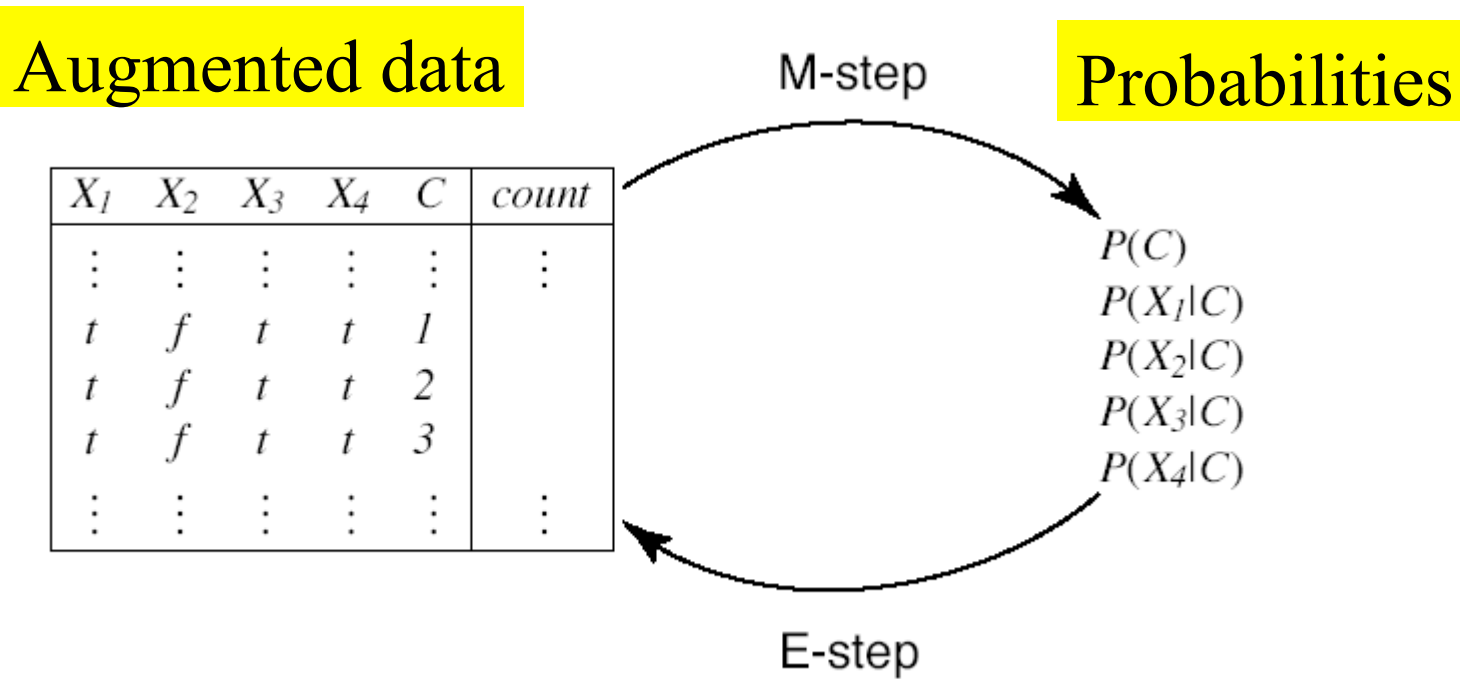
- The E step refines the expected counts in the augmented data by deriving them from the model. So, for instance

$$\begin{aligned} & P(C = 1 \mid x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4) = \\ &= \frac{P(x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4 \mid C = 1)P(C = 1)}{P(x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4)} \\ &= \frac{P(x_1 \mid C = 1)P(\neg x_2 \mid C = 1)P(x_3 \mid C = 1)P(x_4 \mid C = 1)P(C = 1)}{\sum_{i=1}^3 P(x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4 \wedge C = i)} \\ &= \frac{P(x_1 \mid C = 1)P(\neg x_2 \mid C = 1)P(x_3 \mid C = 1)P(x_4 \mid C = 1)P(C = 1)}{\sum_{i=1}^3 P(x_1 \mid C = i)P(\neg x_2 \mid C = i)P(x_3 \mid C = i)P(x_4 \mid C = i)P(C = i)} \end{aligned}$$



# E-Step

- The E-step will be repeated to update all counts in the table
- Ready to start the M-step again

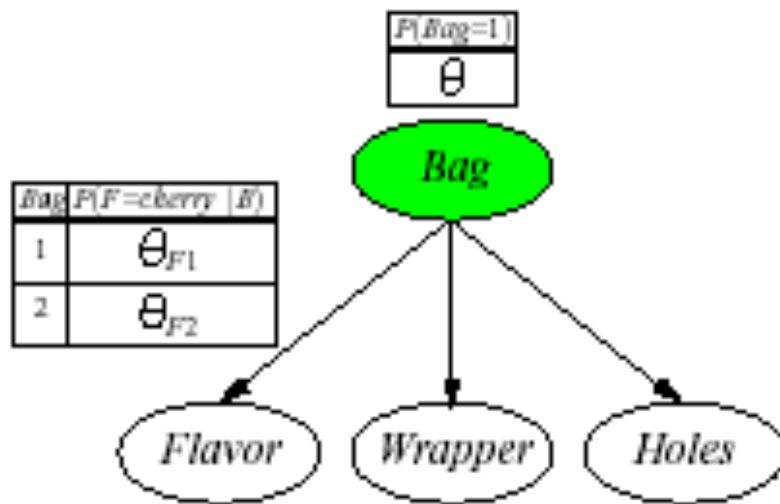


# EM: Discussion

- For more complex Bnets the algorithm is basically the same
  - Basic difference is that the E-Step may involve more complex probabilistic inference to compute the probability of hidden variables given observations
  - The inference can be intractable, in which case there are variations of EM that use sampling algorithms for the E-Step
- The algorithm is sensitive to “degenerated” local maxima due to extreme configurations
  - e.g., data with outliers can generate categories that include only 1 outlier each because these models have the highest log likelihoods
  - Possible solution: re-introduce priors over the learning hypotheses and use the MAP version of EM

# Another Example

- Back to the cherry/lime candy world.
- Two bags of candies (1 and 2) have been mixed together
- Candies are described by 3 features: Flavor and Wrapper as before, plus Hole (whether they have a hole in the middle)
- The distribution of candies in each bag is described again by a naive Bayes model, below



$$\theta = P(\text{Bag} = 1)$$

$$\theta_{Fj} = P(\text{Flavor} = \text{cherry} | \text{Bag} = j)$$

$$\theta_{Wj} = P(\text{Wrapper} = \text{red} | \text{Bag} = j)$$

$$\theta_{Hj} = P(\text{Hole} = \text{yes} | \text{Bag} = j)$$

$$j = 1, 2$$

# Another Example

➤ Assume that the true parameters are

- $\theta = 0.5$ ;
- $\theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8$ ;
- $\theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3$ ;

➤ The following data is generated by sampling the true model

	W=red		W=green	
	H=1	H=0	H=1	H=0
F=cherry	273	93	104	90
F=lime	79	100	94	167

➤ We want to re-learn the true parameters using EM

# Start Point

- This time, we start by directly assigning a guesstimate for the parameters
  - Usually done randomly; here we select numbers convenient for computation

$$\theta^{(0)} = 0.6;$$

$$\theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6;$$

$$\theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$$

- We'll work through one cycle of EM to compute  $\theta^{(1)}$ .

# E-step

- First, we need the expected count of candies from Bag 1,
- Sum of the probabilities that each of the  $N$  data points comes from bag 1
  - Be  $flavor_j$ ,  $wrapper_j$ ,  $hole_j$  the values of the corresponding attributes for the  $j^{\text{th}}$  datapoint

$$\begin{aligned}\hat{N}(\text{Bag} = 1) &= \sum_{j=1}^N P(\text{Bag} = 1 | flavor_j, wrapper_j, hole_j) = \\ &= \sum_{j=1}^N \frac{P(flavor_j, wrapper_j, hole_j | \text{Bag} = 1) P(\text{Bag} = 1)}{P(flavor_j, wrapper_j, hole_j)} \\ &= \sum_{j=1}^N \frac{P(flavor_j | \text{Bag} = 1) P(wrapper_j | \text{Bag} = 1) P(hole_j | \text{Bag} = 1) P(\text{Bag} = 1)}{\sum_i P(flavor_j | \text{Bag} = i) P(wrapper_j | \text{Bag} = i) P(hole_j | \text{Bag} = i) P(\text{Bag} = i)}\end{aligned}$$

## E-step

$$\sum_{j=1}^N \frac{P(\text{flavor}_j | \text{Bag} = 1)P(\text{wrapper}_j | \text{Bag} = 1)P(\text{hole}_j | \text{Bag} = 1)P(\text{Bag} = 1)}{\sum_i P(\text{flavor}_j | \text{Bag} = i)P(\text{wrapper}_j | \text{Bag} = i)P(\text{hole}_j | \text{Bag} = i)P(\text{Bag} = i)}$$

- This summation can be broken down into the 8 candy groups in the data table.
- For instance the sum over the 273 cherry candies with red wrap and hole (first entry in the data table) gives

$$= 273 \frac{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)}}{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)} + \theta_{F2}^{(0)} \theta_{W2}^{(0)} \theta_{H2}^{(0)} (1 - \theta^{(0)})} =$$
$$273 \frac{0.6^4}{0.6^4 + 0.4^4} = 273 \frac{0.1296}{0.1552} = 227.97$$

## M-step

- If we do compute the sums over the other 7 candy groups we get

$$\hat{N}(\text{Bag} = 1) = 612.4$$

- At this point, we can perform the M-step to refine  $\theta$ , by taking the expected frequency of the data points that come from Bag 1

$$\theta(1) = \frac{\hat{N}(\text{Bag} = 1)}{N} = 0.6124$$

# One More Parameter

- If we want to do the same for parameter  $\theta_{F1}$
- E-step: compute the expected count of cherry candies from Bag 1

$$\hat{N}(Bag = 1 \wedge Flavor = cherry) = \sum_{j: Flavor_j = cherry} P(Bag = 1 | Flavor_j = cherry, wrapper_j, whole_j)$$

- Can compute the above value from the Naïve model as we did earlier
- M-step: refine  $\theta_{F1}$  by computing the corresponding expected frequencies

$$\theta_{F1}^{(1)} = \frac{\hat{N}(Bag = 1 \wedge Flavor = cherry)}{\hat{N}(Bag = 1)}$$

# Learning Performance

- After a complete cycle through all the parameters, we get

$$\theta^{(1)} = 0.6124;$$

$$\theta_{F1}^{(1)} = 0.6684; \quad \theta_{W1}^{(1)} = 0.6483; \quad \theta_{H1}^{(1)} = 0.658;$$

$$\theta_{F2}^{(1)} = 0.3887; \quad \theta_{W2}^{(1)} = 0.3817; \quad \theta_{H2}^{(1)} = 0.3827;$$

- For any set of parameters, I can compute the log likelihood as we did in the previous class

$$P(\mathbf{d} | h_{\theta^{(i)}_{F1} \theta^{(i)}_{W1} \theta^{(i)}_{H1} \theta^{(i)}_{F2} \theta^{(i)}_{W2} \theta^{(i)}_{H2}}) = \prod_{j=1}^{1000} P(d_j | h_{\theta^{(i)}_{F1} \theta^{(i)}_{W1} \theta^{(i)}_{H1} \theta^{(i)}_{F2} \theta^{(i)}_{W2} \theta^{(i)}_{H2}})$$

- It can be shown that the log likelihood increases with each EM iteration, surpassing even the likelihood of the original model after only 3 iterations

# EM: Discussion

- For more complex Bnets the algorithm is basically the same
  - In general, I may need to compute the conditional probability parameter for each variable  $X_i$  given its parents  $Pa_i$

- $\theta_{ijk} = P(X_i = x_{ij} | Pa_i = pa_{ik})$

$$\theta_{ijk} = \frac{\hat{N}(X_i = x_{ij}; Pa_i = pa_{ik})}{\hat{N}(Pa_i = pa_{ik})}$$

- The expected counts are computed by summing over the examples, after having computed for each  $P(X_i = x_{ij}, Pa_i = pa_{ik})$  using any Bnet inference algorithm
- The inference can be intractable, in which case there are variations of EM that use sampling algorithms for the E-Step

# EM: Discussion

- The algorithm is sensitive to “degenerated” local maxima due to extreme configurations
  - e.g., data with outliers can generate categories that include only 1 outlier each because these models have the highest log likelihoods
  - Possible solution: re-introduce priors over the learning hypothesis and use the MAP version of EM