Web-Mining Agents

Prof. Dr. Ralf Möller
Dr. Özgür L. Özçep

Universität zu Lübeck
Institut für Informationssysteme

Tanya Braun (Lab)

Acknowledgements

- Slides from AIMA book provided by Cristina Conati, UBC
  http://www.cs.ubc.ca/~conati/index.php

Overview

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

Full Bayesian Learning

- In the learning methods seen in first lecture the idea was always to find the best model that could explain some observations (best concept in classification or best polynomial coefficients in regression)
- 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) $h_1 \ldots, h_n$
  - $P(H)$ = prior probability distribution over hypothesis space
- j_{th} observation d_j gives the outcome of random variable D_j
  - training data $\mathbf{d} = d_1,..,d_k$

Example

- Suppose we have 5 types of candy bags
  - 10% are 100% cherry candies (h_{100})
  - 20% are 75% cherry + 25% lime candies (h_{75})
  - 40% are 50% cherry + 50% lime candies (h_{50})
  - 20% are 25% cherry + 75% lime candies (h_{25})
  - 10% are 100% lime candies (h_0)

- Then we observe candies drawn from some bag 

- θ = the parameter that defines the fraction of cherry candy in a bag

  h_θ = corresponding hypothesis

- Which bag has generated my 10 observations? $P(h_\theta \mid \mathbf{d})$.

- What flavour will the next candy be? Prediction $\mathbf{P}(X|\mathbf{d})$

Full Bayesian Learning

- Given the data so far, each hypothesis h_i has a posterior probability:
  - $P(h_i \mid \mathbf{d}) = \alpha P(\mathbf{d} \mid h_i)\, P(h_i)$ (Bayes theorem)
  - where $P(\mathbf{d} \mid h_i)$ 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:
  - $\mathbf{P}(X \mid \mathbf{d}) =$
    - $= \sum_i \mathbf{P}(X, h_i \mid \mathbf{d})$
    - $= \sum_i \mathbf{P}(X \mid h_i, \mathbf{d})\, P(h_i \mid \mathbf{d})$
    - $= \sum_i \mathbf{P}(X \mid h_i)\, P(h_i \mid \mathbf{d})$
    - $\sim \sum_i \mathbf{P}(X \mid h_i)\, P(\mathbf{d} \mid h_i)\, P(h_i)$
  - The weights are given by the data likelihood and prior of each h
- No need to pick one best-guess hypothesis!

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

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \alpha P(X|Y)P(Y)$$

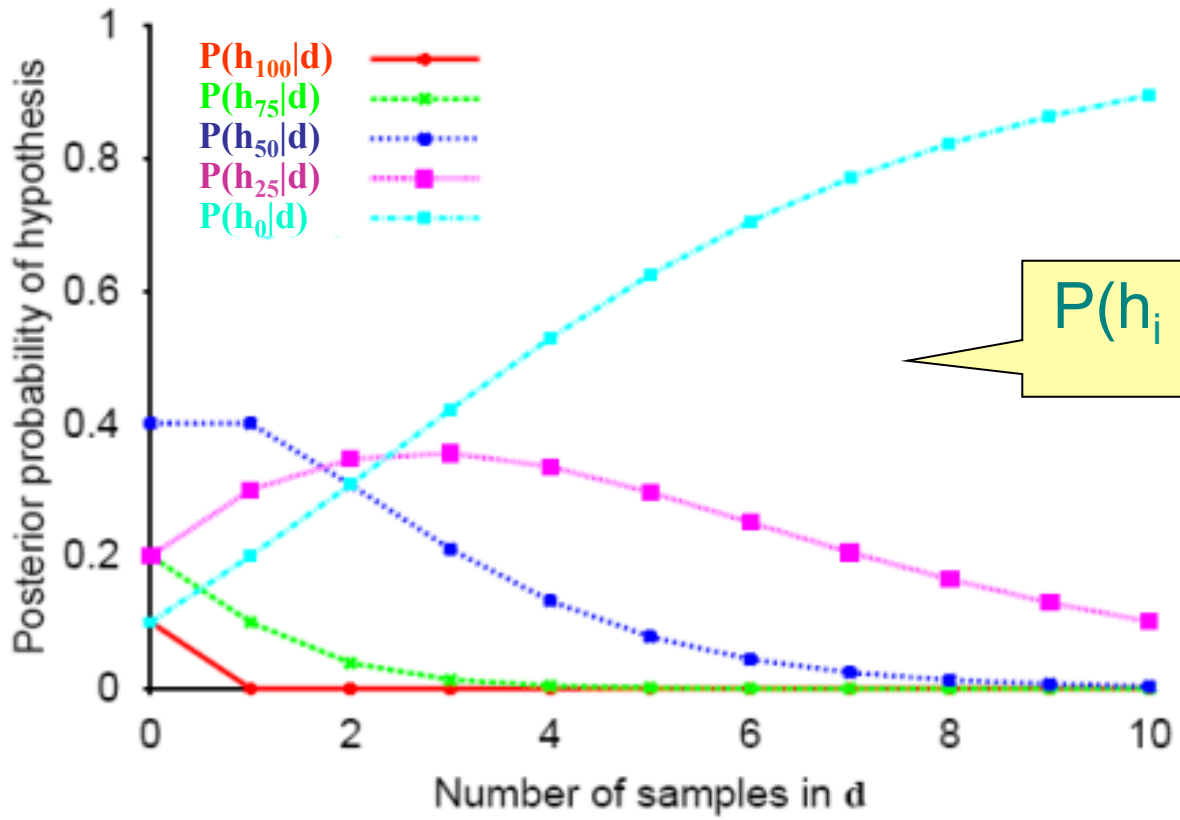
Example

- 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 $j=1,..,10$
- For a given h_θ, the value of $P(d_j| h_\theta)$ is
  - $P(d_j = \text{cherry}| h_\theta) = \theta$; $P(d_j = \text{lime}|h_\theta) = (1-\theta)$
- Given observations, of which c are cherry and l = N-c lime

$$P(\boldsymbol{d} \mid h_\theta) = \prod_{j=1}^{c} \theta \prod_{j=1}^{\ell} (1-\theta) = \theta^c (1-\theta)^\ell$$

  - **Binomial distribution**: probability of # of successes in a sequence of N independent trials with binary outcome, each of which yields success with probability θ.
- For instance, after observing 3 lime candies in a row:
  - $P([\text{lime, lime, lime}] \mid h_{50}) = 0.5^3$ because the probability of seeing lime for each observation is 0.5 under this hypotheses

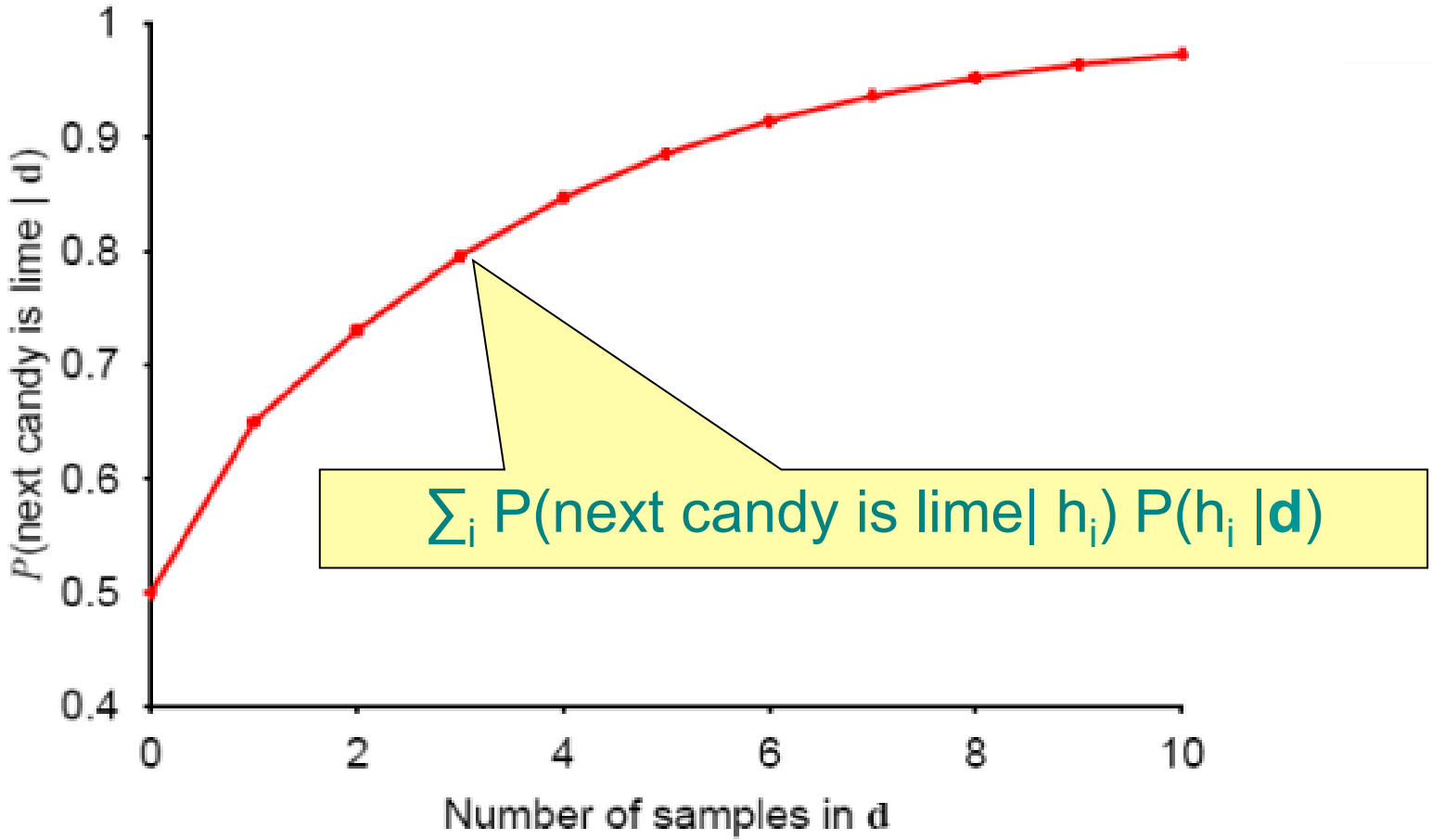
All-limes: Posterior Probability of H



$P(h_i \mid \mathbf{d}) = \alpha P(\mathbf{d} \mid h_i)\, P(h_i)$

- Initially, the hypothesis with higher priors dominate (h_{50} with prior = 0.4)
- As data comes in, the finally best 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

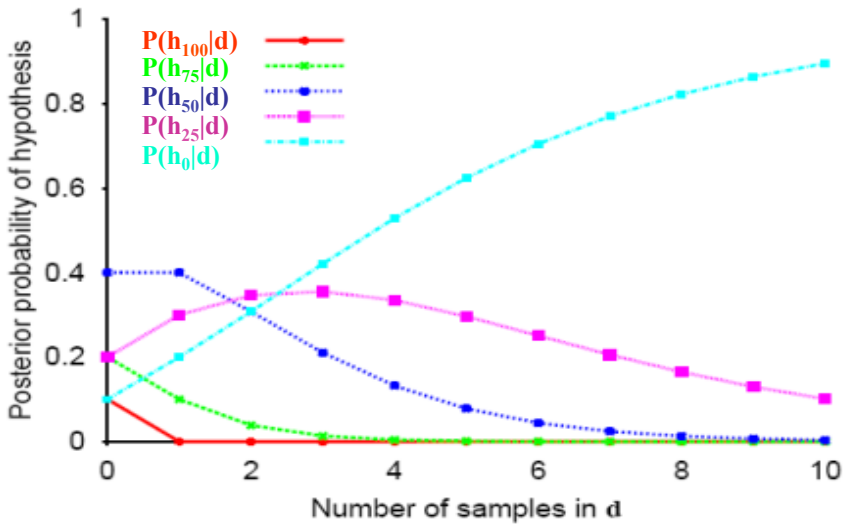
- Full Bayesian Learning
- MAP learning
- Maximun Likelihood Learning
- Learning Bayesian Networks
  - Fully observable
  - With hidden (unobservable) variables

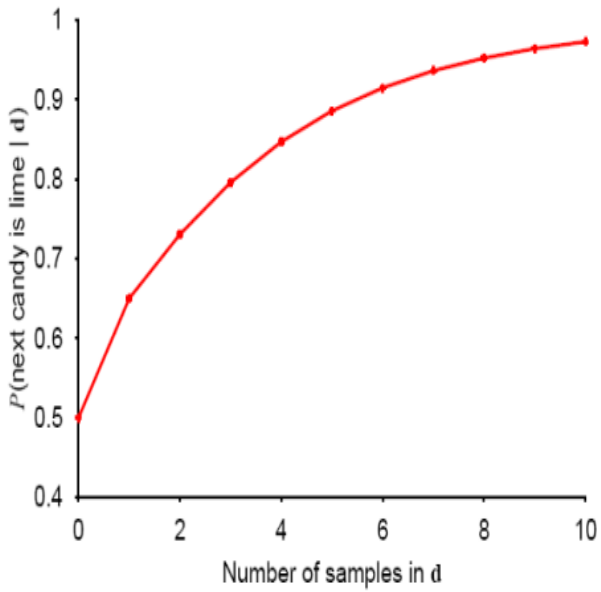
MAP approximation

- 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 posterior (MAP) learning:
  - Instead of doing prediction by considering all possible hypotheses, as in
    - $\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)\, P(h_i|\mathbf{d})$
  - Make predictions based on h_{MAP} that maximizes $P(h_i|\mathbf{d})$
    - I.e., maximize $P(\mathbf{d}|h_i)\, P(h_i)$
    - $\mathbf{P}(X|\mathbf{d}) \sim \mathbf{P}(X|h_{MAP})$

MAP approximation

- MAP is a good approximation when $P(X \mid \mathbf{d}) \approx P(X \mid h_{MAP})$
  - In our example, h_{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 learning bias)
- 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 priory 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
- Maximum 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 maximizing $P(\mathbf{d}|\, h_i)$
- When is ML appropriate?

Howard, R.: Decision analysis: Perspectives on inference, decision, and experimentation. Proceedings of the IEEE 58(5), 632-643, **1970**

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 who 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

A useful distinction for the beginning

- We are going to describe methods for learning BNs (parameters and structure)
- As generated BN provides full joint prob. distribution one can do any kind of inference (prediction, classification, any probability of RVs) one is interested

- > Generative models

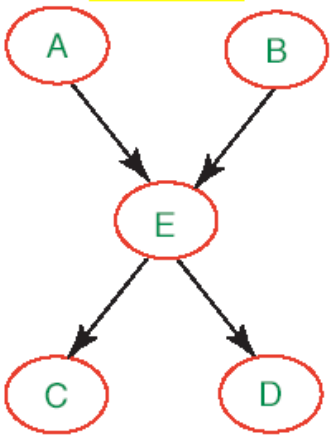
- In contrast there are Discriminative models (s.a. neural networks)
  - specifically designed and trained to maximize performance of classification: $P(Y \mid X)$

    where Y is classification RV and X the vector of features
  - By focusing on modeling the conditional distribution, they generally perform better on classification than generative models when given a reasonable amount of training data

Learning BNets: Complete Data

- We 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 we need to learn are the network's parameters

Model



Data

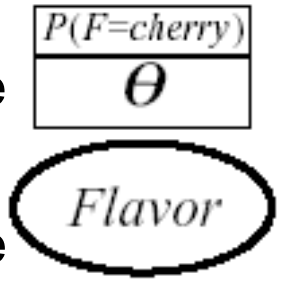
A	B	C	D	E
t	f	t	t	f
f	t	t	t	t
t	t	f	t	f
		$\ldots$		

$\longrightarrow$ Probabilities

$P(A)$
$P(B)$
$P(E|A, B)$
$P(C|E)$
$P(D|E)$

ML learning: example

- Back to the candy example:
  - New candy manufacturer that does not provide data on the probability of fraction θ of cherry candy in its bags
  - Any θ is possible: continuum of hypotheses h_θ
  - Reasonable to assume that all θ are equally likely (we have no evidence of the contrary): uniform distribution $P(h_\theta)$
  - θ 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 θ of cherry candies in the bag



- We want to infer θ 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 θ 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 maximize, we differentiate $L(P(\mathbf{d}|\ h_\theta)$ with respect to θ and set the result to 0

$$\frac{\partial(c\log\theta + \ell\log(1-\theta))}{\partial\theta}$$

$$= \frac{c}{\theta} - \frac{\ell}{1-\theta}$$

$$= \frac{c}{\theta} - \frac{N-c}{1-\theta} = 0$$

- 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 cherries in the data

- Now we have justified why this approach provides a reasonable estimate of node priors

General ML procedure

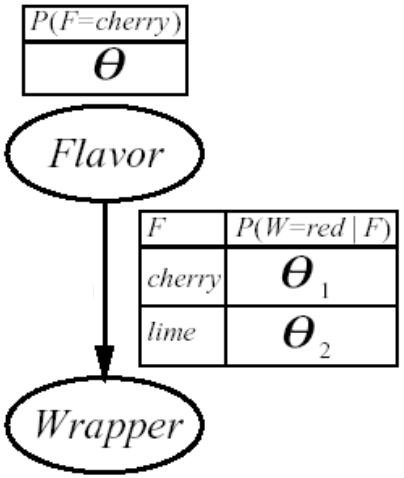
- Express the likelihood of the data as a function of the parameters to be learned
- Take the derivative of the log likelihood with respect to each parameter
- Find the parameter value that makes the derivative equal to 0
- The last step can be computationally very expensive in real-world learning tasks

More complex example

- The manufacturer chooses the color of the wrapper probabilistically for each candy based on flavor, following an unknown distribution
  - If the flavour is cherry, it chooses a red wrapper with probability θ_1
  - If the flavour is lime, it chooses a red wrapper with probability θ_2
- The Bayesian network for this problem includes 3 parameters to be learned
  - θ, θ_1, θ_2

More complex example

- The manufacturer chooses the color of the wrapper probabilistically for each candy based on flavor, following an unknown distribution
  - If the flavour is cherry, it chooses a red wrapper with probability θ_1
  - If the flavour is lime, it chooses a red wrapper with probability θ_2
- The Bayesian network for this problem includes 3 parameters to be learned
  - $\theta\ \theta_1\ \theta_2$

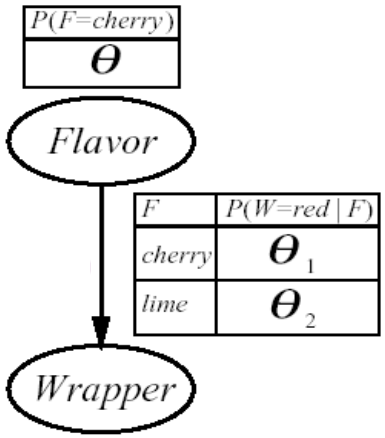


Another example (cont' d)

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

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

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



- 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

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

  $= \prod_j P(d_j | h_{\theta\theta_1\theta_2}) = \theta^c (1-\theta)^l (\theta_1)^{r^c} (1-\theta_1)^{g^c} (\theta_2)^{r^l} (1-\theta_2)^{g^l}$

Another example (cont' d)

- 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)

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

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

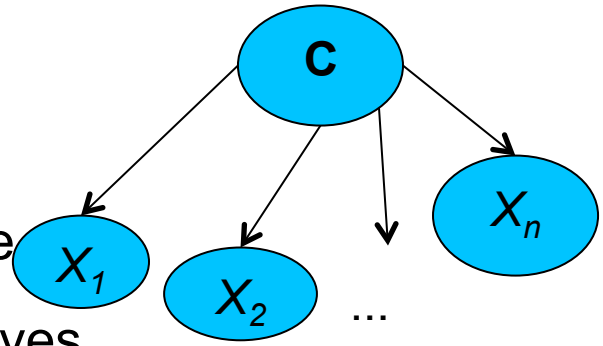
$$\frac{\partial L}{\partial \theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 \qquad \Rightarrow \quad \theta_2 = \frac{r_\ell}{r_\ell+g_\ell}$$

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

Very Popular Application

- Naïve Bayes models: very simple Bayesian networks for classification
  - Class variable (to be predicted) is the root node
  - Attribute variables X_i (observations) are the leaves



- Naïve because it assumes that the attributes are conditionally independent of each other given the class

$$P(C|x_1,x_2,..,x_n) = \frac{P(C,x_1,x_2,..,x_n)}{P(x_1,x_2,..,x_n)} = \alpha P(C)\prod_i P(x_n \mid C)$$

- Deterministic prediction can be obtained by picking the most likely class
- Scales up really well: with n boolean attributes we just need…….

2n+1 parameters

Problem with ML parameter learning

- With small datasets, some of the frequencies may be 0 just because we 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

- 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

  <n,m>

  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 ¬A is observed, increment m alone

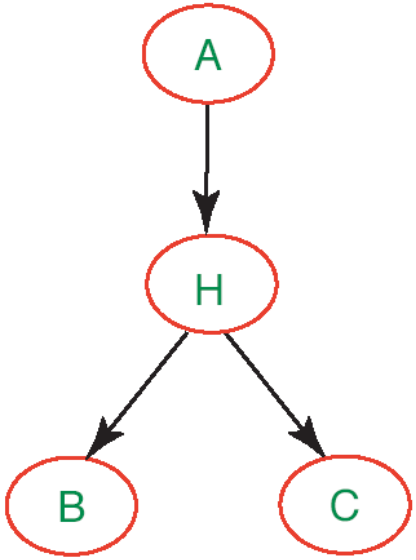
- The absolute values in the pair can be used to express the expert‘s level of confidence in her estimate
  - Small values (e.g., <2,3>) represent low confidence
  - The larger the values, the higher the confidence as it takes more and more data to dominate the initial estimate (e.g. <2000, 3000>)

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 we can collect data on all variables in the network
- What if this is not true, i.e. the network has hidden variables?

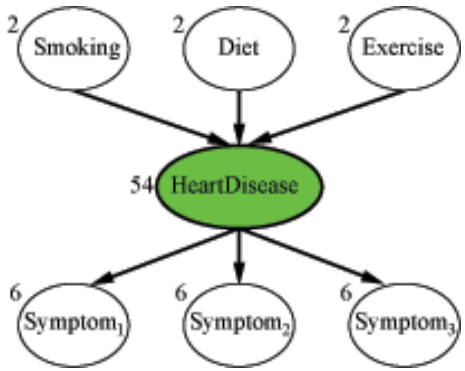


A	B	C
t	f	t
f	t	t
t	t	f
	...	

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

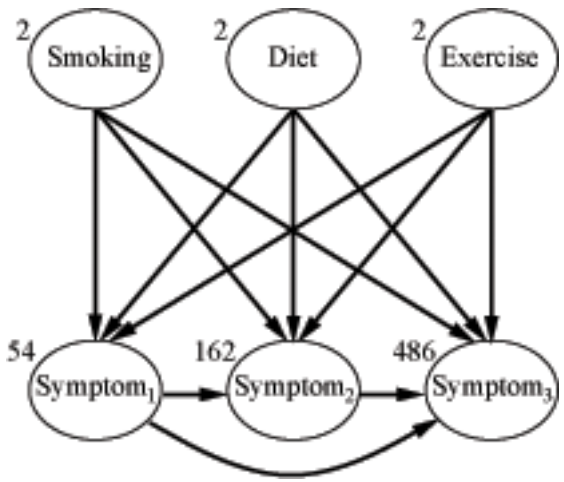
Quick Fix

- 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 attached to the nodes represent how many parameters need to be specified for the CPT of that node
- 78 probabilities to be specified overall

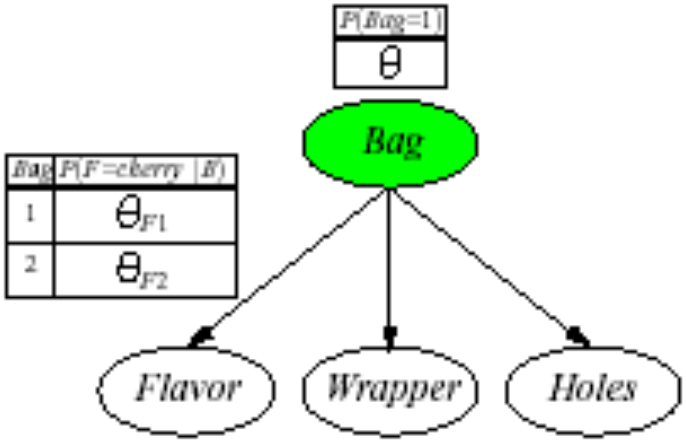
Not Necessarily a Good Fix



- 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

Example: The cherry/lime candy world again

- 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)
- Candies‘ features depend probabilistically from the bag they originally came from
- We want to predict for each candy, which was its original bag, from its features: Naïve Bayes model



$\theta = P(Bag = 1)$

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

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

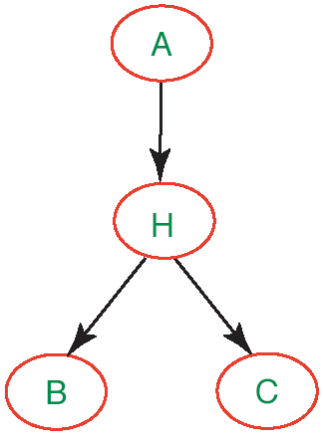
$\theta_{Hj} = P(Hole = yes|Bag = j)$

$j = 1,2$

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 (i.e. no categorization label)
- Expectation-Maximization
  - General algorithm for learning model parameters from incomplete data
  - We'll see how it works on learning parameters for Bnets with discrete variables

EM: general idea



A	B	C
t	f	t
f	t	t
t	t	f
	...	

- If we had data for all the variables in the network, we could learn the parameters by using ML (or MAP) models
  - 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 P(H|A,B,C)

EM: General Idea

- The algorithm starts from "invented" (e.g., randomly generated) information to solve the learning problem, i.e.
  - Determine the network parameters
- 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, update the model parameters using the Maximum Likelihood (ML) approach
    - ✓ This is the same step that we described when learning parameters for fully observable networks

EM: How it Works on Naive Bayes

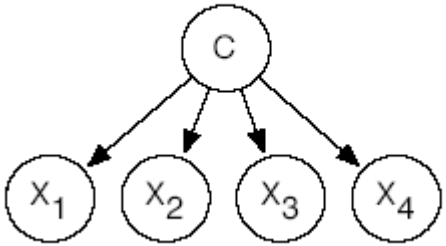
- Consider the following data,
  - N examples with Boolean attributes X1, X2, X3, X4

Data

X_1	X_2	X_3	X_4
t	f	t	t
f	t	t	f
f	f	t	t
		$\ldots$	

- which we want to categorize in one of three possible values of class C = {1,2,3}
- We use a Naive Bayes classifier with hidden variable C

Model



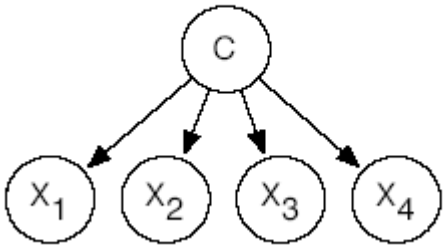
→ **Probabiliti**

$P(C)$	?
$P(X_1\|C)$	?
$P(X_2\|C)$	?
$P(X_3\|C)$	?
$P(X_4\|C)$	?

EM: Initialization

- The algorithm starts from "invented" (e.g., randomly generated) information to solve the learning problem, i.e.
  - Determine the network parameters

Model



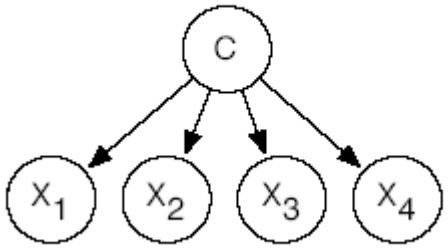
→ Probabiliti

$P(C)$?
$P(X_1|C)$?
$P(X_2|C)$?
$P(X_3|C)$?
$P(X_4|C)$?

Define arbitrary parameters

EM: Expectation Step (Get Expected Counts)

Model



Data

X_1	X_2	X_3	X_4
t	f	t	t
f	t	t	f
f	f	t	t
		$\ldots$	

→ Probabilities

$P(C)$?
$P(X_1|C)$?
$P(X_2|C)$?
$P(X_3|C)$?
$P(X_4|C)$?

- What would we need to learn the network parameters using the ML approach?
  - $P(C = i)$ = #(data with C=i) / #(all datapoints) for i=1,2,3
  - $P(X_h = val_k | C = i)$ = #(data with $X_h = val_k$ and C=i) / #(data with C=i)

    for all values val_k of X_h and i=1,2,3

Remember that the equations result from our already derived knowledge that the most likely paramters (CPTs) are given by frequencies!

EM: Expectation Step (Get Expected Counts)

- We only have #(all datapoints) = N and

  counts of instantiations for non-hidden RVs within data

- We approximate all other necessary counts with expected counts derived from the model with "invented" parameters

- Expected count $\hat{N}(C = i)$ is the sum, over all N examples in my dataset, of the probability that each example is in category i

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

$$= \sum_{j=1}^{N} P(C = i \mid x1_j, x2_j, x3_j, x4_j)$$

EM: Expectation Step (Get Expected Counts)

- How do we get the necessary probabilities from the model?

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

$$= \sum_{j=1}^{N} P(C=i \mid x1_j, x2_j, x3_j, x4_j)$$

- Easy with a Naïve bayes network

$$P(C=i \mid x1_j, x2_j, x3_j, x4_j) = \frac{P(C=i, x1_j, x2_j, x3_j, x4_j)}{P(x1_j, x2_j, x3_j, x4_j)}$$

$$= \frac{P(x1_j \mid C=i)..,P(x4_j \mid C=i)P(C=i)}{P(x1_j, x2_j, x3_j, x4_j)}$$

Also available from Naïve Bayes. You do the necessary transformations

Naïve bayes “invented parameters” (“old” P(C=i))

EM: Expectation Step (Get Expected Counts)

- By a similar process we obtain the expected counts of examples with attribute $X_h = val_k$ and belonging to category i.

- These are needed *later* for estimating $\mathbf{P}(X_h \mid C)$:

$$P(X_h = val_k | C = i) = \frac{\text{Exp.-\#(examples with } X_h = val_k \text{ and } C = i)}{\text{Exp.-\#(examples with } C = i)} = \frac{\hat{N}(X_h = val_k, C = i)}{\hat{N}(C = i)}$$

  - for all values val_k of X_h and i=1,2,3

Again, get these probabilities from model with current parameters

- For instance

$$\hat{N}(X_1 = t, C = 1) = \sum_{e_j \text{ with } X_1 = t} P(C = i \mid x1_j = t, x2_j, x3_j, x4_j)$$

EM: General Idea

- The algorithm starts from "invented" (e.g., randomly generated) information to solve the learning problem, i.e.
  - the network parameters
- It then refines this initial guess by cycling through two basic steps
  - Expectation (E): compute expected counts based on the generated via the current model
  - Maximization (M): given the expected counts, update the model parameters using the Maximum Likelihood (ML) approach
    - ✓ This is the same step that we described when learning parameters for fully observable networks

Maximization Step: (Refining Parameters)

- Now we can refine the network parameters by applying ML to the expected counts

$$P(C = i) = \frac{\hat{N}(C = i)}{N}$$

$$P(X_j = val_k | C = i) = \frac{\hat{N}(X_j = val_k, C = i)}{\hat{N}(C = i)}$$

- for all values val_k of X_j and $i=1,2,3$

EM Cycle

- Ready to start the E-step again

Expected Counts
(“Augmented data”)

M-step

Probabilities

X_1	X_2	X_3	X_4	C	*count*
⋮	⋮	⋮	⋮	⋮	⋮
t	f	t	t	1	
t	f	t	t	2	
t	f	t	t	3	
⋮	⋮	⋮	⋮	⋮	⋮

$P(C)$
$P(X_1|C)$
$P(X_2|C)$
$P(X_3|C)$
$P(X_4|C)$

E-step

Note: Actually you never generate any data in E-step

Procedure EM(X,D,k)

Inputs: X set of features $X=\{X_1,...,X_n\}$; D data set on features $\{X_1,...,X_n\}$; k number of classes

Output: $P(C)$, $P(X_i|C)$ for each $i \in \{1:n\}$, where $C=\{1,...,k\}$.

Local

real array $A[X_1,...,X_n,C]$

real array $P[C]$

real arrays $M_i[X_i,C]$ for each $i \in \{1:n\}$

real arrays $P_i[X_i,C]$ for each $i \in \{1:n\}$

$s \leftarrow$ number of tuples in D

Assign $P[C]$, $P_i[X_i,C]$ arbitrarily

repeat

// E Step

for each assignment $\langle X_1=v_1,...,X_n=v_n \rangle \in D$ **do**

let $m \leftarrow |\langle X_1=v_1,...,X_n=v_n \rangle \in D|$

for each $c \in \{1:k\}$ **do**

$A[v_1,...,v_n,c] \leftarrow m \times P(C=c|X_1=v_1,...,X_n=v_n)$

end for each

end for each

// M Step

for each $i \in \{1:n\}$ **do**

$M_i[X_i,C]=\sum_{X1,...,Xi-1,Xi+1,...,Xn} A[X_1,...,X_n,C]$

$P_i[X_i,C]=(M_i[X_i,C])/(\sum_C M_i[X_i,C])$

end for each

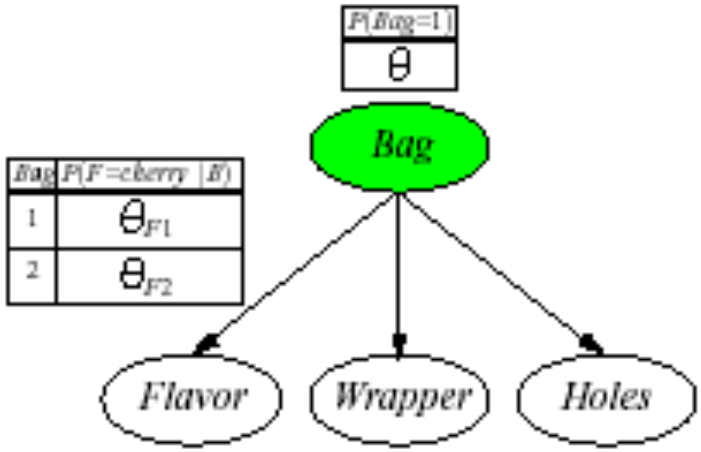
$P[C]=\sum_{X1,...,Xn} A[X_1,...,X_n,C]/s$

until probabilities do not change significantly

end procedure

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)
- Candies‘ features depend probabilistically from the bag they originally came from
- We want to predict for each candy, which was its original bag, from its features: Naïve Bayes model



$\theta = P(Bag = 1)$

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

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

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

$j = 1,2$

Data

- 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 counts are "generated" from **P**(C, F, W, H) (N = 1000)

	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

EM: Initialization

- Assign arbitrary initial 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 jth datapoint

$$\hat{N}(\text{Bag} = 1) = \sum_{j=1}^{N} P(Bag = 1 \mid flavor_j, wrapper_j, hole_j) =$$

$$= \sum_{j=1}^{N} \frac{P(flavor_j, wrapper_j, hole_j \mid Bag = 1) P(Bag = 1)}{P(flavor_j, wrapper_j, hole_j)}$$

$$= \sum_{j=1}^{N} \frac{P(flavor_j \mid Bag = 1) P(wrapper_j \mid Bag = 1) P(hole_j \mid Bag = 1) P(Bag = 1)}{\sum_i P(flavor_j \mid Bag = i) P(wrapper_j \mid Bag = i) P(hole_j \mid Bag = i) P(Bag = i)}$$

E-step

$$\sum_{j=1}^{N} \frac{P(flavor_j|Bag=1)P(wrapper_j|Bag=1)P(hole_j|Bag=1)P(Bag=1)}{\sum_i P(flavor_j|Bag=i)P(wrapper_j|Bag=i)P(hole_j|Bag=i)P(Bag=i)}$$

- 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

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

$$= 273 \frac{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)}}{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)} + \theta_{F2}^{(0)}\theta_{W2}^{(0)}\theta_{H2}^{(0)}(1-\theta^{(0)})} =$$

$$273 \frac{0.6^4}{0.6^4 + 0.4^4} = 273 \frac{0.1296}{0.1552} = 227.97$$

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

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

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

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 θ, 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 θ_{F1}
- E-step: compute the expected count of cherry candies from Bag 1

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

  - Can compute the above value from the Naïve model as we did earlier
  - TRY AS AN EXCERCISE
- M-step: refine θ_{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, we can compute the log likelihood as we did in the previous class

- It can be seen that the log likelihood increases with each EM iteration

- EM tends to get stuck in local maxima, so it is often combined with gradient-based techniques in the last phase of learning

Learning Performance

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

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

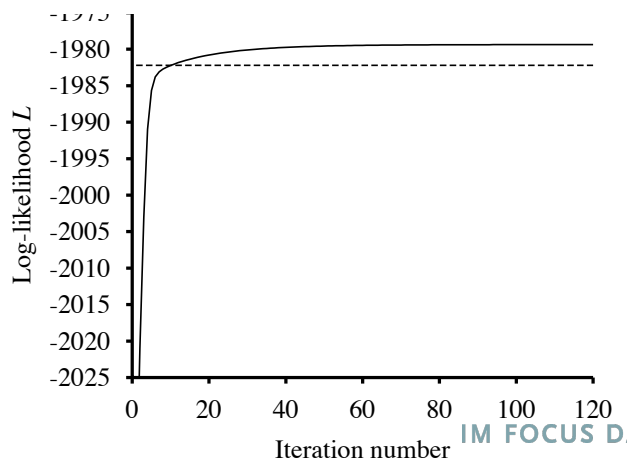
$$\theta_{F1}^{(1)} = 0.6684; \quad \theta_{W1}^{(1)} = 0.6483; \quad \theta_{H1}^{(1)} = 0.658;$$

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

- For any set of parameters, one computes the log likelihood as we did in the previous class

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

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

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

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

- The expected counts are computed by summing over the examples, after having computed all the necessary $P(X_i = x_{ij}, Pa_i = pa_{ik})$ 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

Bayesian learning

- We saw three ways of Bayesian learning:
  - **Full Bayesian Learning aka BMA (Bayesian Model Averaging)**
  - **MAP (Maximum A Posteriori) hypothesis**
  - **MLE (Maximum Likelihood Estimate)**
- Another principle (see later lectures) is
  - **MDL (Minimum Description Length) principle:** Use some encoding to model the complexity of the hypothesis, and the fit of the data to the hypothesis, then minimize the overall description length of $h_i + D$

Parameter estimation

- Assume known structure
- Goal: estimate BN parameters θ
  - CPT entries P(X | Parents(X))
- A parameterization θ is good if it is likely to generate the observed data:

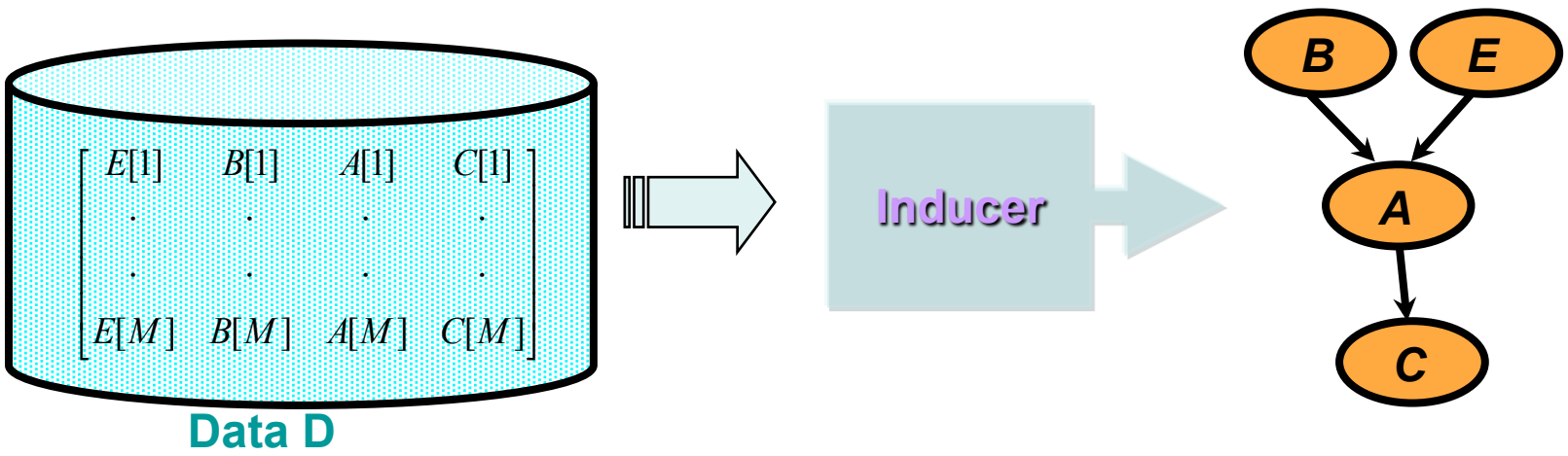
$$Score(\boldsymbol{\theta}) = P(D \mid \boldsymbol{\theta}) = \prod_m P(x[m] \mid \boldsymbol{\theta})$$

i.i.d. samples

- Maximum Likelihood Estimation (MLE) Principle: Choose θ* so as to maximize Score

Learning Bayesian network structures

- Given training set $\boldsymbol{D} = \{\boldsymbol{x}[1],\ldots,\boldsymbol{x}[M]\}$
- Find model that best matches ***D***
  - model selection
  - parameter estimation



Some of the following slides from an AI course „Graphical models“ by Burgard/De Raedt/Kersting/Nebel

Model selection

Goal: Select the best network structure, given the data

Input:

- Training data
- Scoring function

Output:

- A network that maximizes the score

Structure selection: Scoring

- Bayesian: prior over parameters and structure
  - get balance between model complexity and fit to data as a byproduct

Can we learn G's params from D?

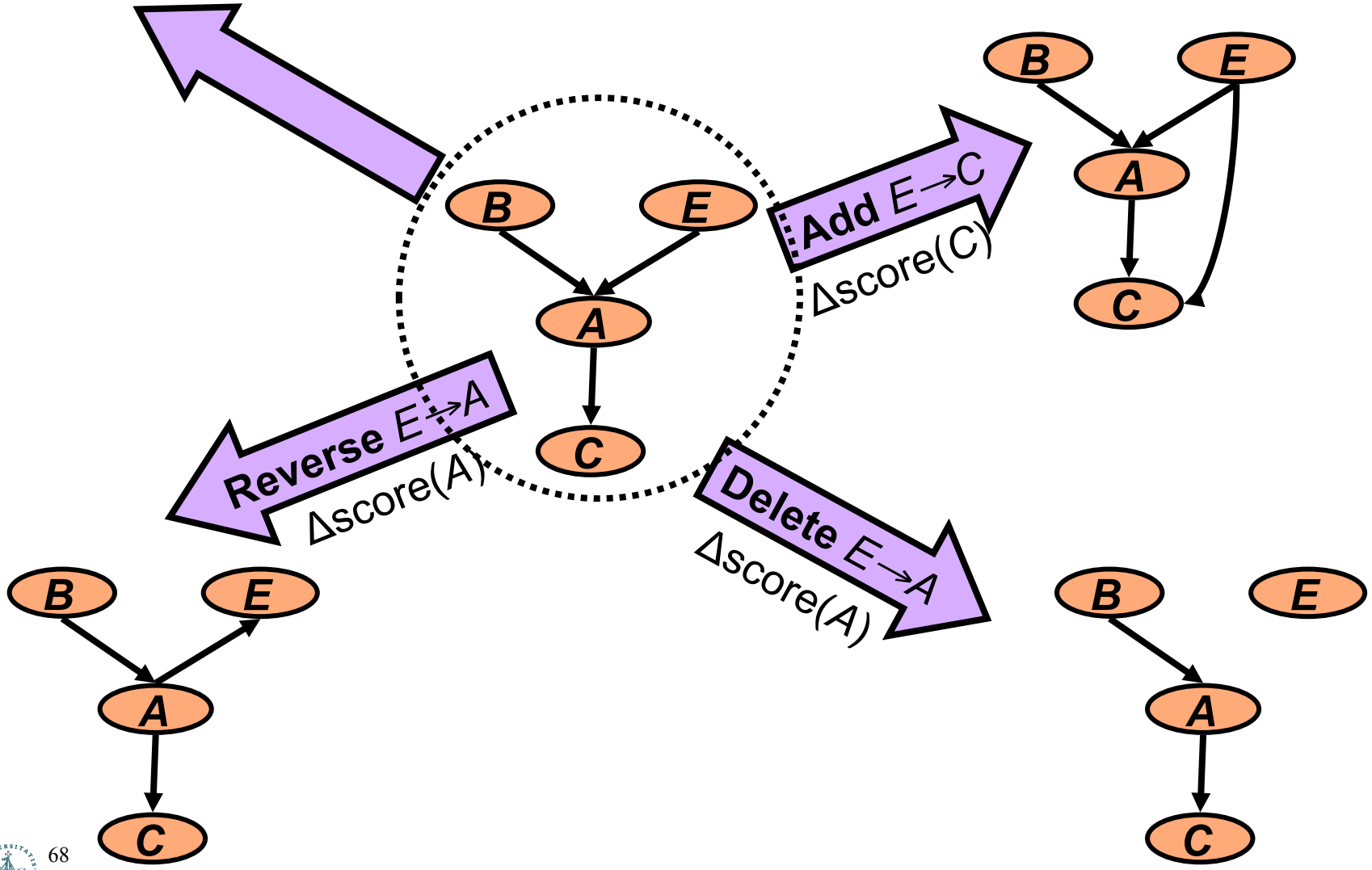
Does G explain D with ML?

Prior w.r.t. MDL

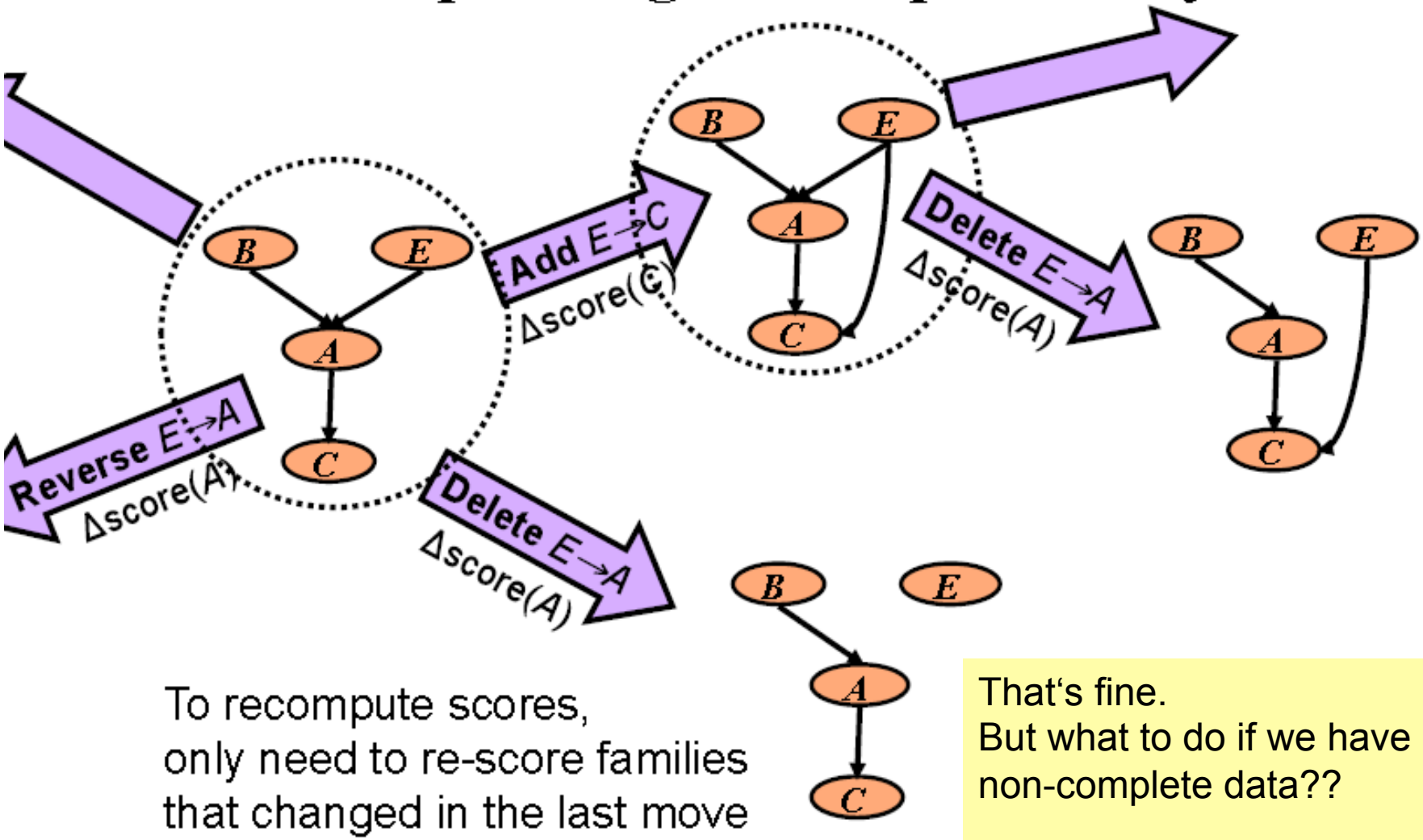
- $\text{Score}_D(G) = \log P(G|D) = \alpha \log [P(D|G)\, P(G)]$
- Marginal likelihood just comes from our parameter estimates
- Prior on structure can be any measure we want; typically a function of the network complexity (MDL principle) **Same key property: Decomposability**

$$\text{Score(structure)} = \Sigma_i \text{ Score(substructure of } X_i)$$

Heuristic search



Exploiting decomposability



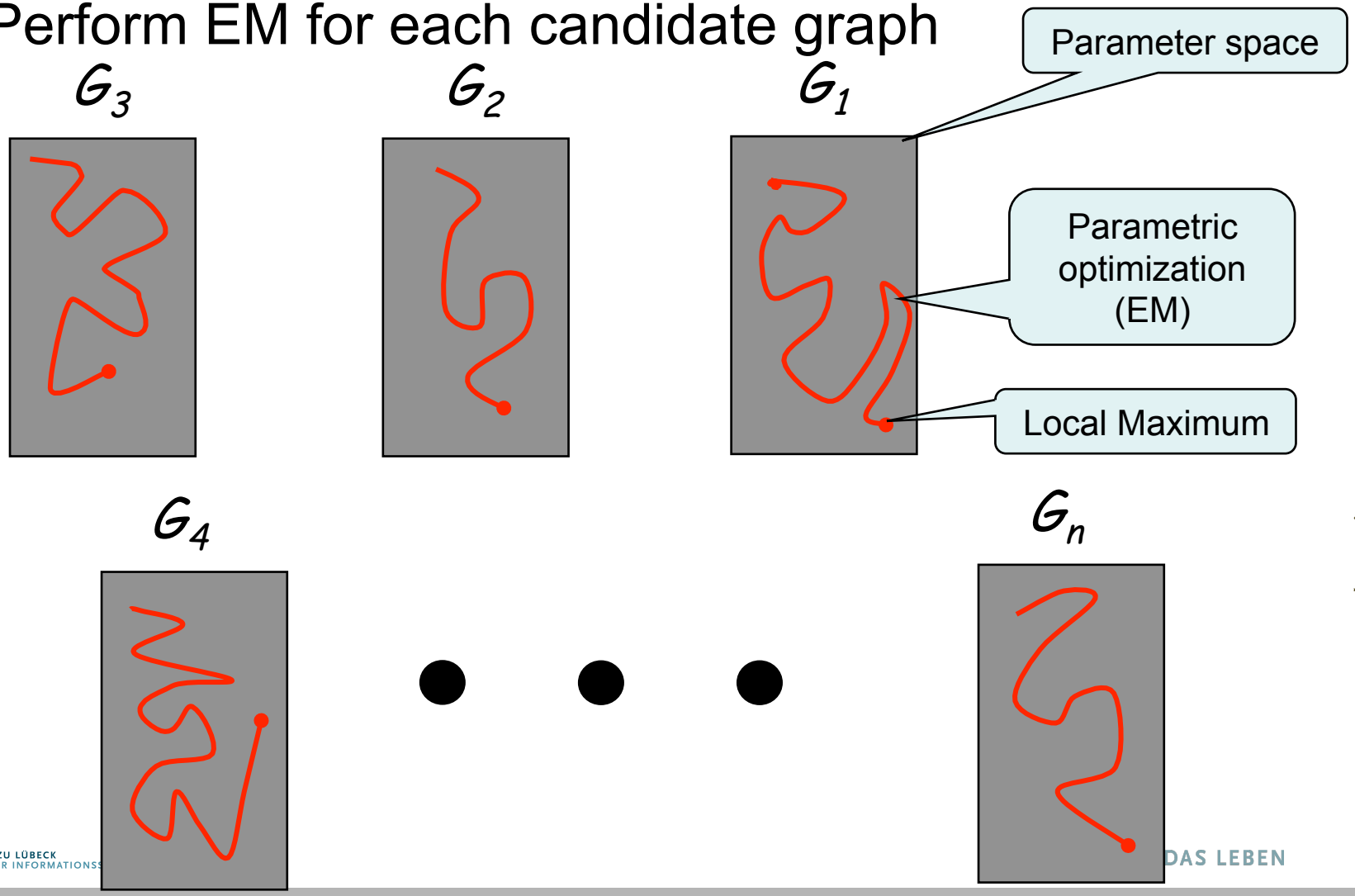
To recompute scores,
only need to re-score families
that changed in the last move

That's fine.
But what to do if we have
non-complete data??

Cannot use decomposability

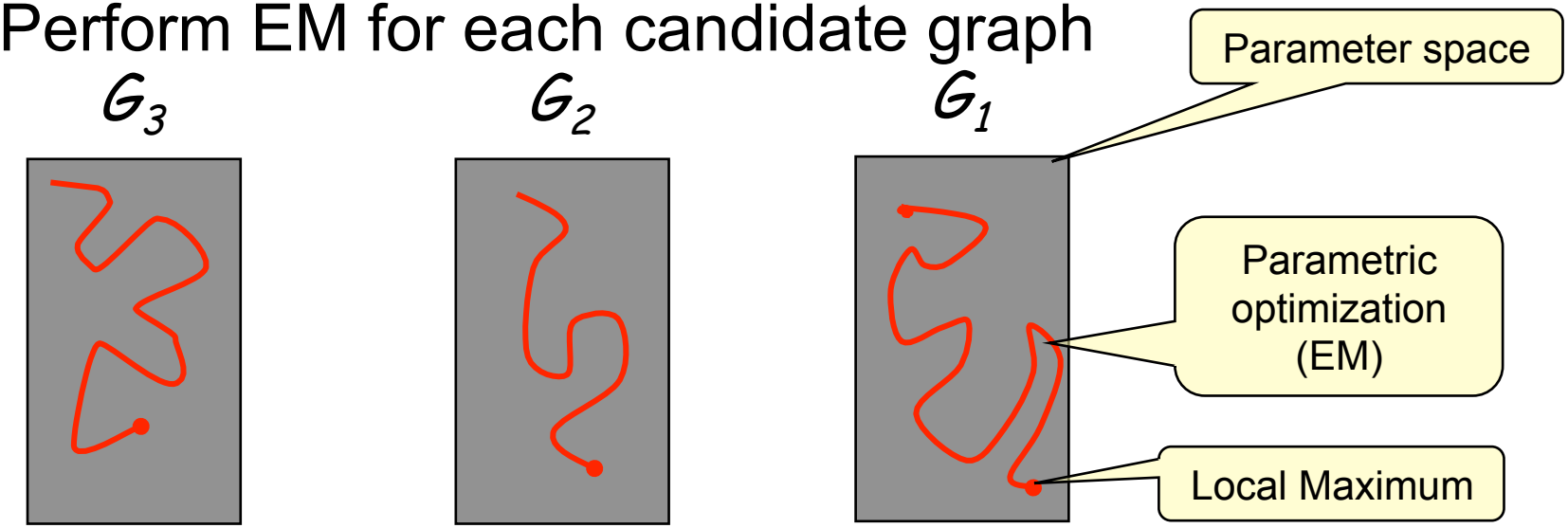
Local Search in Practice

- Perform EM for each candidate graph



Local Search in Practice

- Perform EM for each candidate graph



- Computationally expensive:
  - Parameter optimization via EM — non-trivial
  - Need to perform EM for all candidate structures
  - Spend time even on poor candidates

⇒ In practice, considers only a few candidates

Structural EM [Friedman et al. 98]

Recall, in complete data we had

– Decomposition ⇒ efficient search

Idea:

- Instead of optimizing the real score…
- Find decomposable alternative score
- Such that maximizing new score

⇒ improvement in real score

Structural EM

Idea:

- Use current model to help evaluate new structures

Outline:

- Perform search in (Structure, Parameters) space
- At each iteration, use current model for finding either:
  - Better scoring parameters: “parametric” EM step

  or
  - Better scoring structure: “structural” EM step

- Score for structure G and parameterization Θ given data over observed RVs O

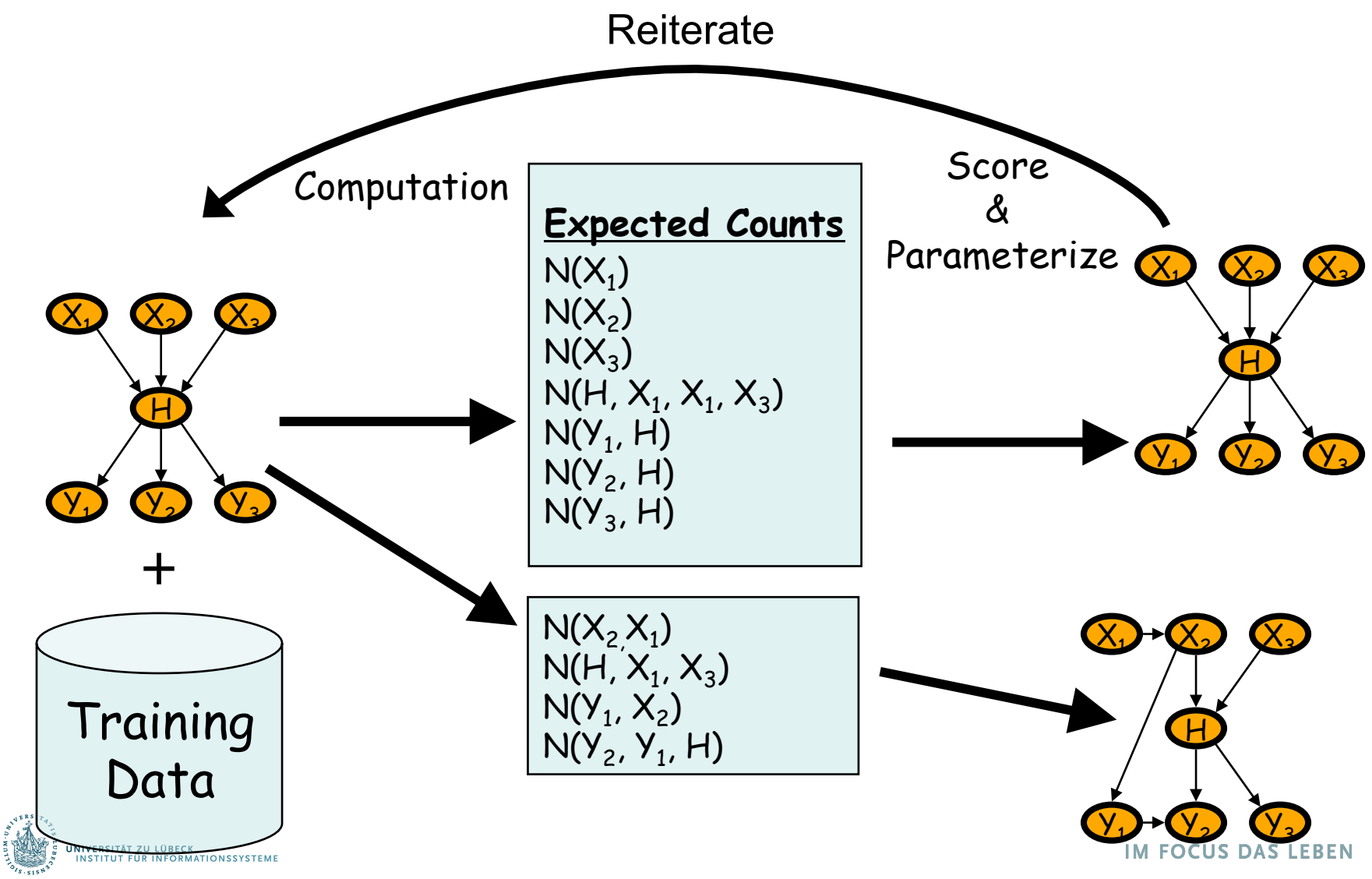
$$\text{Score}_O(G, \Theta) = \log P(O:G,\Theta) - \text{Pen}(G, \Theta, O)$$

- Handle hiddens H (non-observed) by conditionalizing (like in full bayesian learning) using current structure G* and parameterization Θ*

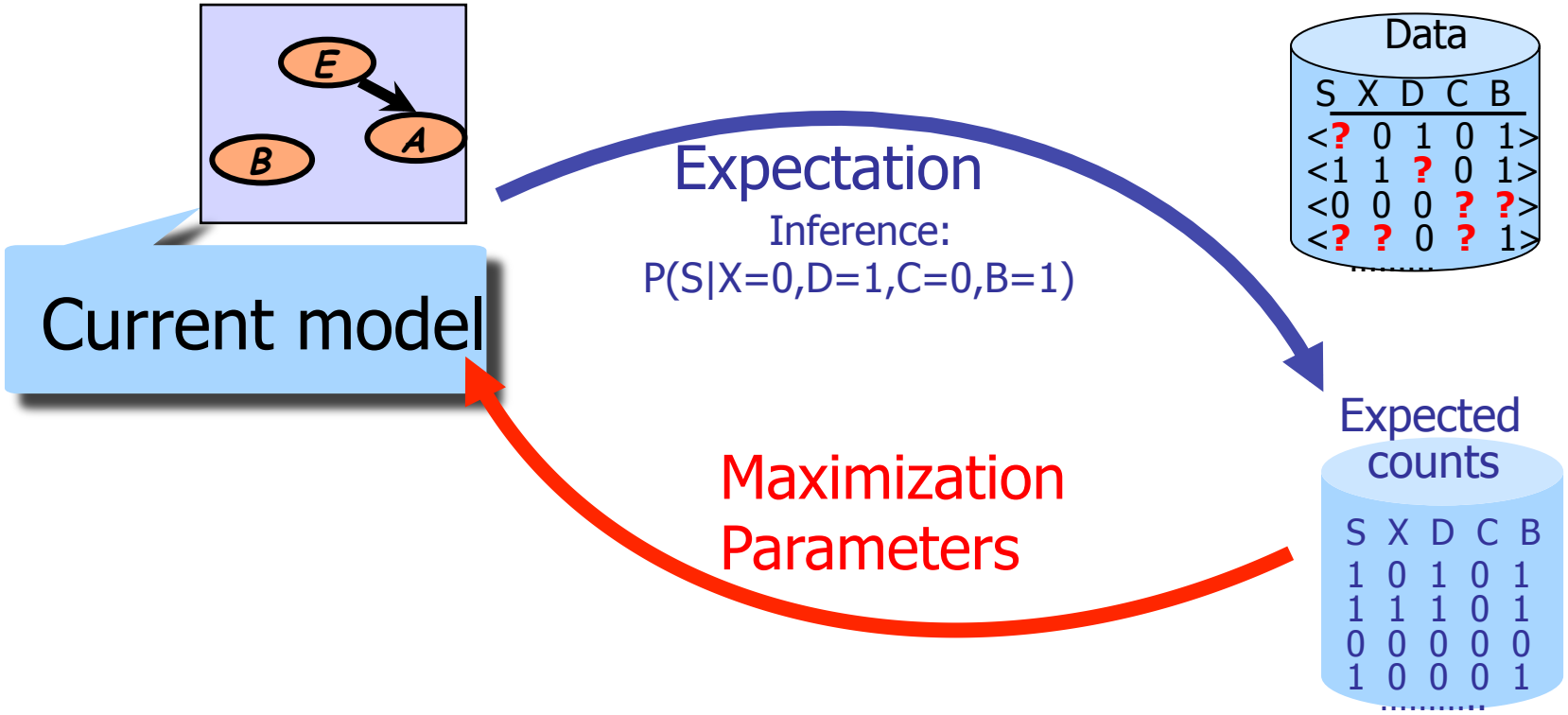
$$Q(G, \Theta : G^*, \Theta^*) = E_{h \sim P(h|O:G^*, \Theta^*)}[\log P(O,h:G, \Theta)] - \text{Pen}(G, \Theta, O)$$

Alternating model selecetion EM
Choose G^0 and Θ^0 randomly
Loop for $n = 0,1, \ldots$ until convergence
  Find G^{n+1}, Θ^{n+1} s.t. $Q(G^{n+1}, \Theta^{n+1}: G^n, \Theta^n) > Q(G^n, \Theta^n: G^n, \Theta^n)$

Structural EM

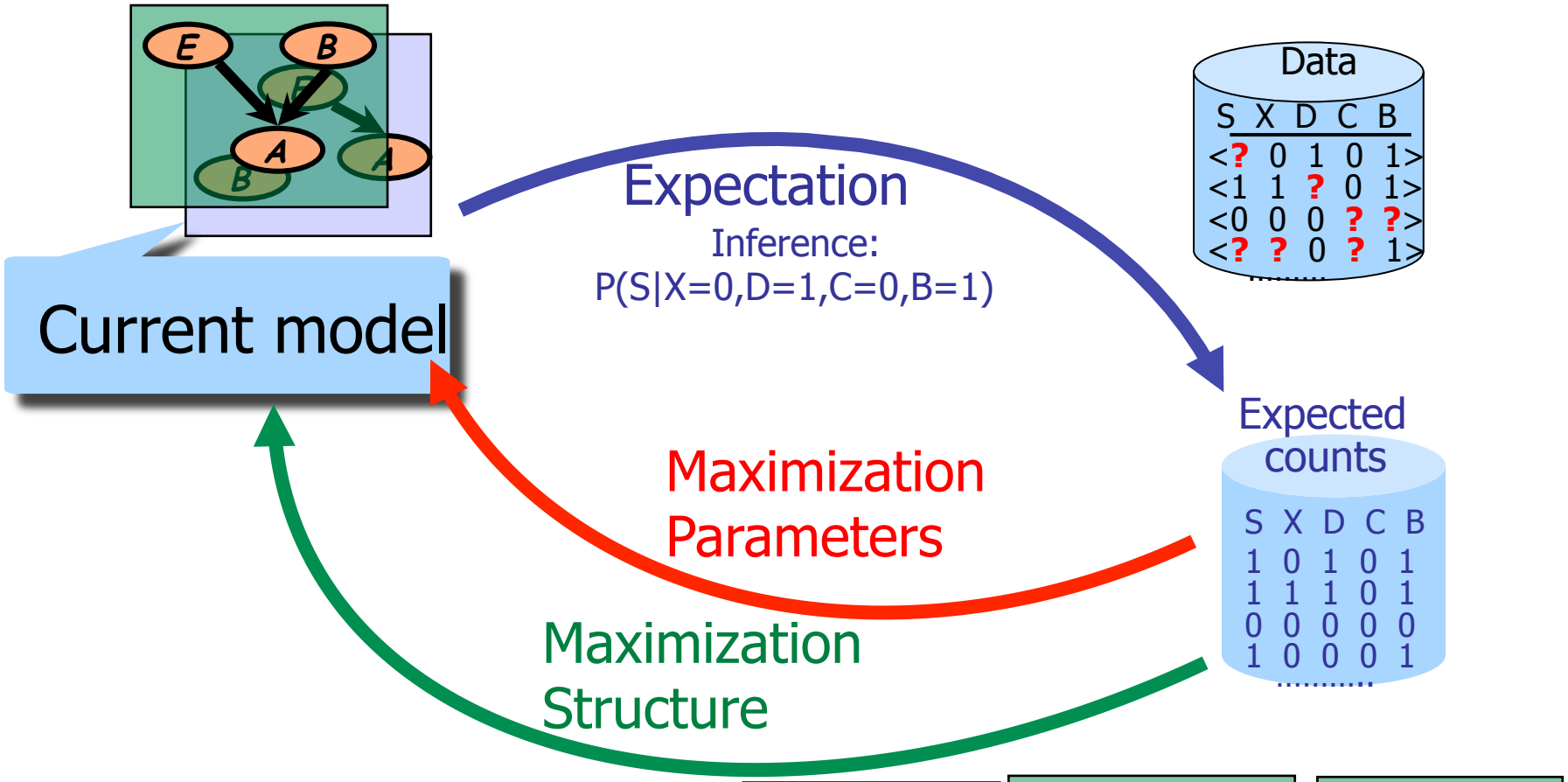


Structure Learning: incomplete data

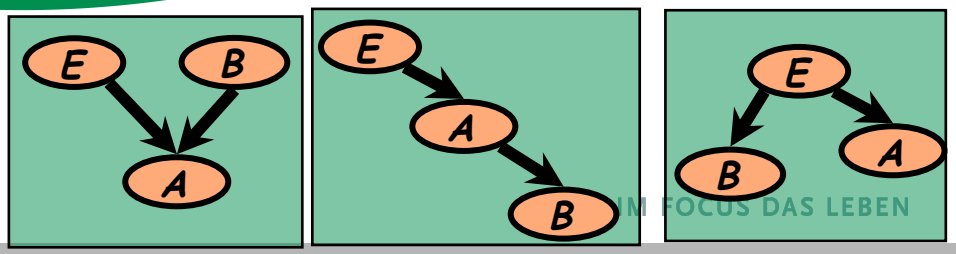


EM-algorithm:
iterate until convergence

Structure Learning: incomplete data



SEM-algorithm:
iterate until convergence



Variations on a theme

- **Known structure, fully observable**: only need to do parameter estimation
- **Known structure, hidden variables:** use expectation maximization (EM) to estimate parameters
- **Unknown structure, fully observable:** do heuristic search through structure space, then parameter estimation
- **Unknown structure, hidden variables:** structural EM

Though we have a procedure for learning parameters and structure, the resulting structure is not -prima facie – guaranteed to reflect causal relationships.
(Remember burglary example with ``wrong“ non-causal order of RVs.)
This motivates considering other algorithms and – of course – considering causality in more detail (see next lectures)