Lecture 11: Bayesian Networks – More on Inference, Maybe Learning

- **Homework 2 due NOW!**

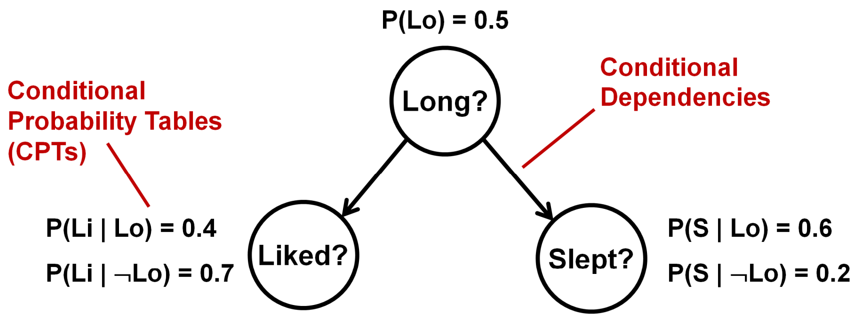
- **Homework 3 out this evening**
- **Due MONDAY, Oct 12th**

(inspect HW3)

Lecture 11: Bayesian Networks – More on Inference, Maybe Learning

Bayesian Networks

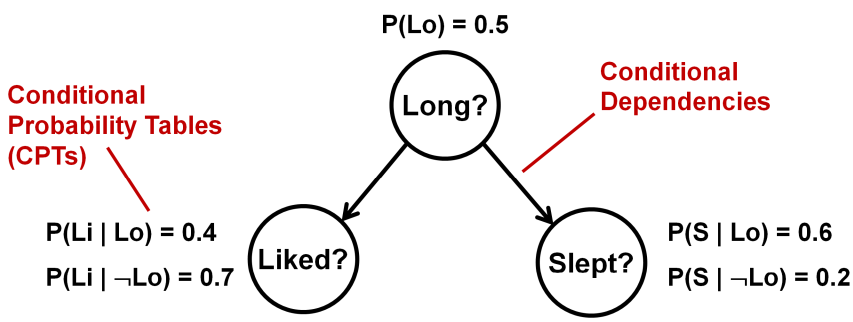
Bayesian networks are directed acyclic graphs with nodes representing random variables and edges representing dependency assumptions



Bayesian Networks

Joint distribution (factorization):

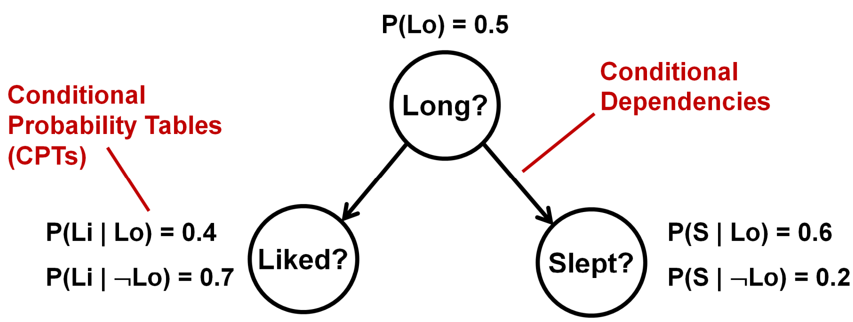
P(Lo, Li, S) = P(Lo) P(Li | Lo) P(S)



Bayesian Networks

Conditional independence:

P(Li | Lo, S)?

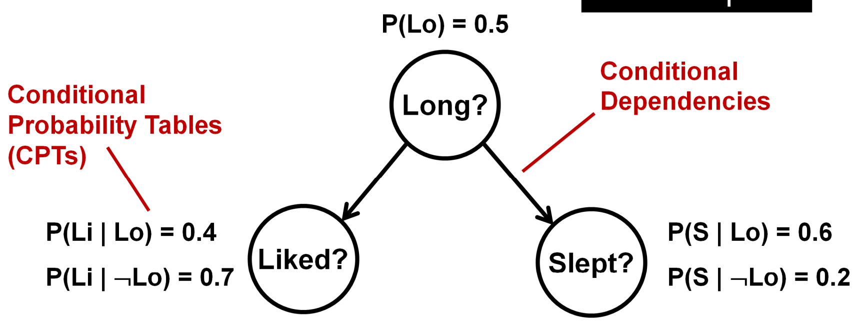


Bayesian Networks

Conditional independence:

$$P(Li \mid Lo, S) = P(Li, Lo, S) / P(Lo, S)$$
$$= P(Li|Lo)\, P(Lo)\, P(S) / P(Lo)\, P(S|Lo)$$
$$= P(Li|Lo)$$

$Li \perp S \mid Lo$



Bayesian networks: Inference

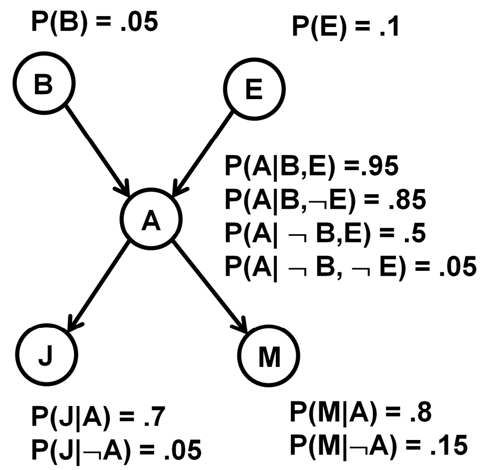
- Algorithms for inferring the values of unobserved variables.
- Last time: Sampling

Stochastic Inference

We can easily sample the joint distribution to obtain possible instances:

1. Sample the free variables
2. For every other variable: If all parents have been sampled, sample based on conditional distribution

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



Weighted Sampl
for Computing

Problem: What if the condition rarely happens?

We would need lots and lots of samples, and most would be wasted

- **Set N_B,N_c = 0**
- **Repeat:**
  - Sample the joint setting the values for *J* and *M*, compute the weight, *w,* of this sample
  - $N_c = N_c + w$
  - If $B = 1$, $N_B = N_B + w$
- **After many iterations, set:**

$$\mathbf{P(B \mid J, \neg M) = N_B / N_c}$$

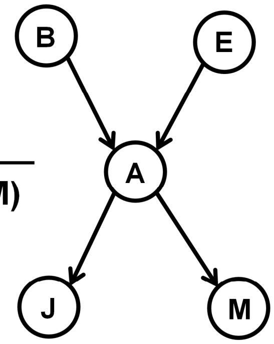
Bayesian networks: Inference

- Algorithms for inferring the values of unobserved variables.

- Last time: Sampling
  - fast, (often) approximate
- Last time: Exact inference

Inference

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



This can also be written as a joint:

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

How do we compute the new joint?

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

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

Computing: $P(B,J,\neg M)$

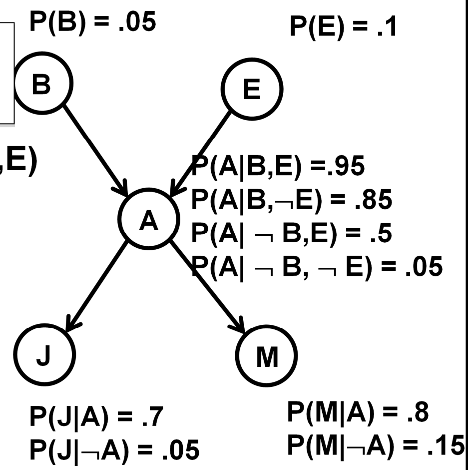
$$P(B, J, \neg M) = \sum_a \sum_e P(B, J, \neg M, a, e)$$

$= P(B,J,\neg M,A,E) + P(B,J,\neg M,\neg A,E) + P(B,J,\neg M,A,\neg E) + P(B,J,\neg M,\neg A,\neg E)$

$= 0.0007+0.00001+0.005+0.0003$

$= 0.00601$

How can we reuse computations?



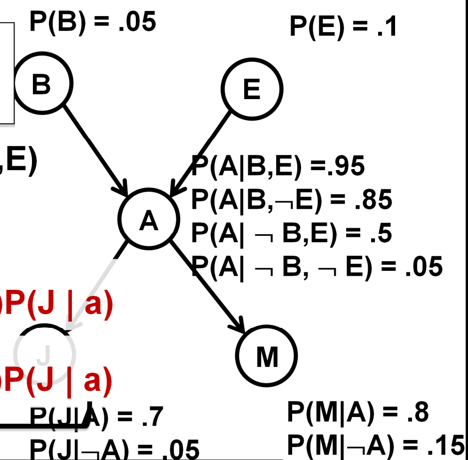
Computing: P(B,J, ¬M)

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

$= P(B,J,\neg M,A,E) + P(B,J,\neg M,\neg A,E)$
$+ P(B,J,\neg M,A,\neg E) +$
$P(B,J,\neg M,\neg A,\neg E)$

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

$$= P(B)\sum_e P(e)\underbrace{\sum_a P(a \mid B,e)P(\neg M \mid a)P(J \mid a)}$$



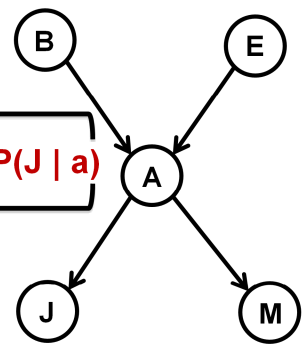
"sum out" A, and store as a function of e and use whenever necessary (no need to recompute for each value of e)

Instead of computing the value for every value of e

Computing: $P(B,J,\neg M)$

"factors"

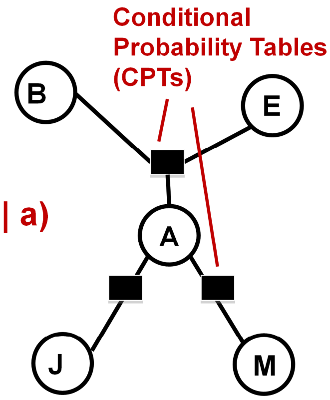
$$P(B)\overbrace{\sum_e P(e)\underbrace{\sum_a P(a \mid B,e)P(\neg M \mid a)P(J \mid a)}_{f_E(e)}}^{f()}$$



"variable elimination"

Computing: P(B,J, ¬M)

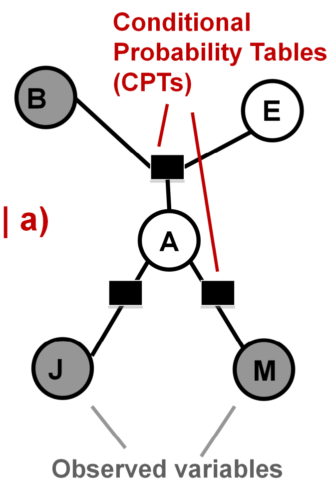
$$P(B)\sum_{e} P(e)\sum_{a} P(a \mid B,e)P(\neg M \mid a)P(J \mid a)$$



"variable elimination"

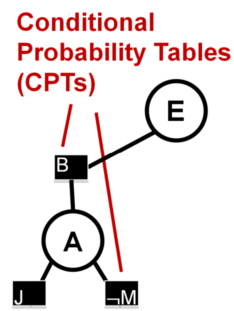
Computing: P(B,J, ¬M)

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



“variable elimination”

Computing: P(B,J, ¬M)

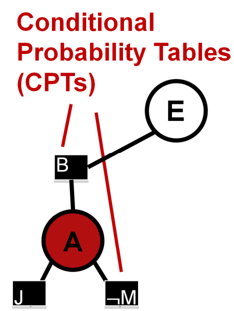


$$P(B)\sum_{e} P(e)\sum_{a} P(a \mid B,e)P(\neg M \mid a)P(J \mid a)$$

Observed variables don’t need to be eliminated (summed out)

“variable elimination”

Computing: P(B,J, ¬M)

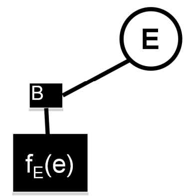


$$P(B)\sum_e P(e)\underbrace{\sum_a P(a \mid B,e)P(\neg M \mid a)P(J \mid a)}_{f_E(e)}$$

“variable elimination”

Computing: P(B,J, ¬M)

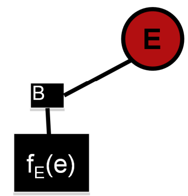
$P(B)\sum_e P(e)\underbrace{\sum_a P(a \mid B,e)P(\neg M \mid a)P(J \mid a)}_{f_E(e)}$



“variable elimination”

Computing: P(B,J, ¬M)

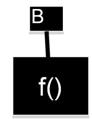
$$\overbrace{P(B)\sum_{e} P(e)\underbrace{\sum_{a} P(a \mid B,e)P(\neg M \mid a)P(J \mid a)}_{f_E(e)}}^{f()}$$



"variable elimination"

Computing: P(B,J, ¬M)

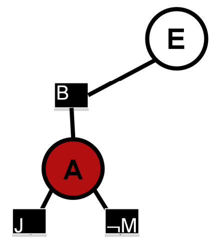
$P(B)\sum_{e} P(e)\sum_{a} P(a \mid B,$ **f()** $P(M \mid a)P(J \mid a)$



“variable elimination”

Actually Computing $f_E(e)$

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



$$P(B)\sum_e P(e)\underbrace{\sum_a P(a \mid B,e)\overbrace{P(\neg M \mid a)P(J \mid a)}}_{f_E(e)}$$

$$f_E(e) = \begin{pmatrix} P(A|B,E)\ f_A(A) \\ +\ P(\neg A|B,E)\ f_A(\neg A) \\ \hline P(A|B,\neg E)\ f_A(A) \\ +\ P(\neg A|B,\neg E)\ f_A(A) \end{pmatrix}$$

Reuse,
don't recompute!

btw, we computed P(B,J,¬M), but wanted P(B|J, ¬M)

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

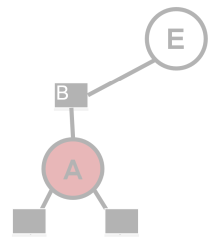
Also need to compute, but can reuse some computation again!

“normalization”

Actually Computing $f_E(e)$

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

$$P(B)\sum_e P(e)\underbrace{\sum_a P(a \mid B,e)\overbrace{P(\neg M \mid a)P(J \mid a)}^{f_A(a)}}_{f_E(e)}$$



$$f_E(e) = \begin{pmatrix} P(A|B,E)\ f_A(A) \\ +\ P(\neg A|B,E)\ f_A(\neg A) \\ \hline P(A|B,\neg E)\ f_A(A) \\ +\ P(\neg A|B,\neg E)\ f_A(A) \end{pmatrix}$$

Reuse,
don't recompute!

Algorithm

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

- **For each *var* in *vars***

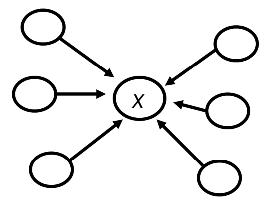
  - *factors* <- make_factor (*var,e*)

  - if *var* is a hidden variable then create a new factor by summing out *var*
- **Compute the product of all factors**
- **Normalize**

Computational Complexity

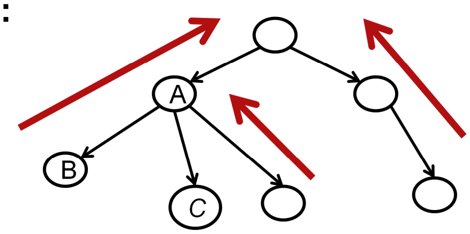
- **We can reuse computations to reduce the running time**
- **However, there are still cases in which this algorithm will lead to exponential running time.**
  - Exact Bayesian Inference is NP-Hard
- **Consider the case of $f_x(y_1 \ldots y_n)$. When factoring x out we would need to account for all possible values of the y's.**
- **e.g. binary:**

$$f_x(y_1 \ldots y_n) = \begin{pmatrix} f_x(0, \ldots 0) \\ \ldots \\ f_x(1, \ldots 1) \end{pmatrix} \leftarrow 2^n \text{ values}$$



Computational Complexity

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



$\sum_B P(B|A) \rightarrow f_1(A)$

$\sum_C P(C|A) \rightarrow f_2(A)$

→ never get functions (factors) with more than 1 argument (size 2)

Bayesian networks: Inference

- Algorithms for inferring the values of unobserved variables.

- Last time: Sampling
  - fast, (often) approximate
- Last time: Exact inference
  - variable elimination
- Also: "belief propagation", "variational inference"

BP on trees = variable elimination

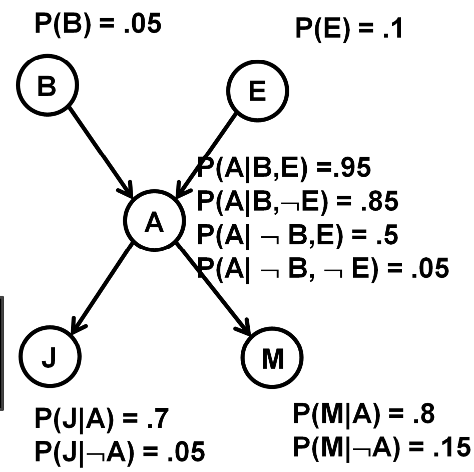
General DAGs need to be

Inference: compute probabilities from CPTs

But where do we get them?

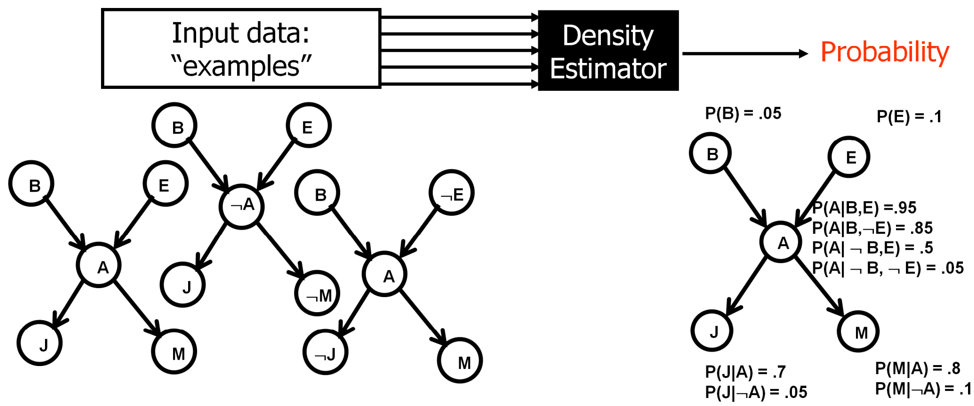
Density estimation

“learning” parameters



Density Estimation

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



Density estimation

- **Binary and discrete variables:**

  Easy: Just count!

- **Continuous variables:**

  Harder (but just a bit): Fit a model

Learning a density estimator

a variable

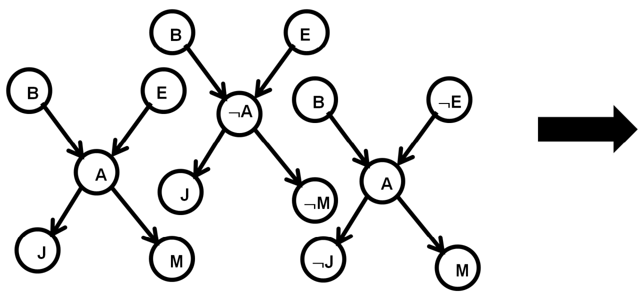
$$\hat{P}(y_i = u) = \frac{\#\,\text{examples in which } y_i = u}{\text{total number of examples}}$$

A trivial learning algorithm!

Learning a density estimator

a variable

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

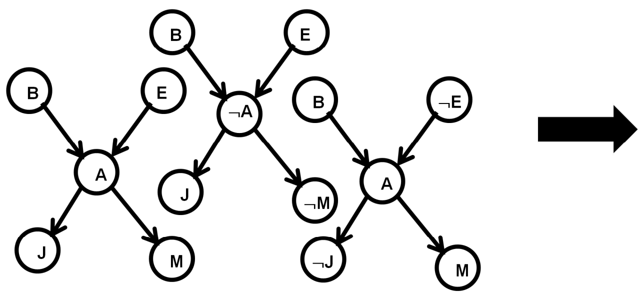


P(B)=
P(E)=
P(A|B,E)=
…

Learning a density estimator

a variable

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



P(B)=1
P(E)=2/3
P(A|B,E)=1/2
…

Maximum Likelihood Principle

examples

$$\hat{P}(\text{dataset}|M) = \hat{P}(\mathbf{x}_1 \wedge \mathbf{x}_2 \ldots \wedge \mathbf{x}_R|M) = \prod_{k=1}^{R} \hat{P}(\mathbf{x}_k|M)$$

Model: CPTs, net structure, gaussian parameters, …

• Fit models by maximizing the probability of generating the observed samples:

$$L(x_1, \ldots, x_n \mid \Theta) = \underbrace{p(x_1 \mid \Theta)}_{\text{e.g. “joint probability” from a CPT}} \ldots p(x_n \mid \Theta)$$

• The examples are assumed to be independent

• For a binary random variable A with P(A=1)=q

  argmax_q Likelihood = #(A=1)/#examples

• Why?

Maximum Likelihood Principle

- For a binary random variable A with P(A=1)=q
  $\arg\max_q$ Likelihood = #(A=1)/#examples
- Why?

n_1: #examples w/ A=1
n_2: #examples w/ A=0

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

We would like to find: $\arg\max_q q^{n_1}(1-q)^{n_2}$

How?

Maximum Likelihood Principle

Data likelihood: $P(D|q) = q^{n_1}(1-q)^{n_2}$

We would like to find: $\arg\max_q q^{n_1}(1-q)^{n_2}$

$$\frac{\partial}{\partial q} q^{n_1}(1-q)^{n_2} = n_1 q^{n_1-1}(1-q)^{n_2} - q^{n_1} n_2 (1-q)^{n_2-1}$$

$$\boxed{\frac{\partial}{\partial q} = 0} \Rightarrow$$

$$n_1 q^{n_1-1}(1-q)^{n_2} - q^{n_1} n_2 (1-q)^{n_2-1} = 0 \Rightarrow$$

$$q^{n_1-1}(1-q)^{n_2-1}(n_1(1-q) - q n_2) = 0 \Rightarrow$$

$$n_1(1-q) - q n_2 = 0 \Rightarrow$$

$$n_1 = n_1 q + n_2 q \Rightarrow$$

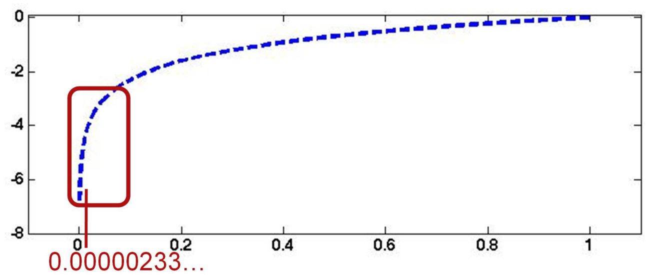
$$q = \frac{n_1}{n_1 + n_2}$$

Log Probabilities

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

$$\log \hat{P}(\text{dataset}|M) = \log \prod_{k=1}^{R} \hat{P}(\mathbf{x}_k|M) = \sum_{k=1}^{R} \log \hat{P}(\mathbf{x}_k|M)$$

Log values between 0 and 1



Maximize that!

Density estimation

- **Binary and discrete variables:**

  Easy: Just count!

- **Continuous variables:**

  Harder (but just a bit): Fit a model

But what if we only have very few samples?

The danger of joint density estimation

P(summer & size ≥ 20 & evaluation = 3) = 0

- No such example in our dataset

Now lets assume we are given a new ("test") dataset. If this dataset contains

Summer	Size	Evaluation
1	30	3

Then the probability we would assign to the *entire* dataset is 0

Summer?	Size	Evaluation
1	19	3
1	17	3
0	49	2
0	33	1
0	55	3
1	20	1

Naïve Density Estimation

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

We need something which generalizes more usefully.

The naïve model generalizes strongly:

Assume that each attribute is distributed independently of any of the other attributes.

Joint estimation, revisited

Assuming independence we can compute each probability independently

P(Summer) = ½ = 0.5

P(Evaluation = 1) = 1/3 = 0.33

P(Size ≥ 20) = 2/3 = 0.66

Summer?	Size	Evaluation
1	19	3
1	17	3
0	49	2
0	33	1
0	55	3
1	20	1

How do we do on the joint?

P(Summer & Evaluation = 1) = 1/6

P(Summer)P(Evaluation = 1) = ½*1/3 = 1/6

P(size ≥ 20 & Evaluation = 1) = 1/3 = 0.33
P(size ≥ 20)P(Evaluation = 1) = 2/3*1/3 = 0.22

Okay

Joint estimation, revisited

Assuming independence we can compute each probability independently

P(Summer) = ½ = 0.5

P(Evaluation = 1) = 1/3 = 0.33

P(Size ≥ 20) = 2/3 = 0.66

How do we do on the joint?

P(Summer & Size ≥ 20) = 1/6 = 0.16667

P(Summer)P(Size ≥ 20) = ½*2/3 = 1/3 = 0.333

Summer?	Size	Evaluation
1	19	3
1	17	3
0	49	2
0	33	1
0	55	3
1	20	1

We must be careful when using the Naïve density estimator

Contrast

Joint DE	Naïve DE
Can model anything	Can model only very boring distributions
No problem to model "C is a noisy copy of A"	Outside Naïve's scope
Given 100 records and more than 6 Boolean attributes will screw up badly	Given 100 records and 10,000 multivalued attributes will be fine

Naïve Density Estimation

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

We need something which generalizes more usefully.

Joint estimator: 2^n-1 parameters

Naïve estimator: n parameters

The naïve model generalizes strongly:

Assume that each attribute is distributed independently of any of the other attributes.

another way to deal with small datasets

- We just discussed one possibility: Naïve estimation

- Assume we want to compute the probability of heads in a coin flip (50/50)

  - What if we can only observe 3 flips?

    1 1 1   0 0 0
    1 1 0   1 0 1
    0 1 1   1 0 0
    0 1 0   0 0 1

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

Pseudo counts

- Use *prior belief* about the 'fairness' of most coins to influence the resulting model.
- We assume that we have "observed" 10 flips with 5 tails and 5 heads
- Thus P(heads) = (#heads+5) / (#flips+10)
- Advantages: 1. Never assign a probability of 0 to an event

  2. As more data accumulates we can get very close to the real distribution (the impact of the pseudo counts will diminish rapidly)

Pseudo counts

- Use *prior belief* about the 'fairness' of most coins to influenc
- We ass 5 tails and 5 h
- Thus P
- Advan nt

  2. As m real distribut apidly)

Sometimes you can even justify this by incorporating a *real* distribution into your model!

Lets go back to Naïve vs.full model

What should I use?

This can be determined based on:

- **Training data size**
- **Cross validation**
- **Likelihood ratio test**

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

Divide up data set into m parts,

train on m-1, test on the 1

(do m times)

→ Which model does better?

Statistically valid!

Important points

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

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