Readings: K&F 10.3, 10.4, 17.1, 17.2

Message Passing Algorithms for Exact Inference & Parameter Learning

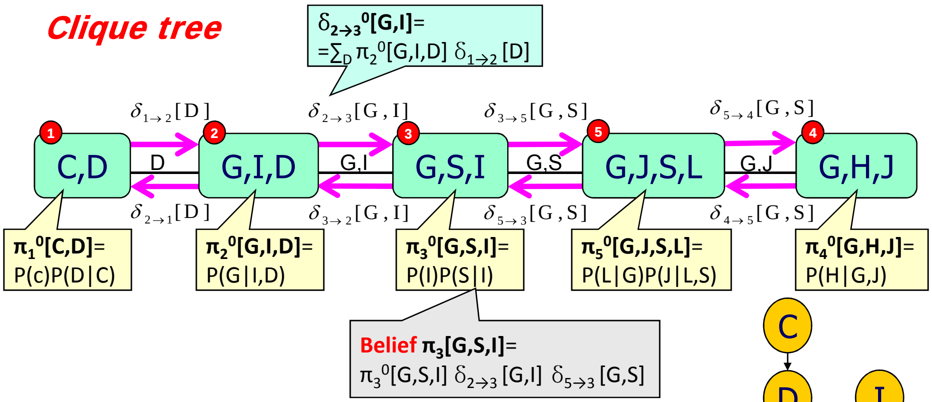
Lecture 8 – Apr 20, 2011
CSE 515, Statistical Methods, Spring 2011

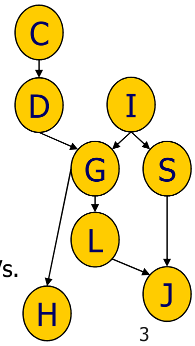
Instructor: Su-In Lee
University of Washington, Seattle

Part I
TWO MESSAGE PASSING ALGORITHMS

Sum-Product Message Passing Algorithm

Clique tree





- Claim: for each clique C_i: $\pi_i[C_i] = P(C_i)$
  - Variable elimination, treating C_i as a root clique
- Compute P(X)
  - Find belief π of a clique that contains X and eliminate other RVs.
  - If X appears in multiple cliques, they must agree

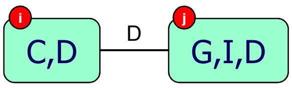
3

Clique Tree Calibration

- A clique tree with potentials $\pi_i[C_i]$ is said to be calibrated if for all neighboring cliques C_i and C_j:

"Sepset belief"

$$\sum_{C_i - S_{i,j}} \pi_i[C_i] = \sum_{C_j - S_{i,j}} \pi_j[C_j]$$



$$\mu_{i,j}(D) = \sum_C \pi_i[C,D] = \sum_{G,I} \pi_j[G,I,D]$$

- Key advantage the clique tree inference algorithm
  - Computes marginal distributions for all variables $P(X_1),\ldots,P(X_n)$ using only twice the computation of the upward pass in the same tree.

4

Calibrated Clique Tree as a Distribution

- At convergence of the clique tree algorithm, we have that:

$$P_\Phi(\mathbf{X}) = \frac{\prod_{C_i \in T} \pi_i[C_i]}{\prod_{(C_i \leftrightarrow C_j)\in T} \mu_{i,j}(S_{i,j})}$$

- Proof:

$$\begin{aligned}
\mu_{i,j}(S_{i,j}) &= \sum_{C_i - S_{i,j}} \pi_i[C_i] = \sum_{C_i - S_{i,j}} \pi_i^0[C_i] \prod_{k \in N_i} \delta_{k\to i} \\
&= \sum_{C_i - S_{i,j}} \pi_i^0[C_i] \delta_{j\to i}(S_{i,j}) \prod_{k \in N_i - \{j\}} \delta_{k\to i} \\
&= \delta_{j\to i}(S_{i,j}) \sum_{C_i - S_{i,j}} \pi_i^0[C_i] \prod_{k \in N_i - \{j\}} \delta_{k\to i} \\
&= \delta_{j\to i}(S_{i,j}) \delta_{i\to j}(S_{i,j})
\end{aligned}$$

Definition

$$\delta_{i\to j} = \sum_{C_i - S_{i,j}} \pi_i^0 \prod_{k \in N_i - \{j\}} \delta_{k\to i}$$

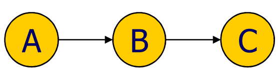
$$\frac{\prod_{C_i \in T} \pi_i[C_i]}{\prod_{(C_i \leftrightarrow C_j)\in T} \mu_{i,j}(S_{i,j})}$$

- Clique tree invariant: The clique beliefs π's and sepset beliefs μ's provide a re-parameterization of the joint distribution, one that directly reveals the marginal distributions.

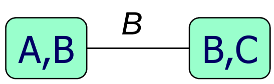
Distribution of Calibrated Tree

- For calibrated tree

Bayesian network



Clique tree



$$P(C \mid B) = \frac{P(B,C)}{P(B)} = \frac{\pi_2[B,C]}{P(B)} = \frac{\pi_2[B,C]}{\mu_{12}[B]}$$

- Joint distribution can thus be written as

$$P(A,B,C) = P(A,B)P(C \mid B) = \frac{\pi_1[A,B]\pi_2[B,C]}{\mu_2[B]}$$

Clique tree invariant

$$P_\Phi(\mathbf{X}) = \frac{\prod_{C_i \in T} \pi_i}{\prod_{(C_i \leftrightarrow C_j)\in T} \mu_{i,j}}$$

An alternative approach for message passing in clique trees?

Message Passing: Belief Propagation

- Recall the clique tree calibration algorithm
  - Upon calibration the final potential (belief) at i is:

$$\pi_i = \pi_i^0 \prod_{k \in N_i} \delta_{k \to i}$$

  - A message from i to j sums out the non-sepset variables from the product of initial potential and all messages except for the one from j to i

$$\delta_{i \to j} = \sum_{C_i - S_{i,j}} \pi_i^0 \prod_{k \in N_i - \{j\}} \delta_{k \to i}$$

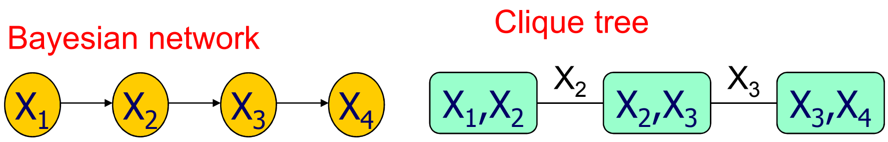
  - Can also be viewed as multiplying all messages and dividing by the message from j to i

"Sepset belief"

$$\delta_{i \to j} = \frac{\sum_{C_i - S_{i,j}} \pi_i^0 \prod_{k \in N_i} \delta_{k \to i}}{\delta_{j \to i}} = \frac{\sum_{C_i - S_{i,j}} \pi_i}{\delta_{j \to i}} \qquad \mu_{i,j}(S_{i,j})$$

    - Forms a basis of an alternative way of computing messages

Message Passing: Belief Propagation



- Root: C_2
- C_1 to C_2 Message: $\delta_{1\to 2}(X_2) = \sum_{X_1} \pi_1^0[X_1, X_2] = \sum_{X_1} P(X_1)P(X_2 \mid X_1)$
- C_2 to C_1 Message: $\delta_{2\to 1}(X_2) = \sum_{X_3} \pi_2^0[X_2, X_3]\delta_{3\to 2}(X_3)$
  - Sum-product message passing

- Alternatively compute $\pi_2[X_2, X_3] = \delta_{1\to 2}(X_2)\delta_{3\to 2}(X_3)\pi_2^0[X_2, X_3]$
- And then: **"Sepset belief"** $\mu_{1,2}(X_2)$

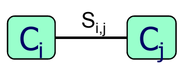
$$\delta_{2\to 1}(X_2) = \frac{\sum_{X_3} \pi_2[X_2, X_3]}{\delta_{1\to 2}(X_2)} = \sum_{X_3} \pi_2^0[X_2, X_3]\delta_{3\to 2}(X_3)$$

→ Thus, the two approaches are equivalent

Message Passing: Belief Propagation

- Based on the observation above,
  - Different message passing scheme, belief propagation
  - Each clique C_i maintains its fully updated beliefs π_i
    - product of initial clique potentials π_i^0 and messages from neighbors $\delta_{k\to i}$
  - Each sepset also maintains its belief $\mu_{i,j}$
    - product of the messages in both direction $\delta_{i\to j}$, $\delta_{j\to i}$
  - The entire message passing process is executed in an equivalent way in terms of the clique and sepset beliefs – π_i's and $\mu_{i,j}$'s.

- Basic idea ($\mu_{i,j}=\delta_{i\to j}\delta_{j\to i}$)



  - Each clique C_i initializes the belief π_i as π_i^0 $(=\prod\phi)$ and then updates it by multiplying with message updates received from its neighbors.
  - Store at each sepset $S_{i,j}$ the previous sepset belief $\mu_{i,j}$ regardless of the direction of the message passed
  - When passing a message from C_i to C_j, divide the new sepset belief $\sigma_{i,j} = \sum_{C_i - S_{i,j}} \pi_i$ by previous $\mu_{i,j}$
  - Update the clique belief π_j by multiplying with $\frac{\sigma_{i,j}}{\mu_{i,j}}$
  - This is called belief update or belief propagation

Message Passing: Belief Propagation

- Initialize the clique tree
  - For each clique C_i set $\pi_i \leftarrow \prod_{\phi:\alpha(\phi)=i} \phi$
  - For each edge C_i—C_j set $\mu_{i,j} \leftarrow 1$
- While uninformed cliques exist
  - Select C_i—C_j
  - Send message from C_i to C_j
    - Marginalize the clique over the sepset $\sigma_{i \to j} \leftarrow \sum_{C_i - S_{i,j}} \pi_i$
    - Update the belief at C_j $\pi_j \leftarrow \pi_j \dfrac{\sigma_{i \to j}}{\mu_{i,j}}$
    - Update the sepset belief at C_i–C_j $\mu_{i,j} \leftarrow \sigma_{i \to j}$
- Equivalent to the sum-product message passing algorithm?
  - Yes – a simple algebraic manipulation, left as PS#2.

Clique Tree Invariant

- Belief propagation can be viewed as reparameterizing the joint distribution
  - Upon calibration we showed $P_\Phi(\mathbf{X}) = \dfrac{\prod_{C_i \in T} \pi_i[C_i]}{\prod_{(C_i \leftrightarrow C_j) \in T} \mu_{i,j}(S_{i,j})}$
    - How can we prove this holds in belief propagation?
  - Initially this invariant holds since $\dfrac{\prod_{C_i \in T} \pi_i[C_i]}{\prod_{(C_i \leftrightarrow C_j) \in T} \mu_{i,j}(S_{i,j})} = \dfrac{\prod_{\phi \in F} \phi}{1} = P_\Phi(\mathbf{X})$
  - At each update step invariant is also maintained
    - Message only changes π_i and $\mu_{i,j}$ so most terms remain unchanged
    - We need to show that for new π', μ' $\dfrac{\pi'_i}{\mu'_{i,j}} = \dfrac{\pi_i}{\mu_{i,j}}$
    - But this is exactly the message passing step $\pi'_i = \dfrac{\mu'_{i,j}\,\pi_i}{\mu_{i,j}}$

→ Belief propagation reparameterizes P at each step

Answering Queries

- Posterior distribution queries on variable X
  - Sum out irrelevant variables from any clique containing X
- Posterior distribution queries on family X,Pa(X)
  - The family preservation property implies that X,Pa(X) are in the same clique.
  - Sum out irrelevant variables from clique containing X,Pa(X)
- Introducing evidence Z=z,
  - Compute posterior of X where X appears in clique with Z
    - Since clique tree is calibrated, multiply clique that contains X and Z with indicator function **I**(Z=z) and sum out irrelevant variables.
  - Compute posterior of X if X does not share a clique with Z
    - Introduce indicator function **I**(Z=z) into some clique containing Z and propagate messages along path to clique containing X
    - Sum out irrelevant factors from clique containing X

$$P_\Phi(\mathbf{X}) = \prod_{\phi \in \Phi} \phi \qquad P_\Phi(\mathbf{X}, Z=z) = \mathbf{1}\{Z=z\} \prod_{\phi \in \Phi} \phi$$

So far, we haven't really discussed how to construct clique trees…

Constructing Clique Trees

- Two basic approaches
  - 1. Based on variable elimination
  - 2. Based on direct graph manipulation

- Using variable elimination
  - The execution of a variable elimination algorithm can be associated with a cluster graph.

  - Create a cluster C_i for each factor used during a VE run
  - Create an edge between C_i and C_j when a factor generated by C_i is used directly by C_j (or vice versa)

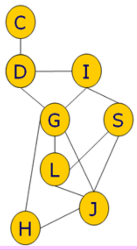
→ We showed that cluster graph is a tree satisfying the running intersection property and thus it is a legal clique tree

Direct Graph Manipulation

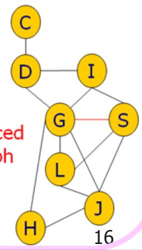
- Goal: construct a tree that is family preserving and obeys the running intersection property
- The induced graph $I_{F,\alpha}$ is necessarily a chordal graph.
  - The converse holds: any chordal graph can be used as the basis for inference.
  - Any chordal graph can be associated with a clique tree (Theorem 4.12)

- Reminder: The induced graph $I_{F,\alpha}$ over factors F and ordering α:
  - Union of all of the graphs resulting from the different steps of the variable elimination algorithm.
  - X_i and X_j are connected if they appeared in the same factor throughout the VE algorithm using α as the ordering

Original graph



Induced graph



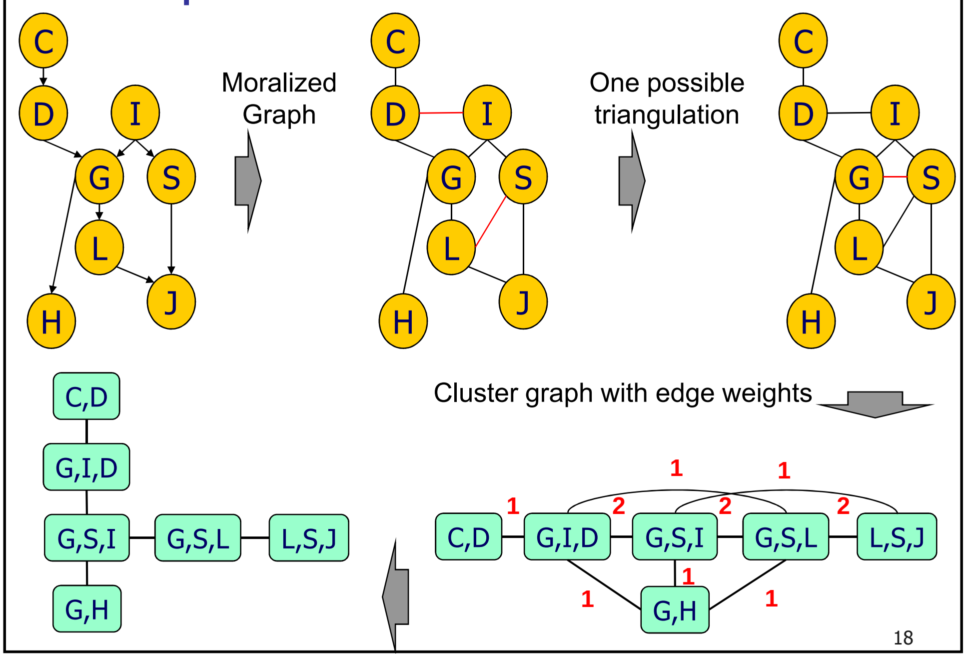
Constructing Clique Trees

- The induced graph $I_{F,\alpha}$ is necessarily a chordal graph.
  - Any chordal graph can be associated with a clique tree (Theorem 4.12)

- Step I: Triangulate the graph to construct a chordal graph H
  - Constructing a chordal graph that subsumes an existing graph H^0
  - NP-hard to find a minimum triangulation where the largest clique in the resulting chordal graph has minimum size
  - Exact algorithms are too expensive and one typically resorts to heuristic algorithms. (e.g. node elimination techniques; see K&F 9.4.3.2)
- Step II: Find cliques in H and make each a node in the clique tree
  - Finding maximal cliques is NP-hard
  - Can begin with a family, each member of which is guaranteed to be a clique, and then use a greedy algorithm that adds nodes to the clique until it no longer induces a fully connected subgraph.
- Step III: Construct a tree over the clique nodes
  - Use maximum spanning tree algorithm on an undirected graph whose nodes are cliques selected above and edge weight is $|C_i \cap C_j|$
  - We can show that resulting graph obeys running intersection → valid clique tree

17

Example



18

Part II

PARAMETER LEARNING

Learning Introduction

- So far, we assumed that the networks were given
- Where do the networks come from?
  - Knowledge engineering with aid of experts
  - Learning: automated construction of networks
    - Learn by examples or instances

Learning Introduction

- Input: dataset of instances D={d[1],...d[m]}
- Output: Bayesian network

- Measures of success
  - How close is the learned network to the original distribution
    - Use distance measures between distributions
    - Often hard because we do not have the true underlying distribution
    - Instead, evaluate performance by how well the network predicts new unseen examples ("test data")
  - Classification accuracy
  - How close is the structure of the network to the true one?
    - Use distance metric between structures
    - Hard because we do not know the true structure
    - Instead, ask whether independencies learned hold in test data

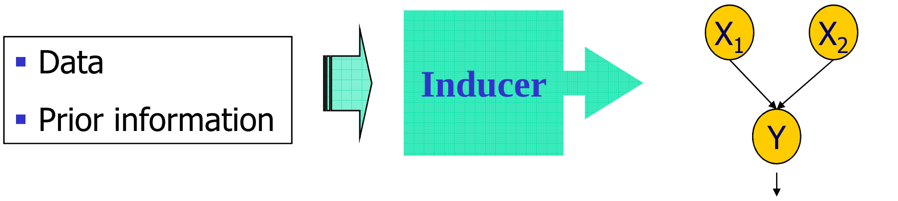
Prior Knowledge

- Prespecified structure
  - Learn only CPDs
- Prespecified variables
  - Learn network structure and CPDs
- Hidden variables
  - Learn hidden variables, structure, and CPDs

- Complete/incomplete data
  - Missing data
  - Unobserved variables

Learning Bayesian Networks

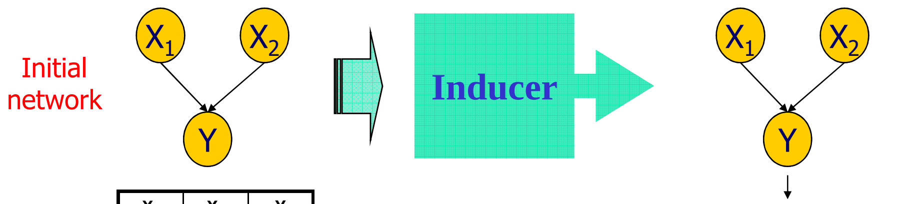
- Four types of problems will be covered



X_1	X_2	$P(Y \mid X_1, X_2)$	
		y^0	y^1
x_1^0	x_2^0	1	0
x_1^0	x_2^1	0.2	0.8
x_1^1	x_2^0	0.1	0.9
x_1^1	x_2^1	0.02	0.98

I. Known Structure, Complete Data

- Goal: Parameter estimation
- Data does not contain missing values



Input Data

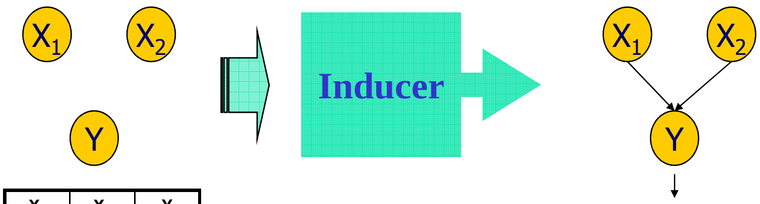
X_1	X_2	Y
x_1^0	x_2^1	y^0
x_1^1	x_2^0	y^0
x_1^0	x_2^1	y^1
x_1^0	x_2^0	y^0
x_1^1	x_2^1	y^1
x_1^0	x_2^1	y^1
x_1^1	x_2^0	y^0

X_1	X_2	$P(Y \mid X_1, X_2)$	
		y^0	y^1
x_1^0	x_2^0	1	0
x_1^0	x_2^1	0.2	0.8
x_1^1	x_2^0	0.1	0.9
x_1^1	x_2^1	0.02	0.98

II. Unknown Structure, Complete Data

- Goal: Structure learning & parameter estimation
- Data does not contain missing values

Initial network



Input Data

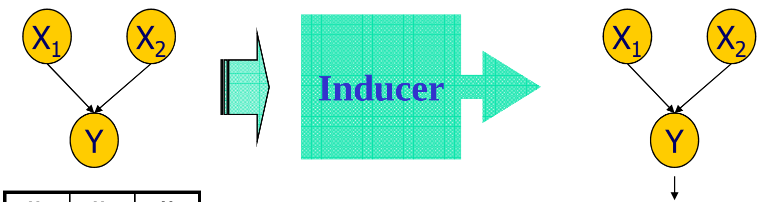
X_1	X_2	Y
x_1^0	x_2^1	y^0
x_1^1	x_2^0	y^0
x_1^0	x_2^1	y^1
x_1^0	x_2^0	y^0
x_1^1	x_2^1	y^1
x_1^0	x_2^1	y^1
x_1^1	x_2^0	y^0

		$P(Y\|X_1,X_2)$	
X_1	X_2	y^0	y^1
x_1^0	x_2^0	1	0
x_1^0	x_2^1	0.2	0.8
x_1^1	x_2^0	0.1	0.9
x_1^1	x_2^1	0.02	0.98

III. Known Structure, Incomplete Data

- Goal: Parameter estimation
- Data contains missing values (e.g. Naïve Bayes)

Initial network



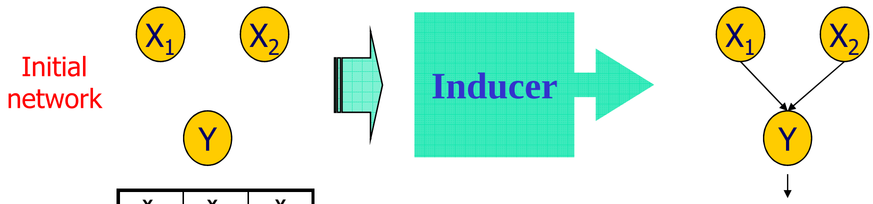
Input Data

X_1	X_2	Y
?	x_2^1	y^0
x_1^1	?	y^0
?	x_2^1	?
x_1^0	x_2^0	y^0
?	x_2^1	y^1
x_1^0	x_2^1	?
x_1^1	?	y^0

		$P(Y\|X_1,X_2)$	
X_1	X_2	y^0	y^1
x_1^0	x_2^0	1	0
x_1^0	x_2^1	0.2	0.8
x_1^1	x_2^0	0.1	0.9
x_1^1	x_2^1	0.02	0.98

IV. Unknown Structure, Incomplete Data

- Goal: Structure learning & parameter estimation
- Data contains missing values



Input Data

X_1	X_2	Y
?	x_2^1	y^0
x_1^1	?	y^0
?	x_2^1	?
x_1^0	x_2^0	y^0
?	x_2^1	y^1
x_1^0	x_2^1	?
x_1^1	?	y^0

		$P(Y\|X_1,X_2)$	
X_1	X_2	y^0	y^1
x_1^0	x_2^0	1	0
x_1^0	x_2^1	0.2	0.8
x_1^1	x_2^0	0.1	0.9
x_1^1	x_2^1	0.02	0.98

Parameter Estimation

- Input
  - Network structure
  - Choice of parametric family for each CPD $P(X_i|Pa(X_i))$
- Goal: Learn CPD parameters
- Two main approaches
  - Maximum likelihood estimation
  - Bayesian approaches

Biased Coin Toss Example

- Coin can land in two positions: Head or Tail

X

- Estimation task
  - Given toss examples x[1],...x[m] estimate P(X=h)= θ and P(X=t)= 1-θ
  - Denote by P(H) and P(T) to mean P(X=h) and P(X=t), respectively.
- Assumption: i.i.d samples
  - Tosses are controlled by an (unknown) parameter θ
  - Tosses are sampled from the same distribution
  - Tosses are independent of each other

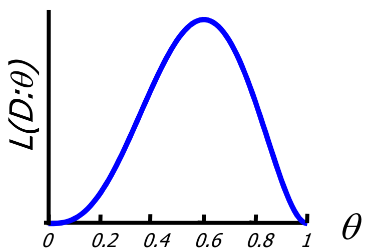
Biased Coin Toss Example

- Goal: find $\theta \in [0,1]$ that predicts the data well
- "Predicts the data well" = likelihood of the data given θ

$$L(D:\theta) = P(D \mid \theta) = \prod_{i=1}^{m} P(x[i] \mid x[1],\ldots,x[i-1],\theta) = \prod_{i=1}^{m} P(x[i] \mid \theta)$$

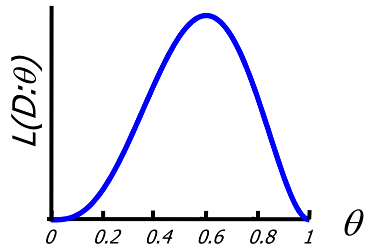
- Example: probability of sequence H,T,T,H,H

$$L(\langle H,T,T,H,H \rangle : \theta) = P(H \mid \theta)P(T \mid \theta)P(T \mid \theta)P(H \mid \theta)P(H \mid \theta) = \theta^3(1-\theta)^2$$



Maximum Likelihood Estimator

- Parameter θ that maximizes L(D:θ)
  - In our example, θ=0.6 maximizes the sequence H,T,T,H,H



Maximum Likelihood Estimator

- General case
  - Observations: M_H heads and M_T tails
  - Find θ maximizing likelihood $L(M_H, M_T : \theta) = \theta^{M_H}(1-\theta)^{M_T}$
  - Equivalent to maximizing log-likelihood

$$l(M_H, M_T : \theta) = M_H \log\theta + M_T \log(1-\theta)$$

  - Differentiating the log-likelihood and solving for θ we get that the maximum likelihood parameter is:

$$\theta_{MLE} = \frac{M_H}{M_H + M_T}$$

Acknowledgement

- These lecture notes were generated based on the slides from Prof Eran Segal.