Bayesian Networks Part 3

Yingyu Liang
Computer Sciences 760
Fall 2017

http://pages.cs.wisc.edu/~yliang/cs760/

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Matt Gormley, Elad Hazan, Tom Dietterich, and Pedro Domingos.

Goals for the lecture

you should understand the following concepts

- structure learning as search
- Kullback-Leibler divergence
- the Sparse Candidate algorithm

- the Tree Augmented Network (TAN) algorithm

Heuristic search for structure learning

- each state in the search space represents a DAG Bayes net structure
- to instantiate a search approach, we need to specify
  - scoring function
  - state transition operators
  - search algorithm

Scoring function decomposability

- when the appropriate priors are used, and all instances in D are complete, the scoring function can be decomposed as follows

$$\text{score}(G, D) = \sum_i \text{score}(X_i, Parents(X_i) : D)$$

- thus we can
  - score a network by summing terms over the nodes in the network
  - efficiently score changes in a *local* search procedure

Scoring functions for structure learning

- Can we find a good structure just by trying to maximize the likelihood of the data?

$$\arg\max_{G,\theta_G} \log P(D \mid G, \theta_G)$$

- If we have a strong restriction on the the structures allowed (e.g. a tree), then maybe.

- Otherwise, no! Adding an edge will never decrease likelihood. Overfitting likely.

Scoring functions for structure learning

- there are many different scoring functions for BN structure search
- one general approach

$$\arg\max_{G,\theta_G} \log P(D \mid G, \theta_G) - \underbrace{f(m)\,|\theta_G|}_{\text{complexity penalty}}$$

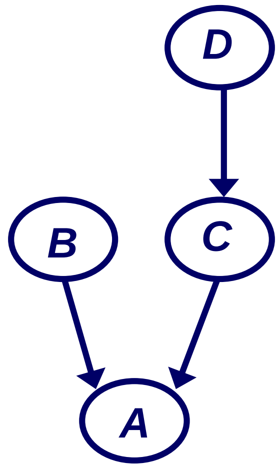
complexity penalty

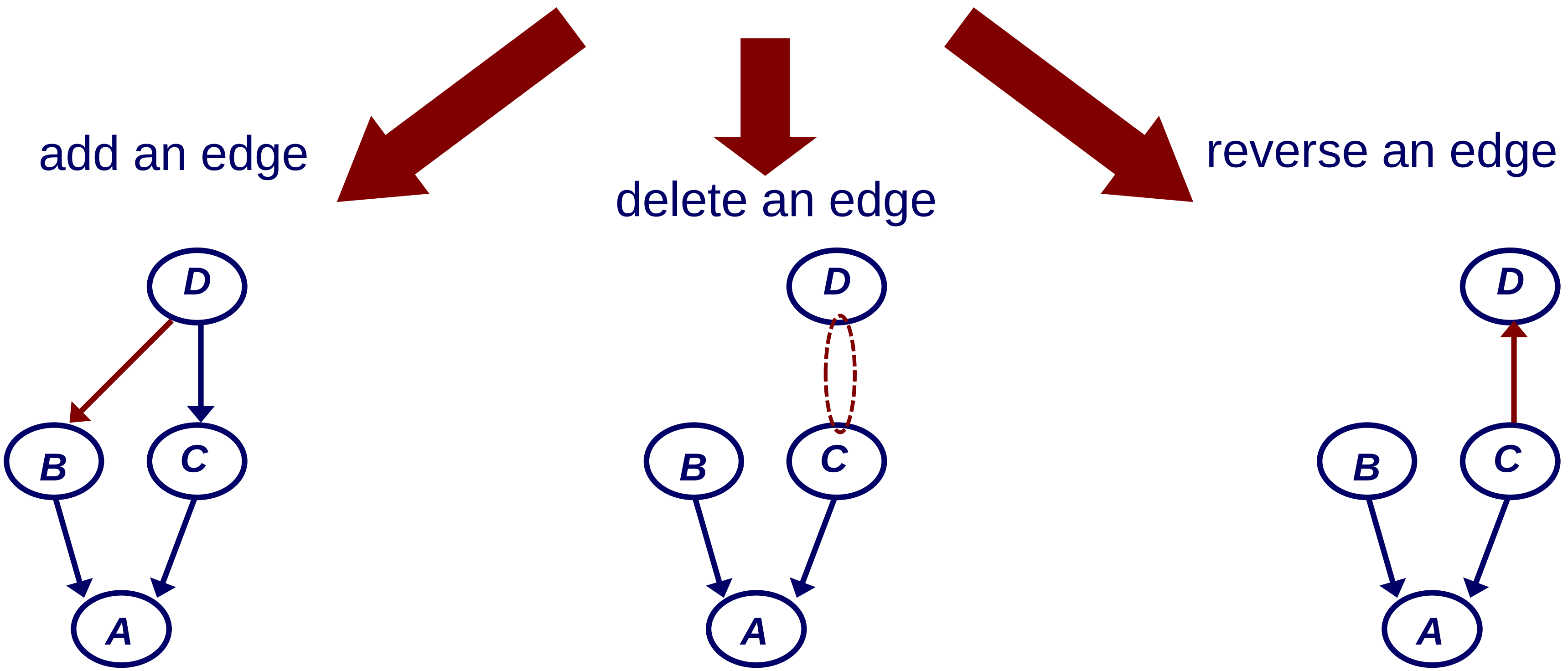
Akaike Information Criterion (AIC): $f(m) = 1$

Bayesian Information Criterion (BIC): $f(m) = \frac{1}{2}\log(m)$

Structure search operators

given the current network
at some stage of the search,
we can…





Bayesian network search: *hill-climbing*

given: data set D, initial network B_0

$i = 0$

$B_{best} \leftarrow B_0$

while stopping criteria not met

{

for each possible operator application a

{

$B_{new} \leftarrow$ apply(a, B_i)

if score(B_{new}) > score(B_{best})

$B_{best} \leftarrow B_{new}$

}

$++i$

$B_i \leftarrow B_{best}$

}

return B_i

Bayesian network search: the *Sparse Candidate* algorithm

[Friedman et al., *UAI* 1999]

given: data set D, initial network B_0, parameter k

$i = 0$

repeat

{

$++i$

// restrict step

select for each variable X_j a set C_j^i of candidate parents ($|C_j^i| \leq k$)

// maximize step

find network B_i maximizing score among networks where $\forall X_j$, Parents(X_j) $\subseteq C_j^i$

} until convergence

return B_i

The *restrict* step in Sparse Candidate

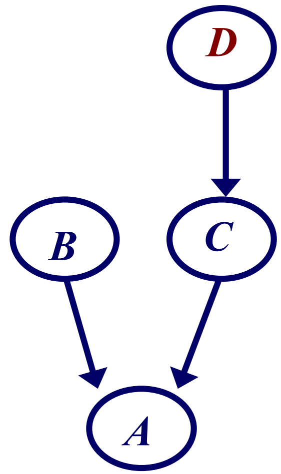
- to identify candidate parents in the first iteration, can compute the *mutual information* between pairs of variables

$$I(X,Y) = \sum_{x \in \text{values}(X)} \sum_{y \in \text{values}(Y)} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

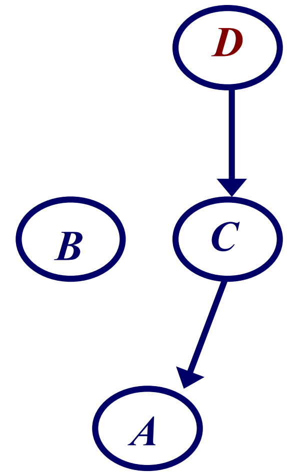
The *restrict* step in Sparse Candidate

- Suppose:

true distribution

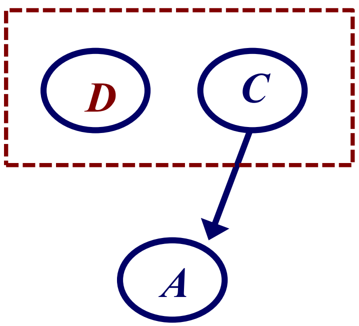


current network



we're selecting two candidate parents for A, and $I(A, C) > I(A, D) > I(A, B)$

- with mutual information, the candidate parents for A would be C and D



- how could we get B as a candidate parent?

The *restrict* step in Sparse Candidate

- *Kullback-Leibler* (KL) *divergence* provides a distance measure between two distributions, P and Q

$$D_{KL}(P(X) \| Q(X)) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

- mutual information can be thought of as the KL divergence between the distributions

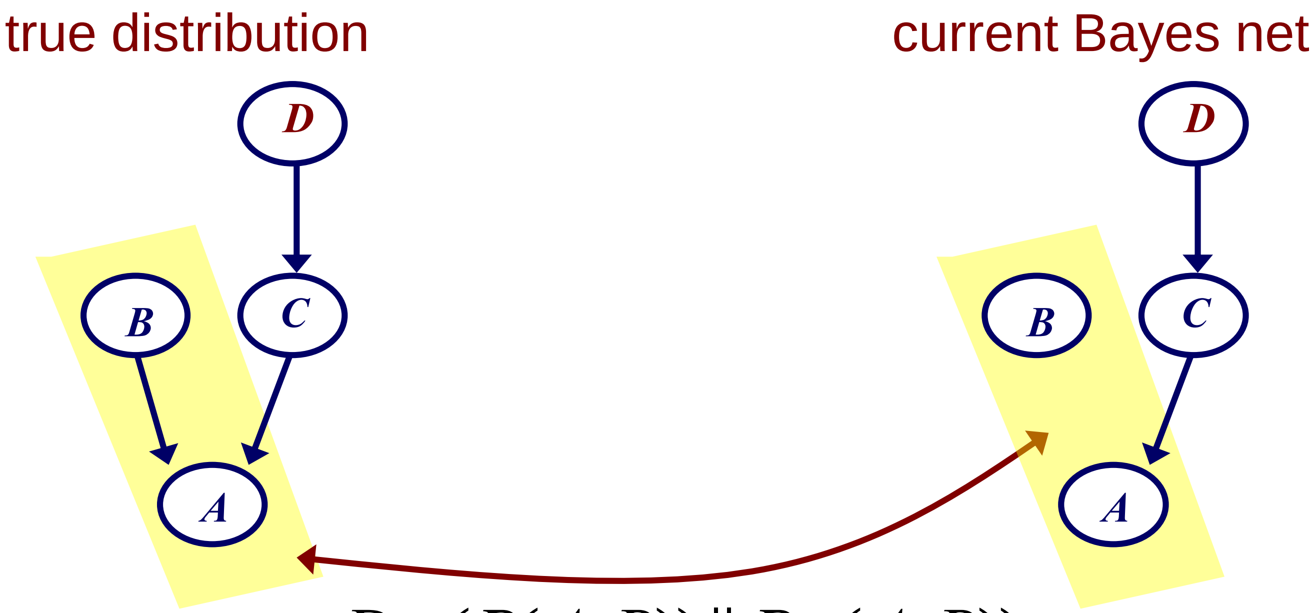
$$P(X,Y)$$

$$P(X)P(Y)$$ (assumes X and Y are independent)

The *restrict* step in Sparse Candidate

- we can use KL to assess the discrepancy between the network's $P_{net}(X, Y)$ and the empirical $P(X, Y)$

$$M(X,Y) = D_{KL}(P(X,Y)) \| P_{net}(X,Y))$$



$$D_{KL}(P(A,B)) \| P_{net}(A,B))$$

- can estimate $P_{net}(X, Y)$ by sampling from the network (i.e. using it to generate instances)

The *restrict* step in Sparse Candidate

given: data set D, current network B_i, parameter k

for each variable X_j

{

calculate M(X_j , X_l) for all $X_j \neq X_l$ such that $X_l \notin$ Parents(X_j)

choose highest ranking $X_1 \ldots X_{k-s}$ where $s = |$ Parents(X_j) $|$

// include current parents in candidate set to ensure monotonic

// improvement in scoring function

$C_j^i =$Parents(X_j) $\cup$ $X_1 \ldots X_{k-s}$

}

return { C_j^i } for all X_j

The *maximize* step in Sparse Candidate

- hill-climbing search with *add-edge*, *delete-edge*, *reverse-edge* operators
- test to ensure that cycles aren't introduced into the graph

Efficiency of Sparse Candidate

n = number of variables

	possible parent sets for each node	changes scored on first iteration of search	changes scored on subsequent iterations
ordinary greedy search	$O\left(2^n\right)$	$O\left(n^2\right)$	$O\left(n\right)$
greedy search w/at most k parents	$O\left(\binom{n}{k}\right)$	$O\left(n^2\right)$	$O\left(n\right)$
Sparse Candidate	$O\left(2^k\right)$	$O\left(kn\right)$	$O\left(k\right)$

after we apply an operator, the scores will change only for edges from the parents of the node with the new impinging edge

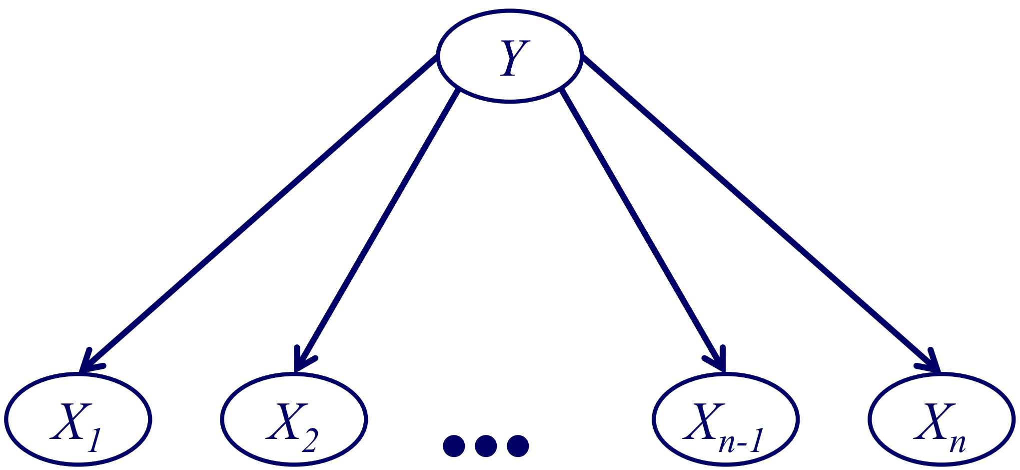
Bayes nets for classification

- the learning methods for BNs we’ve discussed so far can be thought of as being unsupervised
  - the learned models are not constructed to predict the value of a special class variable
  - instead, they can predict values for arbitrarily selected query variables

- now let’s consider BN learning for a standard supervised task (learn a model to predict Y given $X_1 \ldots X_n$)

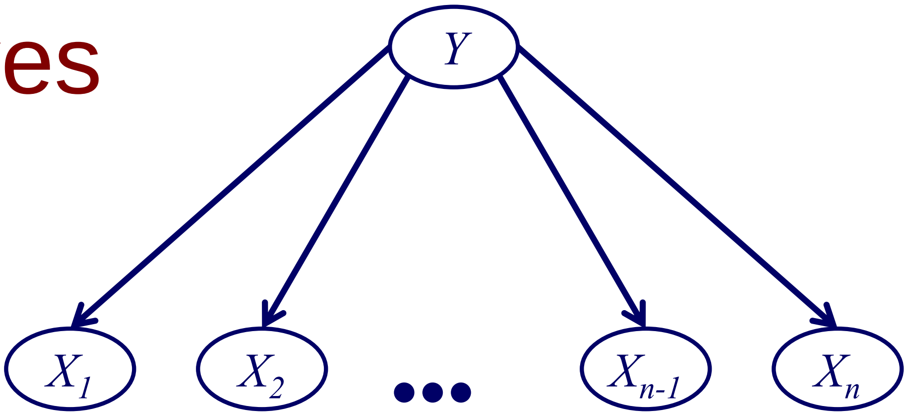
Naïve Bayes

- one very simple BN approach for supervised tasks is *naïve Bayes*
- in naïve Bayes, we assume that all features X_i are conditionally independent given the class Y



$$P(X_1, \ldots, X_n, Y) = P(Y) \prod_{i=1}^{n} P(X_i \mid Y)$$

Naïve Bayes



Learning

- estimate $P(Y=y)$ for each value of the class variable Y
- *estimate* $P(X_i = x \mid Y = y)$ *for each* X_i

Classification: use Bayes' Rule

$$P(Y=y \mid \mathbf{x}) = \frac{P(y)P(\mathbf{x} \mid y)}{\sum_{y'} P(y')P(\mathbf{x} \mid y')} = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{\sum_{y'} \left(P(y')\prod_{i=1}^{n} P(x_i \mid y') \right)}$$

Naïve Bayes vs. BNs learned with an unsupervised structure search

test-set error on 25 classification data sets from the UC-Irvine Repository

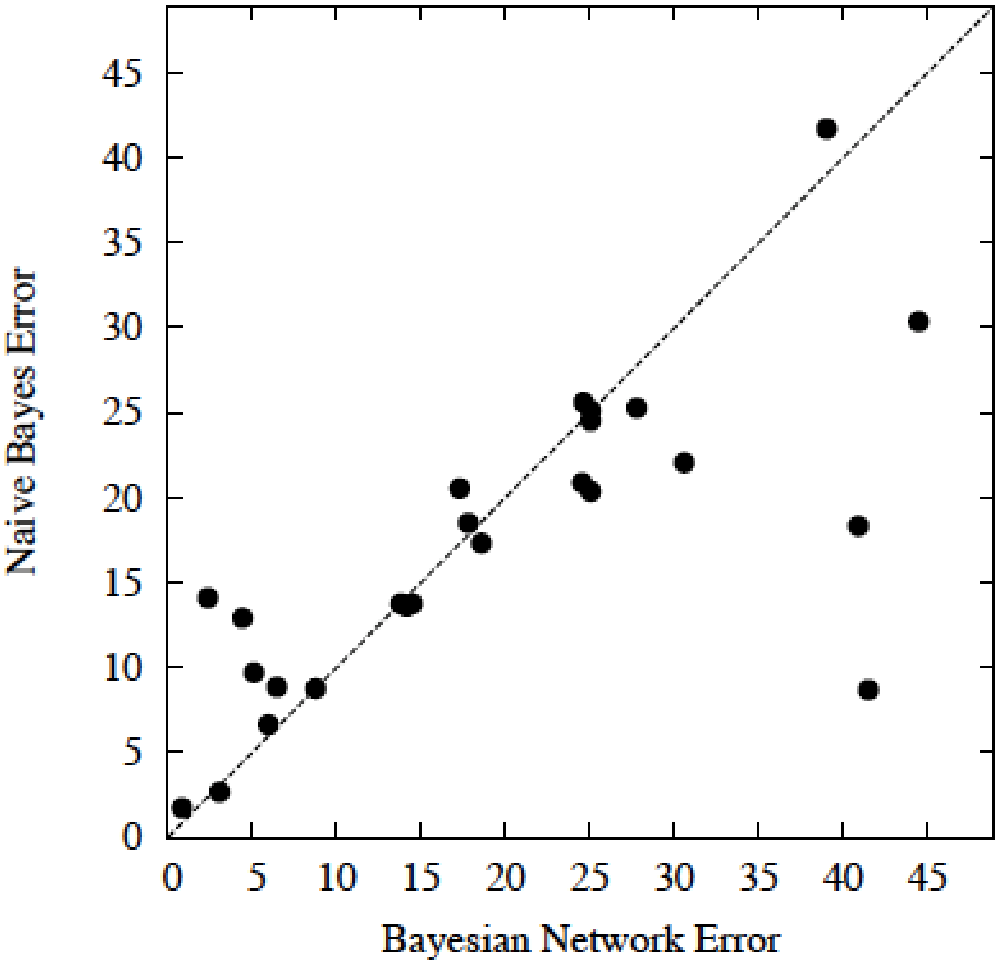


Figure from Friedman et al., *Machine Learning* 1997

The Tree Augmented Network (TAN) algorithm

[Friedman et al., *Machine Learning* 1997]

- learns a <u>tree structure</u> to augment the edges of a naïve Bayes network

- algorithm
  1. compute weight $I(X_i, X_j \mid Y)$ for each possible edge (X_i, X_j) between <u>features</u>
  2. find maximum weight spanning tree (MST) for graph over $X_1 \ldots X_n$
  3. assign edge directions in MST
  4. construct a TAN model by adding node for Y and an edge from Y to each X_i

Conditional mutual information in the TAN algorithm

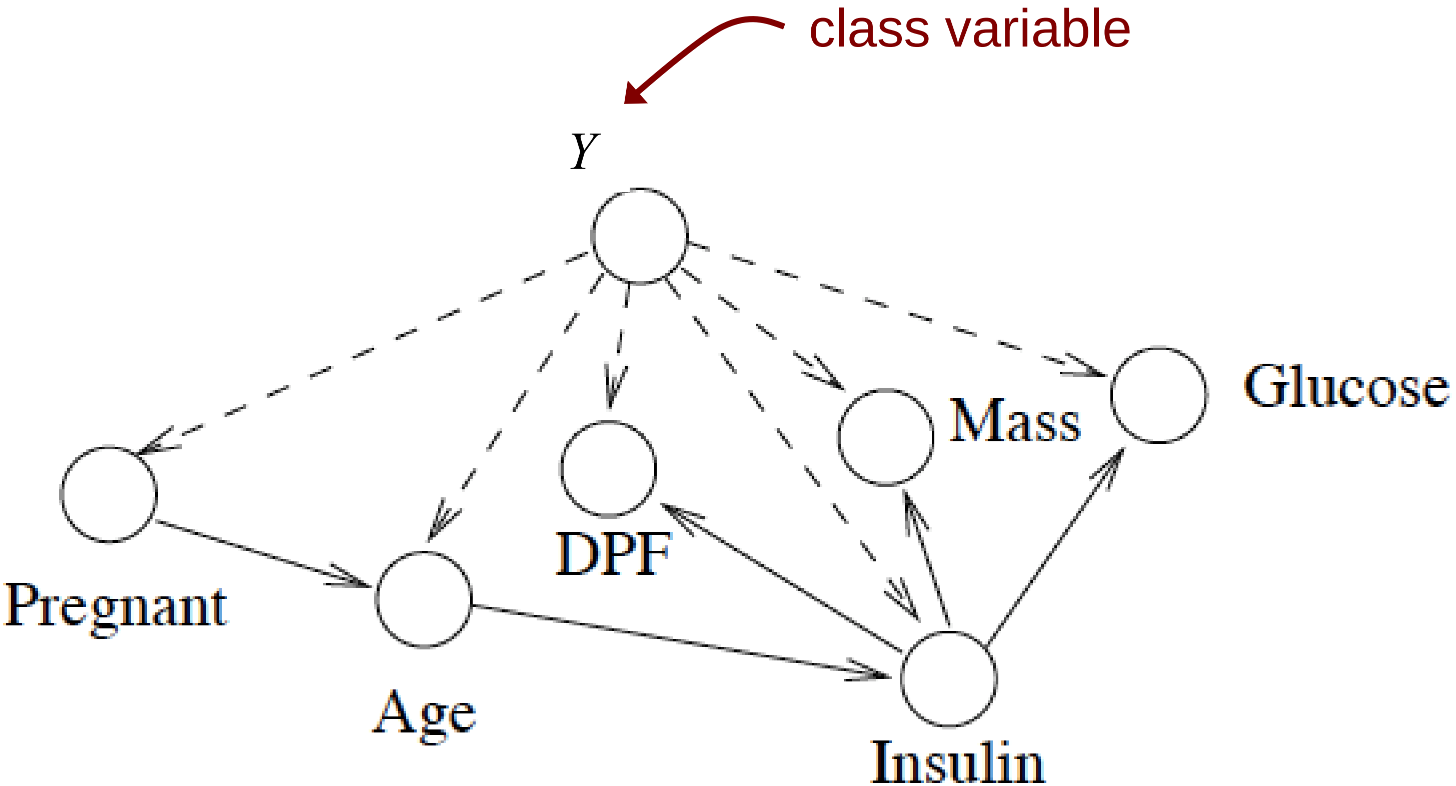
conditional mutual information is used to calculate edge weights

$$I(X_i, X_j \mid Y) =$$

$$\sum_{x_i \in \text{values}(X_i)} \sum_{x_j \in \text{values}(X_j)} \sum_{y \in \text{values}(Y)} P(x_i, x_j, y) \log_2 \frac{P(x_i, x_j \mid y)}{P(x_i \mid y) P(x_j \mid y)}$$

"how much information X_i provides about X_j when the value of Y is known"

Example TAN network



naïve Bayes edges

edges determined by MST

TAN vs. Chow-Liu

- TAN is focused on learning a Bayes net specifically for classification problems
- the MST includes only the feature variables (the class variable is used only for calculating edge weights)
- conditional mutual information is used instead of mutual information in determining edge weights in the undirected graph
- the directed graph determined from the MST is added to the $Y \rightarrow X_i$ edges that are in a naïve Bayes network

TAN vs. Naïve Bayes

test-set error on 25 data sets from the UC-Irvine Repository

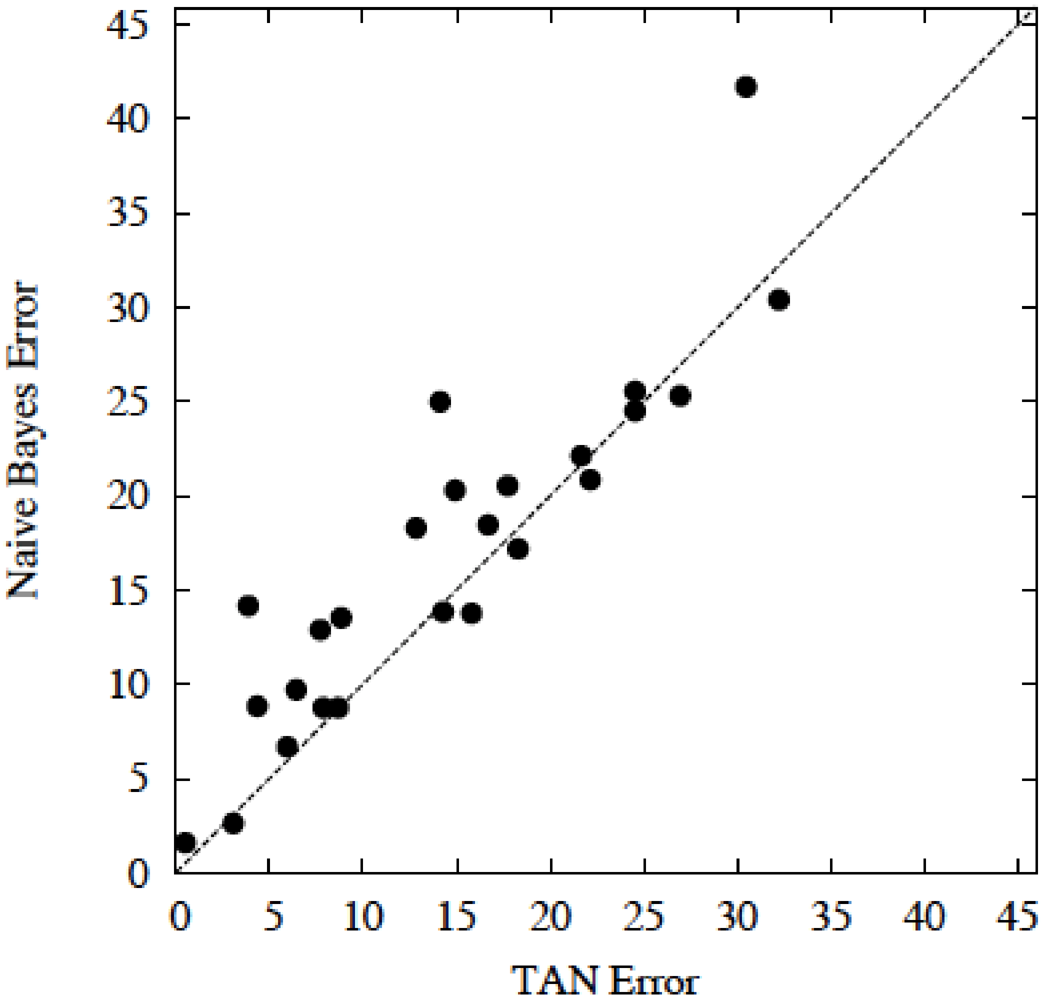


Figure from Friedman et al., *Machine Learning* 1997

Comments on Bayesian networks

- the BN representation has many advantages
  - easy to encode domain knowledge (direct dependencies, causality)
  - can represent uncertainty
  - principled methods for dealing with missing values
  - can answer arbitrary queries (in theory; in practice may be intractable)
- for supervised tasks, it may be advantageous to use a learning approach (e.g. TAN) that focuses on the dependencies that are most important
- although very simplistic, naïve Bayes often learns highly accurate models
- BNs are one instance of a more general class of *probabilistic graphical models*