Bayesian Networks

By Gwyn, Durvasula, Hurst, and Arora

Where We Are

Previously within this course we have covered some of the ideas of graph theory with basic neural networks, as well as markov chains briefly covered in 16A.

Continuing with the ideas of using graph to encode information and quickly and efficiently compute classifications and decision, We introduce Bayesian Networks.

Similarly to how neural network propagate values down their graph, so do Bayes Nets, other ever within bayes nest every node represents concrete real features, and the values we propagate down are the conditional probabilities of the nodes above.

Bayes Nets also utilize hyperparameter turning to solve for the most accurate graph model, and data reguilization.

Probability Distributions

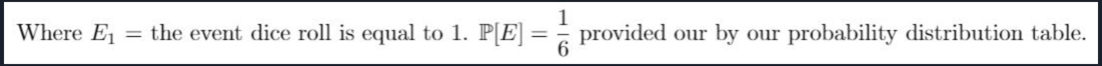
A probability Distribution describes the likelihood of an specific set of events occuring.

An event could take the form of flipping a coin or rolling a dice.

EX. Rolling a dice

#	Likelihood
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

We calculate the Likelihood an event occurs as the number of sample where the event occurs over the total size of the sample space.



Where E_1 = the event dice roll is equal to 1. $\mathbb{P}[E] = \frac{1}{6}$ provided our by our probability distribution table.

Probability Distributions can also describe continuous event, like the exact about on inches of rainfall.

Their likelihood is calculated via probability density function(PDF).

EX. Normal distribution, exponential distribution

Conditional Probabilities

We are able to condition the probability of one event occurring, given that another event occurred. Now we want to find P[A | B] the likelihood of event A given event B

EX. consider Event A = the result of two dice rolls and event B is the sum of the two rolls is ten.

#	Likelihood
4,6	1/4
5,5	1/2
6,4	1/4
?,?	0

Note that all events A which make the B impossible have probability zero.

Also the sum of the Likelihoods should always be 1 of any valid conditional probability table

Conditioning on an B changes our calculation for likelihood to be the total number of table which satisfy both events A and B over our sample Space which in now equal to the number of sample which satisfies event B

Bayes Law

To solve for the conditional probabilities we we used the equivalent equation.

$$\mathbb{P}[E_2 \mid E_1] = \frac{\mathbb{P}[E_1 \cap E_2]}{\mathbb{P}[E_1]}$$

Which we can then set equal to

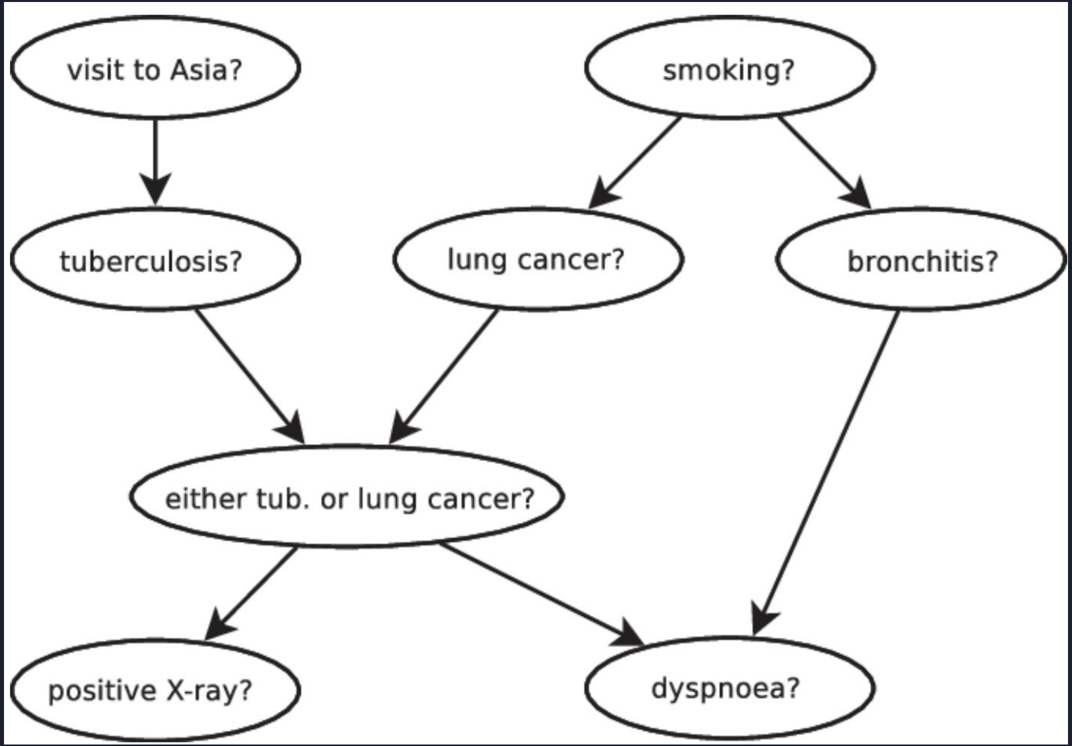
$$\mathbb{P}[E_1 \cap E_2] = \mathbb{P}[E_1] \cdot \mathbb{P}[E_2 \mid E_1]$$

$$\mathbb{P}[E_1 \cap E_2] = \mathbb{P}[E_2] \cdot \mathbb{P}[E_1 \mid E_2]$$

$$\mathbb{P}[E_2 \mid E_1] = \frac{\mathbb{P}[E_2]\mathbb{P}[E_1 \mid E_2]}{\mathbb{P}[E_1]}$$

Bayes law which describes essential manipulations to which will be applied to our bayes net

Bayes Nets


visit to Asia?
smoking?
tuberculosis?
lung cancer?
bronchitis?
either tub. or lung cancer?
positive X-ray?
dyspnoea?

Edges and Nodes

Within a Bayes Net

- Nodes: Each features of data are represented by node

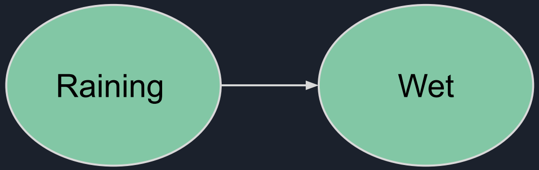
EX.

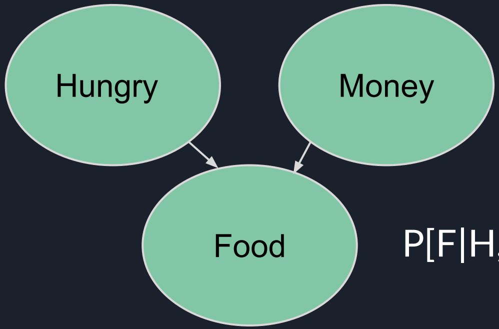
BOOL
True,
False

Categorical
carrot,
Bread,
steak

- Edges: The edges with in our graph represent the dependence between features.

P[W|R]





P[F|H,M]

Acyclic Directed Graphs

All bayes nets but be Directed and Acyclic

Remember that the edges in our graph represent the probability of a node dependent on its parent. Within bayes net parents are not dependent on their childrens value, so our edge must have a strict direction from parent to child.

Also due to the conditioning nature of the edges, our graph must by Acyclic.

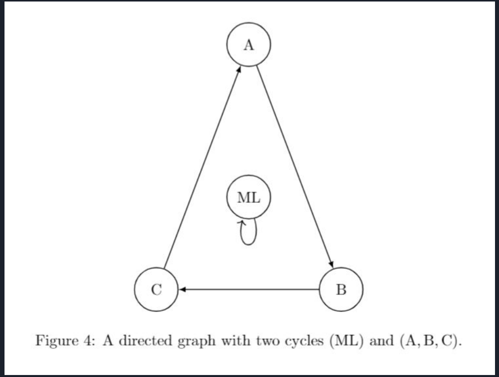
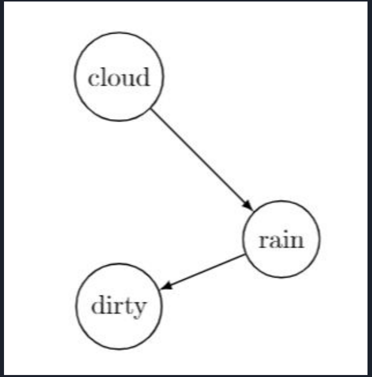


Figure 4: A directed graph with two cycles (ML) and (A, B, C).

Invalid



valid

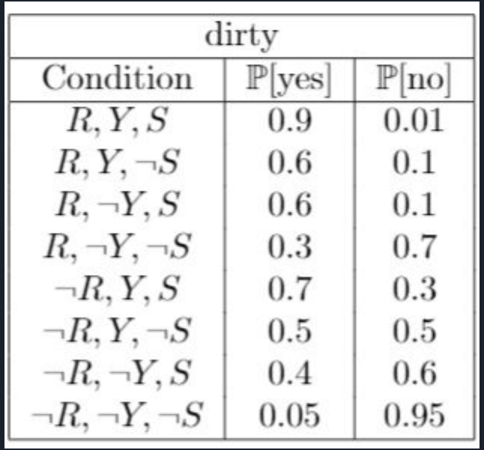
If the graph contain a cycle then multiple events would have circle dependence and cause gridlock within our calculations

Conditional Probability Tables

Each node with in our graph has its own Conditional Probability Table.

These tables store the conditional probabilities for every value that the node can take on conditioned of on every value the joint features of the parents can take on.

Ex. consider the node dirty, which has parents R, Y, S, all nodes store boolean values.



dirty

Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
R, Y, S	0.9	0.01
$R, Y, \neg S$	0.6	0.1
$R, \neg Y, S$	0.6	0.1
$R, \neg Y, \neg S$	0.3	0.7
$\neg R, Y, S$	0.7	0.3
$\neg R, Y, \neg S$	0.5	0.5
$\neg R, \neg Y, S$	0.4	0.6
$\neg R, \neg Y, \neg S$	0.05	0.95

This table then stores N*F^P = 2*2^3 values

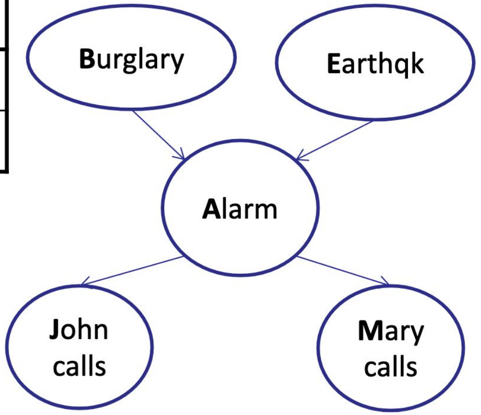
N: Number of values for current node

F: Number of values in parent nodes

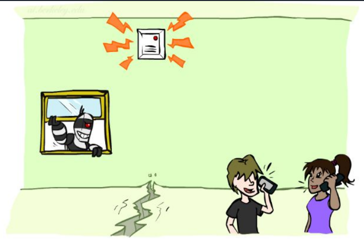
P: Number of parents

Complete Bayes Net

B	P(B)
+b	0.001
-b	0.999



E	P(E)
+e	0.002
-e	0.998



A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

P(M|A)P(J|A)
P(A|B,E)P(E)
P(B)

Inference

Assuming we have a complete Bayes net we now want to make inferences.

In Order to do this we must first understand how to rate and calculate the probability of a particular set of values for each node.

Since we can calculate the probability of a given node via our CPT, and our graphs are acyclic we can then represent the probability of a sample as the product of the probabilities of each of the nodes given our data

$$\mathbb{P}[S] = \prod_{i=1}^{x} \mathbb{P}[n_i]$$

S: sample

X: # nodes

N_i: node i

Inference Missing Data

Now that we underdate how to calculate the probability of each sample, we wish to predict missing values of a sample.

For this problem we use MLE but predicting the missing value as that which maximizes the likelihood of our sample, given the previous equation.

One last common form of inference is predicting the probability of a single value with no other information. This is achieved by summing over all probabilities of the graph for every possible missing sample.

$$\mathbb{P}[T = t] = \sum_{s1,s2,s3,s\ldots} \prod_{i=1}^{x} \mathbb{P}[n_i]$$

Variable Elimination

- We can further simplify the time complexity of our inference equation using a technique known as Variable Elimination.

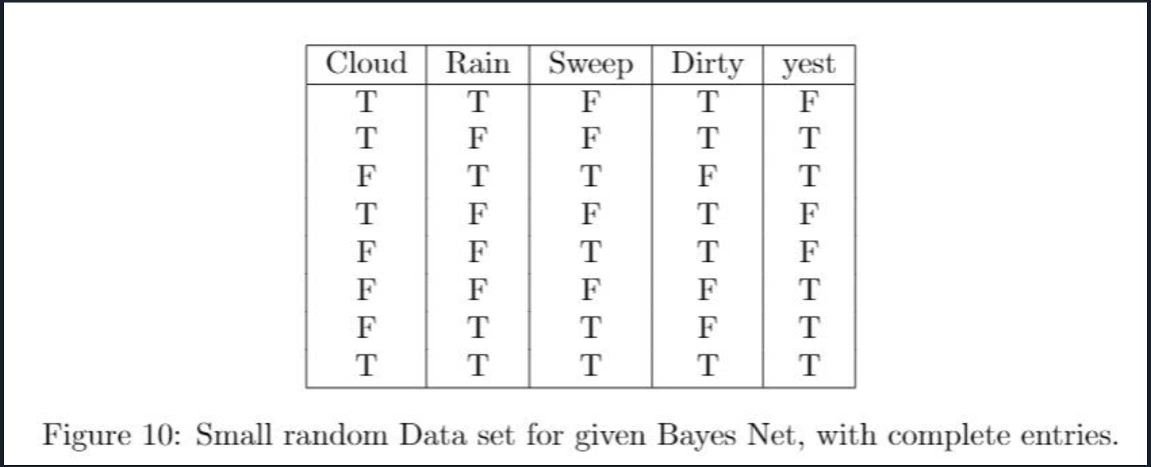
- Variable Elimination also use to excluded the conditional probabilities from nodes which are independent and don't affect our outcome.

- The intricacies of variable Elimination are described within the note and will by vital for implement Bayes nets within Your python assignment.

Data for Parameter Learning

The Conditional probability tables are almost never available within practical use of the bayes nets.

We must calculate them ourselves. Assume that we are provided with a complete data set



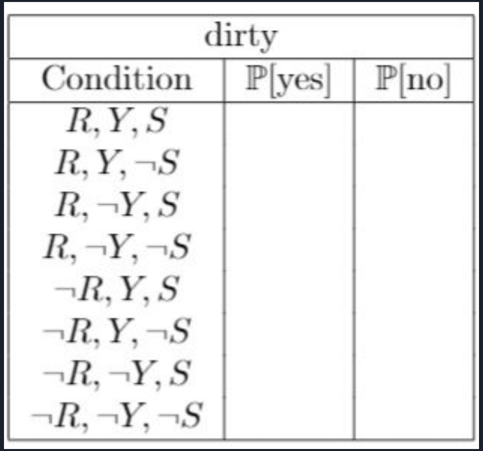
Cloud	Rain	Sweep	Dirty	yest
T	T	F	T	F
T	F	F	T	T
F	T	T	F	T
T	F	F	T	F
F	F	T	T	F
F	F	F	F	T
F	T	T	F	T
T	T	T	T	T

Figure 10: Small random Data set for given Bayes Net, with complete entries.

What are our parameters

If the structure of our bayesian network is already know to us then we also know the structure of the Conditional probability table for each node however what we are missing is the likelihood estimations for each entry.

This missing data is the are the parameters theta which we wish to learn and we will do so by calculating the MLE of each parameter, which can be done by finding the empirical Distribution



dirty		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
R, Y, S		
$R, Y, \neg S$		
$R, \neg Y, S$		
$R, \neg Y, \neg S$		
$\neg R, Y, S$		
$\neg R, Y, \neg S$		
$\neg R, \neg Y, S$		
$\neg R, \neg Y, \neg S$		

Parameters to find

Empirical Distribution

Note that the parameter that that we are training is equal to the conditional probability, with in our CPT, and via MLE we can estimate this value as the Empirical Distribution.

$$\theta_i^n = \mathbb{P}[N = n | S = s_i] = \frac{\#D(N = n, S = S_i)}{\#D(S = S_i)}$$

The above Empirical distribution calculate the conditional probability for each parameter and the count of data sample which satisfies the parent events s_i and the Node event n, over the count of data samples which satisfy the parent events s_i.

Algorithm for Parameter Learning

- For every node/feature in your data create a CPT
- For every value which that node can take on N
- For every combination of values which the parents can take on S.
- For all samples in your dataset
- If the sample satisfies N and S increment your numerator and denominator by +1
- Else if sample only satisfies S increment your denominator by +1
- Insert final value in to your CPT

Small or generic datasets

Within our Naive approach to Parameter learning if a particular event is not within our training data, then it conditional probability of occuring would be Zero. The majority of the time this does not actually represent, the true conditional probability, and better tells us that that our data is not of sufficient size.

It is not in our best interest the majority of the time to have a distribution of Zero, for this reason we have to regulate our data using Laplace smoothing.

Laplace Smoothing

Laplace Smoothing introduced the hyperparameter k to bayes net. What Laplace smoothing then does is assume every possible event occurs k times. This regulates our data assuring that the each event has some non zero probability.

This can be implemented by changing the empirical distribution as shown.

$$(Laplace)\theta_i^n = \frac{\#D(N = n, S = S_i) + K}{\#D(S = S_i) + (K * B)}$$

B is the domain size of the node

The story so far....

- We often know the structure of a Bayes Net through domain knowledge or some other prior information.
- We either know the CPT entries in advance or can use observed data to estimate them.

- However, what happens if we don’t have any priors and as a result do not know the structure of the Bayes Net? Then we also don’t know what our CPT’s are.
- Bayes Net structure is thus crucial. Can we somehow “estimate” a structure like we estimated CPT entries?

Yes! Data is your friend.

- Like we did in parameter learning, we can leverage our data to find a Bayes Net Structure.
- Remember, a Bayes Net is a DAG. So, using the same idea as in parameter learning, we want to find a DAG *(let's call it G)*, that maximizes the likelihood of the data *(let's call it D)* we have.
- We want to consider all DAG's *G*, and choose the one that maximizes the log likelihood (LL) of our data *D*. Mathematically, we can define the LL as:

$$\mathscr{L}\mathscr{L}\,(\mathrm{G}; D) = \prod_{d \in D} P\,(d \mid \mathrm{G})\,.$$

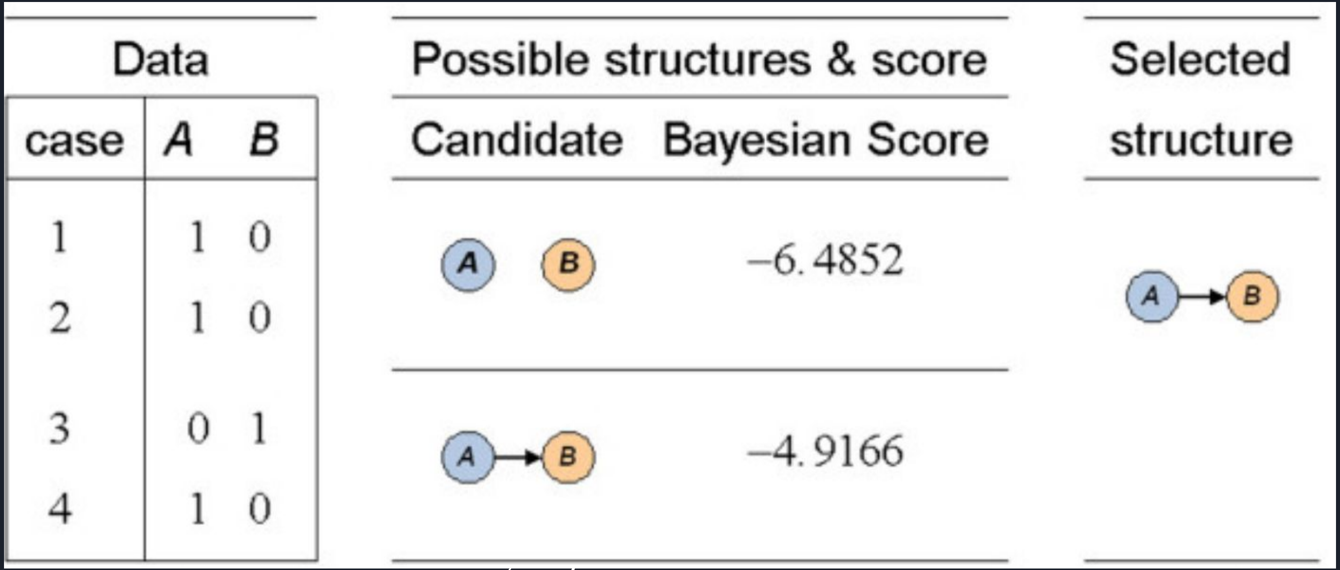
Intuition and Algorithms

- Intuitively, our data comes from an underlying true structure G_{true}, and thus G_{true} is the G that maximizes *LL(G; D)* (along with a regularization term to encourage sparse and simple structures to avoid overfitting).
- Our goal has now become to find a G that ideally = G_{true}, or is as close to it as possible.

Algorithm 1: Score and Search:

- Consider all possible DAG's *G* given the variables you have, and assign each of them a score = *LL(G; D)*. Choose the G with the highest score.
- This should be pretty intuitive, but as you may have noticed, its runtime is exponential with regards to the number of variables (Super Exponential).

Score and Search Naive



Data			Possible structures & score		Selected
case	A	B	Candidate	Bayesian Score	structure
1	1	0	$A \quad B$	−6.4852	$A \rightarrow B$
2	1	0			
3	0	1	$A \rightarrow B$	−4.9166	
4	1	0			

we've already considered?

Algorithm 2: Approximate Learning

- Score and Search is great and makes sense, however its runtime is horrible.
- Approximate Learning Algorithms reduce the search space of Score and Search via methods like:
  - Imposing constraints on the structure of the output. (Eg: Has to be a tree, cannot have more than x edges, etc.)
  - Heuristics and A* search.
  - Dynamic Programming
  - Hill Climbing Methods, Simulated Annealing etc.
- What does this mean for runtime?
- Accuracy/Performance?

Algorithm 2a: Chow Liu Trees

- Output must be a tree.
- Procedure:
  - Add every edge possible with a weight equal to the mutual information (I) between the two variables in question. A higher weight means a higher mutual dependency between the two variables.
  - Using all the edges and their weights, construct a maximum weight spanning tree - ie choose the edges that form a sparse fully connected structure while trying to use edges to encode high mutual dependencies between variables.
  - Choose any root node and assign edge directions such that you have a Directed Tree where each node only has one parent except the root node.
  - Note: Edge directions don’t matter for likelihood as Mutual Information is symmetric.

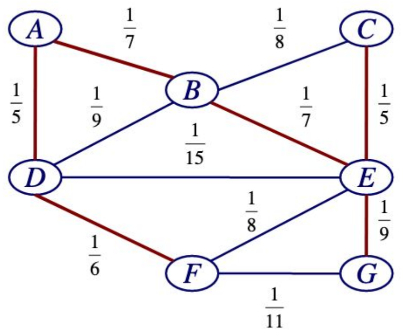
Chow Liu Trees

use mutual information to calculate edge weights

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

- One of the quickest approximate algorithms.
- Runtime = $O(v^2)$ where v is the number of variables in the BN.

find maximum weight spanning tree: a maximal-weight tree that connects all vertices in a graph



Algorithm 3. Constraint Learning

- Yet another class of algorithms for Bayes Net Structure Learning (BNSL).
- Involve calculating correlation or co-occurrence to identify an undirected backbone of edges that could exist in true *G*.
- Prune these edges systematically until a DAG is reached.

- Not as intuitive as the previous algorithms as there is no probabilistic model or function over here. These class of algorithms have been described as you should be aware they exist but are not in scope.

Conclusion

- Often times, we may not know the structure of a Bayes Net but may have data from which we can learn the structure.
- Learning consists of finding a DAG that maximizes the log likelihood of the data being generated from that DAG structure. Thus we have a way of scoring each DAG.
- Number of DAG's is exponential with number of variables, so we resort to approximate methods such as the Chow Liu Tree Algorithms.
- There is an inherent tradeoff between runtime of algorithms and how 'good' a structure they can recover.

References

- https://www.biostat.wisc.edu/~craven/cs760/lectures/BNs-2.pdf
- https://www.jmlr.org/papers/volume7/niculescu06a/niculescu06a.pdf