Structure Learning

Scenario 1

Consider an experiment where we toss two standard coins C_1 and C_2 independently. We are given a data set with 100 instances of this experiment. We would like to **learn a model for this scenario.**

- **27** times the result was C_1= **head** C_2= **head**
- **22** times the result was C_1= **head** C_2= **tail**
- **25** times the result was C_1= **tail** C_2= **head**
- **26** times the result was C_1= **tail** C_2= **tail**

Assessing Empirical Independence

- Given a dataset of observations can we decide if two variables X and Y are independent?
  - Consider the empirical distributions $p(X)$, $p(Y)$ and $p(X, Y)$, the ***mutual information (MI)*** as the "difference" between $p(X, Y)$ and $p(X)p(Y)$ is defined as

$$MI \equiv \sum_{x,y \in dom(X,Y)} P(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

This is known as the Kullback-Leibler divergence (also called relative entropy)

  - If the MI is **equal to 0** then X and Y are **independent**

??

1. Are the two coins **independent** in the empirical distribution?
   Is this expected?
2. How does a BN for this scenario would look like?

- **27** times the result was C_1= **head** C_2= **head**
- **22** times the result was C_1= **head** C_2= **tail**
- **25** times the result was C_1= **tail** C_2= **head**
- **26** times the result was C_1= **tail** C_2= **tail**

Scenario 2

We scan the sports section of our news feed for 100 days and choose an article at random each day. We mark **X=1** if the word “rain” appears in the article, and **X=0**. Similarly, **Y=1** will indicate whether the word “football” appears in the article, and **Y=0** if this is not the case.

- Suppose that we have similar results as before:

- **27** times the result was X = **1** Y = **1**
- **22** times the result was X = **1** Y = **0**
- **25** times the result was X = **0** Y = **1**
- **26** times the result was X = **0** Y = **0**

??

Are the X and Y **independent** ?

Does the model underlying this scenario contains an **edge between X and Y**?

Assessing Empirical Independence

- Given a dataset of observations can we decide if two variables X and Y are independent?
  - Consider the empirical distributions $p(X)$, $p(Y)$ and $P(X, Y)$, the ***mutual information (MI)*** as the "difference" between $P(X, Y)$ and $p(X)p(Y)$ is defined as

$$\sum_{x,y \in dom(X,Y)} P(x, y) \log \frac{P(x, y)}{P(x)P(y)}$$

  - If the MI is equal to 0 then X and Y are independent

Problem: Since we formed $P(X)$ and $P(Y)$ based on a set of observations. It is likely hat **even when $X \perp Y$ holds** in the true distribution p* the **MI will not be zero**

Finding Dependencies and Correlations in Data

Example: the presence or absence of a disease in a human being has a direct influence on whether a test for that disease turns out positive or negative.

Correlations between variables can be revealed, for example, using statistical independence tests.

But we might want to get more…

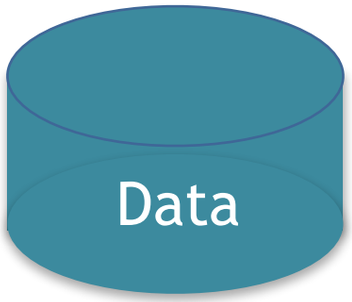
✤ A BN structure can **distinguish between direct and indirect dependencies** (both of which lead to correlation in the resulting distribution).

Learning

True distribution P^*
(corresponding to a BN G^*)

Dataset of instances sampled from P^*

$$D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$$



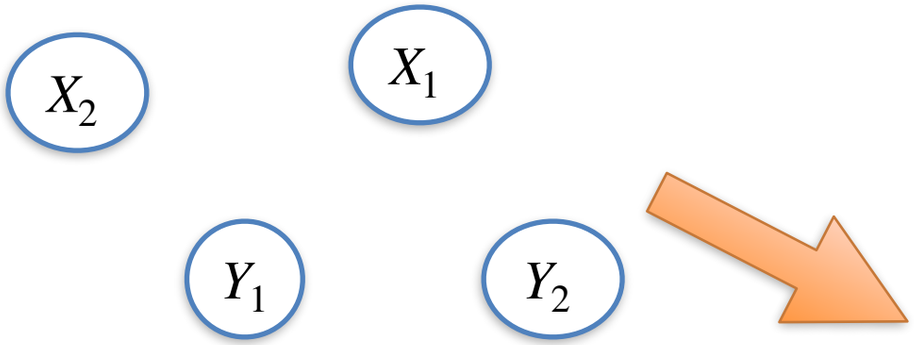
Learning

Domain-Expert Knowledge

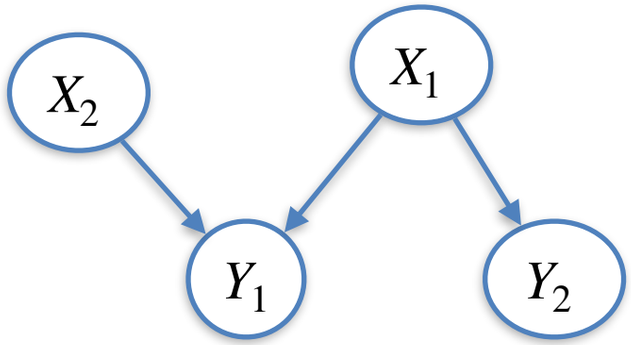
Elicitation

Network
G

Unknown Structure and Complete Data



Learning



$P(Y_2 \mid X_1)$

X_1	$Y_2 = 0$	$Y_2 = 1$
0	0	1
1	0.5	0.5

X_1	X_2	Y_1	Y_2
0	1	0	0
1	0	1	0
1	1	1	0
1	1	0	0
1	0	1	1
0	1	1	1
0	0	0	1

$P(Y_1 \mid X_1, X_2)$

X_1	X_2	$Y_1 = 0$	$Y_1 = 1$
0	0	0	1
0	1	0.5	0.5
1	0	0.7	0.3
1	1	0.25	0.75

The importance of correctly reconstructing the network structure depends on our **learning goal**

- **Goal 1**: Understanding the **domain structure**

- **Goal 2**: Use the learned model for reasoning about **new** instances

Learning the Structure of the Model (from Data)

For knowledge discovery: by examining the dependencies in the learned network, we can learn the dependency structure relating variables in our domain.

- There can be many perfect maps for a distribution P^*

  ➡ G^* is not identifiable from the data

- Learning G^* is hard to achieve

  - The sample data are noisy and do not reconstruct the underlying distribution perfectly

  - Difficult to detect which independencies are present in the underlying distribution

Learning the Structure of the Model (from Data)

Density estimation: Estimate a statistical model of the underlying distribution.

➡ generalise to new instances

Because G^* captures the true dependencies and independencies in the domain, it seems is intuitively reasonable that the best generalisation will be obtained if we recover the structure G^*

What happens if we make mistakes in the structure?

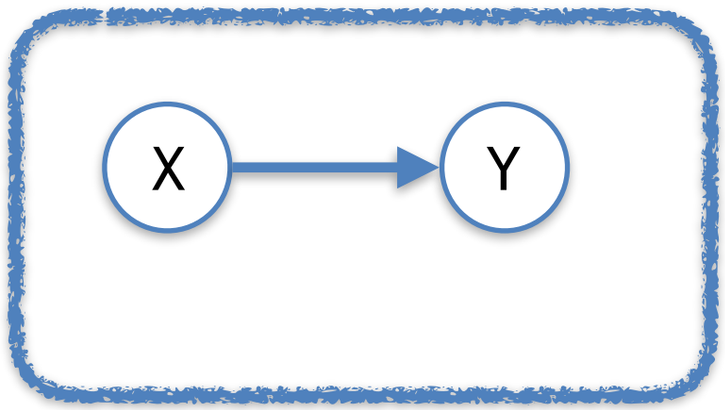
How does a more complex model (one with more edges) behave with respect to the true distribution P^*?

Back to Scenario 1

- Assume 20 instances as follows:

3 times: X = **heads** Y = **heads**
6 times: X = **heads** Y = **tails**
5 times: X = **tails** Y = **heads**
6 times: X = **tails** Y = **tails**

G_1 :



What is of estimation of (using MLE)?

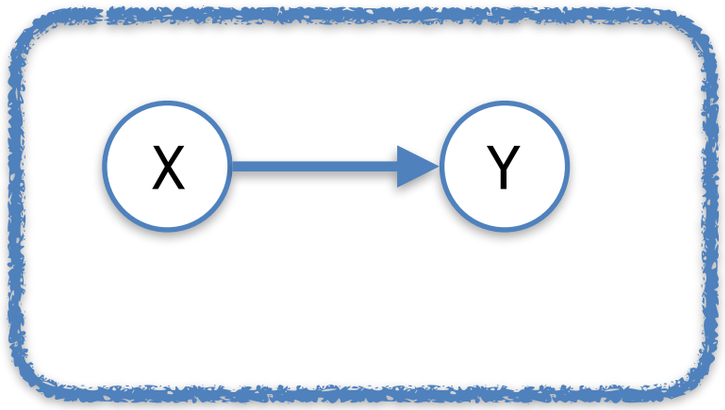
1. p(X = heads)?
2. p(Y = heads | X= heads)?
3. p(Y=heads | X= tails)?

Back to Scenario 1

- Assume 20 instances as follows:

3 times: X = **heads** Y = **heads**
6 times: X = **heads** Y = **tails**
5 times: X = **tails** Y = **heads**
6 times: X = **tails** Y = **tails**

G_1 :



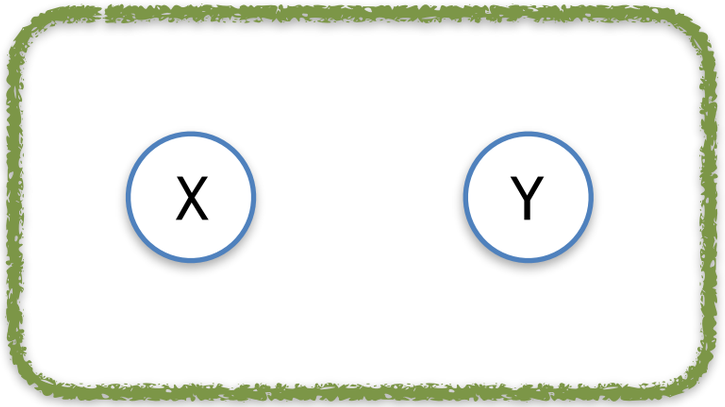
p(X = heads)= 0.45
p(Y = heads | X= heads)= 1/3
p(Y=heads | X= tails)= 5/11

Back to Scenario 1

- Assume 20 instances as follows:

3 times: X = **head** Y = **head**
6 times: X = **head** Y = **tails**
5 times: X = **tails** Y = **head**
6 times: X = **tails** Y = **tails**

G_2 :

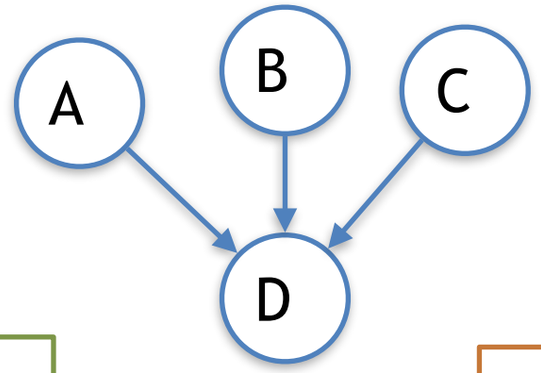


p(Y = heads)= 8/20= 0.4

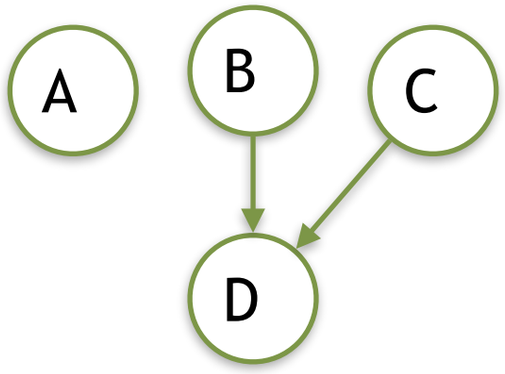
When doing density estimation from limited data, it is often better to prefer a **sparser structure**.

Importance of Accurate Structure

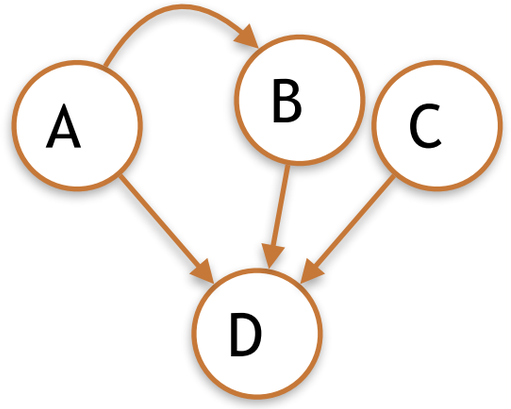
True model



Missing Edge



- Incorrect Dependencies
- Correct distribution P* cannot be learned
- Can generalise better



- Spurious dependencies
- Can correctly learn P*
- Increases the number of parameters
- Worse generalisation

Structure Learning Approaches

- Score-based:
  - Define a **scoring function** that evaluates how well a structure matches the data
  - **Search** for a structure that **maximises** the score
  - It amounts to an optimisation problem over the space of network structures

SEARCHING OVER STRUCTURES (I)

Learning as an Optimisation Problem

Input:

- Training data
- Scoring function
- Set of possible structures

Output:

- A network that maximises the score

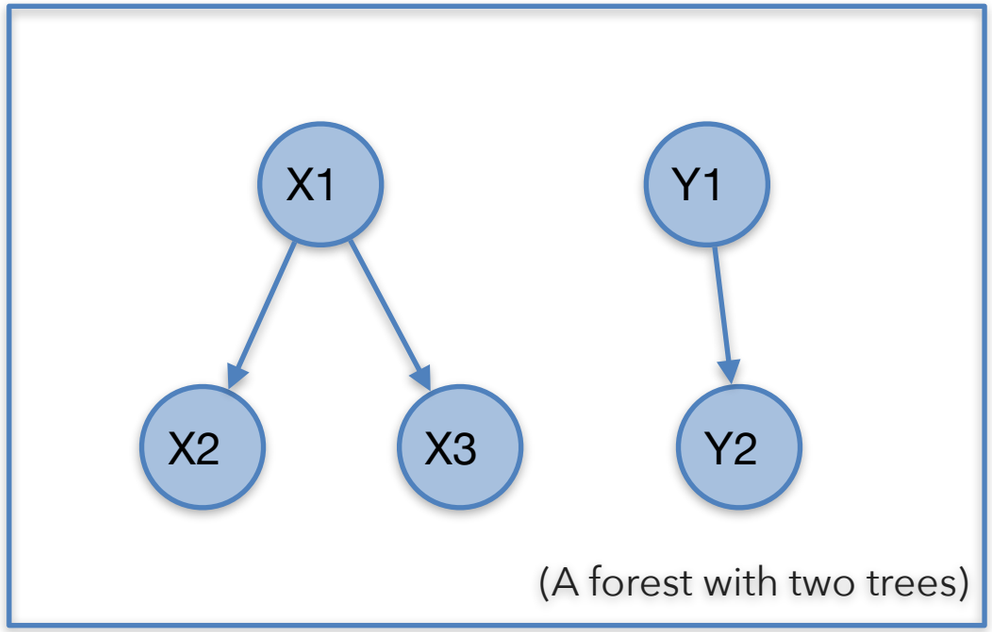
✤ Key property property for computational efficiency is the ***decomposability*** of the score function:

- the score function can be computed by summing up the scores of each family (a node and its parents) in the graph G

$$\textbf{score}(G) = \sum_i \textbf{score}(X_i \,|\, pa(X_i))$$

Learning simple Structures

- At most one parent per variable: trees and forest
- Why?
  - Mathematically elegant
  - Efficient optimisation
  - Sparse parametrisation



(A forest with two trees)

➡ Reducing the possibility of **overfitting the data**

Learning Forest

- Provided that the score function enjoys decomposability

  Score = sum of edge scores + constant

Let us define ***weight*** of edge $X_i \rightarrow X_j$ as

$$w(i \rightarrow j) = \text{score}((X_j \mid X_i)) - \text{score}(X_j)$$

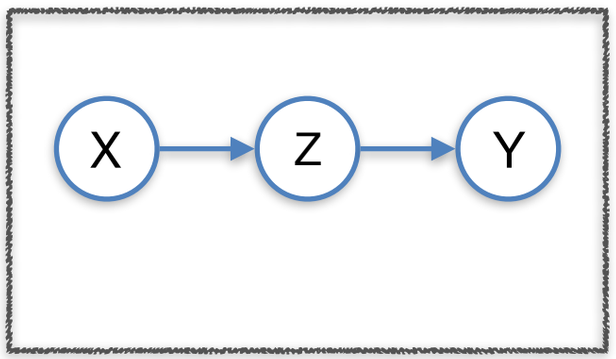
A score satisfies **score equivalence** if is assigns the same score to graphs satisfying the same conditional independency assumptions.

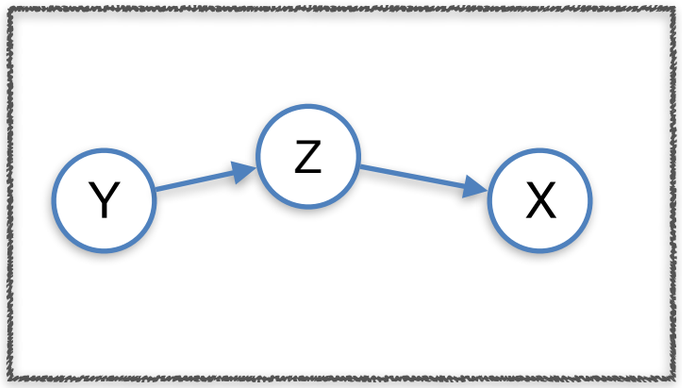
For a score satisfying score equivalence it holds that:

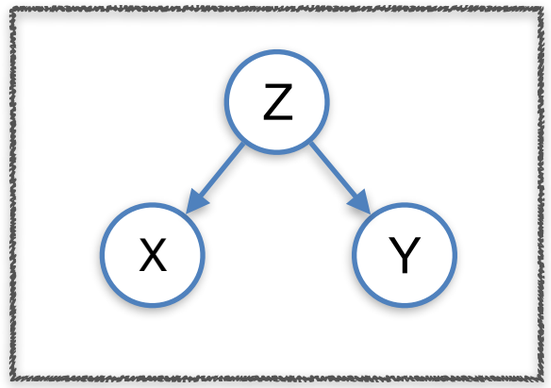
$$w(i \rightarrow j) = w(j \rightarrow i)$$

We can use an undirected graph

Recall







- **Different graphs** can represent the **same conditional independence** assumptions.

- Using the notion of **Markov equivalence** we can tell whether this is indeed the case.

✤ As a recall exercise, test whether the BN above represent the same conditional independence assumptions.

Learning Forest (algorithm)

- Define undirected graph with a node for each variable

$$X_1, \ldots, X_n$$

- Set $w(i,j) = \max[\text{score}(X_j \mid X_i) - \text{score}(X_j)\ ,\ 0]$

We consider 0 to eliminate negative edges. (Some scoring functions might penalise edges with a negative score)

- Find forest with maximal weight
  - We can use any standard algorithm for maximum weight spanning tree (e.g. Prim's or Kruskal's) in $O(n^2)$ time
  - Remove all edges of weight 0 to produce a forest

Summary

- Structure learning can be reduced to an **optimisation problem** over the combinatorial space of graph structures.

- For score-based approaches, **decomposability** of the scoring function allows us to consider scores of each family (i.e., a node and its parents) separately.

- For tree structures, we can use standard maximum weight spanning tree algorithms and solve the problem efficiently in quadratic time.

Summary

- The problem of learning a BN structure can be seen as an optimisation problem consisting of two parts:

  1. Define a scoring function
  2. Come up with an algorithm optimising the scoring function

- For trees we can apply a greedy algorithm for finding de maximum weight spanning tree.

SEARCHING OVER STRUCTURES (II)

Learning as an Optimisation Problem

Input:

- Training data
- Scoring function
- Set of possible structures

Output:

- A network that maximises the score

Greedy Hill Climbing

- **Initial G:**
  - Empty Network
  - Best tree
  - A random network
  - A network constructed using prior knowledge
- **At each iteration:**
  - Consider score for **all possible changes** (based on the operator we have defined)
  - Apply change that **maximally improve** the score
- **Stop when no modification improves the score**

We have then reached a local maximum

Beyond Trees

Finding maximal scoring network structures with at most K parents for each variable is **NP-hard** for k>1.

- Thus, using the simple greedy algorithm is not longer guaranteed to work e.g. on structures allowing two parents.

- No efficient algorithm is likely to be found for this problem

Heuristic Search

- To tackle this problem we can use heuristic hill climbing search

Given a BN G, we can consider **changes** to the network and see whether these changes **improve its score**.

For example: add/remove edges, reverse an edge

Design Choices

- **Search operators:**
  - Local step
  - Global steps
- **Search techniques:**
  - Greedy hill-climbing
  - Best first search
  - Simulated Annealing,
  - …

??

Which of the following might pose problems for learning the correct structure?

1. Discrete steps in the score while changing the structure
2. Small changes of the network lead to no or very small changes in the score
3. Local maxima
4. The inability to express edge deletion as an atomic operation on the network structure

Pitfalls

- Greedy hill-climbing can get stuck in :
  - **Local maxima**
  - **Plateau:** changes on the network do not affect the score
    - ✤ **Equivalent networks** are often neighbours in the search space

Simple Algorithm to avoid Pitfalls

Greedy hill-climbing

+

- **Random restarts**

  Take a number of random steps when stuck, and then start climbing again

- **Forbidden list**

  - Keep a list of the last N steps taken
  - Search cannot reverse any of the steps in the list

Summary

- BN structure learning is useful for building better predictive models
  - Domain experts don't know the structure well
  - For Knowledge discovery
- Finding highest-scoring structures is NP-hard for structures beyond trees

➡ Unlikely to find efficient algorithms

➡ We can resort to simple heuristic search

- Local steps: edge addition, deletion and reversal
- Augmented hill climbing algorithms to avoid local maxima

➡ There are better algorithms

- Make larger progress on the search space
- Computationally more expensive and harder to implement