Access Code: **318159**

CMP9794M
Advanced Artificial Intelligence

[Heriberto Cuayahuitl]

UNIVERSITY OF
LINCOLN

School of Engineering and Physical Sciences

Last Week

- Introduction to Bayesian Networks
  - Networks encode conditional independence
  - Structure via a Directed Acyclic Graph (DAG)
  - Each node has a conditional prob. table (CPT)
  - CPTs can be learnt via Maximum Likelihood Estimation (MLE) with Laplace / Aditive / Dirichlet smooting to avoid zero probabilities
- Introduction to Probabilistic Reasoning
  - Inference by enumeration
  - Inference by variable elimination

Today: Structure Learning

- It is often difficult, if not impossible, to manually specify the structure of a Bayesian network.
  - This difficulty increases according to the number of random variables involved.
  - Due to the lack of agreement between domain experts

- Bayesian network learning is defined as

$$\underbrace{\Pr(\mathscr{B}|\mathscr{D}) = \Pr(G, \Theta|\mathscr{D})}_{\text{learning}} = \underbrace{\Pr(G|\mathscr{D})}_{\text{structure learning}} \cdot \underbrace{\Pr(\Theta|G, \mathscr{D})}_{\text{parameter learning}} .$$

Bayesian network

Data

Directed Acyclic Graph (DAG)

Parameters (probability distributions)

Complexity of Structure Learning

Table 1 Number of directed graphs, directed acyclic graphs, and the percentage of directed graphs which are acyclic for different number of variables

Number of variables, n	Number of directed graphs $(3^{n(n-1)/2})$	Number of DAGs, $\lvert G_n \rvert$	Percentage of directed graphs which are acyclic (%)
2	3	3	100.0
3	27	25	92.59
4	729	543	74.49
5	59,049	29,281	49.59
6	1.4349×10^7	3.7815×10^6	26.35
7	1.0460×10^{10}	1.1388×10^9	10.89
8	2.2877×10^{13}	7.8730×10^{11}	3.42

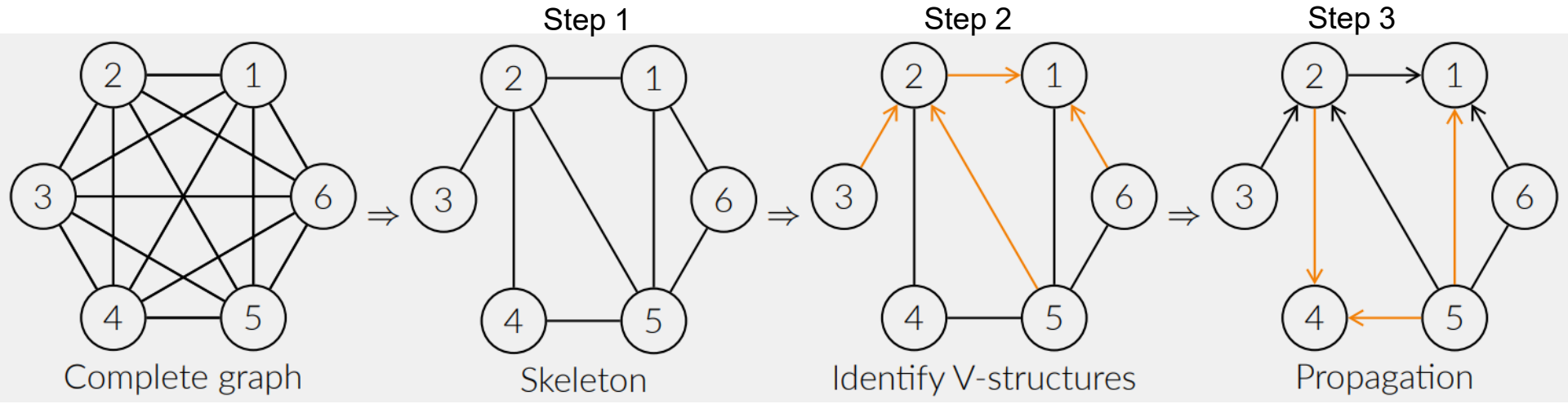
This calculation assumes that the directed graph has at most one arc between each pair of nodes

Picture from: A survey of Bayesian Network structure learning | Artificial Intelligence Review (springer.com)

Today

- **Constraint-based structure learning**
  - PC-stable algorithm
- Score-based structure learning
  - Greedy search (Hill Climbing)
- Hybrid algorithms
  - Min-Max Hill Climbing
- Evaluation metrics for probabilistic models

PC-Stable Algorithm: Key Steps



computationally intensive (depending on data size and network complexity)

Steps 2 and 3 use rules not covered in this module. But all steps are generated by libraries like bnlearn.

Picture from: https://2019.ds3-datascience-polytechnique.fr/wp-content/uploads/2019/06/DS3-403_2019.pdf

PC-Stable: Find Skeleton

Algorithm 1. The First Step in PC-Stable Algorithm

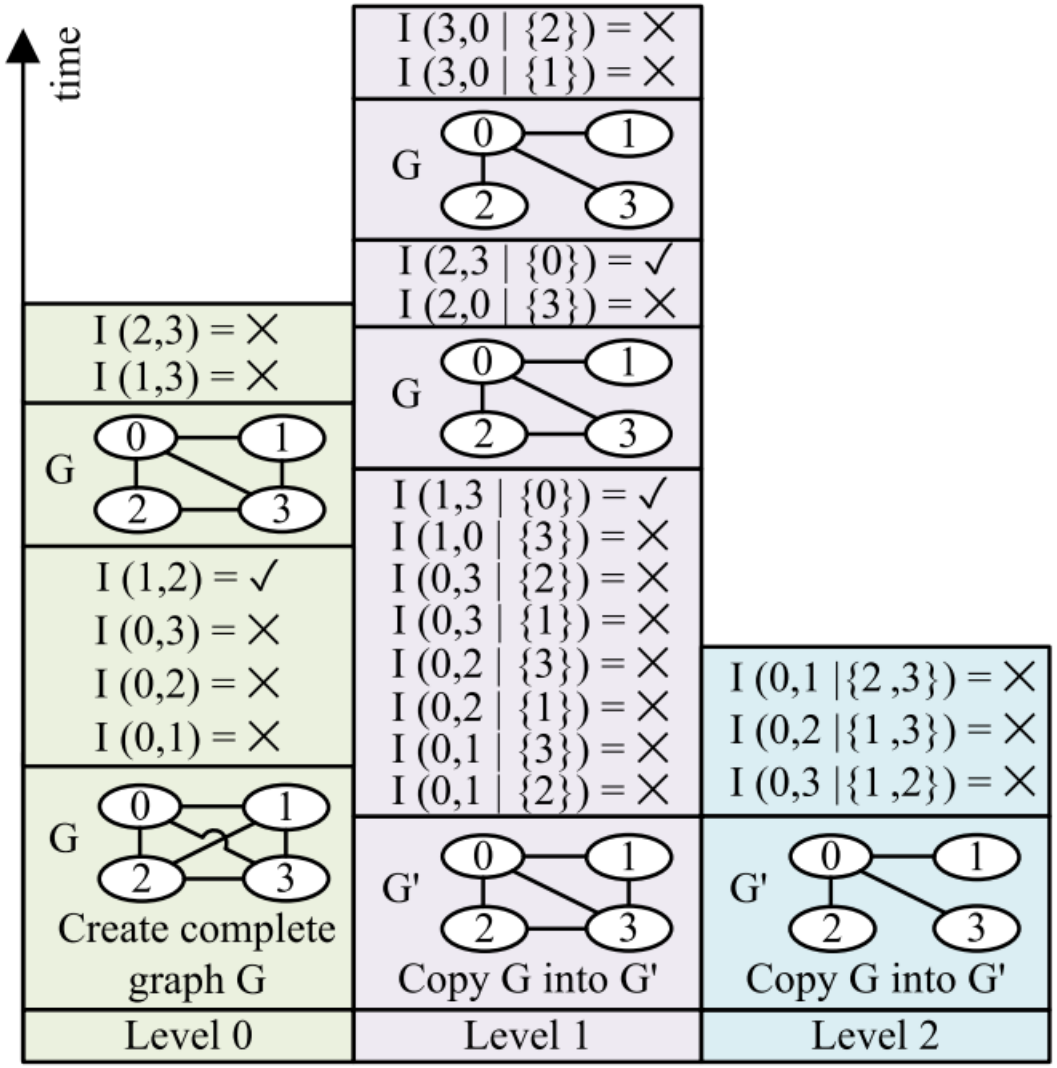
Input: $\mathcal{V}$
Output: G, $SepSet$

1: $G =$ fully connected graph
2: $SepSet = \emptyset$
3: $\ell = 0$
4: **repeat**
5: Copy G into G'
6: **for** any edge (V_i, V_j) in G **do**
7: **repeat**
8: Choose a new $S \subseteq adj(V_i, G') \backslash \{V_j\}$ with $|S| = \ell$
9: Perform $I(V_i, V_j | S)$
10: **if** $V_i \perp\!\!\!\perp V_j | S$ **then**
11: Remove (V_i, V_j) from G
12: Store S in $SepSet$
13: **end if**
14: **until** (V_i, V_j) is removed or all sets S are considered
15: **end for**
16: $\ell = \ell + 1$
17: **until** (max degree $-1 \geq \ell$)

Knowing variables S, V_i does not contribute to knowing more about V_j.

Zarebavani, B., et al. cuPC: CUDA-Based Parallel PC Algorithm for Causal Structure Learning on GPU. IEEE TPDS, Vol. 31(3), 2020.

PC-Stable: Example Skeleton



Zarebavani, B., et al. [cuPC: CUDA-Based Parallel PC Algorithm for Causal Structure Learning on GPU](). IEEE TPDS, Vol. 31(3), 2020.

Conditional Independence Tests

Assuming that each arc (in a Bayesian network) encodes a probabilistic dependence, conditional independence tests tell us whether such probabilistic dependence is supported by the data.

Any two variables are associated in a Bayes net if the test's p-value < significance_level.

typically **0.05** or **0.01**
(default=0.05)

If *p*-value < significance level: keep edges

Else: remove edges

API for Conditional Independence Tests

```
bnlearn.bnlearn.independence_test(model, data, test='chi_square', alpha=0.05, prune=False, verbose=3)
```

The statistical tests:

- chi_square
- g_sq
- log_likelihood
- freeman_tuckey
- modified_log_likelihood
- neyman
- cressie_read

Look for the following in this week's workshop: **bnlearn_ConditionalIndependenceTests.py**

API for PC-Stable Algorithm

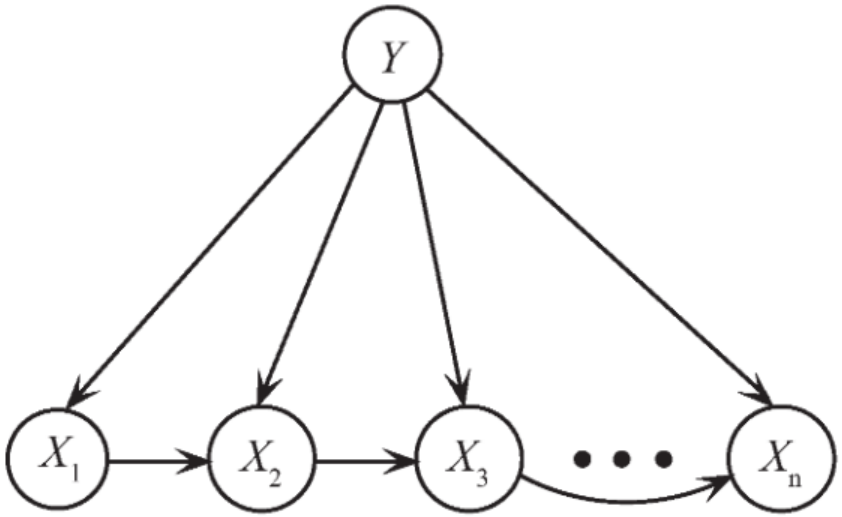
```
bnlearn.structure_learning.fit(data, methodtype='pc', params_pc={'alpha': 0.05, 'ci_test': 'chi_square'}, verbose=3)
```

The statistical tests:

- chi_square
- g_sq
- log_likelihood
- freeman_tuckey
- neyman
- cressie_read
- power_divergence
- pearsonr

Look for the following in this week's workshop: **bnlearn_ConditionalIndependenceTests.py**

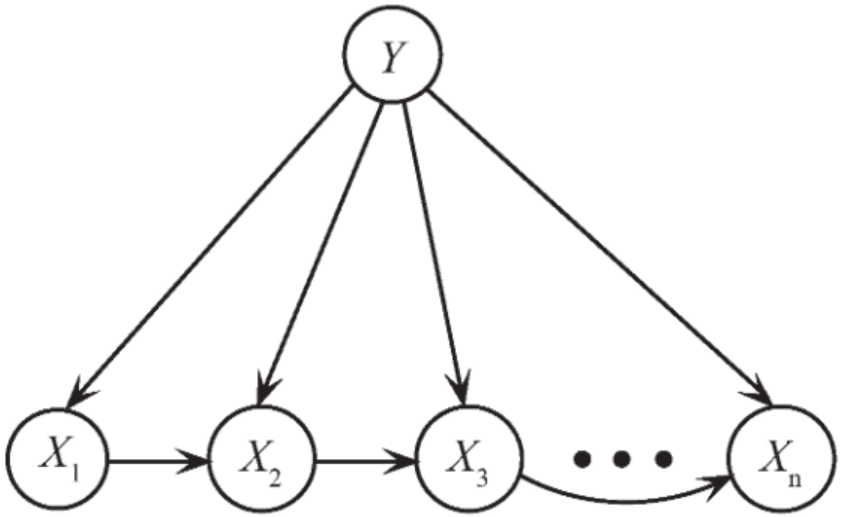
Conditional Independence Tests using the Language Detection data



Knowing Y: Is the link between X_1 and X_2 with a p-value of 0.0 needed?

Is the link between X_1 and X_{15} with a p-value of 0.4692 needed?

Conditional Independence Tests using the Language Detection data



Knowing Y: Is the link between X_1 and X_2 with a p-value of 0.0 needed?

Yes, keep the link.

Is the link between X_1 and X_{15} with a p-value of 0.4692 needed?

No, remove the link.

Today

- Constraint-based structure learning
  - PC-stable algorithm
- **Score-based structure learning**
  - Greedy search (Hill Climbing)
- Hybrid algorithms
  - Min-Max Hill Climbing
- Evaluation metrics for probabilistic models

Greedy Search: Main Ideas

Explore the search space starting from an empty network and adding/deleting/reversing one arc at a time until the score can't be improved anymore.

Doing the above requires knowledge regarding how good a network is against alternative graphs.

The goal is to search for the network with the *highest score:* $\boldsymbol{B^* = \arg\max_{B} \; Score(B|D)}$

Bayesian network

Data

Scoring Function

The score represents a trade-off:

- How well the network fits the data &
- How complex the network is

Assumption: the scoring function is decomposable

$$Score(B|D) = \sum_{i=1}^{n} Score(X_i|parents(X_i), D)$$

random variable *i*

data

Several scoring functions can been applied to structure learning such as LL, BIC, AIC, BDeu, etc.

Liu, Z., Malone, B., Yuan, C. "Empirical Evaluation of Scoring Functions for BN Model Selection", BMC Bio, 2012

Scoring Functions: Log-Likelihood (LL)

LL represents the probability of the data (*D*) given a Bayesian network (*B*):

$$LL(D|B) = \sum_j^N \log P(D_j|B)$$
$$= \sum_i^n \sum_j^N \log P(D_{ij}|PA_{ij})$$

where D_{ij} is the value of random variable X_i in training example D_j, and PA_{ij} are the values of the parents of X_i in D_j, n is the number of random variables, and N is the num. of training examples.

Liu, Z., Malone, B., Yuan, C. "Empirical Evaluation of Scoring Functions for BN Model Selection", BMC Bio, 2012

LL Example for PlayTennis Data using a Naïve Bayes Structure

$$LL = \sum_{i=1}^{N} \log P(PlayTennis_i) + \sum_{i}^{n} \sum_{j}^{N} \log P(X_{ij}|PA_i)$$

LL_1
$= \log P(PlayTennis = no)$
$+ \log P(Outlook = sunny|PlayTennis = no) +$
$+\log P(Temperature = hot|PlayTennis = no) +$
$+\log P(Humidity = high|PlayTennis = no) +$
$+\log P(Wind = weak|PlayTennis = no)$
$= \log\left(\frac{5}{14}\right) + \log\left(\frac{3}{5}\right) + \log\left(\frac{2}{5}\right) + \log\left(\frac{4}{5}\right) + \log\left(\frac{2}{5}\right) = -3.5961$

and so on for $LL_1 + \cdots + LL_{14} = -54.217$

Scoring Functions: Bayesian Information Criterion (BIC)

BIC is a penalised LL function:

$$BIC(D|B) = LL(D|B) - \sum_{i=1}^{n} Penalty(X_i, B, D)$$

where $LL(.)$ is Log-Likelihood function and the penalty term penalises complex networks as $Penalty(X_i, B, D) = \frac{\log N * p_i}{2}$, p_i is the number of parameters/probabilities of random variable X_i, and N is the number of training examples.

Liu, Z., Malone, B., Yuan, C. "Empirical Evaluation of Scoring Functions for BN model Selection", BMC Bio, 2012

BIC Example for PlayTennis Data using a Naïve Bayes Structure

$$BIC(D|B) = LL(D|B) - \sum_i \frac{\log N * p_i}{2}$$

$$= LL(D|B) - \frac{\log N * \sum_i p_i}{2}$$

$$BIC(D|B) = -54.217 - \frac{\log(14) * 22}{2}$$

$$= -83.246$$

N.B. The above assume CPTs with full enumeration (not concise ones)

Scoring Functions: Aikake Information Criterion (AIC)

AIC is a penalised LL function:

$$AIC(D|B) = LL(D|B) - \sum_{i=1}^{n} Penalty(X_i, B, D)$$

where $LL(.)$ is Log-Likelihood function and $Penalty(X_i, B, D) = p_i$ with p_i being the number of parameters/probabilities of random variable X_i, and N is the number of training examples.

AIC favours more complex networks than BIC.

Liu, Z., Malone, B., Yuan, C. "Empirical Evaluation of Scoring Functions for BN model Selection", BMC Bio, 2012

AIC Example for PlayTennis Data using a Naïve Bayes Structure

$$AIC(D|B) = LL(D|B) - \sum_{i=1}^{N} p_i$$

$$AIC(D|B) = -54.217 - (2 + 6 + 6 + 4 + 4)$$
$$= -76.217$$

Using full enumeration ->

$$|CPT(H|PT)| = 4$$
$$|CPT(W|PT)| = 4$$
$$|CPT(T|PT)| = 6$$
$$|CPT(O|PT)| = 6$$
$$|CPT(PT)| = 2$$

AIC Example for PlayTennis Data using a Naïve Bayes Structure

$$AIC(D|B) = LL(D|B) - \sum_{i=1}^{N} p_i$$

$$AIC(D|B) = -54.217 - (1 + 3 + 3 + 2 + 2)$$
$$= -65.217$$

Using concise enumeration ->

$$|CPT(H|PT)| = 2$$
$$|CPT(W|PT)| = 2$$
$$|CPT(T|PT)| = 3$$
$$|CPT(O|PT)| = 3$$
$$|CPT(PT)| = 1$$

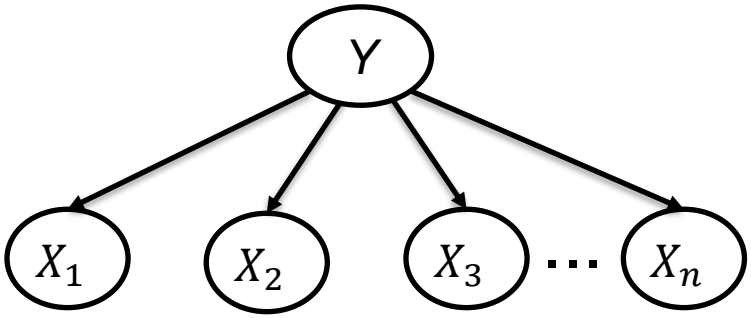
API for Structure Learning with Scoring Functions

```
bnlearn.structure_learning.fit(data,
methodtype='hc', scoretype='bic',
black_list=None, white_list=None, …,
max_iter=1000000, …, n_jobs=-1, verbose=3)
```

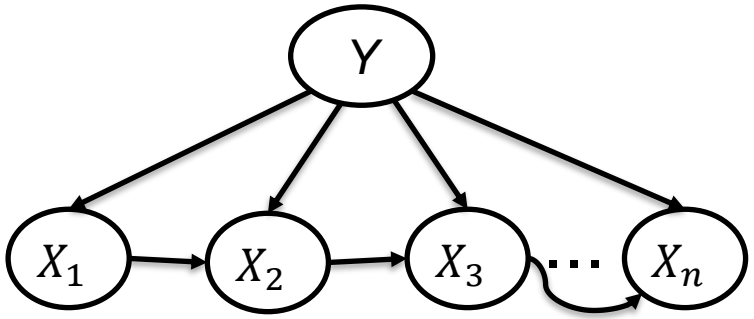
The statistical tests:

- ‘bic’ : Bayesian information criterion
- ‘k2’ : K2 score
- ‘bdeu’ : Bayesian Dirichlet score
- ‘bds’ : Bayesian Dirichlet sparse score
- ‘aic’ : Akaike information criterion

LL/BIC/AIC for Language Detection



BIC=???
AIC=???
Bdeu=???



BIC=???
AIC=???
Bdeu=???

Which structure above fits better the data?

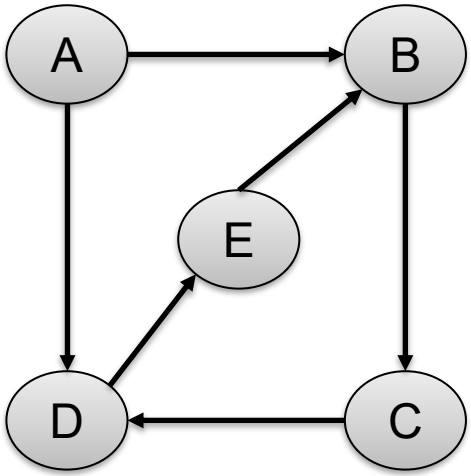
Hill Climbing Algorithm

1. Choose a network structure G over $\mathbf{V}$, usually (but not necessarily) empty.
2. Compute the score of G, denoted as $\text{Score}_G = \text{Score}(G)$.
3. Set $maxscore = \text{Score}_G$.
4. Repeat the following steps as long as *maxscore* increases:
   (a) for every possible arc addition, deletion or reversal not resulting in a cyclic network:
      i. compute the score of the modified network G^*, $\text{Score}_{G^*} = \text{Score}(G^*)$:
      ii. if $\text{Score}_{G^*} > \text{Score}_G$, set $G = G^*$ and $\text{Score}_G = \text{Score}_{G^*}$.
   (b) update *maxscore* with the new value of $Score_G$.
5. Return the DAG G.

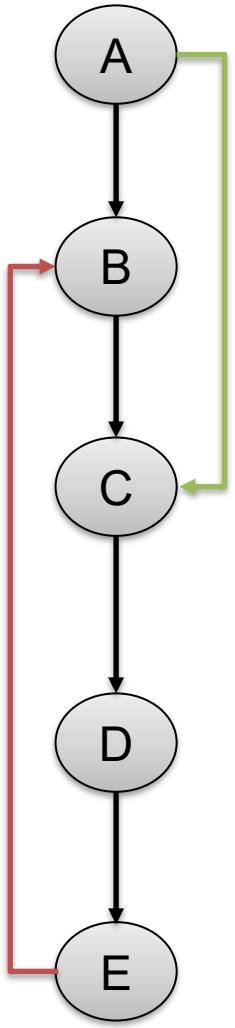
Look for the following in this week's workshop:
bnlearn_StructureLearning.py

Scutari, M. & Denis, JB. "Bayesian Networks with Examples in R", Routledge Taylor & Francis, 2022, chapter 6.

Depth First Search (DFS) for Detecting Cyclic Networks



DAGs do not have back edges



Look for function `has_cycles()` in **BayeNetUtil.py**

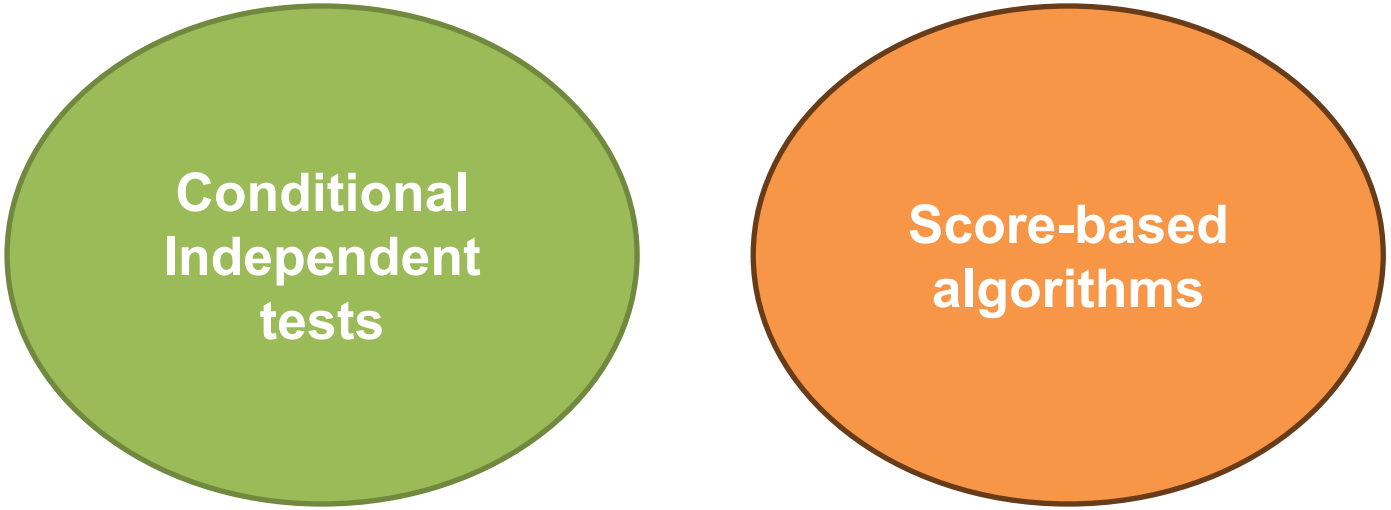
NOTATION:

- Tree edges
- Forward edges
- Back edges

Today

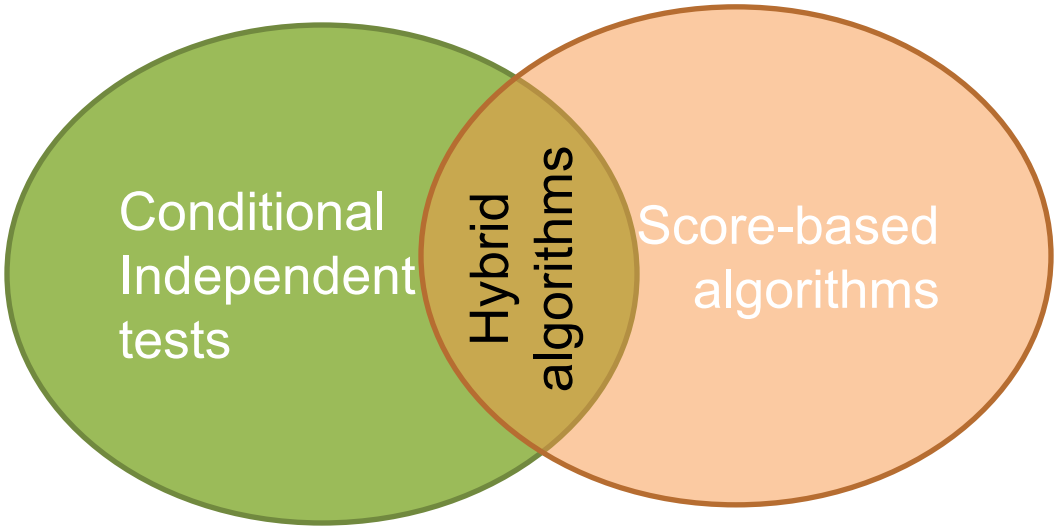
- Constraint-based structure learning
  - PC-stable algorithm
- Score-based structure learning
  - Greedy search (Hill Climbing)
- **Hybrid algorithms**
  - Min-Max Hill Climbing
- Evaluation metrics for probabilistic models

Hybrid Algorithms for Structure Learning in Bayesian Networks



- Conditional Independent tests are partially rule-based
- Score-based approaches can be search intensive
- Is there any other alternative?

Hybrid Algorithms for Structure Learning in Bayesian Networks



Since Conditional Independent tests and Score-based approaches have pros and cons, hybrid algorithms aim for **the best of both worlds**.

MMHC: Min-Max Hill Climbing

1. Choose a network structure G over V (nodes), usually empty (but not necessarily).
2. **Restrict:** select a set $\mathbf{C}_i$ of candidate parents for each node $X_i \in V$, which must include the parents of X_i in G. ← via **the PC-stable** algorithm
3. **Maximise:** find the network structure G^* that maximises $Score(G^*)$ among the networks in which the parents of each node X_i are included in the corresponding set $\mathbf{C}_i$. ← via the **Hill Climbing** algorithm
4. Return DAG G^*.

Today

- Constraint-based structure learning
  - PC-stable algorithm
- Score-based structure learning
  - Greedy search (Hill Climbing)
- Hybrid algorithms
  - Min-Max Hill Climbing
- **Evaluation metrics for probabilistic models**
  - See appendix 1, discussion during the workshop

Today

- Discussion on structure learning
- Algorithms to induce the structure of Bayes nets
- Measuring the performance of probabilistic models/classifiers

Readings:

- Kitson et al. A survey on Bayesian Network structure learning, arXiv, 2023 (main reading)
- Liu et al. Empirical evaluation of scoring functions for Bayesian Network model selection, BMC Bioinformatics, 2012 (optional reading)

This and Next Week

Workshop (tomorrow):

Exercise on Conditional Independent tests

Exercise on scoring functions (BIC, AIC, BDeu)

Python code for structure learning

Lecture (next week):

Gaussian Bayesian Networks

Reading 1: Koller & Friedman 2009. Section 7.2

Reading 2: Bishop. C. 2006. Section 8.1.4

Questions?

Appendix 1

Metrics for Evaluating Probabilistic Models

Metrics for Probabilistic Models

- The higher the better:
  - Balanced accuracy, [0,…,1]
  - F1 Score, [0,…,1]
  - Area under curve, [0,…,1]
- The lower the better
  - Brier score, [0,…,1]
  - KL divergence, [0,…,∞]
  - Expected Calibration Loss, [0,…,1]
  - Training and inference times (in seconds)

Metrics: Balanced Accuracy

Actual Label or Condition	Predicted Label/Condition: POSITIVE (PP)	Predicted Label/Condition: NEGATIVE (PN)
POSITIVE (P)	**True Positive (TP)**	**False Negative (FN)**
NEGATIVE (N)	**False Positive (FP)**	**True Negative (TN)**

$$BA = \frac{TPR + TNR}{2}, \text{where}$$

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR \qquad FNR = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR$$

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR \qquad FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR$$

If $FNR = FPR = 0$ then $TPR + TNR = 2$

Metrics: F1 Score

Predicted Label/Condition

Actual Label or Condition	POSITIVE (PP)	NEGATIVE (PN)
POSITIVE (P)	**True Positive (TP)**	**False Negative (FN)**
NEGATIVE (N)	**False Positive (FP)**	**True Negative (TN)**

$$F_1 = 2 * \left(\frac{Precision * Recall}{Precision + Recall} \right), \text{where}$$

$$Precision = \frac{TP}{PP} = \frac{TP}{TP + FP}$$

Of all positive predictions, what fraction is actually positive.

$$Recall = \frac{TP}{P} = \frac{TP}{TP + FN}$$

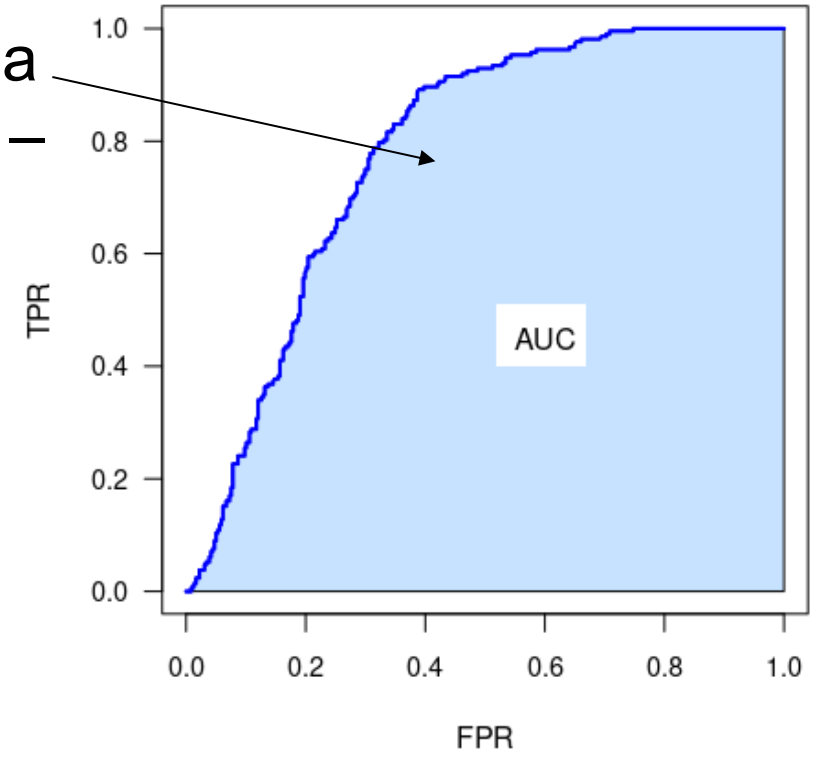
Of all actual positive instances, what fraction is correctly predicted as positive.

Metrics: Area Under the ROC Curve (AUC)

Predicted Label/Condition

Actual Label or Condition	POSITIVE (PP)	NEGATIVE (PN)
POSITIVE (P)	**True Positive (TP)**	**False Negative (FN)**
NEGATIVE (N)	**False Positive (FP)**	**True Negative (TN)**

AUC measures the entire area undern the entire ROC curve – the higher the area the better the classifier.



$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR$$

Metrics: Area Under the ROC Curve (AUC)

Actual Label or Condition	Predicted Label/Condition: POSITIVE (PP)	NEGATIVE (PN)
POSITIVE (P)	**True Positive (TP)**	**False Negative (FN)**
NEGATIVE (N)	**False Positive (FP)**	**True Negative (TN)**

AUC measures the entire area undern the entire ROC curve – the higher the area the better the classifier.

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR$$

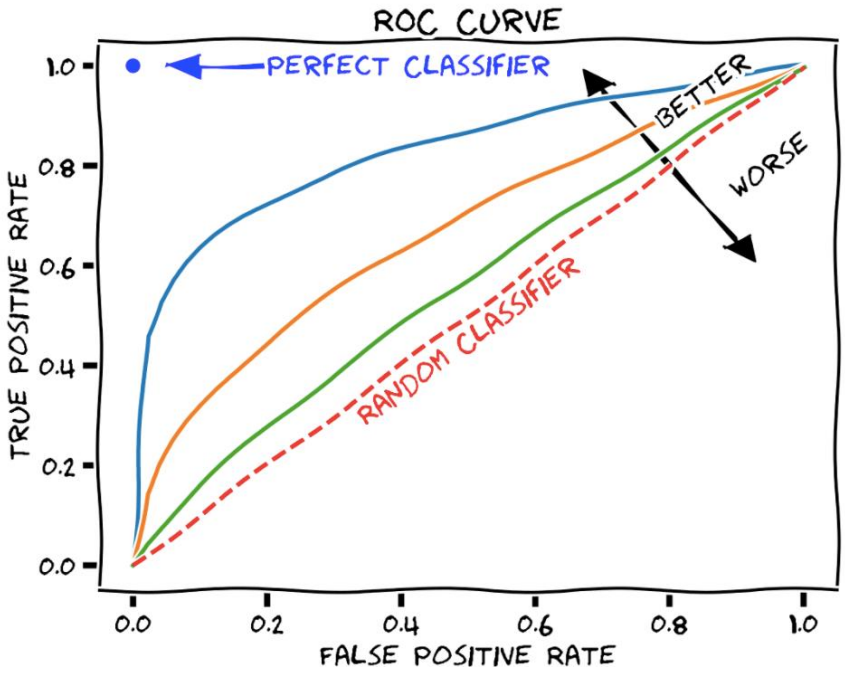


Figure from [this link]

Metrics for Probabilistic Models

- The higher the better:
  - Balanced accuracy, [0,…,1]
  - F1 Score, [0,…,1]
  - Area under curve, [0,…,1]
- The lower the better
  - Brier score, [0,…,1]
  - KL divergence, [0,…,∞]
  - Expected Calibration Loss, [0,…,1]
  - Training and inference times (in seconds)

Predicted probability

Correct/True probability

$$BS = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2$$

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

KL Divergence: Example (1/2)

- True Distribution P = <1.0, 0.0>
- Predicted distribution Q = <0.6, 0.4>

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

True/Correct distribution

Predicted distrubution

$$D_{KL}(P||Q) = P(x_1)\log\left(\frac{P(x_1)}{Q(x_1)}\right) + P(x_2)\log\left(\frac{P(x_2)}{Q(x_2)}\right)$$

$$= 1.0\log\left(\frac{1.0}{0.6}\right) + 0.0\log\left(\frac{0.0}{0.4}\right)$$

$$= 0.5108 + 0$$

KL Divergence: Example (2/2)

- True Distribution P = <1.0, 0.0>
- Predicted distribution Q = <0.4, 0.6>

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

$$D_{KL}(P||Q) = P(x_1) \log\left(\frac{P(x_1)}{Q(x_1)}\right) + P(x_2) \log\left(\frac{P(x_2)}{Q(x_2)}\right)$$

$$= 1.0 \log\left(\frac{1.0}{0.4}\right) + 0.0 \log\left(\frac{0.0}{0.6}\right)$$

$$= 0.9162 + 0$$

N.B. This example uses the natural logarithm (base e) – default in Python. Other choices are: $\log_{10}()$ and $\log_2()$

Expected Calibration Loss (ECL)

It measures the discrepancy between predicted probabilities and expected accuracy.

$$ECL = \sum_{i=1}^{N} b_i \cdot |p_i - c_i|$$

High ECL means the model is either overconfident or underconfident.

Low ECL means model well calibrated.

- N is the number of bins.
- b_i is the fraction of samples in the i_{th} bin.
- p_i is the avg. predicted probability in the i_{th} bin.
- c_i is the observed accuracy in the i_{th} bin.

Pavlovic, M. "Understanding Model Calibration – A gentle Introduction and Visual …", ICLR Blogposts, 2025.

Metrics for Probabilistic Models

- The higher the better:
  - Balanced accuracy, [0,…,1]
  - F1 Score, [0,…,1]
  - Area under curve, [0,…,1]

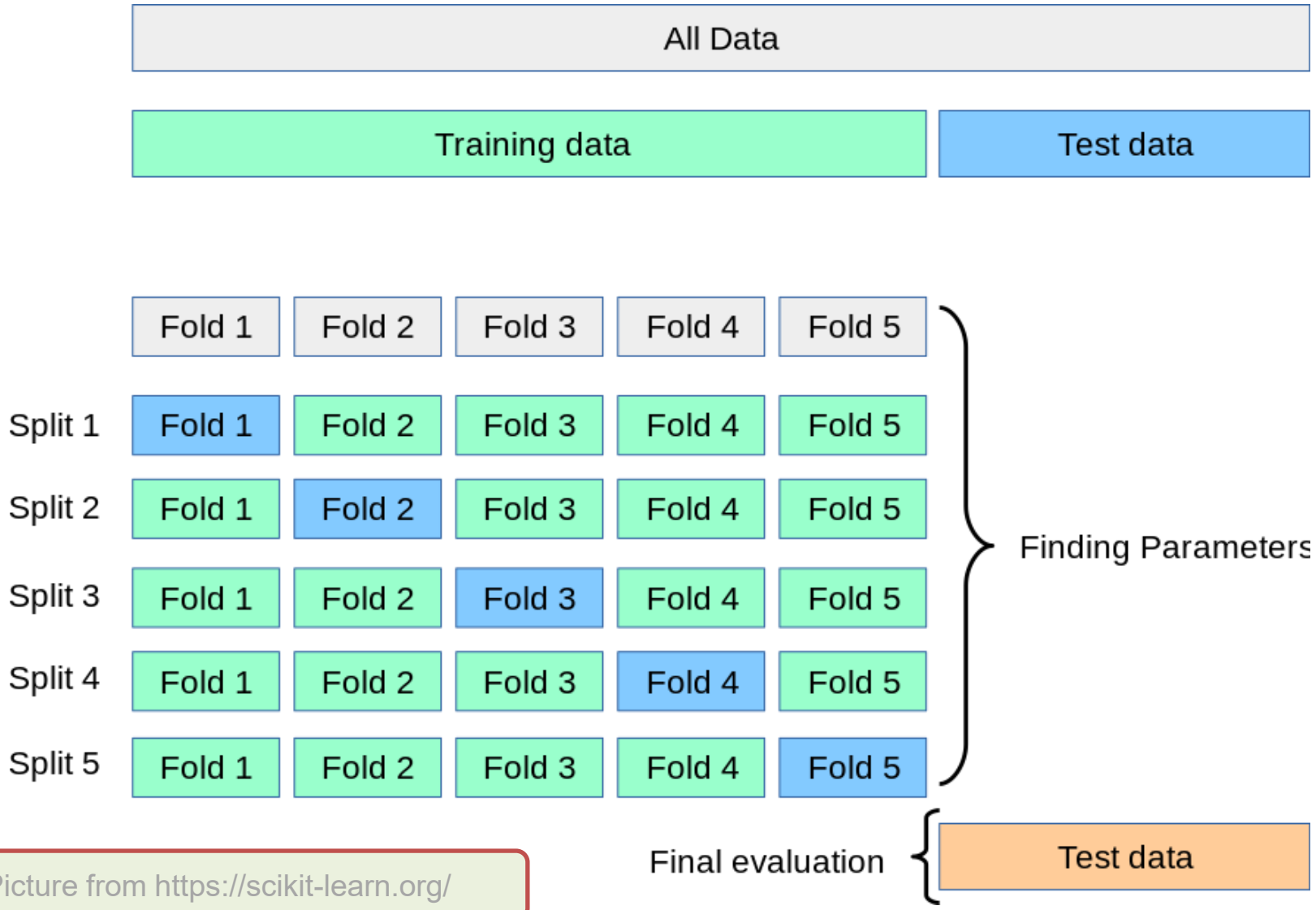
- The lower the better:
  - Brier score, [0,…,1]
  - KL divergence, [0,…,∞]
  - Expected Calibration Loss, [0,…,1]
  - Training and inference times (in seconds)

Consider using these metrics in your assignment. Look at **ModelEvaluator.py**. Fine if you wish to include any others.

Appendix 2

Evaluation of Probabilistic Models Using Cross Validation

K-fold Cross Validation



Picture from https://scikit-learn.org/

K-fold CV: Methodology for Pure Model Evaluation

Suitable for small datasets

1. Discretise your data, if required. (examples)
2. Split the entire dataset in K folds—randomly.
3. Train on K-1 folds and test using the remaining fold.
4. Iterate over all K consistent folds.
5. Average the performance according to difference metrics, and report standard deviation if considered relevant.
6. Report results for different models in a table.