Graphical Models - Bayesian Networks

Machine Learning

IIIy CSE IDP and IDDMP

Refer : Stephen Marsland

Motivation

1. Machine learning unites computer science (graphs, algorithms) and statistics (probability)

2. Probabilistic graphical models use basic graph structures (nodes + directed/ undirected edges) to encode complex distributions

3. Graph algorithms (shortest-path, cycle detection, etc.) provide mature computational tools

4. Nodes $\leftrightarrow$ random variables; edges $\leftrightarrow$ (conditional) dependencies

5. Absence of a direct link $\Rightarrow$ conditional independence (given intervening variables)

Directed Graphical Models (Bayesian Networks)

Directed vs. Undirected: focus here on directed graphs (Bayesian networks)

A directed edge A $\rightarrow$ B encodes

$$P(A,B) = P(A)\,P(B|A)$$

No direct link between two nodes $\Rightarrow$ they are conditionally independent

Specification:

One node per discrete random variable

Conditional Probability Table (CPT) for each node, given its parents

Node types:

Observed nodes: values directly seen

Hidden (latent) nodes: values to be inferred

Bayesian networks generalize to MRFs, HMMs, Kalman filters, particle filters

Bayesian Networks: Structure & Exam-Fear Example

Directed Acyclic Graph (DAG): no loops $\Rightarrow$ encodes a joint distribution via factorization

Bayesian Network = DAG + Conditional Probability Tables (CPTs)

Exam-Fear Example (Figure 16.2 - Next Slide):

Variables:

B = “Course boring”; A = “Attended lectures”; R = “Revised”; S = “Scared before exam"

Top-down (prediction): infer S from observed B, A, R

Bottom-up (diagnosis): infer causes (e.g. R or A) from observed S

Factorization:

$$P(s) = \sum_{b} P(b) \times \sum_{r,a} P(r|b) \times P(a|b) \times P(s|r,a)$$

Figure 16.2

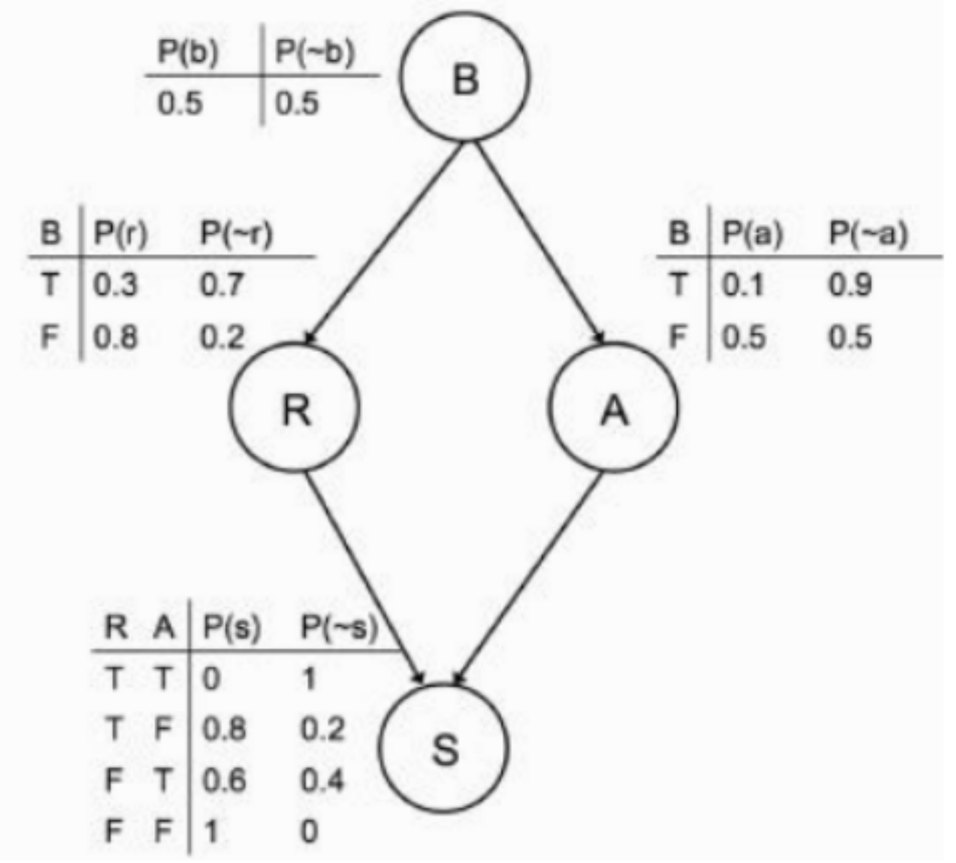

FIGURE 16.2 The sample graphical model. 'B' denotes a node stating whether the exam was boring, 'R' whether or not you revised, 'A' whether or not you attended lectures, and 'S' whether or not you will be scared before the exam.

Complexity of Exact Inference & Variable Elimination

Exact inference in general BNs is NP-hard (more precisely #P-hard) → exponential in number of variables

Polytrees (at most one undirected path) admit linear-time exact inference

Variable Elimination (VE):

1. Converts CPTs into "λ-tables"
2. Eliminates observed vars (remove rows/columns)
3. Marginalises hidden vars by multiplying & summing matching entries
4. Still exponential worst-case, but practical speed-ups via ordering

Scalability issue: adding one extra node (e.g. "Final year = F") increases table sizes dramatically **(Figure 16.3 - Next Slide)**

[Refer to "The Variable Elimination Algorithm" slide for detailed steps]

Figure 16.3

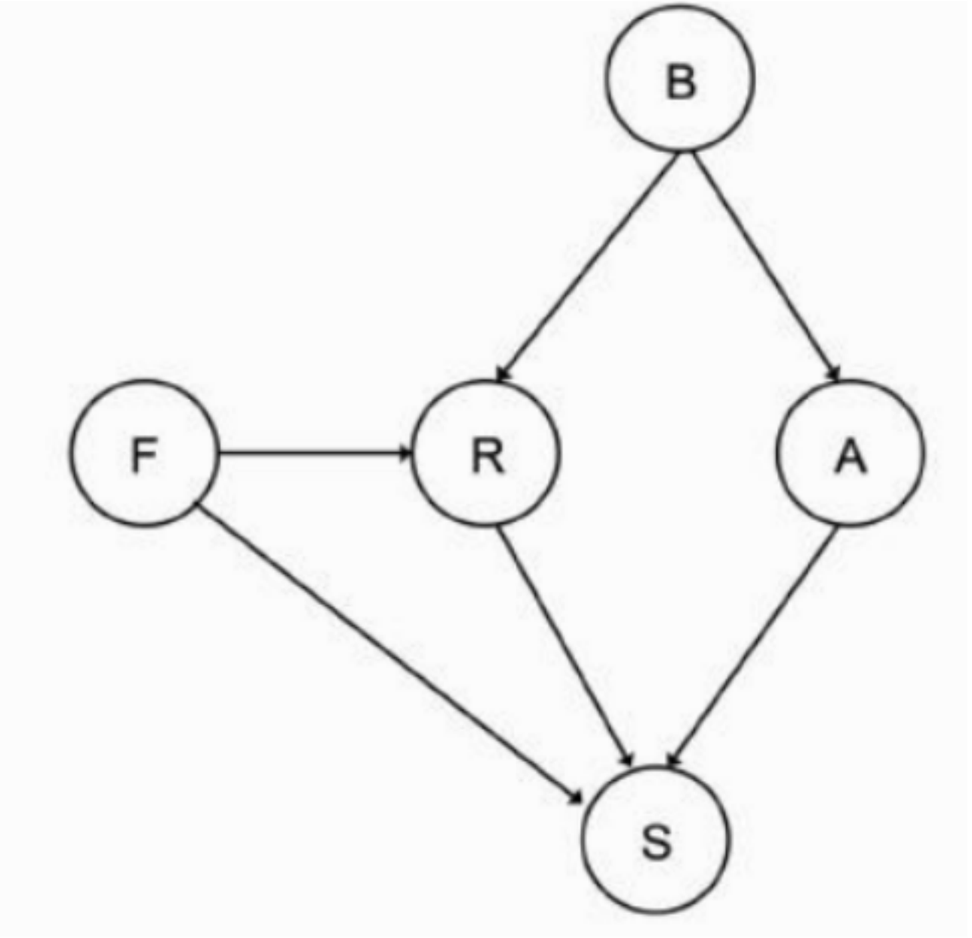

FIGURE 16.3 Adding just one extra node ('F', information about whether or not this is your final year) makes the conditional probability tables significantly more complicated.

The Variable Elimination Algorithm

The Variable Elimination Algorithm

- Create the λ tables:
  - for each variable v:
    - make a new table
    - for all possible true assignments x of the parent variables:
      - add rows for $P(v|x)$ and $1 - P(v|x)$ to the table
    - add this table to the set of tables
- Eliminate known variables v:
  - for each table:
    - remove rows where v is incorrect
    - remove column for v from table

Cont. In next slide

- Eliminate other variables (where x is the variable to keep):
  - for each variable v to be eliminated:
    - create a new table t'
    - for each table t containing v:
      - $v_{\text{true},t} = v_{\text{true},t} \times P(v|x)$
      - $v_{\text{false},t} = v_{\text{false},t} \times P(\neg v|x)$
    - $v_{\text{true},t'} = \sum_t (v_{\text{true},t})$
    - $v_{\text{false},t'} = \sum_t (v_{\text{false},t})$
  - replace tables t with the new one t'
- Calculate conditional probability:
  - for each table:
    - $x_{\text{true}} = x_{\text{true}} \times P(x)$
    - $x_{\text{false}} = x_{\text{false}} \times P(\neg x)$
    - probability is $x_{\text{true}}/(x_{\text{true}} + x_{\text{false}})$

Approximate Inference with MCMC

Why approximate?

1. Exact inference (Variable Elimination) is NP-hard for large BNs
2. MCMC methods offer scalable, anytime approximations

Forward (Ancestral) Sampling:

3. Traverse DAG top-down
4. Sample each variable from its CPT given sampled parents
5. Repeat → empirical distribution converges as sample size ↑

Rejection Sampling:

6. Draw independent samples from joint prior
7. Discard those inconsistent with observed evidence
8. Simple but inefficient if evidence is unlikely

Likelihood-Weighted Sampling:

9. Incorporate evidence by weighting each sample by P(evidence | sampled parents)
10. Reduces wasteful rejections

Gibbs Sampling in Bayesian Networks

Full MCMC Framework:

Initialize all variables (observed = fixed, hidden = random)

Iterate: resample each hidden variable from its conditional given current rest

Markov Blanket : For a node x_j, its "blanket" includes :

1. Parents of x_j (Denoted by $x_{\alpha j}$)
2. Children of x_j ($\beta(j)$ is the set of children of node x_j)
3. Other parents of those children (co-parents)

Gibbs Sampling Step:

For each hidden x_j, sample from $P(x_j \mid \text{Markov blanket})$

Repeat until chain mixes → approximate posterior

In a Bayesian network, any given variable is independent of any node that is not their child, given their parents. So we can w

$$p(x_j|x_{-j}) = p(x_j|x_{\alpha j}) \prod_{k\in\beta(j)} p(x_k|x_{\alpha(k)})$$

The Markov blanket of a node

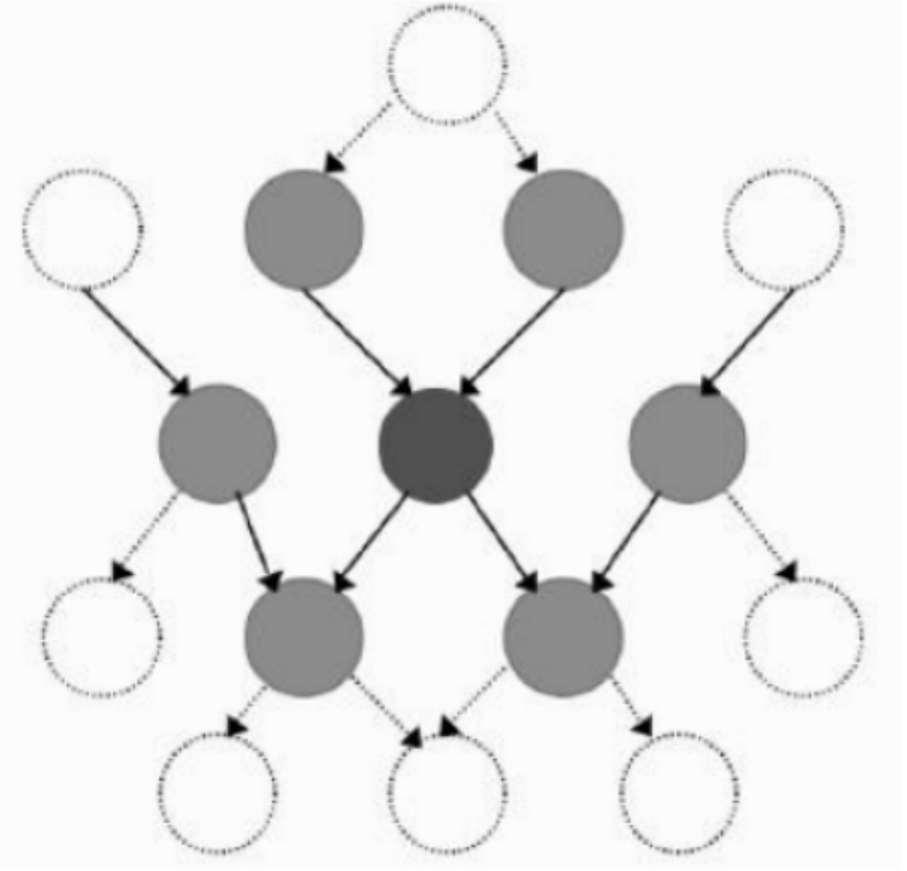

FIGURE 16.4 The Markov blanket of a node is the set of nodes (shaded light grey) that are either parents or children of the node, or other parents of its children (shaded dark grey).

Learning Structure of Bayesian Networks

1. **Hand-built networks:** common but subjective and time-consuming
2. **Combinatorial explosion:**
    a. Number of DAGs over N nodes grows super-exponentially
    b. Even for N = 10, possible graphs ≈ 10^{18} → exhaustive search infeasible
3. **Search approaches:**
    c. Define a scoring function combining data-fit and model simplicity (e.g., MDL or BIC)
    d. Use local search (hill climbing, greedy search) from random starts
    e. Genetic algorithms are theoretically possible but too costly (each evaluation requires inference)
4. **Key challenge:**
    Balancing goodness-of-fit against over-complexity (Occam's razor)

Learning Parameters of Bayesian Networks

Given a fixed DAG structure: compute each conditional probability table (CPT) by maximizing data likelihood

Complete-data case (no hidden nodes):

1. Count frequencies of each child-parent value combination in the data
2. Normalize counts to probabilities

$$L = \frac{1}{M} \sum_{n=1}^{N} \sum_{m=1}^{M} \log P(X_n | \text{parents}(X_n), D_m)$$

With hidden nodes:

3. Use EM algorithm:
    a. **E-step:** infer expected counts for hidden variables (via inference)
    b. **M-step:** update CPT entries to maximize expected likelihood
4. Iterate until convergence

Outcome: learned CPTs that best explain the observed data given the expert-defined structure.

Summary & Conclusion

1. **Graphical models bridge CS and statistics**
    a. Encode complex joint distributions via graph structure + local CPTs
2. **Bayesian Networks (DAGs)**
    b. Directed, acyclic graphs factorize the joint probability
    c. Exact inference (variable elimination) is NP-hard in general; polytrees are tractable
3. **Approximate Inference**
    d. MCMC methods (forward sampling, rejection, likelihood-weighting, Gibbs) scale to large networks
    e. Gibbs sampling leverages the Markov blanket for efficient local updates
4. **Learning a BN**
    f. **Structure learning**: combinatorial search with MDL/BIC scoring and local search heuristics
    g. **Parameter learning**:
        i. With complete data: count-and-normalize frequencies
        ii. With hidden variables: EM algorithm alternates inference and maximization
5. **Takeaway**
    h. Graphical models provide a flexible, interpretable framework for probabilistic reasoning
    i. Inference and learning remain computationally challenging, requiring approximations.