CS 583: Probabilistic Graphical Models

Topic: Structure Learning

Mustafa Bilgic

http://www.cs.iit.edu/~mbilgic

https://twitter.com/bilgicm

Task

- We are given fully observed data
- Our task is to learn the structure (and the parameters)

Goal: knowledge discovery

- There are at least two problems
  - Unidentifiability
    - Given observational data, the best we can hope to learn is an I-Equivalent structure to G*
  - Weak connections
    - Due to noise, two variables will almost never be independent
    - If a connection is weak, how do we determine if it is due to noise or it is a real weak connection?

Goal: density estimation

- Adding an unnecessary edge is <u>very costly</u> for parameter estimation
  - Each unnecessary binary parent divides the data into half – data fragmentation
    - The estimated parameters have high variance
- For better generalization, it is preferable to have a sparse structure
- ***In fact, if the data is limited, it might be even better to have a sparser structure than G****

Bayesian Networks

Three approaches

1. *Constraint-based structure learning*
   - View Bayesian network as a representation of independencies
   - Test for conditional dependence and independence in the data
   - Sensitive to failure in individual independence tests
2. *Score-based structure learning*
   - Hypothesis space of possible models
   - Find the highest scoring structure
   - The hypothesis space is super exponential $2\text{\textasciicircum}(O(n^2))$
3. *Bayesian model averaging*
   - Rather than learning a structure, learn an ensemble of structures
   - Again, the number of structures is immense
   - It is efficient for some while it has to be approximate for others

1. Constraint-based approaches

- In the BN slide deck, we have seen two procedures

1. Find minimal I-MAP
2. Find all I-equivalent structures

- We assumed that, given a distribution P, we could answer any independence question perfectly
- We'll now assume that we don't have access to P but instead we have access to a dataset D

INDEPENDENCE IN THE DATA?

- Assume we toss two coins independently
- We first toss one coin, record it, and then toss the next coin, and record the outcome
- We repeat this process 100 times
- Assume we get 25 H/H, 23 H/T, 27 T/H, 25 T/T
- Based on the data, can we say that the two coins are independent?

Independence tests

- $\mathcal{X}^2$ statistic

$$d_{\chi^2}(D) = \sum_{x,y} \frac{\left(M[x,y] - M \times \hat{P}(x) \times \hat{P}(y)\right)^2}{M \times \hat{P}(x) \times \hat{P}(y)}$$

- Mutual information

$$d_I(D) = \sum_{x,y} \hat{P}(x,y) \log \frac{\hat{P}(x,y)}{\hat{P}(x)\hat{P}(y)}$$

- $\mathcal{X}^2$ and MI are zero when X and Y are independent
- Extension of $\mathcal{X}^2$ statistic to conditional independence queries

$$d_{\chi^2}(D) = \sum_{x,y,z} \frac{\left(M[x,y,z] - M \times \hat{P}(z) \times \hat{P}(x \mid z) \times \hat{P}(y \mid z)\right)^2}{M \times \hat{P}(z) \times \hat{P}(x \mid z) \times \hat{P}(y \mid z)}$$

Three approaches

1. *Constraint-based structure learning*
   - View Bayesian network as a representation of independencies
   - Test for conditional dependence and independence in the data
   - Sensitive to failure in individual independence tests
2. *Score-based structure learning*
   - Hypothesis space of possible models
   - Find the highest scoring structure
   - The hypothesis space is super exponential $2\text{^}(O(n^2))$
3. *Bayesian model averaging*
   - Rather than learning a structure, learn an ensemble of structures
   - Again, the number of structures is immense
   - It is efficient for some while it has to be approximate for others

2. Score-based approaches

- 2.1 Maximum likelihood score
- 2.2. Bayesian score

2.1 Maximum likelihood score

- Find a model, $<\mathcal{G}, \theta_{\mathcal{G}}>$ that would make the data D as probable as possible

$$\begin{aligned}\max_{\mathcal{G},\theta_{\mathcal{G}}} L\left(\left\langle \mathcal{G},\theta_{\mathcal{G}}\right\rangle : D\right) &= \max_{\mathcal{G}}\left[\max_{\theta_{\mathcal{G}}} L\left(\left\langle \mathcal{G},\theta_{\mathcal{G}}\right\rangle : D\right)\right] \\ &= \max_{\mathcal{G}} L\left(\left\langle \mathcal{G},\hat{\theta}_{\mathcal{G}}\right\rangle : D\right)\end{aligned}$$

- In order to find the maximum likelihood pair $<\mathcal{G}, \theta_{\mathcal{G}}>$
  - We should find the structure $\mathcal{G}$ that achieves the highest likelihood when we use the MLE parameters for $\mathcal{G}$

$$score_L\left(\mathcal{G} : D\right) = l\left(\hat{\theta}_{\mathcal{G}} : D\right)$$

Two variable case

- Given two variables X and Y
- Consider two structures: $\mathcal{G}_\varnothing$ and $\mathcal{G}_{X\to Y}$.
- What is $\text{score}_L(\mathcal{G}_{X\to Y} : D)$ - $\text{score}_L(\mathcal{G}_\varnothing : D)$?
- The general equation for $\text{score}_L(\mathcal{G} : D)$ is as follows. Proof is left as an exercise.

$$score_L(\mathcal{G}:D) = M\sum_{i=1}^{n} \boldsymbol{I}_{\hat{P}}\left(X_i; Pa_{X_i}^{\mathcal{G}}\right) - M\sum_{i=1}^{n} \boldsymbol{H}_{\hat{P}}\left(X_i\right)$$

Limitations of ML score

- Can $\text{score}_L(\mathcal{G}_{\text{X}\rightarrow\text{Y}} : D)$ be smaller than $\text{score}_L(\mathcal{G}_{\varnothing} : D)$ ever?
- They are equal only when X and Y are truly independent and otherwise $\text{score}_L(\mathcal{G}_{\text{X}\rightarrow\text{Y}} : D)$ is always bigger
- Exercise: Show that $\boldsymbol{I_P}(\boldsymbol{X}; \boldsymbol{Y} \cup \boldsymbol{Z}) \geq \boldsymbol{I_P}(\boldsymbol{X}; \boldsymbol{Y})$
- Adding a parent can never decrease the maximum likelihood score
- The highest scoring structure is the complete graph
- One remedy is to bound the indegree of the graph

2.2 Bayesian score

- Bayesian approach puts a distribution on anything we are uncertain about
  - In parameter estimation, we put a distribution on θ
- In this case, we have the following distributions
  - $P(\mathcal{G})$
  - $P(\theta_{\mathcal{G}} \mid \mathcal{G})$
- Given data D, the posterior $P(\mathcal{G} \mid D)$ is
  - $P(\mathcal{G} \mid D) = P(D \mid \mathcal{G})\, P(\mathcal{G}) / P(D)$
- P(D) is a normalizing constant
- $\text{score}_B(\mathcal{G} : D) = \log P(D \mid \mathcal{G}) + \log P(\mathcal{G})$

P(D | $\mathcal{G}$)

$$P(D \mid \mathcal{G}) = \int_{\Theta_{\mathcal{G}}} P(D \mid \theta_{\mathcal{G}}, \mathcal{G}) P(\theta_{\mathcal{G}} \mid \mathcal{G}) d\theta_{\mathcal{G}}$$

- There is a closed form solution on page 801 of the book
- If we use a Dirichlet priors for all parameters in the network, as $M \to \infty$

$$\log P(D \mid \mathcal{G}) = l\left(\hat{\theta}_{\mathcal{G}} : D\right) - \frac{\log M}{2} \mathrm{Dim}[\mathcal{G}] + O(1)$$

- $l(\theta$:D) is the loglikelihood and Dim[$\mathcal{G}$] is the model dimension, i.e., the number of independent parameters in $\mathcal{G}$

BIC SCORE

$$\begin{aligned} score_{BIC}(\mathcal{G} : D) &= l(\hat{\theta}_{\mathcal{G}} : D) - \frac{\log M}{2}\mathrm{Dim}[\mathcal{G}] \\ &= M\sum_{i=1}^{n} \boldsymbol{I}_{\hat{P}}\left(X_i; Pa_{X_i}^{\mathcal{G}}\right) - M\sum_{i=1}^{n} \boldsymbol{H}_{\hat{P}}(X_i) - \frac{\log M}{2}\mathrm{Dim}[\mathcal{G}] \end{aligned}$$

- A trade-off between the fit to the data and the model complexity
- The mutual information grows linearly in *M* whereas the model complexity grows logarithmically
  - The larger the *M* the more emphasis on fit to the data

Score equivalence

- A score satisfies score equivalence if for all I-Equivalent structures $\mathcal{G}$ and $\mathcal{G}'$, score($\mathcal{G} : D$) = score($\mathcal{G}' : D$)
- The likelihood score satisfies score equivalence
- The Bayesian score satisfies score equivalence if and only if we use BDe priors

DECOMPOSABLE SCORE

- A structure score score($\mathcal{G}$: D) is decomposable if it can be written as
$$score(\mathcal{G} : D) = \sum_i FamScore\left(X_i \mid Pa_{X_i}^{\mathcal{G}} : D\right)$$
where FamScore(X| $\boldsymbol{U}$: D) measures how well a set of variables $\boldsymbol{U}$ serves as parents of X in D.
- The likelihood score is decomposable
  - FamScore(X| $\boldsymbol{U}$: D) = M*[$\boldsymbol{I}(X;\boldsymbol{U}) - \boldsymbol{H}(X)$]
- The Bayesian score is decomposable under certain natural assumptions

Why is decomposability important?

- When searching for structures, small changes to the structure change the score of the structure, but
- The score of the structure can be updated efficiently by recomputing the scores for only the affected families
- For example
  - Adding or deleting an edge changes the score for only one variable
  - Reversing an edge changes the score for only two variables

Structure search

- So far, we have discussed computing a score for a given structure
- How can we find the highest scoring structure?
- Three scenarios
  1. A tree-structured network
  2. Known variable order
  3. General graphs

Tree-structured networks

- Each variable has at most one parent
- Can be learned efficiently – in polynomial time
- They capture the most important dependencies and they serve as a good baseline
- Define $\Delta(\mathcal{G}) = score(\mathcal{G} : D) - score(\mathcal{G}_\varnothing : D)$
- We know $score(\mathcal{G}_\varnothing : D) = \Sigma_i \text{FamScore}(X_i : D)$
- $\Delta(\mathcal{G}) = \Sigma_j \text{FamScore}(X_j \mid \text{Pa}(X_j) : D) - \text{FamScore}(X_j : D)$
- Algorithm
  - Define a fully connected undirected graph where the edge weight between X_i and X_j is $\text{FamScore}(X_i \mid X_j : D) - \text{FamScore}(X_i : D)$
  - Find the maximum spanning forest
- Complexity: $O(n^2M + n^2\log n)$

Known variable order

- A variable order is given
  - Assuming a variable order is problematic, but in some cases, the order is natural, such as a temporal one
- A variable's parents can be only the variables that precede it

$$Pa^{\mathcal{G}}_{X_i} = \underset{\mathbf{U}_i \subseteq \{X_j : X_j \prec X_i\}}{\arg\max} FamScore(X_i \mid \mathbf{U}_i : D)$$

- To find the parent of the nth node, we need to consider $2^{n\text{-}1}$ potential sets
- We can bound the indegree to be d; then the algorithm is exponential in d but polynomial in n

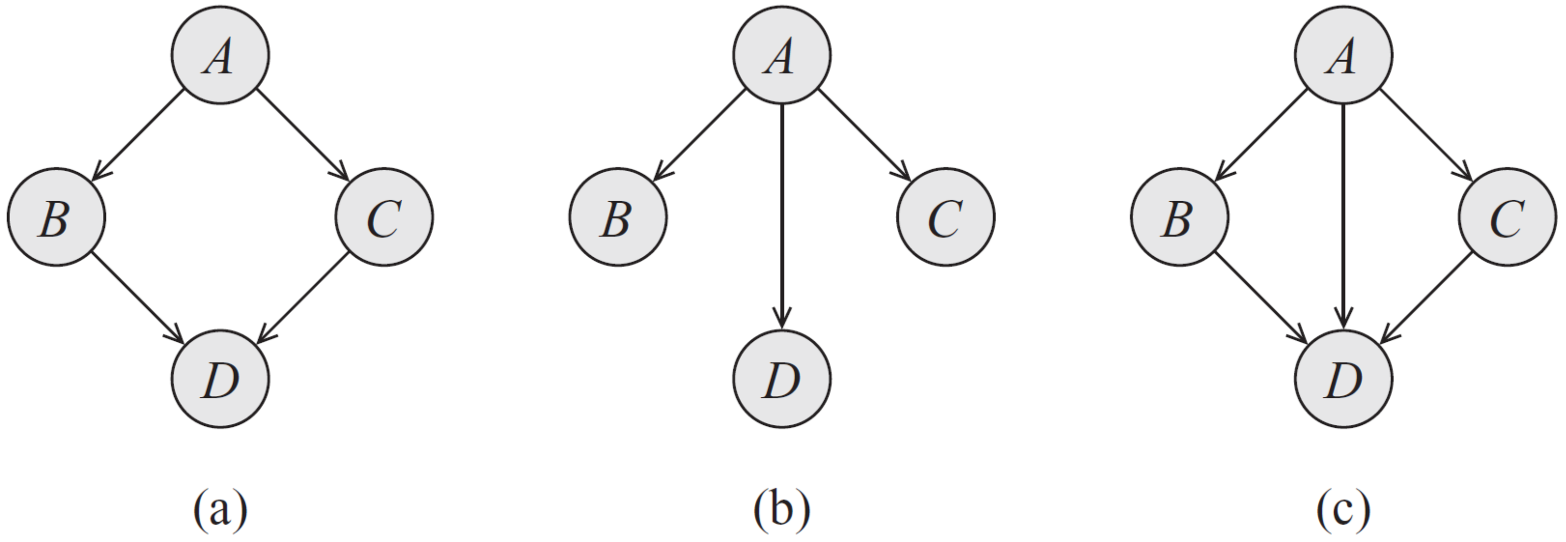
General graphs

- Finding the best scoring $\mathcal{G}$ is NP-hard
- Even if we limit the indegree to be d, finding the best scoring $\mathcal{G}$ is NP-hard when $d \geq 2$
- Heuristic algorithms that might find the best one but not guaranteed
- This is a combinatorial optimization problem
  - Search the search space to find a good scoring $\mathcal{G}$
  - Use local search
- Need three components
  1. The search space
  2. Scoring function
  3. A search procedure

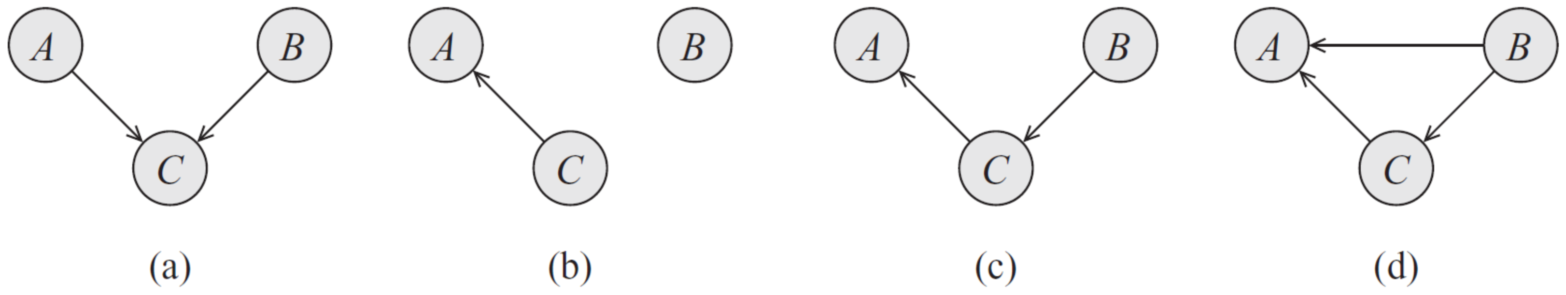
Search space

- The search space is a graph
  - Whose nodes are potential solutions,
  - Connected by search operators that allow us to move from one candidate solution to another
- We choose the following search operators
  - Edge addition
  - Edge deletion
  - Edge reversal
- Why can't we just start from $\mathcal{G}_\emptyset$ and use only edge addition operators? Why do we need edge deletion and reversal?
- When is edge reversal useful? Can't we just do it in two steps by deleting an edge and adding back it in the reverse direction?

Edge Deletion Example – Fig 18.5



Edge Reversal Example – Fig 18.6



Search procedure

- Start with an initial graph
    - It can be the empty graph, a random graph, the best tree, or a network that is hand constructed
- Apply the search operators to generate new candidate structures, score them, and move to the one that has the highest score
- The runtime is roughly $O(K*n^2*(M+n*d))$
    - K is the number of iterations,
    - n is the number of variables,
    - M is the number of data instances,
    - d is the max indegree

Local maxima

- What happens if none of the operators find a better scoring structure than the current one?
  - We hit either a *local maxima* or a *plateau*
- Unfortunately, plateaus are pretty common
  - When the scoring function satisfies score equivalence, then all the I-equivalent structures have equal score
  - The edge reversal operator will generate many I-equivalent structures

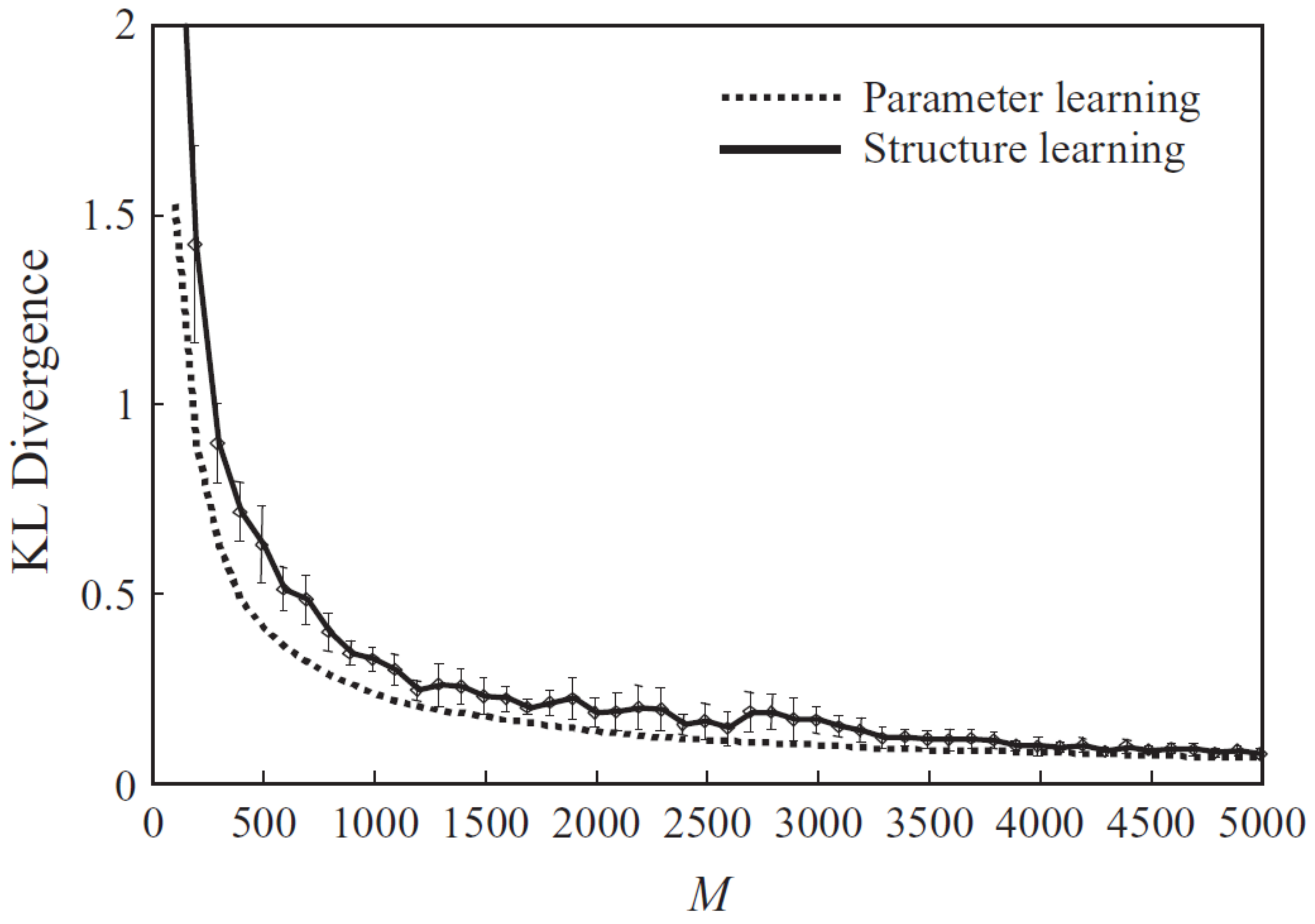
How to deal with local maxima and plateaus?

- *Basin flooding*
  - Keep track of visited states
- *Tabu search*
  - Keep a list of recently performed operations and do not reverse their effect in the next L steps
- *Data perturbation*
  - Make small changes to data D through replicas and removals.
  - The new data will cause similar networks to differ, but the big trends will be preserved

How score decomposition comes in handy

- Adding and deleting an edge changes the family of one variable and reversal changes the family of two variables
- We keep track of how much the score would change for each candidate operation
- Let's say we are considering to add the edge $X \rightarrow Y$.
  - We compute the score change, but we decide to do a better operation which does not change Y's family.
  - In the next step, we do not need to compute the score change for $X \rightarrow Y$; we can use the previous computation.

How good is local search?



SUMMARY

- Not all the dependencies seen in the data are real
- Determining the edge directionality is sometimes impossible
- We have seen two structure learning approaches
    - Constraint-based
        - Sensitive to independence failures, which are common in real-world datasets
    - Score-based
        - Maximum likelihood score
        - Bayesian Information Criterion (BIC) score
- Tree-structured networks can be learned in polynomial time
- General structure learning is NP-hard but there are many local search algorithms that perform fairly well in practice

Markov Networks

Structure learning

- Two approaches
  - Constraint-based
  - Score-based

Constraint-based

- Assume $\mathcal{H}^*$ is a P-Map for P^*
- We already know the algorithm
  - Use the first step of constraint-based structure learning for Bayesian networks

SCORE-BASED

- To compute the score for a candidate structure, we need to estimate the parameters
  - Unlike Bayesian networks, parameter estimation cannot be done independently for each variable
  - Unlike Bayesian networks, parameter estimation requires running inference
- Unlike Bayesian networks, the score for a structure does not decompose over variables
  - There is no clean and efficient way of reusing computations

In Practice

- Log-linear models are learned
- Search is performed over candidate features (over candidate cliques)
- L_1-regularization is used so that the weights of the useless features become zero
  - If all the weights for all features for a candidate clique become zero, that clique is gone
  - If there is a single feature with a non-zero weight, the clique is retained
  - To achieve sparse solutions, *block L_1-reqularization* is used where a block of features for a candidate clique are penalized together