Inference Complexity As Learning Bias

Pedro Domingos
Dept. of Computer Science & Eng.
University of Washington

Joint work with Daniel Lowd
Don’t use model complexity as your learning bias …

Use inference complexity.
The Goal

This talk:
• How to learn accurate and efficient models by tightly integrating learning and inference
• Experiments: exact inference in < 100ms in models with treewidth > 100
Outline

• Standard solutions (and why they fail)
• Background
  – Learning with Bayesian networks
  – Inference with arithmetic circuits
• Learning arithmetic circuits
  – Scoring
  – Search
  – Efficiency
• Experiments
• Conclusion
Solution 1: Exact Inference

Data → Model → Jointree → Answers!

- Data
- Model: A → B → C
- Jointree: SLOW
- Answers: SLOW
Solution 2: Approximate Inference

Approximations are often too inaccurate. More accurate algorithms tend to be slower.
Solution 3: Learn a tractable model

Related work: Thin junction trees

Polynomial in data, but still exponential in treewidth

[E.g.: Chechetka & Guestrin, 2007]
Thin junction trees are thin

- Maximum effective treewidth is 2-5
- We have learned models with treewidth >100
Solution 3: Learn a tractable model

Our work: Arithmetic circuits with penalty on circuit size

Data → Circuit → Answers!

Model

\( (A \times B) + C \)
Outline

• Standard solutions (and why they fail)
• **Background**
  – Learning with Bayesian networks
  – Inference with arithmetic circuits
• Learning arithmetic circuits
  – Overview
  – Optimizations
• Experiments
• Conclusion
Bayesian networks

**Problem:** Compactly represent probability distribution over many variables

**Solution:** Conditional independence

\[ P(A,B,C,D) = P(A) \ P(B|A) \ P(C|A) \ P(D|B,C) \]
… With decision-tree CPDs

**Problem:** Number of parameters is exponential in the maximum number of parents

**Solution:** Context-specific independence

\[
P(D|B,C) = \begin{cases} 
0.2 & \text{true} \\
0.5 & \text{false} 
\end{cases} \]
... Compiled to circuits

**Problem:** Inference is exponential in treewidth

**Solution:** Compile to arithmetic circuits
Arithmetic circuits

- Directed, acyclic graph with single root
  - Leaf nodes are inputs
  - Interior nodes are addition or multiplication
  - Can represent any distribution
- Inference is linear in model size!
  - Never larger than junction tree
  - Can exploit local structure to save time/space
ACs for Inference

- Bayesian network:
  \[ P(A, B, C) = P(A) P(B) P(C|A, B) \]
- Network polynomial:
  \[ \lambda_A \lambda_B \lambda_C \theta_A \theta_B \theta_{C|AB} + \lambda_{\neg A} \lambda_B \lambda_C \theta_{\neg A} \theta_B \theta_{C|\neg AB} + \ldots \]
- Can compute arbitrary marginal queries by evaluating network polynomial.
- Arithmetic circuits (ACs) offer efficient, factored representations of this polynomial.
- Can take advantage of local structure such as context-specific independence.
BN Structure Learning
[Chickering et al., 1996]

- Start with an empty network
- Greedily add splits to decision trees one at a time, enforcing acyclicity

\[
\text{score}(C,T) = \log P(T|C) - k_p n_p(C)
\]
(accuracy – # parameters)
Key Idea

For an arithmetic circuit $C$ on an i.i.d. training sample $T$:

Typical cost function:

$$\text{score}(C,T) = \log P(T|C) - k_p \ n_p(C)$$

(accuracy – # parameters)

Our cost function:

$$\text{score}(C,T) = \log P(T|C) - k_p \ n_p(C) - k_e \ n_e(C)$$

(accuracy – # parameters – circuit size)
Basic algorithm

Following Chickering et al. (1996), we induce our statistical models by greedily selecting splits for the decision-tree CPDs. Our approach has two key differences:

1. We optimize a different objective function

2. We return a Bayesian network that has already been compiled into a circuit
Efficiency

Compiling each candidate AC from scratch at each step is too expensive.

Instead: Incrementally modify AC as we add splits.
Algorithm

Create initial product of marginals circuit
Create initial split list
Until convergence:
  For each split in list
    Apply split to circuit
    Score result
    Undo split
  Apply highest-scoring split to circuit
  Add new child splits to list
  Remove inconsistent splits from list
Before split
After split
How to split a circuit

D: Parameter nodes to be split
V: Indicators for the splitting variable
M: First mutual ancestors of D and V

For each indicator $\lambda$ in V,
  Copy all nodes between M and D or V, conditioned on $\lambda$.
For each m in M,
  Replace children of m that are ancestors of D or V with a sum over copies of the ancestors times the $\lambda$ each copy was conditioned on.
Optimizations

We avoid rescoring splits every iteration by:
1. Noting that likelihood gain never changes, only number of edges added
2. Evaluating splits with higher likelihood gain first, since likelihood gain is an upper bound on score.
3. Re-evaluate number of edges added only when another split may have affected it (AC-Greedy).
4. Assume the number of edges added by a split only increases as the algorithm progresses (AC-Quick).
Experiments

We applied our algorithms (AC-Greedy, AC-Quick) to three real-world datasets, using the WinMine Toolkit as the baseline. WinMine’s algorithm is very similar to that of Chickering et al. (1996).

For inference, we generated queries from the test data with varying numbers of evidence and query variables. We used Gibbs sampling on the WinMine models since exact inference was not feasible.
Learning time

![Bar chart showing learning time for datasets KDD Cup, MSWeb, and EachMovie. The x-axis represents datasets, and the y-axis represents learning time in minutes. Bars are color-coded as follows: AC-Greedy in dark brown, AC-Quick in red, and WinMine in blue.](image-url)
Inference time

![Inference time graph]

- KDD Cup
- MSWeb
- EachMovie

- AC-Greedy
- AC-Quick
- Gibbs-1k
- Gibbs-10k
- Gibbs-100k
- Gibbs-1M

Inference time (s)
Accuracy: KDD Cup

Query variables

- Gibbs-F
- Gibbs-M
- Gibbs-S
- Gibbs-V
- AC-G
- AC-Q
Conclusion

• **Problem:** Learning accurate intractable models = Learning inaccurate models

• **Solution:** Use inference complexity as learning bias

• **Algorithm:** Learn arithmetic circuits with penalty on circuit size

• **Result:** Much faster and more accurate inference than standard Bayes net learning
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>EachMovie</th>
<th>KDD Cup</th>
<th>MSWeb</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC-Greedy</td>
<td>62ms</td>
<td>194ms</td>
<td>91ms</td>
</tr>
<tr>
<td>AC-Quick</td>
<td>162ms</td>
<td>198ms</td>
<td>115ms</td>
</tr>
<tr>
<td>Gibbs: 1k</td>
<td>7.22s</td>
<td>1.46s</td>
<td>1.89s</td>
</tr>
<tr>
<td>Gibbs: 10k</td>
<td>42.5s</td>
<td>11.3s</td>
<td>15.6s</td>
</tr>
<tr>
<td>Gibbs: 100k</td>
<td>452s</td>
<td>106s</td>
<td>154s</td>
</tr>
<tr>
<td>Gibbs: 1M</td>
<td>3912s</td>
<td>1124s</td>
<td>1556s</td>
</tr>
<tr>
<td>Dataset</td>
<td>AC-Greedy</td>
<td>AC-Quick</td>
<td>WinMine</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>EachMovie</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log-likelihood</td>
<td>–55.7</td>
<td>–54.9</td>
<td>–53.7</td>
</tr>
<tr>
<td>Edges</td>
<td>155k</td>
<td>372k</td>
<td></td>
</tr>
<tr>
<td>Leaves</td>
<td>4070</td>
<td>6521</td>
<td>4830</td>
</tr>
<tr>
<td>Treewidth</td>
<td>35</td>
<td>54</td>
<td>281</td>
</tr>
<tr>
<td>Time</td>
<td>&gt;72h</td>
<td>22h</td>
<td>3m</td>
</tr>
<tr>
<td><strong>KDD Cup</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log-likelihood</td>
<td>–2.16</td>
<td>–2.16</td>
<td>–2.16</td>
</tr>
<tr>
<td>Edges</td>
<td>382k</td>
<td>365k</td>
<td></td>
</tr>
<tr>
<td>Leaves</td>
<td>4574</td>
<td>4463</td>
<td>2267</td>
</tr>
<tr>
<td>Treewidth</td>
<td>38</td>
<td>38</td>
<td>53</td>
</tr>
<tr>
<td>Time</td>
<td>50h</td>
<td>3h</td>
<td>3m</td>
</tr>
<tr>
<td><strong>MSWeb</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log-likelihood</td>
<td>–9.85</td>
<td>–9.85</td>
<td>–9.69</td>
</tr>
<tr>
<td>Edges</td>
<td>204k</td>
<td>256k</td>
<td></td>
</tr>
<tr>
<td>Leaves</td>
<td>1353</td>
<td>1870</td>
<td>1710</td>
</tr>
<tr>
<td>Treewidth</td>
<td>114</td>
<td>127</td>
<td>118</td>
</tr>
<tr>
<td>Time</td>
<td>8h</td>
<td>3h</td>
<td>2m</td>
</tr>
</tbody>
</table>
Compiling Bayes nets

• Exact inference: Jointree algorithm
• AC may be more efficient because it can exploit local structure, including determinism and context-specific independencies
Context-specific independence

- Can represent many distributions more efficiently with local structure
- We will focus on Bayesian networks where the conditional probability distributions (CPDs) are represented by decision trees
- This allows compact distributions even for nodes with many parents
- ACs can exploit local structure