Inference Complexity As Learning Bias

Daniel Lowd
Dept. of Computer and Information Science
University of Oregon

Joint work with Pedro Domingos

Don’t use model complexity as your learning bias …

Use inference complexity.

The Goal

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

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

Their junction trees

Our junction trees

• 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

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

\[
P(D|B,C) = \begin{cases} 
0.2 & \text{true} \\
0.5 & \text{false} \\
0.7 & \text{true} \\
\end{cases}
\]
… 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

\[
\text{score}(C,T) = \log P(T|C) - k_p \pi_p(C) \\
(\text{accuracy} - \# \text{parameters})
\]
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:

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

(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.

Before split
**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

**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

Datasets:
- KDD-Cup 2000: 65 variables
- Anonymous MSWeb: 294 variables
- EachMovie (subset): 500 variables

Algorithms:
- WinMine toolkit + Gibbs sampling
- AC + Exact inference

Queries:
Generated queries from the test data with varying numbers of evidence and query variables

Learning time

Inference time

Accuracy: EachMovie
Our method learns complex, accurate models

Treewidth of learned models:

<table>
<thead>
<tr>
<th></th>
<th>AC-Greedy</th>
<th>AC-Quick</th>
<th>WinMine</th>
</tr>
</thead>
<tbody>
<tr>
<td>EachMovie</td>
<td>35</td>
<td>54</td>
<td>281</td>
</tr>
<tr>
<td>KDD Cup</td>
<td>38</td>
<td>38</td>
<td>53</td>
</tr>
<tr>
<td>MSWeb</td>
<td>114</td>
<td>127</td>
<td>118</td>
</tr>
</tbody>
</table>

\[2^{35} = 34 \text{ billion}\]

Log-likelihood of test data:

<table>
<thead>
<tr>
<th></th>
<th>AC-Greedy</th>
<th>AC-Quick</th>
<th>WinMine</th>
</tr>
</thead>
<tbody>
<tr>
<td>EachMovie</td>
<td>−55.7</td>
<td>−54.9</td>
<td>−53.7</td>
</tr>
<tr>
<td>KDD Cup</td>
<td>−2.16</td>
<td>−2.16</td>
<td>−2.16</td>
</tr>
<tr>
<td>MSWeb</td>
<td>−9.85</td>
<td>−9.85</td>
<td>−9.69</td>
</tr>
</tbody>
</table>

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