Machine Learning - Lecture 14

Exact Inference & Learning Bayes Nets

29.06.2010

Bastian Leibe
RWTH Aachen
http://www.mmp.rwth-aachen.de/
leibe@umic.rwth-aachen.de
Course Outline

- **Fundamentals (2 weeks)**
  - Bayes Decision Theory
  - Probability Density Estimation

- **Discriminative Approaches (4 weeks)**
  - Lin. Discriminants, SVMs, Boosting

- **Generative Models (5 weeks)**
  - Bayesian Networks + Applications
  - Markov Random Fields + Applications
  - Exact Inference
  - Approximate Inference

- **Unifying Perspective (1 week)**
Topics of This Lecture

• **Recap: Exact inference**
  - Factor Graphs
  - Sum-Product Belief Propagation
  - Junction Tree algorithm

• **Max-Sum Algorithm for finding most probable value**
  - Key ideas & Derivation
  - Example

• **Algorithms for loopy graphs**
  - Junction Tree algorithm
  - Loopy Belief Propagation

• **Learning Bayesian Networks**
  - Learning with known structure, full observability
  - Learning with known structure, partial observability
  - Structure learning
Recap: Factor Graphs

- **Joint probability**
  - Can be expressed as *product of factors*: \( p(x) = \frac{1}{Z} \prod_s f_s(x_s) \)
  - Factor graphs make this explicit through separate factor nodes.

- **Converting a directed polytree**
  - Conversion to undirected tree creates loops due to moralization!
  - Conversion to a factor graph again results in a tree!
Recap: Sum-Product Algorithm

- **Objectives**
  - Efficient, exact inference algorithm for finding marginals.

- **Procedure:**
  - Pick an arbitrary node as root.
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.

\[
p(x) \propto \prod_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x)
\]

- **Computational effort**
  - Total number of messages = 2 \cdot number of graph edges.
Recap: Sum-Product Algorithm

- Two kinds of messages
  - Message from factor node to variable nodes:
    - **Sum** of factor contributions
      \[
      \mu_{f \rightarrow x}(x) \equiv \sum_{X_s} F_s(x, X_s)
      \]
      \[
      = \sum_{X_s} f_s(x_s) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f_s}(x_m)
      \]
  - Message from variable node to factor node:
    - **Product** of incoming messages
      \[
      \mu_{x_m \rightarrow f_s}(x_m) \equiv \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m)
      \]

⇒ Simple propagation scheme.
Recap: Sum-Product from Leaves to Root

Message definitions:

\[
\mu_{f_s \rightarrow x}(x) \equiv \sum_{x_s} f_s(x_s) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f_s}(x_m)
\]

\[
\mu_{x_m \rightarrow f_s}(x_m) \equiv \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m)
\]

\[
\mu_{x \rightarrow f}(x) = 1 \\
\mu_{f \rightarrow x}(x) = f(x)
\]

Image source: C. Bishop, 2006
Recap: Sum-Product from Root to Leaves

Message definitions:

$$\mu_{f_s \rightarrow x}(x) \equiv \sum_{X_s} f_s(x_s) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f_s}(x_m)$$

$$\mu_{x_m \rightarrow f_s}(x_m) \equiv \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m)$$

$$\mu_x \rightarrow f(x) = 1$$  \hspace{1cm}  $$\mu_f \rightarrow x(x) = f(x)$$

Image source: C. Bishop, 2006
Topics of This Lecture

- Recap: Exact inference
  - Factor Graphs
  - Sum-Product Belief Propagation
  - Junction Tree algorithm

- Max-Sum Algorithm for finding most probable value
  - Key ideas & Derivation
  - Example

- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

- Learning Bayesian Networks
  - Learning with known structure, full observability
  - Learning with known structure, partial observability
  - Structure learning
Max-Sum Algorithm

- **Objective:** an efficient algorithm for finding
  - Value $x^{\text{max}}$ that maximises $p(x)$;
  - Value of $p(x^{\text{max}})$.
  ⇒ Application of dynamic programming in graphical models.

- **In general, maximum marginals ≠ joint maximum.**
  - Example:
    
    |     | $x = 0$ | $x = 1$ |
    |-----|--------|--------|
    | $y = 0$ | 0.3    | 0.4    |
    | $y = 1$ | 0.3    | 0.0    |

  $$\arg\max_x p(x, y) = 1 \quad \arg\max_x p(x) = 0$$
Max-Sum Algorithm - Key Ideas

• Key idea 1: Distributive Law (again)

\[ \max(ab, ac) = a \max(b, c) \]
\[ \max(a + b, a + c) = a + \max(b, c) \]

⇒ Exchange products/summations and max operations exploiting the tree structure of the factor graph.

• Key idea 2: Max-Product → Max-Sum

➢ We are interested in the maximum value of the joint distribution

\[ p(x^{\text{max}}) = \max_x p(x) \]

⇒ Maximize the product \( p(x) \).

➢ For numerical reasons, use the logarithm.

\[ \ln \left( \max_x p(x) \right) = \max_x \ln p(x). \]

⇒ Maximize the sum (of log-probabilities).
Max-Sum Algorithm

- Maximizing over a chain (max-product)

\[
    p(x^{\text{max}}) = \max_x p(x) = \max_{x_1} \cdots \max_{x_M} p(x)
\]

\[
    = \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} \left[ \psi_{1,2}(x_1, x_2) \cdots \psi_{N-1,N}(x_{N-1}, x_N) \right]
\]

\[
    = \frac{1}{Z} \max_{x_1} \left[ \max_{x_2} \left[ \psi_{1,2}(x_1, x_2) \left[ \cdots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right] \right]
\]

- Exchange max and product operators

- Generalizes to tree-structured factor graph

\[
    \max p(x) = \max_{x_n} \prod_{f_s \in \text{ne}(x_n)} \max_{X_s} f_s(x_n, X_s)
\]

Slide adapted from Chris Bishop

Image source: C. Bishop, 2006
Max-Sum Algorithm

- Initialization (leaf nodes)
  \[
  \mu_{x \to f}(x) = 0 \quad \mu_{f \to x}(x) = \ln f(x)
  \]

- Recursion
  - **Messages**
    \[
    \mu_{f \to x}(x) = \max_{x_1, \ldots, x_M} \left[ \ln f(x, x_1, \ldots, x_M) + \sum_{m \in \text{ne}(f_x) \setminus x} \mu_{x_m \to f}(x_m) \right]
    \]
    \[
    \mu_{x \to f}(x) = \sum_{l \in \text{ne}(x) \setminus f} \mu_{f_l \to x}(x)
    \]
  - For each node, keep a record of which values of the variables gave rise to the maximum state:
    \[
    \phi(x) = \arg \max_{x_1, \ldots, x_M} \left[ \ln f(x, x_1, \ldots, x_M) + \sum_{m \in \text{ne}(f_x) \setminus x} \mu_{x_m \to f}(x_m) \right]
    \]

Slide adapted from Chris Bishop
Max-Sum Algorithm

- Termination (root node)
  - Score of maximal configuration
    \[ p_{\text{max}} = \max_x \left[ \sum_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x) \right] \]
  - Value of root node variable giving rise to that maximum
    \[ x_{\text{max}} = \arg \max_x \left[ \sum_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x) \right] \]
  - Back-track to get the remaining variable values
    \[ x_{n-1}^{\text{max}} = \phi(x_n^{\text{max}}) \]
Visualization of the Back-Tracking Procedure

• Example: Markov chain

⇒⇒⇒⇒

Same idea as in Viterbi algorithm for HMMs...

Image source: C. Bishop, 2006
Topics of This Lecture

- Recap: Exact inference
  - Factor Graphs
  - Sum-Product Belief Propagation
  - Junction Tree algorithm
- Max-Sum Algorithm for finding most probable value
  - Key ideas & Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation
- Learning Bayesian Networks
  - Learning with known structure, full observability
  - Learning with known structure, partial observability
  - Structure learning
Junction Tree Algorithm

• Motivation
  - **Exact** inference on general graphs.
  - Works by turning the initial graph into a junction tree and then running a sum-product-like algorithm.
  - **Intractable** on graphs with large cliques.

• Main steps
  1. If starting from directed graph, first convert it to an undirected graph by **moralization**.
  2. Introduce additional links by **triangulation** in order to reduce the size of cycles.
  3. **Find cliques** of the moralized, triangulated graph.
  4. Construct a new graph from the **maximal cliques**.
  5. Remove minimal links to **break cycles** and get a junction tree.

⇒ Apply regular **message passing** to perform inference.
Junction Tree Algorithm

- Starting from an undirected graph...
Junction Tree Algorithm

1. Convert to an undirected graph through moralization.
   - Marry the parents of each node.
   - Remove edge directions.
2. Triangulate

- Such that there is no loop of length > 3 without a chord.
- This is necessary so that the final junction tree satisfies the “running intersection” property (explained later).
3. Find cliques of the moralized, triangulated graph.
4. Construct a new junction graph from maximal cliques.
   - Create a node from each clique.
   - Each link carries a list of all variables in the intersection.
     - Drawn in a “separator” box.
5. Remove links to break cycles ⇒ junction tree.
   - For each cycle, remove the link(s) with the minimal number of shared nodes until all cycles are broken.
   - Result is a maximal spanning tree, the junction tree.
• **Running intersection property**
  
  - *“If a variable appears in more than one clique, it also appears in all intermediate cliques in the tree”*.  
  - This ensures that neighboring cliques have consistent probability distributions.  
  - Local consistency → global consistency
Junction Tree: Example 1

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
Junction Tree: Example 1

(b) Moral graph

(c) Junction graph

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
  3. Find cliques
  4. Construct junction graph

Image source: J. Pearl, 1988
Junction Tree: Example 1

- **Algorithm**
  1. Moralization
  2. Triangulation (not necessary here)
  3. Find cliques
  4. Construct junction graph
  5. Break links to get junction tree

B. Leibe

Image source: J. Pearl, 1988
• Without triangulation step
  - The final graph will contain cycles that we cannot break without losing the running intersection property!

Image source: J. Pearl, 1988
Junction Tree: Example 2

- When applying the triangulation
  - Only small cycles remain that are easy to break.
  - Running intersection property is maintained.
Junction Tree Algorithm

- **Good news**
  - The junction tree algorithm is efficient in the sense that for a given graph there does not exist a computationally cheaper approach.

- **Bad news**
  - This may still be too costly.
  - Effort determined by number of variables in the largest clique.
  - Grows exponentially with this number (for discrete variables).
  \[\Rightarrow\] Algorithm becomes impractical if the graph contains large cliques!
Loopy Belief Propagation

- Alternative algorithm for loopy graphs
  - Sum-Product on general graphs.
  - Strategy: *simply ignore the problem.*
  - Initial unit messages passed across all links, after which messages are passed around until convergence
    - Convergence is not guaranteed!
    - Typically break off after fixed number of iterations.
  - *Approximate* but *tractable* for large graphs.
  - Sometime works well, sometimes not at all.
Topics of This Lecture

- Recap: Exact inference
  - Factor Graphs
  - Sum-Product Belief Propagation
  - Junction Tree algorithm
- Max-Sum Algorithm for finding most probable value
  - Key ideas & Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

- Learning Bayesian Networks
  - Learning with known structure, full observability
  - Learning with known structure, partial observability
  - Structure learning
Bayesian Networks

• What we’ve learned so far...
  ➢ We know they are directed graphical models.
  ➢ Their joint probability factorizes into conditional probabilities,

\[ p(x) = \prod_{k=1}^{K} p(x_k | pa_k) \]

  ➢ We know how to convert them into undirected graphs.
  ➢ We know how to perform inference for them.
    - Sum/Max-Product BP for exact inference in (poly)tree-shaped BNs.
    - Loopy BP for approximate inference in arbitrary BNs.
    - Junction Tree algorithm for converting arbitrary BNs into trees.

• But what are they actually good for?
  ➢ How do we apply them in practice?
  ➢ And how do we learn their parameters?
Parameter Learning in Bayesian Networks

- We need to specify two things:
  - **Structure** of Bayesian network (graph topology)
  - Parameters of each *conditional probability table* (CPT)

- It is possible to learn both from training data.
  - But learning structure is much harder than learning parameters.
  - Also, learning when some nodes are hidden is much harder than when everything is observable.

- Four cases:

<table>
<thead>
<tr>
<th>Structure</th>
<th>Observability</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known</td>
<td>Full</td>
<td>Maximum Likelihood Estimation</td>
</tr>
<tr>
<td>Known</td>
<td>Partial</td>
<td>EM (or gradient ascent)</td>
</tr>
<tr>
<td>Unknown</td>
<td>Full</td>
<td>Search through model space</td>
</tr>
<tr>
<td>Unknown</td>
<td>Partial</td>
<td>EM + search through model space</td>
</tr>
</tbody>
</table>
Learning Parameters

• Example:

\[ p(x) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2) \]

- Assume each variable \( x_i \) is discrete and can take \( K_i \) values.
- The parameters of this model can be represented with 4 tables (called conditional probability tables - CPT):
  - \( p(x_1 = k) = \theta_{1,k} \) \( \theta_1 \) has \( K_1 \) entries.
  - \( p(x_2 = k' | x_1 = k) = \theta_{2,k,k'} \) \( \theta_2 \) has \( K_1 \times K_2 \) entries.
  - \( p(x_3 = k' | x_1 = k) = \theta_{3,k,k'} \)
  - \( p(x_4 = k' | x_2 = k) = \theta_{4,k,k'} \)

- Note that \( \sum_{k'} \theta_{i,k,k'} = 1 \)
Case 1: Known Structure, Full Observability

- Assume a training data set: \( D = \{ x^{(n)} \}_{n=1}^{N} \)
  - How do we learn \( \theta \) from \( D \)?

- Maximum Likelihood:
  \[
p(x^{(n)}|\theta) = p(x_1^{(n)}|\theta_1)p(x_2^{(n)}|x_1^{(n)}, \theta_2)p(x_3^{(n)}|x_1^{(n)}, \theta_3)p(x_4^{(n)}|x_2^{(n)}, \theta_4)
  \]
  \[
p(D|\theta) = \prod_{n=1}^{N} p(x^{(n)}|\theta) = \prod_{n=1}^{N} \prod_{i=1}^{4} p(x_i^{(n)}|x_{pa(i)}^{(n)}, \theta_i)
  \]

- Maximum Log-Likelihood:
  \[
  \ln p(D|\theta) = \sum_{n=1}^{N} \sum_{i=1}^{4} \ln p(x_i^{(n)}|x_{pa(i)}^{(n)}, \theta_i)
  \]

Slide credit: Zoubin Ghahramani
Case 1: Known Structure, Full Observability

- **Maximum Log-Likelihood:**

  \[
  \ln p(D|\theta) = \sum_{n=1}^{N} \sum_{i=1}^{4} \ln p(x^{(n)}_{i}|x^{(n)}_{pa(i)}, \theta_{i})
  \]

  - This decomposes into a sum of functions \( \theta_{i} \).
  - Each \( \theta_{i} \) can be optimized separately:

    \[
    \theta_{i,k,k'} = \frac{n_{i,k,k'}}{\sum_{k''} n_{i,k,k''}}
    \]

    where \( n_{i,k,k'} \) is the number of times in \( D \) that \( x_{i} = k' \) and \( x_{pa(i)} = k \).

- **ML solution**

  \[\Rightarrow\text{Simply calculate frequencies!}\]

<table>
<thead>
<tr>
<th>( x_{1} )</th>
<th>( x_{2} )</th>
<th>( n_{2} )</th>
<th>( \theta_{2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
<td>0.3</td>
</tr>
</tbody>
</table>

  \[\Rightarrow x_{1} \]

<table>
<thead>
<tr>
<th>( x_{2} )</th>
<th>0.6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{1} )</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Slide credit: Zoubin Ghahramani

B. Leibe
Case 2: Known Structure, Hidden Variables

- **ML learning with hidden variables**
  - Assume a model parameterized by $\theta$ with observed variables $X$ and hidden (latent) variables $Z$.

- **Goal**
  - **Maximize parameter log-likelihood given the observed data**
    \[
    L(\theta) = \ln p(X|\theta) = \ln \sum_Z p(X, Z|\theta)
    \]

- **EM Algorithm**: Iterate between two steps:
  - **E-step**: fill-in hidden / missing variables
  - **M-step**: apply complete-data learning to filled-in data.
Learning with Hidden Variables: EM Algorithm

- **Goal:**
  - Maximize parameter log-likelihood given the observed data.
  
  \[ L(\theta) = \ln p(X|\theta) = \ln \sum_{Z} p(X, Z|\theta) \]

- **EM Algorithm: Derivation**
  - We do not know the values of the latent variables in \( Z \), but we can express their posterior distribution given \( X \) and (an initial guess for) \( \theta \).

  \[ \Rightarrow \textbf{E-step: Evaluate } p(Z|X, \theta^{\text{old}}) \]

  - Since we cannot use the complete-data log-likelihood directly, we maximize its expected value under the posterior distribution of \( Z \).

  \[ \Rightarrow \textbf{M-step: Maximize } \theta^{\text{new}} = \arg \max_{\theta} \sum_{Z} p(Z|X, \theta^{\text{old}}) \ln p(X, Z|\theta) \]
Learning with Hidden Variables: EM Algorithm

- **Note on the E-step:**
  - The E-step requires solving the **inference** problem.
  - I.e. finding the distribution over the hidden variables $p(Z|X, \theta^{old})$ given the current model parameters.
  - This can be done using **belief propagation** or the **junction tree** algorithm.

  ⇒ As inference becomes a subroutine of the learning procedure, fast inference algorithms are crucial!
Example Application

- **Mixture-of-Gaussian Fitting with EM**
  - Standard application of EM.
  - Corresponding Bayesian network:

- **Important point here**
  - Bayesian networks can be treacherous!
  - They hide the true complexity in a very simple-looking diagram.
  - E.g. the diagram here only encodes the information that we have a latent variable $\theta$ which depends on observed variables $x_i$
    - The information that $p(x_i | \theta)$ is represented by a mixture-of-Gaussians needs to be communicated additionally!
    - On the other hand, this general framework can also be used to apply EM for other types of distributions or latent variables.
Summary: Learning with Known Structure

- **ML-Learning with complete data (no hidden variables)**
  - Log-likelihood decomposes into sum of functions of $\theta_i$.
  - Each $\theta_i$ can be optimized separately.
  - ML-solution: simply calculate frequencies.

- **ML-Learning with incomplete data (hidden variables)**
  - Iterative EM algorithm.
  - E-step: compute expected counts given previous settings $\theta^{(t)}$ of parameters $E[n_{i,j,k} \mid D, \theta^{(t)}]$.
  - M-step: re-estimate parameters $\theta$ using the expected counts.
    \[
    \theta^{(t+1)}_{i,j,k} \leftarrow \frac{E[n_{i,j,k} \mid D, \theta^{(t)}]}{\sum_{k'} E[n_{i,j,k'} \mid D, \theta^{(t)}]}
    \]
Cases 3+4: Unknown Structure

- **Goal**
  - Learn a directed acyclic graph (DAG) that best explains the data.

- **Constraints-based learning**
  - Use statistical tests of marginal and conditional independence.
  - Find the set of DAGs whose d-separation relations match the results of conditional independence tests.

- **Score-based learning**
  - Use a global score such as BIC (Bayes Information Criterion).
  - Find a structure that maximizes this score.
Cases 3+4: Unknown Structure

- Extremely hard problem
  - NP-hard
  - Number of DAGs on $N$ variables is super-exponential in $N$.
    - 4 nodes: 543 DAGs
    - 10 nodes: $O(10^{18})$ DAGs.
  $\Rightarrow$ Need to use heuristics to prune down the search space and use efficient methods to evaluate hypotheses.

- Additional problem: often not enough data available.
  - Need to make decisions about statistical conditional independence.
  - Typically only feasible if the structure is relatively simple and a lot of data is available...
Example Application

- Analyzing gene expression from micro-array data
  - 1000s of measurement spots (probes) on micro-array, each sensitive to a specific DNA marker (e.g., a section of a gene).
  - The probes measure if the corresponding gene is expressed (=active).
  - Collect samples from patients with a certain disease or condition.
  - Monitor 1000s of genes simultaneously.

- Interesting questions
  - Is there a statistical relationship between certain gene expressions?
  - If so, can we derive the structure by which they influence each other?
References and Further Reading

- A thorough introduction to Graphical Models in general and Bayesian Networks in particular can be found in Chapter 8 of Bishop’s book.

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006