Structured Models:
Hidden Markov Models versus Conditional Random Fields

Eric Xing
Lecture 11, October 13, 2014

Reading:
© Eric Xing @ CMU, 2014
From static to dynamic mixture models

Static mixture

Dynamic mixture

The underlying source:
Speech signal, dice,

The sequence:
Phonemes, sequence of rolls,

© Eric Xing @ CMU, 2014
Hidden Markov Model

- **Observation space**
  - Alphabetic set: \( C = \{c_1, c_2, \ldots, c_K\} \)
  - Euclidean space: \( \mathbb{R}^d \)
- **Index set of hidden states**
  \( I = \{1, 2, \ldots, M\} \)
- **Transition probabilities between any two states**
  \[
p(y_t^j = 1 | y_{t-1}^i = 1) = a_{i,j},
\]
  or
  \[
p(y_t | y_{t-1}^i = 1) \sim \text{Multinomial}(a_{i,1}, a_{i,2}, \ldots, a_{i,M}), \forall i \in I.
\]
- **Start probabilities**
  \[
p(y_1) \sim \text{Multinomial}(\pi_1, \pi_2, \ldots, \pi_M).
\]
- **Emission probabilities associated with each state**
  \[
p(x_t | y_t^i = 1) \sim \text{Multinomial}(b_{i,1}, b_{i,2}, \ldots, b_{i,K}), \forall i \in I.
\]
  or in general:
  \[
p(x_t | y_t^i = 1) \sim f(\cdot | \theta_i), \forall i \in I.
\]
Applications of HMMs

- Some early applications of HMMs
  - finance, but we never saw them
  - speech recognition
  - modelling ion channels

- In the mid-late 1980s HMMs entered genetics and molecular biology, and they are now firmly entrenched.

- Some current applications of HMMs to biology
  - mapping chromosomes
  - aligning biological sequences
  - predicting sequence structure
  - inferring evolutionary relationships
  - finding genes in DNA sequence
A Bio Application: gene finding
GENSCAN (Burge & Karlin)

Transition probabilities: \( p(y_i^j = 1 | y_{i-1}^j = 1) = a_{i,j} \)

\[
p(\bullet | y) = \begin{pmatrix}
\theta_1 \\
\theta_2 \\
\theta_3 \\
\theta_4 \\
\end{pmatrix}
\]

\(~\)
A “Financial” Application: The Dishonest Casino

A casino has two dice:
- Fair die
  \[ P(1) = P(2) = P(3) = P(5) = P(6) = \frac{1}{6} \]
- Loaded die
  \[ P(1) = P(2) = P(3) = P(5) = \frac{1}{10} \]
  \[ P(6) = \frac{1}{2} \]

Casino player switches back-&-forth between fair and loaded die once every 20 turns

Game:
1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2
The Dishonest Casino Model

FAIR

P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

LOADED

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2

© Eric Xing @ CMU, 2014
Puzzles Regarding the Dishonest Casino

**GIVEN:** A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

**QUESTION**

- How likely is this sequence, given our model of how the casino works?
  - This is the **EVALUATION** problem in HMMs

- What portion of the sequence was generated with the fair die, and what portion with the loaded die?
  - This is the **DECODING** question in HMMs

- How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?
  - This is the **LEARNING** question in HMMs
Joint Probability

1245526462146146136136661664661636616366163616515615115146123562344
**Probability of a Parse**

- Given a sequence \( x = x_1 \ldots x_T \)
- and a parse \( y = y_1, \ldots, y_T \),
- To find how likely is the parse:
  (given our HMM and the sequence)

\[
p(x, y) = p(x_1 \ldots x_T, y_1, \ldots, y_T) \quad \text{(Joint probability)}
\]

\[
= p(y_1) p(x_1 | y_1) p(y_2 | y_1) p(x_2 | y_2) \ldots p(y_T | y_{T-1}) p(x_T | y_T)
\]

\[
= p(y_1) P(y_2 | y_1) \ldots p(y_T | y_{T-1}) \times p(x_1 | y_1) p(x_2 | y_2) \ldots p(x_T | y_T)
\]

- Marginal probability:
  \[
p(x) = \sum_y p(x, y) = \sum_{y_1} \sum_{y_2} \ldots \sum_{y_N} \pi_{y_1} \prod_{t=2}^{T} a_{y_{t-1}, y_t} \prod_{t=1}^{T} p(x_t | y_t)
\]

- Posterior probability:
  \[
p(y | x) = \frac{p(x, y)}{p(x)}
\]
Example: the Dishonest Casino

- Let the sequence of rolls be:
  - $x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4$

- Then, what is the likelihood of
  - $y = \text{Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair}$?
  (say initial probs $a_{0\text{Fair}} = \frac{1}{2}$, $a_{0\text{Loaded}} = \frac{1}{2}$)

\[
\frac{1}{2} \times P(1 \mid \text{Fair}) \cdot P(\text{Fair} \mid \text{Fair}) \cdot P(2 \mid \text{Fair}) \cdot P(\text{Fair} \mid \text{Fair}) \cdots P(4 \mid \text{Fair}) = \\
\frac{1}{2} \times \left(\frac{1}{6}\right)^{10} \times (0.95)^9 = 0.000000000521158647211 = 5.21 \times 10^{-9}
\]
Example: the Dishonest Casino

- So, the likelihood the die is fair in all this run is just $5.21 \times 10^{-9}$

- OK, but what is the likelihood of
  - $\pi = \text{Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded}$?

$$\frac{1}{2} \times P(1 \mid \text{Loaded}) P(\text{Loaded} \mid \text{Loaded}) \ldots P(4 \mid \text{Loaded}) =$$

$$\frac{1}{2} \times \left(\frac{1}{10}\right)^8 \times \left(\frac{1}{2}\right)^2 \left(0.95\right)^9 = .00000000078781176215 = 0.79 \times 10^{-9}$$

- Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way
Example: the Dishonest Casino

Let the sequence of rolls be:
- \( x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6 \)

Now, what is the likelihood \( \pi = F, F, \ldots, F \)?
- \( \frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = 0.5 \times 10^{-9}, \text{ same as before} \)

What is the likelihood \( y = L, L, \ldots, L \)?
- \( \frac{1}{2} \times (1/10)^4 \times (1/2)^6 \times (0.95)^9 = 5 \times 10^{-7} \)

So, it is 100 times more likely the die is loaded
Three Main Questions on HMMs

1. Evaluation

GIVEN an HMM $M$, and a sequence $x$,
FIND $\text{Prob}(x | M)$
ALGO. Forward

2. Decoding

GIVEN an HMM $M$, and a sequence $x$,
FIND the sequence $y$ of states that maximizes, e.g., $P(y | x, M)$, or the most probable subsequence of states
ALGO. Viterbi, Forward-backward

3. Learning

GIVEN an HMM $M$, with unspecified transition/emission probs., and a sequence $x$,
FIND parameters $\theta = (\pi_i, a_{ij}, \eta_{ik})$ that maximize $P(x | \theta)$
ALGO. Baum-Welch (EM)
The Forward Algorithm

- We want to calculate \( P(x) \), the likelihood of \( x \), given the HMM
  - Sum over all possible ways of generating \( x \):
    \[
p(x) = \sum_y p(x, y) = \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} \pi_{y_1} \prod_{t=2}^T a_{y_{t-1}, y_t} \prod_{t=1}^T p(x_t \mid y_t)
    \]
  - To avoid summing over an exponential number of paths \( y \), define
    \[
    \alpha(y_t^k = 1) = \alpha_t^k \overset{\text{def}}{=} P(x_1, \ldots, x_t, y_t^k = 1)
    \]
    (the forward probability)
  - The recursion:
    \[
    \alpha_t^k = p(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}
    \]
    \[
    P(x) = \sum_k \alpha_T^k
    \]
The Forward Algorithm – derivation

- Compute the forward probability:

\[ \alpha_t^k = P(x_1, \ldots, x_{t-1}, x_t, y_t^k = 1) \]

\[ = \sum_{y_{t-1}} P(x_1, \ldots, x_{t-1}, y_{t-1}) P(y_t^k = 1 \mid y_{t-1}, x_1, \ldots, x_{t-1}) P(x_t \mid y_t^k = 1, x_1, \ldots, x_{t-1}, y_{t-1}) \]

\[ = \sum_{y_{t-1}} P(x_1, \ldots, x_{t-1}, y_{t-1}) P(y_t^k = 1 \mid y_{t-1}) P(x_t \mid y_t^k = 1) \]

\[ = P(x_t \mid y_t^k = 1) \sum_i P(x_1, \ldots, x_{t-1}, y_{t-1}^i = 1) P(y_t^k = 1 \mid y_{t-1}^i = 1) \]

\[ = P(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k} \]

Chain rule: \[ P(A, B, C) = P(A)P(B \mid A)P(C \mid A, B) \]
The Forward Algorithm

- We can compute $\alpha_t^k$ for all $k$, $t$, using dynamic programming!

**Initialization:**

$$\alpha_1^k = P(x_1 \mid y_1^k = 1) \pi_k$$

$$\alpha_1^k = P(x_1, y_1^k = 1) = P(x_1 \mid y_1^k = 1)P(y_1^k = 1) = P(x_1 \mid y_1^k = 1)\pi_k$$

**Iteration:**

$$\alpha_t^k = P(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}$$

**Termination:**

$$P(x) = \sum_k \alpha_T^k$$
The Backward Algorithm

- We want to compute $P(y_t^k = 1 \mid x)$, the posterior probability distribution on the $t^{th}$ position, given $x$

- We start by computing

$$
P(y_t^k = 1, x) = P(x_1, \ldots, x_t, y_t^k = 1, x_{t+1}, \ldots, x_T)
= P(x_1, \ldots, x_t, y_t^k = 1)P(x_{t+1}, \ldots, x_T \mid x_1, \ldots, x_t, y_t^k = 1)
= P(x_1 \ldots x_t, y_t^k = 1)P(x_{t+1} \ldots x_T \mid y_t^k = 1)
$$

- The recursion:

$$
\beta_t^k = \sum_i a_{k,i} p(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i
$$

Forward, $\alpha_t^k$  
Backward, $\beta_t^k = P(x_{t+1}, \ldots, x_T \mid y_t^k = 1)$
The Backward Algorithm – derivation

- Define the backward probability:

\[
\beta_t^k = P(x_{t+1}, \ldots, x_T \mid y_t^k = 1) \\
= \sum_{y_{t+1}} P(x_{t+1}, \ldots, x_T, y_{t+1} \mid y_t^k = 1) \\
= \sum_i P(y_{t+1}^i = 1 \mid y_t^k = 1) p(x_{t+1} \mid y_{t+1}^i = 1, y_t^k = 1) P(x_{t+2}, \ldots, x_T \mid x_{t+1}, y_{t+1}^i = 1, y_t^k = 1) \\
= \sum_i P(y_{t+1}^i = 1 \mid y_t^k = 1) p(x_{t+1} \mid y_{t+1}^i = 1) P(x_{t+2}, \ldots, x_T \mid y_{t+1}^i = 1) \\
= \sum_i a_{k,i} p(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i 
\]

Chain rule: \( P(A, B, C \mid \alpha) = P(A \mid \alpha) P(B \mid A, \alpha) P(C \mid A, B, \alpha) \)
The Backward Algorithm

- We can compute $\beta_t^k$ for all $k$, $t$, using dynamic programming!

**Initialization:**

$$\beta_T^k = 1, \ \forall k$$

**Iteration:**

$$\beta_t^k = \sum_i a_{k,i} P(x_{t+1} | y_{t+1}^i = 1) \beta_{t+1}^i$$

**Termination:**

$$P(x) = \sum_k \alpha_1^k \beta_1^k$$
Example:

\[ x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \]

\[
\alpha_t^k = P(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k} \\
\beta_t^k = \sum_i a_{k,i} P(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i
\]
\[ x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \]

<table>
<thead>
<tr>
<th>Alpha (actual)</th>
<th>Beta (actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0833 0.0500</td>
<td>0.0000 0.0000</td>
</tr>
<tr>
<td>0.0136 0.0052</td>
<td>0.0000 0.0000</td>
</tr>
<tr>
<td>0.0022 0.0006</td>
<td>0.0000 0.0000</td>
</tr>
<tr>
<td>0.0004 0.0001</td>
<td>0.0000 0.0000</td>
</tr>
<tr>
<td>0.0001 0.0000</td>
<td>0.0001 0.0001</td>
</tr>
<tr>
<td>0.0000 0.0000</td>
<td>0.0007 0.0006</td>
</tr>
<tr>
<td>0.0000 0.0000</td>
<td>0.0045 0.0055</td>
</tr>
<tr>
<td>0.0000 0.0000</td>
<td>0.0264 0.0112</td>
</tr>
<tr>
<td>0.0000 0.0000</td>
<td>0.1633 0.1033</td>
</tr>
<tr>
<td>0.0000 0.0000</td>
<td>1.0000 1.0000</td>
</tr>
</tbody>
</table>

\[
\alpha_t^k = P(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k} \\
\beta_t^k = \sum_i a_{k,i} P(x_{t+1} \mid y_{t+1}^i = 1) \beta_t^i.
\]
**x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4**

<table>
<thead>
<tr>
<th>Alpha (logs)</th>
<th>Beta (logs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.4849</td>
<td>-16.2439</td>
</tr>
<tr>
<td>-4.2969</td>
<td>-14.4185</td>
</tr>
<tr>
<td>-6.1201</td>
<td>-12.6028</td>
</tr>
<tr>
<td>-7.9499</td>
<td>-10.8042</td>
</tr>
<tr>
<td>-9.7834</td>
<td>-9.0373</td>
</tr>
<tr>
<td>-11.5905</td>
<td>-7.2181</td>
</tr>
<tr>
<td>-13.4110</td>
<td>-5.4135</td>
</tr>
<tr>
<td>-15.2391</td>
<td>-3.6352</td>
</tr>
<tr>
<td>-17.0310</td>
<td>-1.8120</td>
</tr>
<tr>
<td>-18.8430</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\alpha_t^k = P \left( x_t \mid y_t^k = 1 \right) \sum_i \alpha_{t-1}^i a_{i,k}
\]

\[
\beta_t^k = \sum_i a_{k,i} P \left( x_{t+1} \mid y_{t+1}^i = 1 \right) \beta_t^i.
\]
What is the probability of a hidden state prediction?

\[
\begin{align*}
    p(y_5^t | x) &= \frac{a}{p(x)} \left( 1 \right) \quad t = 1. \\
    p(y_5^\sim | x) &= \frac{b}{p(x)} \\
    p(y_6^1 | x) &= \frac{a}{a+b} = \frac{\exp(-18.8207)}{\exp(-18.8207) + \exp(-19.845)} = 0.7415 \\
    p(y_6^1 | x) &= \exp(-18.8207) \\
    p(y_8^1 | x) &= \exp(-19.6445) \\
    p(y_8^1 | x) &= 0.6857.
\end{align*}
\]
Posterior decoding

- We can now calculate
  \[ P(y_t^k = 1 | x) = \frac{P(y_t^k = 1, x)}{P(x)} = \frac{\alpha_t^k \beta_t^k}{P(x)} \]

- Then, we can ask
  - What is the most likely state at position \( t \) of sequence \( x \):
    \[ k_t^* = \arg \max_k P(y_t^k = 1 | x) \]
  - Note that this is an MPA of a single hidden state, what if we want to a MPA of a whole hidden state sequence?
  - Posterior Decoding:
    \[ \{ y_t^{k_t^*} = 1 : t = 1 \ldots T \} \]
  - This is different from MPA of a whole sequence states
  - This can be understood as bit error rate vs. word error rate

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( P(x, y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.35</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.05</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Example:
MPA of \( X \) ?
MPA of \( (X, Y) \) ?
Viterbi decoding

- GIVEN $x = x_1, ..., x_T$, we want to find $y = y_1, ..., y_T$, such that $P(y|x)$ is maximized:

$$y^* = \arg \max_y P(y|x) = \arg \max_{\pi} P(y,x)$$

- Let

$$V_t^k = \max_{\{y_1, ..., y_{t-1}\}} P(x_1, ..., x_{t-1}, y_1, ..., y_{t-1}, x_t, y_t^k = 1)$$

  = Probability of most likely sequence of states ending at state $y_t = k$

- The recursion:

$$V_t^k = p(x_t | y_t^k = 1) \max_i a_{i,k} V_{t-1}^i$$

- Underflows are a significant problem

$$p(x_1, ..., x_t, y_1, ..., y_t) = \pi_{y_1} a_{y_1,y_2} ... a_{y_{t-1},y_t} b_{y_1,x_1} ... b_{y_t,x_t}$$

- These numbers become extremely small – underflow

- Solution: Take the logs of all values:

$$V_t^k = \log p(x_t | y_t^k = 1) + \max_i \left( \log(a_{i,k}) + V_{t-1}^i \right)$$
Computational Complexity and implementation details

- What is the running time, and space required, for Forward, and Backward?

\[
\alpha_t^k = p(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}
\]

\[
\beta_t^k = \sum_i a_{k,i} p(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i
\]

\[
V_t^k = p(x_t \mid y_t^k = 1) \max_i a_{i,k} V_{t-1}^i
\]

Time: \(O(K^2N)\); \(\) Space: \(O(KN)\).

- Useful implementation technique to avoid underflows
  - Viterbi: sum of logs
  - Forward/Backward: rescaling at each position by multiplying by a constant
Learning HMM: two scenarios

- **Supervised learning**: estimation when the “right answer” is known
  - **Examples**:
    - GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands
    - GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

- **Unsupervised learning**: estimation when the “right answer” is unknown
  - **Examples**:
    - GIVEN: the porcupine genome; we don’t know how frequent are the CpG islands there, neither do we know their composition
    - GIVEN: 10,000 rolls of the casino player, but we don’t see when he changes dice

- **QUESTION**: Update the parameters $\theta$ of the model to maximize $P(x|\theta)$ --- Maximal likelihood (ML) estimation
Supervised ML estimation

- Given $x = x_1...x_N$ for which the true state path $y = y_1...y_N$ is known,
  - Define:
    \[ A_{ij} = \# \text{ times state transition } i \rightarrow j \text{ occurs in } y \]
    \[ B_{ik} = \# \text{ times state } i \text{ in } y \text{ emits } k \text{ in } x \]
  - We can show that the maximum likelihood parameters $\theta$ are:
    \[
    a_{ij}^{\text{ML}} = \frac{\#(i \rightarrow j)}{\#(i \rightarrow \bullet)} = \frac{\sum_n \sum_{t=2}^T y_{n,t-1}^i y_{n,t}^j}{\sum_n \sum_{t=2}^T y_{n,t-1}^i} = \frac{A_{ij}}{\sum_j A_{ij}}
    \]
    \[
    b_{ik}^{\text{ML}} = \frac{\#(i \rightarrow k)}{\#(i \rightarrow \bullet)} = \frac{\sum_n \sum_{t=1}^T y_{n,t}^i x_{n,t}^k}{\sum_n \sum_{t=1}^T y_{n,t}^i} = \frac{B_{ik}}{\sum_k B_{ik}}
    \]
    (Homework!)

- What if $y$ is continuous? We can treat \(\{(x_{n,t}, y_{n,t}) : t = 1:T, n = 1:N\}\) as $N \times T$ observations of, e.g., a Gaussian, and apply learning rules for Gaussian ...
    (Homework!)
Pseudocounts

- Solution for small training sets:
  - Add pseudocounts
    \[ A_{ij} = \# \text{ times state transition } i \rightarrow j \text{ occurs in } y + R_{ij} \]
    \[ B_{ik} = \# \text{ times state } i \text{ in } y \text{ emits } k \text{ in } x + S_{ik} \]
  - \( R_{ij}, S_{ij} \) are pseudocounts representing our prior belief
  - Total pseudocounts: \( R_i = \sum_j R_{ij}, \quad S_i = \sum_k S_{ik}, \)
    - "strength" of prior belief,
    - total number of imaginary instances in the prior

- Larger total pseudocounts \( \Rightarrow \) strong prior belief

- Small total pseudocounts: just to avoid 0 probabilities --- smoothing
Unsupervised ML estimation

- Given \( x = x_1 \ldots x_N \) for which the true state path \( y = y_1 \ldots y_N \) is unknown,

  - **EXPECTATION MAXIMIZATION**

0. Starting with our best guess of a model \( M \), parameters \( \theta \):

1. Estimate \( A_{ij} \), \( B_{ik} \) in the training data
   - How? \( A_{ij} = \sum_{n,t} \langle y_{n,t-1}^i y_{n,t}^j \rangle \) \( B_{ik} = \sum_{n,t} \langle y_{n,t}^i \rangle x_{n,t}^k \), How? (homework)

2. Update \( \theta \) according to \( A_{ij} \), \( B_{ik} \)
   - Now a "supervised learning" problem

3. Repeat 1 & 2, until convergence

This is called the Baum-Welch Algorithm

We can get to a provably more (or equally) likely parameter set \( \theta \) each iteration
The Baum Welch algorithm

- The complete log likelihood

\[ \ell_c(\theta; x, y) = \log p(x, y) = \log \prod_n \left( \prod_{t=2}^T p(y_{n, t} | y_{n, t-1}) \prod_{t=1}^T p(x_{n, t} | x_{n, t}) \right) \]

- The expected complete log likelihood

\[ \langle \ell_c(\theta; x, y) \rangle = \sum_n \left( \langle y_{n, 1}^i \rangle p(y_{n, 1} | x_n) \log \pi_i \right) + \sum_n \sum_{t=2}^T \left( \langle y_{n, t-1}^i, y_{n, t}^j \rangle p(y_{n, t-1}, y_{n, t} | x_n) \log a_{i,j} \right) + \sum_n \sum_{t=1}^T \left( x_{n, t}^k \langle y_{n, t}^i \rangle p(y_{n, t} | x_n) \log b_{i,k} \right) \]

- EM

  - The E step

  \[ \gamma_{n,t}^i = \langle y_{n,t}^i \rangle = p(y_{n,t}^i = 1 | x_n) \]

  \[ \xi_{n,t}^{i,j} = \langle y_{n,t-1}^i y_{n,t}^j \rangle = p(y_{n,t-1}^i = 1, y_{n,t}^j = 1 | x_n) \]

  - The M step ("symbolically" identical to MLE)

  \[ \pi_i^{ML} = \frac{\sum_n \gamma_{n,1}^i}{N} \]

  \[ a_{i,j}^{ML} = \frac{\sum_n \sum_{t=2}^T \xi_{n,t}^{i,j}}{\sum_n \sum_{t=1}^{T-1} \gamma_{n,t}^i} \]

  \[ b_{i,k}^{ML} = \frac{\sum_n \sum_{t=1}^T \gamma_{n,t}^i x_{n,t}^k}{\sum_n \sum_{t=1}^{T-1} \gamma_{n,t}^i} \]
Summary

- Modeling hidden transitional trajectories (in discrete state space, such as cluster label, DNA copy number, dice-choice, etc.) underlying observed sequence data (discrete, such as dice outcomes; or continuous, such as CGH signals)
- Useful for parsing, segmenting sequential data
- Important HMM computations:
  - The joint likelihood of a parse and data can be written as a product to local terms (i.e., initial prob, transition prob, emission prob.)
  - Computing marginal likelihood of the observed sequence: forward algorithm
  - Predicting a single hidden state: forward-backward
  - Predicting an entire sequence of hidden states: viterbi
  - Learning HMM parameters: an EM algorithm known as Baum-Welch
Shortcomings of Hidden Markov Model

- HMM models capture dependences between each state and only its corresponding observation
  - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, indentation, amount of white space, etc.

- Mismatch between learning objective function and prediction objective function
  - HMM learns a joint distribution of states and observations $P(Y, X)$, but in a prediction task, we need the conditional probability $P(Y|X)$
Solution:
Maximum Entropy Markov Model (MEMM)

- Models dependence between each state and the full observation sequence explicitly
  - More expressive than HMMs
- Discriminative model
  - Completely ignores modeling P(X): saves modeling effort
  - Learning objective function consistent with predictive function: P(Y|X)

\[
P(y_{1:n} | x_{1:n}) = \prod_{i=1}^{n} P(y_i | y_{i-1}, x_{1:n}) = \prod_{i=1}^{n} \frac{\exp(w^T f(y_i, y_{i-1}, x_{1:n}))}{Z(y_{i-1}, x_{1:n})}
\]
MEMM: the Label bias problem

What the local transition probabilities say:

• State 1 almost always prefers to go to state 2
• State 2 almost always prefer to stay in state 2
MEMM: the Label bias problem

Probability of path 1-> 1-> 1-> 1:

- $0.4 \times 0.45 \times 0.5 = 0.09$
MEMM: the Label bias problem

Probability of path 2->2->2->2 :
- 0.2 X 0.3 X 0.3 = 0.018

Other paths:
- 1-> 1-> 1-> 1: 0.09
MEMM: the Label bias problem

Observation 1  Observation 2  Observation 3  Observation 4

State 1
- 0.4
- 0.2
- 0.6

State 2
- 0.2
- 0.2
- 0.6

State 3
- 0.2
- 0.1
- 0.3

State 4
- 0.2
- 0.2
- 0.3

State 5
- 0.2
- 0.3
- 0.2

Probability of path 1→2→1→2:
- 0.6 X 0.2 X 0.5 = 0.06

Other paths:
- 1→1→1→1: 0.09
- 2→2→2→2: 0.018
MEMM: the Label bias problem

Probability of path 1->1->2->2:
• 0.4 X 0.55 X 0.3 = 0.066

Other paths:
1->1->1->1: 0.09
2->2->2->2: 0.018
1->2->1->2: 0.06
MEMM: the Label bias problem

Most Likely Path: 1-> 1-> 1-> 1

• Although **locally** it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.

• **why?**
MEMM: the Label bias problem

Most Likely Path: 1-> 1-> 1-> 1

- State 1 has only two transitions but state 2 has 5:
  - Average transition probability from state 2 is lower
MEMM: the Label bias problem

Label bias problem in MEMM:
• Preference of states with lower number of transitions over others
Solution:
Do not normalize probabilities locally

From local probabilities …. 
Solution:
Do not normalize probabilities locally

From local probabilities to local potentials

- States with lower transitions do not have an unfair advantage!
From MEMM 

\[
P(y_{1:n} | x_{1:n}) = \prod_{i=1}^{n} P(y_i | y_{i-1}, x_{1:n}) = \prod_{i=1}^{n} \frac{\exp(w^T f(y_i, y_{i-1}, x_{1:n}))}{Z(y_{i-1}, x_{1:n})}
\]
CRF is a partially directed model

- Discriminative model like MEMM
- Usage of global normalizer $Z(x)$ overcomes the label bias problem of MEMM
- Models the dependence between each state and the entire observation sequence (like MEMM)
Conditional Random Fields

- General parametric form:

\[
P(y|x) = \frac{1}{Z(x, \lambda, \mu)} \exp\left(\sum_{i=1}^{n} \left( \sum_{k} \lambda_{k} f_{k}(y_{i}, y_{i-1}, x) + \sum_{l} \mu_{l} g_{l}(y_{i}, x) \right) \right)
\]

\[
= \frac{1}{Z(x, \lambda, \mu)} \exp\left(\sum_{i=1}^{n} \left( \lambda^{T} f(y_{i}, y_{i-1}, x) + \mu^{T} g(y_{i}, x) \right) \right)
\]

where \(Z(x, \lambda, \mu) = \sum_{y} \exp\left(\sum_{i=1}^{n} \left( \lambda^{T} f(y_{i}, y_{i-1}, x) + \mu^{T} g(y_{i}, x) \right) \right)\)
CRFs: Inference

- Given CRF parameters $\lambda$ and $\mu$, find the $y^*$ that maximizes $P(y|x)$

$$y^* = \arg \max_y \exp \left( \sum_{i=1}^{n} (\lambda^T f(y_i, y_{i-1}, x) + \mu^T g(y_i, x)) \right)$$

- Can ignore $Z(x)$ because it is not a function of $y$

- Run the max-product algorithm on the junction-tree of CRF:

```
Y_1
  ↙
Y_2
  ↙
...  ...  ...
  ↙
Y_{n-2}   Y_{n-1}
  ↙
Y_{n-1}, Y_n
```

Same as Viterbi decoding used in HMMs!
CRF learning

- Given \( \{(x_d, y_d)\}_{d=1}^N \), find \( \lambda^*, \mu^* \) such that

\[
\begin{align*}
\lambda^*, \mu^* &= \arg\max_{\lambda, \mu} L(\lambda, \mu) = \arg\max_{\lambda, \mu} \prod_{d=1}^N P(y_d|x_d, \lambda, \mu) \\
&= \arg\max_{\lambda, \mu} \prod_{d=1}^N \frac{1}{Z(x_d, \lambda, \mu)} \exp\left(\sum_{i=1}^n (\lambda^T f(y_{d,i}, y_{d,i-1}, x_d) + \mu^T g(y_{d,i}, x_d))\right) \\
&= \arg\max_{\lambda, \mu} \sum_{d=1}^N \left(\sum_{i=1}^n (\lambda^T f(y_{d,i}, y_{d,i-1}, x_d) + \mu^T g(y_{d,i}, x_d)) - \log Z(x_d, \lambda, \mu)\right)
\end{align*}
\]

- Computing the gradient w.r.t \( \lambda \):

\[
\nabla_\lambda L(\lambda, \mu) = \sum_{d=1}^N \left(\sum_{i=1}^n f(y_{d,i}, y_{d,i-1}, x_d) - \sum_{y} P(y|x_d) \sum_{i=1}^n f(y_{d,i}, y_{d,i-1}, x_d)\right)
\]

Gradient of the log-partition function in an exponential family is the expectation of the sufficient statistics.
CRF learning

\[ \nabla_{\lambda} L(\lambda, \mu) = \sum_{d=1}^{N} \left( \sum_{i=1}^{n} f(y_{d,i}, y_{d,i-1}, x_d) \right) - \sum_{y} \left( P(y|x_d) \sum_{i=1}^{n} f(y_i, y_{i-1}, x_d) \right) \]

- Computing the model expectations:
  - Requires exponentially large number of summations: Is it intractable?

\[ \sum_{y} \left( P(y|x_d) \sum_{i=1}^{n} f(y_i, y_{i-1}, x_d) \right) = \sum_{i=1}^{n} \left( \sum_{y} f(y_i, y_{i-1}, x_d) P(y|x_d) \right) \]

\[ = \sum_{i=1}^{n} \sum_{y_i, y_{i-1}} f(y_i, y_{i-1}, x_d) P(y_i, y_{i-1}|x_d) \]

- Tractable!
  - Can compute marginals using the sum-product algorithm on the chain

© Eric Xing @ CMU, 2014

53
CRF learning

- In practice, we use a Gaussian Regularizer for the parameter vector to improve generalizability

\[
\lambda^*, \mu^* = \arg \max_{\lambda, \mu} \sum_{d=1}^{N} \log P(y_d | x_d, \lambda, \mu) - \frac{1}{2\sigma^2} (\lambda^T \lambda + \mu^T \mu)
\]

- In practice, gradient ascent has very slow convergence
  - Alternatives:
    - Conjugate Gradient method
    - Limited Memory Quasi-Newton Methods
CRFs: some empirical results

- Comparison of error rates on synthetic data

Data is increasingly higher order in the direction of arrow

CRFs achieve the lowest error rate for higher order data
CRFs: some empirical results

- Parts of Speech tagging

<table>
<thead>
<tr>
<th>Model</th>
<th>Error</th>
<th>OOV Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>5.69%</td>
<td>45.99%</td>
</tr>
<tr>
<td>MEMM</td>
<td>6.37%</td>
<td>54.61%</td>
</tr>
<tr>
<td>CRF</td>
<td>5.55%</td>
<td>48.05%</td>
</tr>
<tr>
<td>MEMM+</td>
<td>4.81%</td>
<td>26.99%</td>
</tr>
<tr>
<td>CRF+</td>
<td>4.27%</td>
<td>23.76%</td>
</tr>
</tbody>
</table>

+ Using spelling features

- Using same set of features: HMM >=< CRF > MEMM
- Using additional overlapping features: CRF+ > MEMM+ >> HMM
Conditional Random Fields are partially directed discriminative models.

They overcome the label bias problem of MEMMs by using a global normalizer.

Inference for 1-D chain CRFs is exact:
- Same as Max-product or Viterbi decoding.

Learning also is exact:
- Globally optimum parameters can be learned.
- Requires using sum-product or forward-backward algorithm.

CRFs involving arbitrary graph structure are intractable in general:
- E.g.: Grid CRFs.
- Inference and learning require approximation techniques:
  - MCMC sampling
  - Variational methods
  - Loopy BP