Hidden Markov Model

Ying Wu

Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

http://www.eecs.northwestern.edu/~yingwu

Outline

Example: Hidden Coin Tossing

Hidden Markov Model

Inference and Learning in HMM

Tossing Two Biased Coins

- Two biased coins (C_1 and C_2)
- Person A is selecting the coin and tossing it
- Person B is the observer, and is recording the sequence of H/T
- The selection of the coin is a secrete to B (hidden)
- At a certain time instance t, we have two random variables:
  - x_t indicates which coin that is chosen (C_1 or C_2)
  - z_t indicates the toss (H or T)
- The probabilities are

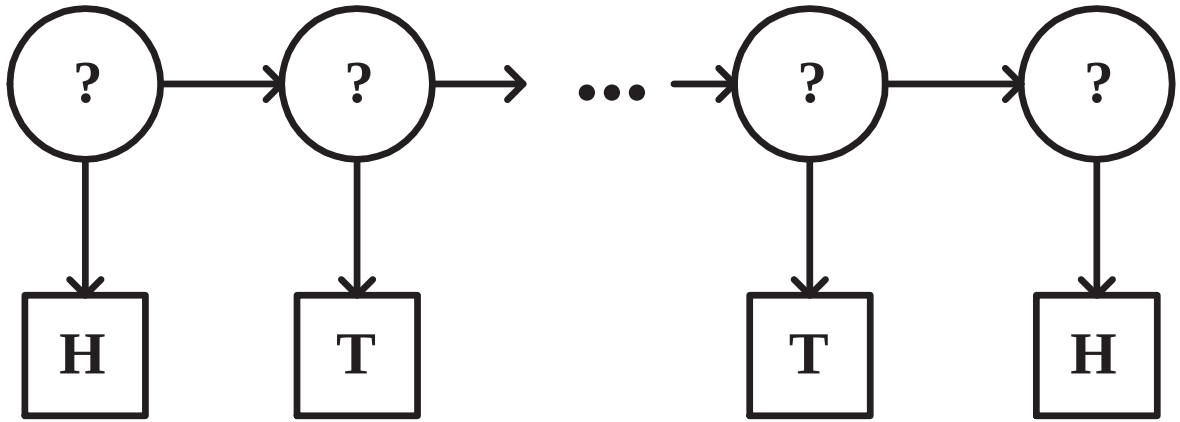
$p(x_t \mid x_{t-1})$	C_1	C_2
C_1	a_{11}	a_{12}
C_2	a_{21}	a_{22}

$p(z_t \mid x_t)$	C_1	C_2
H	p_1	p_2
T	$1-p_1$	$1-p_2$

- $p(x_t \mid x_{t-1})$ is a finite state machine, a discrete Markov random process.

Three Problems

- The Bayesian network



- **P1**: if we have the model, and given an observation sequence, e.g., (*HHTTHTTTH*), determine the likelihood of this sequence
- **P2**: given the observation sequence, estimate which coin was selected at each time, i.e., infer the hidden states
- **P3**: given a number of observation sequences, estimate the parameters of the biased coins, and the parameters of the finite state machine

Outline

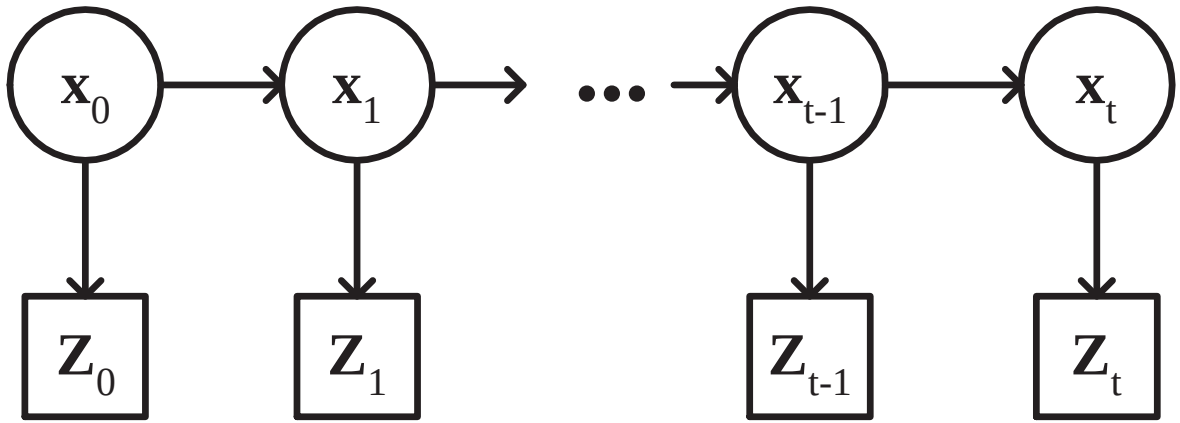
Example: Hidden Coin Tossing

Hidden Markov Model

Inference and Learning in HMM

Graphical model

- A simple Dynamic Bayesian network



- Hidden process x_t
- Hidden states $\mathcal{S} = \{S_1, \ldots, S_N\}$
- Observation process z_t
- Observation symbols $\mathcal{O} = \{O_1, \ldots, O_M\}$

Parameters $\Lambda = (\mathbf{A}, \mathbf{B}, \pi)$

- The Markov process is specified by its transition probability $p(x_t|x_{t-1})$. It is a matrix $\mathbf{A} = \{a_{ij}\}$

$$a_{ij} = p(x_t = S_j|x_{t-1} = S_i), \quad 1 \leq i, j \leq N$$

- The observation is specified by the conditional (or emitting) prob $p(z_t|x_t)$. It is a matrix $\mathbf{B} = \{b_j(k)\}$

$$b_j(k) = p(z_t = O_k|x_t = S_j), \quad 1 \leq j \leq N, \ 1 \leq k \leq M$$

$p(z_t\|x_t)$	S_1	S_2	$\ldots$	S_N
O_1	$b_1(1)$	$b_2(1)$	$\ldots$	$b_N(1)$
O_2	$b_1(2)$	$b_2(2)$	$\ldots$	$b_N(2)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
O_M	$b_1(M)$	$b_2(M)$	$\ldots$	$b_N(M)$

- The initial states (prior) $\pi = \{\pi_i\}$

$$\pi_i = p(x_1 = S_i), \quad 1 \leq i \leq N$$

A Generative Model

- It fully models the generation of the observation sequence
- **S1**: choose initial hidden states, and set $t = 1$
- **S2**: generate observation $z_t = O_k$ according to the state S_i based on $b_i(k)$
- **S3**: transit to a new state $x_{t+1} = S_j$ according to the dynamics
- **S4**: $t \leftarrow t + 1$, go to S2
- This process generates $\underline{\mathbf{Z}}_t = (z_1, \ldots, z_t)$

Three Basic Problems

- **P1: Likelihood**: Given an observation sequence $\underline{\mathbf{Z}}_T = (z_1, \ldots, z_T)$ and the model $\Lambda = (\mathbf{A}, \mathbf{B}, \pi)$, compute the likelihood of this sequence $p(\underline{\mathbf{Z}}|\Lambda)$

- **P2: Inference**: Given an observation sequence $\underline{\mathbf{Z}}_T = (z_1, \ldots, z_T)$ and the model $\Lambda = (\mathbf{A}, \mathbf{B}, \pi)$, determine the hidden states $\underline{\mathbf{X}}_T = (x_1, \ldots, x_T)$

- **P3: Learning**: Estimate the model $\Lambda = (\mathbf{A}, \mathbf{B}, \pi)$, given a set of observation sequences as training data,

$$\underline{\mathbf{Z}}_T^{(1)} = (z_1^{(1)}, \ldots, z_T^{(1)}), \underline{\mathbf{Z}}_T^{(2)} = (z_1^{(2)}, \ldots, z_T^{(2)}), \underline{\mathbf{Z}}_T^{(n)} = (z_1^{(n)}, \ldots, z_T^{(n)})$$

Outline

Example: Hidden Coin Tossing

Hidden Markov Model

Inference and Learning in HMM

Compute the Likelihood: the complexity

- The conditional likelihood

$$\begin{aligned} p(\underline{\mathbf{Z}}_T|\underline{\mathbf{X}}_T, \Lambda) &= \prod_{t=1}^{T} p(z_t|x_t, \Lambda) \\ &= b_{x_1}(z_1) b_{x_2}(z_2) \dots b_{x_T}(z_T) \end{aligned}$$

- We have $p(\underline{\mathbf{X}}_T|\Lambda) = \pi_{x_1} a_{x_1 x_2} a_{x_2 x_3} \dots a_{x_{T-1} x_t}$
- We need to sum over all possible pathes

$$\begin{aligned} p(\underline{\mathbf{Z}}_T|\Lambda) &= \sum_{all\ \underline{\mathrm{X}}_T} p(\underline{\mathbf{Z}}_T|\underline{\mathbf{X}}_T, \Lambda) p(\underline{\mathbf{X}}_T|\Lambda) \\ &= \sum_{x_1, \dots, x_T} \pi_{x_1} b_{x_1}(z_1) a_{x_1 x_2} b_{x_2}(z_2) \dots a_{x_{T-1} x_T} b_{x_T}(z_T) \end{aligned}$$

- Complexity $O(2TN^T)$. This exhaustive method is not realistic
- How to make it more computationally efficient?

Idea: Recursion

$$
\begin{aligned}
\underbrace{p(z_1, \ldots, z_t, x_t)}_{\alpha_t} &= P(z_1, \ldots, z_{t-1}, z_t, x_t) = p(\underline{\mathbf{Z}}_{t-1}, z_t, x_t) \\
&= \int_{x_{t-1}} p(z_t, x_t | \underline{\mathbf{Z}}_{t-1}, x_{t-1}) p(\underline{\mathbf{Z}}_{t-1}, x_{t-1}) \\
&= p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}) \underbrace{p(\underline{\mathbf{Z}}_{t-1}, x_{t-1})}_{\alpha_{t-1}}
\end{aligned}
$$

$$
p(z_1, \ldots, z_t) = \int_{x_t} p(z_1, \ldots, z_t, x_t) = \int_{x_t} \alpha_t
$$

Compute the Likelihood: Forward Algorithm

- Define $\alpha_t(i) = p(z_1, \ldots, z_t, x_t = S_i|\Lambda) = p(\underline{\mathbf{Z}}_t, x_t = S_i|\Lambda)$

S1: Init: $\alpha_1(i) = \pi_i b_i(z_1), 1 \leq i \leq N$

S2: Induction:

$$\alpha_{t+1}(j) = \left[\sum_{k=1}^{N} \alpha_t(k) a_{kj}\right] b_j(z_{t+1})$$

S3: Termination:

$$P(\underline{\mathbf{Z}}_T|\Lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

- Complexity $O(N^2 T)$

Another solution: Backward Algorithm

- Define $\bar{\mathbf{Z}}_t = (z_{t+1}, \ldots, z_T)$
- Define $\beta_t(i) = p(z_{t+1}, z_{t+2}, \ldots, z_T | x_t = S_i, \Lambda)$

S1: Init: $\beta_T(i) = 1, 1 \leq i \leq N$

S2: Induction:

$$\beta_t(i) = \sum_{k=1}^{N} a_{ik} b_k(z_{t+1}) \beta_{t+1}(k)$$

S3: Termination

$$P(\mathbf{Z}_T | \Lambda) = \sum_{i=1}^{N} \beta_1(i) \pi_i$$

- Why does it work? Let's see the recursion

$$\begin{aligned} p(\bar{\mathbf{Z}}_t | x_{t-1}) &= p(z_t, \bar{\mathbf{Z}}_{t+1} | x_{t-1}) \\ &= \int_{x_t} p(z_t, \bar{\mathbf{Z}}_{t+1} | x_t, x_{t-1}) p(x_t | x_{t-1}) \\ &= \int_{x_t} p(z_t | x_t) p(\bar{\mathbf{Z}}_{t+1} | x_t) p(x_t | x_{t-1}) \end{aligned}$$

To See it Clearly

- $\alpha_t(i)$ and $\beta_t(i)$ are closely related

$$\begin{aligned}\alpha_t(i)\beta_t(i) &= p(z_1, \ldots, z_t, x_t = S_i|\Lambda)p(z_{t+1}, \ldots, z_T|x_t = S_i, \Lambda) \\ &= p(\underline{\mathbf{Z}}_T, x_t = S_i|\Lambda)\end{aligned}$$

- This implies (data likelihood)

$$p(\underline{\mathbf{Z}}|\Lambda) = \sum_i \alpha_t(i)\beta_t(i)$$

- This also implies (inference)

$$p(x_t|\underline{\mathbf{Z}}, \Lambda) \propto \alpha_t(i)\beta_t(i)$$

Inferring the Hidden States: belief

- Define $\gamma_t(i) = p(x_t = S_i | \underline{\mathbf{Z}}_T, \Lambda)$ to be the belief of x_t
- The MAP estimate

$$x_t^* = \arg\max_{1 \le i \le N} \gamma_t(i) = \arg\max_{1 \le i \le N} p(x_t = S_i | \underline{\mathbf{Z}}_T, \Lambda)$$

- It it clear that

$$\gamma_t(i) = \frac{p(x_t = S_i, \underline{\mathbf{Z}}_T | \Lambda)}{p(\underline{\mathbf{Z}}_T | \Lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^N \alpha_t(i)\beta_t(i)}$$

- This is the conditional posterior of x_t (i.e., the belief of x_t)
- It does not tell $p(x_1, \ldots, x_T | \underline{\mathbf{Z}}_T, \Lambda)$

Inferring the Hidden States: Viterbi Algorithm

- The MAP of the joint state is actually the *optimal path*
- Define by

$$\delta_t(i) = \max_{x_1,\ldots,x_{t-1}} p(x_1, \ldots, x_t = i, z_1, \ldots, z_t | \Lambda)$$

  the best score of the single path up to time t
- optimal sub-path condition is satisfied

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] b_j(z_{t+1})$$

- Construct the trellis and do dynamic programming
- The optimal path is obtained by backtracking

Learning HMM: Sufficient Statistics

- Define
$$\xi_t(i,j) = p(x_t = S_i, x_{t+1} = S_j | \underline{\mathbf{Z}}_T, \Lambda)$$
- This is the belief of a joint pair
- This can be easily computed

$$\begin{aligned}
\xi_t(i,j) &= \frac{p(x_t = S_i, x_{t+1} = S_j, \underline{\mathbf{Z}}_T | \Lambda)}{p(\underline{\mathbf{Z}}_T, \Lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(z_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(z_{t+1}) \beta_{t+1}(j)}
\end{aligned}$$

- It is straightforward to see the relation to $\gamma_t(i)$
$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

Learning: Baum-Welch Algorithm

- We can have an EM procedure for estimating Λ
- The E-step is done based on the inference process
- Collect the those sufficient statistics

$$\sum_{t=1}^{T} \gamma_t(i) \leftarrow \textit{expected \# of transitions from} S_i$$

$$\sum_{t=1}^{T} \xi_t(i,j) \leftarrow \textit{expected \# of transitions from } S_i \textit{ to } S_j$$

- So the M-step is simply the following

$$\hat{\pi}_i = \textit{expected freq. in } S_i \textit{ at } t = 1 = \gamma_1(i)$$

$$\hat{a}_{ij} = \frac{E[\textit{\# of trans. from } S_i \textit{ to } S_j]}{E[\textit{\# trans. from } S_i]} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\hat{b_j}(k) = \frac{E[\textit{\# of times in } S_j \textit{ obsrv. } O_k]}{E[\textit{\# of times in } S_j]} = \frac{\sum_{t=i, Z_t=O_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$