Processamento Estatístico da Linguagem Natural

Aula 14
Professora Bianca
(Sala 302 – Bloco E)
bianca@ic.uff.br
http://www.ic.uff.br/~bianca/peln/

Aula de Hoje

- Cap. 6 – Jurafsky & Martin – Hidden Markov and Maximum Entropy Models
  - Seções 6.4 e 6.5

Três Problemas Fundamentais de HMMs

1. **Avaliação:** Dado um HMM determinar a probabilidade de uma dada seqüência de observações.
2. **Decodificação:** Dada uma seqüência de observações e um HMM, descobrir a melhor (mais provável) seqüência de estados ocultos.
3. **Aprendizado:** Dada uma seqüência de observações e o conjunto de estados do HMM, aprender os parâmetros do HMM (matrizes A e B.)

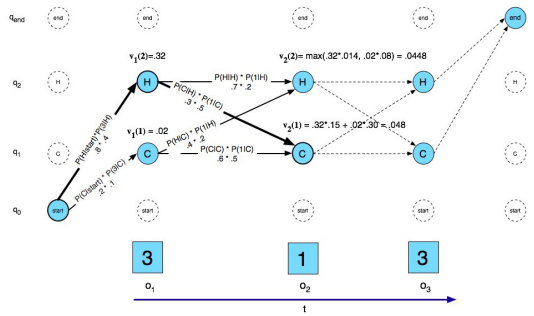
Decodificação

- Dada uma sequência de observações
  Ex.: 3 1 3
  e um HMM, a tarefa do **decodificar** é:
  - Encontrar a melhor sequência de estados **ocultos**.
- Dada a sequência de observações $O=(o_1 o_2 \ldots o_T)$, e um HMM $\Phi = (A,B)$, como podemos encontrar a sequência $Q=(q_1 q_2 \ldots q_T)$ que seja ótima em algum sentido (isto é, que melhor explica as observações).

Decodificação

- Uma possibilidade:
  - Para cada sequencia de estados ocultos
    Ex.: HHH, HHC, HCH,
  - Executar o algoritmo "pra frente" pra calcular $P(\Phi|O)$
- Por que não?
  - O tempo de execução é exponencial: N^T
- Ao invés disso:
  - Utilizamos o algoritmo de Viterbi
  - É um algoritmo de programação dinâmica
  - Usa uma treliça parecida com a do algoritmo "pra frente".

A treliça do algoritmo Viterbi



Intuição do algoritmo Viterbi

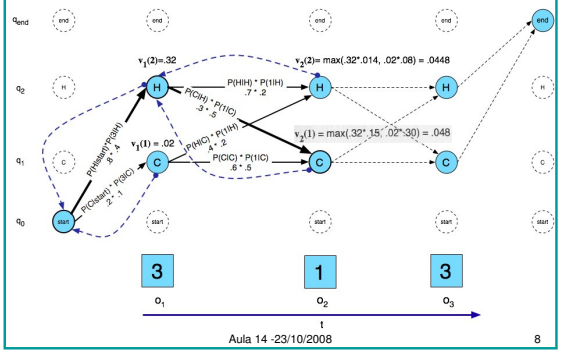
- Processar a sequência de observações da esquerda para direita, preenchendo a treliça.
- Em cada nó da treliça:

$$v_t(j) = \max_{q_0,q_1,\ldots,q_{t-1}} P(q_0,q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda)$$

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

O backtrace do algorimo Viterbi



O algoritmo de Viterbi

function VITERBI(*observations* of len T,*state-graph*) **returns** *best-path*

num-states ← NUM-OF-STATES(*state-graph*)
Create a path probability matrix *viterbi[num-states+2,T+2]*
viterbi[0,0] ← 1.0
for each time step t **from** 1 **to** T **do**
  for each state s **from** 1 **to** *num-states* **do**
    $viterbi[s,t] \leftarrow \max_{1 \le s' \le num\text{-}states} \left[viterbi[s',t-1] * a_{s',s} \right] * b_s(o_t)$
    $back\text{-}pointer[s,t] \leftarrow \operatorname{argmax}_{1 \le s' \le num\text{-}states} \left[viterbi[s',t-1] * a_{s',s} \right]$
Backtrace from highest probability state in final column of *viterbi[]* and return path

A recursão do algoritmo Viterbi

1. **Initialization:**

$$v_1(j) = a_{0,j} b_j(o_1) \quad 1 \le j \le N$$
$$bt_1(j) = 0$$

2. **Recursion** (recall that states 0 and q_F are non-emitting):

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$
$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. **Termination:**

The best score: $P* = v_t(q_F) = \max_{i=1}^{N} v_T(i) * a_{i,F}$

The start of backtrace: $q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{i,F}$

Três Problemas Fundamentais de HMMs

1. **Avaliação:** Dado um HMM determinar a probabilidade de uma dada seqüência de observações.
2. **Decodificação:** Dada uma seqüência de observações e um HMM, descobrir a melhor (mais provável) seqüência de estados ocultos.

Aprendizado: Dada uma seqüência de observações e o conjunto de estados do HMM, aprender os parâmetros do HMM (matrizes A e B.)

Aprendizado

- **Algoritmo Baum-Welch = Forward-Backward = Pra frente-pra trás** (Baum 1972)
- É um caso especial do algoritmo EM ou Expectation-Maximization (Dempster, Laird, Rubin)
- O algoritmo aprende as probabilidades de transição A= $\{a_{ij}\}$ e as probabilidades de emissão B=$\{b_i(o_t)\}$ do HMM.

Entrada para o Algoritmo Baum-Welch

- O = sequência de observações não-rotuladas
- Q = vocabulário de estados ocultos
- Para a tarefa do "sorvete"
  - O = {1,3,2.,,,.}
  - Q = {H,C}

Começando com Cadeias de Markov

- Como treinar?
  - Executar o modelo para a sequencia de observações O.
- Como não é oculto, sabemos os estados percorridos = observações.
- Com essa informação, o treinamento será::
  - B = $\{b_k(o_t)\}$: Como cada estado só pode gerar um símbolo, as probabilidades de observação B são iguais a 1.0
  - A = $\{a_{ij}\}$: $a_{ij} = \frac{C(i \to j)}{\sum_{q \in Q} C(i \to q)}$

Estendendo essa intuição para HMMs

- Para HMMs, não podemos obter essas contagens diretamente a partir das sequências.
- Idéias do Baum-Welch:
  - Estimar as contagens iterativamente.
    - Começar com uma estimativa para a_{ij} e b_k, iterativamente melhorar as estimativas.
  - Como obter as estimativas
    - Calcular a probabilidade de uma observação usando o algoritmo "pra-frente"
    - Dividir essa probabilidade entre todos os caminhos que contribuíram.

O algoritmo "pra-trás" (backward)

- Definimos a probabilidade pra-trás da seguinte maneira:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..o_T, | q_t = i, \Phi)$$

- É a probabilidade de gerar observações parciais O_{t+1}^T a partir do tempo *t+1* até o final, dado que o HMM está no estado *i* no tempo t e dado o HMM Φ.

O algoritmo "pra-trás"

- Calculamos as probabilidades pra-trás por indução:
  - Inicialização:

$$\beta_T(i) = a_{iN}, \quad 1 \le i \le N$$

  - Indução:

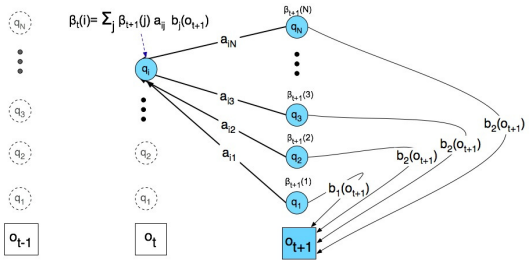
$$\beta_t(i) = \sum_{i=1}^{N-1} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 < i < N, 0 < t < T$$

  - Terminacão:

$$P(O|\lambda) = \alpha_T(N) = \beta_T(1) = \sum_{j=1}^{N-1} a_{1j}\, b_j(o_1)\, \beta_1(j)$$

Passo indutivo do algoritmo "pra-trás"

Cálculo de $\beta_t(i)$ pela soma ponderada de valores sucessivos β_{t+1}



Intuição para re-estimar a_{ij}

- Estimaremos a_{ij} através da equação:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- Intuição para calcular o numerador:
  - Suponha que tenhamos uma estimativa da probabilidade que uma dada transição i->j seja feita no tempo t na sequência de observações.
  - Se sabemos essa probabilidade para cada *t*, podemos somá-las para obter o valor esperado (contagem) para i->j.

Re-estimando a_{ij}

- Seja ξ_t a probabilidade de estar no estado *i* no tempo *t* e estado *j* no tempo *t+1*, dados $O_{1..T}$ e o modelo Φ:

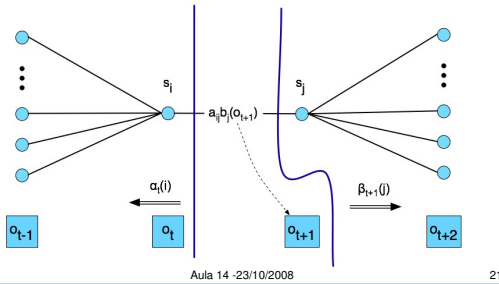
$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda)$$

- Podmos calcular ξ a partir de not-quite-ξ, que é:

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda)$$

Calculando not-quite-ξ

Os quatro componentes de $P(q_t = i, q_{t+1} = j, O|\lambda)$: α, β, a_{ij} **e** $b_j(o_t)$



De not-quite-ξ para ξ

- Queremos:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda)$$

- Temos:

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda)$$

- Que calculamos como:

$$\text{not-quite-}\xi_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

De not-quite-ξ para ξ

- Queremos:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda)$$

- Temos:

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda)$$

- Já que:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)}$$

- Chegamos a:

$$\xi_t(i,j) = \frac{not\_quite\_\xi_t(i,j)}{P(O \mid \lambda)}$$

De not-quite-ξ para ξ

$$\xi_t(i,j) = \frac{not\_quite\_\xi_t(i,j)}{P(O \mid \lambda)}$$

$$\text{not-quite-}\xi_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

$$P(O|\lambda) = \alpha_T(N) = \beta_T(1) = \sum_{i=1}^{N} \alpha_t(j) \beta_t(j)$$

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\alpha_T(N)}$$

De ξ para a_{ij}

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- O número esperado de transições do estado *i* para o estado *j* é a soma de ξ para todos os valores de *t*.
- O número total esperado de transições a partir do estado *i* é a soma da probabilidade de todas as transições que saem do estado *i* para todos os valores de *t*.
- Fórmula final:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$$

Re-estimando a probabilidade de observacão b

- This is the probability of a given symbol v_k from the observation vocabulary V, given a state j: $\hat{b}_j(v_k)$.

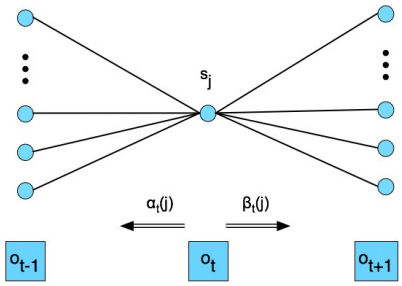
$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state j and observing symbol } v_k}{\text{expected number of times in state } j}$$

Precisamos saber a probabilidade de estar no estado *j* no tempo *t:*

$$\gamma_t(j) = P(q_t = j|O,\lambda) \qquad \gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$$

$$\gamma_t(j) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \qquad \hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Calculando gamma



Em resumo

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$$

É razão entre o número esperado de transições do estado i para o estado j e o número esperado de todas as transições a partir do estado i

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

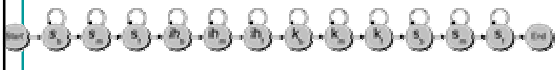
É razão entre o número esperado de vezes que a observação foi emitida a partir do estado j é v_k, e o número esperado de vezes que qualquer emissão é emitida a partir do estado *j*.

Resumo: Algoritmo Pra-frente Pra-Trás

1) Incialize Φ=(A,B)
2) Calcule α, β, ξ
3) Estime novos Φ'=(A,B)
4) Troque Φ por Φ'
5) Se não tiver convergido vá para o passo 2

O problema de aprendizado: Caveats

- A estrutura de rede do HMM é sempre criada manualmente.
  - Nâo existe algoritmo de indução da estrutura ótima + probabilidades
  - É sempre uma rede de Bakis = pra frente no tempo.
  - Subcaso da rede de Bakist: rede "beads-on-string":



- Baum-Welch só garante retornar o máximo local, ao invés do máximo global.