Đại học Quốc gia Thành phố Hồ Chí Minh

Trường Đại học Khoa học Tự nhiên

ĐỒ ÁN
MÔ HÌNH MARKOV ẨN

Nhóm sinh viên thực hiện:

20120014 - Vương Gia Huy

20120021 - Hồ Văn Sơn

20120304 - Phan Trần Khanh

Giảng viên hướng dẫn: PGS. TS. Nguyễn Đình Thúc

Học phần Toán ứng dụng & thống kê

Lớp 20TN

05 - 2022

Mục Lục

Sau đây là những gì nhóm chúng em tìm hiểu trong phần lý thuyết và trình bày lại các thành phần và cơ chế hoạt động của mô hình Markov ẩn.

1. Các thành phần của một mô hình Markov ẩn là gì? Chúng khác gì với mô hình Markov?

Các thành phần của mô hình Markov ẩn:

- $Q = q_1 q_2 ... q_N$ là một tập gồm N trạng thái ẩn.
- $A = a_{11} ... a_{ij} ... a_{NN}$ là một ma trận A gồm các giá trị là xác suất khi chuyển đổi trạng thái, mỗi giá trị a_{ij} đại diện cho xác suất khi chuyển từ trạng thái i sang trạng thái j và tổng tất cả các xác suất này bằng 1 ($\sum_{j=1}^{N} a_{ij} = 1 \ \forall i$).
- $O = o_1 o_2 ... o_T$ là một chuỗi liên tục gồm T dấu hiệu quan sát được, mỗi chuỗi được trích ra từ tập $V = v_1, v_2, ..., v_V$ là tập những dấu hiệu quan sát được ở vật thể bị quan sát.
- $B = b_i(o_t)$ là một chuỗi các dấu hiệu có khả năng xảy ra, hoặc có thể được biết đến như là khả năng bộc phát một dấu hiệu nào đó của một trạng thái ẩn, mỗi giá trị là xác suất một dấu hiệu o_t của vật thể bị quan sát, được sản sinh ra, từ một trạng thái i, còn gọi là emission probability hay output probability.
- $\pi = \pi_1, \pi_2, ..., \pi_N$ là phân phối xác suất khởi đầu đối với các trạng thái ẩn. π_i là xác suất mà xích Markov sẽ bắt đầu ở trạng thái i. Một vài trạng thái j sẽ có thể có $\pi_j = 0$, điều này có nghĩa chúng không thể là trạng thái ban đầu. Đồng thời thì $\sum_{i=1}^{N} \pi_i = 1$.

Sự khác nhau giữa mô hình Markov và mô hình Markov ẩn là:

- Mô hình Markov đơn giản nhất là xích Markov, trong đó các trạng thái được quan sát trực tiếp.
- Đối với mô hình Markov ẩn, bản thân các trạng thái không được quan sát trực tiếp, thay vào đó chúng ta giả sử rằng có một số chuỗi các trạng thái hình thành một xích Markov, tuy không thể quan sát trực tiếp, nhưng những trạng thái này phát sinh ra những dấu hiệu ngẫu nhiên quan sát được. Mỗi trạng thái có phân phối xác suất riêng để sinh ra các dấu hiệu quan sát khác nhau. Vì vậy, không thể xác định chính xác được trạng thái hiện tại của mô hình nếu chỉ dựa vào dấu hiệu được phát sinh ngay lúc đó một cách đơn lẻ.

Có thể tham khảo thêm tại đây [1].

2. Các giả thiết (assumption) đặt ra cho mô hình Markov ẩn là gì? Tìm ví dụ các bài toán mà các giả thiết này hợp lý và bất hợp lý

Có 2 giả thiết (assumption) đặt ra cho mô hình Markov ẩn như sau:

1 - Xích Markov first-order, xác suất của một trạng thái cụ thể chỉ phụ thuộc vào trạng thái liền trước nó. Các chuỗi trạng thái thường được giả định là đồng nhất trong đó xác suất chuyển đổi trạng thái không phụ thuộc vào t. Giả sử trạng thái q_t tạo một xích Markov first-order, thì

$$P(q_t|q_{t-1}, ..., q_1) = P(q_t|q_{t-1}) \text{ với } t = 2, ..., T$$

2 - Xác suất của một dấu hiệu quan sát được o_i được sinh ra chỉ phụ thuộc vào trạng thái q_i tương ứng và độc lập có điều kiện với bất kì dấu hiệu quan sát được hay trạng thái trong quá khứ.

$$P(o_i|q_1, ..., q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = P(o_i|q_i)$$

Ví dụ về các bài toán mà giả thiết hợp lý và bất hợp lý là:

Bạn Khanh là một người cuồng cà phê Starbucks, tôi gặp bạn Khanh ở trường mỗi ngày, tôi muốn biết giá ly cà phê tại Starbucks tăng hay giảm, dữ kiện tôi có duy nhất là bạn Khanh vui hoặc không vui. Nếu bạn Khanh vui thì tôi đoán là giá cà phê đã giảm, còn ngược lại thì tôi đoán là giá cà phê đã tăng, tất nhiên bạn Khanh vui không có nghĩa là giá ly cà phê Starbucks giảm, đó chỉ là dự đoán, vì thực tế có khi giá cà phê Starbucks giảm nhưng bạn Khanh vẫn không vui. Vậy lúc này, giá cà phê tăng hay giảm là trạng thái ẩn đối với tôi, tôi không biết gì về giá cà phê, dữ kiện duy nhất tôi có là bạn Khanh vui hay không vui. **Biết rằng giá của ly cà phê vào hôm nay chỉ phụ thuộc vào giá ly cà phê ngày hôm liền trước và bạn Khanh có vui vào hôm nay hay không thì phụ thuộc vào giá cà phê của hôm đó**. Quan sát bạn Khanh vui hay không vui chính là dữ kiện mà tôi quan sát được (dấu hiệu của vật thể bị quan sát), còn dữ kiện giá là **ẩn**, dựa vào dữ kiện vui hay không vui để đoán xem giá tăng hay giảm. Đây chính là ý cơ bản nhất tại sao gọi là mô hình Markov ẩn. Bảng phân phối xác suất bộc phát các dấu hiệu của vật thể bị quan sát (ở bài toán này là Khanh) của các trạng thái ẩn được biểu diễn như sau:

Trạng thái \ Dấu hiệu	0	1
T	0.8	0.2
G	0.3	0.7

Bảng 1. Bảng phân phối xác suất của các dấu hiệu quan sát do từng trạng thái ẩn bộc phát.

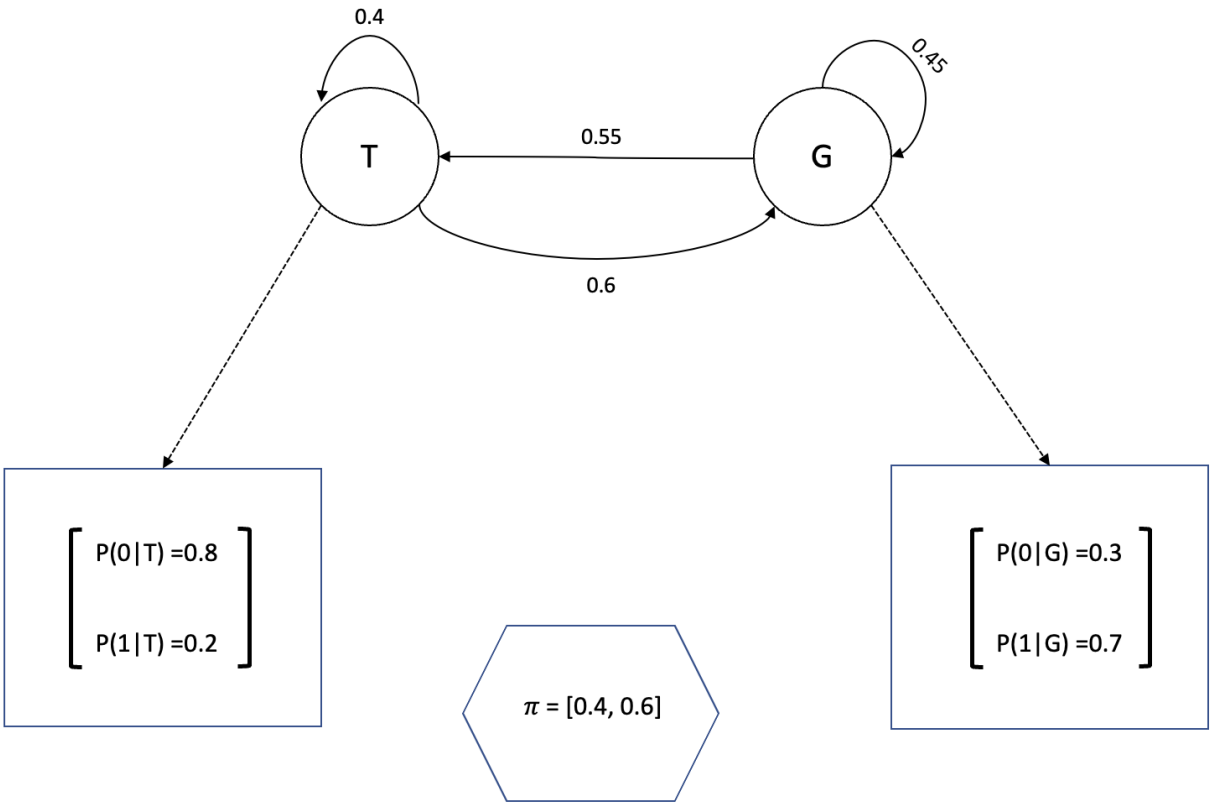
Ma trận trên còn được gọi là ma trận quan sát, thường kí hiệu là B. Biểu diễn giá trị cụ thể là $b_i(o_t)$ với i nhận các giá trị là {T(tăng), G(giảm)}, o_t nhận các giá trị là {0(không vui), 1(vui)}. Ví dụ: $b_T(1) = 0.2$, $b_T(0) = 0.8$, $b_G(1) = 0.7$, $b_G(0) = 0.3$.

Như vậy, dữ kiện bài trên áp dụng mô hình Markov ẩn là hoàn toàn hợp lý vì ta có q_i là trạng thái của ngày thứ i chỉ phụ thuộc vào q_{i-1} là trạng thái của ngày thứ $i-1$ (tức là ngày liền trước ngày i), và o_i là dấu hiệu quan sát được {vui, không vui} chỉ phụ thuộc vào trạng thái q_i.

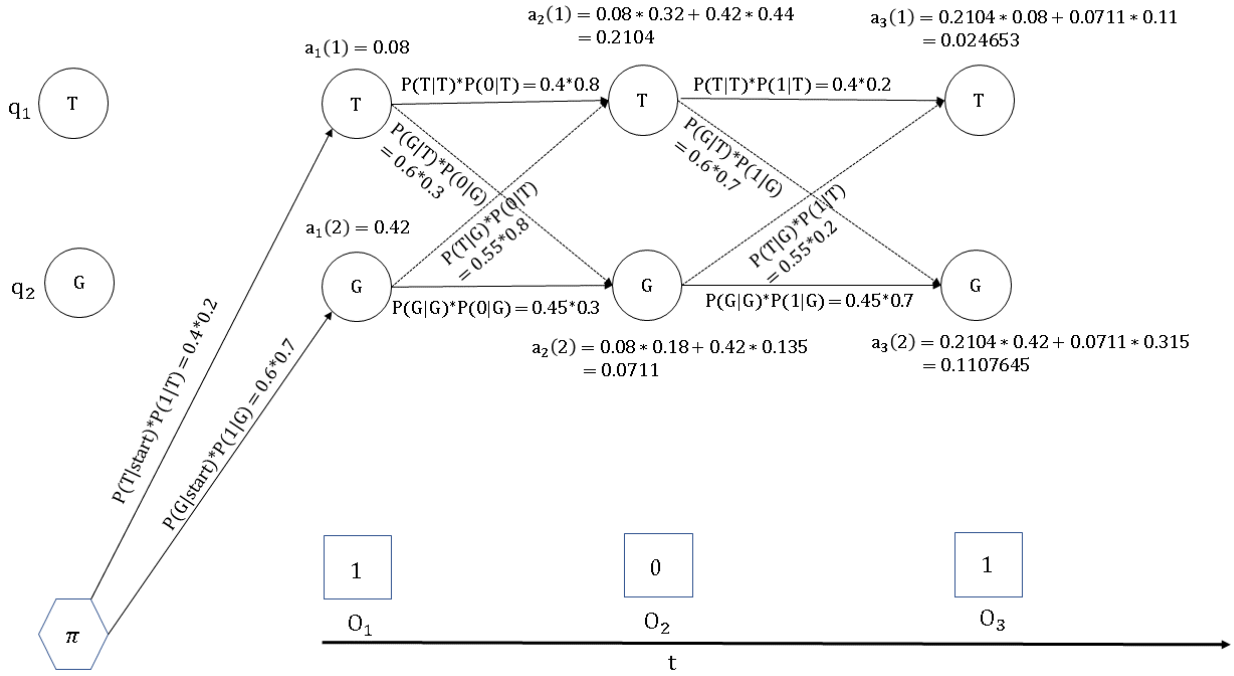
Ngược lại, bài toán trên sẽ trở nên bất hợp lý khi áp dụng mô hình Markov ẩn nếu ta xoá đi dữ kiện, "**Biết rằng giá của ly cà phê vào hôm nay chỉ phụ thuộc vào giá ly cà phê ngày hôm liền trước và bạn Khanh có vui vào hôm nay hay không thì phụ thuộc vào giá cà phê của hôm đó**" vì khi đó ta không biết được liệu q_i là trạng thái của ngày thứ i có phụ thuộc vào q_j là trạng thái ngày thứ j ($\neq i-1$) nào hay không, hơn nữa ta cũng không biết được rằng liệu o_i là dấu hiệu quan sát được {vui, không vui} có chỉ phụ thuộc vào trạng thái q_i hay còn phụ thuộc vào các dấu hiệu quan sát được hay trạng thái của ngày nào đó trước ở quá khứ nữa.

3. Cho một mô hình Markov ẩn với các tham số đã biết, thuật toán tiến trước (forward algorithm) được dùng để xác định độ hợp lý (likelihood) của một chuỗi quan sát (observation). Mô tả và đánh giá độ phức tạp của thuật toán tiến trước.

Để tính độ hợp lý của một chuỗi quan sát, thay vì dùng những giải thuật liên quan đến hàm mũ thì thuật toán tiến trước (**forward algorithm**) thường được sử dụng rộng rãi hơn vì dễ dàng cài đặt cùng với chi phí tính toán thấp $O(N^2T)$. **Forward algorithm** được xem như là một dạng thuật toán quy hoạch động, bởi vì nó lưu những giá trị trung gian để xây dựng nên xác suất của chuỗi quan sát. Cụ thể hơn, ta sẽ dùng bài toán ở câu 2 với các kí hiệu tăng(T), giảm(G), không vui(0), vui(1). Ta giả sử bài toán bằng hình vẽ tường minh như sau:



Giả sử ta cần tìm mức độ hợp lý (likelihood) của chuỗi dấu hiệu quan sát được là $O = \{1, 0, 1\}$, sau đây là kết quả xác suất tính toán theo quy trình hình mắt cáo (trellis) dựa trên thuật toán forward dành cho bài toán đã đề cập:



Dựa vào ví dụ bài toán trên ta đưa ra dạng tổng quát hơn, với giá trị $a_t(j)$ là xác suất ở trạng thái ẩn j, sau t dấu hiệu quan sát được đầu tiên, và có được chuỗi quan sát liên tục $O = \{o_1,\ o_2,\ ...,\ o_t\}$. Đặt mô hình tự động này là λ, như vậy

$$a_t(j) = P(o_1, o_2, ..., o_t, q_t = j|\lambda)$$

Do đó, xác suất $a_t(j)$ tại thời điểm t sẽ được tính hoàn toàn dựa vào thời điểm $t-1$ như sau

$$a_t(j) = \sum_{i=1}^{N} a_{t-1}(i) a_{ij} b_j(o_t)$$

Trong đó,

- $a_{t-1}(i)$ là xác suất ở trạng thái i tại thời điểm $t-1$.
- a_{ij} là xác suất chuyển đổi từ trạng thái i sang trạng thái j.
- $b_j(o_t)$ là độ hợp lý của dấu hiệu o_t quan sát được khi ở trạng thái ẩn j.

Tiếp theo là quy trình của thuật toán tiến tới (forward algorithm) tính $a_t(j)$ bằng đệ quy như sau

1. Khởi tạo:

$$a_1(j) = \pi_j b_j(o_1) \text{ với } 1 \leq j \leq N$$

2. Đệ quy:

$$a_t(j) = \sum_{i=1}^{N} a_{t-1}(i) a_{ij} b_j(o_t) \text{ với } 1 \leq j \leq N,\ 1 < t \leq T$$

3. Kết thúc vào giai đoạn cuối cùng là $t = T$:

$$P(O|\lambda) = \sum_{i=1}^{N} a_T(i)$$

Từ quy trình của thuật toán đã đề cập trên, chúng ta có thể suy được mã giả của forward algorithm, cụ thể như sau

- Input: Ma trận xác suất chuyển đổi trạng thái a, ma trận xác suất bộc phát dấu hiệu b, xác suất khởi đầu π, chuỗi T dấu hiệu quan sát được và N trạng thái ẩn của mô hình.
- Output: Xác suất $forward\_prob$ để sinh ra được chuỗi quan sát đầu vào.

```
function FORWARD(T observations, N hidden states) returns forward_prob
    create a forward probability matrix: forward[N,T]
    for each state s from 1 to N do:    #initialization step
        forward[s,1] = pi_s * b_s(o_1)
    for each time step t from 2 to T do:    #recursion step
        for each state s from 1 to N do:
            sum = 0
            for each state S from 1 to N do:
                sum += forward[S, t - 1] * a_{S -> s} * b_S(o_t)
            forward[s,t] = sum
    forward_prob = 0
    for each state s from 1 to N do:    #termination step
        forward_prob += forward[s,T]
    return forward_prob
```

Nhóm cũng đã tiến hành cài đặt thuật toán tiến tới cho mô hình Markov ẩn bằng ngôn ngữ Python.

```python
def forward_HMMs(O, A, B, pi):

    # O la chuoi quan sat (observations)
    # A la ma tran xac suat chuyen trang thai (transition_probability_matrix)
    # B la ma tran xac suat boc phat (emission_probabilities)
    # pi la phan phoi xac suat khoi dau

    N = A.shape[0] # so trang thai co the xay ra
    T = O.shape[0] # do dai cua chuoi quan sat

    a = np.zeros((T, N)) # Khoi tao ma tran voi a[t][j] la xac suat o trang thai an j sau t dau hieu quan sat duoc dau tien

```

```
    a[0, :] = pi[:]*B[:, O[0]] # Khoi tao cac gia tri cua trang thai t = 0

    for t in range (1, T):
        for j in range (0, N):
            a[t, j] = np.dot(a[t-1, :], A[:, j]) * B[j, O[t]] # thao tac co
    so

    forward_prob = np.sum(a[T-1, :]) # do hop ly cua chuoi quan sat

    return forward_prob
```

với,

- O là chuỗi quan sát. Ở đây trong chuỗi (dãy) chỉ sử dụng giá trị số nguyên để đáp ứng bài toán nêu ra.
- A là ma trận xác suất chuyển đổi giữa hai trạng thái ẩn.
- B là ma trận độ hợp lý của một trạng thái ẩn khi bộc phát ra các dấu hiệu quan sát được. (tương tự Bảng 1)
- $forward\_prob$ là độ hợp lý của một chuỗi quan sát O cho trước.
- $\pi(pi)$ là phân phối xác suất ban đầu của từng trạng thái ẩn.

Đánh giá độ phức tạp của thuật toán tiến tới (forward algorithm).

Thuật toán trên có độ phức tạp là $O(N^2T)$. Thật vậy, ta có thao tác cơ sở sau:

```
forward_prob[i, j] = np.dot(forward_prob[:, j-1].T, a[:, j])*b[i, j]
```

Lệnh np.dot có chi phí là N. Chi phí của hai dòng for là $N \times T$.

$\Rightarrow$ Chi phí của thao tác cơ sở là $N \times N \times T$.

$\Rightarrow$ Chi phí của thuật toán tiến tới (Forward Algorithm) là $O(N^2T)$.

4. Cho một mô hình Markov ẩn với các tham số đã biết, thuật toán Viterbi được dùng để xác định chuỗi trạng thái (state) khả dĩ nhất. Mô tả và đánh giá độ phức tạp của thuật toán Viterbi.

Để tìm ra chuỗi trạng thái khả dĩ nhất $Q = \{q_1, q_2, ..., q_T\}$ đã phát sinh ra chuỗi quan sát O, thuật toán được sử dụng rộng rãi nhất là **Viterbi**. Cũng giống như thuật toán tiến tới (Forward Algorithm), thuật toán **Viterbi** cũng là thuật toán sử dụng quy hoạch động, và tận dụng quy hoạch động để tính toán xác suất theo quy trình hình mắt cáo (dynamic programming trellis).

Thuật toán Viterbi có thể được biểu diễn bằng đẳng thức như sau:

$$q^*_{1:T} = \arg\max_{q_{1:T}} P[q_{1:T}|o_{1:T}]$$

với $q^*_{1:T}$ là chuỗi trạng thái khả dĩ nhất của một chuỗi quan sát cho trước.

Đặt $\delta_t(j)$ là xác suất mà mô hình Markov ẩn λ đang ở trạng thái j sau khi quan sát được t dấu hiệu đầu tiên, có được chuỗi quan sát $O = \{o_1, o_2, ..., o_t\}$, và tất nhiên mô hình λ ở trạng thái t khi đã chuyển đổi hết chuỗi trạng thái khả dĩ nhất là $q_1, q_2, ..., q_{t-1}$.

$$\delta_t(j) = \max_{q_1,q_2,...,q_{t-1}} P(q_1, q_2, ..., q_{t-1}, o_1 o_2 ... o_t, q_t = j|\lambda)$$

$\max_{q_1,q_2,...,q_{t-1}}$ sẽ được hiểu là việc chúng ta lấy chuỗi trạng thái ẩn khả dĩ nhất trước thởi điểm t.

Từ biến $\delta_t(i)$, ta phải đi tìm chuỗi trạng thái khả dĩ nhất đến thời điểm t.

$$q_{1:t} = \arg\max_{1\le i\le N}[\delta_t(i)] \text{ với } 1 \le t \le T$$

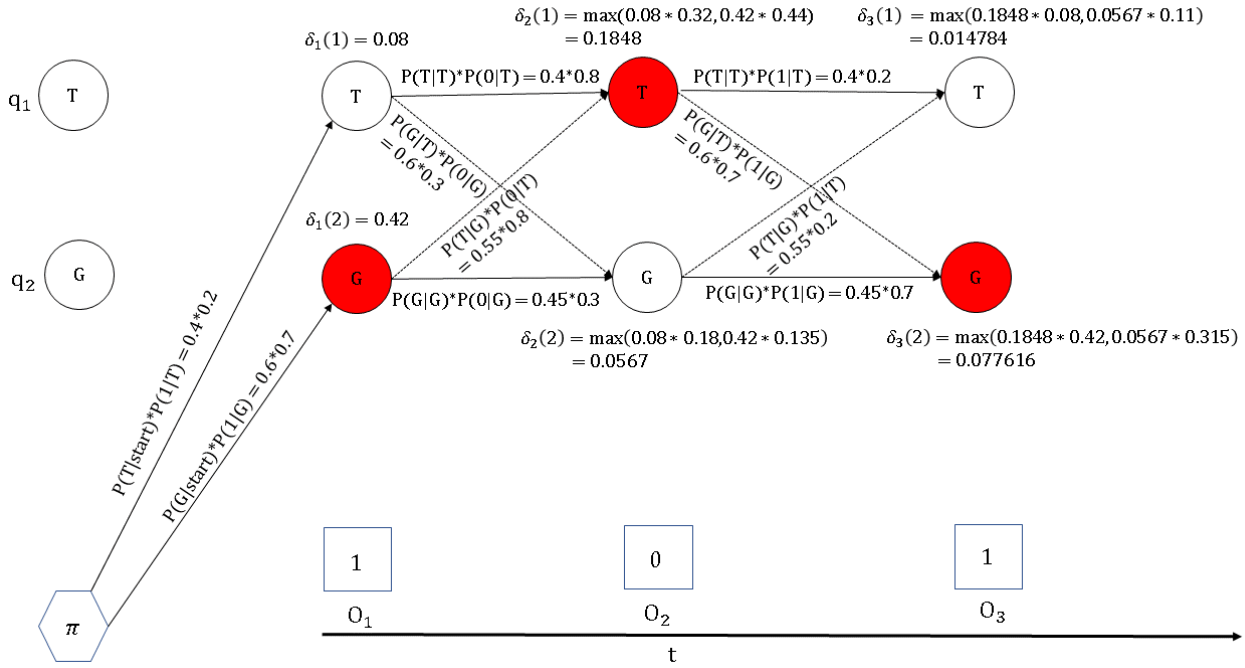
Thuật toán Viterbi sử dụng quy hoạch động để tìm chuỗi trạng thái ẩn khả dĩ nhất, vậy nên để quá trình lập trình được tường minh, chúng ta nên sử dụng công thức đệ quy để tính toán xác suất $\delta_t(i)$.

$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(o_t)$$

trong đó,

- $\delta_{t-1}(i)$ là xác suất cao nhất của đoạn chuỗi trạng thái dẫn đến trạng thái i ở thời điểm $t-1$. (xác suất của đoạn chuỗi trạng thái khả dĩ nhất trước thời điểm t)
- a_{ij} là xác suất chuyển từ trạng thái i sang trạng thái j
- $b_j(o_t)$ là độ hợp lý của dấu hiệu o_t quan sát được khi ở trạng thái ẩn j.

Cụ thể hơn, ta tiếp tục xét bài toán ở câu 2, thủ tục Viterbi được diễn tả qua hình sau



Vậy, ta kết luận được chuỗi trạng thái ẩn khả dĩ nhất được tìm ra ở bài toán là {G, T, G}.

Quy trình vận dụng thuật toán Viterbi có thể được mô tả đơn giản như sau

1. Khởi tạo:

$$\delta_1(j) = \pi_j b_j(o_1),\ 1 \le j \le N$$

$$bt_1(j) = 0,\ 1 \le j \le N$$

2. Đệ quy:

$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(o_t);\ 1 \le j \le N, 1 < t \le T$$

$$bt_t(j) = \arg\max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(o_t);\ 1 \le j \le N, 1 < t \le T$$

3. Kết thúc:

   Xác suất lớn nhất:

$$P* = \max_{i=1}^{N} \delta_T(i)$$

   Điểm bắt đầu của quá trình quay lui:

$$q_T* = \arg\max_{i=1}^{T} \delta_T(i)$$

Từ quy trình vận dụng thuật toán Viterbi đã đề cập trên, chúng ta có thể suy được mã giả của Viterbi algorithm, cụ thể như sau

- Input: Ma trận xác suất chuyển đổi trạng thái a, ma trận xác suất bộc phát dấu hiệu b, xác suất khởi đầu π, chuỗi T dấu hiệu quan sát được và N trạng thái ẩn của mô hình.
- Output: Chuỗi trạng thái ẩn khả dĩ nhất *best_path* để sinh ra chuỗi trạng thái quan sát được và xác suất *path_prob* tương ứng.

```
function VITERBI(T observations, N hidden states) returns best_path, path_prob
create a path probability matrix: viterbi[N,T]
for each state s from 1 to N do:    #initialization step
    viterbi[s,1] = pi_s * b_s(o_1)
    backpointer[s,1] = 0
for each time step t from 2 to T do: #recursion step
    for each state s from 1 to N do:
        MAX = 0
        for each state S from 1 to N do:
            MAX = max(MAX, viterbi[S, t - 1] * a_{S -> s} * b_s(o_t))
        viterbi[s,t] = MAX
        backpointer[s,t] = argmax(MAX) #Get the hidden state corresponding
MAX = 0
for each state s from 1 to N do:    #termination step
    MAX = max(MAX, viterbi[s, T])
path_prob = MAX
best_path_pointer = argmax(MAX) #Get the hidden state corresponding
best_path = [best_path_pointer]
for each time step t from T downto 2 do:
    best_path = [backpointer[best_path_pointer, t]] + best_path
    best_path_pointer = backpointer[best_path_pointer, t]
return best_path, path_prob
```

Nhóm cũng đã tiến hành cài đặt thuật toán Viterbi cho mô hình Markov ẩn bằng ngôn ngữ Python.

```python
# Day la ham tra ve xac suat xay ra quan sat khi biet trang thai
def probOfObservationGivenState(B, index_observation, state, o, Q):
    # B la ma tran xac suat boc phat (emission_probability_matrix)
    # o la tap cac quan sat ban dau da biet
    # Q la tap trang thai ban dau da biet
    # observation va state dung de tinh xac suat xay ra quan sat observation o
    trang thai state
    index_state = Q.index(state)
    return B[index_state, index_observation]

def viterbi_HMMs(O, A, B, pi, Q, o):
    # O la chuoi quan sat dau vao (observations)
    # A la ma tran xac suat chuyen doi trang thai (transition_probability_matrix
    )
    # pi la phan phoi xac suat khoi dau (initial_distribution)
    # B la ma tran xac suat boc phat (emission_probability_matrix)
```

```
    # Q la tap cac trang thai da cho
    # o la tap quan sat da cho
    hidden_states = [] # chuoi trang thai can tim

    prev_state = -1
    for index, observation in enumerate(O):
        index_observation = o.index(O[index])
        delta = []
        for s in range(len(Q)):
            if index == 0:
                transition_prob = pi[s]
            else:
                transition_prob = A[prev_state, s]

            emission_prob = probOfObservationGivenState(B, index_observation, Q[
    s], o, Q)
            delta.append(emission_prob*transition_prob)

        delta_max = max(delta)
        prev_state = delta.index(delta_max)
        hidden_states.append(Q[prev_state])

    return hidden_states
```

với,

- O là chuỗi quan sát. Ở đây trong chuỗi (dãy) chỉ sử dụng giá trị số nguyên để đáp ứng bài toán nêu ra.
- Q là tập các trạng thái ẩn.
- A là ma trận xác suất chuyển đổi giữa hai trạng thái ẩn.
- B là ma trận độ hợp lý của một trạng thái ẩn khi bộc phát ra các dấu hiệu quan sát được. (tương tự Bảng 1)
- *hidden_states* là chuỗi trạng thái ẩn khả dĩ nhất từ chuỗi quan sát O.
- $\pi(pi)$ là phân phối xác suất ban đầu của từng trạng thái ẩn.

Đánh giá độ phức tạp của thuật toán Viterbi (Viterbi algorithm).

Thuật toán trên có độ phức tạp là $O(N^2T)$. Thật vậy, ta có thao tác cơ sở sau:

```
index_state = Q.index(state)
```

Lệnh index có chi phí là n với n là số lượng phần tử của đối tượng, ở bài toán này chi phí sẽ là N. Chi phí của hai dòng for là $N \times T$.

$\Rightarrow$ Chi phí của thao tác cơ sở là $N \times N \times T$.

$\Rightarrow$ Chi phí của thuật toán Viterbi (Viterbi Algorithm) là $O(N^2T)$.

5. Cho một chuỗi quan sát, giả sử ta cho rằng chuỗi quan sát này được sinh ra từ một mô hình Markov ẩn với tham số chưa biết, thuật toán Baum-Welch được dùng để ước lượng các tham số này. Thuật toán Baum-Welch là trường hợp đặc biệt của thuật toán Kỳ vọng-Tối ưu (Expectation-Maximization, hay EM). Thuật toán này gồm 2 bước: bước E (Expectation, hay kỳ vọng) và bước M (Maximization, hay tối ưu).

5.1. Mô tả tổng quát về thuật toán Kỳ vọng-Tối ưu

Thuật toán **EM** được sử dụng để thu thập các tham số khả thi nhất cho mô hình thống kê khi một số dữ liệu trong bộ dữ liệu còn thiếu. Rộng hơn, thuật toán **EM** cũng có thể được áp dụng khi có dữ liệu không quan sát trực tiếp được, tức là dữ liệu ẩn (latent variables) và đồng thời không thể quan sát được ngay từ đầu quá trình thu thập. Trong trường hợp đó, chúng ta chỉ việc giả định rằng bộ dữ liệu bị thiếu dữ liệu và tiến hành áp dụng thuật toán **EM**. Thuật toán **EM** sẽ dự đoán giá trị của chúng bằng cách sử dụng giá trị của các biến dữ liệu mà đã quan sát được khác.

Thuật toán EM này là cơ sở của nhiều thuật toán phân cụm không được giám sát (unsupervised clustering algorithms) trong lĩnh vực máy học. Thuật toán này có hai bước tính toán chính là kỳ vọng và tối ưu.

- Bước E: Sử dụng dữ liệu đã được quan sát sẵn của tập dữ liệu, ước tính (dự đoán) các giá trị của bộ dữ liệu bị thiếu hay bị ẩn.
- Bước M: Bộ dữ liệu hoàn chỉnh sẽ được tạo dựng sau bước kỳ vọng (E) và được sử dụng để cập nhật các tham số.

Đầu tiên, chúng ta sẽ giả định rằng tập dữ liệu hoàn chỉnh bao gồm $Z = (X, Y)$ nhưng chỉ có X được quan sát (hay X là dữ kiện có thể quan sát được). $l(\theta; X, Y)$ là log-likelihood của bộ dữ liệu hoàn chỉnh, trong đó θ là vectơ tham số chưa biết mà chúng ta muốn tìm bằng cách sử dụng MLE (Maximum likelihood estimation).

- Bước E: Bước E của thuật toán EM sẽ tính giá trị kỳ vọng của $l(\theta; X, Y)$ với dữ liệu quan sát được là X và ước lượng tham số hiện tại θ_{old}. Cụ thể thì chúng ta sẽ xác định

$$\begin{aligned} Q(\theta; \theta_{old}) &:= E[l(\theta; X, Y)|X, \theta_{old}] \\ &= \int l(\theta; X, y) p(y|X, \theta_{old})\, dy \end{aligned}$$

  trong đó $p(\cdot|X, \theta_{old})$ là xác suất có điều kiện của Y đối với dữ liệu được quan sát, X, và giả sử rằng $\theta = \theta_{old}$.

- Bước M: Bước M bao gồm việc tối ưu θ bằng kỳ vọng được tính trong bước 1. Đó là, đặt

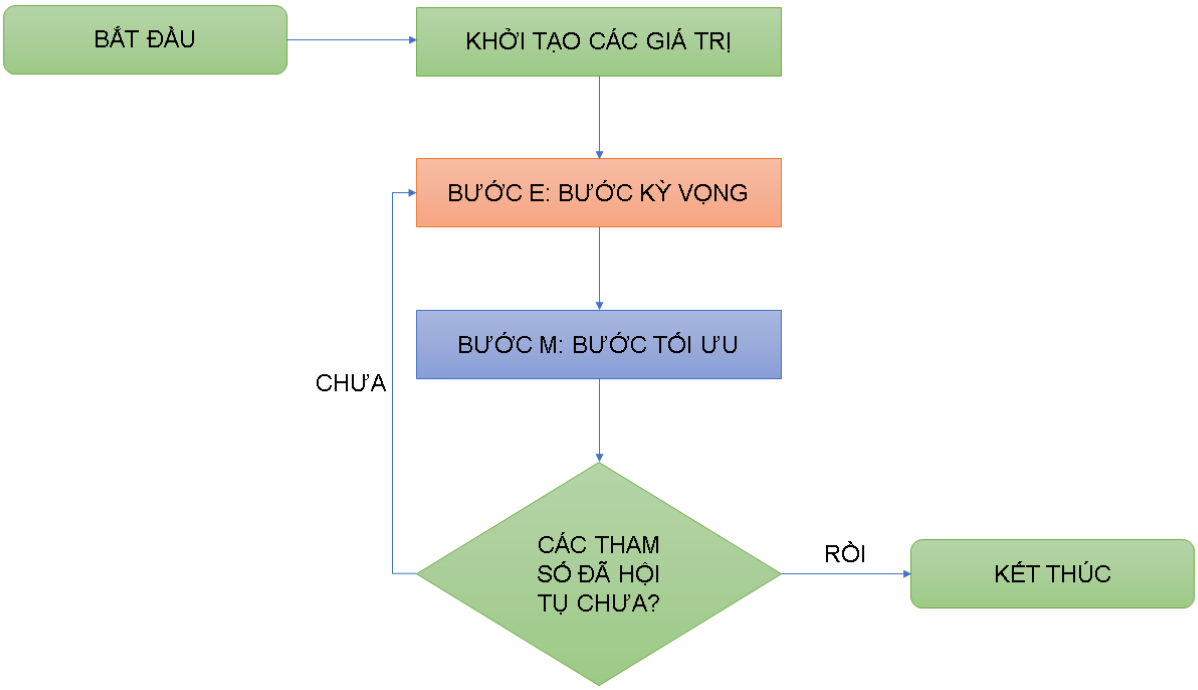
$$\theta_{new} := \max_{\theta} Q(\theta; \theta_{old})$$

Sau đó, chúng ta đặt $\theta_{old} = \theta_{new}$.

Hai bước trên được lặp lại nếu cần thiết cho đến khi chuỗi θ_{new} hội tụ.

Quy trình chi tiết của thuật toán EM cùng flowchart tinh giản sẽ được nêu cụ thể dưới đây.

1. Giả sử, chúng ta có một bộ dữ liệu X không đầy đủ, trong đó có một số dữ liệu không quan sát, thu thập được (hay gọi là tiềm ẩn).
2. Ban đầu, chúng ta có một mô hình với các tham số được khởi tạo sẵn.
3. Bước tiếp theo được gọi là "Kỳ vọng" - bước E. Ở bước này, chúng ta sử dụng dữ liệu đã quan sát để ước tính hoặc dự đoán các giá trị của những dữ liệu nào bị thiếu hoặc không đầy đủ. Tóm gọn thì đây là quá trình điền dữ liệu ẩn dựa vào tham số ban đầu.
4. Bước kế tiếp được gọi là "Tối đa hóa" - bước M. Trong bước này, chúng ta sử dụng bộ dữ liệu đầy đủ được tạo trong bước "Kỳ vọng" trước đó - để cập nhật giá trị của các tham số. Về cơ bản, đây là quá trình cập nhật giả thuyết cho mô hình.
5. Bước cuối cùng, chúng ta kiểm tra xem các giá trị có hội tụ hay không, nếu có, hãy dừng lại, nếu không, hãy lặp lại bước 3 và bước 4, tức là "Kỳ vọng" - bước và "Tối ưu" - bước cho đến khi sự hội tụ xảy ra.



Mô tả chi tiết của thuật toán EM có thể tham khảo tại đây. [2]

5.2. Mô tả và đánh giá độ phức tạp của bước E và bước M của thuật toán Baum-Welch.

Thuật toán **Baum-Welch** hay còn gọi là thuật toán **Forward-Backward** sẽ giúp chúng ta huấn luyện hai ma trận gồm ma trận xác suất chuyển đổi trạng thái A và ma trận xác suất bộc phát của trạng thái ẩn B của mô hình Markov ẩn.

Đầu vào cho thuật toán này sẽ là một chuỗi quan sát O chưa được gắn nhãn (nhãn chuỗi trạng thái ẩn tương ứng) và một tập các trạng thái ẩn khả thi Q của bài toán. Vì vậy, nếu như đối với bài toán ở Câu 3, chúng ta sẽ bắt đầu với một chuỗi các quan sát $O = \{1, 0, 1\}$ và tập hợp các các trạng thái ẩn $Q = \{T, G\}$.

Thuật toán Baum-Welch là một thuật toán có tính lặp, chúng ta sẽ tính toán và ước lượng ban đầu cho các xác suất (hai tham số A và B kể trên), sau đó sử dụng những ước tính đó để tính toán một ước tính tốt hơn, v.v., lặp đi lặp lại việc cải thiện xác suất mà nó học được. Chúng ta sẽ thực hiện bằng việc tính toán xác suất chuyển đổi của một dấu hiệu quan sát được (tương tự forward algorithm: xác suất từ trạng thái ẩn bộc phát thành dấu hiệu quan sát được) và sau đó, chia xác suất đó cho tổng tất cả các chuỗi khác nhau mà có chứa sự chuyển đổi trên của dấu hiệu quan sát được.

Để hiểu được thuật toán Baum-Welch, chúng ta cần biết thêm về một thuật toán là thuật toán quay lui (backward algorithm). Xác suất quay lui (backward probability) β (hay là $b_{t=1:T}$(trạng thái ẩn) trong ví dụ dưới đây) là xác suất chúng ta quan sát thấy dấu hiệu quan sát được (observations) trong khoảng từ thời điểm $t + 1$ đến khi kết thúc chuỗi, tức là $t = T$, giả sử răng mô hình đang ở trạng thái i vào thời điểm t và λ là mô hình tự động này.

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

Quy trình tính toán này cũng theo cách tương tự như thuật toán tiến tới (forward algorithm).

1. Khởi tạo:

$$b_T(i) = 1,\ 1 \leq i \leq N$$

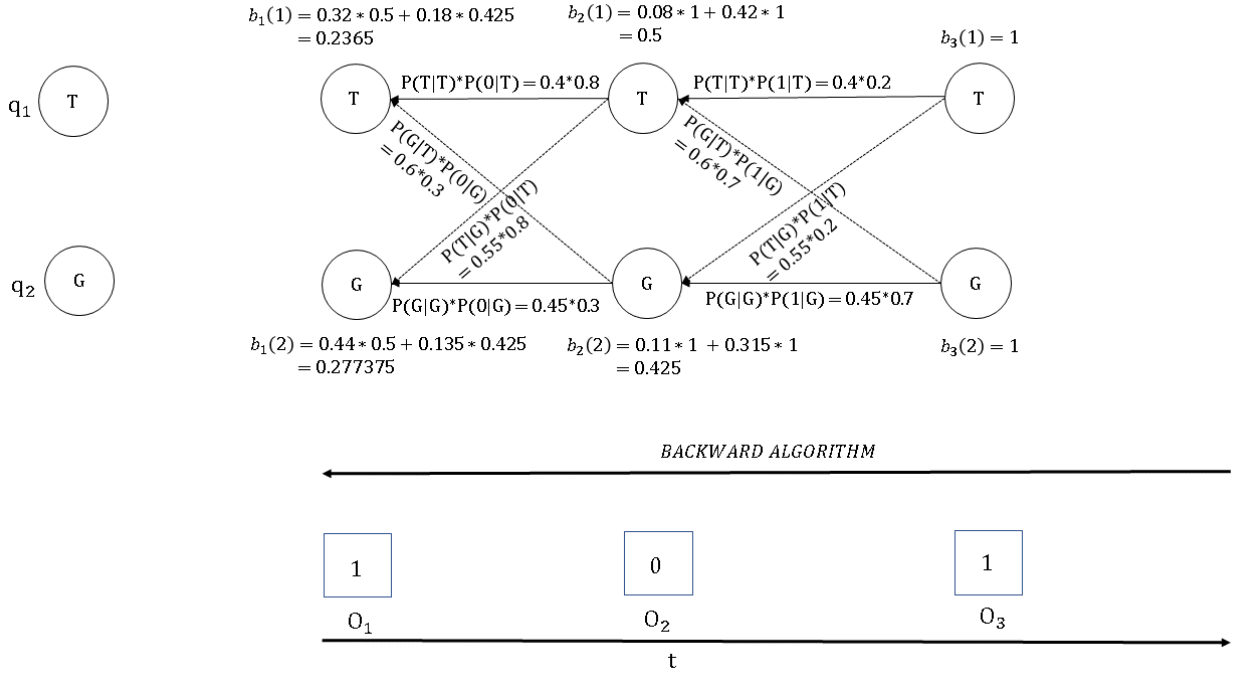
2. Đệ quy:

$$b_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) b_{t+1}(j),\ 1 \leq i \leq N, 1 \leq t < T$$

3. Kết thúc:

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j b_j(o_1) b_1(j)$$

Cụ thể hơn, ta tiếp tục xét bài toán ở Câu 2, quy trình của thuật toán quay lui được diễn tả qua hình



Như vậy,

$$P(O|\lambda) = 0.4 \times 0.2 \times 0.2365 + 0.6 \times 0.7 \times 0.277375 = 0.1354175$$

Giờ chúng ta đã biết được xác suất tiến tới (forward algorithm) và quay lùi (backward algorithm). Tiếp theo, chúng ta sẽ tìm hiểu các thuật toán trên có thể giúp ích như thế nào tính toán ma trận xác suất chuyển đổi trạng thái ẩn a_{ij} và ma trận xác suất bộc phát dấu hiệu quan sát được $b_i(o_t)$ từ một chuỗi quan sát.

Hãy bắt đầu bằng cách ước tính $\hat{a}_{ij}$ bằng một loạt ước tính tối đa hóa khả năng xảy ra (MLE - Maximum likelihood estimation).

$$\hat{a}_{ij} = \frac{\text{Số lượng chuyển đổi trạng thái kỳ vọng từ trạng thái } i \text{ sang trạng thái } j}{\text{Số lượng chuyển đổi trạng thái kỳ vọng từ trạng thái } i \text{ đến các trạng thái khác khả thi}}$$

Giả sử, chúng ta có được ước tính xác suất chuyển đổi trạng thái ẩn $i \to j$ tại một tại thời điểm t trong chuỗi quan sát. Nếu chúng ta biết được xác suất này cho mỗi thời gian t cụ thể, thì chúng ta tính được tổng xác suất tại các thời điểm t để ước tính tổng số lần chuyển đổi trạng thái ẩn $i \to j$.

Đặt ξ_t là xác suất ở trạng thái i tại thời điểm t và trạng thái j tại thời điểm $t+1$, cùng chuỗi quan sát O và mô hình λ.

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

Để tính toán ξ_t, chúng ta cần tính một biến xác suất tương tự ξ_t nhưng khác điều kiện về biến quan sát O.

$$\begin{aligned}\xi_t^{'}(i,j) &= P(q_t = i, q_{t+1} = j, O|\lambda) \\ &= a_t(i) \times a_{ij} b_j(o_{t+1}) \times b_{t+1}(j)\end{aligned}$$

trong đó,

- $a_t(i)$ là xác suất tiến tới (forward probability) tại thời điểm t của trạng thái i.
- $b_{t+1}(j)$ là xác suất quay lui (backward probability) tại thời điểm $t+1$ của trạng thái j.
- $a_{ij}b_j(o_{t+1})$ là tích của xác suất chuyển đổi trạng thái ẩn $i \to j$ và xác suất bộc phát dấu hiệu quan sát được tại thời điểm $t+1$ là o_{t+1} của trạng thái ẩn j.

Như vậy, công thức để tính ξ_t là

$$\begin{aligned}\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) &= \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)} \\ &= \frac{a_t(i) \times a_{ij} b_j(o_{t+1}) \times b_{t+1}(j)}{\sum_{j=1}^{N} a_t(j) b_t(j)}\end{aligned}$$

Sau cùng, để tính được $\hat{a}_{ij}$ chúng ta cần tính toán thêm số lượng chuyển đổi trạng thái kỳ vọng từ trạng thái ẩn i bằng việc tính tương tự như $\xi_t(i,j)$.

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

Tiếp theo, chúng ta cần tính toán lại ma trận xác suất bộc phát dấu hiệu quan sát được của các trạng thái ẩn. Đặt $\hat{b}_j(v_k)$ là xác suất của trạng thái ẩn j bộc phát ra dấu hiệu v_k.

$$\hat{b}_j(v_k) = \frac{\text{Số lần mô hình ở trạng thái ẩn } j \text{ bộc phát dấu hiệu } v_k \text{ kỳ vọng}}{\text{Số lần mô hình ở trạng thái } j \text{ kỳ vọng}}$$

Đầu tiên, chúng ta sẽ cần biết được xác suất $\gamma_t(j)$ để mô hình ở trạng thái ẩn j tại thời điểm t.

$$\begin{aligned}\gamma_t(j) = P(q_t = j|O,\lambda) &= \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \\ &= \frac{a_t(j) b_t(j)}{\sum_{i=1}^{N} a_t(i) b_t(i)}\end{aligned}$$

trong đó,

- $a_t(j)$ là xác suất tiến tới (forward probability) tại thời điểm t của trạng thái j.
- $b_t(j)$ là xác suất quay lui (backward probability) tại thời điểm t của trạng thái j.

Như vậy, để tính được $\hat{b}_j(v_k)$ chúng ta sẽ thực hiện

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 \text{ khi } o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Cuối cùng, chúng ta đã có công thức để ước tính lại ma trận xác suất chuyển đổi A và ma trận xác suất bộc phát B. Thuật toán Baum-Welch bắt đầu bằng việc khởi tạo tham số cho mô hình Markov ẩn $\lambda = (A, B)$, sau đó liên tục lặp lại hai bước: bước Kỳ vọng và bước Tối ưu. [3]

1. Bước Kỳ vọng, chúng ta sẽ tính ξ và γ từ A và B được khởi tạo lúc đầu.
2. Bước Tối ưu, chúng ta sử dụng ξ và γ để ước lượng lại A và B.

Input, output và mã giả của thuật toán Baum-Welch.

- Input: Ma trận xác suất chuyển đổi trạng thái A, ma trận xác suất bộc phát dấu hiệu B, xác suất khởi đầu π, chuỗi T dấu hiệu quan sát được, M dấu hiệu có thể quan sát, N trạng thái ẩn của mô hình và số lần lặp đến khi hội tụ num_iter.
- Output: Mô hình Markov ẩn $\lambda = (A, B)$

```
function BAUM_WELCH(T observations, observations vocabulary V, hidden states Q)
returns HMM=(A,B)
    Input A and B or initialize A and B
    Loop until convergence or num_iter reached
        E-step: Calculates xi and gamma
        M-step: Estimates A and B
        Check convergence using A, B estimated and A', B' from previous loop,
            if TRUE then break
    Return A, B
```

Nhóm cũng đã tiến hành cài đặt thuật toán Baum-Welch cho mô hình Markov ẩn bằng ngôn ngữ Python.

```python
def forward_backward(O, A, B, pi):
  N = A.shape[0] # So trang thai
  T = O.shape[0] # So du lieu quan sat duoc

  # Tinh ma tran alpha
  alpha = np.zeros((T, N))
  alpha[0, :] = pi[:] * B [:, O[0]]
  for t in range(1, T):
    for j in range(N):
```

```
      alpha[t, j] = np.dot(alpha[t - 1], A[:, j]) * B[j, O[t]]

  # Tinh ma tran beta
  beta = np.zeros((T, N))
  beta[-1] = np.ones(N)
  for t in range(T - 2, -1, -1):
    for j in range(N):
      beta[t, j] = np.dot(beta[t + 1] * B[:, O[t + 1]], A[j, :])

  return alpha, beta

def baum_welch(O, A, B, pi, num_iter):
  MIN_DIFF = 1e-5 # Threshold kiem tra hoi tu

  A = A.copy() # Tao mot ban sao cho ma tran A tranh thay doi gia tri sau khi
    ket thuc ham
  B = B.copy() # Tao mot ban sao cho ma tran B tranh thay doi gia tri sau khi
    ket thuc ham

  N = A.shape[0] # So trang thai
  T = O.shape[0] # So du lieu quan sat duoc
  M = B.shape[1] # So dau hieu

  for iter in range(num_iter):
    old_A = A.copy() # Luu lai ma tran A truoc khi cap nhat
    old_B = B.copy() # Luu lai ma tran B truoc khi cap nhat
    # --> 2 ma tran nay dung de kiem tra HMM da hoi tu chua

    # Tinh ma tran alpha, beta
    alpha, beta = forward_backward(O, A, B, pi)
    if (np.sum(alpha == 0) > 0 or np.sum(beta == 0) > 0):
      break

    # E-step
    # Tinh ma tran xi
    xi = np.zeros((N, N, T - 1))
    for t in range(T - 1):
      denominator = np.dot(alpha[t].T, beta[t])
      for i in range(N):
        numerator = alpha[t, i] * A[i, :] * B[:, O[t + 1]] * beta[t + 1]
        xi[i, :, t] = numerator / denominator

    # Tinh ma tran gamma
    mult_ab = alpha * beta
    gamma = mult_ab / np.sum(mult_ab, axis = 1).reshape((T, 1))

    # M-step
    # Cap nhat ma tran A
    numerator_A = np.sum(xi, axis = 2)
    denominator_A = np.sum(numerator_A, axis = 1).reshape((N, 1))
    A = numerator_A / denominator_A

    # Cap nhat ma tran B
    denominator_B = np.sum(gamma, axis = 0)
    for i in range(M):
      B[:, i] = np.sum(gamma[O == i, :], axis = 0) / denominator_B

    # Kiem tra hoi tu dua tren threshold MIN_DIFF
```

```
66      if (np.linalg.norm(A - old_A) < MIN_DIFF and np.linalg.norm(B - old_B) <
        MIN_DIFF):
67          break
68
69  return A, B
```

Đánh giá độ phức tạp của bước E và bước M của thuật toán Baum-Welch (Baum-Welch algorithm).

1. Bước E:

   Ta có thao tác cơ sở ở E-step như sau:

   ```
   1      numerator = alpha[t, i] * A[i, :] * B[:, O[t + 1]] * beta[t + 1]
   2      xi[i, :, t] = numerator / denominator
   3
   ```

   Phép tính A[i, :] * B[:, O[t + 1]] và dòng lệnh xi[i, :, t] = numerator / denominator có chi phí là N. Chi phí của hai dòng for là $N \times T$.

   $\Rightarrow$ Chi phí của hai thao tác cơ sở là $2 \times N \times N \times T$.

   $\Rightarrow$ Độ phức tạp của bước E là $O(2N^2T) = O(N^2T)$.

2. Bước M:

   Ta có thao tác cơ sở ở M-step như sau:

   ```
   1      B[:, i] = np.sum(gamma[O == i, :], axis = 0) / denominator_B
   2
   ```

   Lệnh np.sum(gamma[O == i, :], axis = 0) có chi phí $N \times T$. Chi phí của một dòng for là M với M là tất cả dấu hiệu có thể quan sát.

   $\Rightarrow$ Chi phí của thao tác cơ sở là $N \times T \times M$.

   $\Rightarrow$ Độ phức tạp của bước M là $O(NTM)$.

Tài liệu tham khảo

[1] Jun Xie. *Markov chains and hidden Markov models*. URL: https://www.stat.purdue.edu/~junxie/topic3.pdf.

[2] Ravi Charan. *Expectation Maximization Explained*. URL: https://towardsdatascience.com/expectation-maximization-explained-c82f5ed438e5.

[3] Daniel Jurafsky and James H. Martin. *Hidden Markov Models*. URL: https://web.stanford.edu/~jurafsky/slp3/A.pdf.