Modele Markov Ascunse

De la Teorie la Aplicații

Ghid pentru partea practică

Alexandru Sorici, Tudor Berariu

Asociația Română pentru Inteligență Artificială

în colaborare cu

Laboratorul AI-MAS

Cuprins

1 Mediul de lucru **4**
1.1 Inițializarea mediului de lucru 4
1.2 Fișierele `.stub` 4
1.3 Testerul 4

2 Notații și Denumirile Variabilelor **5**
2.1 Notații folosite în slide-uri și pseudocod 5
2.1.1 Notații generale 5
2.1.2 Algoritmul Forward-Backward 6
2.1.3 Algoritmul Viterbi 7
2.1.4 Algoritmul Baum-Welch 8
2.2 Variabile în fișierele `.m` 8

3 Task-uri de implementare **9**
3.1 Algoritmul Forward-Backward 9
3.1.1 Descriere 9
3.1.2 Teste automate 10
3.1.3 Pseudocod 11
3.2 Algoritmul Viterbi 12
3.2.1 Descriere 12
3.2.2 Testare 13
3.2.3 Pseudocod 13
3.3 Algoritmul Baum-Welch 13
3.3.1 Descriere 13
3.3.2 Testare 14
3.3.3 Pseudocod 15
3.4 Precalcularea Matricei B în cazul mai multor variabile de observație 17
3.4.1 Descriere 17
3.4.2 Testare automată 18
3.4.3 Pseudocod 18
3.5 Recunoașterea Simbolurilor 18
3.5.1 Descriere 18
3.5.2 Testare automată 20
3.6 Aplicația de Recunoaștere a Simbolurilor 20
3.6.1 Descriere 20

4 Soluții **22**
4.1 Algoritmul Forward-Backward 22
4.2 Algoritmul Viterbi 23
4.3 Algoritmul Baum-Welch 24
4.4 Recunoașterea Simbolurilor 25
4.4.1 Precalcularea matricei B 25
4.4.2 Clasificarea (recunoașterea) unui simbol 26

1 Mediul de lucru

1.1 Inițializarea mediului de lucru

1. Deschideți `Matlab` / `Octave`
2. Schimbați directorul de lucru cu `aria-hmm`:
   `cd("[.../]aria-hmm")`
3. Adăugați toate subdirectoarele în path:
   `addpath(genpath('.'))`

1.2 Fișierele `.stub`

- le veți folosi ca schelet de cod pentru sesiunile de implementare
- au secțiuni delimitate de `<label>-start` și `<label>-end` între care veți adăuga liniile de cod
- înlăturați sufixul `.stub` când rezolvați task-urile.

```
1  N = size(Pi, 2); % The number of states
2  T = size(O, 2); % The number of observations
3
4  Scale = zeros (1, T); % Scale is an 1 x T matrix
5  Alpha = zeros (T, N); % Alpha is a T x N matrix
6  Beta = ones (T, N); % Beta is a T x N matrix
7
8  %% Forward variables
9  % alpha_disc-start - Write code below
10
11 % alpha_disc-end - Write code above
```

1.3 Testerul

- Testerul se rulează folosind comanda `hmm_test`.

```
1 Please choose a test to run!
2 Type "list" to list all available tests.
```

```
3 Type "quit" to exit this tester.
4 Test:
```

- Cu `list` afișați toate testele disponibile.
- Pentru un anumit task numele testului coincide cu eticheta care delimitează secțiunea de completat.

2 Notații și Denumirile Variabilelor

2.1 Notații folosite în slide-uri și pseudocod

2.1.1 Notații generale

N - numărul de stări ascunse

S - mulțimea stărilor ascunse

- $S = \{s_1, s_2, \dots \ s_N\}$

A - matricea distribuțiilor de probabilitate ale tranzițiilor între stări

- $A = \{a_{i,j}\}, \quad 1 \leq i \leq N, 1 \leq j \leq N$
- $a_{i,j} = P(q_{t+1} = s_j | q_t = s_i)$
- fiecare linie este o distribuție de probabilitate:
  $\sum_{j=1}^{N} a_{i,j} = 1, \quad 1 \leq i \leq N$

$\mathbf{\Pi}$ - distribuția stării inițiale

- $\mathbf{\Pi} = \{\pi_i\}, \quad 1 \leq i \leq N$
- $\pi_i = P(q_1 = s_i)$
- $\sum_{i=1}^{N} \pi_i = 1$

M - numărul de valori observabile distincte (pentru cazul discret)

V - mulțimea valorilor observabile

B - matricea distribuțiilor de probabilitate ale valorilor observabile

- $B = \{b_{j,k}\}, \quad 1 \leq j \leq N, 1 \leq k \leq M$
- $\sum_{k=1}^{M} b_{j,k} = 1, \quad 1 \leq j \leq N$

λ - parametrii Modelului Markov Ascuns

- $\lambda = (A, B, \Pi)$

Q - o secvență de stări

O - o secvență de observații

T - lungimea unei secvențe de stări / valori observate

2.1.2 Algoritmul Forward-Backward

α - variabilele α (înainte)

- $\alpha_{t,i} = P(o_1, o_2, \ldots, o_t, q_t = s_i|\lambda), \quad {\scriptstyle 1 \leq t \leq T, 1 \leq i \leq N}$
- $P(O|\lambda) = \sum_{i=1}^{N} \alpha_{T,i}$
- Calcul:

  t = 1: $\alpha_{1,i} = \pi_i b_i(o_1), \quad {\scriptstyle 1 \leq i \leq N}$

  t > 1: $\alpha_{t+1,j} = \Big[\sum_{i=1}^{N} \alpha_{t,j} a_{i,j}\Big] b_j(o_{t+1}), \quad {\scriptstyle \substack{1 \leq t \leq T-1 \\ 1 \leq j \leq N}}$

β - variabilele β (înapoi)

- $\beta_{t,i} = P(o_{t+1} o_{t+2} \cdots o_T | q_t = s_i, \lambda)$
- $P(O|\lambda) = \sum_{i=1}^{N} \beta_{1,i}$
- Calcul:

  t = T: $\beta_{T,i} = 1, \quad {\scriptstyle 1 \leq i \leq N}$

$$\mathbf{t < T}\ \ \beta_{t,i} = \sum_{j=1}^{N} a_{i,j} b_j(o_{t+1}) \beta_{t+1,j}, \quad {\scriptstyle t=T-1,T-2,\ldots,1, 1 \le i \le N}$$

$\hat{\alpha}$ - variabilele α scalate

$\hat{\beta}$ - variabilele β scalate

c_t - coeficientul de scalare pentru momentul de timp t

- Notație: $C_t = c_1 \cdot c_2 \cdot \ldots \cdot c_t$
- $\hat{\alpha}_{t,i} = C_t \alpha_{t,i}$

2.1.3 Algoritmul Viterbi

δ - variabilele δ

- $\delta_{t,i}$ - cea mai mare probabilitate a unei secvențe de stări de lungime t care ajunge în s_i și explică primele t valori observate
- $\delta_{t,i} = \max\limits_{q_1,\ldots,q_{t-1}} P([q_1 q_2 \ldots q_{t-1} s_i], [o_1, o_2, \ldots o_t] | \lambda)$
- Calcul:

  $\mathbf{t = 1:}\ \ \delta_{1,i} = \pi_i b_i(o_1), \quad {\scriptstyle 1 \le i \le N}$

  $\mathbf{t > 1:}\ \ \delta_{t,j} = [\max\limits_{i} \delta_{t-1,i} \cdot a_{i,j}] \cdot b_j(o_t) \quad {\scriptstyle 2 \le t \le T, 1 \le j \le N}$

ψ - variabilele ψ

- $\psi_{t,i}$ - starea de la $t-1$ care a dus la valoarea maximă $\delta_{t-1,i} \cdot a_{i,j}$
- Calcul:

  $\mathbf{t = 1:}\ \ \psi_{1,i} = 0, \quad {\scriptstyle 1 \le i \le N}$

  $\mathbf{t > 1:}\ \ \psi_{t,i} = \operatorname{argmax}\limits_{i} \delta_{t-1,i} \cdot a_{i,j} \quad {\scriptstyle 2 \le t \le T, 1 \le j \le N}$

ϕ - variabilele ϕ (δ logaritmate)

- $\phi_{t,i} = \max\limits_{q_1,\ldots,q_{t-1}} \log(P(q_1, \ldots, q_{t-1}, q_t = s_i, o_1, \ldots, o_t | \lambda)) = \log(\delta_{t,i})$
- Calcul:

  $t = 1\mathbf{:}\ \ \phi_{1,i} = \log(\pi_i) + \log(b_i(o_1)), \quad {\scriptstyle 1 \le i \le N}$

  $t > 1\mathbf{:}\ \ \phi_{t,j} = [\max\limits_{i} \phi_{t-1,i} + log(a_{i,j})] + \log(b_j(o_t)) \quad {\scriptstyle 2 \le t \le T, 1 \le j \le N}$

2.1.4 Algoritmul Baum-Welch

ξ - variabilele ξ

- $\xi_{t,i,j} = \xi_t(i,j) = P(q_t = s_i, q_{t+1} = s_j | O, \lambda)$
- $\xi_{t,i,j} = \frac{\alpha_{t,i} \cdot a_{i,j} \cdot b_j(o_{t+1}) \cdot \beta_{t+1,j}}{\sum_{k=1}^{N} \sum_{l=1}^{N} \alpha_{t,k} \cdot a_{k,l} \cdot b_l(o_{t+1}) \cdot \beta_{t+1,l}}$

γ - variabilele γ

- $\gamma_{t,i} = \gamma_t(i) = P(q_t = s_i | O, \lambda)$
- $\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$

2.2 Variabile în fișierele `.m`

- General:
  - `N` - scalar
  - `M` - scalar
  - `T` - scalar
  - `A` - matrice de dimensiune $N \times N$
  - `B` - matrice de dimensiune $N \times M$
  - `Pi` - matrice de dimensiune $1 \times N$
  - `O` - matrice de dimensiune $1 \times T$
- Forward-Backward:
  - `Alpha` - matrice de dimensiune $T \times N$
    - va reprezenta, de fapt, variabilele $\hat{\alpha}$
  - `Beta` - matrice de dimensiune $T \times N$
    - va reprezenta, de fapt, variabilele $\hat{\beta}$
  - `Scale` - matrice de dimesniune $1 \times T$
    - va reprezenta coeficienții de scalare c_i

  - `logP` - scalar

- Viterbi:
  - `Phi` - matrice de dimensiune $T \times N$
  - `Psi` - matrice de dimensiune $T \times N$
  - `Q` - matrice de dimensiune $1 \times T$

- Baum-Welch (avem mai multe observații)
  - `L` - scalar, numărul de observații
  - `TMax` - scalar, lungimea maximă a observațiilor
  - `T` - matrice de dimensiune $1 \times L$, lungimea observației l
  - `Alpha` - matrice de dimensiune $L \times TMax \times N$
  - `Beta` - matrice de dimensiune $L \times TMax \times N$
  - `logP` - matrice de dimensiune $1 \times L$

3 Task-uri de implementare

3.1 Algoritmul Forward-Backward

3.1.1 Descriere

Primul task de programare constă în calcularea valorilor matricelor `Alpha`, `Beta` și a valorii `logP`, date fiind o secvență de observații `O` și parametrii modelului: matricele `A`, `B` și `Pi`.

Scheletul de cod de la care veți pleca se află în fișierul **`hmm/forward_backward_disc.m.stub`**. Eliminați sufixul **`.stub`** și salvați în fișierul **`hmm/forward_backward_disc.m`**.

Funcția pe care o veți completa este `forward_backward_disc`:

```
1 function [logP, Alpha, Beta, Scale] = ...
      forward_backward_disc(O, Pi, A, B)
```

Pentru rezolvare veți completa trei secțiuni. Pentru calculul valorilor matricei `Alpha` veți completa în zona delimitată de etichetele `alpha_disc`.

```
%% Forward variables
% Compute Alpha and Scale
% alpha_disc—start — Write code below

% alpha_disc—end — Write code above
```

Pentru calculul valorilor matricei `Beta` veți completa în zona delimitată de etichetele `beta_disc`.

```
%% Backward variables
% Compute Beta
% beta_disc—start — Write code below

% beta_disc—end — Write code above
```

Pentru calculul valorii `logP` veți completa în zona delimitată de etichetele `prob_disc`.

```
%% The probability of the observed sequence
% Compute logP
% prob_disc—start — Write code below

% prob_disc—end — Write code above
```

3.1.2 Teste automate

Pentru a testa codul folosiți comanda:

- `hmm_test("alpha_disc");` pentru testarea valorilor matricelor `Alpha` și `Scale`
- `hmm_test("beta_disc");` pentru testarea valorilor matricei `Beta`
- `hmm_test("prob_disc");` pentru testarea valorilor matricei `logP`

Indicați apoi numele fișierului (sau tastați simplu ENTER dacă ați folosit numele sugerat).

3.1.3 Pseudocod

Algoritmul 1 Calculul variabilelor α

for $i = 1$ to N **do**
    $\ddot{\alpha}_{1,i} \leftarrow \pi_i \cdot b_i(o_1)$
end for
$c_1 \leftarrow (\sum_{i=1}^{N} \ddot{\alpha}_{1,i})^{-1}$
for $i = 1$ to N **do**
    $\hat{\alpha}_{1,i} \leftarrow c_1 \cdot \ddot{\alpha}_{1,i}$
end for
for $t = 1$ to $T - 1$ **do**
    for $j = 1$ to N **do**
        $\ddot{\alpha}_{t+1,j} \leftarrow \Big[\sum_{i=1}^{N} \hat{\alpha}_{t,i} a_{i,j}\Big] b_j(o_{t+1})$
    end for
    $c_{t+1} \leftarrow (\sum_{i=1}^{N} \ddot{\alpha}_{t+1,i})^{-1}$
    for $i = 1$ to N **do**
        $\hat{\alpha}_{t+1,i} \leftarrow c_{t+1} \cdot \ddot{\alpha}_{t+1,i}$
    end for
end for

Algoritmul 2 Calculul $P(O|\lambda)$

$logP \leftarrow -\sum_{t=1}^{T} \log c_t$

Algoritmul 3 Calculul variabilelor β

for $i = 1$ to N **do**
    $\hat{\beta}_{T,i} \leftarrow c_T$
end for
for $t = (T-1)$ to 1 **do**
    for $i = 1$ to N **do**
        $$\hat{\beta}_{t,i} \leftarrow \sum_{j=1}^{N} a_{i,j} b_j(o_{t+1}) \hat{\beta}_{t+1,j} \cdot c_t$$
    end for
end for

3.2 Algoritmul Viterbi

3.2.1 Descriere

A doua sarcină de programare vă cere să implementați algoritmul Viterbi, mai precis să calculați valorile ϕ și ψ și cea mai bună secvență Q.

Calculați valorile matricelor `Phi` și `Psi` pentru $t > 1$. Completați cu instrucțiunile necesare în secțiunea delimitată de `phi_psi_disc-start` și `phi_psi_disc-end`

```
Phi(1, :) = log(Pi) + logB(:, O(1))'; % Initialization for ...
    Phi (t = 1)

%% Recursion
% compute Phi and Psi for t=2:T
% phi_psi_disc-start

% phi_psi_disc-end
```

Refaceți cea mai bună cale în vectorul `Q`. Scrieți codul în liniile delimitate de `path-disc-start` și `path-disc-end`.

```
%% Backtracking to compute the path Q
% compute Q
% path_disc-start - Write code below

% path_disc-end - Write code below
```

3.2.2 Testare

Pentru a vă testa codul scris folosiți comanda:

- `hmm_test("phi_psi_disc");` pentru testarea valorilor matricelor `Phi` și `Psi`
- `hmm_test("path_disc");` pentru testarea valorilor vectorului `Q`

Indicați apoi numele fișierului (sau tastați simplu ENTER dacă ați folosit numele sugerat).

3.2.3 Pseudocod

Algoritmul 4 Viterbi: Calculul celei mai probabile secvențe Q_{best}

for $i = 1$ to N **do**
  $\phi_{1,i} \leftarrow \log(\pi_i) + \log(b_i(o_1))$
  $\psi_{1,i} \leftarrow 0$
end for
for $t = 2$ to T **do**
  for $j = 1$ to N **do**
    $\phi_{t,j} \leftarrow \max_i [\phi_{t-1,i} + log(a_{i,j})] + \log(b_j(o_t))$
    $\psi_{t,j} \leftarrow \operatorname{argmax}_i [\phi_{t-1,i} + \log(a_{i,j})]$
  end for
end for
$\log(P(Q_{\text{best}}|O, \lambda)) \leftarrow \max_i \phi_{T,i}$
$q_{T_{\text{best}}} \leftarrow \operatorname{argmax}_i \phi_{T,i}$
for $t = T - 1$ to 1 **do**
  $q_{t_{\text{best}}} \leftarrow \psi_{t+1}(q_{t+1_{\text{best}}})$
end for

3.3 Algoritmul Baum-Welch

3.3.1 Descriere

În această parte practică veți implementa o parte a algoritmului Baum-Welch pentru estimarea parametrilor `A`,`B` și `Pi` pe baza a `L` observații de lungime maximă `Tmax`.

Scheletul de cod de la care veți pleca se află în fișierul **hmm/baum_welch_disc.m.stub**. Eliminați sufixul **.stub** și salvați în fișierul **hmm/baum_welch_disc.m**.

Funcția pe care o veți completa este `baum_welch_disc`:

```
function [Pi, A, B] = baum_welch_disc(O, T, N, M, model, ...
    max_iter)
```

Pasul *expectation* este *rezolvat* de algoritmul Forward-Backward.

```
% Compute initial P (and forward and backward variables)
for l=1:L
    [logP(l), Alpha(l, 1:T(l), :), Beta(l, 1:T(l), :), ...
       Scale(l, 1:T(l))] = ...
        forward_backward_multi_disc(O(l,1:T(l)), Pi, A, B);
end
```

Trebuie să implementați doar reestimarea matricelor `A` și `B` pe baza matricelor `Alpha`, `Beta`, `Scale` și `logP` și vechilor valori din `A`, `B`.

Atenție: matricele `Alpha` și `Beta` sunt de dimensiune `L`×`N`×`N`, matricea `Scale` are dimensiunea `L`×`N`, iar `logP` este un vector de dimensiune 1×`L`.

Scrieți codul în secțiunea următoare:

```
% maximization_disc—start — Write code below

% maximization_disc—end — Write code above
```

3.3.2 Testare

Pentru a testa codul folosiți comanda `hmm_test("maximization_disc");`. Indicați apoi numele fișierului (sau tastați simplu ENTER dacă ați folosit numele sugerat).

3.3.3 Pseudocod

Algoritmul 5 Baum-Welch

intrări: $O \leftarrow$ secvența de observații, $\epsilon \leftarrow$ prag de convergență

{*Initializare*}

init. uniformă Π ($\Pi_i = 1/N, 1 \leq i \leq N$)

init. aleatoare $a_{i,j}$, a. î. $\sum_{j=1}^{N} a_{i,j} = 1, \quad 1 \leq j \leq N$

init. uniformă $b_{j,k}$ ($b_{j,k} = 1/M, \quad 1 \leq j \leq N, 1 \leq k \leq M$)

$oldP \leftarrow 0$

{*E STEP - în afara buclei*}

for $l = 1$ to L **do**

    $[logP_l, \hat{\alpha}_l, \hat{\beta}_l, Scale_l] = forward\_backward(O_l, \Pi, A, B)$

end for

$$logP \leftarrow \sum_{l=1}^{L} logP(l)$$

while $|logP - oldP| < \epsilon$ **do**

    $oldP \leftarrow logP$

    {M STEP - recalculeaza Π, A și B}

    $\Pi = update\_pi\_procedure(\hat{\alpha}, \hat{\beta}, Scale)$

    $A = update\_A\_procedure(O, \hat{\alpha}, \hat{\beta}, Scale)$

    $B = update\_B\_procedure(O, \hat{\alpha}, \hat{\beta}, Scale)$

    {*E STEP - calculeaza variantele scalate pentru α și β și probabilitatea curentă (log likelihood - $log(P(O|\bar{\lambda}))$) a secvenței observate*}

    for $l = 1$ to L **do**

        $[logP_l, \hat{\alpha}_l, \hat{\beta}_l, Scale_l] = forward\_backward(O_l, \Pi, A, B)$

    end for

    $$logP \leftarrow \sum_{l=1}^{L} logP(l)$$

end while

Algoritmul 6 Baum-Welch

Function update_pi_procedure($\hat{\alpha}$, $\hat{\beta}$, $Scale$)
for $i = 1$ to N **do**

$$\Pi_i = \frac{\sum_{l=1}^{L} \hat{\alpha}_{l,1,i} \cdot \hat{\beta}_{l,1}(i)/Scale_1}{\sum_{l=1}^{L}\sum_{j=1}^{N} \hat{\alpha}_{l,1}(j) \cdot \hat{\beta}_{l,1}(j)/Scale_1}$$

end for
return Π
EndFunction

Function update_A_procedure(O, $\hat{\alpha}$, $\hat{\beta}$, $Scale$)
for $i = 1$ to N **do**
    for $j = 1$ to N **do**

$$a_{i,j} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T_l-1} \hat{\alpha}_{l,t,i} \cdot a_{ij} \cdot b_{l,j}(o_{l,t+1}) \cdot \hat{\beta}_{l,t+1,j}}{\sum_{l=1}^{L}\sum_{t=1}^{T_l-1}\sum_{j=1}^{N} \hat{\alpha}_{l,t,i} \cdot a_{i,j} \cdot b_{l,j}(o_{l,t+1}) \cdot \hat{\beta}_{l,t+1,j}}$$

    end for
end for
return a
EndFunction

Function update_B_procedure(O, $\hat{\alpha}$, $\hat{\beta}$, $Scale$)
for $j = 1$ to N **do**
    for $k = 1$ to M **do**

$$b_{j,k} = \frac{\sum_{l=1}^{L}\sum_{t=1,O(t)=v_k}^{T(l)} \hat{\alpha}_{l,t,j} \cdot \hat{\beta}_{l,t,j}/Scale_{l,t}}{\sum_{l=1}^{L}\sum_{t=1}^{T(l)} \hat{\alpha}_{l,t,j} \cdot \hat{\beta}_{l,t,j}/Scale_{l,t}}$$

    end for
end for
return b
EndFunction

3.4 Precalcularea Matricei B în cazul mai multor variabile de observație

3.4.1 Descriere

În aplicația de recunoaștere a simbolurilor dezvoltată, în fiecare stare a unui MMA antrenat pentru un anumit simbol întâlnim două variabile de observație:

- coeficienții transformatei Fourier pentru semnalul (mișcarea mouse-ului) pe axa Ox
- coeficienții transformatei Fourier pentru semnalul (mișcarea mouse-ului) pe axa Oy

În framework-ul prezent aceasta se traduce prin faptul că matricea B a probabilităților de emisie mai primeste o dimensiune, a.î. ea trece de la $N \times M$ în cazul normal la $N \times M \times R$ pe cazul multi-variabile. Practic, noua matrice B reține R(numărul de variabile observate în fiecare stare) matrici de dimensiune $N \times M$.
Pentru a simplifica puțin lucrurile, se face presupunerea că cele două variabilele sunt independente una de alta (i.e. în fiecare stare, presupunem ca mișcarea pe Ox este independentă de mișcarea pe Oy). Ținând cont de aceasta, în algoritmul Baum-Welch, variabila auxiliara $\xi_t(i, j)$ se rescrie astfel:

$$\xi_t(i,j) = \frac{\alpha_{t,i} \cdot a_{i,j} \cdot \prod_{r=1}^{R} b_{j,r}(o_{t+1}(r)) \cdot \beta_{t+1,j}}{P(O|\lambda)}$$

Observați în formula de mai sus produsul $\prod_{r=1}^{R} b_{j,r}(o_{t+1}(r))$ al probabilităților de a observa valorile $o_{t+1}(r)$ în starea j.
Sarcina voastră în acest exercițiu de programare este de a face o precalculare a acestor produse pentru o secvență observată O. Astfel, pentru o secvență O de dimensiune $R \times T$ și o matrice de emisie B de dimensiune $N \times M \times R$ veți obține o matrice B_{prod} de dimensiune $N \times T$.
Funcția pe care o veți completa este `baum_welch_multi_disc` și trebuie să scrieți codul în secțiunea următoare:

```
1 % precomp_b_disc-start - Write code below
2
3 % precomp_b_disc-end - Write code above
```

3.4.2 Testare automată

Pentru a testa codul folosiți comanda `hmm_test("precomp_b_disc");`. Indicați apoi numele fișierului (sau tastați simplu ENTER dacă ați folosit numele sugerat).

3.4.3 Pseudocod

Algoritmul 7 Precalcularea matricei B în cazul mai multor variabile de observație per stare

for $l = 1$ to L **do**
  for $t = 1$ to $T(l)$ **do**
    for $i = 1$ to N **do**
      $Bprod(l, i, t) \leftarrow \prod_{r=1}^{R} B(i, O(l, r, t), r)$
    end for
  end for
end for

3.5 Recunoașterea Simbolurilor

3.5.1 Descriere

În acest exercițiu veți implementa algoritmul de clasificare a unei secvențe de intrare (coordonate X si Y ale mișcării mouse-ului) intr-unul din simbolurile definite. Aduceți-vă aminte că discriminarea între un simbol sau altul se face pe baza maximului dintre probabilitățile de observare a unei secvenței O date de modelul MMA antrenat pentru fiecare simbol.

Înainte de a prezenta pseudo-codul algoritmului de clasificare vă atragem atenția asupra următoarelor funcții auxiliare:

- `load(filename, name_of_var1, name_of_var2, ...)` – incarcă din fișierul de tip `.mat` cu numele `filename` variabilele denumite prin string-urile `name_of_var1, name_of_var2, ...`.
- `symbol_get_feature_sequence(track_data, x_codebook, y_codebook, ... resample_interval, hamming_window_size, hamming_window_step)`

– întoarce matricea O de observații pe baza prelucrării secvenței de mișcare `track_data` în funcție de valoarea parametrilor de configurare `x_codebook, y_codebook, resample_interval, etc.`

Observații:

- parametrii unui model MMA (Π, A, B) cu tipul de tranziție `transition_model` (*"bakis"* sau *"ergodic"*) și pragul de recunoaștere a unui simbol (`symbol_rec_threshold`) sunt stocate într-un fișier de forma `<numesimbol>_hmm_<transition_model>.mat`
- numele simbolurilor sunt reținute în matricea `symbol_strings`, câte unul pe fiecare linie
- toți parametrii de configurare și datele de intrare necesare acestui task de implementare au valorile gata instanțiate

Procedura de clasificare în pseudocod este următoarea:

Algoritmul 8 Clasificarea unei secvențe de mișcare într-unul din simbolurile existente

for $s = 1$ to $nr\_simboluri$ **do**
  $nume\_simbol \leftarrow simboluri(s)$
  $hmm\_filename \leftarrow$ fișier MMA pentru simbolul $nume\_simbol$
  $(\Pi, A, B) \leftarrow$ parametrii din fișierul hmm_filename

  $O = symbol\_get\_feature\_sequence(track\_data, x\_codebook, y\_codebook, resample\_interval, hamming\_window\_size, hamming\_window\_step)$

  $[Prob, \ , \ , \ ] \leftarrow forward\_backward\_multi\_disc(O, \Pi, A, B)$
  $ll\_vector(s) \leftarrow Prob$
  if $ll\_vector(s) > max\_ll$ **then**
    $max\_ll \leftarrow ll\_vector(s)$
    $simbol\_probabil \leftarrow simboluri(s)$
  end if
end for

$symbol\_rec\_threshold \leftarrow$ încarcă prag recunoaștere pentru $simbol\_probabil$
if $max\_ll > symbol\_rec\_threshold$ **then**
  $simbol\_recunoscut \leftarrow simbol\_probabil$
else
  $simbol\_recunoscut \leftarrow unknown$
end if
return $ll\_vector, simbol\_recunoscut$

3.5.2 Testare automată

Pentru a testa codul folosiți comanda `hmm_test("symbolrec");`. Indicați apoi numele fișierului (sau tastați simplu ENTER dacă ați folosit numele sugerat).

3.6 Aplicația de Recunoaștere a Simbolurilor

3.6.1 Descriere

Acest task își propune să vă arate modul de utilizare al aplicației demo de recunoaștere a simbolurilor. Veți crea un set de date de simboluri, veți

antrena un MMA pentru fiecare, veți vedea performanța de clasificare a modelelor antrenate și veți recunoaște apoi noi simboluri.
Pentru cele de mai sus urmăriți pașii:

1. Crearea unui set de simboluri
   - în consola Matlab curentă tipăriți comanda `symbol_recording` – aceasta lansează GUI-ul pentru crearea unui nou set de simboluri
   - definiți simbolurile `one`, `two`, `three` ce reprezintă cifrele de la 1 la 3
     - în caseta *Define New Symbol* scrieți numele noului simbol și dați *Add*
     - din caseta *Symbol Name* selectați noul simbol și din caseta *Symbol Purpose* selectați `all`
     - în zona de desenare apasați mouse-stânga și prin *drag* dați forma simbolului dorit, eliberând butonul mouse-stânga la finalul conturării
     - urmăriți caset *Messages* pentru eventuale mesaje de eroare sau succes
     - Apăsați *Save Symbol* pentru a salva noul simbol definit sau *Clear Drawing* pentru a anula desenul curent
     - **repetați** pașii de mai sus până definiți 20 de instanțe din fiecare tip de simbol (1, 2 și 3)
2. Antrenarea modelelor MMA pentru simbolurile nou-definite
   - în consola Matlab curentă tipăriți comanda `symbol_training` – aceasta lansează GUI-ul pentru antrenarea unui MMA pentru un set de simboluri
   - selectați din lista de simboluri disponibile (*All Symbols*) cele 3 pe care le-ati definit anterior (`one, two, three`)
   - parametrii de pre-procesare sunt presetați cu valori recomandate
   - din selection box-ul *Dataset Options* alegeți `Percentage`
   - setați valorile procentuale pentru împărțirea setului de date in bucăți folosite pentru antrenare, validare și testare. Valorile predefinite sunt recomandate, dar puteți opta pentru alte valori atât timp cât procentul de date de antrenare este mai mare ca 50

- din caseta *Transition Model Type* selectați `ergodic`
- apasați butonul *Compute Codebook* și așteptați finalizarea calculelor
- apasați butonul *Train* și așteptați finalizarea calculelor
- urmăriți mesajele din *Output Messages* precum și din **consola Matlab**

3. Recunoașterea unui nou simbol

- în consola Matlab curentă tipăriți comanda `symbol_recognition` – aceasta lansează GUI-ul pentru recunoașterea unui nou simbol
- selectați din lista de simboluri disponibile (*All Symbols*) cele 3 pe care le-ati definit anterior (`one, two, three`)
- din caseta *Transition Model Type* selectați `ergodic`
- în spațiul de desenare urmați același procedeu ca la pasul 1
- apăsați *Test* pentru a testa noul simbol sau *Clear* pentru a anula desenul curent
- urmăriți mesajele din *Log Likelihood Estimates* și *Detected Symbols*

4 Soluții

4.1 Algoritmul Forward-Backward

```
1  %% Forward variables
2  % Compute Alpha and Scale
3  % alpha_disc_start - Write code below
4
5  Alpha(1,:) = Pi .* B(:, O(1))';
6  Scale(1) = 1 / sum(Alpha(1, :));
7  Alpha(1,:) = Alpha(1, :) * Scale(1);
8
9  for t = 2:T
10     Alpha(t,:) = (Alpha(t-1,:) * A) .* B(:, O(t))';
11     Scale(t) = 1 / sum(Alpha(t, :));
12     Alpha(t, :) = Alpha(t, :) * Scale(t);
```

```
13 end
14
15 % alpha_disc_end - Write code above
16
17 %% Backward variables
18 % Compute Beta
19 % beta_disc_start - Write code below
20
21 Beta(T, :) = Beta(T, :) * Scale(T);
22 for t = (T-1):-1:1
23     Beta(t,:) = (A * (B(:,O(t+1)) .* Beta(t+1,:)'))';
24     Beta(t, :) = Beta(t, :) * Scale(t);
25 end
26
27 % beta_disc_end - Write code above
28
29 %% The probability of the observed sequence
30 % Compute logP
31 % prob_disc_start - Write code below
32
33 logP = -sum(log(Scale));
34
35 % prob_disc_end - Write code above
```

4.2 Algoritmul Viterbi

```
1 Phi(1, :) = log(Pi) + logB(:, O(1))'; % Initialization for ...
     Phi (t = 1)
2
3 %% Recursion
4 % compute Phi and Psi for t=2:T
5 % phi_psi_disc-start
6 for t=2:T
7     [Phi(t,:), Psi(t,:)] = max(repmat(Phi(t-1,:)',1,N) + ...
         logA);
8     Phi(t,:) = Phi(t,:) + logB(:,O(t))';
9 end
10 % phi_psi_disc-end
11
12 %% logP
13 [logP, Q(T)] = max(Phi(T, :));
14
```

```
%% Backtracking to compute the path Q
% compute Q
% path_disc-start - Write code below
for t=(T-1):-1:1
    Q(t) = Psi(t+1,Q(t+1));
end
% path_disc-end - Write code below
```

4.3 Algoritmul Baum-Welch

Din testele de performanță făcute, s-a dovedit mai eficient să fie aduse matricele `O`, `A`, `B`, `Alpha` și `Beta` în 4 dimensiuni (prin copiere cu operația `repmat` și interschimbare a dimensiunilor cu operațiile `permute` și `shiftdim`) pentru a fi înmulțite apoi element cu element.

```
% maximization_disc-start - Write code below

    % Auxiliary variables
    A_4D = zeros(L,TMax,N,N);
    B_4D = zeros(L,TMax,N,N);
    Alpha_4D = zeros(L,TMax,N,N);
    Beta_4D = zeros(L,TMax,N,N);
    V = 1 : M;

    for l=1:L
        % Add dimension to multiply element by element
        Alpha_4D(l,1:(T(l)-1),:,:) = ...
            repmat(Alpha(l,1:(T(l)-1),:),[1 1 1 N]);

        A_4D(l,1:(T(l)-1),:,:) = ...
            permute(repmat(A,[1 1 1 (T(l)-1)]), [3 4 1 2]);

        B_4D(l,1:(T(l)-1),:,:) = ...
            permute(repmat(B(:,O(l,2:T(l))),[1 1 1 N]), [3 ...
                2 4 1]);

        Beta_4D(l,1:(T(l)-1),:,:) = ...
            permute(repmat(Beta(l,2:T(l),:), [1 1 1 N]), ...
                [1 2 4 3]);
    end

    A_aux = shiftdim(sum(sum( ...
```

```
26              Alpha_4D .* A_4D .* B_4D .* Beta_4D ...
27          , 1),2),2);
28      % A_aux is now N x N
29      A = A_aux ./ repmat(sum(A_aux, 2), [1 N]);
30
31      % reestimate emission probabilities for each dimension ...
           of the
32      % observed variables
33      % also use laplacian smoothing with a factor of 1.0e-4
34      B = (shiftdim(sum(sum( ...
35          (permute(repmat(repmat(O,[1 1 M]) == ...
36          permute(repmat(V,[L 1 TMax]), [1 3 2]),[1 1 1 N]), ...
               [1 2 4 3])) .* ...
37          repmat(Alpha .* Beta ./ repmat(Scale,[1 1 N]),[1 1 ...
               1 M]) ...
38          ,1),2),2) + ones(N, M) * 1.0e-4) ./ ...
39          (shiftdim(sum(sum( ...
40          repmat(Alpha .* Beta ./ repmat(Scale,[1 1 N]),[1 1 ...
               1 M]) ...
41          ,1),2),2) + ones(N, M) * 1.0e-4 * M);
42  % maximization_disc-end - Write code above
```

4.4 Recunoașterea Simbolurilor

4.4.1 Precalcularea matricei B

```
1   % precomp_b_disc-start - Write code below
2       for l = 1 : L
3           B_prod(l, :, 1:T(l)) = ones(N, T(l));
4           for t = 1 : T(l)
5               obs_symbol_idx = O(l, :, t)';
6               for r = 1 : R
7                   b_idx = sub2ind(size(B), 1:N, ...
8                       repmat(obs_symbol_idx(r), 1, N), ...
                           repmat(r, 1, N));
9                   B_prod_line = B(b_idx);
10                  B_prod(l, :, t) = B_prod(l, :, t) .* ...
                       B_prod_line;
11              end
12          end
13      end
14  % precomp_b_disc-end - Write code above
```

4.4.2 Clasificarea (recunoașterea) unui simbol

```
% Write below

for s=1:size(symbol_strings, 1)
    symbol_name = symbol_strings(s, :);

    hmm_data_filename = strcat(symbol_name, '_hmm_', ...
                               hmm_transition_model, '.mat');

    % load the codebook vectors and the hmm parameters
    % for the alleged symbol
    load(hmm_data_filename, 'Pi', 'A', 'B');

    O = symbol_get_feature_sequence(track_data, ...
            x_codebook, y_codebook, ...
            resample_interval, ...
            hamming_window_size, hamming_window_step);

    [Prob, ¬, ¬, ¬] = forward_backward_multi_disc(O, Pi, ...
       A, B);
    ll_vector(1, s) = Prob;

    if ll_vector(1, s) > max_ll
        max_ll = ll_vector(1, s);
        most_likely_symbol_idx = s;
    end
end

% load recognition threshold for best symbol so far
candidate_symbol_name = ...
   symbol_strings(most_likely_symbol_idx, :);
hmm_data_filename = strcat(candidate_symbol_name, '_hmm_', ...
                           hmm_transition_model, '.mat');
load(hmm_data_filename, 'symbol_rec_threshold');


if max_ll ≥ symbol_rec_threshold
    recognized_symbol = strtrim(candidate_symbol_name);
else
    recognized_symbol = 'unknown';
end

% Write above
```