GESTIONE UNDERFLOW NEI MODELLI HIDDEN MARKOV

RELATORE: Dr. Lorenzo Finesso

LAUREANDO: Alessandro Fabris

A.A. 2011/2012

UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

GESTIONE UNDERFLOW NEI MODELLI HIDDEN MARKOV

RELATORE: Dr. Lorenzo Finesso

LAUREANDO: *Alessandro Fabris*

Padova, 28 settembre 2012

Indice

Capitolo 1

Introduzione

I modelli hidden Markov sono un potente strumento matematico che ha avuto sviluppo fra il 1960 e il 1970, trovando subito impiego in applicazioni di riconoscimento vocale. Ancora poco conosciuti in ambienti non matematici, essi hanno conosciuto una notevole diffusione verso metà degli anni '80 grazie ai primi dettagliati tutorial. Oggi sono impiegati in svariati amibiti tra cui il riconoscimento e la sintesi vocale, la crittoanalisi, oltre che in alcuni ambiti della bioinformatica.

La seguente trattazione si propone di introdurre i modelli hidden Markov ed i problemi matematici ad essi correlati per poi passare alla questione centrale, quella dell'underflow e della sua gestione. Vengono menzionati e spiegati due metodi risolutivi per trattare i problemi legati ad un modello hidden Markov gestendo correttamente l'underflow; i due approcci vengono poi raffrontati nel capitolo finale.

Capitolo 2

Modelli hidden Markov

I modelli hidden Markov costituiscono un'estensione delle catene di Markov. Una catena di Markov è un processo stocastico che comprende una collezione di stati con cui si punta a modellare la dinamica di un fenomeno. In ogni momento si assume che la catena si trovi in uno di questi stati e lo stato della catena al tempo t è indicato con q_t. Parallelamente viene definita una matrice A che per ogni coppia di stati S_i, S_j indica la probabilità di transizione dallo stato S_i allo stato S_j. Infine si costruisce un vettore π che per ogni stato S_i definisce la probabilità che lo stato iniziale al tempo $t = 1$ sia proprio S_i. Tale descrizione matematica risulta esaustiva in quanto si assume che la probabilità che lo stato q_t corrisponda ad uno preciso dei possibili stati S_i sia indipendente dal tempo t ed influenzata unicamente dallo stato q_{t-1} in cui la catena si trova nell'istante precedente. Modellando un fenomeno con una catena di Markov si suppone che gli stati siano direttamente accessibili ad un osservatore.

Un modello hidden Markov invece generalizza il concetto di catena di Markov al caso in cui tali stati non siano direttamente osservabili. Si ha dunque una netta divisione tra osservazioni e stati che in una catena di Markov coincidono, mentre in un modello hidden Markov divengono oggetti matematici distinti. Le osservazioni y_t costituiscono l'output visibile del sistema e misurato dall'osservatore, gli stati, responsabili della generazione delle uscite, rappresentano la struttura nascosta (da cui il nome hidden Markov), a cui l'osservatore non ha accesso in alcun modo. Essi sono collegati tramite una matrice di probabilità B che per ogni simbolo osservabile o_k ed ogni stato S_i definisce la probabilità condizionata di vedere in uscita il simbolo o_k dato che lo stato attuale della catena sia S_i.

Un esempio semplice e piuttosto comune per comprendere il funzionamento e l'utilità di un modello hidden Markov è il seguente: si considerino quattro urne in una stanza, ciascuna contenente una diversa mescolanza di palle colorate. Per esempio si avrà che, in generale, la probabilità di estrarre una palla rossa dalla prima urna sarà differente dalla probabilità di estrarla dalla seconda. La composizione di ciascuna urna è rispecchiata dalla matrice B e sarà nota in qualche forma all'osservatore esterno. Dall'interno della stanza, che non è accessibile né visibile all'osservatore, un incaricato sceglie di volta in volta un'urna da cui prelevare una palla e procede poi ad un'estrazione casuale da quest'urna. Il meccanismo (in generale stocastico) con cui l'incaricato sceglierà l'urna successiva dipende unicamente dall'urna attuale da cui l'ultima palla è stata estratta, non da ulteriori urne precedenti. L'esito di tale procedimento di scelta è aleatorio ed è rappresentato dalla matrice A. Anche in questo caso l'osservatore esterno conosce le probabilità dei possibili esiti diversi del meccanismo decisionale. Una volta che una palla è estratta, il risultato viene comunicato all'esterno e la palla viene reinserita nell'urna da cui era stata prelevata. In questo esempio le urne costituiscono gli stati nascosti, mentre la sequenza di palle estratte rappresenta l'osservazione. Una possibile sequenza di osservazioni è GIALLO-BLU-BLU-VERDE-GIALLO-BLU e non contiene alcuna indicazione sulle urne. L'osservatore esterno ha accesso solo a questo tipo di informazione e partendo da essa non potrà risalire con certezza alla sequenza di urne (gli stati) che l'hanno generata. Egli potrà al meglio effettuare ragionamenti di massima verosimiglianza, supponendo per esempio che l'estrazione di una palla blu sia avvenuta a partire dall'urna con in proporzione il maggior numero di palle blu.

Definizione formale: un modello hidden Markov è costituito da 5 elementi:

- Un insieme di N stati $\mathcal{S} = S_1, S_2, \ldots, S_N$
- Un insieme di M simboli osservabili in uscita $\mathcal{O} = o_1, o_2 \ldots o_M$
- La matrice di transizione $A = \{a_{ij}\}, \quad a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$
- La matrice di output $B = \{b_j(o_k)\}, \quad b_j(o_k) = P(y_t = o_k | q_t = S_j)$
- Il vettore delle probabilità iniziali $\pi = \{\pi_i\}, \quad \pi_i = P(q_1 = S_i)$

Inoltre per ogni sequenza di stati $Q_1Q_2 \dots Q_t \in \mathcal{S}^t$ e per ogni successione di osservazioni $O_1O_2 \dots O_t \in \mathcal{O}^t$ esso gode delle seguenti proprietà:

$$P(y_1^t = O_1^t | q_1^t = Q_1^t) = \prod_{\tau=1}^{t} b_{Q_\tau}(O_\tau) \tag{2.1}$$

$$P(q_1^t = Q_1^t) = \pi_{Q_1} \prod_{\tau=2}^{t} a_{Q_{\tau-1}Q_\tau} \tag{2.2}$$

Una notazione sintetica per l'intero modello è data da $M = (A, B, \pi)$
Un modello hidden Markov M siffatto genera in uscita sequenze di osservazioni $O = O_1O_2 \dots O_T$ con il seguente procedimento:

1. Entra nello stato iniziale q_1 in base alla misura di probabilità π e pone $t = 1$.
2. Genera un'uscita $y_t = O_t$ sulla base dello stato corrente e della matrice di output B.
3. Se $t = T$ termina la procedura altrimenti si sposta in un nuovo stato $q_{t+1} = S_j$ sulla base delle probabilità di transizione dallo stato attuale $q_t = S_i$, ossia in conformità alla misura di probabilità a_{ij}.
4. Imposta $t = t + 1$; torna al passo 2.

Ci sono tipicamente tre problemi d'interesse nell'utilizzo dei modelli hidden Markov, da risolvere ai fini delle applicazioni che questi modelli hanno nella realtà. Essi vengono introdotti nei prossimi tre sottoparagrafi.

2.1 Valutazione

Dati il modello $\mathbf{M} = (\mathbf{A}, \mathbf{B}, \pi)$ e la sequenza di osservazioni $\mathbf{O} = \mathbf{O_1O_2} \dots \mathbf{O_T}$, come calcolare *efficientemente* la probabilità $\mathbf{P(y = O|M)}$?
Questa è una domanda di primaria importanza poiché la risposta consente di quantificare la plausibilità di un'osservazione in un dato modello. Nel seguito per brevità si indicherà la probabilità cercata con $P(O|M)$ sottintendendo che O è una possibile sequenza di valori d'uscita del modello.

Un approccio di forza bruta nella risoluzione del problema potrebbe essere il seguente: si fissa una sequenza di stati

$$Q = Q_1Q_2 \dots Q_T$$

Si procede calcolando la probabilità congiunta

$$P(O, Q|M) = \pi_{Q_1} b_{Q_1}(O_1) a_{Q_1 Q_2} b_{Q_2}(O_2) \dots a_{Q_{T-1} Q_T} b_{Q_T}(O_T)$$

Qui π_{Q_1} rende conto del fatto che il primo stato della sequenza è Q_1, i termini $b_{Q_i}(O_i)$ tengono in considerazione la rilevazione del simbolo osservato O_i in ogni diverso stato Q_i e i termini $a_{Q_i Q_{i+1}}$ testimoniano il passaggio da uno stato all'altro lungo tutta la sequenza Q.
Ora sommando su tutte le possibili sequenze Q si ottiene la probabilità desiderata:

$$P(O|M) = \sum_{Q} P(O, Q|M) = \sum_{Q_1, Q_2 \dots Q_T} \pi_{Q_1} b_{Q_1}(O_1) a_{Q_1 Q_2} b_{Q_2}(O_2) \dots a_{Q_{T-1} Q_T} b_{Q_T}(O_T)$$

Tale calcolo sarebbe tuttavia estremamente laborioso, richiedendo una quantità di operazioni dell'ordine di $2TN^T$ il che risulta impraticabile già per moderati valori di T.

Il metodo risolutivo utilizzato nella pratica è la cosiddetta formula di Baum. Essa fa uso della sequenza $\alpha_t(i)$, detta *forward variable*, così definita

$$\alpha_t(i) = P(O_1 O_2 \dots O_t, q_t = S_i|M) \tag{2.3}$$

Ogni $\alpha_t(i)$ rappresenta la probabilità di avere in output la sequenza parziale $O_1 \dots O_t$ fino al tempo t e di trovare in tale istante il modello nello S_i. La computazione delle *forward variables* avviene come segue:

- **Inizializzazione**

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{2.4}$$

- **Induzione**

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}) \tag{2.5}$$

L'inizializzazione è costituita dalla probabilità congiunta di trovarsi all'istante iniziale in S_i e di rilevare in uscita O_1. Il passo induttivo procede calcolando innanzitutto il prodotto $\alpha_t(i) a_{ij} = P(O_1 O_2 \dots O_t, q_t = S_i, q_{t+1} = S_j|M)$, poi sommando su tutti gli i si ha $\sum_{i=1}^{N} \alpha_t(i) a_{ij} = P(O_1 O_2 \dots O_t, q_{t+1} = S_j|M)$ ed infine moltiplicando per $b_j(O_{t+1})$ si ottiene $P(O_1 O_2 \dots O_t O_{t+1}, q_{t+1} = S_j|M)$, che corrisponde proprio ad $\alpha_{t+1}(j)$.

Una volta calcolato $\alpha_T(i)\,\forall i$, ossia alla fine dell'ultima iterazione, il calcolo di $P(O|M)$ è immediato:

$$P(O|M) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.6}$$

Questo consegue immediatamente dalla definizione di $\alpha_t(i)$ (2.3).
La procedura iterativa richiede, tra somme e moltiplicazioni, un numero di operazioni che va come TN^2, risultando nettamente più efficiente del procedimento brute force descritto in precedenza.
Parallelamente alla *forward variable* si costruisce la *backward variable*. Essa non è fondamentale ai fini del problema della valutazione, ma è comunque opportuno menzionarla per farne utilizzo in seguito. Si definisce

$$\beta_t(i) = P(O_{t+1}O_{t+2}\dots O_T|q_t = S_i, M) \tag{2.7}$$

Il calcolo dei $\beta_t(i)$ avviene con un procedimento del tutto analogo a quello descritto per gli $\alpha_t(i)$:

- **Inizializzazione**

$$\beta_T(i) = 1 \tag{2.8}$$

- **Induzione**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{2.9}$$

$\beta_T(i)$ è inizializzata ad 1, ma non è una probabilità che ha significato di per sé. Basta osservare la definizione di (2.7) per convincersene. Tale valore va però posto ad 1 per rendere corrette le ricorsioni successive, in particolare il calcolo di $\beta_{T-1,T-2,\dots}$, oltre che per utilizzi successi come ad esempio il calcolo di $\gamma_t(i)$ in (2.11). La spiegazione del passo induttivo è analoga a quella per la *forward variable*:

$$\begin{aligned}
\beta_t(i) &= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \\
&= \sum_{j=1}^{N} P(q_{t+1} = S_j|q_t = S_i) P(O_{t+1}|q_{t+1} = S_j) P(O_{t+2}\dots O_T|q_{t+1} = S_j) \\
&= \sum_{j=1}^{N} P(q_{t+1} = S_j|q_t = S_i) P(O_{t+1}O_{t+2}\dots O_T|q_{t+1} = S_j)
\end{aligned}$$

$$= \sum_{j=1}^{N} P(O_{t+1} \dots O_T, q_{t+1} = S_j | q_t = S_i)$$

La complessità asintotica per il calcolo dei $\beta_t(i)$ è nuovamente dell'ordine di TN^2 operazioni richieste.

2.2 Ricostruzione

Dati il modello $\mathbf{M} = (\mathbf{A}, \mathbf{B}, \pi)$ e la sequenza di osservazioni $\mathbf{O_1 O_2 \dots O_T}$, come scegliere la sequenza di stati $\mathbf{Q} = \mathbf{q_1 q_2 \dots q_T}$ ipoteticamente attraversata dal modello, ottimizzandola in maniera oppoortuna?
La ricostruzione pone il problema di risalire alla successione di stati responsabile della sequenza di osservazioni rilevata in uscita. Come già accennato non esistono sequenze corrette e sequenze sbagliate, ci sono sequenze di stati più verosimili di altre in base a ragionevoli criteri di bontà.

Un possibile criterio consiste nello scegliere individualmente gli stati più verosimili. A tal fine si definisce

$$\gamma_t(i) = P(q_t = S_i | O, M) \tag{2.10}$$

Tale sequenza può essere ricavata in funzione delle *forward* e *backward variables*:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|M)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{2.11}$$

A questo punto si ricavano gli stati individualmente più probabili come segue:

---

for $t \leftarrow 1$ **to** T **do**
  $q_t = \arg\max_i [\gamma_t(i)]$
end

In questo modo si massimizza il valore atteso del numero di stati corretti all'interno della sequenza ma non si prende in considerazione la verosimiglianza della sequenza nel complesso. Infatti tale sequenza potrebbe essere del tipo $Q = SSS \dots S_i S_j \dots SS$ con S uno stato qualunque ed $S_i S_j$ tali che valga $a_{ij} = 0$,

ossia che dallo stato S_i il modello M non possa spostarsi direttamente allo stato S_j. Questo renderebbe nulla la probabilità della sequenza Q poiché essa contiene una transizione "proibita" (da S_i ad S_j).

Il criterio più utilizzato si focalizza sulla ricerca della sequenza *globalmente* ottima, puntando a massimizzare $P(Q|O, M)$ o equivalentemente $P(Q, O|M)$. Tale compito è assolto efficientemente dall'algoritmo di Viterbi. Esso fa uso della variabile

$$\delta_t(i) = \max_{q_1, q_2, \ldots q_{t-1}} P(q_1 q_2 \ldots q_{t-1}, q_t = S_i, O_1 O_2 \ldots O_t | M) \tag{2.12}$$

che misura la massima probabilità per il modello M di avere una sequenza di stati che finisca al tempo t nello stato $q_t = S_i$ e che dia origine all'osservazione $O = O_1 \ldots O_t$ fino al tempo t. Il calcolo dei $\delta_t(i)$ è effettuato iterativamente sfruttando la proprietà

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] b_j(O_{t+1}) \tag{2.13}$$

L'intero algoritmo è riportato nella pagina seguente.

```
for i ←N to 1 do
    δ_1(i) ← π_i b_i(O_1);
    ψ_1(i) = 0;
end
for t ←2 to T do
    for j ←1 to N do
        max ← δ_{t−1}(1) a_{1j};
        ψ_t(j) ← 1;
        for i ←2 to N do
            temp ← δ_{t−1}(i) a_{ij};
            if (temp > max) then
                max ← temp;
                ψ_t(j) ← i;
            end
        end
        δ_t(j) ← max · b_j(O_t);
    end
end
P* ← δ_T(1);
q*_T ← 1;
for i ←2 to N do
    if δ_T(i) > P* then
        P* ← δ_T(i);
        q*_T ← i;
    end
end
for t ←T-1 to 1 do
    q*_t ← ψ_{t+1}(q*_{t+1});
end
```

2.3 Stima parametrica

Come adattare i parametri del modello $\mathbf{M} = (\mathbf{A}, \mathbf{B}, \pi)$ in modo da massimizzare $\mathbf{P(O|M)}$?

Questo problema (denominato talvolta *training*) è fondamentale perché consente di aggiustare i parametri del modello sulla base delle osservazioni, rendendoli così più aderenti alla realtà. La sua risoluzione è complessa e non esiste una via analitica per la ricerca del massimo globale. Si può però fare affidamento sull'algoritmo di Baum-Welch il quale fa uso di un procedimento iterativo al termine del quale $P(O|M)$ risulterà localmente massimizzata. A tal fine introduciamo

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, M) \tag{2.14}$$

$\xi_t(i,j)$ misura la probabilità di trovarsi all'istante t nello stato S_i e in quello successivo nello stato S_j, dati la sequenza osservata O e il modello M. Tale quantità può essere ricavata come segue:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|M)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{2.15}$$

Un'interpretazione utile delle grandezze definite in (2.10) e (2.14) è la seguente:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{valore atteso del numero di transizioni da } S_i$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{valore atteso del numero di transizioni da } S_i \text{ a } S_j$$

Sfruttando le semplici considerazioni appena effettuate è possibile ricavare le formule per la stima parametrica:

$$\bar{\pi}_i = \gamma_1(i) = P(q_1 = S_i | O, M) \tag{2.16}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.17}$$

$$\bar{b}_j(o_k) = \frac{\sum_{t:O_t=o_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.18}$$

Definito $M = (A, B, \pi)$ il modello vecchio, sfruttato per calcolare il secondo membro di (2.16)-(2.18), e $\bar{M} = (\bar{A}, \bar{B}, \bar{\pi})$ il nuovo modello ricavato come primo membro di (2.16)-(2.18) si dimostra che o il modello iniziale M costituiva già un massimo locale per $P(O|M)$, nel qual caso si trova $M = \bar{M}$, oppure che la verosimiglianza dell'osservazione O nel nuovo modello $\bar{M}$ è aumentata, ossia vale $P(O|\bar{M}) > P(O|M)$.

Capitolo 3

Gestione underflow

L'utilizzo dei modelli hidden Markov può comportare dei problemi di ordine pratico, tra cui il cosiddetto underflow. In particolare il calcolo dei coefficienti $\alpha_t(i)$ e $\beta_t(i)$ consiste di successive moltiplicazioni di fattori $a_{ij}\ b_j(O_t)$ (cfr 2.5 e 2.9). Tali fattori hanno modulo inferiore all'unità e il loro numero all'interno della produttoria cresce all'aumentare del numero di osservazioni. Pertanto accresccendo il numero delle osservazioni e/o la complessità del modello e si ha che i termini $\alpha_t(i)$ e $\beta_t(i)$ tendono esponenzialmente a zero. Si ricordi infatti che tali termini rappresentano delle misure della probabilità del verificarsi di sequenze di osservazioni, sequenze la cui probabilità diminuisce all'aumentare del numero di osservazioni successive. Sebbene nella teoria questo aspetto sia di scarsa importanza, nella pratica costituisce un problema di tutto rilievo. Infatti, poiché per ovvi motivi di rapidità la computazione è affidata alle macchine, può accadere che le probabilità rappresentate da $\alpha_t(i)$ e $\beta_t(i)$ diventino così piccole da scendere al di sotto del range di precisione dei computer (anche con rappresentazione double), dando luogo all'underflow. Questo fenomeno avviene già per un numero di osservazioni dell'ordine di 10^2 e rappresenta dunque un problema da gestire nella maggioranza dei casi d'interesse.

Nel seguito vengono descritte e raffrontate due tecniche proposte da Rabiner e Mann (cfr [5] e [4]) per la gestione di tale fenomeno che rendono possibile il corretto utilizzo dei modelli hidden Markov nonostante l'aumentare della loro complessità.

3.1 Scaling

L'idea alla base dello scaling è di moltiplicare le probabilità $\alpha_t(i)$ per un coefficiente di scalamento opportuno così da mantenerne il modulo entro il range di computabilità della macchina. Tale coefficiente di scaling è definito come

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{3.1}$$

Gli $\alpha_t(i)$ vengono dunque calcolati in base a (2.4) e (2.5) e poi scalati opportunamente moltiplicandoli per (3.1), ottenendo le nuove sequenze $\hat{\alpha}_t(i)$. Questo avviene ad ogni iterazione; la formula diventa dunque:

$$\hat{\alpha}_1(i) = \frac{\pi_i b_i(O_1)}{\sum_{i=1}^{N} \pi_i b_i(O_1)} \tag{3.2}$$

$$\hat{\alpha}_{t+1}(i) = \frac{\left[\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{ji}\right] b_i(O_{t+1})}{\sum_{i=1}^{N} \left\{\left[\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{ji}\right] b_i(O_{t+1})\right\}} \tag{3.3}$$

I denominatori al secondo membro di (3.2) e (3.3) sono i reciproci dei coefficienti di scaling c_t al tempo 1 e $t+1$ rispettivamente.
In base alla ricorsione definita nelle ultime due formule osserviamo che si instaura fra i vecchi $\alpha_t(i)$ e quelli riscalati il seguente legame:

$$\hat{\alpha}_1(i) = \alpha_1(i) c_1 \tag{3.4}$$

$$\hat{\alpha}_t(i) = \left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i) = C_t \alpha_t(i) \tag{3.5}$$

Dalle relazioni sin qui esposte, notando che $\sum_{i=1}^{N} \hat{\alpha}_t(i) = 1$, è possibile chiarire il significato di C_t:

$$\begin{aligned} \sum_{i=1}^{N} \hat{\alpha}_t(i) &= C_t \sum_{i=1}^{N} \alpha_t(i) \\ &= C_t \sum_{i=1}^{N} P(O_1 \dots O_t, q_t = S_i | M) \\ &= C_t P(O_1 \dots O_t | M) = 1 \\ \text{pertanto} \qquad & C_t = \frac{1}{P(O_1 \dots O_t | M)} \end{aligned} \tag{3.6}$$

Il passo successivo consiste nel riscalare anche i termini $\beta_t(i)$ a mezzo di un'analoga moltiplicazione. C'è però una differenza e risiede nel fatto che i coefficienti utilizzati per il rescaling dei $\beta_t(i)$ sono gli stessi impiegati in precedenza per gli $\alpha_t(i)$, ossia quelli individuati dall'equazione (3.1). Si trova dunque per i $\hat{\beta}_t(i)$ la seguente relazione:

$$\hat{\beta}_t(i) = \left(\prod_{\tau=t}^{T} c_\tau \right) \beta_t(i) = D_t \beta_t(i) \tag{3.7}$$

La procedura di scaling è così ultimata. Essa, come già notato, consente di evitare il fenomeno dell'underflow ed esibisce altresì una fondamentale proprietà: nonostante vada ad alterare le *forward* e *backward variables* $\alpha_t(i)$ e $\beta_t(i)$, essa garantisce comunque un'agevole risoluzione per i tre problemi di valutazione, ricostruzione e stima parametrica, a patto di modificare leggermente i procedimenti esposti in precedenza.

Per risolvere il problema della valutazione non si può utilizzare (2.6) sommando gli $\hat{\alpha}_t(i)$ poiché questi sono stati sottoposti a scaling. Si può però sfruttare (3.6) specializzandola come segue:

$$\begin{aligned} C_T &= \frac{1}{P(O_1 \dots O_T|M)} \\ P(O|M) &= \frac{1}{\prod_{t=1}^{T} c_t} \qquad (3.8) \\ \log[P(O|M)] &= -\sum_{t=1}^{T} \log c_t \qquad (3.9) \end{aligned}$$

Non è possibile calcolare $P(O|M)$ diretamente poiché nuovamente incapperemmo nell'underflow. Grazie a (3.9) si riesce però a stimare $\log[P(O|M)]$ che risulta utile quando, studiando alcuni modelli alternativi l'uno all'altro, si vuole stabilire quale di questi combaci meglio con le osservazioni effettuate. A tal fine sarà sufficiente valutare quale sia il modello che massimizza $\log[P(O|M)]$.

Per quanto concerne il problema della stima parametrica, in linea di principio si dovrebbe modificare la procedura di Baum-Welch, poichè essa fa ampio uso

delle *forward* e *backward variables* originarie, prima cioè che esse vengano riscalate. Tuttavia, come dimostrato nel seguito, le formule (2.16)-(2.18) per ristimare i parametri A, B e π risultano invarianti per rescaling, permettendo di fatto di utilizzare i coefficienti riscalti $\hat{\alpha}_t(i)$ e $\hat{\beta}_t(i)$ invece dei normali coefficienti $\alpha_t(i)$ e $\beta_t(i)$ ottenendo il medesimo risultato.

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)} \tag{3.10}$$

e in base a (3.5) e (3.7) si può scrivere

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} C_t \alpha_t(i) a_{ij} b_j(O_{t+1}) D_{t+1} \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} C_t \alpha_t(i) a_{ij} b_j(O_{t+1}) D_{t+1} \beta_{t+1}(j)} \tag{3.11}$$

Infine osservando che

$$C_t D_{t+1} = \prod_{\tau=1}^{t} c_\tau \prod_{\tau=t+1}^{T} c_\tau = \prod_{\tau=1}^{T} c_\tau = C_T \tag{3.12}$$

indipendentemente dal tempo t, si possono raccogliere tali termini al numeratore e al denominatore elidendoli l'uno con l'altro, riottenendo (2.17).
Analogo ragionamento può essere effettuato per i parametri B e π. Dalle formule di stima parametrica è noto infatti che essi sono immediatamente ricavabili una volta nota la sequenza $\gamma_t(i)$. Anche per essa vale la proprietà di invarianza per rescaling:

$$\gamma_t(i) = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{i=1}^{N} \hat{\alpha}_t(i)\hat{\beta}_t(i)} = \frac{C_T c_t \alpha_t(i)\beta_t(i)}{C_T c_t \sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{3.13}$$

Questo garantisce che il calcolo delle $\gamma_t(i)$ e conseguentemente di B e π possa essere effetuato correttamente in seguito al rescaling delle forward e backaward variables.
È stato così verificato che il problema del training può essere trattato con il metodo di Baum-Welch senza incorrere nell'underflow.
Benché estremamente utile in tal senso, lo scaling presenta degli oneri computazionali aggiuntivi. Talvolta questi possono essere evitati: per tempi piccoli (quando la sequenza di osservazioni in (2.3) è ancora di modeste dimensioni) è improbabile che si verifichi underflow. In questi primi passaggi lo scaling può

essere bypassato ed applicato in un secondo momento quando il range delle probabilità $\alpha_t(i)$ e $\beta_t(i)$ si avvicina ai valori critici. Se ad un istante t lo scaling non è necessario, è sufficiente impostare ad 1 il coefficiente c_t, mantenendo valide le proprietà discusse sopra.

Il problema della ricostruzione non è assolutamente influenzato dallo scaling; è sufficiente osservare la procedura di Viterbi per convincersene. L'algoritmo, infatti, fa uso dei coefficienti A, B e π, i quali, come appena dimostrato, rimangono inalterati.

3.2 Calcolo logaritmico

Una soluzione alternativa allo scaling nella gestione dell'underflow consiste nel calcolare i logaritmi dei coefficienti $\alpha_t(i)$ e $\beta_t(i)$. L'idea, proposta da Durbin e colleghi in [1] e successivamente sviluppata da Mann (cfr [4]), è piuttosto efficace e di semplice realizzazione. Occorre innanzitutto definire delle funzioni ausiliari che consentano di lavorare con probabilità nulle. La funzione logaritmo non è infatti definita su input zero, laddove è invece possibile che una probabilità sia nulla. Per trattare correttamente tale eventualità si estendono le funzioni esponenziali e logaritmiche di modo da poter gestire il valore 0 come argomento della funzione logaritmo. "$\log 0$" verrà nel seguito indicato come *LOGZERO*. Rispettando la notazione di Mann si definisce:

- **eexp(x)** La funzione esponenziale estesa

$$eexp(x) = \begin{cases} e^x & x \text{ numero reale} \\ 0 & x = LOGZERO \end{cases} \tag{3.14}$$

- **eln(x)** La funzione logaritmo estesa

$$eln(x) = \begin{cases} \ln(x) & x \text{ numero reale positivo} \\ LOGZERO & x = 0 \end{cases} \tag{3.15}$$

- **elnsum(eln(x),eln(y))** Il logaritmo esteso della somma di x ed y con input i logaritmi estesi di x e y

$$elnsum(eln(x), eln(y)) = \begin{cases} eln(x+y) & x, y \text{ numeri reali positivi} \\ eln(x) & y = 0 \text{ ossia } eln(y) = LOGZERO \\ eln(y) & x = 0 \text{ ossia } eln(x) = LOGZERO \end{cases} \tag{3.16}$$

- **elnproduct(eln(x),eln(y))** Il logaritrmo esteso del prodotto di x ed y con input i logaritmi estesi di x e y

$$elnproduct(eln(x), eln(y)) = \begin{cases} eln(x) + eln(y) & x, y \text{ numeri reali positivi} \\ LOGZERO & y = 0 \\ LOGZERO & x = 0 \end{cases} \tag{3.17}$$

Lo pseucodice per le funzioni appena definite è riportato di seguito:

Algorithm 1: calcolo di eexp(x)

```
if x = LOGZERO then
    return 0 ;
else
    return exp(x) ;
end
```

Algorithm 2: calcolo di eln(x)

```
if x=0 then
    return LOGZERO;
else if x > 0 then
    return ln(x);
else
    negative input error
end
```

Algorithm 3: calcolo di elnsum(eln(x),eln(y))

```
if eln(x)=LOGZERO OR eln(y)=LOGZERO then
    if eln(x)=LOGZERO then
        return eln(y);
    else
        return eln(x);
    end
else
    if eln(x) > eln(y) then
        return eln(x)+eln[1+exp(eln(y)-eln(x))];
    else
        return eln(y)+eln[1+exp(eln(x)-eln(y))];
    end
end
```

La formula da utilizzare per il calcolo del logaritmo esteso della somma tra x e y si ricava con dei semplici passaggi:

$$\begin{aligned} \ln(x) + \ln(1 + e^{\ln(y)-\ln(x)}) &= \ln(x) + \ln(1 + e^{\ln(y/x)}) \\ &= \ln(x) + \ln\left(1 + \frac{y}{x}\right) \\ &= \ln(x) + \ln\left(\frac{x+y}{x}\right) \\ &= \ln(x+y) \end{aligned}$$

Il controllo sulla dimensione dei logaritmi degli addendi è effettuato ai fini della stabilità numerica. Infatti, all'interno dell'utimo **if else** si verifica che l'esponenziale calcolato abbia argomento negativo, così da risultare un valore molto piccolo. Al limite esso verrà considerato nullo dal computer, determinando un'approssimazione accettabile. Il caso opposto, dal quale ci si tutela, è quello di un esponente positivo. Al limite tale esponente può avere un modulo molto grande, causando un errore di overflow che la macchina non può gestire. Nel primo caso si perviene ad un risultato approssimato ma accettabile, nel secondo ad un errore. Risulta dunque giustificata la precauzione descritta.

Algorithm 4: calcolo di elnproduct(eln(x),eln(y))

```
if eln(x)=LOGZERO OR eln(y)=LOGZERO then
    return LOGZERO ;
else
    return eln(x)+eln(y) ;
end
```

Una volta definita l'estensione delle funzioni d'interesse si procede delineando gli algoritmi per la risoluzione dei tre problemi fondamentali (valutazione, ricostruzione e stima parametrica) mediante le funzioni logaritmiche. Bisogna innanzitutto computare i logaritmi di $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$, e $\xi_t(i,j)$ modificando leggermente le procedure descritte nel capitolo precedente.

Algorithm 5: calcolo di $eln\alpha_t(i)$ per ogni stato S_i e tempo t

```
for i ←1 to N do
    elnα_1(i) ← elnproduct(eln(π_i), eln(b_i(O_1)));
end
for t ←2 to T do
    for j ←1 to N do
        logalpha ← LOGZERO;
        for i ←1 to N do
            logalpha ← elnsum(logalpha, elnproduct(elnα_{t−1}(i), eln(a_ij)))
        end
        elnα_t(j) ← elnproduct(logalpha, eln(b_j(O_t)));
    end
end
```

Algorithm 6: calcolo di $eln\beta_t(i)$ per ogni stato S_i e tempo t

for $i \leftarrow 1$ **to** N **do**
  $eln\beta_T(i) \leftarrow 0$;
end
for $t \leftarrow T\text{-}1$ **to** 1 **do**
  for $i \leftarrow 1$ **to** N **do**
    $logbeta \leftarrow LOGZERO$;
    for $j \leftarrow 1$ **to** N **do**
      $logbeta \leftarrow elnsum(logbeta, elnproduct(eln(a_{ij}),$
        $elnproduct(eln(b_j(O_{t+1})), eln\beta_{t+1}(j))))$;
    end
    $eln\beta_t(i) \leftarrow logbeta$;
  end
end

Algorithm 7: calcolo di $eln\gamma_t(i)$ per ogni stato S_i e tempo t

for $t \leftarrow 1$ **to** T **do**
  $normalizer \leftarrow LOGZERO$;
  for $i \leftarrow 1$ **to** N **do**
    $eln\gamma_t(i) \leftarrow elnproduct(eln\alpha_t(i), eln\beta_t(i))$;
    $normalizer \leftarrow elnsum(normalizer, eln\gamma_t(i))$;
  end
  for $i \leftarrow 1$ **to** N **do**
    $eln\gamma_t(i) \leftarrow elnproduct(eln\gamma_t(i), -normalizer)$;
  end
end

Algorithm 8: calcolo di $eln\xi_t(i,j)$ per ogni coppia di stati S_i, S_j e per ogni tempo t

```
for t ←1 to T − 1 do
    normalizer ← LOGZERO;
    for i ←1 to N do
        for j ←1 to N do
            elnξ_t(i,j) ← elnproduct(elnα_t(i), elnproduct(eln(a_ij),
                elnproduct(elnb_j(O_{t+1}), elnβ_{t+1}(j))));
            normalizer ← elnsum(normalizer, elnξ_t(i,j));
        end
    end
    for i ←1 to N do
        for j ←1 to N do
            elnξ_t(i,j) ← elnproduct(elnξ_t(i,j), −normalizer);
        end
    end
end
```

L'algoritmo 9 risolve il problema della valutazione.

Algorithm 9: calcolo di $P(O|M)$

```
logP ← LOGZERO;
for i ←1 to N do
    logP ← elnsum[logP, elnα_T(i)];
end
```

La procedura di Viterbi per la ricostruzione è descritta dall'algoritmo 10

Gli ultimi tre algoritmi (11, 12 e 13) rappresentano la soluzione al problema della stima parametrica, ossia la procedura di Baum-Welchm, riadattata con il calcolo logaritmico, per ristimare i parametri A, B e π

Algorithm 10: calcolo della sequenza q_t^* più pausibile per la sequenza di osservazioni O_1^T

for $i \leftarrow 1$ **to** N **do**
  $eln\delta_1(i) \leftarrow elnproduct[eln(\pi_i), eln(b_i(O_1))]$;
  $\psi_1(i) = 0$;
end
for $t \leftarrow 2$ **to** T **do**
  for $j \leftarrow 1$ **to** N **do**
    $elnmax \leftarrow elnproduct(eln\delta_{t-1}(1), eln(a_{1j}))$;
    $\psi_t(j) \leftarrow 1$;
    for $i \leftarrow 2$ **to** N **do**
      $temp \leftarrow elnproduct(eln\delta_{t-1}(i), eln(a_{ij}))$;
      if $(temp \neq LOGZERO)$ ***AND*** $(elnmax = LOGZERO$ ***OR*** $temp > elnmax)$ **then**
        $elnmax \leftarrow temp$;
        $\psi_t(j) \leftarrow i$;
      end
    end
    $eln\delta_t(j) \leftarrow elnproduct(elnmax, eln(b_j(O_t)))$;
  end
end
$elnP^* \leftarrow eln\delta_T(1)$;
$q_T^* \leftarrow 1$;
for $i \leftarrow 2$ **to** N **do**
  if $eln\delta_T(i) > elnP^*$ **then**
    $elnP^* \leftarrow eln\delta_T(i)$;
    $q_T^* \leftarrow i$;
  end
end
for $t \leftarrow T\text{-}1$ **to** 1 **do**
  $q_t^* \leftarrow \psi_{t+1}(q_{t+1}^*)$;
end

Algorithm 11: calcolo di $\bar{\pi}_i$ per ogni stato S_i

```
for i ←1 to N do
    π̄_i ← eexp(elnγ_1(i))
end
```

Algorithm 12: calcolo di $\bar{a}_{ij}$ per ogni coppia di stati S_i e S_j

```
for j ←1 to N do
    for i ←1 to N do
        numerator ← LOGZERO;
        denominator ← LOGZERO;
        for t ←1 to T − 1 do
            numerator ← elnsum(numerator, elnξ_t(i, j));
            denominator ← elnsum(denominator, elnγ_t(i));
        end
        ā_ij ← eexp(elnproduct(numerator, −denominator));
    end
end
```

Algorithm 13: calcolo di $\bar{b}_j(o_k)$ per ogni stato S_j e simbolo osservabile o_k

```
for j ←1 to N do
    numerator ← LOGZERO;
    denominator ← LOGZERO;
    for t ←1 to T do
        if O_t = o_k then
            numerator ← elnsum(numerator, elnγ_t(j));
        end
        denominator ← elnsum(denominator, elnγ_t(i));
    end
    b̄_j(o_k) ← eexp(elnproduct(numerator, −denominator));
end
```

Capitolo 4

Simulazioni e conclusioni

I due metodi descritti nel precedente capitolo possono essere raffrontati per determinare quale esibisca una migliore efficienza computazionale.

Da un punto di vista dell'occupazione in memoria essi risultano equivalenti, dovendo entrambi immagazzinare nell'hardware le matrici $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$, $\delta_t(i)$ e $\xi_t(i,j)$ eventualmente riscalate o messe in forma logaritmica.

Un secondo confronto si può effettuare sul piano delle prestazioni, paragonando i tempi necessari per la computazione. Da un punto di vista asintotico entrambi i metodi sono teoricamente efficienti; essendo N il numero di stati della catena di Markov e T la dimensione della sequenza di osservazioni, l'andamento asintotico dei tempi d'esecuzione richiede per entrambi i metodi un numero $\Theta(N^2T)$ di operazioni.
Ciò che differenzia i due algoritmi è il tipo di operazioni richieste. Il metodo logaritmico richiede il calcolo di un logaritmo ed un esponenziale ogni volta che una somma dev'essere effettuata (si veda la definizione di *elnsum* per convincersene) che sono operazioni particolarmente dispendiose. Per contro rispetto allo scaling il metodo logaritmico esibisce il vantaggio di effettuare una somma laddove normalmente un prodotto è richiesto (si veda la funzione *elnproduct*). Cionondimeno sulla carta è il metodo dello scaling quello più conveniente proprio a causa dell'onere computazionale insito nel calcolo di logaritmi ed esponenziali presenti nel metodo logaritmico. L'adozione di tabelle potrebbe velocizzare il calcolo di esponenziali e logaritmi, ma a scapito della precisione.
Ai fini di un discorso quantitativo sulle prestazioni, esso andrebbe specializzato per la singola macchina, prendendo in considerazione il numero di cicli di clock

richiesti per il calcolo di somme, moltiplicazioni, logaritmi ed esponenziali.
Le simulazioni che seguono sono state effettuate su un PC dotatato di processore Intel Core i7 Q740 da 1.73GHz. Il codice matlab utilizzato per effettuare i test è riportato di seguito.

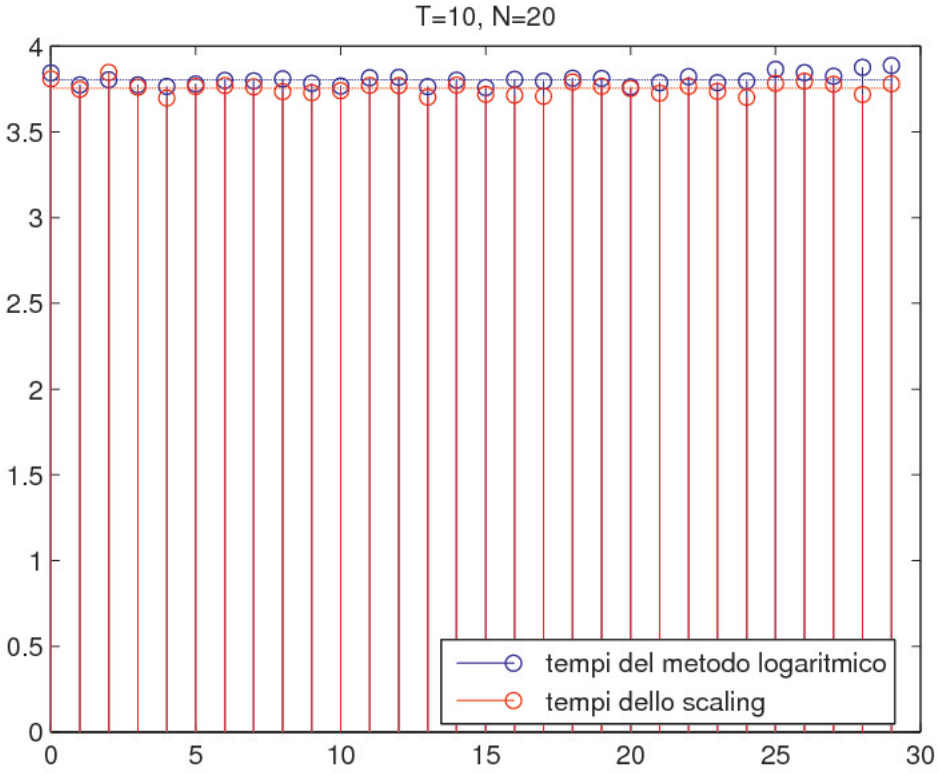
```
close all

%numero di simboli distinti osservabili
M=4

%numero stati distinti
N=5 % altri valori testati: N=10, N=20

%dimensione sequenza osservazioni
T=5 %altri valori testati: T=10, T=50, T=100, T=200

%ripetizioni
P=30


%definizione delle matrici di modello
PI=rand([1,N])
PI=PI/sum(PI)
%affinché PI sommi ad 1

A=rand(N)
s=sum(A')
%ciclo for perché la somma in j degli a_{ij} sia 1
for i=1:N
    for j=1:N
        A(i,j)=A(i,j)/s(i)
    end
end

B=rand(N,M)
s=sum(B')
%ciclo for perché la somma in k dei b_j(O_k) sia 1
```

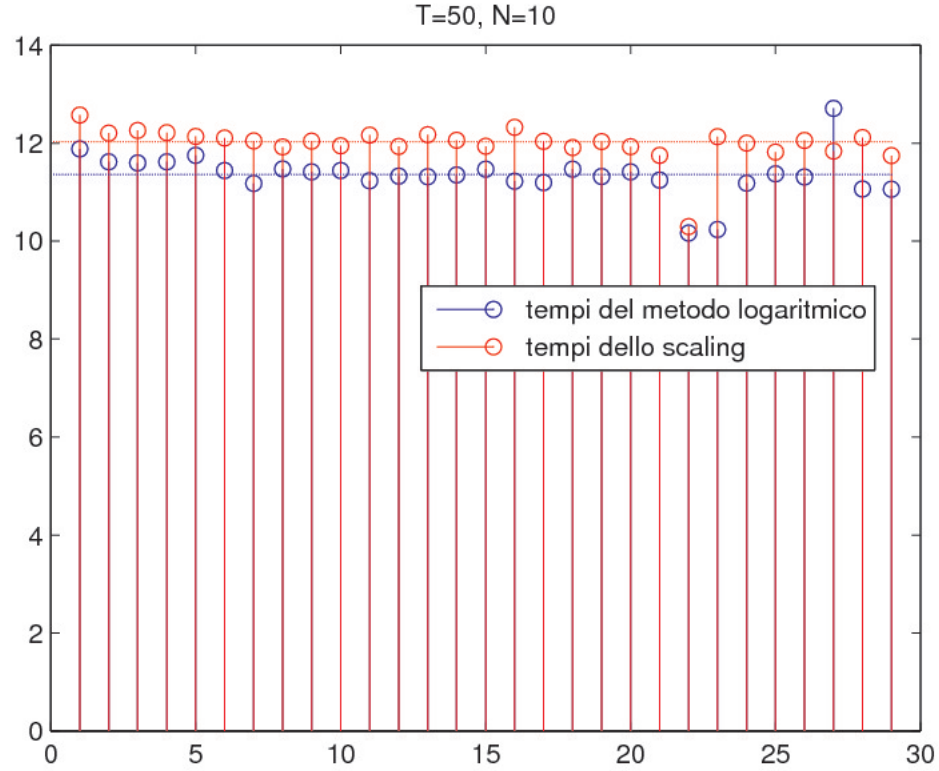
```
33 for i=1:N
       for j=1:M
35         B(i,j)=B(i,j)/s(i)
       end
37 end


39
   TL=zeros(1,P)  %tempo Logspace
41 TS=zeros(1,P)  %tempo Scaling

43 x=0:1:P-1
   y=0:0.1:P-1
45


47 %ciclo for esterno per effettuare P ripetizioni
   for p=1:P
49     %generazione casuale di osservazione
       O=rand([1,T])
51     O=O*M
       O=ceil(O)
53
       STOP=tic
55
       %inizio metodo logaritmico
57     LOGALPHA=zeros(T,N)
       for i=1:N
59         LOGALPHA(1,i)=elnproduct(log(PI(i)),log(B(i,O(1))))
       end
61     for t=1:T-1
           for j=1:N
63             LOGALPHA(t+1,j)=-Inf
               for i=1:N
65                 LOGALPHA(t+1,j)=elnsum(LOGALPHA(t+1,j),
                           elnproduct(LOGALPHA(t,i),log(A(i,j))))
67             end
               LOGALPHA(t+1,j)=elnproduct(LOGALPHA(t+1,j),
69                                 log(B(j,O(t+1))))
```

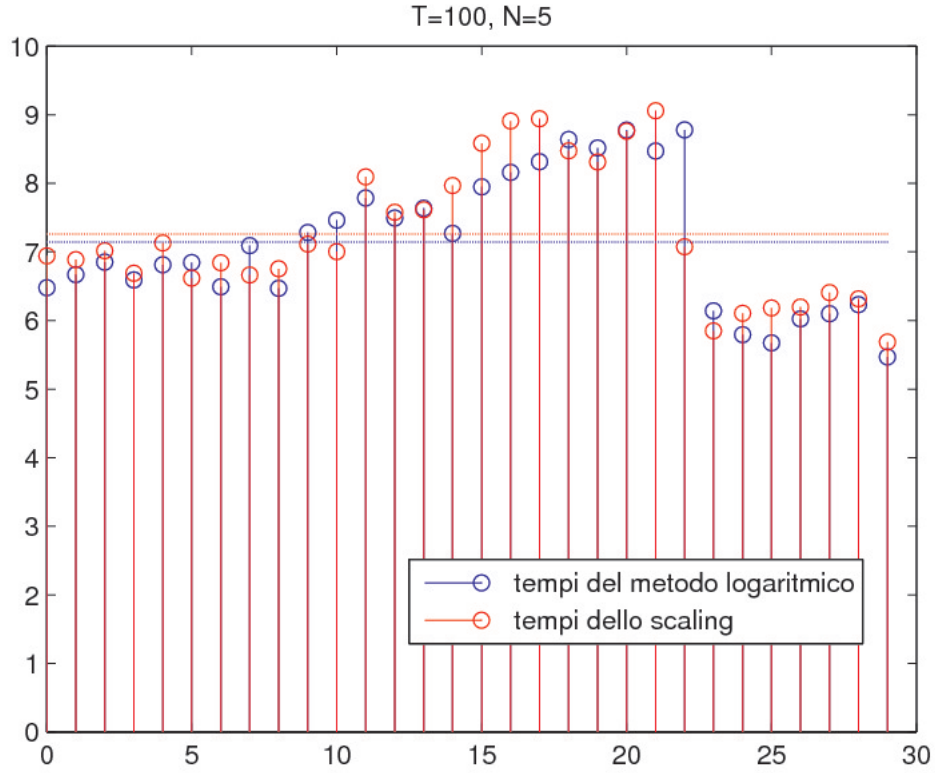
```
        end
71      end

73      logP=-Inf
        for i=1:N
75          logP=elnsum(logP,LOGALPHA(T,i))
        end
77                  %fine metodo logaritmico

79      TL(p)=toc(STOP)
        stop=tic
81
            %inizio scaling
83      alpha=zeros(T,N)
        c=zeros(1,T)
85      s=0
        for i=1:N
87          alpha(1,i)=PI(i)*B(i,O(1))
            s=s+alpha(1,i)
89      end
        c(1)=1/s
91      for i=1:N
            alpha(1,i)=alpha(1,i)*c(1)
93      end
        for t=1:T-1
95          s=0
            for j=1:N
97              for i=1:N
                    alpha(t+1,j)=alpha(t+1,j)+(alpha(t,i)*A(i,j))
99              end
                alpha(t+1,j)=alpha(t+1,j)*B(j,O(t+1))
101             s=s+alpha(t+1,j)
            end
103         c(t+1)=1/s
            for j=1:N
105             alpha(t+1,j)=alpha(t+1,j)*c(t+1)
            end
```

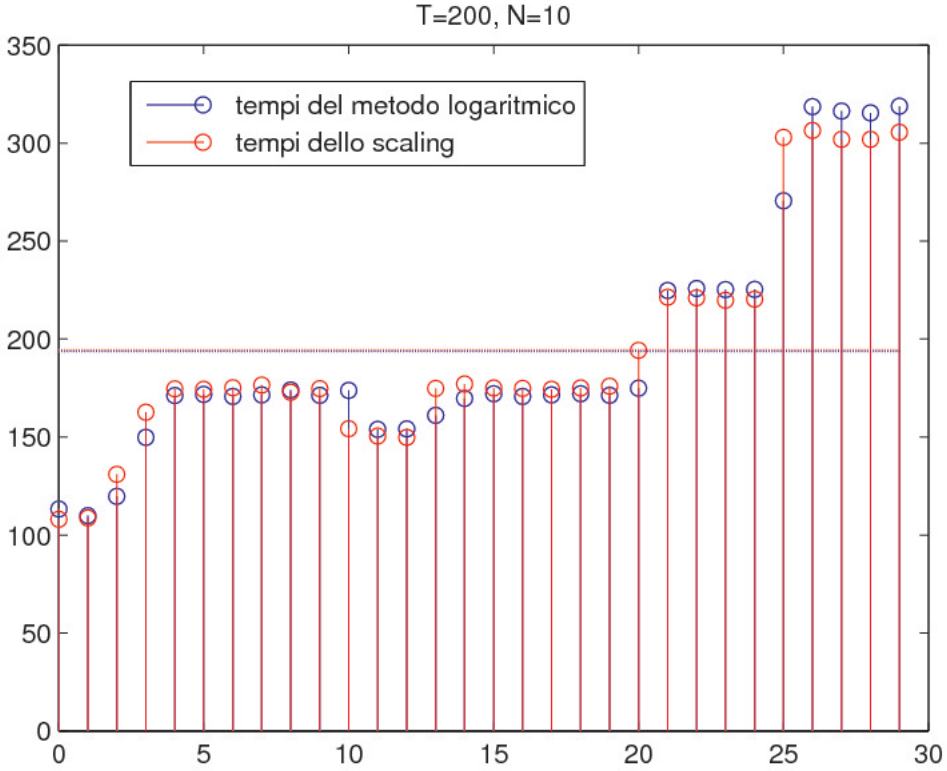
```
107     end

109     LOGP=0
        for t=1:T
111         LOGP=LOGP-log(c(t))
        end
113
        TS(p)=toc(stop)
115     %fine scaling
    end
117


119 stem(x,TL)
    hold on
121 stem(x,TS,'r')
    hold on
123 plot(y,sum(TL)/P,'b')
    hold on
125 plot(y,sum(TS)/P,'r')
    legend('tempi del metodo logaritmico','tempi dello scaling')
```

I grafici derivati vengono riportati di seguito.


T=10, N=20
tempi del metodo logaritmico
tempi dello scaling
0
0.5
1
1.5
2
2.5
3
3.5
4
0
5
10
15
20
25
30


T=50, N=10
tempi del metodo logaritmico
tempi dello scaling
0
2
4
6
8
10
12
14
0
5
10
15
20
25
30


T=100, N=5
tempi del metodo logaritmico
tempi dello scaling
0
1
2
3
4
5
6
7
8
9
10
0
5
10
15
20
25
30


T=200, N=10
tempi del metodo logaritmico
tempi dello scaling
0
50
100
150
200
250
300
350
0
5
10
15
20
25
30

Tali grafici sono stati ottenuti modificando di volta in volta i parametri N e T, conformemente a quanto riportato nei rispettivi titoli, e ripetendo ogni simulazione trenta volte con osservazioni casuali sempre differenti. La linea orizzontale tratteggiata rappresenta la media sulle trenta ripetizioni. Quello che si riscontra è una sostanziale comparabilità dei tempi per i due metodi, laddove era lecito aspettarsi un vantaggio apprezzabile per il metodo dello scaling.
L'utilizzo di un linguaggio di alto livello come quello di matlab rende l'analisi condotta poco significativa poiché l'utente non ha controllo sulla specifica implementazione del codice e sulla sua esecuzione. Uno studio più approfondito potrebbe essere effettuato in ambienti il più vicini possibile al linguaggio macchina, ma tale approfondimento esula dai fini di questa trattazione.

Il metodo logarimico presenta anche dei piccoli vantaggi. Il codice è di semplice impletmentazione, non richiede artifici algebrici (moltiplicazione per coefficienti opportuni) e consente di mantenere un saldo controllo sulle variabili in gioco le quali sono semplicemente i logaritmi di probabilità ben definite. Lo scaling di contro è un metodo più artificioso e la sua adozione causa la perdita dell'intuizione per quanto concerne le variabili scalate $\hat{\alpha}_t(i)$ e $\hat{\beta}_t(i)$. Il metodo logaritmico, nonstante il teorico svantaggio da un punto di vista computazionale, costituisce dunque una valida alternativa allo scaling.

Bibliografia

[1] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. 1998.

[2] B.H. Juang and L.R. Rabiner. Hidden markov models for speech recognition. *IEEE*, 33:251–272, August 1991.

[3] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *THE BELL SYSTEM TECHNICAL JOURNAL*, 62(4), 1983.

[4] T.P. Mann. *Numerically Stable Hidden Markov Model Implementation*, February 2006.

[5] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *IEEE*, 77, 1989.