Università degli Studi di Siena

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Analisi di sequenze di aminoacidi mediante modelli di Markov nascosti:

Previsione dello Stato di Legame delle Cisteine

Relatore:
Prof. Marco Maggini
Correlatore:
Dott. Edmondo Trentin

Candidato:
Guido Bartoli

Anno accademico 2002–2003

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

La *bioinformatica* ha guadagnato al giorno d'oggi grande popolarità in vari settori, sia nell'ambito scientifico, sia nella nostra quotidianità, grazie soprattutto ad eventi ampiamente pubblicizzati, come la mappatura del genoma umano o la clonazione degli animali; inoltre, in virtù dei più recenti sviluppi tecnologici, le capacità di calcolo dei moderni elaboratori sono aumentate in maniera esponenziale e molti progetti che non potevano nemmeno essere affrontati con i metodi tradizionali, possono ora essere risolti o almeno semplificati utilizzando software e hardware dedicati. In un progetto come quello del genoma umano, ad esempio, un ruolo chiave è stato svolto dai computer multiprocessore e dai clusters, affiancati da sofisticati sistemi di archiviazione di massa.

Fra le varie applicazioni della *bioinformatica*, esiste anche quella relativamente recente dell'analisi di sequenze di aminoacidi, e più in generale di classi di proteine. In questo contesto, si inserisce un progetto ministeriale curato da un gruppo di biologi ed informatici di Firenze e dal dott. Trentin per conto dell'Università degli Studi di Siena.

L'obiettivo principale di questo progetto è quello di realizzare un sistema che, una volta addestrato su opportuni esempi già classificati, riesca a prevedere la formazione di ponti fra coppie di cisteine all'interno di una sequenza: ciò può contribuire in maniera significativa a valutare la struttura secondaria della proteina, ed indirettamente anche quella terziaria (spaziale). Da un punto di vista biologico, analizzando la presenza di questi legami, è possibile anche dedurre se l'RNA che verrà prodotto potrà dare origine a cellule anomale, come ad esempio quelle cancerogene.

Il progetto parte dal presupposto che in natura esista una qualche "regola" che determina la formazione di legami in base agli aminoacidi presenti nelle proteine; regola che non sembra però essere derivabile con metodi logici, ma solo interpretabile su base probabilistica. Per questo motivo si utilizza una collezione di proteine nella quale biologi specialisti hanno individuato sperimentalmente i legami della cisteina: basandosi sulla maggior parte di questi esempi (*training set*, circa 96% del totale nel nostro caso), il sistema proposto tenta di apprendere, con un approccio tipico dell'*Intelligenza Artificiale*, ciò che mette in relazione le strutture delle proteine alla formazione dei ponti di cisteina; i parametri di valutazione acquisiti vengono poi applicati ai rimanenti esempi (*test set*) per verificarne l'efficienza confrontando le sequenze riconosciute con quelle note del *test set* stesso.

In sintesi, si è cercato di valutare le prestazioni di un riconoscitore basato su HMM applicandolo ad un problema di bioinformatica fino ad ora affrontato con altri metodi.

1 Modelli di Markov nascosti

I modelli di Markov si sono rivelati uno strumento consono ad affrontare questo progetto, in quanto un approccio tradizionale mediante un classificatore a reti neurali non avrebbe garantito buoni risultati su lunghe sequenze di dati; i classificatori basati su HMM, "dilatandosi" e "contraendosi" all'occorrenza, sono più adatti a catturare le caratteristiche statistiche di un processo che varia considerevolmente nel tempo.

Prima di passare alla descrizione del materiale a disposizione e della modalità operativa scelta, qui di seguito è riportata un'introduzione teorica ai modelli di Markov nascosti.

1.1 Calcolo di probabilità con HMM

L'adozione di un sistema che usi uno o più modelli di Markov permette di affrontare tre problematiche principali:

- trovare la probabilità di occorrenza di una sequenza a partire da un modello (*stima*)
- identificare la sequenza di stati che più probabilmente ha generato una certa sequenza (*decodifica*)
- generare un HMM basandosi su una sequenza di osservazioni (*apprendimento*)

I modelli di Markov rappresentano un'estensione delle catene di Markov, che costituiscono il più semplice modello in cui la probabilità di un evento dipende solo dall'evento precedente; una catena $\{X_i\}_{i \in I \subseteq \mathbb{N}}$ è un processo stocastico discreto, che si dice "di Markov" se:

$$\mathrm{P}(X_n = i_n \mid X_{n-1} = i_{n-1}, \ldots, X_1 = i_1) = \mathrm{P}(X_n = i_n \mid X_{n-1} = i_{n-1}) \ (\forall n \in I)$$

Se gli stati sono in un numero finito, una catena di Markov può essere rappresentata nella forma di un automa stocastico, ossia di un grafo (in generale completo) i cui nodi sono etichettati con gli stati (o con gli eventi corrispondenti), e i cui archi sono etichettati con le probabilità di passare allo stato successivo.

In Figura 1 è riportato un esempio di automa stocastico, in cui ogni cammino genera stringhe di DNA composte dagli amminoacidi A,C,G,T: un automa di questo tipo è detto *generativo*. Usando le probabilità che etichettano gli archi nel cammino, la proprietà di Markov e la formula della probabilità composta, è possibile calcolare nel modo seguente la probabilità di occorrenza

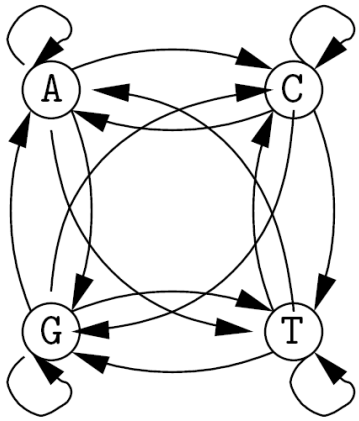

Figura 1: Un automa stocastico che genera sequenze di DNA. A ciascun arco del grafo è associata una probabilità di transizione (non mostrata in figura).

di una sequenza $x = x_1, \ldots, x_n$: per s, t appartenenti all'insieme dei simboli (gli aminoacidi A,C,T,G in questo caso) e posto $a_{s,t} = \mathrm{P}(x_i = t \mid x_{i-1} = s)$, si ha che

$$\begin{aligned} \mathrm{P}(x) &= \mathrm{P}(x_1, x_2, \ldots, x_n) \\ &= \mathrm{P}(x_n \mid x_{n-1}, \ldots, x_1) \cdot \mathrm{P}(x_{n-1} \mid x_{n-2}, \ldots, x_1) \cdots \mathrm{P}(x_1) \\ &= \mathrm{P}(x_n \mid x_{n-1}) \cdot \mathrm{P}(x_{n-1} \mid x_{n-2}) \cdots \mathrm{P}(x_1) \\ &= \mathrm{P}(x_1) \cdot \prod_{i=2}^{n} a_{x_{i-1}, x_i} \end{aligned} \tag{1}$$

In una catena di Markov c'è una corrispondenza biiettiva tra i simboli emessi dall'automa e gli stati corrispondenti; in un modello di Markov nascosto, invece, non è così: gli stati interni sono *nascosti* e l'osservatore può accedere soltanto ad una sequenza di simboli *osservabili*, in base alla quale egli può ricavare la probabilità degli stati corrispondenti.

Riassumendo, un modello di Markov nascosto è costituito da una catena di Markov *nascosta* (stati interni) e da un processo *osservabile* (sequenza di simboli), descritto mediante funzioni probabilistiche.

Facciamo un esempio: se un'eremita naufragato su un'isola volesse fare delle previsioni metereologiche basandosi solo sull'analisi di alghe trovate in mare, egli sarebbe costretto a fare delle inferenze sullo stato delle alghe per valutare se il tempo cambierà, perché l'esperienza suggerisce che questi due fenomeni sono probabilisticamente legati tra loro. In questo caso, i gruppi di stati sono due: quelli *osservabili* (lo stato delle alghe) e quelli *nascosti*

(lo stato del tempo atmosferico). I modelli di Markov permettono, dopo un'opportuna fase di training e di test, di risolvere questo problema senza necessariamente implementare un algoritmo specifico per ogni caso distinto.

Più rigorosamente, un modello di Markov nascosto (*Hidden Markov Model*, HMM) è definito da:

- un insieme finito $S = \{1, 2, \ldots, m\}$ di stati che il modello può assumere nel tempo;
- una distribuzione di probabilità condizionata $A = \{a_{i,j}\}$, che specifica, per ogni $k, l \in S$, la probabilità di passare dallo stato k allo stato l:

  $$a_{k,l} = \mathrm{P}(\pi_i = l \mid \pi_{i-1} = k)$$

  Per ogni $k \in S$, $a_{0,k}$ è la probabilità che k sia lo stato iniziale, mentre $a_{k,0}$ è la probabilità che sia quello finale;
- un insieme Σ di simboli osservabili;
- una distribuzione di probabilità condizionata $B = \{b_{i,j}\}$ che specifica, per ogni $b \in \Sigma$ e per ogni $k \in S$, la probabilità di osservare l'emissione del simbolo b a partire dallo stato k. Tale probabilità è detta *emission probability*: $e_k(b) = \mathrm{P}(x_i = b \mid \pi_i = k)$

La Figura 2 mostra un possibile HMM che genera sequenze in base ai simboli A,C,T,G.

1.2 Algoritmi di decodifica

Dato quindi un HMM, la probabilità congiunta di osservare una successione di simboli $x = x_1, \ldots, x_n$ generata da una sequenza di stati $\pi = \pi_1, \ldots, \pi_n$ è

$$\mathrm{P}(x, \pi) = a_{0,\pi_1} \cdot \prod_{i=1}^{n} e_{\pi_i}(x_i) \cdot a_{\pi_i,\pi_{i+1}}$$

dove per convenzione si pone $\pi_{n+1} = 0$.

Purtroppo nella maggior parte dei casi la sequenza degli stati attraversati non è nota a priori; ricavare il "cammino" nell'HMM che ha portato alla generazione di una certa stringa si può ridurre semplicemente alla determinazione di quello più probabile (*most probable path*, MPP), vale a dire alla sequenza di stati

$$\pi^* = \arg\max_{\pi} \mathrm{P}(x, \pi)$$

che massimizza la probabilità delle varie transizioni.

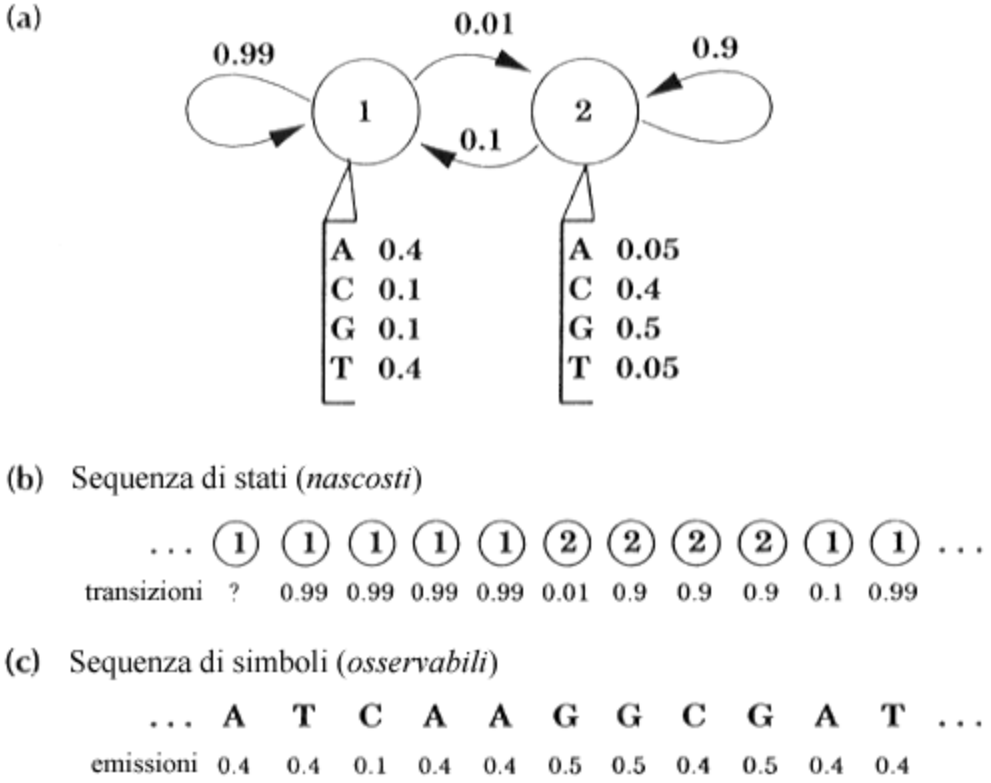

Figura 2: Esempio di automa stocastico con associate probabilità di transizione e di emissione (in questo caso $\Sigma = \{A, C, T, G\}$).

La proprietà delle catene di Markov permette di calcolare π^* mediante un algoritmo di programmazione dinamica, noto come ALGORITMO DI VITERBI.

Detta $v_k(i)$ la probabilità che il MPP termini nello stato k alla posizione i, i passi di questo procedimento possono essere calcolati ricorsivamente come segue:

$$\begin{aligned} v_0(0) &= 1 \\ v_l(i+1) &= e_l(x_{i+1}) \cdot \max_k \left(v_k(i) \cdot a_{k,l}\right) \end{aligned}$$

Mantenendo un puntatore $\mathrm{ptr}_i(l)$ allo stato precedente per ogni stato l e posizione i attuali, è possibile ricostruire all'indietro il cammino seguito. L'ALGORITMO DI VITERBI si compone dunque dei seguenti passi:

1. Inizializzazione: $v_0(1) = 1$, $v_k(0) = 0$, per $k > 0$
2. Passo di ricorsione:

$$\begin{aligned} v_l(i) &= e_l(x_i) \cdot \max_k \left(v_k(i-1) \cdot a_{k,l}\right) \\ \mathrm{ptr}_i(l) &= \arg\max_k \left(v_k(i-1) \cdot a_{k,l}\right) \end{aligned}$$

3. Terminazione:

$$\begin{aligned} \mathrm{P}(x, \pi^*) &= \max_k \left(v_k(n) \cdot a_{k,0}\right) \\ \pi^*_n &= \arg\max_k \left(v_k(n) \cdot a_{k,0}\right) \end{aligned}$$

4. Traceback:

$$\pi^*_{i-1} = \mathrm{ptr}_i(\pi^*_i)$$

La probabilità di occorrenza di una stringa x, indipendentemente dal cammino, può essere calcolata come la somma delle $\mathrm{P}(x, \pi)$ su tutti i cammini:

$$\mathrm{P}(x) = \sum_\pi \mathrm{P}(x, \pi)$$

In Figura 3 è riportato un esempio di decodifica eseguita mediante l'algoritmo di Viterbi su una sequenza di 4 simboli con un HMM a 4 stati.

Disporre del valore di $\mathrm{P}(x)$ equivale al calcolo della formula (1) applicata alle catene di Markov; tuttavia, il numero di possibili cammini cresce in esponenzialmente con la lunghezza della sequenza. Una soluzione al problema consiste nell'ignorare tutti i cammini tranne il MPP, e quindi definire $\mathrm{P}(x) = \mathrm{P}(x, \pi^*)$.

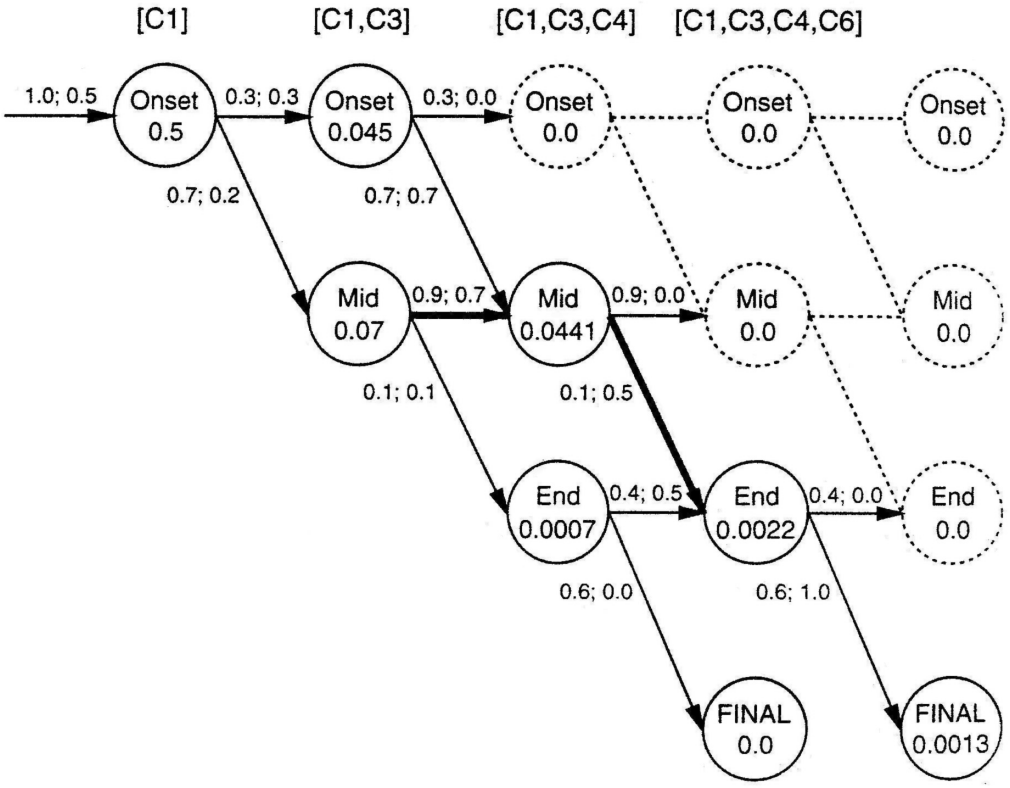

Figura 3: Esempio di ALGORITMO DI VITERBI (*decodifica*). In neretto è evidenziato il percorso più probabile (MPP).

Nella pratica, quest'approssimazione dà spesso buoni risultati; non è però necessario ricorrervi sempre, dato che, sfruttando la proprietà di Markov, è possibile progettare un altro algoritmo di programmazione dinamica simile all'ALGORITMO DI VITERBI.

Sia $f_k(i) = \mathrm{P}(x_1, \ldots, x_i, \pi_i = k)$ la probabilità di osservare la (sotto)sequenza $x_1, \ldots, x_i$ e di trovarsi nello stato k; l'algoritmo seguente, detto "in avanti" (FORWARD ALGORITHM), permette quindi di calcolare ricorsivamente la probabilità di $x = x_1, \ldots, x_n$: i passi dell'algoritmo sono i seguenti:

1. Inizializzazione: $f_0(0) = 1$, $f_k(0) = 0$, per $k > 0$
2. Passo di ricorsione: $f_l(i) = e_l(x_i) \cdot \sum_k \left(f_k(i-1) \cdot a_{k,l}\right)$
3. Terminazione: $\mathrm{P}(x) = \sum_k \left(f_k(n) \cdot a_{k,0}\right)$

È naturale porre anche il problema inverso: nota la sequenza x, qual'è la probabilità $\mathrm{P}(\pi_i = k \mid x)$ che l'HMM sia nello stato k al tempo i? Anche questo valore può essere ottenuto ricorsivamente partendo dalla condizione $\mathrm{P}(x, \pi_i = k)$ e applicando la formula della probabilità congiunta e la proprietà di Markov:

$$\begin{aligned} \mathrm{P}(x, \pi_i = k) &= \mathrm{P}(x_1, \ldots, x_i, \pi_i = k) \cdot \mathrm{P}(x_{i+1}, \ldots, x_n \mid x_1, \ldots, x_i, \pi_i = k) \\ &= \mathrm{P}(x_1, \ldots, x_i, \pi_i = k) \cdot \mathrm{P}(x_{i+1}, \ldots, x_n \mid \pi_i = k) \\ &= f_k(i) \cdot b_k(i) \end{aligned}$$

avendo posto $b_k(i) = \mathrm{P}(x_{i+1}, \ldots, x_n \mid \pi_i = k)$. Il calcolo ricorsivo all'indietro di $b_k(i)$ permette di calcolare il valore di $\mathrm{P}(\pi_i = k \mid x)$: infatti, applicando la definizione di probabilità condizionata, si ha:

$$\mathrm{P}(\pi_i = k \mid x) = \frac{f_k(i) \cdot b_k(i)}{\mathrm{P}(x)}$$

La probabilità $\mathrm{P}(x)$ può essere calcolata usando il suddetto algoritmo "in avanti", mentre i passi dell'algoritmo "all'indietro" (BACKWARD ALGORITHM) per il calcolo di $b_k(i)$ sono i seguenti:

1. Inizializzazione: $b_k(n) = a_{k,0}$, per ogni k
2. Passo di ricorsione: $b_k(i) = \sum_l a_{k,l} \cdot e_l(x_{i+1}) \cdot b_l(i+1)$

1.3 Stima dei parametri di un HMM

Due dei problemi più difficili legati all'utilizzo degli HMM sono la determinazione della topologia del modello (gli stati di cui è composto), e l'attribuzione dei valori alle probabilità di transizione e di emissione. Assumendo che la topologia del modello sia già stata correttamente individuata, possiamo affrontare anche il secondo problema sfruttando il calcolo probabilistico.

Supponiamo che sia disponibile un campione di sequenze indipendenti (*training set*) del tipo che vogliamo modellare con uno o più HMM; se i cammini associati alle sequenze sono noti, è possibile adottare un metodo di massima verosimiglianza per correggere i valori degli $a_{k,l}$ e $e_k(b)$.

Sia $A_{k,l}$ il numero di volte in cui si passa dallo stato k allo stato l all'interno del campione e sia $E_k(b)$ il numero di volte in cui si osserva l'emissione del simbolo b nello stato k, sempre all'interno del campione: per procedere si pone

$$a_{k,l} = \frac{A_{k,l}}{\sum_{l'} A_{k,l'}} \quad \text{e} \quad e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')} \tag{2}$$

Questo metodo di stima è chiaramente sensibile alla distribuzione statistica e alla dimensione del campione; inoltre, se un certo stato non si presenta mai nel campione, le relative equazioni restano indefinite. A quest'ultimo difetto si rimedia utilizzando valori precalcolati (*pseudocount*) ad $A_{k,l}$ e $E_k(b)$.

Vi sono casi in cui anche le sequenze di stati associate al campione non sono note a priori; allora, la stima dei parametri richiede l'uso di un metodo iterativo. A questo scopo, la procedura standard per gli HMM è l'algoritmo di Baum-Welch.

Questo algoritmo stima $A_{k,l}$ e $E_k(b)$ considerando i vari MPP relativi al training set, calcolati usando i valori attuali di $a_{k,l}$ e $e_k(b)$; successivamente si usano le formule (2) per derivare nuovi valori per $a_{k,l}$ e $e_k(b)$. Si può dimostrare che ad ogni iterazione la log-verosimiglianza del modello aumenta e il procedimento converge ad un massimo locale; di solito si presentano più massimi locali, cosicché il valore che si raggiunge dipende fortemente dai valori iniziali attribuiti ai parametri.

1.4 Gli HMM applicati alle sequenze di aminoacidi

Una volta presa visione della teoria necessaria ad affrontare il problema del classificatore, una prima impostazione degli HMM è stata ideata in questo modo:

- L'insieme dei simboli osservabili è stato fissato a 20, cioè il numero totale degli aminoacidi di base[1];
- Due modelli sono stati inseriti per le cisteine che fanno o meno ponte (indicati rispettivamente con una `P` ed una `N`);
- Un HMM aggiuntivo (`@bg`) è stato inserito per modellare il "rumore di fondo" presente nelle sequenze[2]: ciò ha permesso di incrementare notevolmente le prestazioni, in quanto il classificatore in questo modo è molto più flessibile e distingue meglio sequenze di cisteine a grande distanza l'una dall'altra;
- I valori di partenza delle probabilità degli HMM vengono calcolati nella fase di inizializzazione mediante la procedura di *segmental K-Means*;
- Le iterazioni di inizializzazione sono basate sull'intero training set;
- Per la decodifica è stato scelto l'ALGORITMO DI VITERBI;
- In fase di decodifica sono state abilitate le transizioni al modello `@bg`;
- Alla fine dell'elaborazione, un programma aggiuntivo fornisce il rapporto dettagliato dei parametri immessi, delle stringhe originali presenti nel test set, di quelle riconosciute dagli HMM e dei percorsi seguiti; fornisce inoltre le percentuali di riconoscimento delle sequenze e dei modelli, distinguendo gli errori in inserimenti, cancellazioni e sostituzioni.

  La seguente tabella mostra il significato di questi tipi di errore prendendo come esempio da riconoscere la sequenza `P P N N P P`:

Stringa originale	Stringa riconosciuta	ERRORI
`P P N N P P`	`P P N N`	2 cancellazioni
`P P N N P P`	`P P N N P P N`	1 inserimento
`P P N N P P`	`P N N P P N`	3 sostituzioni

Tabella 1: Questa tabella contiene i tipi di errore possibili che il simulatore può commettere in fase di test.

[1] Vedi Appendice per la nomenclatura.

[2] Il "rumore di fondo" è costituito da quei simboli (aminoacidi) che non influenzano significativamente il riconoscimento.

Escludendo il modello del rumore composto da un unico stato, il numero di stati associati agli altri due modelli non è stato fissato a priori, ma bensì variato nel corso degli esperimenti per vedere quale topologia avrebbe ottenuto la migliore percentuale di riconoscimento.

Similarmente sono stati variati anche i seguenti parametri, appartenenti alle fasi di training e di test (tra parentesi è indicato il range di variazione):

1. Training
    - numero di iterazioni per l'inizializzazione (da 1 a 3);
    - numero delle misture di gaussiane associate alle probabilità di transizione (da 4 a 10);
    - numero di epoche globali di training (da 2 a 8).
2. Test
    - probabilità di transizione (da 10^{-30} a 10^{-1}): essa indica la facilità con cui si può passare da un modello all'altro mentre si tenta di decodificare la stringa di test[3].

Come per l'ASR (vedi [8]) in cui ad ogni fonema è solitamente associato un HMM, il simulatore addestra contemporaneamente più modelli in modo da scegliere in fase di test quello che presenta la probabilità maggiore.
Per l'elenco completo dei parametri, consultare le Tabelle 3 e 4.

Dopo queste premesse, è possibile esporre e commentare gli esperimenti ed i risultati ottenuti con il dataset.

2 Sperimentazioni

Un aspetto rilevante della tesi è stato quello delle sperimentazioni eseguite sul dataset con vari programmi appositamente scritti, tra cui un generatore di liste per adattare i dati al formato necessario al simulatore di HMM (usato in precedenza per applicazioni di *Automatic Speech Recognition*, ASR (vedi [8])).

L'obiettivo è quello di eseguire una serie sufficiente di tests al fine di trovare i parametri in grado di fornire risultati soddisfacenti.

[3] Un valore eccessivo darà origine ad una sovrabbondanza di modelli riconosciuti, mentre uno troppo basso ne individuerà meno del necessario.

2.1 Il dataset

Prima di esaminare in dettaglio il lavoro svolto ed i risultati ottenuti, qui di seguito è riportata la struttura generale del dataset fornito dai biologi per questo progetto, che consiste in un archivio di circa 35 MBytes organizzato in cartelle:

- La cartella `ProfilesNonNan` contiene i files dei *profili* sintetici (circa 960) di ogni famiglia di proteine[4]; ognuno di essi dà una visione d'insieme delle caratteristiche di ogni sequenza. In ciascun file (`10mhA.seq`, `153l_.seq`, ecc...) è presente una lista degli aminoacidi componenti la proteina corrispondente, mostrando, per ogni cisteina (`C`), un numero indicante la possibilità di formazione del legame (`1` o `2` rappresenta un legame con altre cisteine, mentre `-1` indica che nessun legame è possibile)[5].
  Questa lista è un "riassunto" della struttura di una proteina, in quanto vi sono presenti solo gli aminoacidi con probabilità maggiore, che sono semplici riferimenti e non verrano utilizzati dagli HMM che lavorano sulle probabilità, come vedremo in seguito.

- La cartella `Profiles` contiene invece la versione estesa dei profili sopracitati[6]; stavolta per ogni proteina esiste un file con una matrice contentente la probabilità con cui ogni aminoacido è presente nelle varie posizioni della sequenza: le 20 colonne indicano gli aminoacidi[7], mentre le righe sono le posizioni all'interno della proteina (di numero variabile, con totale per ogni riga uguale a 1)[8].
  Nella Figura 4 a pagina 13 è riportata la parte di una matrice di probabilità (relativa alla proteina `1b0pa`, con indicato a fianco il numero delle righe), che racchiude il maggior numero possibile di informazioni ricavabili da una proteina. Per l'esempio di Figura 4, il corrispondente file `1b0pa.seq` dei profili sintetici sarebbe strutturato come mostrato in Figura 5, sempre a pagina 13.

- I files della cartella `AA` sono identici a quelli di `ProfilesNonNan`, tranne per il fatto che non contengono la colonna con i legami della cisteina.

[4]Per comodità, da qui in avanti si farà riferimento ad una famiglia di proteine direttamente con il nome di "proteina".

[5]Indica un altro tipo di legame, anche se non utile ai fini della sperimentazione.

[6]In questo caso si tratta del risultato statistico degli *allineamenti multipli* eseguiti dai biologi, in cui un gran numero di proteine viene raggruppato in classi.

[7]Per la tabella dei codici degli aminoacidi consultare l'Appendice.

[8]Per la proprietà $\sum_{i=1}^{N} \mathrm{P}(x_i) = 1$.

```
     D    E    A    R    N    C    F    G    Q    I    H    L
01) 0.04 0.10 0.00 0.74 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ....
02) 0.20 0.11 0.14 0.11 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ....
03) 0.00 0.00 0.00 0.03 0.00 0.00 0.00 0.00 0.09 0.09 0.12 ....
04) 0.11 0.05 0.23 0.37 0.00 0.00 0.00 0.05 0.00 0.00 0.03 ....
05) 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 ....
06) 0.00 0.00 0.00 0.00 0.00 0.96 0.00 0.00 0.02 0.00 0.00 ....
07) 0.00 0.00 0.00 0.00 0.35 0.00 0.00 0.00 0.03 0.00 0.03 ....
08) 0.02 0.00 0.02 0.02 0.00 0.00 0.06 0.05 0.12 0.00 0.03 ....
09) 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.03 0.00 0.97 0.00 ....
10) 0.25 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.53 0.07 ....
11) 0.00 0.00 0.00 0.00 0.00 0.71 0.00 0.02 0.06 0.00 0.11 ....
12) 0.00 0.00 0.00 0.02 0.00 0.07 0.25 0.02 0.10 0.00 0.12 ....
13) 0.00 0.00 0.00 0.00 0.00 0.06 0.00 0.02 0.71 0.00 0.11 ....
14) 0.00 0.73 0.02 0.02 0.00 0.21 0.00 0.00 0.00 0.00 0.00 ....
15) 0.00 0.22 0.10 0.22 0.30 0.00 0.01 0.00 0.00 0.00 0.12 ....
16) .... .... .... .... .... .... .... .... .... .... .... ....
```

Figura 4: Esempio di matrice di probabilità ricavata dagli allineamenti multipli di una classe di proteine.

```
01) R  0
02) D  0
03) H  0
04) R  0
05) D  0
06) C  1
07) N  0
08) Q  0
09) I  0
10) I  0
11) C  1
12) F  0
13) Q  0
14) E  0
15) N  0
16) .. ..
```

Figura 5: Profilo sintetico ricavato dalla matrice di probabilità di Figura 4; in questo caso, due cisteine della proteina fanno ponte (`1`) tra di loro.

- Le cartelle numerate da `0` a `19` contengono 20 esperimenti diversi già impostati all'interno del dataset. Ogni esperimento è costituito da due files, uno per il training (`sequence-list.train`) ed uno per il test (`sequence-list.test`): ognuno contiene una lista delle proteine che il simulatore dovrà analizzare (circa 860 per il training e 30 per il test), con indicato a fianco il numero delle cisteine presenti.
  Per diminuire i tempi di elaborazione, è stato utilizzato solo il primo (lo `0`), e quindi un solo training set ed un solo test set; come si dirà nelle conclusioni, gli altri esperimenti potranno essere inclusi in un secondo momento per ulteriori raffinamenti.

- Esistono anche altre due versioni degli esperimenti (`Descriptor24` e `Descriptor29`) che includono un *descrittore* aggiuntivo, utile per migliorare le prestazioni del riconoscitore; allo stato attuale della tesi, quest'informazione non è stata ancora sfruttata, per dedicare più attenzione alla valutazione del solo modello statistico applicato ai profili.

2.2 Il generatore di liste continue

Il simulatore fornito per questa tesi è stato precedentemente utilizzato per l'ASR (vedi [8]), e si è reso quindi necessario adattare il dataset delle proteine a quel formato: a questo scopo è stato scritto un programma in linguaggio C che produce come output i files `list.train` e `list.test`.

Entrambi i files sono strutturati come mostrato in Figura 6 (a sinistra è indicato il numero della riga, mentre a destra c'è il commento).

```
01 Nome del primo profilo          (informativo,non vincolante)
02 Modelli da riconoscere          (Es: @bg P @bg N @bg ...)
03 Numero di righe della matrice   (45 in questo caso)
04 Numero di colonne della matrice (20)
05  /------------------------\
..  | Matrice di probabilità |     (di lunghezza variabile)
50  \------------------------/
51 Nome del secondo profilo        (informativo,non vincolante)
52 ..............................
.. ..............................
```

Figura 6: Formato dei files di training e di test

Il generatore di liste continue, sinteticamente, opera in questo modo[9]:

1. Lettura della prima riga dal file di training (`sequence-list.train`) e scrittura del nome della proteina corrispondente in output.

2. Elaborazione del relativo profilo sintetico per ricavare il numero delle cisteine presenti, i cui relativi modelli (`P` o `N`) vengono riportati nelle righe successive dell'output.
   Se il simulatore, oltre ai modelli di ponte e non ponte, usa anche un HMM associato al rumore (`@bg`), questo verrà intercalato alle `P` e alle `N` nella lista, come mostrato nel listato precedente.

3. Calcolo delle dimensioni della matrice (numero di righe e colonne) associata al profilo e scrittura sull'output.

4. Caricamento della matrice dalla cartella `Profiles` ed "append" nel file di train.

5. Il ciclo prosegue con le righe successive della sequenza di train continuando dal punto 2 fino all'ultima proteina della lista.

L'intera procedura viene poi ripetuta per generare anche la lista di test (input: `sequence-list.test` $\longrightarrow$ output: `list.test`).

Con questo procedimento sono stati costruiti un training ed un test set ripuliti da incoerenze[10] composti rispettivamente da 750 e 26 proteine.

2.3 Il simulatore di HMM

Per regolare i parametri di training e di test, si interviene manualmente su uno script di shell (`experiment.sh`) che provvede a passarli ai programmi lanciati durante la simulazione.

Al fine di ottimizzare i tempi di alcuni esperimenti e valutare meglio l'influenza di certi parametri, a volte si è rivelato utile usare un raffinamento progressivo di modelli già addestrati (mediante il parametro `ModelsInFile`), oppure eseguire solo la fase di test commentando le righe di training in `experiment.sh`.

I programmi sono stati interamente progettati e compilati in ambiente `Linux` e la loro descrizione è riportata nella Tabella 2.

[9] Tutti gli output sono indirizzati in modo sequenziale sul file `list.train`.
[10] Profili vuoti o con valori errati.

SIMULAZIONE

FASE	PROGRAMMA	DESCRIZIONE
Training	`TrainHMM`	Addestra i modelli HMM in base ai parametri immessi a riga di comando
Test	`demoRec`	Esegue la decodifica di Viterbi
	`TextGenerator`	Ricava le stringhe a partire dai MPP
	`WER.sh`	Script che confronta le sequenze di modelli riconosciuti con quelle del *test set*
	`CheckString`	Programma invocato da `WER.sh`
Training e **Test**	`etrace`	Registra il tracciato delle operazioni eseguite negli esperimenti (HISTORY) includendo risultati parziali e globali

Tabella 2: Elenco dei programmi usati per la simulazione

Qui di seguito è riportata la sequenza dei comandi di `experiment.sh`:

1. Prototipi (funzioni `HMM_Training()` e `Viterbi_Decoding()`) dei parametri a riga di comando da passare ai rispettivi programmi;
2. Settaggio dei valori dei parametri relativi alla fase di *training*; (per l'elenco completo, consultare la Tabella 3)
3. Esecuzione di `TrainHMM` mediante la funzione `HMM_Training()`;
4. Settaggio dei valori dei parametri relativi alla fase di *test*; (per l'elenco completo, consultare la Tabella 4)
5. Esecuzione di `demoRec` mediante la funzione `Viterbi_Decoding()`;
6. Esecuzione di `TextGenerator`;
7. Esecuzione di `WER.sh`;
8. Esecuzione di `etrace`.

2.4 Parametri della simulazione

Le Tabelle 3 e 4 contengono la descrizione completa dei parametri usati rispettivamente per il training ed il test[11].

[11] Un asterisco contrassegna i valori modificati durante gli esperimenti. Per ulteriori informazioni, vedere il § 1.4.

TRAINING

PARAMETRO	DESCRIZIONE
`nModels`	numero di HMM da addestrare
`ModelsNamesFile`*	file contenente la topologia degli HMM
`ModelsInFile`	`TrainHMM` può opzionalmente partire da HMM già addestrati per affinarne i parametri interni
`ModelsOutFiles`	indica in quali files memorizzare i modelli una volta addestrati durante la simulazione
`WholeModelOutFile`	
`nComponents`*	numero di misture di gaussiane da usare per le probabilità interne degli HMM
`nEpochs`*	numero di iterazioni globali
`nObservations`	numero di proteine presenti nel training set
`TrainingFile`	specifica il file contenente il training set
`NormalizationFile`	file opzionale usato per normalizzare i valori
`maxFileIdLen`	lunghezza massima del nome dei profili
`maxTextStringLen`	lunghezza massima delle sequenze di aminoacidi
`maxTranscriptionLen`	lunghezza massima delle trascrizioni
`maxUnitNameLen`	lunghezza massima del nome dei modelli
`InitializationType`	tipo di inizializzazione (`SK_Means` o `Random`)
`InitializationIter`*	numero di iterazioni per l'inizializzazione
`rndSeed`	questi parametri (seme di generazione e range) vengono utilizzati solo se l'inizializzazione è di tipo `Random`
`rndMin`	
`rndMax`	
`KMeanObservations`	questi parametri (proteine da includere e range) vengono utilizzati solo se l'inizializzazione è di tipo `SK_Means`
`KMeanMinIter`*	
`KMeanMaxIter`*	
`Verbose`	abilita la modalità di debug

Tabella 3: Elenco completo dei parametri di Training di `experiment.sh`.

TEST

PARAMETRO	DESCRIZIONE
`MaxSequenceLength`	lunghezza massima delle proteine
`StatesNumber`	numero complessivo di stati degli HMM
`TestObservationsNumber`	numero di proteine presenti nel test set
`ModifyHMM`	abilita le transizioni al modello `@bg`
`TransitionValue`*	indica la probabilità di transizione da un modello all'altro
`TestFile`	specifica il file contente il test set
`RecognizedIndexesFile`	file dove memorizzare gli indici dei percorsi (MPP) seguiti dagli HMM
`OriginalTextFile`	file contenente le sequenze da riconoscere

Tabella 4: Elenco completo dei parametri di Test presenti in `experiment.sh`.

2.5 Primi risultati

Nelle Tabelle 5 e 6 a pagina 19 sono riportati, ordinati per prestazioni crescenti, i risultati ottenuti con l'uso di 2 HMM (senza `@bg`) o di 3 HMM (con `@bg`) e variando di volta in volta i parametri della simulazione.

I modelli `P` e `N` sono stati impostati con lo stesso numero di stati, quindi un numero totale di stati pari a 8 significa aver assegnato 4 stati ad ognuno di essi; nel caso dei 3 HMM, dato che il rumore è stato sempre modellato con un solo stato, un numero totale di 7 stati indica che sono stati assegnati 3 stati sia per `P` che per `N`.

DUE MODELLI (senza `@bg`)

stati	gauss.	inizializ.	epoche	trans.	riconosc.	canc.	ins.	sub.
4	4	2	2	10^{-10}	2.17 %	150	0	1
6	4	2	10	10^{-20}	7.45 %	136	0	8
6	6	1	5	10^{-20}	8.26 %	129	0	6
8	8	1	10	10^{-30}	10.90 %	128	0	7
8	2	2	40	0.1	11.54 %	127	0	11
6	2	2	20	0.1	12.82 %	121	0	9
8	2	2	10	0.1	14.37 %	119	0	7
8	4	2	10	0.1	15.26 %	115	0	4
8	6	2	5	0.1	17.49 %	108	0	2
4	6	1	5	0.1	18.31 %	104	0	32
4	8	1	5	0.1	18.96 %	103	0	30
6	8	2	10	0.1	20.75 %	115	10	27
6	8	1	5	0.1	23.16 %	83	18	25
8	8	1	2	0.1	24.92 %	42	32	20

Tabella 5: Risultati ottenuti con l'uso dei soli modelli `P` e `N`.

TRE MODELLI (con `@bg`)

stati	gauss.	inizializ.	epoche	trans.	riconosc.	canc.	ins.	sub.
7	4	2	5	0.1	3.85 %	147	0	3
9	6	2	5	0.1	8.13 %	136	0	2
13	6	2	3	0.1	10.26 %	134	0	6
11	8	2	3	0.1	11.54 %	132	0	6
9	8	1	3	10^{-30}	13.90 %	110	74	29
7	8	1	5	10^{-20}	14.10 %	54	44	36
11	10	1	5	10^{-20}	16.67 %	50	51	29
11	10	1	10	10^{-20}	17.33 %	45	48	27
9	8	1	2	10^{-20}	20.75 %	37	67	29
9	8	1	5	10^{-20}	23.72 %	49	36	34
9	10	1	5	10^{-20}	25.88 %	46	35	35
9	8	1	10	10^{-20}	26.92 %	37	50	27
11	8	1	5	10^{-30}	28.21 %	58	23	31
11	10	1	5	10^{-40}	31.56 %	49	28	29

Tabella 6: Risultati ottenuti con l'uso dei modelli `P`, `N` e `@bg`.

2.6 Un altro approccio: la segmentazione

Purtroppo, come si può intuire dalle percentuali di riconoscimento (∼25 % con 2 HMM e ∼30 % con 3 HMM), nonostante le numerose topologie testate, le prestazioni non sono state molto elevate (con un classificatore a reti neurali si possono raggiungere anche più del doppio di questi valori); per proseguire con la sperimentazione, si è deciso quindi di provare a sostituire l'approccio "a lista continua" con uno maggiormente focalizzato sulla cisteina.

L'idea è quella di addestrare i modelli `P` e `N` su sottosequenze (matrici di probabilità parziali) della stessa lunghezza, ciascuna *centrata* su *una sola* cisteina. In tal modo, viene anche a perdere di significato l'utilizzo del modello `@bg` del rumore, in quanto si suppone che questo non si trovi in prossimità delle cisteine.

Sequenza completa: W T F A $\underbrace{\text{D D } \mathbf{C} \text{ E H}}_{segmento\,1}$ T M Y $\underbrace{\text{S F } \mathbf{C} \text{ A T}}_{segmento\,2}$ K

Figura 7: Esempio di segmentazione con intervalli di ampiezza 5 (raggio 2).

Ovviamente, dato che la larghezza del segmento viene fissata a priori per tutto il dataset, potrà capitare che l'intervallo da considerare possa non avere al centro la cisteina (cisteina all'inizio o alla fine della sequenze, o troppo vicina ad un'altra cisteina). Per superare questa difficoltà, si può operare in due modi diversi:

- si mantiene ugualmente la matrice decentrata, con il rischio però di produrre una sfasatura nei parametri appresi, che potrebbe causare errori di riconoscimento durante il test.
- si sostituiscono le righe mancanti con vettori in cui tutti gli elementi hanno valore $1/N$, con $N = 20$ in questo caso, cioè si aggiungono nella sequenza posizioni in cui tutti i simboli sono equiprobabili; questa scelta risulta più coerente dal punto di vista matematico, in quanto rispetta l'indeterminazione presente all'esterno dell'intervallo.

Nell'esempio di Figura 7, questo inconveniente si verificherebbe scegliendo un intervallo con raggio maggiore di 4.[12]

In questa soluzione, si passa da un trattamento *globale* della proteina elaborata in un unico processo dai modelli `P`, `N` e `@bg`, ad un trattamento *locale*, che addestra solo i modelli `P` ed `N` su sottosequenze elaborate separatamente per ogni cisteina.

[12]Da qui in avanti, le dimensioni dell'intervallo saranno espresse in funzione del raggio.

2.7 Il generatore di liste segmentate

Nell'intento di proseguire con le sperimentazioni, il generatore di liste è stato rivisto, aggiungendo appunto la possibilità di generare un training ed un test set segmentati in base all'ampiezza desiderata degli intervalli. Questa volta i files `list.train` e `list.test` avranno il formato mostrato in Figura 8, con tante matrici quanto è il numero di cisteine in una proteina (mediamente 4) moltiplicato per il numero di profili da includere; nell'esempio di Figura 7 si è imposto 10 come valore per il raggio dell'intervallo.

```
01 Nome del primo profilo          (informativo,non vincolante)
02 Modello da riconoscere          (P o N)
03 Numero di righe della matrice   (ampiezza del segmento)
04 Numero di colonne della matrice (20)
05  /--------------------------------------\
..  | Matrice del 1° segmento del 1° profilo |
26  \--------------------------------------/
27 Nome del primo profilo          (informativo,non vincolante)
28 Modello da riconoscere          (P o N)
29 Numero di righe della matrice   (ampiezza del segmento)
30 Numero di colonne della matrice (20)
31  /--------------------------------------\
..  | Matrice del 2° segmento del 1° profilo |
52  \--------------------------------------/
.. ............................
70 Nome del secondo profilo        (informativo,non vincolante)
71 Modello da riconoscere          (P o N)
72 Numero di righe della matrice   (ampiezza del segmento)
73 Numero di colonne della matrice (20)
74  /--------------------------------------\
..  | Matrice del 1° segmento del 2° profilo |
95  \--------------------------------------/
96 ............................
.. ............................
```

Figura 8: Formato dei files di training e di test segmentati.

Il nuovo flusso del programma per generare i file `list.train` e `list.test` può essere riassunto nei seguenti passi[13]:

1. Lettura del nome della proteina dal file (`sequence-list.train`) e scrittura nel file di output.
2. Per la proteina in esame:
   (a) Elaborazione del relativo profilo sintetico per ricavare il numero delle cisteine presenti e quindi anche il numero dei segmenti.
   (b) Per ogni segmento:
      i. Scrittura, nel file di output, del nome del modello associato al segmento.
      ii. Calcolo della lunghezza della matrice segmentata in base ai criteri esposti nel § 2.6 e scrittura nel file di output.
      iii. Trascrizione della matrice segmentata nel file di output.
   (c) Si riprende dal punto 1 per elaborare la successiva proteina.

Rispetto alla modalità a "lista continua", adesso il training set ed il test set risulteranno composti da molte più osservazioni (rispettivamente 3780 e 155), ma con lunghezza complessiva minore.

2.8 Nuovi risultati

Nelle Tabella 7 sono raccolti i risultati ottenuti con il nuovo approccio di segmentazione.

Per questi esperimenti, il raggio dell'intervallo di segmentazione è stato fissato a 10 e le eventuali righe vuote sono state riempite con vettori di simboli equiprobabili (v. § 2.6).

[13] Anche stavolta, tutti gli output sono scritture sequenziali sui file di training e di test.

SEGMENTAZIONE (2 modelli)

stati	gauss.	inizializ.	epoche	trans.	riconosc.	canc.	ins.	sub.
2	2	1	1	0.1	3.87 %	0	0	149
4	2	2	5	0.1	29.03 %	0	0	110
4	8	1	1	0.1	36.13 %	0	0	97
6	2	1	10	0.1	44.52 %	0	0	86
4	8	1	20	0.1	51.61 %	0	0	75
4	4	1	3	0.1	59.35 %	0	0	32
4	4	1	5	0.1	60.65 %	0	0	61
6	8	2	10	0.1	65.81 %	0	0	53
4	4	2	5	0.1	66.45 %	0	0	52
4	8	2	20	0.1	67.74 %	0	0	50
6	8	1	10	0.1	68.39 %	0	0	49
6	1	1	10	0.1	69.03 %	0	0	48
4	8	3	20	0.1	70.97 %	0	0	45
6	1	2	10	0.1	71.61 %	0	0	44
4	4	2	10	0.1	72.90 %	0	0	42
6	1	2	15	0.1	73.55 %	0	0	41
8	4	3	10	0.1	74.19 %	0	0	40
4	4	3	5	0.1	75.48 %	0	0	38
4	6	3	5	0.1	77.02 %	0	0	35
6	8	3	10	0.1	77.42 %	0	0	34

Tabella 7: Risultati ottenuti con l'uso dei modelli `P`, `N` e `@bg`.

3 Conclusioni e sviluppi futuri

Dopo aver raccolto ed analizzato i risultati delle sperimentazioni con entrambi gli approcci, è possibile fare un resoconto dell'influenza dei vari parametri nella simulazione.

1. Approccio con "lista continua":
    - In generale, introdurre il modello del rumore ha migliorato le prestazioni del riconoscitore.
    - Usare un maggior numero di gaussiane incrementa spesso la percentuale di riconoscimento; dalle tabelle si intuisce però che un numero troppo elevato potrebbe causare sovrastime dei parametri, mentre un numero troppo basso condurrebbe a sottostime.
    - Mentre si cercava di aumentare la percentuale di riconoscimento, si è provato anche a mantenere il più possibile bilanciato il numero di inserimenti e cancellazioni.
    - Nel caso dell'uso di 3 HMM, un numero di stati per modello pari a 5 o a 6 ha dato in genere risultati migliori.
    - Le iterazioni di inizializzazione non si sono dimostrate eccessivamente influenti; spesso una iterazione è stata sufficiente.
    - Epoche di training maggiori di 5 non hanno determinato un miglioramento sensibile dei risultati.
2. Approccio con segmentazione[14]:
    - Le prestazioni generali sono molto maggiori rispetto all'approccio "a lista continua".
    - Diversamente dall'approccio precedente, buone prestazioni si raggiungono con un basso numero di stati e di gaussiane, in quanto le sequenze sono più brevi e complessivamente devono essere appresi meno parametri, con una notevole riduzione dei tempi di elaborazione.
    - Mentre il numero delle iterazioni di inizializzazioni si è rivelato determinante, epoche maggiori di 5 e più di 6 gaussiane non hanno contribuito a migliorare sensibilmente il riconoscimento.

[14] Ovviamente, la percentuale di riconoscimento per unità stavolta sarà uguale a quella globale e tutti gli errori commessi potranno essere soltanto sostituzioni, come si deduce da quanto detto nel § 2.6.

Con i risultati ottenuti da queste sperimentazioni, si è potuto capire che le configurazioni locali degli aminoacidi adiacenti alla cisteina determinano significativamente la presenza di un legame della cisteina stessa; questa ipotesi sembra essere confermata anche da esperimenti fatti in laboratorio dai biologi.

La qualità dei modelli elaborati potrebbe risultare penalizzata dal fatto che la quantità di dati forniti dai biologi (v. § 2.1) non è proporzionata alla complessità del problema, mentre è noto che, in tutte le applicazioni di *Intelligenza Artificiale*, più alta è la quantità e la significatività dei dati a disposizione, tanto più efficace risulterà l'addestramento; questo fattore è ancora più rilevante in un contesto come quello della *bioinformatica*, in cui le possibili configurazioni delle proteine sono pressoché infinite.

Nonostante la suddetta limitazione, questi esperimenti preliminari potranno essere usati come linee guida per proseguire con le ricerche in questa direzione (modelli di Markov e approcci simili di apprendimento automatico); durante lo svolgimento della tesi è stato possibile individuare alcuni interessanti sviluppi:

- Utilizzare una topologia differenziata per i modelli `P` e `N`.
- Aggiungere un controllo di parità[15] per ridurre gli errori in fase di test.
- Raffinare l'addestramento utilizzando anche gli altri 19 esperimenti, possibilmente con un sistema distribuito di calcolo parallelo (*cluster*) in modo da ridurre i tempi di elaborazione[16].
- Sfruttare anche l'informazione biologica del descrittore, che potrebbe ulteriormente migliorare le prestazioni.
- Sostituire il riconoscitore attuale con uno basato su *modelli ibridi*, risultanti dalla fusione degli HMM con le reti neurali, sfruttando anche gli studi già fatti sull'ASR (vedi [8]).

[15] Le cisteine di una proteina sono sempre in numero pari ed i legami possono avvenire solo fra due di esse.

[16] Gli esperimenti di questa tesi sono stati eseguiti su un computer con processore da 2.0 Ghz ed i tempi medi sono stati di circa 45 minuti per l'addestramento ed 1 minuto per il test.

4 Appendice

Nella Tabella 8 è riportata la nomenclatura standard dei 20 aminoacidi conosciuti in natura, le cui combinazioni costituiscono la base del DNA e del RNA; sono state incluse sia le sigle con tre lettere sia quelle con una.

La Tabella 9 riporta tutte le possibili permutazioni degli aminoacidi A,C,T,G (che fanno parte del nostro codice genetico) in sequenze di lunghezza pari a 3.

Nome	Cod1	Cod2
Acido Aspartico	Asp	D
Acido Glutammico	Glu	E
Alanina	Ala	A
Arginina	Arg	R
Asparagina	Asn	N
Cisteina	**Cys**	**C**
Fenilalanina	Phe	F
Glicina	Gly	G
Glutammina	Gln	Q
Isoleucina	Ile	I
Istidina	His	H
Leucina	Leu	L
Lisina	Lys	K
Metionina	Met	M
Prolina	Pro	P
Serina	Ser	S
Tirosina	Tyr	Y
Treonina	Thr	T
Triptofano	Trp	W
Valina	Val	V

Tabella 8: Nomenclatura dei 20 aminoacidi base

CODICE GENETICO

TTT	Phe		TCT	Ser		TAT	Tyr		TGT	Cys
TTC	Phe		TCC	Ser		TAC	Tyr		TGC	Cys
TTA	Leu		TCA	Ser		TAA	*stop*		TGA	*stop*
TTG	Leu		TCG	Ser		TAG	*stop*		TGG	*stop*
CTT	Leu		CCT	Pro		CAT	His		CGT	Arg
CTC	Leu		CCC	Pro		CAC	His		CGC	Arg
CTA	Leu		CCA	Pro		CAA	Gln		CGA	Arg
CTG	Leu		CCG	Pro		CAG	Gln		CGG	Arg
ATT	Ile		ACT	Thr		AAT	Asn		AGT	Ser
ATC	Ile		ACC	Thr		AAC	Asn		AGC	Ser
ATA	Ile		ACA	Thr		AAA	Lys		AGA	Arg
ATG	Met		ACG	Thr		AAG	Lys		AGG	Arg
GTT	Val		GCT	Ala		AAT	Asp		AGT	Gly
GTC	Val		GCC	Ala		AAC	Asp		AGC	Gly
GTA	Val		GCA	Ala		AAA	Glu		AGA	Gly
GTG	Val		GCG	Ala		AAG	Glu		AGG	Gly

Tabella 9: Quest'ultima tabella mostra come combinazioni degli aminoacidi danno origine a segmenti di DNA e RNA.

Riferimenti bibliografici

[1] Attwood T. K. e Parry-Smith D. J.,
Introduction to Bio Informatics.
Prentice Hall. (1999)

[2] Durbin R., Eddy S., Krogh A., e Mitchison G.,
Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.
Cambridge University Press. (1998)

[3] Haussler D., Krogh A., Mian I. S. e Sjölander K.,
Protein modeling using hidden Markov models: analysis of globins.
Proceedings of the 26th Annual Hawaii International Conference on System Sciences, volume 1, pagine 792-–802. (1993)

[4] Isaacson D. L. e Madsen R. W.,
Markov Chains Theory and Applications.
John Wiley and Sons. (1976)

[5] Krogh A., Brown M., Mian I., Sjölander K. e Haussler D.,
Hidden Markov models in computational biology:
Applications to protein modeling.
Journal of Molecular Biology, 235:1501—31. (1994)

[6] Krogh A.,
Hidden Markov models for labeled sequences.
Proceedings of the 12th IAPR International Conference on Pattern Recognition, pagine 140—144. IEEE Computer Society Press. (1994)

[7] Rabiner, L. R.,
A tutorial on hidden Markov models and selected applications in speech recognition.
Proceedings of the IEEE. (1989)

[8] Trentin E.,
Robust Combination of Neural Networks and Hidden Markov Models for Speech Recognition. (2001)