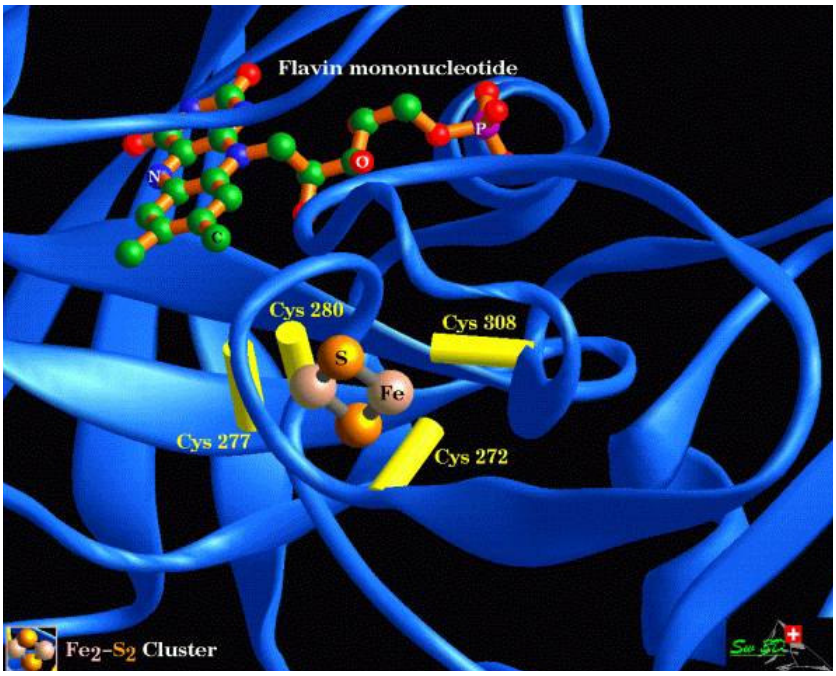
UNIVERSITA' DEGLI STUDI DI FIRENZE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Elaborato per l'esame di
"Intelligenza Artificiale" A.A. 1998-99
Prof. G. Soda

Andrea Fedeli Sauro Menchetti

Allineamento tramite Modelli di Markov Nascosti



Sommario

Introduzione

L'analisi computazionale di sequenze biologiche (descrizioni di molecole di proteine, di DNA e di RNA) ha completamente cambiato le sue caratteristiche alla fine degli anni '80. La principale causa di questi cambiamenti è stata l'avvento di nuove, efficienti tecniche sperimentali, come il sequenziamento del DNA, che hanno portato ad una crescita esponenziale dei dati a disposizione. L'interesse si sta progressivamente spostando dall'accumulo dei dati alla loro interpretazione, dato che anche altri progetti di sequenziamento come quello del genoma stanno andando avanti velocemente. Strumenti computazionali per la classificazione di sequenze, per separare le regioni del DNA che codificano proteine da quelle che non le codificano, per predire la struttura molecolare, per ricostruire la storia dell'evoluzione e per individuare deboli similarità tra le sequenze, sono divenuti una componente essenziale del processo di ricerca. La bioinfirmatica sta emergendo come una disciplina alla frontiera tra la biologia e l'informatica ed ha ripercussioni in medicina, nelle biotecnologie e nella società in vari modi. Le grandi basi di dati di informazioni biologiche hanno tratto giovamento dai comuni algoritmi, ma tale approccio non è più in grado di indirizzare molti dei più interessanti problemi di analisi delle sequenze. Questo è dovuto all'inerente complessità dei sistemi biologici ed alle lacune delle teorie finora sviluppate sull'organizzazione delle vita a livello molecolare. L'approccio dell'apprendimento automatico (reti neurali, modelli di Markov nascosti, reti bayesiane) è adatto in tutti quei domini caratterizzati dalla presenza di grandi quantità di dati anche rumorosi e dall'assenza di teorie generali. L'idea fondamentale di questo approccio è di *imparare la teoria automaticamente dai dati*, attraverso processi di inferenza, di adattamento di modelli e di apprendimento da esempi; questo rappresenta un ap-

proccio percorribile complementare ai metodi convenzionali. I metodi di apprendimento automatico sfruttano pesantemente la potenza dei calcolatori e traggono grandi benefici dal progresso in velocità delle macchine. È stato osservato che sia la velocità dei computer, sia il volume delle sequenze biologiche si sono incrementate della stessa quantità dagli anni '80, raddoppiando circa ogni 15-18 mesi. Dal punto di vista teorico, l'ambiente di lavoro probabilistico bayesiano unifica tutti i metodi di apprendimento automatico e la confluenza di tre fattori - dati, calcolatori e teoria – servirà allo sviluppo dell'apprendimento automatico in bioinformatica e altrove. Una critica spesso sollevata ai metodi di apprendimento automatico è che sono degli approcci a scatola nera: non si può sempre capire come mai un dato modello dia una certa risposta. È importante capire comunque, che molte altre tecniche nella biologia molecolare contemporanea sono usate solamente su base empirica.

1.1 I dati biologici come sequenze di simboli

Una caratteristica fondamentale delle catene molecolari che sono le responsabili delle funzionalità e dell'evoluzione degli organismi viventi, è che possono essere descritte come *sequenze di simboli* scelti da un *alfabeto* di piccole dimensioni. Sperimentalmente le sequenze biologiche possono essere determinate con completa certezza, ed in una posizione di una determinata sequenza ci sarà una sola lettera e non un misto di diverse possibilità. La natura discreta dei dati genetici li rende abbastanza diversi da molti altri tipi di dati scientifici, dove le leggi fondamentali della fisica o la sofisticazione delle tecniche sperimentali necessitano di limiti inferiori di incertezza. Questa caratteristica ha anche un profondo impatto sui tipi di algoritmi che sono stati sviluppati e applicati per un'analisi di tipo computazionale. Mentre spesso lo scopo è quello di studiare una particolare sequenza con le sue funzionalità e la sua struttura molecolare, l'analisi tipicamente procede attraverso lo studio di un insieme di sequenze consistente di differenti versioni tra le varie specie oppure di differenti versioni della stessa specie. Un confronto tra le sequenze di varie specie deve tenere conto della natura "rumorosa" dei dati risultante in parte da eventi casuali amplificati dall'evoluzione: poi-

ché le sequenze di DNA o di amminoacidi aventi le stesse funzionalità saranno diverse, i modelli delle sequenze *devono essere probabilistici*.

1.1.1 Qualità delle basi di dati

Nonostante le sequenze di dati possono essere determinate sperimentalmente con alta precisione, non sono generalmente disponibili ai ricercatori se non con addizionale rumore proveniente da cattiva interpretazione dell'esperimento e incorretta gestione e memorizzazione nei database pubblici. Dato che le sequenze biologiche sono immagazzinate in formato elettronico e che i database pubblici sono curati da gruppi di persone altamente diversi, è forse comprensibile che in molti casi la percentuale di errori cresce come conseguenza della loro gestione. Un altro contributo non trascurabile a questa situazione è il modo in cui i dati sono memorizzati: le caratteristiche delle sequenze sono normalmente indicate per mezzo di una lista delle posizioni rilevanti in formato numerico e non tramite la loro funzionalità. Gli algoritmi di ricerca tentano di costruire approcci associativi per trovare specifiche sequenze che si accordano ad una rappresentazione spesso non troppo certa del loro contenuto: questo è molto diverso dal ricercare le sequenze in base alla loro funzionalità. In questa situazione è importante che la bioinfirmatica tenga conto di queste potenziali sorgenti di errore quando crea approcci di apprendimento automatico per la predizione e la classificazione. Tali tecniche danno un modo alternativo e potente di trovare informazioni e annotazioni erronee. In un insieme di dati, se qualcosa è difficile da imparare, è altamente probabile che esso rappresenti o un caso atipico o un errore. Una delle ragioni del successo delle tecniche di apprendimento automatico su domini di dati imperfetti, è che tali metodi sono capaci di gestire il rumore presente nei dati.

1.1.2 Ridondanza delle basi di dati

Un altro problema che ricorre nell'analisi delle sequenze biologiche, è le ridondanza dei dati. Diversi gruppi di persone possono inserire dati che si riferiscono a sequenze già esistenti, provocando delle duplicazioni delle stesse sequenze che posso essere o meno identiche a quelle presenti. L'uso di dati ridondanti può essere una sorgente per vari tipi di errore. Per lo più tutti gli approcci di apprendimen-

to automatico hanno dei problemi quando certe sequenze sono rappresentate molte volte: ad esempio, se l'insieme dei dati è usato per predire certe caratteristiche, le sequenze usate per addestrare il modello saranno molto correlate a quelle usate per il test del modello e si avrà una sovrastima della capacità predittiva. Benché siano stati proposti alcuni algoritmi per risolvere questo problema, è spesso meglio "pulire" l'insieme dei dati in modo da rendere tutte le sequenze ugualmente rappresentate; può essere cioè necessario evitare le sequenze "molto correlate". D'altra parte, una precisa definizione di "molto correlate" dipende strettamente dal problema in esame. Una strategia alternativa consiste nel pesare le varie sequenze in accordo alla loro novità. Un grande rischio di questo approccio è quello di pesare molto i dati errati. I metodi di apprendimento automatico sono capaci di estrarre le caratteristiche essenziali dai vari esempi e di scartare l'informazione non desiderata quando presente; inoltre sono capaci di trovare più complesse correlazioni e non linearità presenti nelle sequenze esaminando le proprietà statistiche dei segmenti che sono altamente informative.

1.2 L'allineamento delle sequenze

Per riuscire a trovare caratteristiche funzionali o strutturali comuni ad un insieme di dati, le nuove sequenze aggiunte ad un a base di dati sono normalmente allineate a tutte le altre presenti. Il problema fondamentale che sorge è di determinare quando le similarità sono sufficientemente grandi in modo che si possa inferire una somiglianza strutturale o funzionale dall'allineamento delle due sequenze. In altre parole, dato un metodo di allineamento che ha scoperto certe sovrapposizioni tra le sequenze, si può definire una soglia che mi dice se le sequenze sono omologhe? La risposta a questa domanda è non banale e dipende interamente dalla particolare struttura o funzionalità che si vuole esaminare; tale soglia sarà differente per ogni compito. La misura di quanto due sequenze sono allineate non è la stessa attraverso l'intero dominio delle sequenze e le corrispondenze prodotte dagli algoritmi di allineamento dipendono da vari fattori come ad esempio se l'algoritmo è stato progettato per ottimizzare una funzione localmente o globalmente. Alcuni problemi di rilevanza biologica hanno bisogno di effettuare un confronto globale tra due sequenze mentre altri analizzano solo una sottosequenza dei

segmenti: si parla allora di allineamento globale o locale. I classici algoritmi di allineamento sono basati sulla programmazione dinamica che ha però il difetto di essere esponenziale nel numero di sequenze: detto *K* il numero di sequenze da allineare ed *N* la lunghezza media delle sequenze, si ha che la complessità è dell'ordine di $O(N^K)$.

1.2.1 La matrice di sostituzione

Lo schema di allineamento è influenzato dalla scelta della matrice di sostituzione che stabilisce un costo per i vari eventi prodotti dall'evoluzione. Una matrice di sostituzione specifica un insieme di costi s_{ij} per la sostituzione di una lettera dell'alfabeto con un'altra. Alcune matrici sono generate da un modello semplificato dell'evoluzione delle proteine che involve le frequenze p_i degli amminoacidi e le frequenze di sostituzione q_{ij} di coppie di amminoacidi osservate negli allineamenti naturali delle sequenze. Una corrispondenza che interessa un amminoacido raro dovrebbe contare molto di più di una che interessa un comune amminoacido, mentre un errore tra due amminoacidi intercambiabili dovrebbe contribuire di più di uno tra due amminoacidi funzionalmente incorrelati. Un esempio di matrice di sostituzione che si riferisce a sequenze di DNA è la seguente:

	A	C	G	T	Gap (-)
A	0	1	1	1	2
C	1	0	1	1	2
G	1	1	0	1	2
T	1	1	1	0	2
Gap (-)	2	2	2	2	100

Si può dimostrare che ogni elemento della matrice di sostituzione può essere scritto nella forma:

$$s_{ij} = \frac{1}{\lambda}\left(\ln \frac{q_{ij}}{p_i p_j}\right)$$

dove λ è un fattore di scala. Modifiche del valore di λ cambieranno i valori assoluti dei costi ma non i costi relativi dei differenti allineamenti e questo non andrà ad influenzare l'allineamento. Le più semplici matrici di sostituzione sono quelle

in cui tutti gli elementi della diagonale hanno lo stesso valore positivo *s* mentre gli altri hanno lo stesso valore negativo $\underline{s}$. Date quindi due sequenze X_1 ,…, X_N e Y_1 ,…, Y_M che si sono evolute nel tempo, l'allineamento consiste nel minimizzare un certo costo indotto dalla matrice di sostituzione.

L'ambiente probabilistico

L'apprendimento automatico è un diretto discendente di una disciplina più vecchia, l'adattamento di modelli statistici ai dati osservati e come quest'ultima ha lo scopo di estrarre informazioni utili da un insieme di dati costruendo dei buoni modelli probabilistici. La principale novità che introduce è quella di automatizzare il processo finché possibile, spesso utilizzando modelli con un grande numero di parametri; questo approccio si adatta bene in quelle situazioni in cui sono disponibili grandi quantità di dati ma dove non esiste ancora una teoria consolidata. Inoltre, se i dati disponibili sono affetti da rumore e siamo quindi costretti a ragionare in un ambiente incerto, l'approccio bayesiano ci fornisce una teoria robusta che unifica differenti tecniche. Le principali caratteristiche dell'approccio bayesiano possono essere riassunte nei seguenti punti:

1. Utilizza ipotesi o modelli con tutta l'informazione di sottofondo e i dati disponibili.
2. Usa il linguaggio della teoria della probabilità per assegnare probabilità a priori a modelli o ipotesi.
3. Usa il calcolo delle probabilità per valutare le probabilità a posteriori delle ipotesi o dei modelli alla luce dei dati disponibili e per fornire una risposta *univoca* a certi quesiti.

Può essere provato in senso matematico stretto, che questo è il solo modo di ragionare consistente in presenza di incertezza.

2.1 Gli assiomi

In generale un modello può essere visto come un'ipotesi con la differenza che i modelli tendono ad essere ipotesi molto complesse che interessano un grande numero di parametri. In generale un modello verrà indicato con $M = M(w)$, dove w è il vettore di tutti i parametri. Data una certa conoscenza di fondo I, si può associare ad ogni ipotesi X un certo *grado di credenza* che indicheremo con $h(X \mid I)$ che deve soddisfare ai seguenti assiomi:

1. se $h(X \mid I) > h(Y \mid I)$ e $h(Y \mid I) > h(Z \mid I)$, allora $h(X \mid I) > h(Z \mid I)$
2. $h(\underline{X} \mid I) = F[h(X \mid I)]$ dove $\underline{X}$ rappresenta la negazione di X
3. $h(X, Y \mid I) = G[h(X \mid I), h(Y \mid X, I)]$

Si può provare che esiste sempre un fattore di scala k dei gradi di credenza per cui:

$$P(X \mid I) = k\, h(X \mid I) \in [0, 1]$$

dove P è unico e soddisfa tutte le regole delle probabilità. Da tali assiomi si può ricavare il fondamentale teorema di Bayes:

$$P(X \mid Y, I) = P(Y \mid X, I)\, P(X \mid I) / P(Y \mid I)$$

2.2 L'inferenza bayesiana

Il principale tipo di inferenza a cui siamo interessati è quella di derivare un modello $M = M(w)$ da un insieme di dati D. Per semplicità eliminiamo l'informazione di fondo dalle equazioni che seguiranno. Dal teorema di Bayes si ha che:

$$P(M \mid D) = P(D \mid M)\, P(M) / P(D)$$

La probabilità a priori $P(M)$ rappresenta la stima che il modello sia corretto prima di aver ottenuto qualunque dato. La probabilità a posteriori $P(M \mid D)$ rappresenta un aggiornamento della probabilità del modello M dopo che sono stati esaminati i dati D. Il termine $P(D \mid M)$ viene chiamato verosimiglianza. In molti casi le probabilità possono essere molto piccole, così che è più facile lavorare con i logaritmi:

$$\log P(M \mid D) = \log P(D \mid M) + \log P(M) - \log P(D).$$

Per quanto riguarda le probabilità a priori, quando le distribuzioni non sono uniformi, si utilizzano distribuzioni gaussiane, Gamma o di Dirichlet: la caratteristica fondamentale di tali distribuzioni è quella di non assegnare la probabilità nulla ad nessun evento, casa invece che può accedere con una distribuzione uniforme.

2.2.1 Stima dei parametri e selezione di un modello

Data una certa classe di modelli, due modelli M_1 e M_2 possono essere confrontati tramite le loro probabilità $P(M_1 \mid D)$ e $P(M_2 \mid D)$. Spesso si pone il problema di trovare il miglior modello di una classe, cioè di trovare un insieme di parametri w che massimizzi la probabilità a posteriori $P(M \mid D)$ o $\log P(M \mid D)$: questa è chiamata stima dei parametri massimizzando la probabilità a posteriori (MAP). Per lavorare con quantità positive, questo equivale a minimizzare:

$$E = - \log P(M \mid D) = - \log P(D \mid M) - \log P(M) + \log P(D).$$

Il termine $P(D)$ gioca il ruolo di costante di normalizzazione e non dipende dai parametri w, quindi il suo ruolo è ininfluente per l'ottimizzazione. Se la probabilità a priori $P(M)$ è uniforme su tutti i modelli considerati, allora il problema si riduce a trovare il massimo di $P(D \mid M)$ o $\log P(D \mid M)$: questa è chiamata stima dei parametri massimizzando la verosimiglianza (ML). Riassumendo, la stima MAP cerca di minimizzare la quantità

$$E = - \log P(D \mid M) - \log P(M)$$

mentre la stima ML cerca di minimizzare la quantità

$$E = - \log P(D \mid M).$$

Trovare un modello ottimo per una data classe di modelli ha senso solo se la distribuzione $P(M \mid D)$ è concentrata attorno ad un unico picco. In molti casi, in cui c'è un elevato grado di incertezza ed in cui la quantità di dati è relativamente piccola, siamo interessati alla funzione $P(M \mid D)$ sopra l'intero spazio dei modelli piuttosto che solo su un certo massimo e soprattutto siamo interessati nel calcolare le medie rispetto a $P(M \mid D)$. A parità di probabilità, è meglio scegliere il modello più semplice.

2.3 I modelli grafici

Una volta che l'ambiente probabilistico bayesiano è stato definito, l'idea successiva è quella di utilizzare i modelli grafici. Poiché nell'analisi bayesiana il punto di partenza è per lo più sempre una distribuzione di probabilità di grado elevato, tale espressione deve essere decomposta e semplificata. La più comune semplificazione è quella di assumere che alcune variabili siano indipendenti o più precisamente che alcuni insiemi di variabili siano indipendenti, data la dipendenza condizionale con altri insiemi di variabili. Queste relazioni di indipendenza possono essere spesso rappresentate da un grafo dove le variabili sono associate con i nodi e un collegamento mancante rappresenta una particolare relazione di indipendenza. Le relazioni di indipendenza permettono la fattorizzazione della distribuzione di probabilità di grado elevato in un prodotto di semplici locali distribuzioni associate a piccoli gruppi di variabili correlate fra loro. I modelli grafici possono essere suddivisi in due grandi categorie a seconda se gli archi associati sono orientati o meno. I modelli grafici non diretti sono impiegati in quelle situazioni in cui le interazioni sono considerate completamente simmetriche, mentre i modelli grafici diretti sono utili in quelle situazioni in cui le interazioni non sono simmetriche e riflettono relazioni casuali o irreversibilità temporale. Il linguaggio di rappresentazione dei modelli grafici è utile nella maggior parte delle applicazioni di apprendimento automatico; le reti bayesiane, le reti neurali, i modelli di Markov nascosti rappresentano un esempio di tali metodologie.

2.4 Due semplici modelli probabilistici

Il più semplice modello probabilistico per le sequenze biologiche consiste nell'usare un singolo dado. Supponiamo che i dati D siano delle sequenze di DNA su un alfabeto A = {A, C, G, T} di quattro lettere. Il più semplice modello è quello di assumere che le sequenze siano state ottenute lanciando un dado con quattro facce (una per lettera dell'alfabeto) e che i vari lanci siano indipendenti tra loro. Qualunque sia la classe di modelli utilizzata, il primo passo consiste nel calcolare esplicitamente la verosimiglianza e nell'assegnare una probabilità a priori. Supponiamo che i dati siano costituiti da una sola sequenza di lunghezza N: $D = \{S\}$ con $S = S_1, \ldots, S_N$ e $S_i \in A$. I parametri del nostro modello M sono le quattro probabilità p_A, p_C, p_G, p_T , la cui somma deve fare 1. La verosimiglianza è data da:

$$P(D \mid M) = \prod_{X \in A} p_X^{n_X}$$

dove n_X è il numero di volte che la lettera X appare nella sequenza S. Un altro modello semplice consiste nell'usare più dadi. Supponiamo che i dati siano costituiti da K sequenze ognuna di lunghezza N. Per esempio, si potrebbe pensare ad un allineamento multiplo delle K sequenze dove il simbolo "-" potrebbe essere considerato un simbolo dell'alfabeto. In questo modello assumiamo che ci siano N dadi indipendenti, uno per ciascuna posizione e che ogni sequenza è il risultato di aver lanciato gli N dadi in un ordine fissato. Sia p_X^i la probabilità di produrre la lettera X con il dado numero i e sia n_X^i il numero di volte che la lettera X appare nella posizione i; poiché si assume che il dado e le sequenze siano indipendenti, la verosimiglianza è data da:

$$P(D \mid M) = \prod_{i=1}^{N} \prod_{X \in A} p_X^{n_X^i}$$

I successivi modelli che costruiremo non sono altro che raffinamenti di questi semplici modelli che prendono in considerazione aspetti aggiuntivi di cui non si è tenuto conto: i modelli di Markov nascosti sono un'ottima classe di modelli per riuscire ad allineare gruppi di sequenze con una complessità lineare nel numero di sequenze.

Gli algoritmi di apprendimento

Dopo aver costruito un modello parametrizzato $M(w)$ per l'insieme dei dati, il passo successivo consiste nell'eseguire uno dei seguenti punti:

- la stima completa della distribuzione $P(w, D)$ e della probabilità a posteriori $P(w| D)$
- la stima dell'insieme ottimo dei parametri w massimizzando $P(w| D)$
- la stima dei valori medi rispetto alla probabilità a posteriori cioè, per esempio, dell'integrale della forma $E(f) = \int f(w)P(w \mid D)dw$.

Gli algoritmi di apprendimento possono quindi essere suddivisi in tre categorie, a seconda se l'obiettivo è quello di stimare una densità di probabilità, un insieme dei suoi parametri o le corrispondenti medie. Si può affermare anche che un qualunque problema può essere riformulato come un problema di ottimizzazione.

3.1 Programmazione dinamica

Il termine programmazione dinamica si riferisce ad una tecnica generale di ottimizzazione che può essere applicata ad un problema scomponibile in sottoproblemi simili ma di dimensione più piccola, così che la soluzione del problema originario può essere ottenuta mettendo insieme le soluzioni dei problemi più piccoli. Il tipico problema a cui può essere applicata la programmazione dinamica è quello

di trovare il percorso più breve tra due nodi di un grafo. Chiaramente il percorso più breve dal nodo *A* al nodo *B* che passa per il nodo *C* è la concatenazione del percorso più corto da *A* a *C* con il percorso più corto da *C* ad *A*. Questo è anche noto come il principio di Bellman. La soluzione del problema generale è costruita ricorsivamente unendo le soluzioni dei sottoproblemi più semplici. Gli algoritmi di allineamento possono essere visti in termini di trovare il percorso più breve in un appropriato grafo con un determinata metrica. Allineare due sequenze di lunghezza N richiede di trovare il percorso più breve in un grafo con N^2 vertici. Poiché la programmazione dinamica essenzialmente richiede di visitare tutti i vertici una volta, è facile vedere che la complessità in tempo è dell'ordine di $O(N^2)$. La programmazione dinamica e l'algoritmo di Viterbi saranno usati per calcolare la verosimiglianza e l'allineamento delle sequenze usando i modelli di Markov nascosti.

3.2 Discesa lungo il gradiente

Spesso siamo interessati alla stima dei parametri, cioè a trovare il miglior modello $M(w)$ che minimizza la probabilità a posteriori $f(w) = -\log P(w \mid D)$ o la verosimiglianza $-\log P(D \mid w)$. Tutte le volte che $f(w)$ è differenziabile, si può cercare di trovare i suoi minimi usando uno dei più vecchi algoritmi di ottimizzazione, la discesa lungo il gradiente. Come indica il nome, la discesa lungo il gradiente è una procedura iterativa che può essere espressa vettorialmente come:

$$w^{t+1} = w^t - \eta \frac{\partial f}{\partial w^t}$$

dove η è la lunghezza del passo (o learning rate) che può essere fissa o aggiustata durante il processo di apprendimento. Mentre l'idea generale della discesa lungo il gradiente è semplice, usando modelli parametrizzati molto complessi si possono avere varie implementazioni, dipendenti da come realmente viene calcolato il gradiente. Nei modelli grafici spesso si richiede la propagazione dell'informazione all'indietro: questo è il caso dei modelli di Markov nascosti in cui servono una procedura di propagazione in avanti e una all'indietro (la procedura forward ba-

ckward). Ovviamente il metodo di discesa lungo il gradiente dipende dalle condizioni iniziali e se la funzione che deve essere ottimizzata ha molti minimi o massimi locali, non è detto che si riesca ad individuare l'ottimo globale.

3.3 Algoritmo EM

Un'altra classe importante di algoritmi di ottimizzazione è la massimizzazione della media (EM) (che nel casi dei modelli di Markov nascosti è chiamata algoritmo di Baum-Welch). L'utilità di questi algoritmi è rivolta soprattutto ai modelli di Markov nascosti e fa uso del concetto di energia libera. L'algoritmo EM è utile nei modelli con variabili nascoste: tipici esempi di variabili nascoste sono i dati mancanti o non osservabili e i nodi nascosti in un modello grafico. Se D denota i dati, supponiamo che sia disponibile una versione parametrizzata della distribuzione di probabilità congiunta delle variabili osservate e nascoste $P(D, H \mid w)$. Nel caso di maggiore interesse, w indica i parametri di un modello. Supponiamo di voler massimizzare la verosimiglianza $\log P(D \mid w)$ (le stesse idee possono essere estese ad una stime di tipo MAP). Poiché in generale è difficile ottimizzare $\log P(D \mid w)$ direttamente, l'idea base è quella di cercare di ottimizzare la sua media:

$$E(\log P(D \mid w)) = E(\log P(D, H \mid w) - \log P(H \mid D, w)).$$

L'algoritmo EM è un algoritmo iterativo che procede in due passi che si alternano, il calcolo della media ed il passo di massimizzazione. Durante il calcolo della media, viene calcolata la distribuzione delle variabili nascoste, avendo a disposizione i dati osservati e la corrente stima di w. Durante il passo di massimizzazione, i parametri sono aggiornati al miglior valore possibile, essendo nota la presunta distribuzione delle variabili nascoste. I calcoli dei vari passi fanno uso dell'energia libera del modello: si tratta di minimizzare tale funzione che è strettamente collegata alla media della verosimiglianza.

3.4 Vari aspetti degli algoritmi di apprendimento

In un modo o nell'altro, la scelta di un modello deve tenere conto del numero dei parametri per cercare di evitare fenomeni di sovrapprendimento o sotto-apprendimento dei dati. Un approccio per questo problema è quello di pesare la funzione da apprendere con un termine che tiene conto della complessità del modello. Un modo per evitare overfitting quando il modello ha troppi parametri rispetto ai dati, è quello di suddividere l'insieme dei dati in due parti: una parte servirà per addestrare il modello e verrà chiamata training set, mentre l'altra servirà per valutare le prestazioni del modello e verrà chiamata test set. Ognuno dei due insiemi di dati darà origine ad un certo errore: l'errore di addestramento decrescerà monotonicamente al crescere del numero delle epoche, mentre l'errore sull'insieme di test raggiungerà un minimo e poi comincerà a crescere. Il fenomeno dell'overfitting è associato con la memorizzazione dei dati insieme al loro rumore fino al punto che è dannoso per la generalizzazione. L'approccio corretto in una tale situazione sarebbe quello di modificare il modello. Un'altra alternativa è quella di fermare l'addestramento quando l'errore sul test set incomincia a crescere o quando l'errore sul training set ha raggiunto una certa soglia. Comunque quest'ultima tecnica può lasciare un parziale overfitting dei dati. Inoltre, per ottenere un corretto addestramento, tutte le classi di dati dovrebbero essere egualmente rappresentate nel training set.

Gli algoritmi di apprendimento si possono suddividere in due classi, a seconda se l'aggiornamento avviene dopo ogni esempio oppure dopo l'intero insieme di esempi. L'addestramento è detto on-line se l'aggiustamento dei parametri del modello avviene dopo la presentazione di ogni esempio, mentre è detto batch se i parametri sono aggiornati dopo la presentazione di un grande numero di esempi, se non di tutti. L'apprendimento on-line non richiede di tenere in memoria molti esempi ed è più flessibile e più facile da implementare; può però introdurre un certo grado di casualità legato al fatto che l'aggiornamento avviene sulla base di un solo esempio. Può essere dimostrato che l'apprendimento on-line fatto con un tasso di apprendimento sufficientemente piccolo, approssima l'apprendi-mento batch.

Un'ultima osservazione riguarda i modelli che si ottengono dopo l'addestramento. Quando un modello complesso viene adattato ai dati per mezzo di una

ottimizzazione di tipo ML o MAP, si ottengono dei parametri differenti se si variano certi fattori durante la procedura di apprendimento come ad esempio la procedura di addestramento, l'ordine di presentazione degli esempi, il training set. Inoltre possono essere impiegate classi di modelli differenti. È naturale pensare che la migliore classificazione o predizione possa essere raggiunta mediando i parametri dei vari modelli ottenuti in modi diversi.

I modelli di Markov nascosti (HMM)

Il problema che si pone ogni volta che viene individuata una nuova sequenza biologica, è quello di vedere se presenta delle similarità con le altre sequenze già presenti nell'archivio: questo può essere fatto confrontando le sequenze due a due. Il problema diventa ancora più complesso nel caso sia abbia a che fare con sequenze incomplete o con frammenti di sequenze, cosa che è molto comune se si ha che fare con il genoma umano. È certamente di grande interesse riconoscere e classificare tali frammenti, anche perché si pensa che coprano una sostanziale frazione se non tutto il genoma umano. Gli HMM formano una classe utile di modelli grafici probabilistici che può servire a confrontare non coppie di sequenze, ma insiemi di sequenze, per scoprirne le caratteristiche comuni.

4.1 Definizione

Un HMM discreto del 1° ordine è un modello probabilistico per le serie temporali definito da:

- un insieme finito di stati S
- un alfabeto discreto di simboli A
- una matrice delle probabilità di transizione $T = (P(i \mid j))$
- una matrice delle probabilità di emissione $E = (P(X \mid i))$

dove $P(i \mid j)$ indica la probabilità di transizione dal nodo j verso il nodo i e $P(X \mid i)$ indica la probabilità che il nodo i emetta il simbolo X. Il sistema evolve casualmente da uno stato all'altro mentre emette simboli dall'alfabeto. Si parla di modello del 1° ordine perché le transizioni e le emissioni dipendono solo dallo stato corrente e non dal passato. In aggiunta agli stati precedentemente nominati, ci sono due stati speciali, lo stato *Start* e lo stato *End*. Al tempo zero il sistema si troverà sempre nello stato *Start*. Le probabilità di transizione e quelle di emissione sono i parametri del modello. Data una certa architettura di HMM, è di interesse calcolare la probabilità di una sequenza dato il modello (verosimiglianza), qual è il percorso più probabile associato con una data sequenza e infine supposti non noti i parametri, si vogliono valutare i loro valori alla luce dei dati osservati.

4.1.1 HMM per le sequenze biologiche

Per quanto riguarda l'alfabeto, si avrà un alfabeto costituito da 20 amminoacidi nel caso delle proteine, mentre si avrà un alfabeto costituito da 4 nucleotidi nel caso di DNA o RNA. La scelta di una architettura per l'HMM dipende fortemente dal problema. Nel caso di sequenze biologiche, l'aspetto lineare delle sequenze è ben rappresentato dalla cosiddetta architettura left-right.

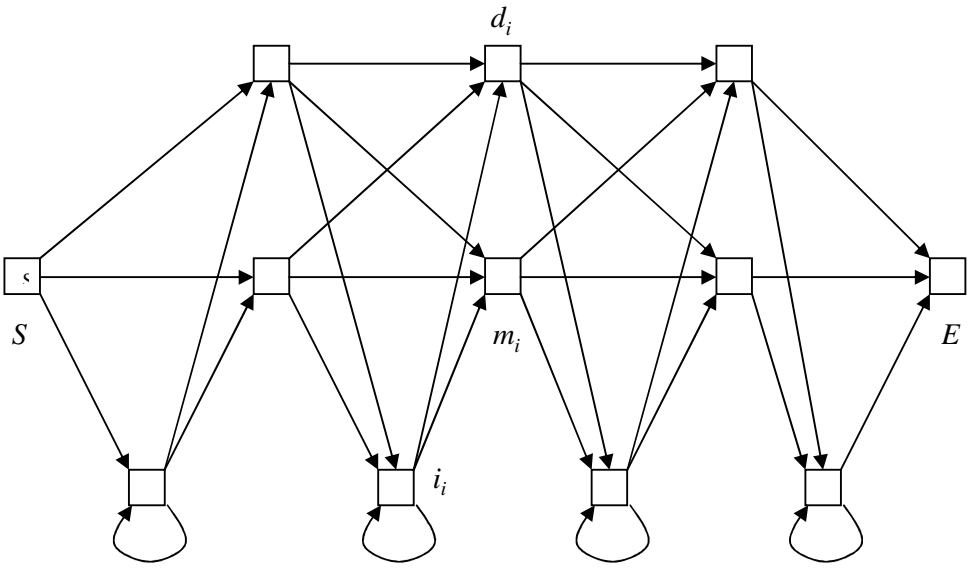


Figura 1: L'architettura lineare standard

Un'architettura left-right non permette il ritorno in uno stato se è avvenuta una transizione da quello stato in un altro stato. L'architettura left-right più usata per le sequenze biologiche è l'architettura standard lineare della figura 1.

Cominciamo cercando di modellare una famiglia di sequenze correlate fra loro. L'architettura standard rappresenta una semplice ma fondamentale variazione del modello con il dado. Il principale problema di tale modello è che la lunghezza delle sequenze non è costante ma sono presenti cancellazioni e inserimenti: l'architettura standard risolve tale problema in modo semplice. Oltre ai due stati *Start* ed *End*, ci sono altri tre tipi di stati chiamati rispettivamente *main*, *insert* e *delete*. L'insieme totale dei stati viene indicato con $S = \{Start, m_1, \ldots, m_N, i_1, \ldots, i_{N+1}, d_1, \ldots, d_N, End\}$. La lunghezza del modello N è di solito uguale alla lunghezza media dell'insieme di sequenze. Gli stati main ed insert emettono sempre un simbolo dell'alfabeto, mentre i delete sono muti. Come si può vedere, il modello contiene uno stato di cancellazione per ogni nodo di tipo main, mentre c'è una corrispondenza biunivoca tra gli stati di inserimento e le transizioni tra i nodi della spina dorsale (fila centrale); inoltre gli stati di inserimento hanno un loop su se stessi. L'architettura del grafo e la tipologia dei nodi risolvono in modo efficiente il problema delle cancellazioni e degli inserimenti; il loop sugli stati di inserimento consente anche degli inserimenti multipli.

Cerchiamo adesso di valutare il numero di parametri di tale modello. Detta $|A|$ la cardinalità dell'alfabeto, si hanno $(N + N + 1)\ |A| = (2N + 1)\ |A|$ parametri di emissione. Per quanto riguarda il numero dei parametri di transizione, valutiamo gli archi uscenti da ogni nodo: tutti i nodi hanno tre archi uscenti meno che lo stato *End* e gli ultimi stati di ogni tipo; quindi $(3N + 2 + 3\ (N - 1) + 2 + 3N + 2) = = 9N + 3$ parametri di transizione. Nel caso di DNA o RNA il numero totale di parametri è $17N + 4$, mentre se si ha che fare con le proteine tale numero sale a $49N + 4$. Inoltre si hanno $2N + 1$ vincoli di normalizzazione sulle probabilità di emissione e $3N + 1$ vincoli di normalizzazione sulle probabilità di transizione.

4.1.2 Informazione a priori e inizializzazione

A causa della natura multinomiale del modello associata alle emissioni ed alle transizioni, la probabilità a priori più naturale per i parametri è la distribuzione di Dirichlet. Per transizioni, si possono usare delle distribuzioni di Dirichlet con lo

stesso parametro oppure con parametri differenti; è da notare che il vettore su cui è centrata la distribuzione non è uniforme, in quanto dovrebbero essere predominanti le transizioni verso gli stati main. Per i parametri di emissione, si possono usare distribuzioni con lo stesso parametro o con parametri distinti, mentre il vettore su cui è centrata la distribuzione può essere uniforme od uguale alla composizione media frequenziale del training set. Per quanto riguarda l'inizializzazione dei parametri, le transizioni devono essere aggiustate in modo da favorire gli spostamenti verso gli stati main, mentre le emissioni possono essere inizializzate in modo uniforme, a caso o con le frequenze delle composizioni medie del training set. Una inizializzazione che devia molto da quella uniforme può produrre degli effetti indesiderati se viene usato l'algoritmo di Viterbi per l'addestramento.

4.2 Verosimiglianza ed algoritmi fondamentali

In questo paragrafo vedremo come calcolare la verosimiglianza e il percorso più probabile associato ad una data sequenza biologica. Tutti gli algoritmi usati sono di tipo ricorsivo e possono essere visti come un'applicazione di programmazione dinamica o come una propagazione dei dati nel grafo diretto associato con l'HMM. Questi algoritmi costituiscono la base per i successivi algoritmi di apprendimento.

Cerchiamo innanzitutto di calcolare la verosimiglianza $P(S \mid w)$ di una sequenza $S = S_1,\ldots, S_t ,\ldots, S_T$ rispetto ad un certo HMM $M = M(w)$ dove w è l'insieme dei parametri. Si definisce un percorso π in M, un insieme di stati consecutivi che partono dallo stato *Start* e che finiscono nello stato *End*, con associata l'emissione di una lettera per ogni stato di emissione (main ed insert). La probabilità di trovare un percorso le cui emissioni coincidano con quelle della sequenza osservata è data da:

$$P(S,\pi \mid w) = \prod_{Start}^{End} P(j \mid i) \prod_{t=1}^{T} P(S_t \mid i)$$

dove il primo prodotto riguarda tutte le transizioni lungo il percorso π, ed il secondo corrisponde a tutte le emissioni delle lettere della sequenza dagli stati *i* facenti parte del percorso π. Ovviamente se una sequenza di emissioni lungo il percorso non coincide con *S*, si ha che $P(S, \pi \mid w) = 0$. La verosimiglianza può essere calcolata come:

$$P(S \mid w) = \sum_{\pi} P(S, \pi \mid w) .$$

Questa espressione non costituisce un metodo di calcolo computazionalmente efficiente per calcolare la verosimiglianza, in quanto il numero di percorsi in un'architettura è tipicamente esponenziale. Si deve quindi trovare un metodo più efficiente per calcolare tale valore, e questo può essere fatto usando un meccanismo di propagazione iterativo lungo il grafo dell'HMM. Di seguito vengono riportate le due procedure fondamentali di propagazione in avanti ed indietro che serviranno come base per le altre procedure.

4.2.1 L'algoritmo Forward

Definiamo innanzitutto la probabilità congiunta $\alpha(i, t)$ di trovarsi nello stato *i* avendo osservato i primi $t - 1$ simboli della sequenza, dati i parametri del modello, nel seguente modo:

$$\alpha(i,t) = P\left(N_t = i, S_1,...,S_{t-1} \mid w\right)$$

dove $N_t = i$ indica l'evento di trovarsi nel nodo *i* al tempo *t*, mentre S_1 ,…, S_{t-1} indicano i simboli della sequenza al tempo 1,…, $t - 1$. Ovviamente tutte le probabilità utilizzate sono condizionate dal modello, quindi per semplicità di notazione, non sempre verrà riportato questo particolare. L'algoritmo iterativo per il calcolo di tali probabilità può essere inizializzato nel seguente modo:

$$\alpha(Start,0) = 1$$
$$\alpha(emis,0) = 0$$

dove *emis* si riferisce a tutti gli altri nodi di emissione. Si può osservare che:

$$P(S \mid w) = \alpha(End, T+1).$$

Il generico elemento $\alpha(i, t)$ può essere calcolato ricorsivamente nel seguente modo, distinguendo i nodi di emissione da quelli di cancellazione:

$$\alpha(i,t) = P\left(N_t = i, S_1, ..., S_{t-1}\right) =$$

$$= \sum_{j \in G(i)} P(N_{t-1} = j, S_1, ..., S_{t-2}) P(N_t = i \mid N_{t-1} = j) P(S_{t-1} \mid N_t = i) =$$

$$= \sum_{j \in G(i)} \alpha(j, t-1) P(i \mid j) P(S_{t-1} \mid i) \quad se \quad i \in E$$

$$= \sum_{j \in G(i)} P(N_t = j, S_1, ..., S_{t-1}) P(N_t = i \mid N_{t-1} = j) =$$

$$= \sum_{j \in G(i)} \alpha(j, t) P(i \mid j) \quad se \quad i \in D$$

dove $P(i \mid j)$ indica la probabilità di transizione dal nodo j verso il nodo i e $P(S_{t-1} \mid i)$ indica la probabilità che il nodo i emetta il $(t - 1)$-esimo simbolo della sequenza; E indica l'insieme di tutti i nodi di emissione, D quello dei nodi di cancellazione, $G(i)$ l'insieme dei genitori del nodo i. Ovviamente $P(N_t = i \mid N_{t-1} = j) = P(i \mid j)$ poiché le transizioni non dipendono dal tempo e così anche per le emissioni. Da qui in poi si sottenderà la dipendenza dal tempo quando non necessaria e dunque si scriverà, per esempio $P(N_t = i \mid N_{t-1} = j)$ come $P(i \mid j)$. Per quanto riguarda gli stati di emissione, il generico $\alpha(i, t)$ sarà dato dalla somma su tutti i genitori del nodo i degli $\alpha(G(i), t - 1)$ moltiplicati per la probabilità di transizione dell'arco che va da $G(i)$ ad i, il tutto moltiplicato per la probabilità di emissione del simbolo S_{t-1} da parte del nodo i. Notiamo che per gli stati di cancellazione è necessario uti-

lizzare una relazione differente: questo perché tali stati non emettono alcun simbolo dell'alfabeto e quindi la probabilità $\alpha(i, t)$ sarà data dalla somma su tutti i genitori del nodo i degli $\alpha(G(i), t)$ moltiplicati per la probabilità di transizione dell'arco che va da $G(i)$ ad i. Questo potrebbe creare qualche problema nella fase di aggiornamento degli stati di cancellazione, poiché in generale al tempo t non sono noti tutti gli $\alpha(i, t)$, in quanto la conoscenza di tali probabilità dipenderà dall'ordine in cui le calcoliamo. Se però ogni stato di cancellazione ha al più un genitore che sia esso stesso stato di cancellazione, sfruttando questa proprietà che induce un ordinamento tra gli stati di cancellazione, possiamo calcolare gli $\alpha(i, t)$ per tali stati nel seguente modo:

1. Si calcolano prima tutti gli $\alpha(i, t)$ per gli stati di emissione.
2. Si considera poi quello stato di cancellazione che non ha tra i suoi genitori un altro stato di cancellazione e si calcola il corrispondente valore di $\alpha(i, t)$. Quindi si procede al calcolo di $\alpha(i, t)$ per quel figlio del precedente nodo che è stato di cancellazione; si prosegue così ricorsivamente in modo da disporre, per ogni nodo di cancellazione, di tutti gli $\alpha(G(i), t)$ necessari.

Al tempo $T + 1$, come anche al tempo 0, non disponiamo di alcun simbolo della sequenza e dunque ci limitiamo ad aggiornare le precedenti formule usando la sola probabilità di transizione. Ecco quella parte di programma che si occupa di aggiornare gli $\alpha(i, t)$:

```
parnodi[0].setAlfa(0, 1);

for (t = 0; t <= l+1; t++)
{
   // aggiornamento dei nodi di emissione
   for (i = 0; i < nnodi; i++)
   if (nodi[i].getTipo() != 2)
   {
      if (t == 0 && i != 0)
         parnodi[i].setAlfa(0, 0);
      else if (t > 0 && t <= l)
         parnodi[i].calcAlfa(t, *archi, seq[t-1], parnodi);
      else if (t == l+1)
         parnodi[i].calcAlfa(t, *archi, 0, parnodi, 1);
   }

   // aggiornamento dei nodi di cancellazione in ordine
```

```
    for (i = 0; i < nnodi; i++)
    if (nodi[i].getTipo() == 2)
      parnodi[i].calcAlfa(t, *archi, seq[t-1], parnodi);

    // scalatura degli alfa
    scalcoef[t] = 0;
    for (i = 0; i < nnodi; i++)
      scalcoef[t] += parnodi[i].getAlfa(t);
    for (i = 0; i < nnodi; i++)
      parnodi[i].setAlfa(t, parnodi[i].getAlfa(t)/scalcoef[t]);
  }
```

Si fa uso di due cicli annidati: quello esterno scandisce tutti i tempi a partire da 0 fino a $T + 1$ incluso, mentre quello interno esamina tutti i nodi. Come prima cosa vengono aggiornati i nodi di emissione (tipo diverso da 2), poi quelli di cancellazione (tipo uguale a 2) secondo l'ordine imposto dall'architettura del grafo. Infine tutti i coefficienti calcolati vengono scalati con un opportuno coefficiente di scalatura, pari alla somma di tutti gli $\alpha(i, t)$ a quel tempo: si terrà conto di questi coefficienti quando ne sarà necessario, in modo che i risultati finali non ne siano influenzati. È tuttavia molto utile scalare gli $\alpha(i, t)$ per stabilizzare computazionalmente l'algoritmo. Valutiamo la complessità di tale algoritmo: detta T la lunghezza della sequenza corrente, si ha che il numero totale di nodi dell'architettura è $N + 2 + N + 1 + N = 3N + 3$. Tralasciando termini moltiplicati e costanti che non influiscono sulla complessità asintotica, si ha che la complessità dell'algoritmo per ogni sequenza è data da $O(NT)$. Supponendo che la lunghezza media delle sequenze sia uguale alla lunghezza del modello (ipotesi generalmente verificata), si ottiene una complessità $O(N^2)$ in tempo e spazio per ogni sequenza. La complessità sarebbe notevolmente maggiore qualora la connessione dei nodi di cancellazione fosse qualunque o comunque differente da quella lineare qui esaminata. Si otterrebbe una complessità superiore anche nel caso in cui la procedura per calcolare gli $\alpha(i, t)$ avesse la necessità di reperire i genitori del nodo i: per evitare questo ulteriore incremento, ogni nodo porta con sé anche l'informazione necessaria a trovare i suoi genitori senza effettuare iterazioni aggiuntive.

4.2.2 L'algoritmo Backward

Definiamo adesso la probabilità $\beta(i, t)$ di aver osservato i simboli della sequenza da S_t a S_T dato lo stato i e dati i parametri del modello, nel seguente modo:

$$\beta(i,t) = P(S_t,...,S_T \mid N_t = i, w).$$

L'algoritmo iterativo per il calcolo di tali probabilità può essere inizializzato nel seguente modo:

$$\beta(End, T+1) = 1$$
$$\beta(emis, T+1) = 0$$

in quanto si parte sempre dal nodo *End* (in quanto è una procedura di propagazione all'indietro): il tempo $T + 1$ sta ad indicare che tutta la sequenza è stata emessa. Tutti gli altri nodi di emissione indicati da *emis* vengono posti a 0 al tempo $T + 1$. Il generico elemento $\beta(i, t)$ può essere calcolato ricorsivamente nel seguente modo:

$$\beta(i,t) = P(S_t,...,S_T \mid N_t = i) =$$

$$= \sum_{j \in F(i) \cap E} P(S_{t+1},...,S_T \mid N_{t+1} = j) P(j \mid i) P(S_t \mid j) +$$

$$+ \sum_{j \in F(i) \cap D} P(S_t,...,S_T \mid N_t = j) P(j \mid i) =$$

$$= \sum_{j \in F(i) \cap E} \beta(j, t+1) P(j \mid i) P(S_t \mid j) + \sum_{j \in F(i) \cap D} \beta(j,t) P(j \mid i)$$

Il coefficiente $\beta(i, t)$ viene calcolato eseguendo una somma su tutti i figli $F(i)$ del nodo i e distinguendo i due seguenti casi:

1. Il figlio è un nodo di cancellazione: in questo caso si calcola la probabilità $\beta(F(i), t)$ moltiplicata per la probabilità di transizione $P(j \mid i)$.
2. Il figlio è un nodo di emissione: in questo caso si calcola la probabilità $\beta(F(i), t + 1)$ moltiplicata per la probabilità di emissione $P(S_t \mid j)$ (se esiste) e per la probabilità di transizione $P(i \mid j)$.

L'aggiornamento dei $\beta(i, t)$ può dare dei problemi se il nodo i ha come figlio uno stato di cancellazione, in quanto in generale al tempo t non abbiamo calcolato ancora alcun $\beta(i, t)$ che riguardi i nodi di cancellazione. Se però ogni stato di cancellazione ha al più un figlio che sia esso stesso stato di cancellazione, sfruttando questa proprietà che induce un ordinamento tra gli stati di cancellazione, possiamo calcolare i $\beta(i, t)$ nel seguente modo:

1. Per ogni nodo i, si calcola la parte di $\beta(i, t)$ definita dal punto 2. precedente, cioè la parte che riguarda i figli di emissione: non si ha alcun problema in quanto si utilizzano i $\beta(i, t)$ già calcolati in precedenza relativi a tempi successivi.
2. Si considera quindi il nodo di cancellazione che non ha tra i suoi figli un altro nodo di cancellazione: questo può essere calcolato correttamente in quanto si dispone di tutti i dati necessari. A partire da questo nodo, si propaga il calcolo dei $\beta(i, t)$ a tutti i suoi figli di cancellazione in modo ricorsivo.
3. Adesso che si dispone di tutti i $\beta(i, t)$ dei nodi di cancellazione, possiamo completare il calcolo anche per gli altri nodi di emissione.

Ordinando i nodi del grafo in modo opportuno, se ad ogni nodo di emissione è collegato al più un nodo di cancellazione, è possibile fondere i passi 2. e 3. in un solo passo. Ecco quella parte di programma che si occupa di aggiornare i $\beta(i, t)$:

```
parnodi[nodoend].setBeta(l+1, 1);

for (t = l+1; t >= 0; t--)
{
  // aggiornamento dei soli nodi di emissione
  for (i = nnodi-1; i >= 0; i--)
  {
    if (t == l+1 && i != nodoend)
```

```
        parnodi[i].setBeta(l+1, 0);
      else if (t == l)
        parnodi[i].calcBeta(t, *archi, 0, parnodi, 1);
      else if (t < l)
        parnodi[i].calcBeta(t, *archi, seq[t], parnodi);
    }

    // aggiornamento dei rimanenti nodi di cancellazione
    for (i = nnodi-1; i >= 0; i--)
      parnodi[i].calcBeta(t, *archi, 0, parnodi, 2);

    // scalatura dei beta
    for (i = 0; i < nnodi; i++)
      parnodi[i].setBeta(t, parnodi[i].getBeta(t)/scalcoef[t]);
  }
```

Questo algoritmo è analogo al precedente con la sola differenza che i coefficienti di scalatura sono gli stessi dell'algoritmo precedente: tali coefficienti non devono essere quindi ricalcolati. La complessità asintotica di tale algoritmo è $O(NT)$, dove T e' la lunghezza della sequenza corrente ed N la lunghezza del modello: supponendo che la lunghezza media delle sequenze sia uguale alla lunghezza del modello, si ottiene una complessità $O(N^2)$ in tempo e spazio per ogni sequenza, la stessa del precedente algoritmo.

4.2.3 Utilizzo dei coefficienti alfa e beta

Noti tutti gli $\alpha(i, t)$ e tutti i $\beta(i, t)$, possiamo calcolare la probabilità $P(N_t = i \mid S)$ di trovarsi in un certo nodo i al tempo t, data una sequenza S: tale probabilità verrà indicata con $\gamma(i, t)$. Per far questo, calcoliamo prima la probabilità congiunta $P(N_t = i, S)$ dove S è la sequenza:

$$P(N_t = i, S) = \alpha(i,t)\beta(i,t) = P(N_t = i, S_1,...,S_{t-1})P(S_t,...,S_T \mid N_t = i).$$

Se poi marginaliziamo su i, otteniamo la probabilità della sequenza $P(S)$. A questo punto è sufficiente dividere la probabilità congiunta $P(N_t = i, S)$ per $P(S)$ per ottenere $P(N_t = i \mid S)$. Quindi il coefficiente $\gamma(i, t)$ e' dato da:

$$\gamma(i,t) = P(N_t = i, | S, w) = \frac{\alpha(i,t)\beta(i,t)}{\sum_{j \in I} \alpha(j,t)\beta(j,t)}$$

dove I indica l'insieme di tutti i nodi del grafo. È utile anche calcolare la probabilità $\gamma(j, i, t)$ della transizione dal nodo i al nodo j al tempo t che è data da:

$$\gamma(j, i, t) = P(N_t = j, N_{t-1} = i \mid S, w) \text{ se } j \in E$$
$$\gamma(j, i, t) = P(N_t = j, N_t = i \mid S, w) \text{ se } j \in D.$$

Si può procede in modo analogo, calcolando prima le due probabilità congiunte $P(N_t = j, N_{t-1} = i, S)$ e $P(N_t = j, N_t = i, S)$ e quindi dividendo per $P(S)$. Il coefficiente $\gamma(j, i, t)$ è dunque dato da:

$$\gamma(j, i, t) = \alpha(i, t)\, P(j \mid i)\, P(S_{t+1} \mid j)\, \beta(i, t) / P(S) \text{ se } j \in E$$
$$\gamma(j, i, t) = \alpha(i, t)\, P(j \mid i)\, \beta(i, t) / P(S) \text{ se } j \in D.$$

Inoltre se marginaliziamo $\gamma(j, i, t)$ rispetto ad i, otteniamo $\gamma(j, t) = P(N_t = j \mid S)$:

$$\gamma(j,t) = P(N_t = j \mid S, w) = \sum_{i \in I} \gamma(j,i,t) .$$

Utilizzando questi coefficienti, possiamo calcolare alcuni parametri utili per gli algoritmi di apprendimento:

1. La probabilità $f(i) = P(i \mid S)$ che la sequenza passi attraverso il nodo i: tale probabilità si ottiene marginalizzando $\gamma(i, t)$ su t:

$$f(i) = P(i \mid S) = \sum_{t=0}^{T} \gamma(i,t) .$$

2. La probabilità $f(i, X) = P(i, X \mid S)$ che un simbolo X sia emesso dal nodo i (se i è un nodo di emissione) data la sequenza S: tale probabilità si ottiene marginalizzando $\gamma(i, t)\ P(S_t = X)$ su t:

$$f(i, X) = P(i, X \mid S) = \sum_{t=0}^{T} \gamma(i,t) P(S_t = X) .$$

Poiché $P(S_t = X)$ vale o 0 o 1 essendo nota la sequenza, questa marginalizzazione equivale a sommare solo quei $\gamma(i, t)$ per cui $S_t = X$ al tempo t. Ovviamente la marginalizzazione di $f(i, X)$ su X dà $f(i)$:

$$f(i) = \sum_{X \in S} f(i, X) .$$

3. La probabilità $f(j, i) = P(N_t = j, N_{t-1} = i \mid S)$ che la sequenza attraversi un arco: tale probabilità si ottiene marginalizzando $\gamma(j, i, t)$ su t:

$$f(j, i) = P(N_t = j, N_{t-1} = i \mid S) = \sum_{t=0}^{T} \gamma(j, i, t) .$$

Inoltre marginalizzando $f(j, i)$ su i si ottiene $f(j)$:

$$f(j) = \sum_{i \in G(j)} f(j, i) .$$

Il calcolo di tali coefficienti avviene nel seguente modo:

```
for (t = 0; t <= lungseq+1; t++)
{
   d = 0;

   // calcolo del denominatore della formula 7.13 pag. 153
   for(i = 0; i < nnodi; i++)
      d += parnodi[i].getAlfa(t)*parnodi[i].getBeta(t);

   // calcolo della formula 7.13 pag. 153
   for(i = 0; i < nnodi; i++)
   {
      n = parnodi[i].getAlfa(t)*parnodi[i].getBeta(t);
      if (n != 0)
         n /= d;
      parnodi[i].setGamma(t, n);
   …
   }
   …
}
```

La complessità di questo algoritmo è $O(NT)$: poiché T è direttamente proporzionale ad N, si ottiene $O(N^2)$. Per quanto riguarda il calcolo dei coefficienti $f(i)$ e $f(j, i)$, che non è altro che una semplice sommatoria sui $\gamma(i, t)$ e sui $\gamma(j, i, t)$, la complessità è dell'ordine di $O(T)$ e quindi non influisce sulla complessità globale dell'algoritmo.

4.2.3 Il percorso più probabile

Il percorso più probabile di una sequenza S lungo il grafo viene individuato in modo semplice: si cerca qual è il nodo più probabile ad ogni istante di tempo t. Tale nodo sarà sempre un nodo di emissione e mai un nodo di cancellazione. Infatti, se la probabilità di trovarsi al tempo t in un nodo di emissione MAX è p_{MAX} e questa è la massima probabilità considerando i soli nodi di emissione, si ha che la probabilità di trovarsi in un nodo di cancellazione al tempo t sarà sempre minore o uguale a p_{MAX}. Infatti nel caso migliore ci dovrà essere almeno una transizione tra il nodo MAX e il nodo di cancellazione $CANC$, il che porta la probabilità di trovarsi nel nodo di cancellazione $CANC$ a $p_{MAX} P(CANC \mid MAX)$ ove $P(CANC \mid MAX)$, che è una probabilità di transizione, è sempre minore o uguale a 1. Questo è vero per ogni istante di tempo t. Un modo per calcolare quale sia il nodo più probabile

al tempo t è quello di scegliere quello che massimizza $\gamma(i, t)$, fissato t. Infatti $\gamma(i, t)$ è proprio $P(N_t = i \mid S)$, cioè la probabilità di trovarsi nel nodo i al tempo t, data S. Un altro metodo è quello di utilizzare l'algoritmo di Viterbi.

4.2.4 L'algoritmo di Viterbi

Iniziamo col definire le seguenti variabili:

$$\delta(i,t) = \max_{\pi(i,t)} P(\pi(i,t) \mid w)$$

dove $\pi(i, t)$ rappresenta la parte iniziale di una sequenza $S_1, \ldots, S_t$ che termina nello stato i. Si ha che $\delta(i, t)$ rappresenta la probabilità associata al percorso più probabile dei primi t simboli della sequenza S che termina nello stato i. Queste variabili possono essere aggiornate usando un meccanismo di propagazione simile all'algoritmo forward, dove le sommatorie sono sostituite con delle massimizzazioni:

$$\delta(i,t) = \left(\max_{j \in G(i)} \delta(j, t-1) P(i \mid j) \right) P(S_t \mid i) \ \ se \ \ i \in E$$

$$\delta(i,t) = \max_{j \in G(i)} \delta(j,t) P(i \mid j) \ \ se \ \ i \in D .$$

Per ricostruire il percorso più probabile, si deve tenere traccia ad ogni tempo t del precedente stato ottimale.

4.2.5 Calcolo dell'allineamento e del percorso più probabile

L'allineamento delle sequenze è basato sul percorso più probabile delle medesime nel modello di Markov nascosto. Leggendo una sequenza, si evidenziano in un qualche modo tutti i simboli che, secondo il percorso più probabile, vengono emessi da uno stato main. Questo perché gli stati main tendono a contenere la parte non rumorosa della sequenza dove è quindi possibile rintracciare delle caratteristi-

che comuni. Si mostrerà più avanti negli esempi come questo accada effettivamente. Tuttavia non è possibile procedere all'utilizzo del percorso più probabile così come è stato definito in quanto non è necessariamente un percorso, cioè non è necessario che esista un cammino tra un nodo al tempo t - 1 ed un nodo al tempo t. In effetti è piuttosto l'elenco dei nodi più probabili nei vari istanti di tempo 1,…,T in quanto è semplice costruire un esempio per cui quello che viene chiamato "percorso più probabile" non sia affatto un cammino sul grafo. Per questo non possiamo scegliere il nodo più probabile al tempo t, come nodo di emissione del simbolo al tempo t, ma dobbiamo curarci del fatto che esista un cammino tra il nodo al tempo t - 1 e tale nodo. Si può procedere quindi nel modo seguente: si scandiscono tutti i nodi tenendo traccia del più probabile per il quale esista un cammino con il nodo al tempo t - 1. Si segnerà in un modo opportuno il simbolo della sequenza se il nodo così calcolato sia un nodo main o un nodo insert. Ecco la parte di programma che si occupa di individuare il percorso più probabile:

```
void TGrafo::PercorsoProbabile(int path[], int seq[], int T,
                                                        int quale)
{
   for (int t = 1; t <= T; ++t)
   {
      int nodoPiuProbabile = nnodi-1;
      double max = parnodi[nodoPiuProbabile].getGamma(t);

      for (int n = nnodi-2; n > 0; n--)
      if ((nodi[n].getTipo() != 2) && (n != nodoend))
      {
         int nodoSuccessivo = 1;

         if (t > 1)
         {
            if (nodi[path[t - 2]].getTipo() == 0)
            {
               if (nodi[n].getId() <= nodi[path[t - 2]].getId())
                  nodoSuccessivo = 0;
            }
            else if (nodi[path[t - 2]].getTipo() == 1)
            if (nodi[n].getId() < nodi[path[t-2]].getId()
               nodoSuccessivo = 0;
         }
```

```
            if (nodoSuccessivo && max < parnodi[n].getGamma(t))
            {
               max = parnodi[n].getGamma(t);
               nodoPiuProbabile = n;
            }
         }

         path[t-1] = nodoPiuProbabile;
      }
      …
   }
```

Come si vede il flag `nodoSuccessivo` controlla se il nodo abbia o meno un cammino con il nodo al tempo t - 1. Con le assunzioni fatte nel nostro modello, questa ricerca è lineare nel numero dei nodi in quanto l'esistenza del cammino per la particolare struttura può essere verificata attraverso due semplici controlli sul numero che identifica il nodo. In genere il controllo dell'esistenza di un cammino richiederà un tempo lineare nel numero dei nodi e dunque l'algoritmo generale avrà una complessità quadratica. Dunque la complessità totale dell'algoritmo in tempo è $O(NT)$ analoga a quella in spazio.

Dobbiamo notare inoltre che il percorso più probabile *dipende* dal quali coefficienti vengono massimizzati. Infatti se il punto di massimo dei coefficienti definiti dall'algoritmo di Viterbi corrisponde al punto di massimo dei coefficienti definiti dall'algoritmo generale, questo in generale non vale per l'intero vettore dei coefficienti: se da entrambi eliminiamo il punto di massimo comune, otteniamo due vettori con punti di massimo differenti. Dunque il percorso più probabile sarà differente a seconda dell'algoritmo con il quale viene calcolato: in particolare con Viterbi sarà meno accurato in quanto quei coefficienti contengono molta meno informazione, proprio per come sono stati definiti. D'ora in poi chiameremo il percorso più probabile o l'allineamento calcolato con l'algoritmo di Viterbi, "Viterbi" e quello calcolato con l'algoritmo generale "gradiente". Vedremo negli esempi come l'allineamento così come l'addestramento effettuato con Viterbi sia peggiore di quello con il gradiente.

A questo punto, disponendo di un vero e proprio percorso più probabile, possiamo calcolare l'allineamento che consiste soltanto nell'evidenziare i simboli emessi da nodi di tipo main. Sono necessarie anche altre procedure per aggiungere

spazi e rendere gradevole la visione dell'allineamento, ma che non influiscono in modo rilevante sulla complessità generale.

4.3 Gli algoritmi di apprendimento

Gli algoritmi di addestramento implementati per l'addestramento dell'HMM sono:

- l'algoritmo EM
- l'algoritmo di discesa lungo il gradiente
- l'algoritmo di Viterbi (variazione dell'algoritmo di discesa lungo il gradiente).

Viene trattata la stima ML, supponendo che le probabilità a priori siano uniformi. Nel caso in cui le sequenze di allineamento siano più di una, queste possono essere considerate indipendenti e la verosimiglianza totale è data dal prodotto delle verosimiglianze di tutte le sequenze.

4.3.1 L'algoritmo EM

Questo algoritmo si propone di adattare il modello all'insieme di sequenze considerato, massimizzando una certa funzione di energia. È un algoritmo di tipo batch in quanto esegue l'aggiornamento ad ogni epoca, dopo aver osservato tutte le sequenze. Il passo di aggiornamento dell'algoritmo comporta la sostituzione di certe probabilità con la loro stima ottenuta massimizzando l'energia libera del modello:

$$P(X \mid i) = f(X, i) / f(i)$$
$$P(j \mid i) = f(j, i) / f(i).$$

dove X è un simbolo della sequenza. Tali formule riescono ad adattare molto bene il modello alle sequenze ma purtroppo portano ad una cattiva generalizzazione in quanto provocano l'assorbimento del rumore presente nell'insieme di addestra-

mento. Quindi è bene utilizzare questo algoritmo solo in modalità batch e se non è necessaria un'elevata capacità di generalizzazione. In questi casi è meglio utilizzare l'algoritmo di discesa lungo il gradiente. L'aggiornamento delle variabili avviene nel seguente modo:

```
for (int n = 0; n < nnodi; ++n)
{
   for (int c = 0; c < cardalf; ++c)
      nodi[n].setProbEmis(c, eta *sommanix[n][c] / sommani[n]);

   for (c = 0; c < nodi[n].getNumFig(); c++)
   {
      int i = nodi[n].getArcFig(c);
      archi->aggiornaP(i, eta * sommanji[i] / sommani[n]);
   }
}
```

Vediamo che le probabilità di emissione e di transizione vengo semplicemente sostituite con le appropriate stime. La variabile `eta` che rappresenta il learning rate, viene sempre posta sempre ad 1 per questo algoritmo. La complessità dell'algoritmo EM è la stessa di quello di discesa lungo il gradiente, con le stesse costanti additive e moltiplicative.

4.3.2 L'algoritmo di discesa lungo il gradiente

Una possibile realizzazione dell'algoritmo di discesa lungo il gradiente può essere quella di massimizzare la verosimiglianza, o equivalentemente di minimizzare l'inverso del logaritmo della stessa. Si ha che la verosimiglianza di un percorso è data da:

$$P(S,\pi \mid w) = \prod_{Start}^{End} P(j \mid i) \prod_{t=1}^{T} P(S_t \mid i)$$

ove con π si è indicato un percorso qualsiasi. Ovviamente $P(S \mid w)$ si otterrà marginalizzando la $P(S, \pi \mid w)$:

$$P(S \mid w) = \sum_{\pi} P(S, \pi \mid w) .$$

Si ha che:

$$\frac{\partial P(S,\pi)}{\partial P(K|i)} = \frac{n(i,X,\pi,S)}{P(K|i)} P(S,\pi) \Rightarrow$$

$$\Rightarrow \frac{\partial \log P(S)}{\partial w_{iX}} = \sum_{K \in A} \frac{\partial \log \sum_{\pi} P(S,\pi)}{\partial P(K|i)} \frac{\partial P(K|i)}{\partial w_{iX}} =$$

$$= \sum_{K \in A} \frac{\sum_{\pi} n(i,X,\pi,S) P(\pi|S)}{P(K|i)} \frac{\partial P(K|i)}{\partial w_{iX}} =$$

$$= \sum_{K \in A} \frac{1}{\sum_{\pi} P(S,\pi)} \frac{\sum_{\pi} n(i,K,\pi,S) P(S,\pi)}{P(K|i)} \frac{\partial P(K|i)}{\partial w_{iX}} =$$

$$= \sum_{\pi} n(i,X,\pi,S) P(\pi \mid S) + \sum_{K \in A} \sum_{\pi} n(i,K,\pi,S) P(\pi \mid S) P(K \mid i)$$

dove A indica l'insieme dei simboli dell'alfabeto. La funzione $n(i, K, \pi, S)$ compare nel calcolo della derivata di $P(S,\pi)$.Infatti:

$$\frac{\partial P(S,\pi)}{\partial P(K|j)} P(K \mid j) = \frac{\partial \prod_{Start}^{End} P(j|i) \prod_{t=1}^{T} P(S_t|i)}{\partial P(K|j)} P(K \mid j) =$$

$$= n(i,K,\pi,S) \prod_{Start}^{End} P(j \mid i) \prod_{t=1}^{T} P(S_t \mid i)$$

in quanto non possiamo sapere in generale quanto vale $n(i, K, \pi, S)$, cioè quante volte si ripete $P(K \mid j)$ nella produttoria, poiché tale numero dipende per esempio dalla sequenza S. Possiamo interpretare $n(i, K, \pi, S)$ quindi come il numero di volte che il simbolo K viene emesso dal nodo i in un certo percorso π legato ad una sequenza S. Sommando su K possiamo calcolare $n(i, \pi, S)$. Moltiplicando gli $n(i, K, \pi, S)$ e gli $n(i, \pi, S)$ per la probabilità $P(\pi \mid S)$ di ciascun percorso π data la sequenza S e quindi marginalizzando sui percorsi π, otteniamo:

1. per gli $n(i, K, \pi, S)$ la probabilità $P(i, K \mid S)$ che è già stata calcolata nel coefficiente $f(i, K)$:

$$\sum_{\pi} n(i, K, \pi, S) P(\pi \mid S) = f(i, K)$$

2. per gli $n(i, \pi, S)$ la probabilità $P(i \mid S)$ che è già stata calcolata nel coefficiente $f(i)$:

$$\sum_{\pi} n(i, \pi, S) P(\pi \mid S) = f(i) .$$

A questo punto è utile introdurre una riparametrizzazione del modello, usando i seguenti esponenti normalizzati:

$$P(X \mid i) = \frac{e^{w_{iX}}}{\sum_{Y} e^{w_{iY}}} \qquad P(j \mid i) = \frac{e^{w_{ji}}}{\sum_{k} e^{w_{ki}}}$$

dove w_{iX} e w_{ji} sono le nuove variabili. Questa riparametrizzazione ha tre vantaggi:

- si mantengono automaticamente i vincoli di normalizzazione sulle emissioni e sulle transizioni;
- le probabilità di emissione e di transizione non possono mai raggiungere il valore 0;
- è invariante rispetto alle traslazioni.

Le formule di aggiornamento dell'algoritmo di discesa lungo il gradiente sono quindi le seguenti:

$$\Delta w_{iX} = \eta\big(f(i, X) - f(i) P(X \mid i)\big)$$

dove η indica il tasso di apprendimento o learning rate. Per le probabilità di transizioni vale la seguente formula che si ricava in modo del tutto simile:

$$\Delta w_{ji} = \eta\big(f(j,i) - f(i)P(j \mid i)\big).$$

Questo algoritmo può essere utilizzato sia in modalità on-line, sia in modalità batch. Nel nostro programma viene utilizzato sempre nella forma on-line. L'implementazione di tale algoritmo è la seguente:

```
for (i = 0; i < nnodi; i++)
{
   n1 = parnodi[i].getN();
   for (j = 0; j < cardalf; j++)
   {
      n2 = parnodi[i].getNX(j);
      prob = nodi[i].getProbEmis(j);
      delta = eta*(n2 - n1*prob);
      delta += nodi[i].getEmis(j);
      nodi[i].setEmis(j, delta);
   }
   nodi[i].calcProb();

   for (j = 0; j < nodi[i].getNumFig(); j++)
   {
      nar = nodi[i].getArcFig(j);
      n2 = pararchi[nar].getN();
      ar = archi->getArco(nar);
      prob = ar->ptrans;
      delta = eta*(n2 - n1*prob);
      archi->aggiorna(nar, delta);
   }
}
archi->calcPTrans();
```

La complessità di questo algoritmo è dell'ordine di $O(N)$ in quanto il numero di figli di ogni nodo, dato da `getNumFig()`, è costante e quindi viene assorbita dalle complessità per il calcolo degli $\alpha(i, t)$, $\beta(i, t)$ e $\gamma(i, t)$. Concludendo, l'addestramento tramite l'algoritmo di discesa lungo il gradiente di M sequenze richiede un tempo dell'ordine di $O(MN^2)$, lineare quindi nel numero delle sequenze come si voleva. Lo spazio richiesto è dell'ordine di $O(N^2)$, indipendentemente dal numero delle sequenze. Pagando una penalizzazione nel tempo non si ottiene co-

munque un miglioramento della complessità spaziale, se non sulle costanti additive e moltiplicative.

4.3.3 L'algoritmo di addestramento di Viterbi

Il precedente algoritmo di discesa lungo il gradiente utilizza tutti i percorsi relativi ad una sequenza per l'aggiornamento. L'idea generale dell'algoritmo di Viterbi è quella di sostituire i calcoli relativi a tutti i percorsi con dei calcoli che riguardano solo un piccolo numero di percorsi, tipicamente uno solo. In questa versione dell'algoritmo, si ignorano tutti i percorsi tranne il più probabile: questo porta ad una modifica abbastanza semplice dei coefficienti $f(i)$ ed $f(i, K)$ che non sono più delle probabilità, pur mantenendo il loro significato iniziale:

- $f(i)$ che indicava la probabilità di essere nel nodo i data la sequenza S, viene posto ad 1 nella versione proposta. Un'altra possibilità potrebbe essere quella di porlo ad 1 se il nodo fa parte in un qualche senso del percorso più probabile. È comunque vero che quasi tutti i nodi fanno parte del percorso più probabile e la distinzione viene fatta negli $f(i, K)$.
- $f(i, K)$ che indicava la probabilità di emettere il simbolo K dal nodo i, data la sequenza S, viene modificato nel seguente modo: vale 1 se il simbolo K viene emesso almeno una volta dal nodo i nel percorso più probabile; vale 0 altrimenti. Nella versione proposta non vale più che la marginalizzazione di $f(i, K)$ su K sia $f(i)$ in quanto può accadere che per ogni K tutti gli $f(i, K)$ valgano 0.
- $f(i, j)$ che indicava la probabilità di andare dal nodo i al j, data la sequenza S, viene sostituita in modo analogo alla precedente: vale 1 se il nodo fa parte del percorso più probabile, 0 altrimenti. Anche queste non sono più probabilità perché può accadere che dal nodo j gli archi uscenti che fanno parte del percorso più probabile siano 2 (per esempio se j è uno stato di inserimento, l'arco che lo riporta in se stesso e un altro arco uscente verso un main od un insert).

Apportando queste modifiche a tali coefficienti, l'algoritmo di aggiornamento è lo stesso del gradiente.

Risultati e Test

Per poter misurare quanto è allineato un insieme di sequenze, definiamo innanzitutto delle metriche che quantifichino il concetto di allineamento. Confronteremo poi i vari algoritmi di apprendimento su un insieme di sequenze di DNA e di proteine, con particolare riguardo al problema della generalizzazione.

5.1 Misure degli allineamenti

Prima di entrare nei dettagli delle prove è necessario definire alcuni strumenti con cui effettuare delle misure. In effetti cosa significa "allineamento"? Di questo è già stata data una definizione che tuttavia non può essere applicata in quanto il numero delle sequenze compare all'esponente nel calcolo della complessità, mentre nel nostro algoritmo compare come fattore moltiplicativo. Ovviamente ciò che si perde è proprio una misura rigorosa dell'allineamento: questo può in effetti essere calcolata anche a posteriori, verificando così che non sarà stata del tutto minimizzata, ma poiché anche la misura dipende molto dalla natura di ciò che vogliamo allineare, è interessante chiedersi se non si possa utilizzare una misura o un insieme di misure che prescindano dalla natura delle sequenze trattate: d'altra parte un essere umano ha un chiaro concetto qualitativo di allineamento e quindi ha senso chiedersi come tradurre in numero questo concetto.

Una prima possibilità è quella di contare in ogni colonna come si ripartiscono in percentuale i vari simboli dell'alfabeto. In questo modo possiamo dividere in due classi le colonne prodotte dell'allineamento: quelle allineate e quelle non

allineate dove si considerano allineate tutte quelle colonne in cui esiste un simbolo *X* che appare con una percentuale molto elevata, per esempio il 95%. Questo perché alla comune esperienza appare ragionevole chiamare 'allineata' una colonna in cui compare sempre lo stesso simbolo. In genere si dirà che una colonna è allineata al tot% se esiste un simbolo che compare almeno con quella frequenza.

Questa divisione è tuttavia abbastanza drastica e si potrebbe anche pensare che esistano dei casi in cui questo numero è sempre 0 perché le colonne sono allineate solo al 94% dunque appare naturale definire una seconda misura che tenga conto di questa sfumatura. Ogni colonna contribuisce alla misura con:

- 0,25 se la colonna è allineata allo 80%;
- 0,5 se la colonna è allineata al 90%;
- 1 se la colonna è allineata al 95%.

Sotto lo 80% possiamo ragionevolmente affermare che la colonna non è affatto allineata in quanto su 5 simboli ce ne è uno differente.

Un'altra possibilità, del tutto differente, per misurare l'allineamento è quello di fare riferimento ad una ipotetica (o reale) sequenza più probabile (o più comune). Definiamo la sequenza più probabile in un insieme di sequenze allineate in tante colonne nel seguente modo: il simbolo *i*-esimo della sequenza più probabile è il simbolo più probabile della colonna *i*-esima. Non va confuso questo con il percorso più probabile. A questo punto, avendo un parametro di confronto, definiamo la seguente misura: ogni simbolo della colonna *i*-esima contribuisce con +1 alla misura se è differente dal simbolo omologo della sequenza più probabile, con 0 altrimenti. Quindi se tutte le sequenze sono uguali, questa misura vale 0, se le sequenze sono casuali la misura sarà massima. In particolare la media di questo valore, se i simboli sono *M*, tutti equiprobabili, su *K* sequenze di lunghezza *N* sarà con probabilità 1, quando *K* tende a infinito, $(1-1/M)N$, ovvero dividendo per il numero di colonne in modo da avere un valore normalizzato $(1-1/M)$.

Questa misura tuttavia non fa differenza tra colonne allineate e colonne non allineate: infatti non dovrebbe avere molta influenza il fatto che sia diverso un simbolo in una colonna allineata solo al 20%. Quindi si potrebbe procedere ad una misura simile alla precedente che tuttavia pesi l'allineamento delle colonne. In particolare ogni simbolo contribuisce alla misura:

- +8 se la colonna è allineata al 97%;
- +4 se la colonna è allineata al 95%;
- +2 se la colonna è allineata al 90%;
- +1 se la colonna è allineata allo 80%;
- +0,5 se la colonna è allineata al 60%.

Quindi un simbolo diverso dal più probabile in una colonna allineata quasi perfettamente viene pesato 16 volte di più che non in una colonna allineata solo al 60% (quindi con poco allineamento). Questa misura vale 0 sia per le colonne perfettamente allineate sia per quelle che non lo sono affatto: questo può fornire una utile indicazione. Infatti questa misura man mano che l'algoritmo produce allineamenti migliori tenderà prima a salire, in quanto cominciano ad esserci alcune colonne allineate, quindi a diminuire. Si dovrà fare attenzione al fatto che questa diminuzione non indichi un overfitting.

Per ognuna di queste misure è inoltre interessante considerare, oltre al valore medio, anche il valore massimo ed il numero di sequenze comprese tra la (media + massimo) / 2 ed il massimo che rappresentano il numero di sequenze che non sono state allineate, qualora la differenza tra massimo e media sia rilevante; altrimenti è un numero abbastanza casuale. È importante osservare come nessuna di queste misure possa funzionare da sola:

- il numero di colonne allineate, anche pesato, non ci dice nulla sull'allineamento delle singole sequenze;
- il numero di simboli differenti dalla sequenza più probabile non ci dice nulla su quanto sia lo scostamento dalle colonne allineate;
- la misura dello scostamento dalle colonne allineate, in quanto vale 0 sia per sequenze perfettamente allineate sia per sequenze poco allineate non ci dice se quel numero è piccolo perché ci sono poche colonne allineate o perché l'allineamento è buono.

Un buon metodo è quello di considerare l'andamento di tali misure: è atteso crescente per la prima, decrescente per la seconda e prima crescente, poi decrescente per la terza. In particolare ad una diminuzione della seconda misura, è un fatto po-

sitivo l'aumento della terza in quanto segno che le colonne si stanno allineando e dovrebbe corrispondere anche ad un aumento della prima.

5.2 Test

Sono stati effettuati dei test per misurare la bontà degli allineamenti, secondo le misure appena date. Abbiamo utilizzato due insiemi di addestramento differenti: uno inventato ad hoc che utilizza i simboli del DNA come alfabeto ed un altro che utilizza un insieme di immunoglobuline. Per ognuno di questi insiemi sono stati effettuati vari addestramenti utilizzando solo una parte, da 1/3 a 2/3, dell'insieme di sequenze in modo da poter verificare la capacità di generalizzare in quanto l'allineamento poi è stato sempre calcolato su tutte le sequenze. Abbiamo in questo modo operato una sorta di validazione incrociata.

5.2.1 Esempi di allineamento

Questi test vogliono, tra l'altro, chiarire il significato delle misure proposte e dell'allineamento. L'insieme delle sequenze considerato è il seguente:

```
AAACTGTTGGGCCCC
AAACTTTGTTGGCCCC
AAACTTTGGGCCACC
AACTTTGGGCCCC
AAACTTTGTTGGGCCCC
AAACTTTGGAAGCCACC
AATGACTTTGGGCCCC
AAACTTTGGGCCCC
AAACTTTGACGGCCCC
AACACTTTGGGCCCC
AACTTTGGCCC
AAACTTTGGGCGGCCC
AGAACTTTGGGCCCC
AAACTTTGGGCCATGCC
AAACTATTGGGCCCC
AAACTTTGGGCCCC
AAAGCTTTGGGCCCC
AAACTTTGGGCCCC
AAACTTTGGGCCCC
AAATCTTGGGCCCC
AAACTTTGGGCCCC
AAACTTTGGGCCCCGGGTTTTAAACTTTGGGCCCCGGGTTTT
```

Come si vede sono tutte “variazioni sul tema” che è AAACTTTGGGCCCC che tolgono, spostano o aggiungono uno o più simboli. Da questo insieme si riesce ad ottenere il seguente allineamento effettuato con Viterbi utilizzando per l’addestramento lo stesso insieme di sequenze:

```
A A  A CTgTTG  G  G C    CC   C
A A  A CT TTGttG  G C    CC   C
A A  A CT TTG  G  G C    CA   Cc
A Ac    T TTG  G  G C    CC   C
A A  A CT TTGttG  GGC    CC   C
A A  A CT TTG  GaaG C    CA   Cc
A AtgA CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  A CT TTGacG  G C    CC   C
A Ac A CT TTG  G  G C    CC   C
A Ac    T TTG  G    C    CC
A A  A CT TTG  G  G Cggcc     C
AgA  A CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CAtgcC
A A  A CTaTTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  AgCT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  AtCT T G  G  G C    CC   C
A A  A CT TTG  G  G C    CC   C
A A  A CT TTG  G  G C    CC   Cgggttttaaactttgggccccgggtttt
```

Questo è l’allineamento vero e proprio: notiamo che in maiuscolo sono stati scritti i simboli che nel percorso più probabile vengono emessi da nodi di tipo main, mentre in minuscolo quelli emessi da nodi di tipo insert che rappresentano il rumore. Come si può vedere, l’algoritmo è riuscito ad eliminare quasi completamente il rumore introdotto. Oltre all’allineamento vengono anche stampate le informazioni sulle misure sopra introdotte:

```
Sequenza più probabile : AAAC-TTTGGG-CCCC
```

Questa è la sequenza più probabile secondo la definizione data: notiamo che è proprio il “tema” comune. Le lineette ‘-‘ indicano che quello stato ‘main’ generalmente non emette alcun simbolo.

```
Misura 1 0.590909 / lungmod = 0.03693
```

```
Misura 1 max 4.00000 #seq comprese tra 2.29545 e 4.00000 = 3
```

Questa è la misura del numero di simboli differenti medio, normalizzato e massimo. Notiamo che il valore normalizzato è bassissimo. Il numero di colonne considerate non allineate 3 dovrebbe essere casuale, ma non lo è troppo: per esempio le sequenze 4 e 6 appaiono non allineate.

```
Misura 3 1.454545 / lungmod = 0.09091
Misura 3 max 12.00000 #seq comprese tra 6.72727 e 12.00000 = 3
```

Questa è la misura dello scostamento dalle colonne allineate. Le 3 colonne considerate non allineate sono le stesse individuate dall'altra misura.

```
Misura 4a 13 / lungmod = 0.81250
Misura 4b 14 / lungmod = 0.89062
```

Questo sono le misure sul numero di colonne allineate al 95% con di seguito la normalizzazione. La prima misura è quella non pesata, la seconda quella pesata.

```
Sequenza prob. >95% : AA**-TTTGGG-CC*C
```

Infine questo riporta la stessa sequenza più probabile eliminando con degli * quelle colonne che non sono allineate al 95%. L'allineamento con il gradiente è il seguente:

```
A A  A CTgTTG   G  G C    CC                              C
A A  A CT TTGtt G  G C    CC                              C
A A  A CT TTG   G  GCC    AC                              C
A A    CT TTG   G  G C    CC                              C
A A  A CT TTGttgG  G C    CC                              C
A A  A CT TTG   GaaGCC    AC                              C
A AtgA CT TTG   G  G C    CC                              C
A A  A CT TTG   G  G C    CC                              C
A A  A CT TTGac G  G C    CC                              C
A Ac A CT TTG   G  G C    CC                              C
A A    CT TTG      G C     C                              C
A A  A CT TTG   G  G Cgg  CC                              C
AgA  A CT TTG   G  G C    CC                              C
A A  A CT TTG   G  G Ccatg C                              C
A A  A CTaTTG   G  G C    CC                              C
A A  A CT TTG   G  G C    CC                              C
```

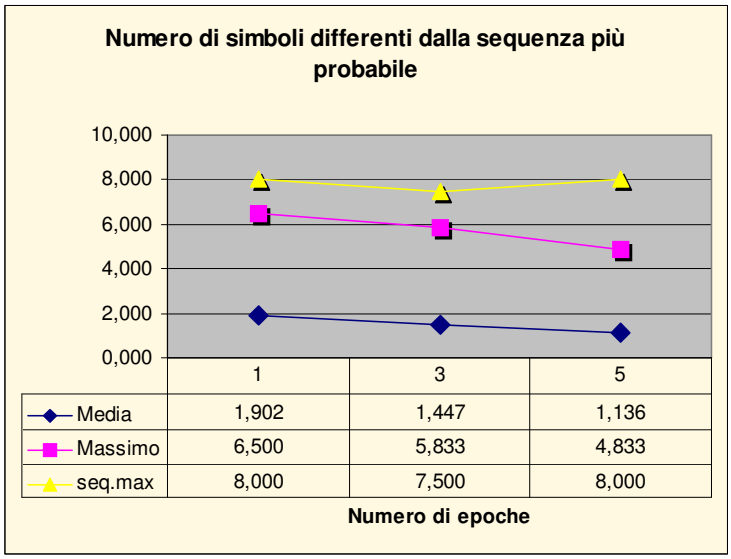
```
A A  AgCT TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CC                                C
A A  AtC  TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CC                                C
A A  A CT TTG   G  G C    CCcgggttttaaactttgggccccgggtttt

Sequenza più probabile : AAAC-TTTGGG-CCCC

Misura 1 0.500000 / lungmod = 0.03125
                  / lungmod / cardalf  0.00781
Misura 1 max 3.00000 #seq comprese tra 1.75000 e 3.00000 = 9
Misura 3 1.090909 / lungmod = 0.06818
                  / lungmod / cardalf  0.01705
Misura 3 max 7.00000 #seq comprese tra 4.04545 e 7.00000 = 3
Misura 4a 13 / lungmod = 0.81250
Misura 4b 14 / lungmod = 0.89062

Sequenza prob. >95% : AA*C-TTTGGG*C*CC
```

Come si vede i risultati ottenuti sono leggermente migliori, ma tutto sommato simili, com'era giusto visto che il modello di partenza era lo stesso per entrambi gli allineamenti.

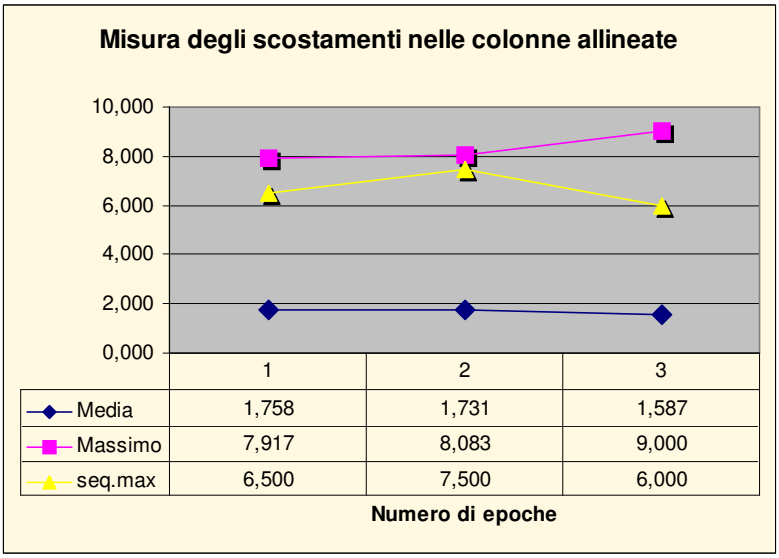
5.2.2 Test sul DNA

Innanzitutto vediamo i test effettuati su un campione inventato con l'alfabeto del DNA. Il modello è stato addestrato con un insieme di sequenze più piccolo di quello su cui poi è stato effettuato l'allineamento in modo, da effettuare una specie di validazione incrociata, osservando se vengono allineate correttamente anche le sequenze su cui il modello non è stato direttamente addestrato. Sono stati effettuati addestramenti con tutti gli algoritmi e su un numero differente di epoche.
Nei grafici si indicherà con:

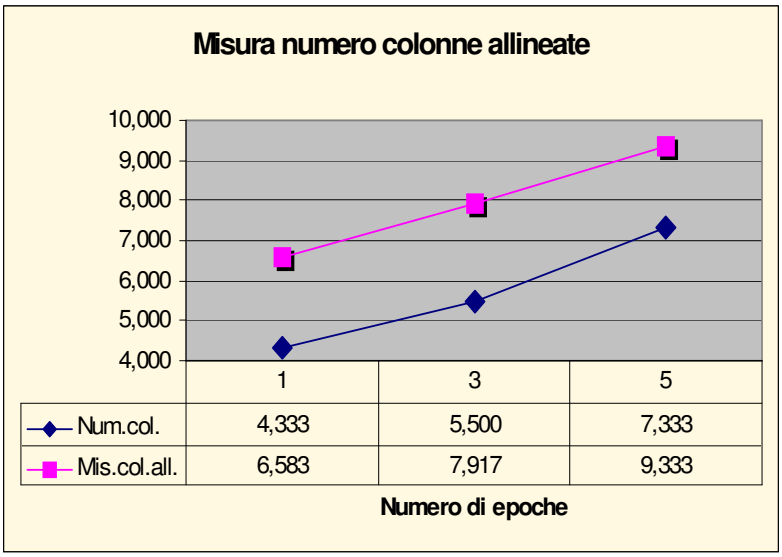
- Media: la media su tutti gli algoritmi di addestramento, di allineamento e su tutte le sequenze.
- Massimo: media dei valori massimi registrati su tutti gli algoritmi di addestramento, di allineamento e su tutte le sequenze.
- Seq.Max. : il numero di sequenze tra (media + massimo) / 2 e il massimo.



Questo grafico è relativo all'andamento, durante l'addestramento, della misura indicata nella sua media, nel suo valore massimo e nel numero di sequenze che hanno una misura alta. I valori non sono normalizzati e sono relativi ad una media effettuata su tutti gli addestramenti dopo 1, 3 e 5 epoche. Notiamo che la media ed il valore massimo diminuiscono, mentre sul numero di sequenze a valore alto c'è un andamento non uniforme: questo è dovuto soprattutto al fatto che questo allineamento è particolarmente semplice per cui si avrebbe già un ottimo allineamento anche solo dopo 3 epoche. Nell'esempio precedente erano state usate solo 3 epoche.

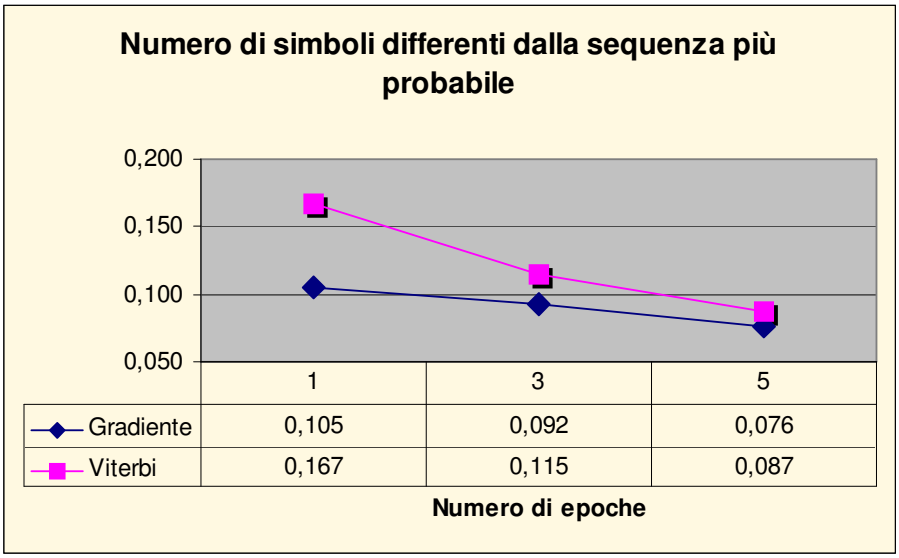


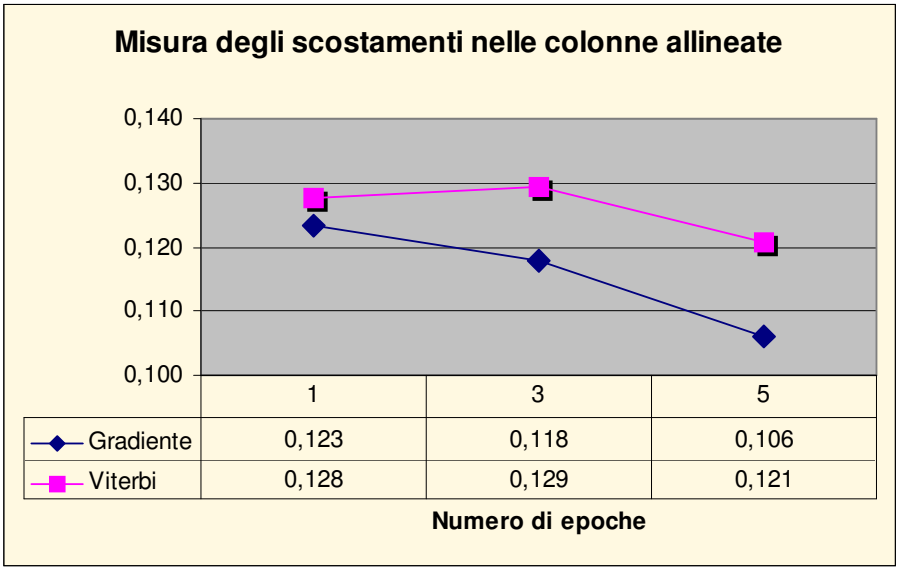
Qui l'andamento sembra abbastanza casuale: la media diminuisce leggermente, il valore massimo aumenta. Questo in realtà è perfettamente accettabile in quanto, per come è definita la misura, è ragionevole attendersi che la media diminuisca in quanto l'allineamento migliora, mentre il valore massimo può anche salire perché mentre l'allineamento migliora e con questo la misura media, la penalizzazione per una sequenza che viene mal allineata aumenta.



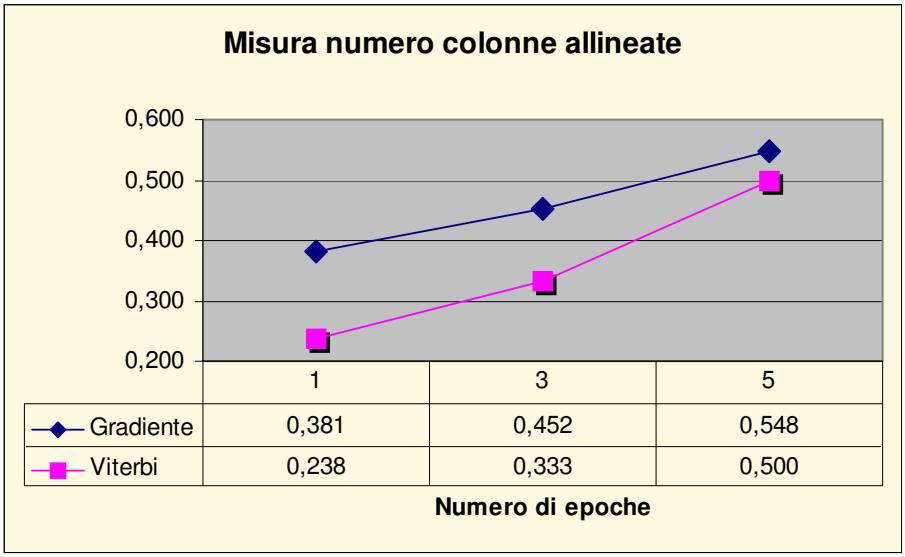
Vediamo che le due misure sul numero di colonne allineate aumentano concordemente. Questa caratteristica si ripete quasi sempre.

Questi grafici si limitano ad indicare il funzionamento delle misure e degli algoritmi nella media, ma non permettono di cogliere le differenze tra i vari algoritmi di addestramento e/o allineamento. Difatti un allineamento fatto con il gradiente richiede un tempo circa 4 volte maggiore di uno fatto con Viterbi: infatti solo il primo utilizza tutte l'informazione contenuta nel grafo. Da questo e dalle considerazioni fatte prima ci attendiamo un funzionamento migliore dell'algoritmo a gradiente. Vediamo che il gradiente riesce ad ottenere risultati migliori anche su un numero basso di epoche, mentre Viterbi necessita di un numero di epoche maggiore.





Anche da qui possiamo trarre conclusioni analoghe a quelle del grafico precedente: infatti l'andamento di Viterbi è a parabola, mentre l'altro è scende costantemente. L'andamento a parabola è indicativo del fatto che c'è stato bisogno di un numero di epoche maggiore per pervenire ad un allineamento buono, in quanto dopo una sola epoca Viterbi, in media, non è stato in grado di riuscire in questo. Il gradiente invece non ha avuto questo problema come dimostra l'andamento sempre decrescente della misura: segno che già dopo un'epoca si è oltrepassato il punto in cui questa misura ha il suo massimo.

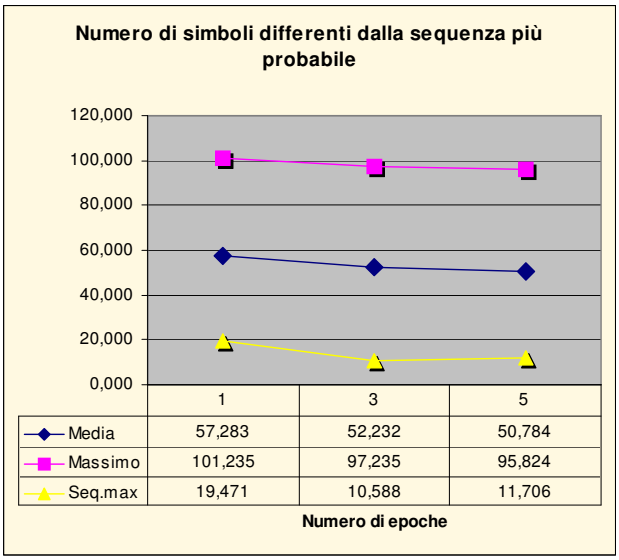


Anche questo conferma quanto detto prima: si può vedere che su un numero alto di epoche i due algoritmi di allineamento si comportano allo stesso modo com'è giusto, prima che intervengano fenomeni di sovraddestramento che qui non sono apparsi. Il numero di colonne allineate è un buon indicatore del sovraddestramento qualora ricominci a scendere: significa che le sequenze fuori dall'insieme di addestramento non vengono più allineate: poiché sono in questo esempio ed in tutti gli altri tra 1/3 e 2/3 dell'insieme totale delle sequenze il loro non allineamento porta ad un immediato peggioramento di questa misura. Si può notare un sovraddestramento soprattutto utilizzando un addestramento EM.

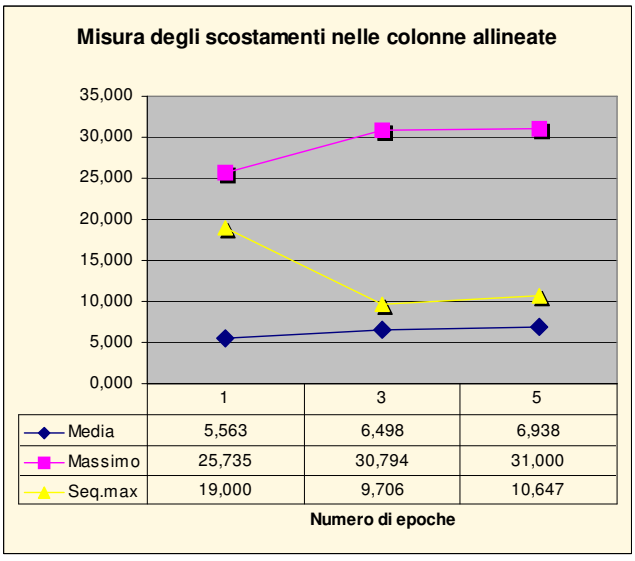
5.2.3 Test sulle proteine

Un test analogo a quello precedente è stato fatto su un insieme di sequenze che hanno un significato biologico. Qui infatti abbiamo utilizzato delle immunoglobuline di diversa provenienza biologica: queste possiedono, per motivi chimici e fisici, almeno due colonne che devono allinearsi. L'insieme di 150 immunoglobuline è stato diviso in più file ognuno con un numero di proteine tra 70 e 110 su cui sono stati fatti gli addestramenti con tutti gli algoritmi. Quindi sono stati fatti gli allineamenti sull'intero insieme. Notiamo che perché una colonna sia allineata al 95% possono esserci solo 5 simboli differenti lungo quella colonna e gli altri 145 devono esseri corretti: quindi se una colonna viene allineata il modello è riuscito a generalizzare, in quanto è pressoché impossibile che più di trenta simboli, scelti in

una rosa di 20, siano casualmente gli stessi. Come prima innanzitutto vediamo i grafici che riassumo l'intero addestramento.

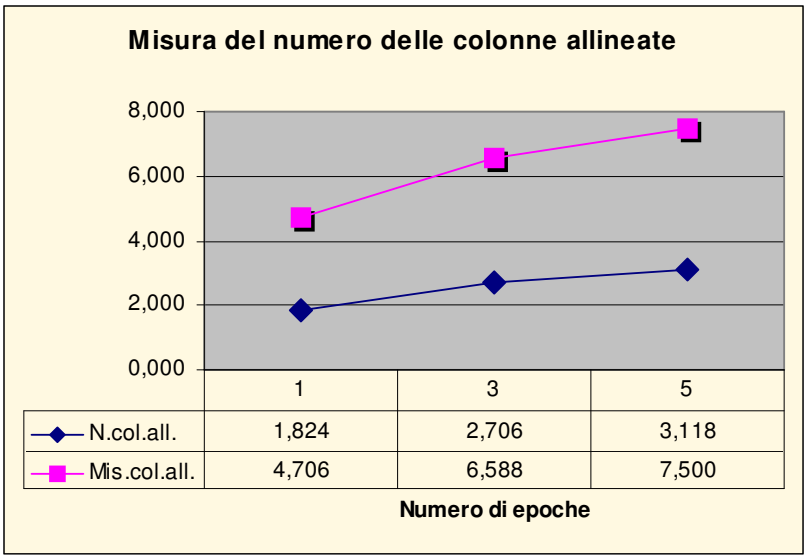


Notiamo che in media c'è una leggera diminuzione di questa misura sia in media sia nel suo valore massimo.

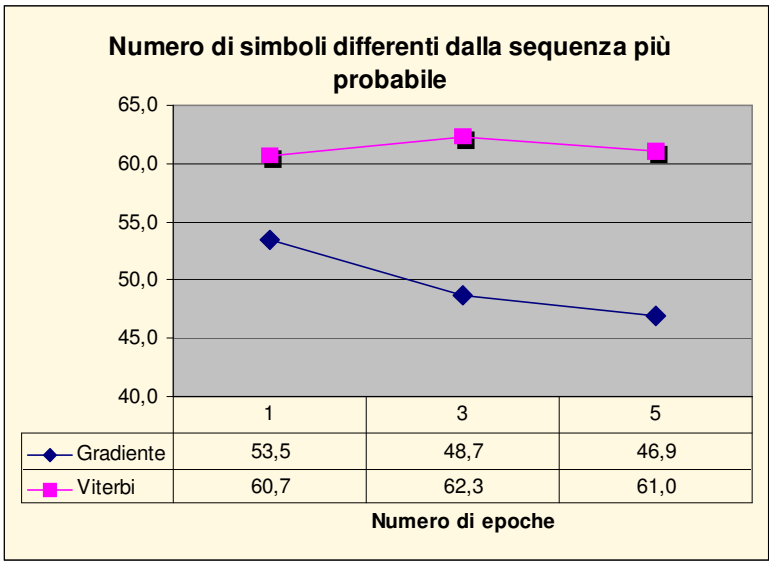


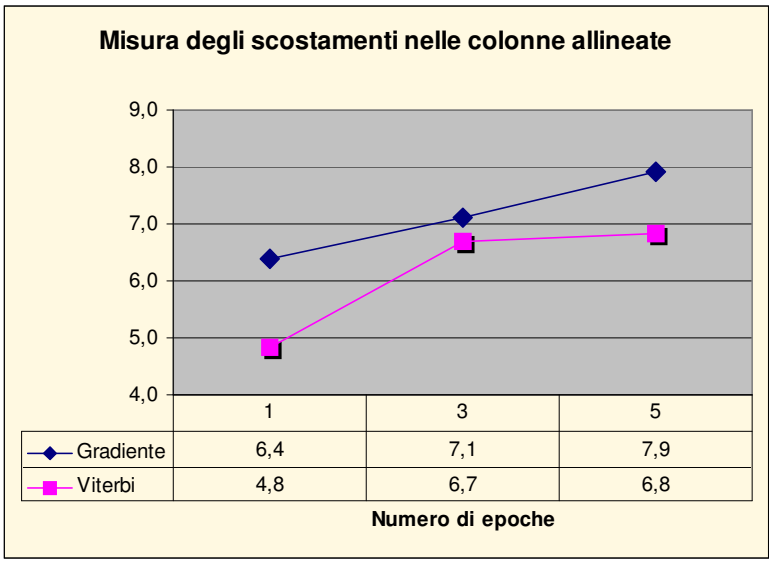
Qui notiamo che questa misura cresce anche se tende a stabilizzarsi dopo 5 epoche almeno nella media: questo può indicare che dopo 5 epoche la misura abbia raggiunto il valore massimo e possa d'ora in poi solo migliorare o andare in sovraddestramento. In realtà sappiamo che sarà comunque un sovraddestramento in quanto la proprietà biologica prima discussa viene evidenziata dopo 3 epoche con

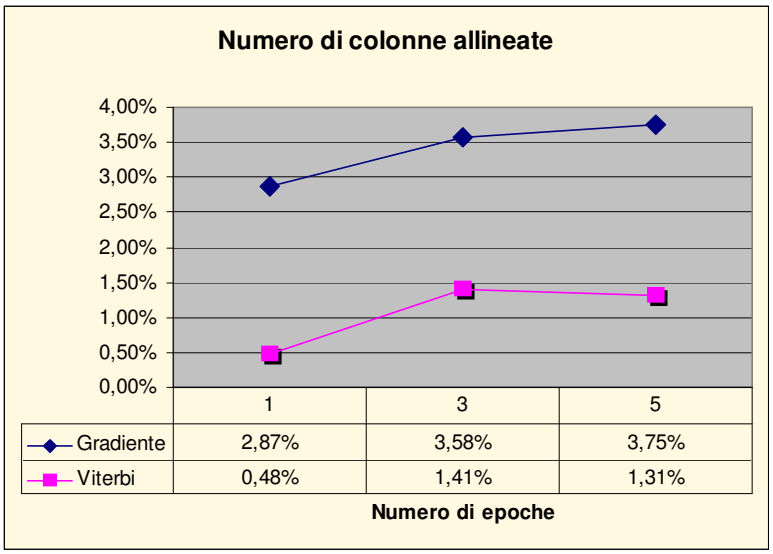
Viterbi e con il gradiente e dopo 5 con EM. Ulteriori allineamenti, interessanti come studio degli allineamenti, non hanno una rilevanza biologica.



Anche questo grafico è concorde con gli altri: questo sottolinea anche la coerenza delle misure adottate e la loro forte correlazione. Notiamo che questi risultati, con eccezione della misura degli scostamenti, abbiamo ottenuto risultati il cui andamento è analogo a quello dell'esempio inventato precedentemente: un altro segno dell'efficacia delle misure considerate. A questo punto è utile considerare i risultati differenziati per algoritmo di allineamento.


Numero di simboli differenti dalla sequenza più probabile
65,0
60,0
55,0
50,0
45,0
40,0
1
3
5
Gradiente
53,5
48,7
46,9
Viterbi
60,7
62,3
61,0
Numero di epoche

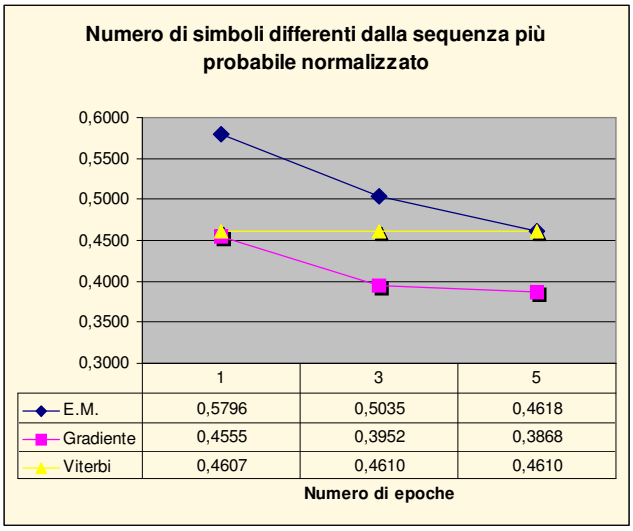

Misura degli scostamenti nelle colonne allineate
9,0
8,0
7,0
6,0
5,0
4,0
1
3
5
Gradiente
6,4
7,1
7,9
Viterbi
4,8
6,7
6,8
Numero di epoche

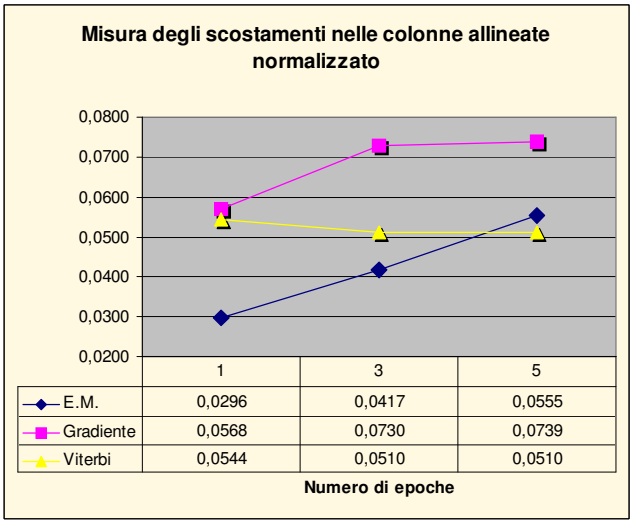

Numero di colonne allineate
4,00%
3,50%
3,00%
2,50%
2,00%
1,50%
1,00%
0,50%
0,00%
1
3
5
Gradiente
2,87%
3,58%
3,75%
Viterbi
0,48%
1,41%
1,31%
Numero di epoche

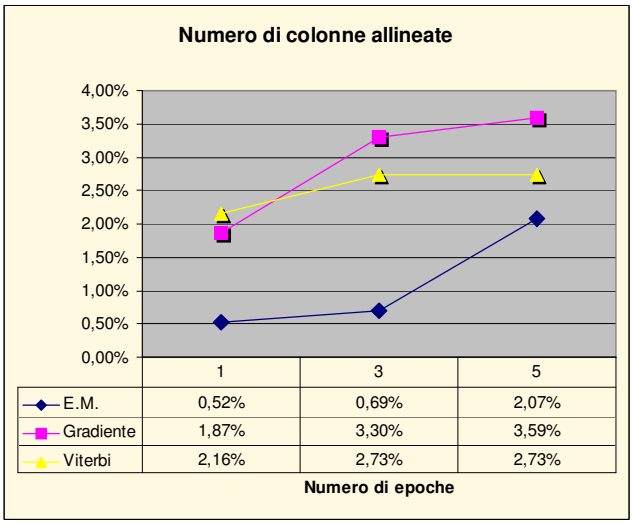
Notiamo che l'algoritmo di Viterbi ha un comportamento anomalo, mentre quello del gradiente è molto più netto nei suoi andamenti: questo è indicativo di come il primo riesca a estrarre in modo peggiore informazioni dal modello poco addestrato, mentre ci riesce molto meglio il secondo.

Un ultimo confronto interessante è tra i vari algoritmi di addestramento.
I grafici della pagina successiva rendono l'idea del funzionamento dei tre algoritmi:

- L'algoritmo che sfrutta tutte le informazioni presenti per l'aggiornamento, chiamato "gradiente" è indubbiamente il migliore, come si evidenzia in tutte e tre le misure.
- L'algoritmo di Viterbi per l'aggiornamento ha un comportamento abbastanza singolare in quanto sembra abbastanza indipendente dal numero di epoche seppure stabilizzato su dei valori del tutto inaccettabili. In realtà questo è dovuto ad un andamento del tutto irregolare nei vari esempi per cui si sono registrati in certi casi aumenti ed in certi altri diminuzioni. Come si era già detto non è un buon algoritmo per l'aggiornamento in quanto considera solo una parte del modello. Peraltro si presta molto male a generalizzare in quanto è stato sempre aggiornato lungo i percorsi più probabili del solo insieme di addestramento.
- L'algoritmo EM ha una non buona velocità di convergenza su poche epoche, mentre su un numero alto di epoche tende ad avviarsi verso gli stessi valori del gradiente. In realtà su un numero alto di epoche è ancora più veloce, ma tende ad adattarsi troppo all'insieme di addestramento e ad andare in sovraddestramento facilmente. Ad esempio, sull'insieme di sequenze di DNA, l'algoritmo EM va in sovraddestramento dopo 25 epoche imparando in modo pressoché perfetto l'insieme di addestramento, mentre il gradiente anche dopo 150 epoche, pur in presenza di risultati ottimi di allineamento già dopo 10 epoche, continua a funzionare bene.


Numero di simboli differenti dalla sequenza più probabile normalizzato
0,6000
0,5500
0,5000
0,4500
0,4000
0,3500
0,3000
1
3
5
E.M. 0,5796 0,5035 0,4618
Gradiente 0,4555 0,3952 0,3868
Viterbi 0,4607 0,4610 0,4610
Numero di epoche


Misura degli scostamenti nelle colonne allineate normalizzato
0,0800
0,0700
0,0600
0,0500
0,0400
0,0300
0,0200
1
3
5
E.M. 0,0296 0,0417 0,0555
Gradiente 0,0568 0,0730 0,0739
Viterbi 0,0544 0,0510 0,0510
Numero di epoche


Numero di colonne allineate
4,00%
3,50%
3,00%
2,50%
2,00%
1,50%
1,00%
0,50%
0,00%
1
3
5
E.M. 0,52% 0,69% 2,07%
Gradiente 1,87% 3,30% 3,59%
Viterbi 2,16% 2,73% 2,73%
Numero di epoche