Fondamenti di Teoria dell'Informazione ed Analisi di Sequenze di Dati

E. Caglioti

8 marzo 2005

Indice

1 Introduzione

Argomento del corso è lo studio delle sequenze di caratteri (DNA, proteine, testi,...): studio delle loro proprietà statistiche e dei metodi che si usano per classificarle riconoscerle confrontarle.

Questo problema è particolarmente interessante ed attuale in quanto sempre più sono accessibili enormi quantità di dati biologici (e non) in questa forma.

Lo studio delle sequenze di caratteri ha una lunga storia, ed è argomento dela teoria dell'informazione, della biologia, della statistica, della teoria della probabilità, della meccanica statistica, etc...

Attraverso le tecniche matematiche sviluppate nell'ambito della teoria dell'informazione e della statistica è spesso possibile dare una caratterizzazione della sequenza che può essere particolarmente utile da un punto di vista biologico. Problemi biologici che si possono affrontare in questo ambito sono: distinguere la zona codificante di un gene da quella non codificante, e più in generale segmentare in frazioni con diverse funzioni una porzione di DNA - identificare la famiglia cui appartiene una data proteina - costruire un albero filogenetico - etc.

Qui, utilizzando se possibile esempi semplici, verranno esemplificati alcuni concetti come l'entropia di una sequenza, l'entropia relativa, cos'è una catena di Markov. Questi concetti verranno poi utilizzati in alcuni esempi pratici tratti dalla biologia, dalla linguistica e dall'informatica:
- impareremo come funzionano i programmi per la compressione di dati (zipper) che sono di uso comune su tutti i computer e vedremo come questi programmi possano essere utilizzati per classificare un insieme di testi o di sequenze genomiche
vedremo come funzionano alcuni dei programmi più comuni di allineamento di sequenze
- impareremo a segmentare delle sequenze di dati in base alle loro proprietà statistiche
- vedremo alcuni dei concetti che sono alla base della costruzione degli alberi filogenetici
- infine vedremo alcuni rudimenti di un linguaggio doi programmazione, il PERL, particolarmente adatto per operare su sequenze di caratteri.

Molti degli argomenti trattati qui saranno utili per il corso di bioinformatica a questo parallelo.

In particolare uno degli scopi del corso è quello di permettere allo studente di capire quali sono i concetti matematici (e non) che sono alla base dei programmi che sono utilizzati nell'ambito della bionformatica. Tale conoscenza, spero, sarà utile per utilizzare tali strumenti criticamente e creativamente.

Per seguire questo corso queste dispense sono più che sufficienti. Alla fine ho aggiunto alcuni riferimenti bibliografici che possono essere utili per eventuali approfondimenti.

2 La teoria dell'informazione

La teoria dell'informazione, nasce, con Claude Shannon, per risolvere dei problemi molto pratici. Per esempio: volete trasmettere un messaggio - lo trasmettete utilizzando un "canale" (la linea telefonica, il telegrafo,...) - il canale è disturbato quindi ci sono degli errori.

Quanta informazione in più dovete mandare per essere sicuri che il destinatario sia in grado di ricostruirlo correttamente?

Un altro esempio, simile, ma più facile da capire, è il seguente: avete una sequenza da memorizzare (per esempio un film in formato digitale, un immagine o la Divina Commedia) volete codificarlo in maniera tale che occupi il minimo spazio di memoria - cosa potete fare ?

Qui ci occuperemo di questo problema e vedremo che per risolverlo saranno necessari una serie di concetti matematici che altrimenti sarebbero potuti sembrare astratti.

Prendiamo un esempio semplice: dovete trasmettere la sequenza

"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

Se voi doveste comunicare al telefono questa sequenza a qualcuno, probabilmente non gli direste

"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

ma

"50 volte a".

Potete notare che questa sequenza è più corta, molto più corta della precedente.

Se invece voi doveste tramettere la sequenza

"Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura"

la leggereste per intero. In realtà come vedremo anche questa sequenza può essere "accorciata" ma non tanto quanto l'altra.

Rozzamente il problema che vogliamo affrontare qui è quindi il seguente: dato un testo vogliamo memorizzarlo nella maniera più corta possibile. Cerchiamo di essere più precisi: dato un testo vogliamo trasformarlo in un altro testo (più corto) dal quale si possa ricavare univocamente il testo di partenza. Anche così non siamo stati precisissimi. Dobbiamo dire cosa intendiamo per lunghezza di una sequenza: ci serve una definizione universale. Questa è data dal concetto di bit. Un bit è una cifra binario : 0 o 1. Con un bit potete rappresentare due possibilità, con due 4, e con k bit 2^k possibilità. Per inciso,

su tutti i computer che usiamo, un testo è una sequenza di caratteri, ed un carattere è rappresentato con un byte : 1byte = 8 bit. Quindi abbiamo a disposizione 256 caratteri diversi. Ciò vuol dire che la Divina Commedia che è costituita da circa 550.000 caratteri viene rappresentata da 550kbytes. Se invece la zippate (per esempio con gzip o winzip che è sostanzialmente equivalente) vi bastano 219kbytes e con bzip scendete a 170kbytes. Per Manzoni (I promessi sposi) 1,285Mbytes originali con gzip si riducono a 479kbytes, mentre con bzip diventano 356kbyte.

Per inciso il DNA nella parte codificante non viene compresso dai normali programmi di compressione (cioè la lunghezza del file dopo la compressione è sostanzialmente uguale a quella del file originale) mentre la parte non codificante si comprime bene.

Un problema molto interessante è quello della compressione di sequenze proteiche come singole proteine o proteomi. Queste sequenze sono infatti molto difficili da comprimere. In un lavoro recente [NW], si veda anche [**?**], l'utilizzazione di metodi piuttosto raffinati per la compressione di sequenze, opportunamente modificati al fine di comprimere proteine, non ha permesso di comprimere alcuni proteomi più dell 1%.

2.1 L'entropia di Shannon

Per procedere oltre abbiamo bisogno di introdurre un importante concetto : l'entropia.

L'entropia è una misura della quantità di informazione.

Vediamo alcuni esempi.

Dobbiamo indicare un numero da 1 a 10. Quanti bit ci vogliono?

Prima di tutto possiamo notare che con 4 cifre binarie siamo in grado di scrivere 16 numeri diversi. Quindi ci bastano 4 bit per scrivere un numero da 1 a 10. Si potrebbe, giustamente, obiettare che questa codifica non è ottimale: infatti 6 numeri verranno sprecati.

In effetti supponiamo di tramettere 2 numeri da 1 a 10. Una possibilità è trasmettere 2 sequenze di 4 bit, utilizzando in tutto 8 bit. Un altra è notare che le coppie di numeri da 1 a 10 sono in tutto 100 - un numero da 1 a 100 può essere codificato con solo 7 cifre binarie ($2^7 = 128$) - quindi possiamo tramettere due cifre decimali con 3.5 bit per carattere.

La regola generale dovrebbe essere chiara. Codificare k numeri da 1 a 10 equivale a codificare un numero da 1 a 10^k. Per farlo ci basteranno n bit dove n è il più piccolo intero per cui $2^n \geq 10^k$.

Notate che estraendo i logaritmi in base 2 troviamo $n \geq k \log_2(10)$ ed anche $n \leq k \log_2(10)+1$ Ciò vuol dire che possiamo scegliere $n = [k \log_2(10)]+1$,

dove $[x]$ denota il più piccolo intero minore o uguale a x. Per capirsi $[1.9] = 1$, $[2.1] = 2$.[1]

Quindi il numero di bit per carattere che usiamo è $\frac{[klog_2(10)]+1}{k}$.

Questo numero è sempre maggiore di $\log_2(10)$ ma per k grandi tende al limite ottimale di $\log_2(10)$ bit per carattere.

Quest'ultimo numero è l'entropia di Shannon di una sequenza di 10 caratteri indipendenti ed equiprobabili. Il risultato si generalizza immediatamente a sequenze di n caratteri indipendenti. La loro entropia è $\log_2(n)$. Ciò vuol dire che la quantità di informazione di un numero scelto tra n è in media $\log_2(n)$.

Possiamo notare che l'entropia è una grandezza additiva (estensiva): cioè che l'entropia di due stringhe indipendenti è la somma delle entropie delle due stringhe prese separatmente. Più precisamente l'entropia è in generale subadditiva. Cioè l'entropia di due stringhe (o di due eventi generici) è minore o uguale della somma delle entropie dei due eventi.

Infatti se la seconda stringa non fosse indipendente dalla prima allora essa avrebbe una entropia minore una volta nota la prima. L'esempio più banale si ha quando la seconda stringa è uguale alla prima stringa. In questo caso l'entropia delle due stringhe è (essenzialmente) uguale a quella della prima.

Veniamo ora ad un caso leggermente più difficile. Consideriamo sempre stringhe di caratteri indipendenti ma non equiprobabili. Per semplificarci la vita consideriamo stringhe di 0 ed 1 estratti indipendentemente con probabilità p ed $1-p$. Quando $p = 1/2$ siamo esattamente nel caso descritto pocanzi per $k = 2$. In questo caso l'entropia per carattere è $\log_2 2 = 1$, cioè un bit per carattere.

Vediamo ora un altra sequenza di 0 ed 1 indipendenti ma con diverse probabilità. Per esempio prendete un dado e ogni volta che esce un numero diverso da 6 scrivete 0 altrimenti scrivete 1. Questa sequenza può essere compressa in una sequenza più corta. Cerchiamo di capire perchè.

Nella sequenza che abbiamo descritto per seconda gli 1 escono più raramente (una volta su sei).

Se prendete una sequenza abbastanza lunga essa conterrà circa $5/6L$ 0 ed $1/6L$ 1. Ora il numero di sequenze che hanno $5/6L$ 0 ed $1/6L$ 1 è dato da $\binom{L}{5/6L}$. Questa grandezza è circa $2^S L$, dove $S = -1/6\log_2(1/6) - 5/6 log_2(5/6)$, è l'entropia per carattere della nostra sequenza.

L'ultima affermazione è una conseguenza della formula di Stirling (si veda la sezione corrispondente).

Quindi, per L abbastanza grande, codificare le sequenze di lunghezza L

[1] In realtà la funzione da usare sarebbe non $[x] + 1$ ma *l'intero successivo ad* x. La differenza si ha solo quando x è intero cosa che in questo esempio non accade.

è equivalente a codificare un numero da 1 a $2^L S$, e quindi il costo è di S bit per carattere.

Generalizzando questo risultato troviamo che una sequenza di simboli indipendenti con probabilità $p_1, ..., p_n$ hanno un entropia per carattere pari a

$$-\sum p_k \log_2 p_k.$$

Questa formula si può anche leggere così: la quantità di informazione di un carattere che ha probabilità p è $-\log_2(p)$. Quindi la quantità di informazione media per carattere sarà data dalla somma sui caratteri della probabilità del carattere per la sua quantità di informazione.

2.2 La complessità di Chaitin-Kolmogorov

Un altro concetto matematico fondamentale in quest'ambito è la complessità di Chaitin-Kolmogorov.

La complessità (o impropriamente entropia) di Chaitin Kolmogorov di una stringa è

la lunghezza del più piccolo programma in grado di riprodurre quella stringa.

Questo concetto andrebbe precisato matematicamente: la lunghezza del più piccolo programma dipende ovviamente dalla macchina su cui state lavorando.

Esiste però una classe universale di macchine cui potete fare riferimento che permette di definire precisamente questo concetto.

Qui ci occuperemo solamente della differenza pratica dei due approcci. L'entropia di Shannon è un concetto statistico: cioè ci dice in media quanti caratteri dobbiamo usare per codificare una stringa emessa da una certa sorgente. Per esempio abbiamo considerato una sorgente che emette numeri da 1 a 10 indipendenti ed abbiamo visto come trametterе un tale segnale. La complessità di Chaitin-Kolmogorov si può definire per una singola stringa indipendentemente dalle proprietà statistiche della sorgente che l'ha emessa o addirittura dal fatto che esista una tale sorgente.

Per esempio se la sorgente che emette numeri da 1 a 10 indipendenti avesse emesso una stringa composta di 100 uni questa stringa sarebbe stata potuta essere zippata in maniera molto più efficiente che con $\log_2(10)$ bit per carattere.

Ma per quanto visto prima è chiaro che in media non potremo codificare le sequenza emessa da questa sorgente con meno di $\log_2(10)$ bit per carattere. Ciò vorrà dire che devono esistere delle sequenze che verranno compresse peggio della media.

Senza entrare in ulteriori dettagli vogliamo notare qui che l'idea, molto importante ed attuale di Chaitin e Kolmogorov, è quella di comprimere una stringa in quanto tale e non la tipica stringa che esce da una sorgente.

3 Algoritmi di Compressione

Quello degli Algoritmi di compressione è un campo dei più fecondi dell'informatica e della teoria dell'informazione. Su tutti i computer sono infatti presenti programmi che permettono di comprimere e decomprimere un testo, una foto, un film, un brano musicale . Per fare solo alcuni esempi programmi di compressione tra i più comuni sono gzip, winzip, stuffit expander... (testi e sequenze simili) - gif, jpeg,.... (per le immagini) - mp3 e molti altri per la musica - dvd, mp4 (filmati).

La necessità è ovvia. I file compressi occupano meno spazio e si trasmettono in meno tempo. Per esempio un non grande incremento nella capacità di compressione dei filmati permetterebbe di utilizzare i cd invece dei dvd.

Probabilmente uno dei più antichi "algoritmi d compressione" è il codice Morse. Morse voleva tramettere un testo inglese con solo tre caratteri punti linee e spazi. Il problema che lui affrontò era quello di renfdere il messaggio Morse il più corto possibile. Decise quindi che il codice di un carattere fosse tanto più corto tanto più il carattere era probabile. Per esempio la codifica della lettera "e" è ., mentre il codice per la $''z''$ è $--...$

Ovviamente Morse aveva molti vincoli, in particolare il fatto che il ricevente fosse analogico (un uomo) il che non permetteva di ottimizzare ulteriormente il codice tramettendo, per esempio, due caratteri alla volta o in altro modo.

Qui vedremo alcuni dei moderni algoritmi di compressione. In particolare il codice Huffman che è uno dei più usati algoritmi di compressione statistici e l'algoritmo di Lempel e Ziv - LZ77 - che è la base del più comune algoritmo di compressione: gzip, winzip ...

3.1 L'algoritmo di Huffman

L'algoritmo di Huffman è il migliore metodo per comprimere, carattere per carattere, una sequenza di caratteri indipendenti la cui probabilità sia nota. Questa sua proprietà di ottimalità fa si che esso compaia in ogni programma di compressione dati. Ciò è dovuto al fatto che tutti i programmi di compressione dati funzionano traformando la sequenza da comprimere in un altra sequenza di caratteri (il più possibile) indipendenti.

L'idea alla base dell'algoritmo è molto semplice: ogni carattere sarà cod-

ificato da una certa sequenza di bit - più un carattere è probabile, più lo codificheremo con una sequenza corta.

Siamo più precisi: abbiamo n caratteri di probabilità $p_1, ..., p_n$. Se codifichiamo il carattere k-esimo con una stringa lunga L_k bit la lunghezza media per carattere che otterremo sarà $\sum p_k L_k$. Dobbiamo quindi minimizzare sulle scelte possibili delle lunghezze l'espressione precedente. Notiamo che ci sono dei vincoli: a carattere diverso va associata una sequenza diversa, inoltre devo essere in grado di riconoscere dove termina una sequenza.

Per esempio: supponiamo di avere i caratteri A, B, C e di volerli codificare con delle stringhe di bit. Non potrò scegliere le stringhe: $A = 0, B = 1, C = 01$, perchè in questo caso leggendo per esempio 01 non potrò dire se rappresenta una C o AB.

Questo problema si può risolvere facilmente costruendo un albero di decisione. Vediamo cosè un albero di decisione ed in particolare come si costruisce l'albero di Huffman.

Conviene iniziare da un esempio. Supponiamo di avere 4 caratteri: A, B, C, D. Siano $p_A = .4, p_B = .3, p_C = .25, p_D = .05$, la loro probabilità di occorrenza. L'algoritmo di Huffman fuziona così: si prendono le due probabilità pi'u basse e si uniscono ad un nuovo nodo con due rami. Al ramo che va al simbolo di probabilit'a più alta associeremo il simbolo 0 all'altro 1. Siamo quindi rimasti con 3 nodi: il nodo A, il nodo B ed il nodo 1, di probabilità $p_A = .4, p_B = .4, p_1 = p_C + p_D = .25$. A questo punto ripetiamo l'operazione. Quindi uniamo B ed 1 attraverso un nuovo nodo, 2, che avrà probabilità $p_2 = p_B + p_1 = .6$. Il procedimento finirà al prossimo passo quando uniremo in nodi A e B attraverso il nuovo nodo 3 che sarà l'inizio dell'albero.

La sequenza che codifica per un dato carattere si ottiene semplicemnte seguendo lungo l'albero il percorso che porta dalla radice ad un dato carattere. Quindi A sarà codificato con 1, B con 00, C con 010, e D con 011.

Quindi A sarà codificato con un bit, B con due, e C e D con 3. Il rate di compressione r sarà dunque $r = p_A + 2p_B + 3(p_C + p_D) = 1.9$ bit per carattere.

Per quanto abbiamo detto questa r sarà maggiore dell'entropia di Shannon S. Per inciso in questo caso l'entropia di Shannon è data da $S \simeq 1.766$. bit per carattere.

Possiamo notare che la codifica così trovata è univoca. Infatti ad ogni carattere è associato un percorso che arriva ad una foglia terminale dell'albero.

Tornando al caso generale notiamo che ci sono dei vincoli sulle L_k. Il

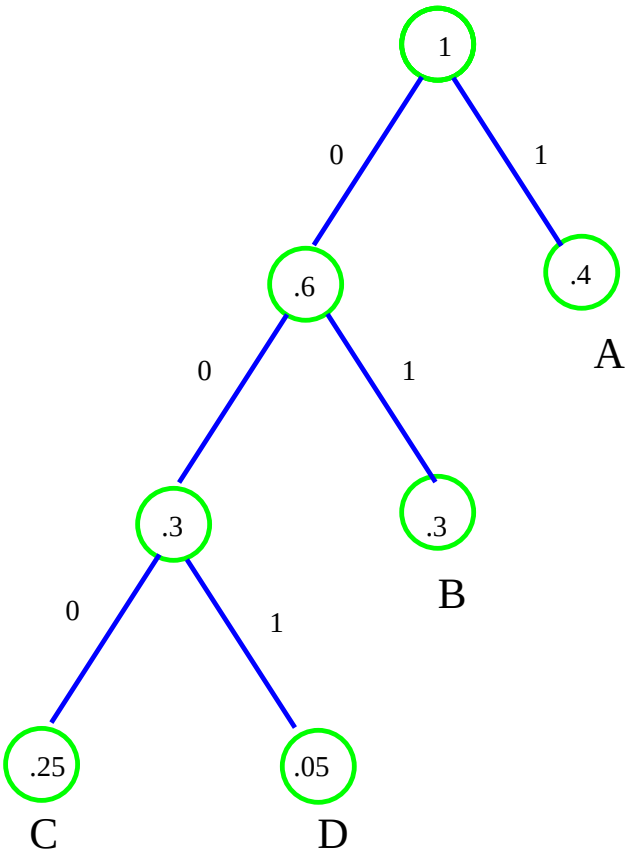


Figura 1: Algoritmo di Huffman

vincolo è che la somma sulle foglie terminali dell'albero di 2^{-L_k}, dove L_k è il numero di passi per arrivare dalla radice al nodo è esattamente 1. Dimostriamo questo fatto. L'albero composto dalla sola radice ha una sola foglia terminale (coincidente con la radice) a distanza 0 dalla radice. In effetti $2^0 = 1$. Da quest'albero possiamo biforcare passando all'albero con due foglie terminali. Le due foglie sono a distanza 1 dalla radice, quindi ognuna vale 1/2 e la somma fa 1. Il risultato si generalizza a qualunque albero. Infatti ogni albero si può ottenere partendo dall'albero costituito dalla sola radice e effettuando successive biforcazioni. Ad ogni biforcazione il valore dei due nodi è metà del valore del punto a cui sono attaccati. Quindi il loro valore totale è eguale a quello di quel nodo nell'albero precedente. Quindi il valore dell'albero nuovo è eguale al valore dell'albero vecchio. Ciò vuol dire che l'operazione di biforcazione non cambia il valore dell'albero che quindi è eguale a quello di partenza che è 1.

Quindi dobbiamo minimizzare

$$\sum_k p_k L_k, \tag{3.1}$$

con il vincolo che

$$\sum_k 2^{-L_k} = 1,$$

ed ovviamente con il vincolo che gli L_k siano numeri interi. Possiamo vedere che il costo della codifica sarà sempre maggiore o uguale al limite ideale dato dall'entropia di Shannon.

Infatti il minimo della (3.1) si ottiene quando $L_k = -\log_2 p_k$. Con questa scelta il costo medio sarà proprio dato dall'entropia di Shannon. Però in generale gli L_k così scelti non saranno interi. Quindi non potremo sceglierli così e non potremo quindi raggiungere il limite ottimale.

Dimostriamo questa affermazione. Essa si potrebbe ottenere come applicazione del metodo dei moltiplicatori di Lagrange. Qui otterremo questo risultato in maniera diretta, utilizzando la diseguaglianza di Jensen per la funzione logaritmo che è concava.

Consideriamo la differenza Δ tra il costo ideale ed uno generico:

$$\begin{aligned}\Delta &= \sum_k -p_k \log_2(p_k) - \sum_k p_k L_k = \sum_k p_k \left(log_2(p_k) - log_2(2^{L_k})\right) \\ &= \sum_k p_k \log_2\left(\frac{2^{-L_k}}{p_k}\right) \leq \log_2\left(\sum_k \frac{p_k 2^{-L_k}}{p_k}\right) = \log_2\left(\sum_k 2^{-L_k}\right) \\ &= \log_2(1) = 0.\end{aligned} \tag{3.2}$$

La diseguaglianza di Jensen è stata applicata nella seconda riga nella forma $\sum_k x_k \log y_k \leq \log \sum_k x_k y_k$,

dove $x_k = p_k$, e $y_k = 2^{-L_k}/p_k$.

Quindi in generale non possiamo raggiungere il limite ottimale.

Si può però dimostrare, cosa che non faremo, che il massimo errore è di un bit per carattere.

3.2 LZ77

L'algoritmo di compressione inventato da Lempel e Ziv nel 1977 ha rivoluzionato il modo di comprimere i file. Si vuole comprimere un file senza avere nessuna conoscenza a priori della sua statistica. Questo algoritmo, che è alla base di molti dei compressori che si trovano su tutti i computer (gzip,

winzip), è molto semplice e gode della seguente rimarchevole proprietà: nel limite per $L \to \infty$ il numero di bit per carattere tende all'entropia di Shannon (cioè al limite ottimale). Ovviamente non è ottimale ma è una buona approssimazione dell'ottimo.

L'idea è molto semplice. Vogliamo comprimere la stringa $\sigma_1, \sigma_2, ...$ Supponiamo di avere già codificato fino a σ_k e vediamo come si codifica la parte successiva. Si considera la stringa $\sigma_{k+1}, ...$ Si cerca nella stringa $\sigma_1, ..., \sigma_k$ la più lunga sottosequenza (stringa di simboli consecutivi) uguali alla stringa che parte da $k+1$. Chiamiamo la lunghezza di questa stringa l e supponiamo che sia la stringa $\sigma_j, ..., \sigma_{j+l-1}$. Allora la stringa $\sigma_{k+1}, ..., \sigma_{k+l}$ verrà codificata con due numeri: la distanza tra le due stringhe $k-j$ e la lunghezza della stringa l. Utilizzando il teorema di Shannon-McMillan (si veda la sezione corrispondente) è possibile vedere che la lunghezza tipica della stringa trovata è $\frac{\log_2(k)}{S}$, dove S è l'entropia della sorgente. Quindi con questa operazione codificheremo in media $\frac{\log_2(k)}{S}$ caratteri. La codifica di questa stringa ci costerà invece $\log_2(k)$ bit (il costo per codificare un numero da 1 a k) cui va aggiunto il costo per la codifica della lunghezza. Quest'ultimo sarà quindi circa $\log_2\left(\frac{\log_2(k)}{S}\right)$. La rate di compressione r, si otterrà dividendo il costo di codifica per il numero di caratteri codificati.

Otteniamo quindi

$$r = \frac{\frac{\log_2(k)}{S} + \log_2\left(\frac{\log_2(k)}{S}\right)}{\log_2(k)} = S + O\left(\frac{\log_2(\log_2(k))}{\log_2(k)}\right). \tag{3.3}$$

Possiamo quindi notare che quando k è grande il rate di compressione si avviacina sempre di più al limite ottimale dato da S. Potete anche notare che la convergenza è molto lenta, infatti i logaritmi crescono molto poco rapidamente. Per avere un idea, se $k = 10^6 \simeq 2^{20}$, abbiamo $\log_2(k) \simeq (20)$, e $\log_2(\log_2(k)) \simeq \log_2(20) \simeq 4.3$. Potete quindi notare che la correzione è molto grande.

Non abbiamo considerato il caso in cui non ci sia nessun match. Ciò accade quando il carattere σ_{k+1} non era ancora apparso. In questo caso l'algoritmo invece di scrivere una distanza ed una lunghezza scrive direttamente il carattere. Possiamo però notare che questa eventualità si presenta solo poche volte.

Nelle applicazioni pratiche l'algoritmo è ovviamente ottimizzato in vari modi. Per esempio gzip cerca il massimo match all'indietro fino ad una distanza di 32kbytes e sostituisce solo match più lunghi di due caratteri. Inoltre gzip utilizza una codifica Huffman per le distanze e per le lunghezze ottimizzata con tutti i possibili accorgimenti.

3.3 L'esperimento di Shannon

Descriviamo qui l'esperimento di Shannon per misurare l'entropia della lingua inglese.

L'idea di Shannon è molto semplice. Per calcolare l'entropia dell'inglese devo comprimere al meglio un testo scritto in inglese. Il problema è che non posso usare una macchina per farlo: una macchina, per quanto brava sia, non è in grado di battere un uomo nella comprensione di una lingua (almeno fino ad ora).

Quindi Shannon inventò una procedura che coinvolgeva un uomo e sopratutto che fosse facile per un uomo.

La procedura è la seguente: dopo aver letto un certo numero di caratteri (diciamo 100) di un testo viene chiesto alla persona di indovinare il carattere seguente. Se lo indovina al primo colpo si scrive 1 e si passa al carattere successivo. Altrimenti la persona fa un altro tentativo e così via fino a quando non lo indovina. Se lo indovina in k tentativi scrive k e si passa al carattere successivo.

Adesso dobbiamo associare una procedura di decompressione. Questa è molto semplice. Bisogna immaginare che il nostro uomo abbia un clone che agisca esattamente come lui. (Notare che questa è una richiesta che è banalmente verificata da qualunque programma per calcolatore) Allora il clone farà esattamente le stesse domande e quindi il sapere dopo quanti tentativi il carattere è stato indovinato determina completamente il carattere.

A questo punto possiamo comprimere la sequenza (se sufficientemente lunga) con esattamente $\sum_{k=1} -p_k \log_2 p_k$, dove p_k è la probabilità di azzeccare in k tentativi.[1]

Si può vedere che questa ottenuta è in effetti solo una stima dall'alto dell'entropia dell'inglese. Non vogliamo addentrarci su questo punto. In ogni caso Shannon misurando questa quantità in esperimenti reali trovò una stima dall'alto di 1.3 bit per carattere. La stima dal basso da lui trovata era di 0.7 bit per carattere.

Notare che nessuno dei compressori di uso comune scende sotto due bit per carattere.

In ogni caso compressori costruiti appositamente per questo problema sono arrivati ad 1.46 bit per carattere.

Perchè? Questo è un problema molto interessante.

[1] Potete provare il metodo di Shannon con l'applet che trovate a
http://math.ucsd.edu/~ crypto/java/ENTROPY/
mentre in
http://www.mat.uniroma1.it/~ caglioti/entit2.html
ho scritto un applet che implementa l'esperimento nel caso della lingua italiana.

Quando un uomo legge un testo ha un modo di categorizzare quanto sta leggendo in maniere molto difficile da insegnare ad una macchina. Non solo a livello grammaticale e sintattico ma anche a livello semantico.

Un compressore invece riesce a capire strutture che non si estendono su di un testo più che per una lunghezza dell'ordine di $\log_2 L$, dove L è la lunghezza del testo memorizzato. Anche pensando ad un testo che occupi 1000 DVD da 10 Gigabytes l'uno non supereremmo una lunghezza di 20 caratteri.

Quindi il calcolatore perde sull'uomo perchè è più stupido (primo) e (secondo) perchè non può porre rimedio alla sua stupidità utilizzando una memoria infinita.

3.4 Cross Entropy

I metodi che si usano per comprimere una sequenza si possono usare per confrontare due diverse sequenze.

Partiamo da un esempio molto semplice. Supponiamo di trasmettere un testo italiano con un codice Morse ottimizzato per l'inglese. La lunghezza del testo la chiameremo $L_{I,E}$. Trasmettiamo poi lo stesso testo con un codice Morse ottimizzato per l'italiano. Otterremo un testo di lunghezza $L_{I,I}$. Ragionevolmente $L_{I,I} < L_{I,E}$ quindi $L_{I,E} - L_{I,I}$ sarà una grandezza positiva. Mentre $L_{I,I}$ (divisa per il numero di caratteri) può essere vista come una misura dell'entropia dell'italiano $L_{I,E}$ e $L_{I,E} - L_{I,I}$ (sempre divise per il numero di caratteri) forniscono una misura della cross entropy e dell'entropia relativa tra inglese ed italiano rispettivamente.

La grandezza $L_{I,E} - L_{I,I}$ sarà chiaramente tanto più piccola quanto più le due lingue sono simili. Darà quindi una misura della distanza tra le due lingue.

Il codice Morse viene dall'ottimizzazione della lunghezza di tramissione di un testo che viene codificato un carattere alla volta con punti linee e spazi. È quindi ben lontana dall'essere una codifica ottimale. Allo stesso modo quelle date sopra sono solo delle rozze approssimazioni dell'entropia della cross entropy e dell'entropia relativa.

L'idea però è corretta. Supponiamo di avere due sorgenti A e B con diverse proprietà statistiche. Consideriamo una codifica ottimale di A e codifichiamo con questa una sequenza emessa da B. Quello che perderemo nel fare questa codifica sarà una misura dell'entropia relativa di B rispetto ad A.[1]

[1] Quanto appena detto non si può prendere alla lettera. Per codifica ottimale intendiamo una codifica il cui rapporto di compressione tende all'entropia della sorgente quando la lunghezza della stringa tende ad infinito. In realtà perchè a partire di una codifica ottimale per una sorgente si possa calcolare la cross entropy tra questa ed un altra sorgente bisogna

3.4.1 Algoritmo di Ziv e Merhav

L'algoritmo base per calcolare la cross entropy e quindi l'entropia relativa usando tecniche di compressione è l'algoritmo di Ziv e Merhav. In quest'algoritmo, molto simile ad LZ77, si codifica il testo B non rispetto a se stesso ma rispetto ad un testo di riferimento A.

Più precisamente sia $a_1, ..., a_{N_a}$, il testo di riferimento e sia $b_1, ..., b_{N_b}$ il testo da comprimere. Si parte dall'inizio di b e si cerca il più grande k per il quale $b_1, ..., b_k$ appare in a. Quindi si riparte da b_{k+1} et cetera. Il costo di ogni sostituzione è $\log_2 N_a$, che è il numero di bit necessari per individuare quale stringa di A viene sostituita.

Alla fine avremo codificato tutta la stringa b utilizzando M sostituzioni. Allora la stima della cross entropy così ottenuta sarà

$$\frac{M \log_2 N_a}{N_b}.$$

Nota. Quest'algoritmo funziona quando N_a, N_b, sono sono molto maggiori di $\log N_a$ e di $\log N_b$. Nell'articolo originale di Ziv e Merhav si considera $N_a = N_b$.

3.4.2 Cross entropy tra sequenze Bernoulli

Supponiamo di avere due sequenze Bernoulli che emettono sequenze di k simboli (per semplicità i numeri da 1 a k) con probabilità rispettivamente $p_1, ..., p_k$ e $q_1, ..., q_k$.

In questo caso sappiamo che un metodo di codifica ottimale per la prima sorgente permetterà, asintoticamente, di codificare il simbolo i con $-\log_2 p_i$ bit.

Se allora utilizzassimo questa codifica per codificare una sequenza emessa dalla seconda il numero di bit per carattere (cross entropy) che utilizzeremmo sarebbe

$$-\sum_{i=1}^{k} q_i \log_2 p_i \tag{3.4}$$

Il codice ottimale per la seconda sorgente avrebbe invece utilizzato

$$-\sum_{i=1}^{k} q_k \log_2 q_k \tag{3.5}$$

fare qualche richiesta in più.

bit per carattere pari all'entropia di questa sorgente.[1]

Il numero di bit sprecati per carattere, l'entropia relativa, è dato dalla differenza tra la cross entropy (3.4) e l'entropia (3.5):

$$-\sum_{i=1}^{k} q_k \log_2 \frac{p_k}{q_k} \tag{3.6}$$

3.4.3 Distanze tra sequenze

Un algoritmo, basato su idee simili, che permette di calcolare una "distanza" tra sequenze è il seguente. Prendete un testo A gli attaccate un testo B comprimete il testo $A + B$ così ottenuto e ne misurate la lunghezza L_{A+B}. Dopodichè sottraete a questa grandezza la lunghezza del file A compresso L_A.

In questo modo ottenete una misura della "distanza" tra i due testi che potete in prima approssimazione pensare come una approssimazione della cross entropy tra i due testi.

Algoritmi come questi possono essere usati per classificare dei testi od anche delle sequenze genomiche.

4 Catene di Markov

Abbiamo visto finora sequenze di caratteri indipendenti (sequenze di Bernoulli).

Queste rappresentano una classe molto ristretta e non molto frequente nelle applicazioni. In realtà le sequenze che si incontrano in pratica non hanno una statistica così semplice. Per esempio, in un testo italiano ci sono enormi correlazioni tra una lettera e la successiva: per esempio ad una "a" non segue mai un altra, ad una "q" segue sempre una "u", etc.., ma anche correlazioni a più lettere. Per esempio se una frase inizia con "ieri il cielo era nuv" voi potete facilmente inferire che la frase finirà con "nuvoloso".

[1] Notare che per la diseguaglianza di Jensen il minimo di (3.4) al variare delle p (con il vincolo $\sum p_i = 1$) si ottiene proprio per $p_k = q_k$. Se non fosse così la codifica ottimale per la sorgente di probabilità $q_1, ..., q_k$ non sarebbe quella data sopra.

4.1 Definizione di Catena di Markov - Probabilità di Transizione

In una catena di Markov la probabilità di un carattere dipende solo dal carattere precedente.[1]

Le catene di Markov e più in generale i processi di memoria finita non sono la classe più generale di processi che si possono trattare.

Una classe naturale è quella di sorgente ergodica che però non tratteremo qui rimandando a testi più specifici.

Diremo solo che per una sequenza estratta da una sorgente ergodica le probabilità dei singoli caratteri o delle parole (sequenze di un numero finito di caratteri) sono ben definite e non dipendono dal punto in cui guardate la sequenza, e che inoltre anche se non sono di memoria finita hanno una memoria che "in media" decade con la distanza.

Consideriamo una catena di Markov ad un passo con due stati possibili 0 ed 1. Le catena di Markov è completamente definita da una matrice 2 per 2

$$\begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} \tag{4.1}$$

che da le probabilità che ad un carattere ne segua un altro: cioè p_{ab} è la probabilità che al carattere a segua il carattere b.

Ovviamente $p_{00} + p_{01} = 1$ e $p_{10} + p_{11} = 1$.

In generale se abbiamo n caratteri possibili $\sigma_1, ..., \sigma_n$ le proprietà statistiche della catena di Markov saranno completamente determinate da una matrice $n \times n$:

$$A = \begin{pmatrix} p_{11}, & \cdots & , p_{1n} \\ p_{21}, & \cdots & , p_{2n} \\ & & \\ & & \\ p_{n1}, & \cdots & , p_{nn} \end{pmatrix} \tag{4.2}$$

Questa matrice si chiama generalmente matrice delle probabilità di transizione o matrice di transizione.

In particolare la probabilità della sequenza $\sigma_1 ... \sigma_k$, sarà $\mu_{\sigma_1} p_{\sigma_1,\sigma_2} ... p_{\sigma_{k-1}\sigma_k}$, dove p_σ è la probabilità di occorrenza di un carattere.

Notare che una volta che è stata data la matrice di transizione anche le probabilità di occorrenza di un carattere sono date.

[1] Una classe di processi che si possono ricondurre mediante un opportuno cambiamento di variabili a delle catene di Markov sono quelli in cui la probabilità di un carattere dipende da un numero finito di caratteri precedenti (memoria finita). In alcuni testi anche anche questi vengono chiamati processi di Markov.

Si può infatti dimostrare, sotto ipotesi molto generali, per esempio nel caso in cui tutti gli elementi della matrice di transizione sono positivi, che μ_σ è l'unico autovettore di autovalore 1 della trasposta della matrice di transizione.

Che debba essere un autovettore è evidente. Infatti la probabilità μ_σ, di avere il carattere σ, sarà eguale alla somma su tutti i caratteri τ di p_τ per la probabilità che a τ segua σ. Questa eguaglianza scritta in forma matriciale è proprio

$$\mu = A^T \mu. \tag{4.3}$$

dove A è la matrice di transizione e dove μ è il vettore $\mu = (\mu_{\sigma_1}, ..., \mu_{\sigma_n})$.

Il fatto che esista un solo autovettore di autovalore 1 della matrice di transizione è insieme ad altri importanti risultati conseguenza del, non banale, Teorema di Perron Frobenius che qui non discuteremo.

4.2 L'entropia di sequenze generiche

Finora abbiamo visto solo l'entropia di sequenze di caratteri indipendenti. In generale l'entropia si può definire per qualunque sequenza ergodica. Qui definiremo l'entropia senza però darne una giustificazione matematica.

Cercheremo però di convincere il lettore che la definizione è corretta.

Diciamo, molto rozzamente, che l'entropia è

la quantità di informazione media che ci serve per codificare un carattere una volta che sono noti i caratteri precedenti

Partiamo da una sequenza di caratteri indipendenti. L'entropia è la somma su tutti i caratteri di $-\log_2(p)$ moltiplicata per p la probabilità del carattere.

Prendiamo adesso una sequenza generica (non di caratteri indipendenti). In questo caso la statistica di un carattere dipende dai caratteri precedenti, Per esempio, se in italiano avessimo trovato la lettera q l'incertezza sul carattere che segue è 0. Infatti siamo sicuri che a q seguirà u. Se invece il carattere precedente fosse stato una a allora per dire quale era il carattere successivo avremmo avuto bisogno di una certa quantità d informazione. Infatti ad una a possono seguire con varie probabilità una b, una c ....

In generale se invece di un carattere avessimo conosciuto i due caratteri precedenti al carattere da codificare avremmo avuto una incertezza minore, e così via.

Questo discorso ci porta naturalmente a definire le seguenti quantità:

$$s_n = \sum_{\sigma_1,...,\sigma_{n-1}} p_{\sigma_1}...p_{\sigma_{n-1}} \left[-\sum_{\sigma_n} P(\sigma_n|\sigma_1,...,\sigma_{n-1}) \log_2(P(\sigma_n|\sigma_1,...,\sigma_{n-1})) \right]. \tag{4.4}$$

Questa grandezza è la media su tutte le stringhe di $n-1$ caratteri dell'entropia del carattere σ_n una volta conosciuti gli altri.

Chiaramente se la nostra sequenza è una sequenza di caratteri indipendenti allora l'entropia così definita sarà eguale alla s_1 per ogni n. Infatti se i caratteri sono indipendenti non guadagniamo nessuna informazione su di un carattere dal fatto di conoscere i precedenti.

Però in generale non è così.

In generale la s_n decrescerà all'aumentare di n raggiungendo un limite, quando n tende all'∞, che è detto entropia di Shannon e che indichiamo con S. [2]

4.2.1 L'entropia di una Catena di Markov

Nel caso di una sequenza di Markov ad una passo, già al secondo passo troviamo $s_2 = S$.

Infatti per una catena di Markov ad un passo la conoscenza del carattere precedente identifica completamente la statistica del carattere che segue e non aggiugiamo nulla dando i caratteri precedenti.

Quindi

$$S = s_2 = \sum_{\sigma_1} \mu_{\sigma_1} \sum_{\sigma_2} P_{\sigma_2|\sigma_1} \log_2 P_{\sigma_2|\sigma_1}.$$

4.2.2 L'entropia dell'inglese

I discorsi fatti finora possono sembrare molto astratti.

Per capire quanto siano, invece, concreti possiamo considerare il caso della lingua inglese.

In questo caso troviamo

Con S_0 abbiamo indicato l'entropia dei 27 caratteri considerati come equiprobabili: cioè $S_0 = \log_2 27$; con S_{parola} abbiamo indicato l'entropia, per carattere, dell'inglese codificato parola per parole; con S_∞ quella che

[2] Il fatto che la successione s_n sia decrescente, o meglio non crescente, è ovvio, almeno a chiacchiere. Si può dimostrare utilizzando la disuguaglianza di Jensen. Questo fatto unitamente al fatto che s_n è limitata dal basso (si può infatti dimostrare facilmente che $s_n \geq 0$,) implica che la successione s_n ha un limite per $n \to \infty$.

S_0	S_1	S_2	S_3	S_{parola}	S_∞
4.71	4.03	3.32	3.1	2.14	$\simeq 1$

Tabella 1: Entropia dell'inglese

si suppone essere la vera entropia dell'inglese in seguito all'esperimento di Shannon.

Potete quindi vedere che la conoscenza delle S_n vi da un idea della struttura statistica della sorgente che state considerando.

In particolare una lingua ha una memoria molto lunga. In effetti anche una parola letta una pagina prima (quindi circa 1500 caratteri prima) in un testo può influenzare la statistica di quanto stiamo leggendo.

Per le sequenze biologiche la situazione è forse anche più complicata.

Infatti non è neanche chiaro che per una sequenza di DNA si possa parlare di ergodicità.

In ogni caso una sequenza di DNA non è una sequenza di caratteri indipendenti e neanche una catena di Markov.

Però si può sempre pensare di approssimarla con catene di Markov a k passi.

Questo approccio è particolarmente utile nello studio di tali sequenze perchè permette di caratterizzare diverse porzioni della sequenza da un punto di vista funzionale ed anche di capire se sequenze diverse sono o no simili tra loro.

4.2.3 Un altra defiinzione di entropia

Un altra definizione di entropia, equivalente per sequenze ergodiche a quella data precedentemente, è la seguente. Si considerano le parole di n caratteri: $n = 1, 2, ...$, e si codifica il testo a blocchi di n lettere. L'entropia del blocco di n lettere sarà data (per quanto visto in precedenza) da

$$H_n = -\sum_{\sigma_1,...\sigma_n} p_{\sigma_1,...,\sigma_n} \log_2 p_{\sigma_1...\sigma_n}$$

quindi l'entropia per carattere sarà data da

$$S_n = \frac{H_n}{n} = -\frac{1}{n} \sum_{\sigma_1,...\sigma_n} p_{\sigma_1,...,\sigma_n} \log_2 p_{\sigma_1...\sigma_n}$$

Si può verificare facilmente che $H_1 = s_1$, e che per $n \geq 1$, valgono le seguenti relazioni:

$$s_n = H_n - H_{n-1}, \tag{4.5}$$

dove le s_n sono le entropie condizionate definite nella (4.4).

Inoltre si può dimostrare che

$$\lim_{n\to\infty} S_n = \lim_{n\to\infty} s_n \equiv S. \tag{4.6}$$

[1]

5 Segmentazione di Sequenze

Segmentare una sequenza di DNA, per esempio, vuol dire dividerla in parti con diversa funzionalità.

Per esempio vogliamo dividere la sequenza in parti codificanti e non.

Questo problema si può affrontare sfruttando il fatto che la statistica dei nucleotidi nella parte codificante ed in quella non codificante sono diverse e quindi distinguibili con appropriati procedimenti statistici.

Un altro esempio di segmentazione di una sequenza è quello del riconoscimento automatico del parlato. Abbiamo una sequenza sonora (la registrazione di una persona che parla per esempio) e vogliamo segmentare le diverse prozioni di questo segnale nelle vocali, consonanti e silenzi che lo compongono.

Esistono vari metodi per caratterizzare la statistica di una sequenza e o di una sua porzione. Iniziamo con un esempio molto semplice.

5.1 Formula di Bayes ed Inferenza Statistica

Ci sono due monete: una onesta ed una truccata che da' sempre croce. Una persona prende una delle due monete con probabilità 1/2 ed 1/2, dopodichè tira la moneta 5 volte. In 5 tiri di moneta è uscito sempre testa, Volete inferire da questa misura (il risultato dei cinque tiri di moneta) la probabilità che la moneta scelta sia quella onesta o quella truccata.

Chiamiamo A_1 l'evento "moneta truccata", A_2 l'evento "moneta onesta".

Questo problema si può risolvere mediante la formula di Bayes.

La formula fondamentale della probabilità condizionata ci dice che

$$P(A,B) = P(A)P(B|A) = P(B)P(A|B), \tag{5.1}$$

[1] Come esercizio può essere istruttivo dimostrare che $S_2 \leq S_1$ notando che $S_2 = S_1$ se e solo se le probabilità delle sequenze di due caratteri sono il prodotto di quelle ad un carattere.

dove A e B sono due eventi. Da questa possiamo ottenere la formula di Bayes:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}, \tag{5.2}$$

o, scritta per più eventi A_k,

$$P(A_j|B) = \frac{P(A_j, B)}{P(B)} = \frac{P(A_j)P(B|A_j)}{\sum_k P(A_k)P(B|A_k)}. \tag{5.3}$$

Nel nostro caso $P(B|A_1) = 1$ e $P(B|A_2) = 1/32$, mentre $P(A_1) = 1/2$, e $P(A_2) = 1/2$. Dalla formula di Bayes

$$P(A_1|B) = \frac{P(A_1)P(B|A_1)}{P(A_1) + P(A_2)/32}, \tag{5.4}$$

e

$$P(A_2|B) = \frac{P(A_2)P(B|A_2)}{P(A_1) + P(A_2)/32}. \tag{5.5}$$

Sostituendo i valori dati sopra troviamo quindi

$$\begin{aligned} P(A_1|B) &= \frac{32}{33}, \\ P(A_2|B) &= \frac{1}{33}. \end{aligned} \tag{5.6}$$

Notiamo che la formula di Bayes va applicata con molta attenzione.

Essa si applica all'evento idealizzato qui descritto. Cioè un signore prende con probabilità uguali o una moneta normale o una moneta truccata quindi la lancia cinque volte ..... .

Se però non abbiamo informazioni sulle probabilità a priori, cioè sulle $P(A_1)$ e $P(A_2)$, la risposta che possiamo avere da un tale test potrebbe essere solo indicativa.

Non solo perchè in generale non ha senso dire quali sono le probabilità a priori date alle diverse monete ma anche perchè lo spettro delle possibilità potrebbe essere sbagliato. Per esempio potrebbero esserci tre monete.

Quindi non si può realmente rispondere alla domanda: tirando una moneta esce 5 volte croce, qualè la probabilità che la moneta dia sempre croce. Non sappiamo infatti le probabilità a priori degli eventi A_1, A_2.

Diciamo qui, ma vale in generale, che l'applicazione del calcolo delle probabilità ai problemi concreti, va fatta con molta attenzione; e che spesso è necessario testare sperimentalemte la bontà di un metodo basato su di esso.

Nell'esempio precedente abbiamo cercato di determinare se una data sequenza era stata emessa o meno da una certa sorgente di eventi indipendenti (Bernoulli).

Sempre nell'ambito di sequenze Bernoulli a tratti possiamo porre il seguente problema. Consideriamo una sequenza di 20 zeri ed uni. Sappiamo, a priori, che la sequenza è stata estratta nel seguente modo. Per k volte sono stati estratti 0 ed 1 con probabilità $1/2$ ed $1/2$, le restanti $20 - k$ volte sono stati estratti 0 ed 1 con probabilità .7 e .3 rispettivamente. Il numero k può essere qualunque numero tra 0 e 20. Guardando la sequenza vogliamo determinare la probabilità di k, o meglio, la probabilità che k abbia un certo valore. Di nuovo questo problema può essere affrontato con la formula di Bayes.

Quindi, come nel caso precedente, ci saranno delle ambiguità legate alla scelta delle probabilità a priori. Comunque se suppononiamo di conoscere le probabilità a priori degli eventi

- la moneta è sempre stata truccata (k=0)
- la moneta è stata cambiata al tempo k
- la moneta è sempre stata buona (k=20)

possiamo, guardando la sequenze, utilizzare la formula di Bayes per inferire il punto in cui la moneta è stata cambiata.

Per semplicità consideriamo il caso in cui le probabilità dei 21 eventi descritti sopra sono tutte eguali:

$$p(0) = p(1) = ... = p(20) = \frac{1}{21}. \tag{5.7}$$

Supponiamo inoltre che la sequenza sia la seguente:

00011101001000100100

In questo caso possiamo calcolare la $p(k)$. Il risultato è ? (esercizio non banale ma molto istruttivo)

5.2 Hidden Markov Model

Un strumento molto potente appositamente concepito per risolvere questo tipo di problemi è quello degli Hidden Markov Models (Modelli di Markov nascosti) spesso chiamati anche HMM.

Un modello di Markov nascosto è una catena di Markov i cui stati non sono osservabili direttamente. Più precisamente:

- la catena ha un certo numero di stati
- gli stati evolvono secondo una catena di Markov

- ogni stato genera un evento con una certa distribuzione di probabilità che dipende solo dallo stato
- l'evento è ossevabile ma lo stato no

Il problema è inferire gli stati o altre caratteristiche del sistema

Un esempio concreto è quello che abbiamo considerato nel paragrafo precedente.

In fatti il problema moneta onesta - moneta truccata si può decrivere come un HMM.

Gli stati saranno due: moneta onesta e moneta truccata.

Gli eventi saranno i risultati del tiro della moneta. Possiamo infatti notare che, una volta noto lo stato, le probabilità dei due eventi (0 od 1) sono determinati completamente.

La catena di Markov tra gli stati è definita dalla matrice delle probabilità di passaggio da uno stato ad un altro.

Visto che vogliamo modellare una situazione in cui la moneta viene cambiata una volta possiamo pensare che questa probabilità sia $\epsilon = \frac{1}{numero lanci}$.

Notare che il modello che stiamo considerando non è esattamtne quello descritto nel paragrafo precedente. Qui stiamo supponendo che la moneta possa cambiare quante volte vuole ma che la probabilità di cambiare sia ϵ. Sopra invece consideravamo il caso in cui la moneta cambiava una volta sola (o nessuna).

A costo di ripetermi vorrei notare ancora che una volta che un modello è stato definito precisamente si possono calcolare matematicamente (almeno in linea di principio) tutte le grandezze cui siamo interesati.

La scelta del modello invece non è un problema matematico: nello scegliere il modello si fanno delle ipotesi che vanno gistificamente biologicamente (o fisicamente o ...).

È molto importante distinguere queste due fasi dello studio.

5.2.1 Algoritmo di Viterbi

Nell'ambito degli HMM un problema molto naturale è quello di trovare la sequenza di stati più probabile. Per esempio, nel problema descritto sopra, trovare, una volta che la sequenza di 0 ed 1 è data, la sequenza di stati (moneta onesta/moneta corretta) più probabile.

Questo non è l'unico problema importante. Ci si potrebbe porre molti altri problemi. Per esempio qualè la probabilità, una volta che la sequenza di eventi è conosciuta) che la moneta venga cambiata k volte (per esempio 3 volte).

Il problema che studiamo è particolarmente importante e si risolve mediante l'algoritmo di Viterbi. Questo tipo di algoritmi come quelli di allineamento che vedremo (che in certi casi sono identici) sono spesso detti *programmazione dinamica.*

Prima di tutto serve un po' di notazione.

Chiamiamo N la lunghezza della sequenza che vogliamo studiare, $s_1, ..., s_N$, gli stati, $e_1, ..., e_N$, gli eventi.

Per definizione di probabilità condizionata sappiamo che:

$$p(s_1, ..., s_N | e_1, ..., e_N) = \frac{p(s_1, ..., s_N, e_1, ..., e_N)}{p(e_1, ..., e_N)} \propto p(s_1, ..., s_N, e_1, ..., e_N).$$

Quindi possiamo cercare direttamente il massimo di $p(s_1, ..., s_N, e_1, ...e_N)$.

Possiamo poi notare che

$$p(s_1, ..., s_N) = p(s_1) \times p(s_2|s_1) \times ... \times p(s_N|s_{N-1}),$$

e che

$$p(e_1, ..., e_N) = p(e_1|s_1) \times p(e_2|s_2) \times ... \times p(e_N|s_N).$$

Dobbiamo quindi massimizzare, sulle possibili scelte di $s_1, ..., s_N$,

$$p(s_1) \times p(e_1|s_1) \times p(s_2|s_1) \times p(e_2|s_2) \times ... \times \times p(s_N|s_{N-1}) \times p(e_N|s_N).$$

La grandezza da massimizzare, prendendo i logaritmi, diventa una somma di termini.

Definiamo per un generico valore di k,

$$\gamma_k(s_k) = \max_{s_1,...,s_{k-1}} p(s_1) \times p(e_1|s_1) \times p(s_2|s_1) \times p(e_2|s_2) \times p(s_k|s_{k-1}) \times p(e_k|s_k).$$

Allora vale la seguente formula di ricorsione

$$\gamma_k(s_k) = p(e_k|s_k) \max_{s_{k-1}} p(s_k|s_{k-1}) \gamma_{k-1}(S_{k-1}).$$

possiamo quindi trovare la sequenza ottimale (o le sequenze ottimali) con il seguente algoritmo:

Inizializzazione Definiamo

$$\gamma(s_1) = p(e_1|s_1) p(s_1)$$

Iterazione

- $\gamma(s_k) = p(e_k|s_k) \max_{s_{k-1}} p(s_k|s_{k-1})$

- $puntatore(s_{k-1}) = argmax_{s_{k-1}} P(s_k|s_{k-1})\gamma_{k-1}(s_{k-1})$

Fine $s_N = argmax(\gamma(s_N))$

Traceback $S_k = puntatore(s_{k+1})$

L'algoritmo funziona perchè la definizione dei puntatori viene fatta in generale: cioè per ogni possibile valore di s_{k-1} si calcola il valore di s_k migliore. A questo punto basta tornare indietro per trovare la sequenza ottimale.

Note.

- L'algoritmo di Viterbi trova la sequenza più probabile. Non sempre quest'informazione è quella che ci serve. Per esempio potremmo essere interessati a sapere qual'è la probabilità che lo stato i abbia un certo valore o ad altre domande cui l'algoritmo di Viterbi non può rispondere.

- Uno dei problemi importanti degli HMM è l'ottimizzazione dei suoi parametri.

  Come abbiamo già detto più volte la scelta di un modello statistico è una operazione arbitraria che va motivata di volta in volta e ciò vale anche per gli HMM.

  Una volta che si è scelto un modello rimane ancora da determinare i parametri del modello.

  Cioè le probabilità di transizione da stato a stato e le probabilità degli eventi una volta noti gli stati.

  Questo problema è molto generale e si può affrontare in molti modi diversi. Un approccio molto naturale al problema è dato dall' algoritmo di Baum e Welch per il quale rimandiamo a BSA.

6 Confronto di sequenze: distanze - allineamento

Finora ci siamo occupati delle proprietà statistiche di una data sequenza.

Il fine di tutto quanto stiamo vedendo è però sopratutto la classificazione di sequenze. Per esempio data una sequenza codificante una proteina ci piacerebbe poter dire a quale famiglia di proteina questa appartiene, o data una sequenza di DNA se questa è codificante o no.

Per fare ciò le proprietà statistiche di una sequenza non sempre sono sufficienti.

Per esempio sapere le frequenze dei vari nucleotidi generalmente non discrimina tra diverse proprietà funzionali delle sequenze in esame.

Risulta quindi importante avere dei metodi che permettano di confrontare due o più sequenze.

Per classificare delle sequenze o per riconoscere sequenze simili può essere molto utile poter definire una distanza tra due diverse sequenze.

La distanza dovrà essere definita in base a ragioni biologiche ed una volta definita dovrà essere possibile calcolarla (in un tempo ragionevole) mediante opportuni algoritmi.

La distanza più semplice che si può definire tra diverse sequenze è la distanza di Hamming. Purtroppo essa, come vedremo, è insoddisfacente da un punto di vista biologico.

La distanza di Hamming tra due sequenze di lunghezza uguale è il numero di basi diverse. Cioè la distanza tra

$$\begin{matrix} ACAGATGTACAT \\ ACTGAGTACGTT \end{matrix} \tag{6.1}$$

è 7. Infatti ci sono 7 posizioni in cui il nucleotide nelle due sequenze è diverso.

6.1 Edit distance

La distanza di Hamming purtroppo non è utile in biologia per varie ragioni. La prima, é che tipicamente in biologia si è interessati a definire una distanza tra sequenze di lunghezza diversa. La seconda, più importante, è che ci si aspetta che i processi evolutivi possano variare una sequenza inserendo o eliminando piccole (in alcuni casi grandi) porzioni della stessa, o mediante errori casuali. Possiamo schematizzare queste tre operazioni nel modo seguente:

- inserimento di un carattere
- delezione di un carattere
- sostituzione di un carattere

Per esempio le sequenze

$$\begin{matrix} ACGTATGTGATGA \\ AACGTATGTGATG \end{matrix} \tag{6.2}$$

hanno una distanza di Hamming pari a 12 ma in realtà potete passare dall'una all'altra con solo due operazioni.

Si definisce quindi una "edit" distance che è data dal minimo numero di operazioni che portano da una sequenza all'altra.

Questa distanza può essere calcolata con un algoritmo molto efficiente e molto bello, l'algoritmo di Needleman e Wunsch, che andiamo a decrivere. In realtà nel seguito descriveremo l'applicazione di questo algoritmo ad un problema diverso anche se molto simile.

6.2 Allineamento: Algoritmo di Needleman e Wunsch

Il problema di calcolare la edit distance è equivalente ad un problema di allineamento.

Allineare due sequenze vuol dire mettere in corrispondenza gli elementi di una sequenza con quelli dell'altra (mantenendo l'ordine degli elementi) aggiungendo eventualmente degli spazi vuoti nell'una e/o nell'altra. Ad ogni accoppiamento di due simboli è dato un punteggio (che ha motivazioni biologiche) e ad ogni spazio vuoto (*gap*) è dato un valore.

Un esempio semplice potrebbe essere:

- due simboli accoppiati uguali valgono 1
- due simboli accoppiati diversi valgono a
- un vuoto vale b

dove a e b sono due numeri negativi.

Nel caso in cui $a = b = 0$ questo problema si riduce a trovare la più lunga sottosequenza comune ordinata tra due sequenze.

Il fatto che l'allineamento di due sequenze non sia un problema troppo difficile si può capire nel seguente modo.

Sia $D_{i,j}$ il valore dell'allineamento dei primi i simboli della prima sequenza con i primi j simboli della seconda sequenza. Allora $D_{i,j}$ sarà il massimo tra

- $W_{i,j} + D_{i-1,j-1}$ che si ottiene allineando a_i con b_j,
- $\max(D_{i-1,j}, D_{i,j-1})$ se non allineiamo a_i con b_j.

Queste relazioni permettono di definire una procedura iterativa per definire $D_{i,j}$ a partire da $D_{0,0} = 0$, $D_{0,k} = 0$ per ogni $k = 1, 2, ..., m$ e $D_{k,0} = 0$ per ogni $k = 1, 2, ..., n$.

Per capire il procedimento conviene pensare D come una matrice $m+1 \times n+1$.

Quando abbiamo trovato $D_{m,n}$ abbiamo trovato il valore del massimo allineamento possibile. A questo punto dobbiamo trovare l'allineamento.

Esso si può ottenere mediante un procedimento all'indietro. Infatti partiamo dalla posizione $m+1, n+1$ e torniamo indietro fino al punto $0, 0$ muovendoci in modo da passare da una casella a quella dalla quale il suo valore è stato determinato.

Consideriamo un esempio. Vogliamo trovare la sottosequenza ordinata comune più lunga tra due sequenze (il caso descritto prima quando $a = b = 0$.) Le sequenze sono $\alpha = AGCTTA$, e $\beta = AGGTGTA$.

La prima sequenza ha $m = 6$ nucleotidi e la seconda ha $n = 7$ nucleotidi.

Costruiamo prima di tutto la matrice $W_{i,j}$:

$$\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

A questo punto possiamo iniziare a costruire la matrice D.

È conveniente riprodure la matrice W orlata con le prime riga e colonna di 0.

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

A questo punto possiamo iterare la procedura: l'elemento $1, 1$ della matrice sarà il massimo tra $W_{1,1} + D_{0,0} = 1$, $D_{0,1} = 0$, e $D_{1,0} = 0$, quindi sarà 1. Dopo che abbiamo calcolato la prima linea la matrice apparirà così:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Iterando la procedura troviamo

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1
0	1	2	2	2	2	2	2
0	1	2	2	2	2	2	2
0	1	2	3	3	3	3	3
0	1	2	2	3	3	4	4
0	1	2	2	3	3	4	5

quindi sappiamo che il valore dell'allineamento migliore è 5.

Per trovare l'allineamento migliore basta partire dal punto di valore 5 andando da ogni casella a quella che ha determinato il suo valore fino a raggiungere la prima riga e/o la prima colonna.

Uno dei percorsi migliori (non ne esiste infatti in generale uno solo) è il seguente.

	A	G	G	T	G	T	A
A	$\nwarrow$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
G	$\bullet$	$\nwarrow$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
C	$\bullet$	$\bullet$	$\nwarrow$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
T	$\bullet$	$\bullet$	$\bullet$	$\nwarrow$	$\leftarrow$	$\bullet$	$\bullet$
T	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\nwarrow$	$\bullet$
A	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\nwarrow$

Esso corrisponde al seguente allineamento

A	G	C	T	$-$	T	A
A	G	G	T	G	T	A

Notare che questo procedimento si generalizza ovviamente ad altre funzioni obbiettivo. Per esempio si può definire una matrice $W_{i,j}$ che tenga conto della similarità degli aminoacidi.

L'algoritmo per calcolare la edit distance è solo leggermente più complicato di quello qui descritto.

Qui non lo analizzeremo in dettaglio.

Sulla rete ci sono molti siti con delle "applet" che allineano sequenze e calcolano la "edit distance". Segnalo qui qualcuno di questi siti. Per la edit distance si può vedere, per esempio,
http://www.csse.monash.edu.au/ lloyd/tildeAlgDS/Dynamic/Edit/

Per l'allineamento di proteine, con la possibilit'a di variare la matrice di punteggio, si può vedere
http://www.dkfz-heidelberg.de/tbi/bioinfo/PracticalSection/AliApplet/

Finiamo questo capitolo notando che mentre l'allineamento di due sequenze può essere ottenuto molto rapidamente mediante l'algoritmo di Needleman e Wunsh, ciò non è vero per l'allineamento di 3 o più sequenze. Per allineare tre o più sequenze non sono noti algoritmi che funzionano in tempo polinomiale. Tipicamente, quindi, quando si devono allineare più di due sequenze si usano degli algoritmi empirici. Per esempio se ne allineano due poi se ne aggiunge una e così via. Altrimenti si utilizzano degli algoritmi di tipo Montecarlo. Uno dei metodi più usati per allineare più sequenze si basa sugli HMM.

Per maggiori dettagli si può vedere BSA che dedica una sezione a quest'argomento.

7 Alberi filogenetici

La filogenesi si occupa di descrivere l'evoluzione di un insieme di specie.

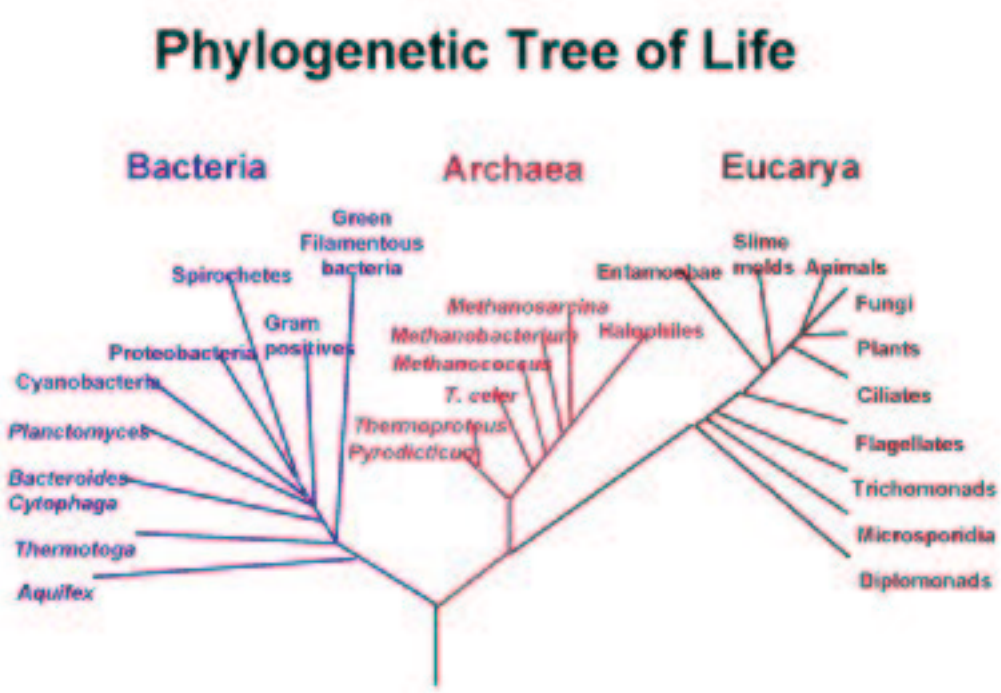
La decrizione viene fatta mediante un albero. Quest'albero (per semplicità iniziamo dagli alberi con radice) ha una radice che può essere la specie primitiva. I punti di diramazione rappresentano la separazione di una specie in due specie. Credo che in questo caso una figura valga più di molte parole. L'albero che segue è tratto da Wikipedia:

$$http://en.wikipedia.org/wiki/Phylogenetic\_tree$$

,

$$http://en.wikipedia.org/wiki/Carl\_Woese$$

.



Un altro tipo di alberi sono gli alberi senza radice (unrooted tree). In questo caso si rinuncia a dire dov'è la radice e si cerca di costruire la struttura che meglio si accorda con i dati.

Ovviamente alberi di questo tipo possono essere utilizzati per molti altri scopi come la tassonomia che ha applicazioni anche al di fuori della biologia: per esempio si può studiare la tassonomia di un insieme di documenti o altro.

Questi alberi vengono tipicamente costruiti a partire da una matrice di distanze secondo alcuni criteri che sono dettati dalla specifico probelma che si vuole risolvere.

Da un punto di vista matematico, come vedremo, questi alberi sono un importante metodo per rappresentare l'informazione che è contenuta in una matrice di distanze.

Vediamo qui alcuni degli algoritmi che si usano per costruire alberi filogenetici.

Ognuno di questi metodi ha una giustificazione basata sul tipo di dato che

si vuole rappresentare che quando possibile cercheremo di descrivere almeno parzialmente.

7.0.1 UPGMA

Considerino prima di tutto il metodo più semplice: UPGMA : Unweighted Pair Group Method with Arithmatic Mean.

Supponiamo di avere n punti (sequenze per esempio) e chiamiamo d_{ij} la distanza tra i e j.

L'algoritmo UPGMA funziona raggruppando i punti in clusters (grappoli). Un cluster contiene un certo numero di punti e la distanza tra due clusters si definisce come la media delle distanze dei punti dei due clusters.

Cioè per due clusters A e B la distanza è data da

$$D_{AB} = \frac{1}{|A||B|} \sum_{i \in A, j \in B} d_{ij}.$$

Vediamo l'algoritmo:

Inizializzazione
All'inizio ci sono n clusters $C_1, ..., C_n$ ognuno costituito da un punto

Iterazione

- si trovano i due clusters più vicini (che chiamiamo A e B);
- si raggruppano i due clusters A e B in un nuovo cluster contenente i due clusters precedenti;
- si calcolano le distanze tra il nuovo clusters e gli altri clusters;
- si ripete la procedura con un cluster in meno: abbiamo infatti due clusters sono stati raggruppati in uno solo;

Fine
Il procedimento termina quando si rimane con un solo cluster.

Questo algoritmo è basato sull'ipotesi che il tasso di evoluzione sia costante. infatti la distanza dei due clusters "figli" dal clusters "padre", è uguale.

Si può dimostrare che se le distanze sono additive sull'albero e soddisfano alla proprietà ultrametrica allora UPGMA ricostruisce l'albero giusto.

In generale la disuguaglianza triangolare assicura che dati 3 punti A, B e C sia soddisfatta

$$d_{AC} \leq d_{AB} + d_{BC}.$$

La proprietà di ultrametricità è invece molto più forte:

$$d_{AC} = \max(d_{AB}, d_{BC}).$$

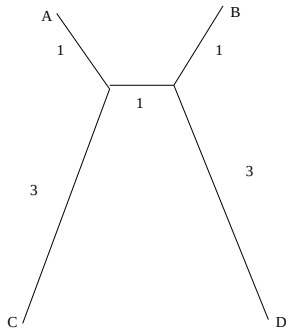
In generale si crede che questa ipotesi non sia soddisfatta, ciò malgrado l'algoritmo è molto rapido (il numero di operazioni cresce come n^3) e molto spesso algoritmi basati su idee diverse danno risultati molto simili o addirittura uguali. Quando questa ipotesi è molto lontana dall'essere verificata l'algoritmo può sbagliare molto.

7.0.2 Neighbour Joining

L'algoritmo NJ è un algoritmo molto bello che ricostruisce l'albero corretto se le distanze soddisfano l'ipotesi di additività. Quindi in un caso più generale rispetto ad UPGMA che richiedeva anche l'ipotesi di ultrametricità.

Quest' algoritmo è molto simile al precedente. Anche in questo caso si uniscono due punti alla volta, la differenza sta nel criterio con cui si scelgono i punti da unire e nella definizione delle distanze.

La prima cosa che possiamo notare è che non sempre i due punti più vicini sono uniti. Nel caso esemplificato in figura infatti la distanza tra A e B è 3 mentre la distanza tra A e C (che sono collegati allo stesso nodo) è 4.



Serve quindi una procedura più raffinata. L'idea è (rozzamente) di sottrarre alle distanze la distanza media dagli altri punti.

Più precisamente, date le distanze d_{ij} si definiscono le grandezze D_{ij} nel seguente modo:

$$D_{ij} = d_{ij} - (r_i + r_j)$$

dove

$$r_i = \frac{1}{L-2} \sum_k d_{i,k}$$

Si può dimostrare, vedi [BSA],non facilmente, che i due punti i e j che rendono minima la D sono sicuramente (se vale l'ipotesi di additività ovviamente) attaccati allo stesso nodo.

Una volta che sappiamo che due punti sono attaccati allo stesso nodo dobbiamo ricalcolare le distanza tra questo nodo ed i punti rimanenti.

Se i due punti sono i e j, ed il nodo è k per la proprietà di additività troviamo:

$$d_{xk} = \frac{1}{2}\left(d_{ik} + d_{jk} - d_{ij}\right).$$

Inoltre le distanze tra x ed i punti i e j sono date da $d_{ix} = \frac{1}{2}\left(d_{ij} + r_i - r_j\right)$, e $d_{jx} = \frac{1}{2}\left(d_{ij} + r_j - r_i\right)$.

Siamo quindi in grado di definire l'algoritmo.

- dato l'insieme di punti e le distanze tra loro (d_{ij}) si calcolano le grandezze D_{ij}
- si trovano i punti p e q che rendono minima la D;
- si collegano i due punti p e q ad un nuovo nodo x;
- si calcolano le distanze tra il nuovo punto x e gli altri (come descritto sopra);
- si eliminano i due punti p e q;
- si riparte con un punto in meno fino a quando non rimane un punto solo;

7.0.3 Algoritmi di parsimonia

Gli algoritmi di parsimonia cercano l'albero che minimizza il numero di sostituzioni. Sono molto usati perchè danno buoni risultati anche se in realtà non c'è nessuna evidenza che l'evoluzione sia parsimoniosa ed inoltre non ci siano degli algoritmi rapidi per trovare l'albero migliore.

Consideriamo un esempio. Le sequenze che vogliamo rappresentare nell'albero sono

$$\begin{array}{cccc} A & A & C & G \\ A & C & C & G \\ A & C & G & G \\ A & C & G & T \end{array} \tag{7.1}$$

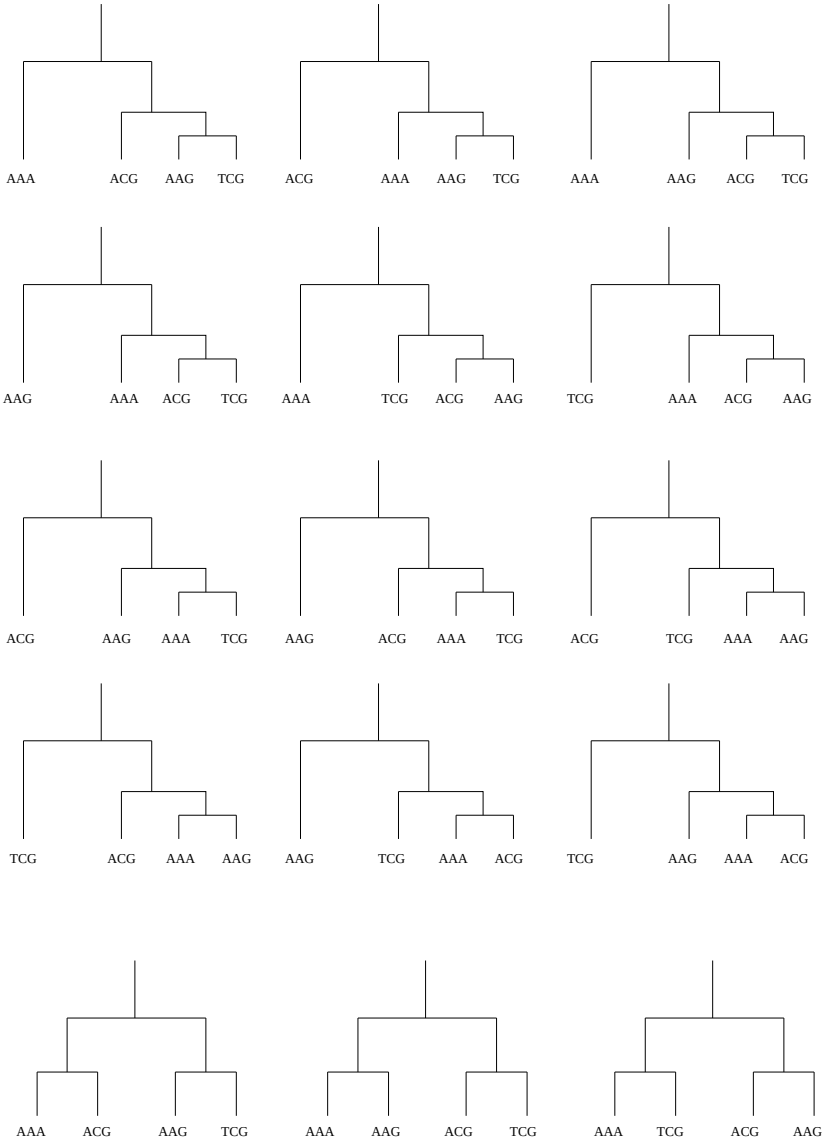
Esistono 15 alberi filogenetici etichettati con 4 foglie, quindi possiamo in questo caso trovare facilmente l'albero migliore. Notare che, anche se gli alberi non sono molti

In generale non è possibile utilizzare questo metodo esaustivo, per esempio con 20 foglie il numero di alberi etichettati è 8200794532637891559375 (!) ,

e quindi si utilizzano algoritmi euristici che cercano di dare un buon albero ma che in genrerale non saranno in grado di dare l'albero migliore.

Non discuteremo qui questi algoritmi rimandando alla bibliografia.

Analizziamo però in dettaglio un caso con 4 "specie": AAA, AAG, ACG, TCG.



Il costo dell'albero sembrerebbe dipendere anche da quali sequenze vengono messe sui nodi interni. In realtà, data la topologia e date le specie sulle foglie, il costo dell'albero è completamente determinato mediante il seguente algoritmo di Fitch.

Prima di tutto notiamo che la quantità di cambiamenti si può calcolare

sito per sito. Quindi consideriamo un sito. Il valore sui nodi è dato dall'intersezione dei valori sui figli se questa è non vuota, altrimenti dall'unione. Il costo dell'albero è il numero di volte che l'intersezione è risultata vuota. Facendo il conto si vede che il numero di mutazioni minimo è 3 e che viene realizzato dagli alberi che nella figura precedente sono il quarto, il nono, il decimo ed il quattordicesimo.

Un istruttivo esercizio consiste nella costruzione di una possibile realizzazione di un albero ottimale: cioè nel indicare quali sequenze vanno inserite nei nodi interni.

8 Alcuni utili risultati matematici

8.1 La formula di Stirling

La formula di Stirling è una formula approssimata per $n!$. Essa dice che $n! \simeq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$. Essenzialmente, cioè trascurando termini polinomiali in n, ci dice che

$$n! \simeq \left(\frac{n}{e}\right)^n. \tag{8.1}$$

Qui ci accontenteremo di mostrare che

$$\left(\frac{n}{e}\right)^n \leq n! \leq n\left(\frac{n}{e}\right)^n \tag{8.2}$$

Notiamo che

$$\log n! = \sum_{k=1}^{n} \log k. \tag{8.3}$$

Dato che il logaritmo è una funzione crescente per ogni k vale

$$\int_k^{k+1} \log x dx \geq \log k \geq \int_{k-1}^{k} \log x dx \tag{8.4}$$

Quindi sommando su k ed usando il fatto che $\log 1 = 0$, troviamo

$$\int_1^{n+1} \log x dx \geq \log k \geq \int_1^{n} \log x dx \tag{8.5}$$

La primitiva di $\log x$ è $xlogx - x$, quindi troviamo

$$(n+1)log(n+1) - (n+1) \geq n! \geq nlogn - n, \tag{8.6}$$

che era quanto volevamo dimostrare.

8.2 Il teorema di Shannon McMillan

Come sappiamo i singoli caratteri emessi da una sorgente ergodica non sono in generale equiprobabili e neanche indipendenti.

Ciò è vero, in generale, anche per stringhe di L caratteri consecutivi.

Il teorema di Shannon-McMillan ci dice però che, per una sequenza ergodica, le stringhe sufficientemente lunghe sono sostanzialmente equiprobabili e che la loro probabilità è data, circa, da

$$2^{LS}$$

dove S è l'entropia di Shannon della sorgente.

Questa formulazione è rozza. Il vero teorema è il seguente.

Teorema di Shannon McMillan. Sia $\epsilon > 0$ (cioè dato ϵ positivo comunque piccolo). Allora esiste L (dipendente da ϵ) tale che le sequenze di lunghezza L possono essere suddivise in due classi: I_1, I_2. Le sequenze nella classe I_2 hanno tutte insieme probabilità di apparire minore di ϵ mentre quelle in I_1 soddisfano:

$$S - \epsilon \leq \frac{1}{L} \log_2 [p_{\sigma_1,\ldots,\sigma_n}] \leq S + \epsilon$$

Il teorema di Shannon Mc Millan è di difficile dimostrazione in generale. Qui lo dimostreremo nel caso di sequenze Bernoulli.

Il teorema di Shannon McMillan per sequenze di caratteri indipendenti può essere visto come un applicazione della legge dei grandi numeri.

Consideriamo qui per semplicità una sequenza di L 0 e 1, estratti rispettivamente con probabilità p ed $1 - p$.

Chiamiamo k il numero di 0 della stringa ed $L - k$ il numero di 1. Fissiamo un numero ϵ positivo piccolo a piacere. Per la legge dei grandi numeri sappiamo che esiste un L_ϵ tale che quando $L \geq L_\epsilon$ allora la probabilità che la frequenza degli 0 (e degli 1) si discosti più di ϵ da p (da $1 - p$) è minore di ϵ.

Questo vuol dire, per esempio che il numero di 0, n_0, soddisfa

$$(p - \epsilon)n \leq n_0 \leq (p + \epsilon) \tag{8.7}$$

per un insieme di realizzazioni I_0, la cui probabilità è maggiore di $1 - \epsilon$.

Occupiamoci di questo insieme. Quale sarà la probabilità di una sequenza in questo insieme. Sarà

$$\mathcal{P} = p^{L_0}(1 - p)L - n_0 \simeq p^{pL}(1 - p)^{(1-p)L} \tag{8.8}$$

avendo trascurato termini di ordine ϵ.

Quindi

$$\log \mathcal{P} \simeq L\left(p \log p + (1-p)\log(1-p)\right) = LS(p), \tag{8.9}$$

dove abbiamo indicato con $S(p)$ l'entropia di una sorgente Bernoulli di probabilità p ed $1-p$. Questo è esattamente il teorema di Shannon Mc Millan[3] Ovviamente possiamo anche calcolare quante sequenze sono contenute in questo insieme. Dato che la loro probabilità è circa $2^{LS(p)}$, il loro numero sarà circa $2^{-LS(p)}$.

Questo risultato ci permette almeno rozzamente di capire perchè LZ77 sia asintoticamente ottimale.

Se abbiamo già codificato una parte della sequenza lunga L quanto sarà lunga la stringa che troveremo all'indietro?

Per il teorema di Shannon-McMillan la probabilità di una stringa di lunghezza n sarà circa 2^{-Sn}, dove S è l'entropia della sorgente. Ragionevolmente una stringa di probabilità p si troverà in una stringa di lunghezza $1/p$. A causa di ciò ci aspettiamo che la stringa più lunga che troviamo soddisfi approssimativamente $2^{Sn} = L$. Quindi $n = \frac{\log_2(L)}{h}$.

A questo punto dobbiamo codificare la distanza all'indietro alla quale avete trovato la stringa e la lunghezza della stringa. Questa sarà circa L, e quindi ci serviranno $\log_2(L)$ bit. Quindi abbiamo codificato $\log_2 L/S$ caratteri con $\log_2 L$, bit. Il rapporto di compressione è quindi S. [4]

8.3 Il Lemma di Jensen

Lemma di Jensen Sia f una funzione convessa e siano $m_1, ..., m_n$ numeri positivi a somma 1. Allora

$$f(\sum_{i=1}^{n} m_i x_i) \leq \sum_{i=1}^{n} m_i f(x_i). \tag{8.10}$$

Il significato è molto semplice: $m_1, ..., m_n$ sono dei pesi a somma 1. Quindi il teorema sta dicendo che f della media è minore o uguale della media della f.

Notare che nel caso di $n = 2$ la (8.10) coincide con la definizione di convessità per f.

[3] Attraverso la formula di Stirling potremmo stimare esplicitamente di quanto si discosta la probabilità di una sequenza nell'insieme I_0 da quella limite. Ciò richiede però un po' di lavoro che non faremo qui.

[4] Abbiamo qui trascurato la quantità di bit necessaria per scrivere la lunghezza della stringa codificata. Questo contributo si può mostrare essere proporzionale a $\log_2 \log_2(L)$. Quindi nel limite $L \to \infty$ è trascurabile.

Infatti f è convessa se e solo se

$$f(m_1x_1 + m_2x_2) \leq m_1f(x_1) + m_2f(x_2), \tag{8.11}$$

quando $m_1 \geq 0$, $m_2 \geq 0$, $m_1 + m_2 = 1$.

Se poniamo $m_1 = \lambda$, e quindi $m_2 = 1 - \lambda$ possiamo notare che

$$\lambda f(x_1) + (1 - \lambda)f(x_2) \tag{8.12}$$

è proprio la retta che va dal punto $(x_1, f(x_1))$ al punto $(x_2, f(x_2))$ e che questa retta, essendo f convessa, è sopra $f(\lambda x_1) + f(1 - \lambda)x_2)$.

Il risultato generale si dimostra facilmente per induzione. Sappiamo che è vero per $n = 2$. Supponiamo che sia vero per $n - 1$ e dimostriamo che è vero per n. Chiamiamo

$$m_1 + m_2 + ... + m_{n-1} = \lambda, \tag{8.13}$$

$$m_n = (1 - \lambda), \tag{8.14}$$

$$\frac{X = \sum_{i=1}^{n-1} m_i x_i}{\sum_{i=1}^{n-1} m_i,} \tag{8.15}$$

e

$$Y = x_n. \tag{8.16}$$

Utilizzando la convessità troviamo

$$f(\lambda X + (1 - \lambda)Y) \leq \lambda f(X) + (1 - \lambda)f(Y). \tag{8.17}$$

Cioè

$$f(\sum_{i=1}^{n} m_i x_i) \leq \lambda f(X) + m_n f(x_n). \tag{8.18}$$

Ma per l'ipotesi induttiva (il fatto che il Lemma sia vero per $n-1$) sappiamo che

$$f(X) \leq \sum_{i=1}^{n-1} m_i f(x_i), \tag{8.19}$$

il che dimostra il Lemma.

Riferimenti bibliografici

[BSA] *Biological Sequence Analysis*, Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison, Cambridge University Press.

[W] Wyner AD,*Typical Seqeunces and all that: Entropy, Pattern Matching, and Data Compression*,1994 Shannon Lecture. Si trova in rete sul sito dell'IEEE all'indirizzo

`www.ieeeits.org/publications/nltr/95_jun/shannonrev.ps`

[AT] *Bioinformatica*, Anna Tramontano, Zanichelli.

[MG] *Managing Gigabytes*, Witten IH, Moffat A, Bell TC, Morgan Kaufmann editore.

[NW] Neville-Mannig CG and Witten IH, *Protein is incompressible*, Proc. of the IEEE Data Compression Conference, 257-266, 1999.

[WJH] Weiss O, Jimenez-Montano MA, and Herzel H, *Information content of protein sequences*,Jour. Theor. Biology, 206, 379-386,2000.