Tecnologie del Linguaggio Naturale: Viterbi e Translator

Marco Scaletta

marco.scaletta@edu.unito.it

Repository viterbi: https://github.com/marcoscaletta/viterbi

Repository translator: https://github.com/marcoscaletta/translator

Sito web Viterbi: https://viterbi.herokuapp.com

Febbraio 2019

Contents

1 Introduzione

1.1 Motivazioni

Prima di descrivere il lavoro fatto ritengo importante motivare la scelta del progetto implementato. Nonostante il notevole interesse personale nei confronti degli argomenti del corso riguardanti la sintassi e soprattutto la semantica (con una particolare attenzione al *lambda calcolo*), ho preferito optare per il **traduttore direct IT-EN** per le seguenti ragioni: innanzitutto sono rimasto affascinato dall'eleganza e dalla semplicità dell'*Algoritmo di Viterbi*, utilizzato per trovare la migliore sequenza di stati, *Viterbi Path*, in una sequenza di eventi osservati. Un aspetto che ho trovato interessante, che è alla base della fortuna dell'idea introdotta con questo metodo, è senz'altro l'utilizzo delle *Hidden Markov Model (HMM)*. Inoltre, dal momento che ho già avuto modo di studiare con interesse le proprietà delle *catene di Markov*, ho apprezzato la funzione fondamentale che hanno svolto nella procedura di analisi delle frasi. In aggiunta ho preferito dedicare la mia attenzione a un progetto che avrei potuto sviluppare *from scratch*. Questo perché il metodo utilizzato nell'implementazione proposta, che ha l'obiettivo di ottenere un'unica **interpretazione morfologica** di una parola, oltre ad essere a basso livello, ignorando sintassi e semantica, è anche parziale, dal momento che non considera importanti variazioni morfologiche, quali la flessione e la derivazione. Questo perché la disambiguazione viene fatta sfruttando unicamente metodi di calcolo delle probabilità.

Un semplice esempio può essere il seguente: supponiamo di aver addestrato il *PoS Tagger* con un dataset contenente anche il termine `arzigogolato` taggato come `ADJ` (aggettivo). Se il *PoS Tagger* dovesse incontrare una frase contenente `arzigogolati`, saremmo convinti del fatto che, avendo la medesima *radice*, `arzigogolat-`, si tratti di una *flessione*, in particolare di una *declinazione*, dell'aggettivo `arzigogolato` e quindi dovrebbe essere etichettato nello stesso modo. Al contrario, trattandosi di una parola che, molto probabilmente, non è anch'essa presente nel dataset, sarà solo possibile *sperare* che si trovi in una posizione della frase che consenta una corretta interpretazione.

1.2 Presentazione del progetto

Nel presente caso di studio si implementa l'algoritmo di Viterbi per effettuare il *Part of Speech Tagging (PoS Tagging)* di una frase: quindi la sequenza di eventi osservati è lista di parole di una frase, mentre il *Viterbi Path* è la sequenza di tag assegnati a tali parole. Successivamente, grazie a un semplice dizionario, avviene la *traduzione direct* o *literal*, cioè *parola per parola*,

dall'italiano all'inglese. Oltre al dizionario sono stati utilizzati anche altri strumenti, come una piccola base di conoscenza, un riordinatore di parole e l'applicazione di regole grammaticali. Anticipo ora che, senza considerare le iniziali maiuscole delle parole, il miglior risultato ottenuto testando l'accuratezza del *PoS Tagging* è del 94.23%. Considerando, invece, anche le iniziali maiuscole si ottiene un'*accuracy* del 94.7%. Le implementazioni specifiche verranno presentate successivamente. Purtroppo è necessario precisare che questa accuratezza non è sufficiente dal momento che i verbi *essere* e *avere* sono spesso erroneamente taggati come `[AUX]`.

1.2.1 Documentazione e sito

In conclusione, nonostante non si tratti di uno strumento da utilizzare in un contesto professionale, sono sia soddisfatto del lavoro svolto, sia curioso di sapere quanto possa ancora aumentare la *bravura* del mio *PoS Tagger*, se si integrassero metodologie alternative di interpretazione morfologica. Naturalmente è mio interesse proseguire e migliorare l'attuale progetto, che si trova nei seguenti repository `GitHub`:

- https://github.com/marcoscaletta/viterbi
- https://github.com/marcoscaletta/translator

La documentazione per l'utilizzo del *translator* verrà presentata in un'apposita sezione di questa relazione. Dal momento che ho ritenuto che sarebbe stato più comodo poter utilizzare il PoS Tagger implementato senza dover scaricare codice o effettuare configurazioni, ho creato un **semplice sito** tramite il quale è possibile **testare l'applicazione** di *tagging*. È possibile trovare tale sito al seguente link: http://viterbi.herokuapp.com.

2 Viterbi

2.1 I processi Markoviani e il PoS Tagging

Intuitivamente, nell'NLP, è possibile sfruttare la proprietà markoviana nel seguente modo: quando si elabora il ruolo di ogni parola rispetto a una frase (*Part of Speech*) si giunge alla conclusione che, maggiore è la distanza tra due parole, minore è la probabilità che il *PoS* assegnato alla prima sia rilevante per il *tagging* della seconda. Il seguente esempio può chiarire questo concetto. La frase *Quando il sole sorge* viene taggata nel seguente modo dalla mia implementazione di Viterbi:

`[[quando|SCONJ], [il|DET], [sole|NOUN], [sorge|VERB]]` [1]

Il fatto che la frase inizi con il tag `SCONJ` non è rilevante rispetto al *PoS Tag* dell'ultima parola della frase, che in questo caso è `VERB`. Infatti, se si inverte l'ordine delle parole della frase mantenendolo sintatticamente equivalente a prima, si nota che l'assegnazione dei tag non viene influenzata: *Quando sorge il sole* ha come tag

`[[quando|SCONJ], [sorge|VERB], [il|DET], [sole|NOUN]].`

Dunque è chiaro che si possa optare per un'approssimazione ai fini di una meno costosa computazione; al contrario sarebbe necessario sprecare molte risorse (soprattutto temporali) per effettuare il training prima, e per eseguire l'algoritmo di tagging poi, considerando ogni possibile combinazione tra i tag di una frase, cosa che è oggettivamente priva di senso. Questo perchè, come già accennato, da una parte il tempo di calcolo necessario sarebbe esagerato rispetto all'effettivo guadagno che si avrebbe dal risultato; in secondo luogo buona parte di questo lavoro sarebbe proprio inutile, perchè, il ruolo di una parola all'interno di un discorso è condizionato da quello di un'altra con una rilevanza inversamente proporzionale alla distanza tra le due.

Sfruttando la *proprietà markoviana*, è possibile effettuare il calcolo delle probabilità delle assegnazioni in modo più semplice; questo può essere fatto limitando l'influenza tra due parole a una distanza di misura costante (cioè fissa rispetto alla lunghezza della frase). In altre parole, se si decide che non ha senso definire un algoritmo di tagging che prenda in considerazione due parole collocate a una distanza maggiore di m, si calcolano le probabilità dei cosiddetti *m-grams*:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1}, \ldots, t_{i-m+1})$$

cioè, la probabilità di avere una certa sequenza di tag (t_1^n) di lunghezza n è data dal prodotto delle probabilità di ogni sottosequenza di lunghezza massima m. Quindi ogni tag viene assegnato in base ai precedenti $m-1$ tag. In particolare l'algoritmo implementato sfrutta quest'approssimazione definendo la lunghezza $m = 2$ e quindi la distanza pari a 1:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1}).$$

[1] Tutti gli esempi di PoS Tagging presentati sono eseguiti dal PoS Tagger implementato ed è possibile replicarli al seguente link https://viterbi.herokuapp.com

2.2 Descrizione dell'implementazione

Il progetto è stato implementato nel linguaggio `JAVA`, sia per una preferenza personale, sia per i vantaggi dati dal paradigma di programmazione ad oggetti. Le principali classi sono:

- `Tag`: enumerazione dei possibili *tag*.
- `BiGram`: rappresentazione della sequenza `Tag-Tag`.
- `PoSTag`: implementazione dell'interpretazione morfologica `Word-Tag`.
- `TreeBankReader`: lettore del tree bank.
- `PoSTagger`: contiene la funzione di *PoSTagging* che definisce la baseline per l'accuracy.
- `Viterbi`: sottoclasse di `PoSTagger` che implementa un *PoSTagging* più specifico (in particolare utilizza l'algoritmo di Viterbi).

Non entrerò nel merito dei metodi specifici delle classi sia perchè risulterebbe pedante e soprattutto perchè nel repository github sono presenti i `JAVADOC` per tutti i metodi implementati.

2.2.1 Lettura del TreeBank

Per effettuare il PoS Tagging è necessario analizzare il tree bank ai fini del training: viene creato un oggetto `TreeBankReader` che, durante la lettura da file (uno specifico tree bank) effettua il conteggio degli oggetti `BiGram` e `PoSTag`.

Prima di eseguire il PoS Tagging, vengono calcolate le probabilità in base alle informazioni ricavate dal `TreeBankReader`, grazie alle quali è possibile interpretare la frase di input.

2.2.2 Algoritmo di default per il PoSTagging

Ogni parola viene interpretata con il tag a cui è stata assegnata più volte nel dataset di training. Si tratta del metodo più banale di PoS Tagging con apprendimento, quindi si considera il valore dell'accuratezza di questo algoritmo per definire la *baseline* sopra la quale è necessario trovarsi per avere risultato interessante.

```
function VITERBI(observations of len T,state-graph) returns best-path

  num-states ← NUM-OF-STATES(state-graph)
  Create a path probability matrix viterbi[num-states+2,T+2]
  viterbi[0,0] ← 1.0
  for each time step t from 0 to T do
    for each state s from 0 to num-states do
      for each transition s' from s specified by state-graph
        new-score ← viterbi[s, t] * a[s,s'] * b_s'(o_t)
        if ((viterbi[s',t+1] = 0) || (new-score > viterbi[s', t+1]))
          then
                viterbi[s', t+1] ← new-score
                back-pointer[s', t+1] ← s
  Backtrace from highest probability state in the final column of viterbi[] and
return path
```

Figure 1: Algoritmo di Viterbi implementato

2.2.3 Viterbi per il PoS Tagging

L'algoritmo in Figura 1 è stato utilizzato come linea guida per calcolare il *Viterbi Path*, mentre quello presentato a lezione è quello in Figura 2. Tale scelta è dovuta al fatto che il primo pseudocodice è più chiaro, più semplice ed è a un livello di astrazione più basso, infatti il secondo utilizza gli operatori di alto livello `max` e `argmax`, che nel primo sono implementati nel terzo ciclo annidato sfruttando il calcolo del valore da assegnare allaa variabile `new-score`. Il risultato dell'algoritmo è un percorso (*path*) sulla matrice ricavato facendo backtrack rispetto ai valori contenuti nella matrice `backtrace`.

Dal momento che i valori calcolati possono arrivare, soprattutto con frasi molto lunghe, ad essere estremamente piccoli, si è adottata la scelta implementativa di utilizzare i logaritmi nella procedura di calcolo. Inoltre le conseguenze di una propagazione di errori sarebbero state molto gravi, visto il numero di volte in cui si sarebbero dovute applicare operazioni di moltiplicazione.

2.2.4 Smoothing per parole sconosciute

Dal momento che è molto probabile incontrare parole sconosciute durante la procedura di PoS Tagging, è necessario definire un approccio per la risoluzione di questo problema. Il metodo più semplice è quello di interpretare come

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
viterbi[s,1] $\leftarrow a_{0,s} * b_s(o_1)$
backpointer[s,1] $\leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
viterbi[s,t] $\leftarrow \max_{s'=1}^{N}$ *viterbi*[s',$t-1$] $* a_{s',s} * b_s(o_t)$
backpointer[s,t] $\leftarrow \operatorname{argmax}_{s'=1}^{N}$ *viterbi*[s',$t-1$] $* a_{s',s}$
viterbi[q_F,T] $\leftarrow \max_{s=1}^{N}$ *viterbi*[s,T] $* a_{s,q_F}$; termination step
backpointer[q_F,T] $\leftarrow \operatorname{argmax}_{s=1}^{N}$ *viterbi*[s,T] $* a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from *backpointer*[q_F, T]

Figure 2: Algoritmo di Viterbi visto a lezione

`PROPN` le parole sconosciute, definendo un nuovo `PoSTag [ParolaNuova|PROPN]` e la probabilità

$$P(\texttt{[ParolaNuova|PROPN]}) = 1.$$

Al contrario, la soluzione che ho adottato è un po' più complessa, ma anche *user friendly*: durante la creazione dell'oggetto `Viterbi`, è possibile definire come parametro del costruttore un array contenente i possibili tag da assegnare alle parole sconosciute. Se ad esempio si pensa che abbia senso assegnare a tali vocaboli i tag `PROPN` e `VERB`, allora è sufficiente definire come parametro un array `[Tag.PROPN,Tag.VERB]` e il metodo di PoS Tagging, nel momento in cui viene analizzato un nuovo termine (`ParolaNuova`), definisce le seguenti probabilità:

$$P(\texttt{[ParolaNuova|PROPN]}) = 1/2$$
$$P(\texttt{[ParolaNuova|VERB]}) = 1/2.$$

In generale, dato un array di tag $\texttt{[Tag}_1\texttt{,Tag}_2\texttt{, ...,Tag}_n\texttt{]}$ di dimensione n, vengono generate le seguenti n probabilità:

$$P(\texttt{[ParolaNuova|Tag}_1\texttt{]}) = 1/n$$
$$\vdots$$
$$P(\texttt{[ParolaNuova|Tag}_n\texttt{]}) = 1/n.$$

Nell'attuale implementazione viene utilizzato il seguente array

```
[Tag.NOUN, Tag.ADJ, Tag.PROPN, Tag.VERB],
```

assegnando una probabilità pari a 1/4 ad ogni possibile `PoSTag`. Tale scelta è dovuta al fatto che sia i nomi comuni che gli aggettivi che i verbi, oltre ad essere estremamente numerosi, sono le parti del discorso in continua evoluzione. Tale approccio implementativo risulta vincente, dal momento che usando solo il tag `PROPN` l'accuratezza viene degrata quasi del 3%.

2.2.5 Accuratezza

L'accuratezza viene calcolata per l'oggetto `PoSTagger` e, dal momento che `Viterbi` ne è una sottoclasse, si utilizza il medesimo metodo sia per il calcolo della *baseline* che per l'accuracy di *Viterbi*. I risultati sono i seguenti:

1. Tutte le parole sono lette ed elaborate come *minuscole*
   - **Baseline**: 90.4%
   - **Viterbi**: 94.2%
2. Solo le parole taggate come `PROPN` sono memorizzate ed elaborate con la lettera iniziale *maiuscola*
   - **Baseline**: 86.7%
   - **Viterbi**: 92.4%
3. Tutte le parole **non** taggate come `PROPN` sono memorizzate sia con l'iniziale maiuscola che con l'iniziale minuscola.
   - **Baseline**: 90.5%
   - **Viterbi**: 94.7%
4. Tutte le parole **non** taggate come `PROPN` sono memorizzate due volte
   - **Baseline**: 90.8%
   - **Viterbi**: 94.88%
5. Le parole **non** taggate come `PROPN` sono memorizzate due volte, nel caso inizino con la lettera maiuscola vengono salvate una volta con la lettera iniziale maiuscola e una volta con la lettera iniziale minuscola.
   - **Baseline**: 90.9%

- **Viterbi**: 94.92%

Le giustificazioni per le prime due modifiche alla versione 1: la 2 evidenzia l'importanza di memorizzare i nomi propri con la lettera maiuscola, ma, come si può intuire dai risultati, non si tratta della strategia migliore. La numero 3 pone maggiore attenzione sul fatto che le parole che non sono nomi propri, possano essere presenti nella stringa anche con l'iniziale maiuscola. In questo caso l'accuracy aumenta in modo interessante, anche se si mantiene in un piccolo intorno del 94%. Le versioni 4 e 5, invece, pongono l'accento sul fatto che il ruolo dei termini che non sono nomi propri, durante il procedimento di PoS Tagging, possa essere più rilevante degli altri, e quindi tali parole vengono memorizzate due volte. Il fatto che l'accuratezza aumenti non è un risultato che mi sarei aspettato, per questo motivo ritengo che ci siano due possibili spiegazioni: o l'ipotesi precedente può essere considerata come vera, oppure questi risultati sono segnali di un bug nel procedimento di training e di testing, problema che riguarda i dataset stessi. Questo errore può provocare l'overfitting nell'apprendimento e quindi falsare la procedura di verifica dell'accuratezza.

In conclusione, per quanto riguarda la precisione dell'interpretazione, si può essere soddisfatti dei risultati ottenuti, poichè l'accuratezza ha approssimativamente una baseline al 90% (ad esclusione della versione 2) e un incremento delle prestazioni del 4%, circa, con l'utilizzo di *Viterbi*.

Siccome le variazioni morfologiche, anche se importanti, vengono ignorate, sarebbe interessante sapere quanto ancora possano essere migliorate le performance, considerando anche questa caratteristica del linguaggio.

3 Translator

La procedura di **traduzione direct**, che avviene solo dopo il PoS Tagging, è suddivisa nelle seguenti sottoparti:

- Manipolazione della frase
- Traduzione
- Applicazione regole grammaticali

3.1 Manipolazione della frase

Vengono gestite le parole composte:

`[[spada|NOUN],[laser|ADJ/NOUN]]` → `[[spada-laser|NOUN]]`

Inoltre, prima della traduzione, vengono aggiunti i possibili soggetti sottointesi di inizio frase:

```
[[ha|AUX],[fatto|VERB]]
```

viene modificato in

```
[[qualcuno-qualcosa|PRON],[ha|AUX],[fatto|VERB]]
```

3.2 Traduzione

La traduzione parola-per-parola viene effettuata grazie a un dizionario che definisce le associazioni tra termini italiani, PoSTag e termini inglesi. Il formato è il seguente

```
<parola-italiana>/<PoSTag>/<parola-inglese>/[IRR]
```

ad esempio

```
spada-laser/NOUN/lightsaber
fatto/VERB/made/IRR
```

Il parametro `IRR` è opzionale e indica se la parola inglese è un verbo irregolare, in modo da poter applicare le corrette regole grammaticali successivamente.

3.3 Applicazione delle regole grammaticali

È necessario ricordare che, per risolvere il problema del **genitivo sassone**, ho utilizzato una piccola base di conoscenza, in modo da sapere quali siano le parole per le quali si debba effettuare il riordino della frase e l'inserimento del termine `'s`. Le regole applicate sono le seguenti

- Se un verbo irregolare è preceduto da `do/did`, questi ultimi vengono eliminati.
- Le parole della frase vengono riordinate:

  `[[mossa|NOUN],[leale|ADJ]]` $\rightarrow_{trad}$ `[[move|NOUN],[fair|ADJ]]`
  `[[move|NOUN],[fair|ADJ]]` $\rightarrow_{riorder}$ `[[fair|ADJ],[move|NOUN]]`

- Se il termine `of` fa riferimento a un termine salvato nella base di conoscenza come `PERSON`, avviene uno shifting di tutti i termini a cui `of` è legato, in modo da poter inserire il **genitivo sassone** e viene eliminato l'articolo nel caso compaia prima dell'oggetto del genitivo.

- `someone-something` viene sostituito con `it` se precede il termine `is` e con `he` se precede il termine `made`. (Si tratta di un escamotage necessario dal momento che l'obiettivo è la traduzione di poche frasi che rispettano questa regola).
- Infine le parole a inizio frase e i nomi propri vengono scritte con l'iniziale maiuscola.

3.4 Risultati

Le tre frasi da tradurre per esercizio sono:

- `È la spada laser di tuo padre`
- `Ha fatto una mossa leale`
- `Gli ultimi avanzi della vecchia repubblica sono stati spazzati via`

I risultati del PoS Tagging sono i seguenti

- `[è AUX][la DET][spada NOUN][laser NOUN][di ADP][tuo DET][padre NOUN]`
- `[ha AUX][fatto VERB][una DET][mossa NOUN][leale ADJ]`
- `[gli DET][ultimi ADJ][avanzi NOUN][di ADP][la DET][vecchia ADJ][repubblica NOUN][sono AUX][stati AUX][spazzati VERB][via ADV]`

Le traduzioni direct ottenute sono:

- `It is your father 's lightsaber`
- `He made a fair move`
- `The last remnants of the old republic have been swept away`