UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI INFORMATICA

SCUOLA DI SCIENZE DELLA NATURA

Corso di Laurea Magistrale in Informatica

Progetto Tecnologie del Linguaggio Naturale
La Magia NER Nascosta

Davide Mauri

ANNO ACCADEMICO
2021/2022

Indice

1 Introduzione

Questo documento è la documentazione richiesta per lo sviluppo del progetto della prima parte del corso Tecnologie del Linguaggio Naturale tenuto dall'Università degli Studi di Torino nell'anno accademico 2021/2022. L'unico autore di quanto presentato è Davide Mauri.

L'obiettivo di questo progetto era implementare l'algoritmo di Viterbi per effettuare NER tagging e successivamente valutarlo su un corpus di test, anche confrontandolo con una baseline.

L'implementazione è contenuta in un notebook scritto in Python disponibile al repository `https://github.com/Davidesfed/TLN-21-22`.

2 Implementazione

2.1 Dati

I corpus necessari per l'addestramento dei modelli sono stati ottenuti dal seguente repository: `https://github.com/Babelscape/wikineural/tree/master/data/wikineural`. Le lingue scelte sono l'italiano e l'inglese.

I tag considerati sono quelli presenti nei corpus annotati e sono quindi `'O'` `'ORG'` `'LOC'` `'PER'` `'MISC'`. Si suppone che la probabilità che due Named Entity diverse siano perfettamente adiacenti sia sostanzialmente zero, in modo da poter ignorare i prefissi B- e I- presenti nel corpus, in quanto questi sono immediati una volta che

2.2 Hidden Markov Model

Ho scelto di implementare un Hidden Markov Model utilizzando una classe. Per generare un'oggetto HMM è necessario fornire un corpus e un insieme di tag da considerare. Sul corpus vengono quindi calcolate le probabilità per ciascun tag di essere quello iniziale o quello finale, oltre che le probabilità di transizione da un tag ad un altro ed infine le probabilità di emissione di ciascuna parola presente nel corpus per ogni tag del tagset. Vi sono poi dei metodi getter per recuperare queste informazioni. Laddove si cerchi di ottenere la probabilità di emissione di una parola non presente nel corpus, il comportamento seguito dal modello è dettato dalla strategia di smoothing, che può essere settata tramite un metodo apposito a uno dei seguenti tre valori: `'always_o'`, `'misc_or_o'`, `'uniform'`. Nel primo caso si avrà probabilità di emissione pari a uno se il tag è `'O'` e 0 altrimenti. Nel secondo caso si avrà probabilità di emissione pari a 0.5 se il tag è `'O'` o `'MISC'` e 0 altrimenti. Nell'ultimo caso, invece la probabilità di emissione sarà uniforme, cioè pari a $\frac{1}{|tagset|}$ per ogni tag.

2.3 Viterbi

L'algoritmo di viterbi è stato implementato tramite una funzione principale chiamata, appunto, `viterbi`. Essa prende in input un modello e una frase. Il modello si suppone che sia un HiddenMarkovModel e che quindi abbia i getter per le probabilità di cui si parla nella sezione precedente. L'algoritmo

si divide in quattro step fondamentali, ciascuno dei quali è evidenziato da una sua propria funzione:

1. **Inizializzazione.** Vengono calcolate le probabilità di inizio: sapendo la prima parola della frase, si moltiplica le probabilità iniziale di ciascun tag per la probabilità che quel tag emetta quella parola. Queste quantità sono memorizzate in un'apposita matrice V.

$$V[t, p_0] = P_{start}(t)P_{emission}(p_0|t)$$

2. **Calcolo iterativo.** Per ogni parola successiva p, per ogni tag t, si seleziona quale sia il tag t' assegnato alla parola precedente p' che massimizza la probabilità di assegnare t a p. Per farlo si moltiplica il valore della matrice di viterbi relativo alla parola p' e al tag t' per la probabilità di transizione da t' a t per la probabilità di emissione di p da t. Si tiene traccia di quale sia il tag che massimizza questa quantità nella matrice backpointer B. In formule:

$$V[t, p] = \max_{t'} V[t', p']P_{transtion}(t', t)P_{emission}(p|t)$$

$$B[t, p] = \operatorname*{argmax}_{t'} V[t', p']P_{transtion}(t', t)$$

   Ad ogni step, la quantità $V(t, p)$ rappresenta la probabilità che assegnando un tag a tutte le parole prima di p si assegni a p il tag t.

3. **Terminazione.** Per ogni tag t si moltiplica il contenuto della matrice $V[t, p_f]$ per la probabilità finale di t. Tra i valori così ottenuti si prende il massimo e il tag ad esso corrispondente. Questo sarà memorizzato in quanto serve per iniziare il passo successivo.

4. **Backtrace.** Partendo dal tag ottenuto al punto precedente, si utilizza la matrice B procedendo a ritroso e generando, quindi, la sequenza di tag (in ordine inverso) che sarà assegnata alla frase.

L'output di viterbi è quindi proprio la sequenza di tag assegnati alle parole che compongono la frase.

2.4 Baseline

In questo caso si è scelta una baseline molto semplice: il modello, partendo dal corpus, calcola le frequenze dei tag per ogni parola.

L'algoritmo di NER tagging invece, per ogni parola in una frase, se tale parola è contenuta nel corpus, allora le assegna il tag più frequente; altrimenti le assegna `'MISC'`.

2.5 Classe Tagger

Ho deciso di implementare anche una classe Tagger, il cui scopo è facilitare la valutazione dei modelli, fornendo un'unica interfaccia che viene utilizzata dalle varie funzioni di valutazione. In particolare la un oggetto Tagger viene costruito dando un modello e un algoritmo di NER tagging. Avremo quindi `hmm_viterbi` e `baseline` come due oggetti Tagger.

3 Valutazione

Il corpus utilizzato nella valutazione è preso dallo stesso repository del corpus utilizzato nel training. La dimensione del test set è di 11597 frasi per l'inglese e 11069 frasi per l'italiano. I risultati qui presentati sono stati ottenuti utilizzandole tutte sia per l'italiano che per l'inglese.

Per valutare ciascun tagger utilizziamo alcuni metodi diversi: accuracy, precision, recall e alcune frasi di esempio. La strategia di smoothing utilizzata dall'Hidden Markov Model è: `'always_o'`.

L'accuracy è stata calcolata semplicemente contando quante parole (punteggiatura compresa) hanno ricevuto il tag corretto. Precision e recall sono invece state calcolate per ogni tag, in modo da poter vedere quali siano i tag meglio assegnati dal modello. Le frasi di esempio sono, invece, le seguenti:

- Harry Potter's true home is Hogwarts Castle.
- Harry told her about their meeting at Diagon Alley.
- Mr Dursley was director of a company named Grunnings that manufactured drills.

con le loro rispettive traduzioni in italiano.

3.1 Risultati

Vediamo quindi i risultati in forma tabellare, e per semplicità riportiamo solamente quelli relativi alla lingua inglese. I risultati relativi all'italiano si possono facilmente osservare nel notebook.

Modello	Accuracy
Viterbi	0.9376
Baseline	0.9173

Precision					
Modello	O	ORG	LOC	PER	MISC
Viterbi	0.9504	0.7973	0.7942	0.8999	0.7725
Baseline	0.9614	0.7428	0.7378	0.8697	0.3342

Recall					
Modello	O	ORG	LOC	PER	MISC
Viterbi	0.9614	0.7428	0.6785	0.7094	0.4335
Baseline	0.9753	0.4315	0.6061	0.6907	0.4386

Possiamo ora vedere i test sulle frasi di esempio. Nelle tabelle sottostanti sono presenti solamente i nomi per ragioni estetiche. Ciascuna delle parole omesse è stata correttamente taggata con `'O'` da tutti i tagger. In ogni tabella la linea "Target" contiene i tag definiti da me e ritenuti corretti.

Esempio 1					
	Harry	Potter	home	Hogwarts	Castle
Viterbi	PER	PER	O	MISC	MISC
Baseline	PER	MISC	O	MISC	LOC
Target	PER	PER	O	LOC	LOC

Esempio 2					
	Harry	her	meeting	Diagon	Alley
Viterbi	PER	O	O	O	LOC
Baseline	PER	O	O	MISC	LOC
Target	PER	O	O	LOC	LOC

Esempio 3						
	Mr	Dursley	director	company	Grunnings	drills
Viterbi	O	LOC	O	O	O	O
Baseline	MISC	LOC	O	O	MISC	O
Target	PER	PER	O	O	ORG	O

In ultimo possiamo vedere i tempi di calcolo sul test set. Si noti che, se vi fosse la necessità di sperimentare setting diversi per la valutazione, si è notato che le performance non variano di molto anche con una dimensione del test set molto ridotta.

Tempi di calcolo		
	Italiano	Inglese
Viterbi	1726s	1259s
Baseline	353s	222s

4 Conclusioni

Nel lavorare a questo progetto ho avuto occasione di implementare a mano l'algoritmo di Viterbi, nonché una classe che rappresentasse un Hidden Markov Model.

Le performance che ho ottenuto sono piuttosto soddisfacenti, almeno sul test set fornito. Come si può vedere dai tre esempi giocattolo, però, il sistema probabilmente non performerebbe altrettanto bene di fronte a molte entità nuove. A fronte di ciò potrebbe essere utile cambiare la strategia di smoothing, con l'accortezza che l'eventuale nuova strategia non dovrebbe impiegare troppo tempo a decidere quale tag assegnare a una parola nuova, pena perdere significativamente in velocità di calcolo.