Relazione del progetto di Tecnologie del Linguaggio Naturale

Opzione 1: La magia NER nascosta

Luca Demma**. Matricola**: 803694

Introduzione

L'elaborazione del linguaggio naturale (NLP) è una disciplina in rapida crescita all'interno dell'intelligenza artificiale, che si concentra sullo sviluppo di tecniche per l'analisi e la comprensione del linguaggio umano. In questo progetto, ci concentriamo su una sottotecnica specifica dell'elaborazione del linguaggio naturale nota come Named Entity Recognition (NER).

Il NER è il processo di individuazione e classificazione delle parole in un testo in base a categorie predefinite, come nomi propri, luoghi, organizzazioni e così via. Questa tecnologia è di fondamentale importanza in molti campi, come l'elaborazione del testo, la ricerca e l'analisi dei dati.

In questo progetto, abbiamo implementato un algoritmo di NER basato sull'Hidden Markov Model (HMM) per il learning e l'algoritmo di Viterbi per il decoding. L'HMM è un modello probabilistico che ci permette di rappresentare la sequenza di parole in un testo come una serie di stati nascosti e osservabili. L'algoritmo di Viterbi, d'altra parte, ci permette di trovare la sequenza di stati nascosti più probabile che ha generato una determinata sequenza di osservazioni.

Obiettivo

L'obiettivo principale del progetto consiste nell'implementare un algoritmo che permetta di effettuare il task di Named Entity Recognition (NER) in grado di individuare e classificare correttamente le entità presenti in un testo. In particolare, l'obiettivo è quello di utilizzare l'Hidden Markov Model (HMM) per il learning e l'algoritmo di Viterbi per il decoding.

Per il learning è stato usato un dataset disponibile gratuitamente a questo indirizzo: https://github.com/Babelscape/wikineural/tree/master/data/wikineural

Gli HMM sono stati maggiormente utilizzati in NLP per il riconoscimento del parlato e per il PoS-tagging, e meno per il NER.

L'algoritmo di Viterbi, d'altra parte, è un algoritmo di decodifica efficiente utilizzato per trovare la sequenza più probabile di stati in un modello HMM.

Concetti teorici

Hidden Markov Model

L’ Hidden Markov Model è un modello statistico che è spesso usato nell’ambito del Natural Language Processing in cui gli stati sono nascosti. Gli stati nascosti non sono direttamente osservabili, ma gli output dipendenti dagli stati, sono visibili.
Si basa sulla nozione di Markov Chain in cui si ipotizza che lo stato futuro dipende soltanto dallo stato attuale. Questa proprietà è chiamata *Markov Property*

- **Markov Property**: Per calcolare la proprietà di transire in un certo stato al tempo x dobbiamo tenere in considerazione soltanto lo stato al tempo $x - 1$.

Un Hidden Markov Model è composto da una Markov Chain con degli stati nascosti e da un insieme di variabili osservabili.

Nel caso del progetto:

- **Variabili osservabili** $\Rightarrow$ le singole parole della frase in esame
- **Stati nascosti** $\Rightarrow$ i NER tag che vogliamo assegnare alle singole parole

Gli HMM forniscono la sequenza di stati nascosti più probabile data una sequenza di variabili osservabili. Cioè nell’ambito del progetto, restituiscono la sequenza di NER tag più probabile dando in input una sequenza di parole (frasi).
Gli HMM per eseguire questo task calcolano la maggiore *Joint Probability* delle probabilità di passare da uno stato all’altro, quindi la probabilità che da un certo NER tag si passi a un altro specifico NER tag, e la probabilità che una data parola abbia assegnata il NER tag in questione.

In modo più formale: $\underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(w_i \mid t_i\right) P\left(t_i \mid t_{i-1}\right)$

Dove:

- w_i rappresenta la parola w al time step i
- t_i rappresenta il tag t al time step i
- n rappresenta il numero di parole presenti nella frase in analisi

HMM Learning step

L’ HMM essendo un modello probabilistico ha ovviamente bisogno di conoscere le probabilità della Joint Probability per calcolare la sequenza di stati con la probabilità massima. Nella formula troviamo due probabilità:

- **Transition Probability**: $P\left(t_i \mid t_{i-1}\right) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$ la probabilità di passare da uno stato all’altro (da un NER tag a un altro)
- **Emission Probability**: $P\left(w_i \mid t_i\right) = \frac{C(t_i, w_i)}{C(t_i)}$ la probabilità che una parola abbia assegnata un certo NER tag

Per recuperare queste due probabilità per le parole e per i tag abbiamo bisogno di un training set da cui poter calcolare le probabilità citate, semplicemente “contando” il numero di occorrenze. Come training set per questo progetto sono stati utilizzati i dati presenti su https://github.com/Babelscape/wikineural/tree/master/data/wikineural per l’inglese e per l’italiano.

HMM Decoding step

Dopo aver effettuato l’addestramento del modello sul training set scelto, e quindi avendo a disposizione le matrici delle probabilità delle Transition e delle Emission, è possibile effettuare il decoding di una frase non presente nel training set e ricavare la sequenza di NER tag più probabile.

Come già accennato precedentemente, per calcolare la sequenza di tag più probabile è necessario calcolare la Joint Probability per ogni combinazione di parola/tag e trovare la sequenza con la probabilità maggiore: $\underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(w_i \mid t_i\right) P\left(t_i \mid t_{i-1}\right)$.

L’utilizzo di quest’approccio ha un grosso problema in termini di costo, dato che il tempo di esecuzione diventa esponenziale. Più precisamente abbiamo $O\left(T^n\right)$ con:

- T numero di NER tag possibili
- n numero di parole presenti nella frase in esame.

Un metodo per migliorare i costi è di utilizzare l’algoritmo di Viterbi che usa la programmazione dinamica per evitare di calcolare quei percorsi che sappiamo con certezza avere minore probabilità di altri, così avendo $O\left(n * T^2\right)$.

L’algoritmo di Viterbi

L’algoritmo di Viterbi è un algoritmo che usa la programmazione dinamica per trovare il cammino ottimo del HMM. L’algoritmo comincia inizializzando le probabilità di essere in ogni stato al primo time step, usando le probabilità dello stato iniziale presenti nel HMM. A ogni step successivo, l’algoritmo calcola la probabilità di essere in ogni stato dato lo stato precedente e l’osservazione corrente, e la probabilità che la sequenza fino al time step in esame sia stata prodotta da una sequenza di stati. La sequenza di stati nascosti più probabile è quindi determinata facendo backtracking tra le probabilità calcolate a partire dall’ultimo time step e andando a ritroso.

In pratica ogni volta che abbiamo un nodo e percorsi multipli per arrivare a quel nodo, consideriamo tutte le probabilità dei cammini che ci portano al nodo in esame e manteniamo soltanto il cammino con probabilità maggiore.

Come già affermato in precedenza, la complessità dell’algoritmo di Viterbi è molto minore rispetto al metodo naive, quindi abbiamo: $O(n * T^2)$.

Implementazione

Per l’implementazione è stato scelto di utilizzare Python come linguaggio di programmazione per la sua semplicità di utilizzo e per le numerose librerie in ambito data science che hanno semplificato l’implementazione del progetto. Ho utilizzato le seguenti librerie Python:

- pandas: per convertire le matrici (rappresentate da dizionari in 2 dimensioni) in file CSV
- conllu: per effettuare il parsing del file di training set in formato CoNLL-U

Sono stati implementati l’algoritmo di Learning della HMM e di Decoding usando Viterbi.

Il Learning salva in formato CSV (comma delimited, con string delimiter nullo) le matrici del conteggio delle emission e delle transition in formato CSV e le matrici delle probabilità in formato CSV e pickle, quest’utlimo formato è stato scelto per salvare gli oggetti rappresentanti le matrici così da poter essere utilizzati dalla parte di decoding senza dover ricalcolare le probabilità ogni qualvolta.

L’output del Decoding salva in un file CSV (tab delimited, con string delimiter nullo) in formato CoNLL-U il risultato del decoding effettuato sul test set, comparandolo con il tag presente nel training set.

Smoothing

Come smoothing per le parole non presenti nel training set è stato scelto uno smoothing uniforme. Cioè assegniamo alla parola sconosciuta la stessa probabilità a ogni NER tag:

$P(unk|t_i) = 1/count(NERtags)$

Baseline

Per fare una valutazione dell'algoritmo implementato è stata usata una baseline che consiste nell'assegnare a una parola il tag più frequente se è presente nel training set e *B-MISC* altrimenti

Accorgimenti utilizzati

- La moltiplicazione di molte probabilità tra loro può portare a un underflow numerico: essendo i valori molto piccoli, se li moltiplichiamo tra loro molte volte il risultato è un numero molto piccolo e può diventare instabile. Per risolvere questo problema ho usato la somma dei logaritmi delle probabilità invece che la moltiplicazione. Ciò risolve il problema dell'underflow numerico e fornisce score di probabilità più ragionevoli
- Ho utilizzato lo pseudocounting per risolvere il problema della moltiplicazione di probabilità di parole presenti nel training set ma che per un certo assegnamento di NER tag appaiono zero volte. Nel caso in cui appaia questa coppia parola/tag la probabilità sarebbe 0 e il risultato dello score sarebbe zero in questo caso. L'obiettivo dello pseudocounting è di assegnare probabilità molto basse a quelle coppie non presenti nel training aggiungendo alla fase di counting un +1 a tutte le coppie word/tag. Questo permette di avere una probabilità seppur molto bassa ma che non azzera completamente lo score di probabilità nella moltiplicazione.

Risultati, Valutazioni e considerazioni finali

Di seguito verranno esposti i risultati del training e del decoding effettuati sui dataset in italiano e in inglese, sulle frasi fornite nelle slide e il confronto con la baseline.

Learning (IT)

Transitions Count

	START	O	B-MISC	I-MISC	B-LOC	I-LOC	B-ORG	B-PER	I-PER	I-ORG
O	85972	1814462	6784	14729	58587	17036	9131	8842	43271	8182
B-PER	1764	50105	7	1	60	13	121	38	8	37
B-MISC	168	21052	282	41			7		3	2
B-LOC	380	75323			214	13	5			
B-ORG	116	17454				1	8			
I-MISC		357	14481	19146						
I-PER			1	2	10	1	1	43272	6292	1
I-LOC				1	17063	9491				
I-ORG							8304			4175

Transitions Probabilities (valori arrotondati)

	START	O	B-MISC	I-MISC	B-LOC	I-LOC	B-ORG	B-PER	I-PER	I-ORG
B-PER	0.02	0.03	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
I-PER	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.83	0.13	0.00
B-ORG	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I-ORG	0.00	0.00	0.00	0.00	0.00	0.00	0.47	0.00	0.00	0.34
B-LOC	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I-LOC	0.00	0.00	0.00	0.00	0.22	0.36	0.00	0.00	0.00	0.00
B-MISC	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I-MISC	0.00	0.00	0.67	0.56	0.00	0.00	0.00	0.00	0.00	0.00
O	0.97	0.92	0.31	0.43	0.77	0.64	0.52	0.17	0.87	0.66

Emissions Count (estratto)

	Nella	stagione	2003	-	2004	vince	un'	altra	Supercoppa	italiana
O	1186	2533	315	6554	381	349	2391	538		508
B-MISC	3		1		6				56	
I-MISC			10	574	14		2	4		32
I-LOC				7						1
I-ORG				1						2
B-LOC										
B-ORG										
I-PER										
B-PER										

Emissions Probabilities (estratto, valori arrotondati)

	Nella	stagione	2003	-	2004	vince	un'	altra	Supercoppa	italiana
B-PER	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
I-PER	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
B-ORG	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
I-ORG	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01
B-LOC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
I-LOC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
B-MISC	0.00	0.00	0.01	0.00	0.02	0.00	0.00	0.00	0.88	0.00
I-MISC	0.00	0.00	0.03	0.08	0.04	0.00	0.00	0.01	0.02	0.06
O	0.99	1.00	0.94	0.92	0.93	0.98	1.00	0.98	0.02	0.92

Per i dati del learning in inglese e i risultati del decoding di entrambe le lingue si rimanda ai file Excel presenti nella cartella /data/outputs/ , nella cartella DECODING/EXCEL sono presenti gli spreadsheet con i risultati e con le relative formule delle valutazioni.

Decoding per le frasi presenti nelle slide

La	O
vera	O
casa	O
di	O
Harry	B-PER
Potter	I-MISC
è	O
il	O
castello	O
di	O
Hogwards	O
.	O

Harry	O
le	O
raccontò	O
del	O
loro	O
incontro	O
a	O
Diagon	O
Alley	O
.	O

Mr	O
Dursley	O
era	O
direttore	O
di	O
una	O
ditta	O
di	O
nome	O
Grunnings	O
,	O
che	O
fabbricava	O
trapani	O
.	O

Valutazioni

Per valutare la bontà dell'algoritmo sono stati utilizzati due metodi formali di valutazione e confrontati con quelli della baseline:

- **Accuracy**: ci dice quanti input sono stati classificati correttamente con la seguente formula: $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$. Fornisce una vista generale sulla correttezza della classificazione ma non è molto significativo su dataset molto sbilianciati come quello utilizzato, in cui sono presenti moltissimi NER tag == O
- **Precision e Recall**: calcolati per ogni NER tag. Questo metodo ci da una vistione migliore in quanto:
  - **Precision**: fornisce quanti assegnamenti del tag in esame sono corretti: $\text{Precision} = \frac{TP}{TP+FP}$
  - **Recall**: fornisce quanti tag "veramente" positivi sono stati assegnati correttamente: $\text{Recall} = \frac{TP}{TP+FN}$

Accuracy

	IT	IT baseline	EN	EN baseline
Accuracy	0.928	0.941	0.907	0.931

Precision / Recall

IT :

	Precision IT	Precision IT baseline	Recall IT	Recall IT baseline
O	0.93	0.99	1.00	0.98
B-PER	0.96	0.88	0.37	0.81
I-PER	0.99	0.79	0.65	0.66
B-ORG	0.84	0.80	0.08	0.70
I-ORG	0.97	0.66	0.27	0.47
B-LOC	0.88	0.84	0.51	0.80
I-LOC	0.94	0.73	0.29	0.39
B-MISC	0.65	0.12	0.04	0.58
I-MISC	0.93	0.64	0.16	0.31

EN :

	Precision EN	Precision EN baseline	Recall EN	Recall EN baseline
O	0.91	0.99	1.00	0.98
B-PER	0.91	0.79	0.22	0.72
I-PER	0.97	0.70	0.40	0.51
B-ORG	0.80	0.69	0.05	0.50
I-ORG	0.95	0.63	0.30	0.56
B-LOC	0.77	0.72	0.37	0.68
I-LOC	0.84	0.69	0.42	0.58
B-MISC	0.83	0.24	0.06	0.67
I-MISC	0.99	0.68	0.15	0.34

Considerazioni finali

Dai dati sulle valutazioni possiamo constatare che usare le HMM non ha portato un sostanziale beneficio rispetto all’utilizzo di un algoritmo naive di NER tagging. Questo

perchè l’HMM è più adatto per problemi come quello del PoS tagging in cui ci sono sequenze di tag molto più varie e che sono maggiormente collegate semanticamente dal loro ordine rispetto al NER tagging in cui inoltre notiamo uno sbilanciamento molto accentuato sul tag O (OTHER).
L’accuracy della baseline risulta maggiore di quella del HMM per il problema appena descritto.

Dalle tabelle di valutazione notiamo che l’ HMM, rispetto alla baseline, si comporta meglio per quanto riguarda la *Precision.*

L’HMM spesso non riesco a trovare un tag adatto per parole non presenti nel training set, soprattutto se precedute da un NER tag == O , in quanto essendo quello più presente (+ del 90%) va spesso a soppiantere le probabilità di altri tag. Diverso è il caso invece se la parola sconosciuta si trova dopo un tag di inizio (B-) in cui l’algoritmo si comporta meglio. Per migliorare questo problema, potrebbe essere utilie utilizzare un training set contenente più dati da cui apprendere.

In questo progetto l’utilizzo della programmazione dinamica con l’alogritmo di Viterbi ci permette di passare da un tempo di esecuzione esponenziale $O(T^n)$ a uno $O(n * T^2)$ che è molto più efficiente.