RELAZIONE "LA MAGIA DEL NER NASCOSTA"

Nelle seguenti pagine verrà illustrata la relazione sull'esercizio **"la magia NER nascosta"**, relativamente agli aspetti e alle scelte implementativi, alle metriche di valutazione e ai risultati dei test. Come quadro riassuntivo, l'esercitazione "la magia NER nascosta" ha avuto come obiettivo:

1) Implementazione di un Named Entity Reconition(NER) tagger
2) Definizione di un criterio semplicistico (Baseline) per confrontare le prestazioni del sistema implementato
3) Testing del sistema su un corpus annotato
4) Analisi dei risultati

Il NER tagger

Il **Named Entity Reconition tagging** è un task della linguistica computazionale generale, affrontato a livello morfologico che trae ispirazione dal Part Of Speech tagging per condurre, ad un livello che si colloca sul piano semantico, un processo di disambiguazione semantico-lessicale su un insieme di frasi in linguaggio naturale.
Il NER tagging si può vedere come un task di sequence labelling, secondo il quale, dato in input una sequenza di parole e un insieme di possibili etichette semantico-lessicale, denominate di qui in poi **"entità NER"**, un disambiguatore interviene per restituire in output la sequenza di entità più probabile da associare alle parole di input.
Differentemente dal POS tagging, il NER tagging ha l'obiettivo di individuare ed etichettare sequenze di token, anziché singoli token, come appartenenti ad un'unica entità semantica. Il ruolo del NER tagging si colloca ad un piano superiore rispetto al POS tagging, che interviene a livello morfologico. Le informazioni restituite dal NER sono utilizzate per risolvere altri task linguistici, primo fra tutti la **sentiment analisys**. Sapere che all'interno di una frase, la parola "Torino" occorre come organizzazione, perché parte di "Torino Football Club", anziché come luogo geografico, ha una valenza migliore rispetto a sapere che Torino è un sostantivo.
Un NER tagger esegue 2 operazioni fondamentali:

- Tokenizzazione
- Disambiguazione dei token

La tokenizzazione è un'operazione di pre-processing che consente di segmentare le parole che compongono le frasi date in input al NER tagger in un insieme di token. A livello di POS tagging, ogni token è una sequenza di caratteri separata da simboli di punteggiatura e da caratteri vuoti. Nel Named Entity Reconition, i token sono da considerarsi unità più grandi rispetto a quelle del POS tagging. L'individuazione dei giusti token è un'operazione complessa che si può affrontare con la tecnica dei BIO tag (Begin, Internal, Other), secondo la quale, le parole che formano le entità vengono segmentate in token di lunghezza 1, e per ciascuna di esse si associa una etichetta che

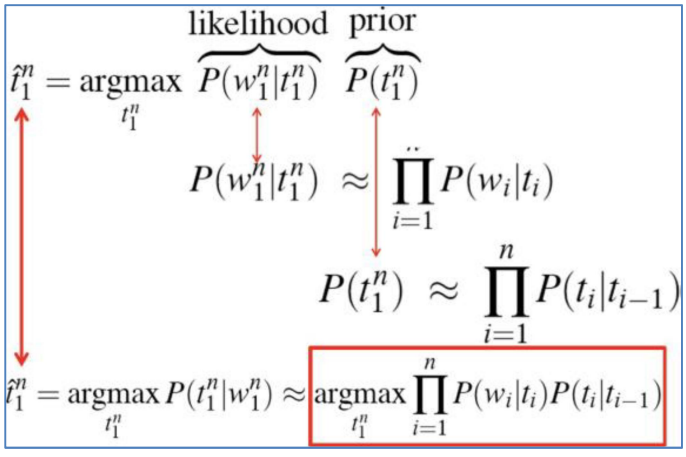
simbolicamente individua la posizione della parola all'interno della sequenza. Le entità vengono organizzate in Begin-tag, Internal-tag e Other. In questo modo il problema del NER si può affrontare con le stesse tecniche presenti in letteratura per il POS Tagging, avendo ridotto la complessità delle entità a singoli token.
Nel caso di studio, abbiamo affrontato il NER tagging avendo un corpus di frasi già etichettate con la tecnica BIO tag. Nel corpus sono presenti 4 tipi entità:

- LOC: Luogo geografico
- ORG: Organizzazione (società, istituzioni, altro)
- PER: Persona
- MISC: Miscellanea

La suddivisione in BIO tag delle precedenti 4 entità ha portato alla definizione del seguente tagset: **[B-LOC, I-LOC, B-ORG, I-ORG, B-PER, I-PER, B-MISC, I-MISC, O].** Il tag "O", (Other) individua tutto ciò che all'interno di una frase non è una fra le 4 entità, inoltre risponde anche alla funzione di separatore tra le varie entità.
La disambiguazione è l'operazione successiva, nella quale si individuano dei criteri per rimuovere le ambiguità sulle varie categorie di entità che ciascun token può ricevere.
Il NER tagging si può affrontare secondo 2 approcci:

1) Approccio basato su regole
2) Approccio stocastico

Nel caso di studio si è seguito un approccio stocastico, affrontabile nel contesto degli algoritmi di machine learning. La caratterizzazione dell'algoritmo usato segue la prassi standard, in letteratura, nella definizione delle 3 fasi di modeling, learning e decoding.
Nel modeling si cerca di fornire un modello, inteso come una descrizione formale del problema. L'intero processo di NER tagging è individuabile come un fenomeno probabilistico e la ricerca della sequenza di NER tag più adeguata, data una frase in input, diventa la ricerca della sequenza più probabile. Per tale ragione si è adottato un modello probabilistico discriminativo. Applicando la regola di Bayes la probabilità a posteriori è stata riscritta come prodotto di 2 probabilità più semplici: probabilità di verosimiglianza e probabilità a priori. La prima risponde alla domanda, qual è la probabilità di osservare una parola, nella sequenza, etichettata con un certo NER tag?. La seconda, invece, risponde alla domanda: qual è la probabilità di vedere un tag in una certa posizione della sequenza?



Come approssimazione alla complessità del problema, il fenomeno probabilistico è stato ricondotto ad un processo di tipo markoviano con memoria 1. Sfruttando l'ipotesi di Markov sulle transizioni e sulle emissioni è stato possibile riscrivere le 2 probabilità in termini di produttorie di probabilità elementari, calcolate sulle singole parole.

L'evoluzione del processo probabilistico è tracciabile definendo un automa a stati finiti, in cui le variabili nascoste, che sono proprio i NER tag si comportano come gli stati di sistema, le probabilità a priori diventano probabilità di transizione da stato a stato e la verosimiglianza si traduce in termini di probabilità di emissione da uno stato ad un suo stato terminale.

Nel learning si individua, dato un corpus di frasi con funzione di training set, un criterio per impostare i parametri che caratterizzano l'equazione del modello. L'apprendimento dei parametri è basato su dei conteggi. Le due probabilità vengono calcolate in maniera indipendente vedendo il corpus. In particolare, la probabilità di transizione **P(Tag1 | Tag2)** si ricava andando a contare, per ogni frase, il numero delle volte in cui il Tag2 precede il Tag1 nella sequenza di tag e dividendo tale risultato per il numero totale di occorrenze del Tag2 all'interno del corpus. Si ottiene una frequenza relativa, ovvero un valore compreso tra 0 e 1.
La probabilità di emissione **P(Token | Tag)** si ricava contando nel corpus quante volte un certo token è etichettata con un certo tag dividendo questo risultato per il numero di volte in cui il tag compare nel corpus. Anche in questo caso il valore è compreso tra 0 e 1.

Nel caso di studio, durante la fase di learning sono stati utilizzati 2 training set, uno in lingua italiana, l'altro in lingua inglese, recuperabili agli indirizzi:

- https://github.com/Babelscape/wikineural/tree/master/data/wikineural/en
- https://github.com/Babelscape/wikineural/tree/master/data/wikineural/it

Nei link sono presenti, per ogni lingua, 3 file in formato CONLL:

- **Training set:** 88400 frasi per l'italiano, 92720 per l'inglese
- **Testing set:** 11069 frasi per l'italiano, 11597 per l'inglese
- **Validation set:** 11050 frasi per l'italiano, 11590 per l'inglese

CoNLL è il nome convenzionale per i formati TSV(Tab separated Values) per il natural language processing. I formati CoNLL differiscono nella scelta e nell'ordine delle colonne. Tuttavia, in ogni formato vengono rispettate una serie di regole, in particolare:

1. ogni parola (token) è rappresentata in una riga;
2. le frasi sono separate una dall'altra per mezzo di una riga vuota;
3. ogni colonna rappresenta un'annotazione;
4. ogni parola in una frase ha lo stesso numero di colonne.
5. ogni parola nella frase è enumerata progressivamente a partire da 0

Nel caso specifico si è utilizzato per il training e il testing un file CONLL a 3 colonne con le indicazioni della posizione della parola all'interno della frase, la parola e il NER tag associato.
Per organizzare in modo compatto le fasi di learning e di decoding del sistema, è stata definita la classe **NERtagger** che fornisce le definizioni di alcuni metodi pubblici usati per recuperare il tagset, per eseguire il decoding su un testing set e su singole frasi e per analizzare i risultati delle predizioni.

L'apprendimento delle probabilità di transizione e di emissione dal corpus è portato avanti dal metodo privato **"learning()",** invocato all'atto della creazione dell'instanza della classe. Il metodo riceve in ingresso il path ad un training set in formato CoNLL (file testuale .txt), quindi esegue l'apprendimento recuperando dal corpus il tagset e le frequenze relative per il calcolo delle probabilità di transizione e di emissione. Per questioni di ottimizzazione, in una prima lettura, il metodo analizza l'intero corpus recuperando quanti più dati possibile relativamente all'indicazione sul numero di occorrenze per tag, il numero di occorrenze iniziali e finali per tag e il numero di frasi presenti nel corpus. Infine, esegue le chiamate ai metodi privati **transition_probabilities_matrix(), initial_probability() , end_probability(),** responsabili di calcolare, rispettivamente, la matrice delle probabilità di transizione di Markov, la distribuzione di probabilità di partenza per il sistema di Markov e le probabilità finali per l'algoritmo di Viterbi, discusso in seguito.

La matrice delle probabilità di transizione è una matrice quadrata di dimensione N x N, con N pari al numero di NER tag presenti nel tagset. Preso un tag, ad esempio B-MISC, ogni cella nella matrice associata alla riga di B-MISC contiene l'informazione sulla probabilità **P(B-MISC | TAG),** per ogni NER TAG nel tagset. Alcune transizioni possono essere a 0, in quanto nel dataset non tutte sono definite. In particolare, probabilità come P(I-MISC | I-PER) restituiscono inevitabilmente valori a 0, in quanto la sequenza **[I-PER, I-MISC]** non esiste e segna intuitivamente una frase inconsistente. Le probabilità a 0 sono state "smussate" con valori infinitesimali. Lo scopo dello smoothing è quello di premiare con una quantità molto piccola le probabilità che sono a 0, evitando, in fase di decoding, di annichilire i risultati parziali, portando a premiare strade meno promettenti. Una variabile di istanza "**__transition_smoothing**" regola il valore di smoothing da applicare per gestire questi casi che possono occorrere nel calcolo di tutte le probabilità di transizione. Il calcolo delle probabilità di partenza è eseguito dal metodo "initial_probability" che si avvale dei conteggi eseguiti in fase di learning per calcolare le probabilità di transizione **P(TAG | START)**, per ogni tag presente nel tagset. Tale probabilità viene calcolata contando il numero di frasi che iniziano con un certo NER tag e dividendo tale risultato per il numero totale di frasi nel corpus. Similmente, le probabilità finali **P(END | TAG),** definite dal metodo "final_probability", vengono calcolate come il rapporto tra il numero di frasi che terminano con un certo TAG e il numero di occorrenze del tag considerato. A termine dell'esecuzione del metodo "learning", la maggior parte dei dati, utili in fase di decoding, sono stati appresi.

Nel decoding si cerca di capire qual è l'algoritmo che permette di applicare il modello appreso per recuperare la soluzione ottimale. Nel nostro caso, la soluzione ottimale consiste nella sequenza di NER tag più probabile, data una sequenza di token in input. Sfruttando l'ipotesi di Markov sulla memoria 1 e usando la programmazione dinamica possiamo evitare di calcolare tutte le possibili sequenze di tag per poi scegliere quella con probabilità massima. In questo modo la probabilità di una sequenza lunga "j elementi" si può suddividere in 2 parti:

1. la probabilità della sequenza migliore da 0 a j-1
2. la probabilità di transizione da j-1 a j

Quindi è possibile calcolare la probabilità di una sequenza come il prodotto della probabilità di transizione per la probabilità di emissione per la probabilità della sequenza migliore calcolata al passo precedente. Il criterio di decoding impiegato è l'algoritmo di **viterbi** che sfrutta la programmazione dinamica per suddividere il calcolo della sequenza più probabile in una maniera tale da non ripetere gli stessi passi più volte. Man mano che si analizzano le parole in input si va a memorizzare solo la sotto-sequenza di tag con probabilità maggiore. Questo consente di ridurre la complessità temporale poiché si evita di affrontare il calcolo di tutte le possibili sequenze, pagando un prezzo, però, in termini di memoria. La complessità è lineare nel numero delle parole e quadratica rispetto al numero di tag.

La classe NERtagger fornisce la definizione metodo "viterbi" che implementa il criterio di decoding appena discusso. La funzione prende in input 2 valori:

1. una sequenza di parole da etichettare
2. una strategia di smoothing per le probabilità di emissione sulle parole sconosciute

All'ingresso del metodo vengono eseguiti dei controlli di conformità sui parametri di input, in seguito viene invocata la funzione di utilità **"all_emission_probability"** alla quale vengono passati i 2 precedenti argomenti. La funzione calcola tutte le probabilità di emissione P(Parola | TAG) a partire dai valori ricavati dal corpus. In questa fase viene eseguita una seconda lettura del training set. Un dizionario python, **"__emissions**" ricorda, per ogni parola presente nel corpus, una lista della dimensione del tagset, nella quale ogni posizione viene associata ad un NER tag e ciascuna di esse ricorda il numero di coppie "parola-tag" presenti nel corpus. A termine della lettura si itera sulla struttura e si ricavano, per ogni parola, le probabilità di emissione, dividendo i precedenti risultati per il numero totale di occorrenze dei tag. L'operazione di lettura viene eseguita una sola volta, perciò la prima esecuzione dell'algoritmo di viterbi impiegherà più tempo. Nelle chiamate successive, la funzione si limiterà a confrontare le parole del corpus con quelle della sequenza in input, quindi applicherà una strategia di smoothing a tutte le parole sconosciute per gestire le probabilità a 0. Riguardo a quest'ultimo aspetto sono state definite 4 strategie di smoothing:

1. assegna il 100% di probabilità al tag **"O"**;
2. assegna il 50% di probabilità ai tag **"B-MISC" e "O"**;
3. assegna una distribuzione di probabilità uniforme a tutti i tag, calcolata come rapporto **1/numero di NER tag**;
4. Assume che le parole sconosciute abbiano una distribuzione di probabilità simile alle parole che occorrono una sola volta in un **validation set.** Conta le frequenze di ogni tag su tali parole e divide la frequenza per il numero di parole trovate.

L'algoritmo di viterbi esegue in 3 fasi: inizializzazione, iterazione, terminazione. Durante la prima fase, viene creata una matrice di dimensione N x M, **viterbi_matrix**, con N pari al numero di NER tag e M pari al numero di parole della sequenza data in input. All'atto dell'inizializzazione, la matrice conterrà nella prima colonna, per ogni TAG, il prodotto tra la probabilità di partenza P(TAG | START) per la probabilità di emissione sulla prima parola, **P(Parola1| TAG)**. Una seconda matrice, "**backpointer**", di dimensione N x M viene inizializzata a 0. Questa struttura servirà per ricordare,

ad ogni passo, l'indice della sotto-sequenza nella colonna precedente che ha generato la probabilità più alta e verrà usata per ricostruire all'indietro la sequenza di tag più probabile per la frase in input. Al passo iterativo vengono riempite le colonne della matrice, dalla seconda fino all'ultima. L'idea è mettere una sorta di finestra che va a riempire una parte della matrice e, considerando solo 2 colonne per volta, riempie la colonna di destra in base ai valori della colonna di sinistra, partendo da uno stato iniziale arrivando ad uno stato finale. All'interno della matrice si inseriscono dei valori chiamati di Viterbi, che ricordano la probabilità della sotto-sequenza trovata sino a quel punto. Segue un estratto dal codice dell'algoritmo di viterbi.

```
#iteration step
for o in range(1, len(observations)):
    for s in range(len(self.__tag_set)):
        prec = viterbi_matrix[:, o-1]
        trans = self.__transitions[self.__tag_set[s]]
        em = self.__emissions[observations[o]][s]
        product = [prec[i] * trans[i] * em for i in range(len(prec))]
        maxIndex = numpy.argmax(product)
        viterbi_matrix[s,o] = viterbi_matrix[maxIndex,o-1] * self.__transitions[self.__tag_set[s]][maxIndex] * em
        backpointer[s, o] = maxIndex
```

Per ogni parola, dalla seconda in poi, viene calcolato il valore di Viterbi come il massimo del valore di Viterbi della parola precedente (viterbi_matrix[maxIndex,o-1]) per la probabilità di transizione (self.__transitions[self.__tag_set[s]][maxIndex]) tra il "tag s" e il tag in posizione "maxIndex" moltiplicato per la probabilità di emissione P(parola(O)|TAG(S)). Il backpointer accumula, per ogni stato, l'indice della sotto-sequenza più probabile. Infine c'è una fase di terminazione che indica la transizione tra uno dei possibili stati rappresentati nell'ultima colonna e lo stato finale. Si ricava il massimo tra i valori di Viterbi dell'ultima colonna moltiplicati per le probabilità di transizione (__final_probability) verso lo stato finale. Il backpointer viene percorso all'indietro dallo stato finale fino allo stato iniziale nella matrice di Viterbi, ricavando la sequenza di tag più probabile.

La Baseline

Per avere un metodo di confronto delle prestazioni del sistema implementato, solitamente si fa riferimento ad un algoritmo di tagging che implementa un criterio di predizione semplicistico. In letteratura questo sistema prende nome di Baseline. Nel caso di studio la Baseline assegna per ogni parola della sequenza data in input, il tag più frequente, se la parola è presente nel corpus, altrimenti assegna il tag "B-MISC". Il criterio semplicistico alla base di questo sistema permette di raggiungere un'accuratezza del 93% sul testing set italiano. Nella classe NERtagger è presente il metodo pubblico "baseline" che implementa tale sistema. La funzione riceve 2 argomenti:

1. Una sequenza di parole da etichettare
2. Una strategia di smoothing per gestire le parole sconosciute.

Infatti, per eseguire più confronti si è data la possibilità all'utente di scegliere il NER tag da assegnare come smoothing delle parole sconosciute. Nella parte finale di questa relazione sono riportati i risultati dei test condotti con le varie strategie.

Testing del sistema su un corpus annotato

La classe NERtagger offre la definizione di due metodi pubblici usati per fare il testing dei sistemi implementati(HMM, Baseline) su un test set. Il metodo **"testing_viterbi"** esegue il test del NER tagger tramite l'algoritmo di decoding discusso in precedenza. Riceve in input 3 argomenti:

1. **test_set_path**: il percorso al file annotato in formato CoNLL con funzione di test set
2. **smoothing**: la strategia di smoothing per le parole sconosciute. Il parametro riceve una tra le seguenti stringhe:
   a. "smoothing1", "smoothing2", "smoothing3" per le prime tre tecniche di smoothing;
   b. il percorso ad un validation set (file testuale semplice .txt) per la quarta strategia.
3. **number_sentence**: [Opzionale] il numero di frasi del test set da analizzare, se non viene specificato l'algoritmo analizza l'intero dataset.

La funzione legge le prime "number_sentence" frasi estratte dal test set, le quali vengono organizzate in liste di coppie "parola-tag", per facilitare i confronti durante la fase di valutazione delle prestazioni, successivamente invoca il metodo "viterbi" passandogli una sequenza di parole alla volta e la strategia di smoothing per le parole sconosciute. Il comportamento dell'altro metodo di testing, **"testing_baseline",** è analogo al precedente, con la differenza che il decoding della migliore sequenza viene affrontato con il criterio semplicistico implementato dalla Baseline. A termine dell'esecuzione del decoding, la sequenza di tag restituita viene confrontata con quella corretta. Nel seguito della relazione queste due sequenze verranno chiamate rispettivamente system e gold. Una funzione di utilità **"__evaluation_measures"** calcola una serie di metriche di valutazione sulle singole entità individuate dal sistema. Le misure considerate per la valutazione delle performance del sistema sono le seguenti:

1. accuratezza generale
2. accuratezza per entità
3. precision per entità
4. recall per entità
5. F1-score per entità

La prima metrica misura, a partire dalle due sequenze, gold e system, la proporzione dei singoli tag individuati dal sistema sul totale dei tag presenti nella sequenza. Si itera sul numero di elementi nella system, quindi si fa un confronto per posizione con l'elemento nella sequenza gold (le due liste hanno sempre la stessa lunghezza). Il risultato del rapporto è un numero compreso tra 0 e 1. Il calcolo dell'accuratezza, così come per le altre metriche, viene ripetuto per ogni frase analizzata dal sistema. Per tutte le metriche di valutazione, i risultati parziali vengono accumulati in due dizionari python, **"__accuracy"** e **"__metrics**". A termine del testing l'accuratezza generale viene divisa per il numero di frasi analizzate ottenendo, in media, una misura di quanto è bravo il sistema ad individuare i singoli tag. Tuttavia, l'accuratezza generale per il NER è una misura poco informativa, in quanto sarebbe più appropriato disporre di una misura della bontà della predizione estesa alle singole entità. La funzione **"__metrics_per_tag_entity"** affronta il calcolo dell'accuratezza relativa e delle altre misure di precision, recall e F1-score. Riceve in input le

sequenze gold e system e i tag BIO associati alle diverse entità. Per quanto riguarda l'accuratezza relativa, il calcolo affrontato non è dissimile da quello visto in precedenza. Nello specifico, date le due sequenze, si ricava il numero di tag BIO per ogni entità presenti nella gold, poi si attraversa la sequenza system verificando posizione per posizione con la precedente se un certo tag begin o internal è presente. I risultati parziali dei conteggi vengono accumulati nella struttura **"__accuracy"**, al cui interno sono presenti 3 tipi di chiavi:

- **GENERAL**: gestisce l'accuratezza generale
- **Il nome dell'entità**: gestisce l'accuratezza relativa
- **O**: gestisce l'accuratezza espressa sul tag O

Per ciascuna chiave viene associato un dizionario che, per le entità e per il tag O, ospita 3 ulteriori campi: TOT, COR, ACCURACY. Nei campi TOT e COR si ricorda rispettivamente il numero totale di tag BIO per una certa entità presenti in tutte le sequenze gold analizzate e il numero di quelli individuati correttamente dal sistema. A termine del testing viene riempito il campo "ACCURACY" con il rapporto COR / TOT.

Il calcolo delle altre metriche procede per passi:

- eseguendo una lettura sulle due sequenze di tag, gold e system, si individuano tutte le entità di ogni tipo in esse presenti ricavando 2 insiemi di entità. Questi possono differire per diversi motivi, ad esempio a causa di predizioni incorrette da parte del sistema.
- In seguito si contano il numero di entità predette correttamente dal sistema (veri positivi), il numero di entità mancate (falsi negativi) e il numero di entità ipotizzate come corrette (falsi positivi), eseguendo opportuni confronti fra i due insiemi di entità.

I risultati ottenuti vengono accumulati nella seconda struttura, **"__metrics"**, organizzata in modo simile a quella usata per gestire le accuratezze. I valori ricordati sono ottenuti analizzando tutte le frasi presenti nel test set. Per ogni entità viene salvato un dizionario formato da 6 campi:

- **COR**: ricorda il numero totale di entità individuate correttamente dal sistema
- **MIS**: ricorda il numero totale di entità classificate come "O" dal sistema
- **HYP**: ricorda il numero totale di entità predette
- **PRECISION**: metrica di valutazione calcolata come rapporto "**COR / (COR + HYP)"**
- **RECALL**: metrica di valutazione calcolata come rapporto **"COR / (COR + MIS)"**
- **F1SCORE**: media armonica tra le due precedenti misure, (2 * PRECISION * RECALL) / (PRECISION + RECALL)

La precision misura la percentuale di entità NER trovate dal sistema, nell'insieme di entità predette, che sono corrette, mestre la Recall misura la percentuale di entità NER presenti nel corpus che vengono predette correttamente dal sistema. Nelle seguenti pagine verranno illustrati i risultati dei test eseguiti con i due sistemi implementati, considerando le varie tecniche di smoothing sulle parole sconosciute.

Analisi dei risultati

Testing del sistema HMM con decoding viterbi sul test set italiano.

- Il training set Italiano contiene 88400 frasi. Il learning impiega in media 8 secondi.
- Il decoding tramite algoritmo di viterbi completa l'esecuzione in circa 50 secondi, in media, per ogni strategia di smoothing

Strategia di smoothing: **smoothing1**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1748	481	345	78.558%	83.516%	78.421%	80.888%
PER	5934	2443	1565	80.745%	79.131%	70.837%	74.755%
MISC	1206	1136	663	61.255%	64.526%	51.494%	57.278%
LOC	7793	1995	1237	80.744%	86.301%	79.618%	82.825%
O				99.681%			
General				96.744%			

Strategia di smoothing: **smoothing2**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1748	481	341	78.528%	83.676%	78.421%	80.963%
PER	5934	2443	1555	80.706%	79.236%	70.837%	74.801%
MISC	1224	1118	788	62.084%	60.835%	52.263%	56.224%
LOC	7794	1994	1236	80.736%	86.312%	79.628%	82.835%
O				99.664%			
General				96.743%			

Strategia di smoothing: **smoothing3**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1766	463	338	80.116%	83.935%	79.228%	81.514%
PER	6851	1526	754	87.447%	90.085%	81.783%	85.733%
MISC	1257	1085	641	63.247%	66.228%	53.672%	59.293%
LOC	7801	1987	1238	80.923%	86.304%	79.700%	82.871%
O				99.645%			
General				97.164%			

Strategia di smoothing: **validation_set_ITA.txt**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1760	469	344	79.505%	83.650%	78.959%	81.237%
PER	5989	2388	1567	81.372%	79.262%	71.493%	75.177%
MISC	1226	1116	662	62.031%	64.936%	52.348%	57.966%
LOC	7802	1986	1247	80.877%	86.219%	79.710%	82.837%
O				99.673%			
General				96.802%			

Testing del sistema Baselise sul test set italiano

- Smoothing1: se la parola è sconosciuta etichettala come B-MISC
- smoothing2: se la parola è sconosciuta etichettala come O
- Il training set Italiano contiene 88400 frasi. Il learning impiega in media 8 secondi.
- Il decoding completa l'esecuzione in circa 7 secondi, in media.

Strategia di smoothing1: **assegna il tag B-MISC**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1361	868	542	61.759%	71.519%	61.059%	65.876%
PER	5011	3366	2656	73.673%	65.358%	59.819%	62.466%
MISC	897	1445	10192	42.147%	8.089%	38.301%	13.357%
LOC	7045	2743	2286	70.332%	75.501%	71.976%	73.696%
O				97.819%			
General				93.801%			

Strategia di smoothing2: **assegna il tag O**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1361	868	542	61.759%	71.519%	61.059%	65.876%
PER	5011	3366	2656	73.673%	65.358%	59.819%	62.466%
MISC	661	1681	849	36.718%	43.775%	28.224%	34.320%
LOC	7045	2743	2286	70.332%	75.501%	71.976%	73.696%
O				99.794%			
General				95.240%			

Dall'analisi dei risultati immediatamente ci si accorge che la terza strategia di smoothing restituisce i valori più alti, non solo in termini di accuratezza generale ma anche rispetto alle altre metriche di valutazione. Le prime due strategie di smoothing restituiscono valori molto simili, con leggerissime flessioni del valore di accuratezza che premiano la prima. Guardando con più attenzione, la strategia 1 individua più entità MISC, ed è questo il principale valore di discrimine fra le 2 tecniche. Infatti la precision per l'entità MISC ci dice che smoothing 1 commette meno errori di predizione (HYP) su questa entità portando il denominatore della frazione a valori bassi. Un altro confronto significativo è tra le altre 2 strategie di smoothing che riportano valori simili fra loro. In particolare, si nota come smoothing3 è più brava a individuare le entità PER e MISC, infatti guardando i risultati dei conteggi parziali si nota che la quarta strategia commette più errori di classificazione (falsi negativi) sulle entità PER, abbassando il valore della recall di 10 punti percentuali rispetto alla prima. Per quanto riguarda la Baseline, il primo criterio di smoothing porta alle prestazioni più basse fra tutti i test eseguiti. Tuttavia restituisce informazioni significative sulla difficoltà, in generale, nell'individuazione dell'entità MISC che qui raggiunge una precisione di appena 8%. Questo risultato così scarso è dato dall'eccessivo numero di parole sconosciute presenti nel test set, infatti il sistema riconosce 10192 entità come MISC, tuttavia solo 897 fra tutte quelle presenti nel test set sono individuate correttamente. Andando ad eseguire uno smoothing con il tag "O" le cose cambiano molto. Dato che la predizione di questo tag riporta i più alti valori di accuratezza, in generale, questo test ha restituito un'accuratezza generale vicino alla seconda tecnica di smoothing usata dal NER tagger, tuttavia i valori di precision e recall fra i due sistemi sono molto diversi. Mettendo a confronto le 2 baseline notiamo che l'accuratezza non è informativa sulla capacità di predizione del sistema, perché è facile osservare che rispetto all'entità MISC la seconda restituisce un valore di accuratezza minore, sebbene la precisione indica l'esatto opposto. Dovendo classificare le varie tecniche sul dataset italiano, la migliore è smoothing3, segue smoothing4 sul validation set, smoothing1, smoothing2, la baseline con tag O e infine la baseline con tag B-MISC.

Testing del sistema HMM con decoding viterbi sul test set inglese.

- Il training set inglese contiene 92720 frasi. Il learning impiega in media 8 secondi.
- Il decoding completa l'esecuzione su un test set di 11597 frasi in circa 50 secondi in media, per ogni strategia di smoothing.

Strategia di smoothing: **smoothing1**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	2237	1212	774	70.499%	74.294%	64.859%	69.257%
PER	2911	2296	1520	68.863%	65.696%	55.906%	60.407%
MISC	2578	1927	1429	62.253%	64.337%	57.225%	60.573%
LOC	4265	1690	1172	72.898%	78.444%	71.620%	74.877%
O				99.566%			
General				95.405%			

Strategia di smoothing: **smoothing2**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	2237	1212	770	70.499%	74.393%	64.859%	69.300%
PER	2911	2296	1514	68.830%	65.785%	55.906%	60.445%
MISC	2603	1902	1568	62.870%	62.407%	57.780%	60.004%
LOC	4265	1690	1170	72.849%	78.473%	71.620%	74.890%
O				99.564%			
General				95.422%			

Strategia di smoothing: **smoothing3**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	2291	1158	731	72.657%	75.811%	66.425%	70.808%
PER	3809	1398	731	80.079%	83.899%	73.152%	78.158%
MISC	2623	1882	1388	63.944%	65.395%	58.224%	61.602%
LOC	4365	1590	1082	75.543%	80.136%	73.300%	76.566%
O				99.540%			
General				96.022%			

Strategia di smoothing4: **validation_set_ENG.txt**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	2267	1182	755	71.603%	75.017%	65.729%	70.067%
PER	3011	2196	1500	70.511%	66.748%	57.826%	61.968%
MISC	2585	1920	1400	62.689%	64.868%	57.381%	60.895%
LOC	4339	1616	1133	74.694%	79.295%	72.863%	75.943%
O				99.564%			
General				95.572%			

Strategia di smoothing1: **assegna il tag B-MISC**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1465	1984	993	51.952%	59.601%	42.476 %	49.602%
PER	2331	2876	2254	61.957%	50.840%	44.767 %	47.611%
MISC	2587	1918	9770	49.862%	20.936%	57.425 %	30.685%
LOC	3769	2186	2029	66.538%	65.005%	63.291 %	64.137%
O				98.055%			
General				92.892%			

Strategia di smoothing1: **assegna il tag O**

NER Tag	COR	MIS	HYP	Accuratezza	Precision	Recall	F1Score
ORG	1465	1984	993	51.952%	59.601%	42.476 %	49.602%
PER	2331	2876	2254	61.957%	50.840%	44.767 %	47.611%
MISC	2017	2488	1724	42.927%	53.916%	44.772 %	48.920%
LOC	3769	2186	2029	66.538%	65.005%	63.291 %	64.137%
O				98.055%			
General				93.945%			

Il testing sul dataset inglese ha restituito dei risultati quasi in proporzione a quelli ottenuti sul dataset italiano, con un leggero cambiamento rispetto alla strategia di smoothing 2 che in questo caso restituisce valori migliori rispetto alla prima. Valgono, dunque, le stesse considerazioni fatte sul dataset italiano.

Test delle frasi su Harry Potter:

In seguito, come da traccia d'esercizio, verranno fatte alcune considerazioni sui risultati dei test eseguiti su 3 frasi che riguardano Harry Potter. Le frasi sono riportate in italiano, quindi il learning è stato eseguito sul corpus italiano. La prima considerazione da fare è che le uniche parole sconosciute sono: "Hogwards, Grunnings, fabbricava, trapani", infatti la baseline ha risposto B-MISC su queste parole. In secondo luogo è da riportare che le frasi non erano etichettate. Si possono fare alcune congetture:

- Harry Potter dovrebbe essere PER
- Hogwards dovrebbe essere LOC
- Diagon Alley dovrebbe essere LOC
- Mr Dursley dovrebbe essere PER
- Grunning dovrebbe essere ORG

Testing del sistema HMM con decoding viterbi con smoothing1

[('La', 'O'), ('vera', 'O'), ('casa', 'O'), ('di', 'O'), ('Harry', 'B-MISC'), ('Potter', 'I-MISC'), ('è', 'O'), ('il', 'O'), ('castello', 'O'), ('di', 'O'), ('Hogwards', 'O')]

[('Harry', 'B-PER'), ('le', 'O'), ('raccontò', 'O'), ('del', 'O'), ('loro', 'O'), ('incontro', 'O'), ('a', 'O'), ('Diagon', 'O'), ('Alley', 'I-PER')]

[('Mr', 'B-MISC'), ('Dursley', 'O'), ('era', 'O'), ('direttore', 'O'), ('di', 'O'), ('una', 'O'), ('ditta', 'O'), ('di', 'O'), ('nome', 'O'), ('Grunnings', 'O'), ('che', 'O'), ('fabbricava', 'O'), ('trapani', 'O')]

Testing del sistema HMM con decoding viterbi con smoothing2

[('La', 'O'), ('vera', 'O'), ('casa', 'O'), ('di', 'O'), ('Harry', 'B-MISC'), ('Potter', 'I-MISC'), ('è', 'O'), ('il', 'O'), ('castello', 'O'), ('di', 'O'), ('Hogwards', 'O')]

[('Harry', 'B-PER'), ('le', 'O'), ('raccontò', 'O'), ('del', 'O'), ('loro', 'O'), ('incontro', 'O'), ('a', 'O'), ('Diagon', 'B-MISC'), ('Alley', 'I-PER')]

[('Mr', 'B-MISC'), ('Dursley', 'O'), ('era', 'O'), ('direttore', 'O'), ('di', 'O'), ('una', 'O'), ('ditta', 'O'), ('di', 'O'), ('nome', 'O'), ('Grunnings', 'O'), ('che', 'O'), ('fabbricava', 'O'), ('trapani', 'O')]

Testing del sistema HMM con decoding viterbi con smoothing3

[('La', 'O'), ('vera', 'O'), ('casa', 'O'), ('di', 'O'), ('Harry', 'B-MISC'), ('Potter', 'I-MISC'), ('è', 'O'), ('il', 'O'), ('castello', 'O'), ('di', 'O'), ('Hogwards', 'O')]

[('Harry', 'B-PER'), ('le', 'O'), ('raccontò', 'O'), ('del', 'O'), ('loro', 'O'), ('incontro', 'O'), ('a', 'O'), ('Diagon', 'B-PER'), ('Alley', 'I-PER')]

[('Mr', 'B-MISC'), ('Dursley', 'I-MISC'), ('era', 'O'), ('direttore', 'O'), ('di', 'O'), ('una', 'O'), ('ditta', 'O'), ('di', 'O'), ('nome', 'O'), ('Grunnings', 'O'), ('che', 'O'), ('fabbricava', 'O'), ('trapani', 'O')]

Testing del sistema HMM con decoding viterbi con smoothing4 sul validation set italiano

[('La', 'O'), ('vera', 'O'), ('casa', 'O'), ('di', 'O'), ('Harry', 'B-MISC'), ('Potter', 'I-MISC'), ('è', 'O'), ('il', 'O'), ('castello', 'O'), ('di', 'O'), ('Hogwards', 'O')]

[('Harry', 'B-PER'), ('le', 'O'), ('raccontò', 'O'), ('del', 'O'), ('loro', 'O'), ('incontro', 'O'), ('a', 'O'), ('Diagon', 'B-PER'), ('Alley', 'I-PER')]

[('Mr', 'B-MISC'), ('Dursley', 'O'), ('era', 'O'), ('direttore', 'O'), ('di', 'O'), ('una', 'O'), ('ditta', 'O'), ('di', 'O'), ('nome', 'O'), ('Grunnings', 'O'), ('che', 'O'), ('fabbricava', 'O'), ('trapani', 'O')]

Testing del sistema Baseline con assegnazione del tag O alle parole sconosciute

[('La', 'O'), ('vera', 'O'), ('casa', 'O'), ('di', 'O'), ('Harry', 'B-PER'), ('Potter', 'I-MISC'), ('è', 'O'), ('il', 'O'), ('castello', 'O'), ('di', 'O'), ('Hogwards', 'O')]

[('Harry', 'B-PER'), ('le', 'O'), ('raccontò', 'O'), ('del', 'O'), ('loro', 'O'), ('incontro', 'O'), ('a', 'O'), ('Diagon', 'O'), ('Alley', 'I-PER')]

[('Mr', 'B-MISC'), ('Dursley', 'O'), ('era', 'O'), ('direttore', 'O'), ('di', 'O'), ('una', 'O'), ('ditta', 'O'), ('di', 'O'), ('nome', 'O'), ('Grunnings', 'O'), ('che', 'O'), ('fabbricava', 'O'), ('trapani', 'O')]

Testing del sistema Baseline con assegnazione del tag B-MISC alle parole sconosciute

[('La', 'O'), ('vera', 'O'), ('casa', 'O'), ('di', 'O'), ('Harry', 'B-PER'), ('Potter', 'I-MISC'), ('è', 'O'), ('il', 'O'), ('castello', 'O'), ('di', 'O'), ('Hogwards', 'B-MISC')]

[('Harry', 'B-PER'), ('le', 'O'), ('raccontò', 'O'), ('del', 'O'), ('loro', 'O'), ('incontro', 'O'), ('a', 'O'), ('Diagon', 'B-MISC'), ('Alley', 'I-PER')]

[('Mr', 'B-MISC'), ('Dursley', 'B-MISC'), ('era', 'O'), ('direttore', 'O'), ('di', 'O'), ('una', 'O'), ('ditta', 'O'), ('di', 'O'), ('nome', 'O'), ('Grunnings', 'B-MISC'), ('che', 'O'), ('fabbricava', 'B-MISC'), ('trapani', 'B-MISC')]

Dai risultati ottenuti si nota come la strategia smoothing 3 con HMM risulta quella più coerente fra tutte. E' interessante notare come "Harry Potter", nella prima frase venga etichettato dal NER tagger sempre allo stesso modo con B-MISC e I-MISC, mentre la Baseline fa uno sforzo in più sulla parola Harry che intuitivamente viene etichettata nel modo giusto con il tag B-PER. Per quanto riguarda la terza frase, in tutti i test viene restituito su Mister il tag B-MISC mentre Dursley alle volte non viene individuato. Stesso discorso per la parola Harry nella seconda frase. Infine, "Grunnings" e "Hogwards" vengono etichettate con B-MISC solo dalla Baseline.

Esempio d'uso: ner_tagger_main

Ad uso dimostrativo, l'utente può eseguire i propri test invocando il file di scripting **"ner_tagger_main.py**". Il programma istanza un oggetto "NERtagger" ricevendo come parametro del costruttore il percorso al training set dichiarato tra gli argomenti di input, definiti all'atto dell'esecuzione.

La sintassi per eseguire correttamente il main è la seguente:
"python ner_tagger_main.py ***-decoding*** *training_set.txt testing_set.txt* ***smoothing_strategy****".*
Segue una breve spiegazione dell'uso dei parametri:

- "**-decoding**": specifica l'algoritmo di decoding. Le scelte sono:
  - "-viterbi" come strategia di decoding per l'HMM,
  - "-baseline" come criterio di confronto semplicistico alle strategie più raffinate
- "Training_set": il path al file testuale in formato CONLL per l'apprendimento del modello
- "Testing_set": il path al file testuale in formato CONLL per il testing del modello
- **"smoothing_strategy":** In accordo all'algoritmo di decoding scelto, le alternative sono:
  - [smoothing1, smoothing2, smoothing3] oppure il percorso ad un validation set, per il decoding tramite algoritmo di viterbi.
  - Uno dei tag presenti nel tag set per il criterio semplicistico "baseline".

In risposta all'esecuzione del codice, il programma restituisce l'indicazione sul tempo impiegato nello svolgimento delle operazioni di learning e decoding, il tagset estratto dall'analisi delle probabilità di emissione e di transizione sul training set e i risultati delle principali metriche di valutazione delle prestazioni.

```
Start reading file:  .\training_set_ENG.txt
Time to read the file:  9.11  seconds
Tagset:  ['O', 'B-LOC', 'I-LOC', 'B-ORG', 'B-MISC', 'I-MISC', 'B-PER', 'I-PER', 'I-ORG']

Starting baseline decoding!
Smoothing strategy: TAG B-MISC
Time for decoding:  6.08  seconds

- Accuracy:
GENERAL {'ACCURACY': 92.892}
ORG {'TOT': 6071, 'COR': 3154, 'ACCURACY': 51.952}
PER {'TOT': 9166, 'COR': 5679, 'ACCURACY': 61.957}
MISC {'TOT': 9402, 'COR': 4688, 'ACCURACY': 49.862}
LOC {'TOT': 8018, 'COR': 5335, 'ACCURACY': 66.538}
O {'TOT': 234498, 'COR': 229936, 'ACCURACY': 98.055}

- Precision and recall per entity:
ORG {'COR': 1465, 'MIS': 1984, 'HYP': 993, 'PRECISION': 59.601, 'RECALL': 42.476, 'F1SCORE': 49.602}
PER {'COR': 2331, 'MIS': 2876, 'HYP': 2254, 'PRECISION': 50.84, 'RECALL': 44.767, 'F1SCORE': 47.611}
MISC {'COR': 2587, 'MIS': 1918, 'HYP': 9770, 'PRECISION': 20.936, 'RECALL': 57.425, 'F1SCORE': 30.685}
LOC {'COR': 3769, 'MIS': 2186, 'HYP': 2029, 'PRECISION': 65.005, 'RECALL': 63.291, 'F1SCORE': 64.137}
```

Esempio di esecuzione di "ner_tagger_main.py".