NER Multilingua

Introduzione

La Named Entity Recognition (NER) tagging multilingua è un campo di studio del Natural Language Processing (NLP) che consiste nell' l'individuazione e l'etichettatura di entità come nomi di persone, luoghi e organizzazioni all'interno di un testo.
La NER è una tecnica fondamentale nel NLP in quanto consente l'estrazione di informazioni strutturate da testi non strutturati. Ad esempio, in un articolo di giornale, la NER può identificare i nomi delle persone coinvolte, le organizzazioni menzionate e i luoghi citati, facilitando la creazione di un database di informazioni utilizzabile per varie applicazioni, dall'analisi dei trend alla costruzione di grafi di conoscenza.

Applicazioni pratiche del NER

Le applicazioni della NER sono diverse e ampie, includono:

- **Estrazione di informazioni**: Automatizzazione della raccolta di dati da grandi quantità di testo.
- **Analisi del sentiment**: Identificazione di entità per analizzare il sentimento associato al testo che le contiene.
- **Raccomandazioni personalizzate**: Utilizzo delle entità riconosciute per migliorare i sistemi di raccomandazione.
- **Assistenza virtuale:** Integrare la NER nei sistemi di assistenza virtuale consente di migliorare la comprensione del contesto e delle intenzioni dell'utente.
- **Anonimizzazione**: Oscuramento o eliminazione di entità per preservarne la privacy all'interno di testi pubblici.

Scopo del progetto

Questo progetto si propone di sviluppare un sistema di NER tagging multilingua utilizzando un Hidden Markov Model (HMM) con l'algoritmo di Viterbi. Il sistema sarà addestrato e valutato su tre lingue: inglese, italiano e spagnolo. A tal fine, saranno utilizzati tre dataset distinti, ciascuno contenente testi annotati con le entità di interesse, suddivisi in insiemi di addestramento (train), validazione (validation) e test.

Le entità prese in esame saranno di quattro tipi: persona (PER), località (LOC), organizzazione (ORG) e altre entità (MISC). Poiché le entità possono comprendere più parole, per etichettarle correttamente verrà utilizzato il formato BIO (Beginning, Inside, Outside). In questo formato, la prima parola di un'entità è etichettata con "B-" e le parole successive con "I-", mentre tutte le parole che non compongono un'entità sono etichettate con "O". Ad esempio, la corretta sequenza di etichette per "*Ieri ho visto Star Wars*" sarà: [O, O, O, B-MISC, I-MISC].

Il Progetto

Processo di Sviluppo del Sistema NER Tagging Multilingua

Lo sviluppo del sistema di NER tagging multilingua ha seguito una serie di fasi chiave, ognuna delle quali ha contribuito al successo complessivo del progetto.

Acquisizione dei Dati

La fase iniziale del progetto ha coinvolto l'acquisizione dei dati necessari per addestrare e valutare il sistema di NER tagging. Per ottenere dataset ricchi e diversificati, abbiamo fatto affidamento sui dataset messi a disposizione da Babelscape tramite il loro progetto WikiNeural. Questi dataset, specificamente preparati per l'addestramento di modelli di NER su Wikipedia, sono stati una risorsa preziosa per il nostro lavoro.

Abbiamo utilizzato i seguenti dataset di WikiNeural per le lingue inglese, italiana e spagnola:

- Inglese: Link al dataset di WikiNeural in lingua inglese
- Italiano: Link al dataset di WikiNeural in lingua italiana
- Spagnolo:Link al dataset di WikiNeural in lingua spagnola

Questi dataset sono stati già suddivisi in insiemi di addestramento, validazione e test, facilitando il processo di preparazione dei dati per il nostro progetto. Inoltre, essendo basati su Wikipedia, offrono una vasta gamma di testi in diverse categorie e stili linguistici, garantendo una rappresentazione ricca e diversificata delle entità linguistiche nelle rispettive lingue.

L'utilizzo di dataset provenienti da WikiNeural ha permesso di avviare il progetto con una solida base di dati annotati, accelerando il processo di sviluppo del sistema di NER tagging multilingua.

Schema generale del NER tagger

Escludendo il lato teorico del “modelling” con HMM, l’implementazione del tagging attraverso Viterbi è composta da due fasi principali:

1. **Learning**:In questa fase, vengono contate le occorrenze delle coppie di tag consecutivi e le occorrenze di ogni parola con ogni tag nei file di training per calcolare le probabilità di transizione e le probabilità di emissione. In altre parole, si calcola la probabilità di passare da uno stato a un altro (ad esempio, da B-PER a I-PER) e la probabilità di una parola data un'etichetta (ad esempio, la probabilità che l’etichetta B-PER sia associata a "John").

2. **Decoding**: In questa fase, viene selezionata la sequenza più probabile di etichette di entità utilizzando l’algoritmo di Viterbi. Viterbi utilizza una strategia di programmazione dinamica e considerando tutte le possibili sequenze di etichette sceglie quella con la massima probabilità cumulativa, tenendo conto sia delle probabilità di transizione che delle probabilità di emissione calcolate nella fase precedente.

Algoritmo di Viterbi

Idealmente, una volta completato l'apprendimento, sarebbe possibile individuare la sequenza più probabile enumerando tutti i possibili tagging e calcolando ogni probabilità come prodotto delle

corrispondenti probabilità di emissione e transizione. Tuttavia, con T tag e N parole, questo calcolo avrebbe un costo di $O(N^T)$. L'utilizzo dell'algoritmo di Viterbi introduce un enorme vantaggio computazionale, individuando e scartando dalla ricerca tutti quei percorsi che non possono portare a un massimo durante il calcolcolo cumulativo delle probabilità, e permettendo quindi il calcolo in tempo polinomiale $O(N^3)$ di un problema strutturalmente esponenziale.

L'algoritmo di Viterbi è stato implementato in Python per determinare la sequenza ottimale di etichette di entità per una data sequenza di parole. La sua implementazione è stata realizzata traducendo direttamente lo pseudo codice fornito nelle slide del docente. Poiché durante il calcolo delle probabilità di transizione e di emissione si possono verificare valori molto piccoli, si è reso necessario utilizzare la scala logaritmica per evitare l'underflow, un problema comune quando si lavora con probabilità molto basse.

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 $backpointer[q_F,T]$

```python
#implementazione dell'algoritmo di viterbi con la prevenzione dell'underflow tramite logaritmo e probabilità iniziale omogenea

def viterbi(emission_df, transition_df):
    # Numero di stati
    num_states = len(transition_df)

    # Inizializzazione della matrice di probabilità
    dp = pd.DataFrame(index=range(num_states), columns=range(len(emission_df.columns)))
    pi = 1 / num_states #Probabilità iniziale equiprobabile
    dp.iloc[:, 0] = np.log(pi) + np.log(emission_df.iloc[:, 0] + 1e-10)

    # Inizializzazione del percorso ottimale
    path = {state: [state] for state in range(num_states)}

    # Ciclo attraverso le osservazioni
    for t in range(1, len(emission_df.columns)):
        new_path = {}

        # Ciclo attraverso i possibili stati
        for state in range(num_states):
            # calcolo della probabilità massima
            max_prob = float('-inf')
            max_state = None
            for prev_state in range(num_states):
                prob = dp.iloc[prev_state, t-1] + np.log(transition_df.iloc[prev_state, state] + 1e-10) + np.log(emission_df.iloc[state, t] + 1e-10)
                if prob > max_prob:
                    max_prob = prob
                    max_state = prev_state

            dp.iloc[state, t] = max_prob

            # Aggiornamento del percorso ottimale
            new_path[state] = path[max_state] + [state]

        path = new_path

    # Ritorno del percorso ottimale
    max_prob = dp.iloc[:, len(emission_df.columns)-1].max()
    max_path = path[dp.iloc[:, len(emission_df.columns)-1].idxmax()]

    # Stampa a schermo il percorso di Viterbi
    print('Il percorso di Viterbi è:', ' -> '.join(emission_df.index[max_path]))

    return pd.DataFrame({'POSITION': range(len(max_path)), 'WORD': emission_df.columns, 'TAG': [emission_df.index[state] for state in max_path]})
```

Tecniche di Smoothing per Parole Sconosciute

Una delle sfide principali del NER tagging è come trattare le parole sconosciute, cioè le parole assenti nel dataset di addestramento. Per risolvere questo problema, abbiamo implementato diverse strategie di smoothing. Le ipotesi di smoothing utilizzate sono le seguenti:

1. **Sempre O**
   Si presume che le parole sconosciute appartengano sempre alla classe "O" (Outside), cioè non siano entità rilevanti. La probabilità assegnata è:

   $$P(unk \quad O)=1$$

2. **Sempre O o MISC**
   Si presume che le parole sconosciute possano appartenere alla classe "O" o alla classe "B-MISC" (Miscellaneo). Le probabilità sono distribuite uniformemente tra queste due classi:

   $$P(unk \quad O)=P(unk \quad B\text{-}MISC)=0.5$$

3. **Uniforme**
   Le parole sconosciute vengono distribuite uniformemente tra tutte le possibili etichette di entità (NER_TAGs). La probabilità assegnata a ciascuna etichetta è:

   $$P(unk|ti) = 1/\#(NER\_TAGs)$$

   dove **#(NER_TAGs)** è il numero totale di etichette di entità.

4. **Statistica TAG sul Development Set**
   Questa strategia utilizza la distribuzione delle etichette nel development set per stimare le probabilità delle parole sconosciute. In particolare, si considerano le parole che compaiono una sola volta nel development set per stimare le probabilità di ciascuna etichetta. Questo approccio sfrutta la distribuzione empirica delle etichette nel development set per migliorare l'accuratezza delle previsioni per le parole sconosciute.

Metriche di valutazione

Durante lo sviluppo del progetto, la performance del sistema di NER tagging è stata valutata attraverso diverse metriche, ciascuna con un ruolo specifico nell'analisi dei risultati.

- *Accuratezza Complessiva del Tagging*
  L'accuratezza complessiva del tagging misura la percentuale di token etichettati correttamente rispetto al numero totale di token nel testo. Questa metrica fornisce una valutazione generale della precisione del sistema nel riconoscere le entità all'interno del testo.

- *Precisione*
  La precisione rappresenta la percentuale di entità identificate correttamente rispetto al totale delle entità predette dal sistema. In altre parole, misura la proporzione di entità riconosciute correttamente rispetto a tutte le entità etichettate dal sistema come rilevanti. Una precisione elevata indica una bassa probabilità di falsi positivi, cioè di identificare erroneamente un'entità.

- *Recall*
  La recall, anche chiamata sensitivity o true positive rate, misura la percentuale di entità identificate correttamente rispetto al totale delle entità presenti nel testo. In sostanza, indica la capacità del sistema di riconoscere tutte le entità rilevanti presenti nel testo. Una recall elevato indica una bassa probabilità di falsi negativi, cioè di non identificare un'entità che è effettivamente presente nel testo.

La precision e la recall sono state calcolate sulla capacità di riconoscere le entità, rappresentate da quadruple descritte nel seguente formato: E (TAG, Frase-n, indice iniziale, indice finale). Questa rappresentazione descrive l'entità nel modo seguente:

TAG: Indica il tipo di entità, come per esempio persona, luogo, organizzazione, ecc.

Frase-n: Specifica il numero della frase all'interno del testo in cui si trova l'entità.

Indice iniziale: Indica la posizione iniziale dell'entità all'interno della frase, misurata in termini di parole

Indice finale: Indica la posizione finale dell'entità nella frase, misurata anch'essa in termini di parole

In sintesi, questa rappresentazione fornisce una descrizione completa dell'entità identificata, specificando il tipo di entità (TAG), la sua posizione nel testo (numero della frase), e i suoi limiti esatti (indici iniziale e finale).

Inoltre le metriche di precision e recall sono anche state suddivise per tipologia di entità, in modo da poter individuare quali entità risultano più complesse da individuare

I risultati di queste metriche sono stati salvati nel file "evaluation_results.txt", insieme alla tipologia di smoothing applicata (se del caso) e alla dimensione del dataset di test utilizzato. Questi risultati forniscono una panoramica delle performance dei modelli nei diversi scenari considerati, consentendo un confronto diretto.

Struttura delle Baseline

Le metriche sono state confrontate con quelle ottenute da altri approcci di tagging, come un tagger di tipo Naive e un tagger di tipo MEMM (Maximum Entropy Markov Model), per valutare l'efficacia del nostro sistema rispetto ai metodi di riferimento.

Modello Naive

Nel modello Naive , per ogni parola all'interno di una frase, viene valutata la probabilità condizionata di assegnare un'etichetta di entità data l'osservazione della parola stessa. In pratica, se una parola è conosciuta (ovvero presente nel dataset di addestramento), il modello assegna all'etichetta di entità della parola la classe con la maggiore probabilità di emissione. D'altra parte, se una parola è sconosciuta (non presente nel dataset di addestramento), il modello adotta un'approccio più semplice, assegnando all'etichetta di entità della parola la classe "B-MISC" con una probabilità uniforme. Questo approccio "ingenuo" ignora ogni tipo di dipendenza e contesto, permettendo di darci una base minima di performance da cui partire.

Modello MEMM

Nel MEMM, per ogni parola in una frase, il modello valuta la probabilità condizionata di assegnare un'etichetta di entità, tenendo conto delle etichette delle parole precedenti nella sequenza. Inoltre permette di inserire la conoscenza linguistica considerando una quantità notevole di feature booleane che permettono una più attenta preparazione del modello (Regressore Logistico. Tuttavia, l'addestramento e l'uso di un MEMM possono essere computazionalmente più intensivi, poiché richiede la stima e l'ottimizzazione di un gran numero di parametri. Questo approccio più sofisticato e complesso ci offre una possibile performance a cui puntare.

Si noti che l'approccio naive è stato implementato all'interno del progetto, mentre per il modello MEMM è stato utilizzato e opportunamente modificato il seguente codice: https://github.com/Michael-Tu/ML-DLNLP/tree/master/MEMM-POS-Tagger

Risultati

Di seguito vengono riportati i risultati di un test completo su 1000 frasi (28000 token) in tutte le lingue e con tutti i metodi. Ulteriori test sono stati salvati nel file "evaluation_results.txt".

```
################################################################################################################################################
Data: 2024-05-31 09:57:56
Dimensione del dataset: 28000
Tipo di Smoothing: 3
------------------------------------------------------------------------------------------------------------------------------------------------
Risultati:
Lang | Alg     | Acc   | Prec  | Rec   |           | LOC_Prec | LOC_Rec | PER_Prec | PER_Rec | ORG_Prec | ORG_Rec | MISC_Prec | MISC_Rec |
------------------------------------------------------------------------------------------------------------------------------------------------
en   | naive   | 92.8  | 39.2  | 38.2  |           | 45.0     | 60.6    | 35.2     | 46.4    | 31.0     | 62.7    | 45.0      | 19.3     |
en   | viterbi | 94.9  | 49.5  | 74.1  |           | 51.8     | 70.4    | 65.3     | 82.9    | 47.9     | 78.6    | 27.7      | 57.2     |
en   | memm    | 96.1  | 73.1  | 69.8  |           | 85.1     | 62.4    | 78.4     | 78.5    | 62.3     | 84.9    | 61.3      | 60.9     |
------------------------------------------------------------------------------------------------------------------------------------------------
it   | naive   | 93.9  | 63.5  | 46.3  |           | 73.8     | 76.0    | 57.8     | 61.4    | 51.7     | 59.8    | 46.8      | 10.7     |
it   | viterbi | 96.5  | 72.2  | 80.0  |           | 78.5     | 79.7    | 74.8     | 85.7    | 57.6     | 73.9    | 48.5      | 66.1     |
it   | memm    | 97.4  | 79.7  | 78.4  |           | 87.0     | 77.1    | 81.6     | 81.6    | 66.1     | 80.4    | 53.6      | 72.7     |
------------------------------------------------------------------------------------------------------------------------------------------------
es   | naive   | 93.9  | 50.1  | 46.8  |           | 64.2     | 71.1    | 49.5     | 63.5    | 37.3     | 57.1    | 35.5      | 17.3     |
es   | viterbi | 96.4  | 62.5  | 80.9  |           | 68.3     | 79.2    | 70.5     | 90.9    | 58.2     | 88.1    | 45.1      | 64.7     |
es   | memm    | 97.3  | 67.1  | 81.5  |           | 74.7     | 86.8    | 72.9     | 74.8    | 55.1     | 89.0    | 55.2      | 77.2     |
------------------------------------------------------------------------------------------------------------------------------------------------
################################################################################################################################################
```

Si nota che l'approccio banale porta comunque ad un'accuratezza piuttosto elevata (oltre il 93% a seconda della lingua); questo perché la maggior parte delle parole non sono particolarmente ambigue. Dunque, l'incremento delle prestazioni di pochi punti percentuali è in realtà molto significativo.

Per esempio, in un test su 80000 token in lingua inglese, abbiamo ottenuto un'accuratezza per i tre modelli Naive, Viterbi e MEMM rispettivamente del 93.2%, 94.7% e 96.1%. Anche precision e recall tendono ad aumentare, con un notevole distacco nella recall tra Naive e Viterbi e un distacco nella precision tra Viterbi e MEMM.

Per quanto riguarda le tecniche di smoothing, otteniamo risultati migliori con la tecnica 3, che considera una distribuzione uniforme. La tecnica 4, che costruisce una distribuzione personalizzata sul development set, per quanto sofisticata, non ha fornito una conoscenza corretta e utilizzabile. Si consideri che questa tecnica è molto dipendente dai dati utilizzati, quindi con altri dataset potrebbe portare a risultati radicalmente diversi.

Infine, in base ai nostri test, considerando le metriche suddivise per tipologia di entità ottenute da tutti gli algoritmi, sembrerebbe che le entità più facili da etichettare siano di tipo Location, mentre le più difficili siano di tipo Miscellaneous, probabilmente a causa della diversità degli elementi al loro interno.

Conclusioni

L'analisi e la valutazione dei test confermano i risultati attesi: le performance migliorano all'aumentare della complessità del modello. In particolare, il nostro modello HMM con Viterbi si colloca tra il modello Naive e il MEMM. La capacità di considerare il contesto porta un vantaggio significativo rispetto all'approccio banale, ma non è sufficiente per rilevare le entità non conosciute all'interno del train set, nonostante le varie tecniche di smoothing. In questi casi, il MEMM si comporta meglio, prevedendo una possibile entità nuova utilizzando le feature delle parole che la compongono. Tuttavia, è bene notare che il suo tempo di esecuzione è molto esteso, il che non ci ha permesso di eseguire test particolarmente ampi come negli altri due approcci.