

UNIVERSITA’ DEGLI STUDI DI TORINO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Anno Accademico 2024/2025

**Tecnologie del Linguaggio Naturale-Parte II Prof. D. Radicioni**

Autore: Annalisa Sabatelli Matr. 866879

ESERCITAZIONE 1 (2 punti)

Word Sense Disambiguation (WSD)

1. **Introduzione**

L’obiettivo dell’esercitazione WSD è quello di implementare una versione dell'algoritmo di Lesk al fine di permettere la disambiguazione del significato delle parole.

Dal corpus annotato SemCor vengono selezionate casualmente 50 frasi. Per ciascuna frase, si identifica un sostantivo che soddisfa specifici criteri (definiti in seguito), su cui viene applicato l’algoritmo di disambiguazione.

Il processo viene ripetuto per un numero prefissato di iterazioni (n\_iterazioni = 10).

Al termine di questo, viene calcolata l’accuratezza media dell’algoritmo nel determinare correttamente il senso delle parole polisemiche. Tale valore viene poi confrontato con l’accuratezza ottenuta utilizzando l’implementazione dell’algoritmo di Lesk disponibile nella libreria NLTK.

Per la disambiguazione è stato adottato un approccio basato sul modello Bag of Words, in cui le parole del contesto vengono confrontate con i termini presenti nelle definizioni dei sensi associati al lemma.

1. **Struttura del codice**

Il codice è diviso in 3 file:

• *utils.py*: contiene metodi di utility tra cui

* *def extraction\_lemmi\_from\_sentence(sentence):* metodo che prende in input una frase e la elabora restituendo un set costituito dai lemmi delle parole che la costituiscono. In particolare, l’elaborazione si articola nei seguenti passi:
  + rappresentazione di tutte le parole in lower case;
  + rimozione delle stop-words;
  + rimozione della punteggiatura;
  + rimozione degli spazi vuoti;
  + creazione di una lista di lemmi;
  + trasformazione della lista in un set.
* *def extraction\_terms\_from\_corpus(num\_words):* metodo che restituisce una lista di cardinalità *num\_words* di termini ambigui in WordNet e presenti nel corpus Semcor.

• *LeskAlgorithm.py*: contiene il metodo che implementa l’algoritmo di Lesk senza l’utilizzo della libreria NLTK. In particolare:

* *def lesk\_algorithm(term, sentence)*: metodo che prende in input la parola da disambiguare e la frase che contiene la parola e successivamente, per ogni synset della parola, tramite un approccio bag of word, calcola la misura di overlap tra i termini nella frase passata in input e i termini presenti nelle informazioni del synset trattato. Al termine ritorna il synset con massimo overlap.

• *main.py:* contiene il metodo main*.* Dopo aver verificato e scaricato le risorse necessarie da NLTK, come WordNet e SemCor, il programma esegue l’esperimento per un numero predefinito di iterazioni (10). In ogni iterazione, vengono selezionati casualmente 5 termini (sostantivi) dal corpus SemCor tramite la funzione di utilità descritta al punto precedente. Successivamente, viene calcolata la disambiguazione sia per la versione personalizzata dell’algoritmo di Lesk che per quella di NLTK. Alla fine delle iterazioni, viene calcolata l'accuratezza media di entrambi gli algoritmi e stampata in percentuale. Questo approccio consente un confronto diretto tra le due implementazioni in termini di prestazioni*.*

1. **Risultati ottenuti**

Si riporta, di seguito a titolo di esempio, i risultati ottenuti a valle di una esecuzione del codice.

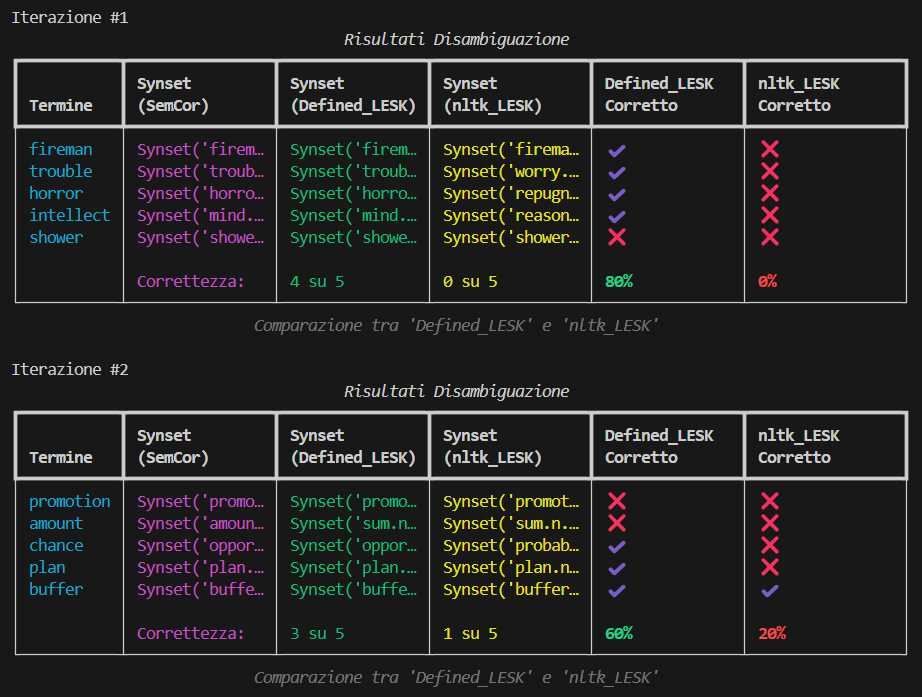


Immagine che contiene testo, schermata, software

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software, design

Il contenuto generato dall'IA potrebbe non essere corretto.

Accuratezza media 'coded\_LESK': **44.0%**

Accuratezza media 'nltk\_LESK': **16.0%**

1. **Conclusioni**

L'accuratezza media ottenuta sulle dieci esecuzioni si attesta intorno al 44%. Tale risultato può essere attribuito, da un lato, alla limitata lunghezza di alcune frasi presenti nel corpus che, combinata con l’approccio BOW (Bag of Words), non fornisce un contesto sufficiente per una distinzione efficace dei diversi sensi. Dall’altro lato, si osservano leggere discrepanze tra i POS tag presenti nel corpus e quelli assegnati dallo strumento NLTK: in alcuni casi, termini etichettati come NN nel corpus non vengono riconosciuti come tali da NLTK, ma classificati diversamente (ad esempio come ADJ), influenzando negativamente le prestazioni complessive del sistema.

ESERCITAZIONE 2 (2 punti)

Document Similarity and Retrieval using the VSM:

A. With Sparse Embeddings

1. **Introduzione**

La seguente esercitazione ha come finalità l’applicazione del Vector Space Model (VSM) per il recupero e la categorizzazione automatica di testi, utilizzando Sparse Embeddings basati su TF-IDF. L’attività è stata svolta impiegando il *News Category Dataset*, un corpus contenente circa 200.000 titoli di notizie provenienti da HuffPost, ciascuno associato a una specifica categoria tematica come politica, tecnologia, sport ed economia. A partire da questo dataset, è stato selezionato un sottoinsieme casuale di 10.000 titoli, sui quali è stata applicata una pipeline di pre-elaborazione che ha incluso la tokenizzazione, la conversione in minuscolo, la rimozione di stopwords e punteggiatura, e la lemmatizzazione. I documenti preprocessati sono stati quindi trasformati in vettori TF-IDF tramite la libreria fornita da scikit-learn. Successivamente, è stato implementato un sistema di recupero documentale basato sulla cosine similarity tra i vettori TF-IDF che sono stati precedentemente creati e query testuali composta da uno o più parole inserite dall’utente. Ciascuna query viene preprocessata e confrontata con i documenti estratti per identificare e restituire i 5 titoli più rilevanti.

1. **Struttura del codice**

Il codice è suddiviso in due file:

• *utils.py*: contiene metodi di utility tra cui:

* *def extraction\_lemmi\_from\_sentence(sentence):* metodo che prende in input una frase e la elabora restituendo una stringa costituita dai lemmi delle parole che la costituiscono. In particolare, l’elaborazione si articola nei seguenti passi:
  + tokenizzazione;
  + rappresentazione di tutte le parole in lower case;
  + rimozione delle stop-words;
  + rimozione della punteggiatura;
  + rimozione degli spazi vuoti;
  + creazione di una lista di lemmi;
  + trasformazione della lista di lemmi in una stringa dove ogni lemma è separato dall’altro da un solo spazio.
* *def pipeline\_vectorize\_training (sentences, vectorizer):* il metodo prende in input una lista di frasi e un oggetto vectorizer già addestrato. Per ogni frase applica la funzione di preprocessing che estrae i lemmi. Successivamente, trasforma le frasi preprocessate in una rappresentazione numerica usando fit\_transform, che adatta il vettorizzatore ai dati e genera la matrice TF-IDF.
* *def pipeline\_retrieval (queries, vectorizer):* il metodo prende in input una lista di query testuali fornite dall’utente e un oggetto vectorizer già addestrato. Ogni query viene preprocessata per estrarne i lemmi. Le query preprocessate vengono poi trasformate in vettori TF-IDF usando transform (senza rifare il fit).  
  In questo modo, le query sono rappresentate nello stesso spazio vettoriale dei documenti di training. La funzione restituisce la matrice vettoriale delle query, utile per confronti e calcolo della similarità.
* *def search\_and\_display\_queries (query\_vector, queries, X\_tfidf, df\_sampled, TOP\_N):* il metdo si occupa di cercare e visualizzare i documenti più rilevanti per ciascuna query fornita. Prende in input i vettori delle query (query\_vector), le query testuali originali, la matrice TF-IDF dei documenti (X\_tfidf), un DataFrame con i titoli e le categorie (df\_sampled), e il numero di risultati da mostrare (TOP\_N). Per ogni query, calcola la similarità coseno tra il vettore della query e tutti i documenti. Ordina i documenti per similarità decrescente e seleziona i TOP\_N risultati più rilevanti. Crea poi una tabella formattata con score di similarità, titolo della notizia e categoria. Infine, stampa la tabella su console per ciascuna query fornendo, quindi, un output leggibile che mostra i documenti più pertinenti rispetto a ciascuna query.

• *main.py*: contiene il metodo main. Dopo aver scaricato il dataset di riferimento, lo carica in formato JSON direttamente in memoria, selezionando casualmente 10.000 titoli di notizie. Per ogni notizia selezionata, estrae le headline e le preprocessa in lemmi usando la funzione *extraction\_lemmi\_from\_sentence*. Le frasi preprocessate vengono convertite in una matrice TF-IDF tramite TfidfVectorizer. Tramite standard input, l’utente inserisce 10 query testuali che a loro volta vengono preprocessate e trasformate in vettori TF-IDF nello stesso spazio vettoriale dei documenti. Infine, per ogni query, viene calcolata la cosine similarity rispetto a tutti i documenti, e i 5 più simili vengono visualizzati in una tabella con punteggio, titolo e categoria.

Lo script consente di valutare l’efficacia del recupero basato su VSM per interrogazioni libere.

1. **Risultati ottenuti**

Di seguito si riportano, a titolo esemplificativo, i risultati ottenuti a seguito dell’esecuzione del codice con le 10 query mostrate in figura. Le query sono state selezionate con l’intento di coprire un’ampia gamma di ambiti semantici, includendo sia concetti astratti sia concreti, provenienti da diversi contesti tematici.

Immagine che contiene testo, schermata, Carattere, software

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, software

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

1. **Conclusioni**

I risultati appaiono in generale coerenti con le query inserite, anche se in alcuni casi si notano anomalie nei punteggi o nella pertinenza. Si analizzano di seguito separatamente le singole query.

1. **Gun:** i titoli selezionati sono chiaramente incentrati sul tema delle armi, con riferimenti espliciti a regolamentazioni e contesti politici, confermando una buona rilevanza semantica.
2. **Supermodel:** i primi due articoli sono adeguati, ma i successivi sembrano completamente irrilevanti, probabilmente a causa di una bassa soglia di similitudine o mancanza di sinonimia semantica nel modello.
3. **Car:** i risultati spaziano da sicurezza a modelli automobilistici, con coerenza tematica, indicando una buona copertura lessicale nel corpus per questa parola.
4. **Bottle:** i titoli trovati sono molto centrati su bevande, riuso e cucina. Probabilmente il termine viene trattato prevalentemente in un contesto alimentare influenzando il recupero dei documenti.
5. **Justice:** i risultati mostrano una varietà di angolazioni (politica, religione, impatto sociale), che suggerisce un buon raggio semantico del termine, anche se alcune sfumature possono risultare vaghe.
6. **Challenge:** i titoli sono molto coerenti con l'idea di “sfida personale” o “iniziativa”, riflettendo un contesto motivazionale o di crescita, probabilmente favorito da frasi ricorrenti nei dataset.
7. **Crime:** alcuni risultati sono solo figurativamente legati al termine, suggerendo un limite nella capacità del sistema di distinguere usi metaforici da quelli letterali.
8. **Paper**: i risultati sono misti ma tutti tematicamente validi. Si passa dalla carta fisica alla pubblicazione accademica indicando una certa ricchezza polisemica del termine nel corpus.
9. **Meat:** i titoli estratti sono tutti coerenti con il cibo e anche con implicazioni etiche e di salute dimostrando una rilevanza contestuale ben conservata.
10. **Bone:** i primi tre articoli sono rilevanti, mentre gli ultimi due sembrano completamente fuori tema.

In sintesi, l’approccio mostra buone capacità di recupero per termini con alta frequenza e contesto ben definito, ma presenta debolezze nei casi di polisemia, ambiguità semantica o bassa soglia di similarità.