Marco Ottina - 795130
Novembre 2019

Scuola di Scienze della Natura
Dipartimento di Informatica

Relazione sugli esercizi di laboratorio
Tecnologie del Linguaggio Naturale – Prof. Radicioni

Esercizio 1

Parte 1 – Word Similarity

L’esercitazione consiste nel calcolare, dato un insieme di coppie di parole,la similarità delle parole di ciascuna coppia tramite tre metodologie che utilizzano la struttura di WordNet; dopodichè confrontare i risultati ottenuti per mezzo di tali metodologie con il valore atteso associato a ciascuna coppia.
Le metodologie hanno il nome di *Wu & Palmer*, *Shortest Path* e *Leakcock & Chodorow*, le funzioni che sono state utilizzate hanno il nome di *coefficiente di correlazione di Pearson* e *coefficiente di correlazione del rank di Spearman.*

Il file contenente l’esercitazione ha per nome *marco_ottina795130_WordSim.py*, è costituito sostanzialmente da una classe che gestisce l’intero esercizio, composto da quattro parti
- lettura del file contenente le coppie di parole con relativo valore atteso, nella funzione *loadTabFile*
- calcolo della similarità tramite le tre metodologie per ciascuna coppia, nella funzione *computeSimilaritiesOfLoadedPairs* invocata da *performExercise*
- calcolo della correlazione tra i risultati delle metodologie ed il valore atteso, eseguito nella funzione *performExercise* stessa
- stampa dei risultati così ottenuti, eseguita nella funzione *printExercise.*

I risultati sono i seguenti:
1) Wu & Palmer,:
- *Pearson*; [[7.65168727, -0.00811761], [-0.00811761, 0.13143374]]
- *Spearman*: correlation -0.013464633593065661, pvalue 0.8009702234382405

2) Shortest Path :
- *Pearson*; [[19.49715361, 0.16174231], [0.16174231, 0.05158137]]
- *Spearman*: correlation 0.2895253335747299, pvalue 3.0336884637747205e-08

3) Leakcock & Chodorow:
- *Pearson*; [[2.82250081, 0.31904184], [0.31904184, 0.35631164]]
- *Spearman*: correlation 0.2895253335747299, pvalue 3.0336884637747205e-08

Conclusione parte 1

Le implementazioni *Shortest Path* e *Leakcock & Chodorow* forniscono risultati simili tra di loro nell se confrontati tramite il *coefficiente di correlazione del rank di Spearman* con i valori associati alle coppie di parole, diversamente da quanto invece mostra il *coefficiente di correlazione di Pearson,* quindi sono misure di similarità simili, potenzialmente complementari.
L'implementazione della similarità *Wu & Palmer* fornisce risultati profondamente diversi dagli altri, quindi essa non può sostituire una delle altre funzioni di similarità senza conseguenza alcuna in generale. Tale differenza può essere dovuta alla funzione che, dato un synset, ne calcola la profondità nella struttura di *WordNet* secondo la gerarchia formata da iponimi e iperonimi: la profondità viene calcolata come la lunghezza minima tra tutti i percorsi possibili che connettono tale dato synset dal synset più generale, ossia *entità*. Infatti tale percorso minimo per un dato synset potrebbe non includere, nei suoi nodi, il *Lowest Common Subsumer* utilizzato nella definizione della funzione di similarità *Wu & Palmer*.

Parte 2 – Algoritmo Lesk

L'esercitazione è suddivisa in due parti:
- implementare ed utilizzare l'algoritmo *Lesk* su ciascuna di un insieme di frasi, ottenute da un file, per disambiguare una specifica parola all'inerno di ciascuna frase.
- dato un insieme di frasi (50 estratte dal corpus SemCor), disambiguare almeno un nome per frase utilizzando l'algoritmo *Lesk,* confrontando poi il risultato con l'algoritmo già implementato nelle API di WordNet.

Il risultato della prima esercitazione evidenzia un problema: le disambiguazioni non sono sempre corrette, ossia vengono attribuiti dei significati talvolta discostanti a quelli intesi da un essere umano. Ciò può essere imputabile alla inadeguatezza del contesto fornito da WordNet, ossia esistono dei synset a cui è associato un contesto insufficiente, inteso come golssa poco descrittiva o numero di esempi esiguo (ad alcuni synsets non è associato alcun esempio).

Il risultato della prima esercitazione fornisce inoltre una probabile spiegazione al risultato della seconda: solo circa il 58% dei nomi analizzati dall'estratto del corpus viene correttamente disambiguato, mentre negli altri casi vengono individuati dei

nomi talvolta molto diversi, talvolta molto smili. Difatti in certi casi la parola individuata è la stessa ma con differente significato associato, anche se di poco. Collassando queste ultime differenze, ossia considerando come *positiva* anche l'individuazione di parole simili con identica sequenza di caratteri ma significati al più simili, le prestazioni crescono fino a quasi il 78%. Ciò però è un errore in quanto i significati diversi devono essere trattati come tali, quindi le prestazioni dell'algoritmo rimangono il 58.07560137457045%.
Inoltre, analizzando a fondo il codice si può notare che la fase di *stemming*, ossia la riduzione a *forma normale* come quella singola, maschile o coniugata al tempo infinito, presenta delle inesattezze: talvolta la parola restituita non è quella corretta ma è una parola arbitriaria. A seconda delle implementazioni può variare, nella versione del software usata dal sottoscritto tale parola è la prima trovata tra le più corte, intendendo la lunghezza come il conteggio dei caratteri. Quindi i controlli sviluppati dal sottoscritto non riescono a riconoscere la similitudine e classificano tale nome come *mismatch*. La fase di *stemming* segue la fase di *pos tagging* ed è una fase tra le ultime, in certi casi proprio l'ultima, nel procedimento di creazione di contesto, quindi è possibile aggirare il problema invocando la funzione di WordNet *_morphy* (in linguaggio *Python*) su ogni parola del contesto per ottenere un insieme di alternative, ma computare il cosiddetto *overlapping* tra la frase di riferimento ed un particolare *synset* della parola da disambiguare nell'algoritmo *Lesk* diviene assai complicato. Infatti, tale computazione sarebbe implementabile, in forma intuitiva, come la sovrapposizione massima tra tutte le possibili alternative sopra citate, il che ha una complessità esponenziale. Considerando che mediamente ci sono due alternative al massimo, la computazione non risulterebbe eccessiva, ma rimarrebbe un problema non triviale.

Esercizio 2 – Nasari

L’esercizio consiste di produrre un estratto o un abstract di ciascuno di tre articoli. A tal fine si utilizza la struttura derivante dal paper *Nasari.*

Il codice è suddiviso in classi, di cui una per il *gestore* che si occupa di leggere i file necessari, fornire gli articoli da elaborare ed eseguire l’elaborazione. Tale elaborazione costituisce il nucleo dell’esercitazione ed è implementata in una classe esterna, *NasariExtractor_ParagraphByTitle.*

L’algoritmo di estrazione è un ibrido tra il *unsupervised learning*, *Cohesion: word co-occurence* e *title method*. Si usa il titolo per generare il contesto (*bag of words*), analogamente per ogni paragrafo si ricava il contesto, dopodichè assegna un *punteggio* ad ogni paragrafo. A tal proposito, si ricavano sia l’insieme di vettori di BabelNet delle parole del titolo sia l’analogo insieme di vettori delle parole del paragrafo, dopodichè si somma la *similarità* di ogni possibile coppia di vettori, di cui il primo appartiene al titolo ed il secondo al paragrafo. Il calcolo della similarità è definito come il *weighted overlap*. Dopo aver ottenuto il punteggio di ogni paragrafo, si ottiene un sottoinsieme di paragrafi formato dai paragrafi aventi il maggior punteggio.

Il codice sorgente mostra sia l’algoritmo appena descritto sia una versione semplificata basata su WordNet. Similarmente a quanto descritto prima, si genera il contesto (*bag of words*) sia del titolo sia dei singoli capitoli e si contano le ripetizioni di ogni singola parola, filtrando infine le parole aventi un conteggio superiore o uguale ad una certa soglia.

Eseguendo il codice, viene prima mostrato il risultato del primo algoritmo, poi di quest’ultimo.

Esercizio 3 – Babelnet

L’esercizio consiste nel valutare la similarità di cento coppie di parole utilizzando BabelNet e la sua struttura, confrontando poi tale valutazione con quella fornita a priori dal sottoscritto.
Per ogni coppia di parole, tramite alcune mappature, si ottengono i vettori di BabelNet di tali parole e si calcola la similarità di questi vettori tramite la funzione *cosine similarity*. In particolare, i vettori di BabelNet sopracitati sono un insieme di vettori, associati ad una data parola, che rappresentano ciascuno una parola simile per significato semantico alla parola data, similarmente ai synsets di Wordnet. Date due parole quindi si calcolano le similarità di ogni coppia di vettori e si sceglie la coppia avente similarità più elevata. In tale modo si calcola inoltre il significato di una parola che più plausibilmente è semanticamente simile all’altra parola della coppia, sicché ogni parola funge da contesto di disambiguazione per l’altra.
I risultati prodotti sono spesso simili. Le differenze sono dovte a collegamenti assenti, talvolta metaforici e talvolta riguardanti una conoscenza culturale più approfondita di quella inseria in BabelNet (la quale è limittata per varie ragioni, tra cui il contenimento dello spazio di memorizzazione). Infatti in certi casi BabelNet non ha reso disponibile delle parole, informazioni, contesti, glossa od esempi correlati ed in alcuni rari il sottoscritto non ha considerato delle definizioni o ha considerato le parole correlate in maniera diversa a causa di un diverso modo di valutare una data coppia rispetto alla valutazione fornita durante lo sviluppo di BabelNet stesso.