Relazione Esercizio wordnet

L'esercitazione 1 si articola in 2 parti:

1. conceptual similarity
2. wordsense disambiguation

Conceptual similarity

Nella prima parte si è affrontato il problema del calcolo delle metriche di similarità basate sullo strumento di WordNet. L'esercitazione ha avuto come obiettivo, dapprima implementare le metriche di similarità di "wu and palmer", "shortest path" e "leacock chodorow" e successivamente testare tali misure su una lista di termini presenti in un file wordsin353.csv. Le similarità trovate, infine, vengono correlate con quelle presenti nel file .csv usando diverse metriche di correlazione.

Di seguito viene illustrato il criterio usato per calcolare le metriche di similarità.
Le varie misure sono dichiarate come funzioni nel file **"similarity.py".**
Il calcolo della similarità di **wu_and_palmer** vede l'individuazione del synset antenato comune (**Lowest Common Subsumer o LCS**) tra i sensi dati in input.
Il LCS è stato calcolato considerando i percorsi più lunghi tra tutti i possibili percorsi dei synset in input fino alla radice dei loro alberi wordnet (la profondità dai sensi). Per calcolare tali percorsi si è ricorso alla funzione di libreria **"hypernym_paths"** che
restituisce una lista di tutti i possibili cammini dal senso alla sua radice sfruttando la struttura di wordnet e gli iperonimi dei sensi.
In seguito sono stati confrontati i percorsi più lunghi dei 2 sensi in modo da trovare, se esiste, l'antenato comune. Ottenuto l'antenato comune, viene calcolata la sua profondità nell'albero di wornet.
Il risultato del calcolo della metrica di “wu and palmer” è espresso come il rapporto tra il doppio della lunghezza del percorso più lungo dal LCS alla sua radice e la somma delle profondità dei sensi in input

La metrica **"shortest_path"** si avvale della misura di distanza tra i synset dei nodi che viene sottratta al doppio della profondità massima di wordnet. La distanza è calcolata come somma del numero di nodi da attraversare dal **syn1** fino al LCS calcolato tra **syn1 e syn2**, (se esiste, altrimenti None) sommato al numero dei nodi da attraversare dal **syn2** al LCS tra **syn1 e syn2**. La profondità massima è calcolata come la lunghezza del percorso massimo fra tutti i percorsi più lunghi di tutti i sensi in wordnet, ovvero:
MAX_DEPTH = max(max(len(hyp_path) for hyp_path in ss.hypernym_paths()) for ss in wn.all_synsets())

Il calcolo della similarità di **leacock e chodorow** sfrutta la nozione di distanza tra synset usata nella metrica di "shortest_path" e quella di profondità massima di wordnet. La formula per calcolare questa metrica è la seguente:

-math.log((minP + 1) / (2 * MAX_DEPTH + 1))

con: minP = distanza minima trai sensi

con: MAX_DEPTH = profondità massima di wordnet

Il progetto ha visto la realizzazione di una classe **"conceptualSimilarity"** che predispone la lettura del file wordsin353, all'atto della creazione. Il metodo "calculate_similarity", data in input una metrica di similarità, scelta tra:

- **wupalmer**: (wu and palmer)
- **shortest**: (shortest_path)
- **leacock**: (leacock e Chodorow)

restituisce la similarità tra per ogni coppia di termini Word1 Word2 letti dal file .csv, come massimo della similarità tra ogni coppia di sensi dei 2 termini.

Il metodo "similarity_correlation" calcola la correlazione (pearson o spearman) tra le similarità presenti nel file.csv e quelle restituite dal calcolo della similarità con le metriche di wordnet.

Un programma di esempio

Ad uso dimostrativo è stato definito il file **"conceptual_similarity_main.py"** che permette all'utente di eseguire dei test sul file wordsin353 considerando le diverse strategie di similarità e le metriche di correlazione. La sintassi per eseguire correttamente il main è:

python conceptual_similarity_main.py csv_path_file SIMILARITY CORRELATION

- dove, SIMILARITY in [wupalmer, shortest, leacock]
- dove, CORRELATION in [pearson, spearman]

Risultati dei test

Di seguito vengono riportati i risultati delle correlazioni sulle varie metriche di similarità.

Wu and Palmer:

- Correlazione di pearson : 0.2668794407449972
- Correlazione di spearman : 0.30053507236232363

Il primo risultato segna che le 2 variabili sono direttamente correlate, tuttavia si tratta di una correlazione debole. Similmente, il secondo risultato mette in evidenza una correlazione tra ranghi debole

shortest path:

- Correlazione di pearson : -0.06049101059773906
- Correlazione di spearman : 0.09571937623870949

Il risultato dato dall'indice di pearson segna una evidente correlazione negativa molto debole. Rispetto alla similarità di wu and palmer, la shortest path restituisce più risultati meno correlati. Anche l'indice di spearman segna una correlazione tra ranghi molto debole sebbene positiva rispetto alla correlazione di pearson.

leacock and chodorow:

- Correlazione di pearson : 0.11220303448810581
- Correlazione di spearman : 0.09571937623870949

Anche in questo caso le due metriche segnano una correlazione positiva molto debole. Sebbene le correlazioni di pearson tra le precedenti 2 metriche sono diversi, i valori restituiti dalle correlazioni di spearman per le due metriche di shortest path e leacock and chodorow sono identici.

Wordsense disambiguation

Nella seconda parte dell'esercitazione abbiamo affrontato il problema della disambiguazione semantica utilizzando lo strumento di wordnet (wordsense disamiguation). Il file **LeskAlgorithm.py** contiene la definizione di una funzione **"lesk"** che implementa l'algoritmo di Lesk, usato per disambiguare il significato delle parole. Tale algoritmo funziona avvalendosi di una semplice euristica: il miglior significato è quello che realizza il più alto valore di overlap tra i termini che compaiono nel contesto (context) e quelli nelle definizioni e negli esempi (signature) associati ai vari synset della parola da disambiguare. L'overlap, in questa implementazione, è calcolato come intersezione tra contesto e signature. Un primo approccio alla definizione dell'algoritmo di lesk ha visto una fase di pre-processing dei dati eseguendo una lemmatizzazione dei termini che occorrono nella signature e nel contesto. Tale operazione è stata condotta per categorie lessicali, di modo che si potesse ricondurre la riduzione in lemmi delle parole al giusto POS tag. La funzione **"lemmatization"** nel file "LeskAlgorithm.py" esegue tale operazione di pre-processing, riceve in input una lista di parole e restituisce un insieme di lemmi senza ripetizioni. Inizialmente, la funzione considerava, nell'insieme di lemmi quelli provenienti dalle principali categorie lessicali: "sostantivi, verbi, avverbi, aggettivi", in seguito si è fatto uso solo dei sostantivi ritenendo che generassero meno ambiguità. Eseguendo l'algoritmo di lesk su un insieme di frasi e osservando i risultati degli overlap si è notato che, in molte occasioni, le intersezioni tra contesto e signature fossero poco significative. Parole come "be" oppure "not", rispetto ad altre, non contribuiscono al significato della parola e inevitabilmente spingono l'overlap verso valori alti, di modo che, se ci sono synset più promettenti con lo stesso valore di overlap, questi non vengono considerati. Per tale ragione si è deciso di includere una seconda operazione di pre-processing, nella quale si rimuovono tutte le stopword dalla lista di parole data input all'algoritmo di lesk. Infine, come ultima ottimizzazione, osservando che l'accuratezza nelle predizioni fosse un valore basso che oscillava fra il 30% e il 40%, si è deciso di migliorare ulteriormente l'overlap andando a considerare, nell'intersezione, i lemmi dei sostantivi provenienti dalle definizioni e dagli esempi di tutti gli iponimi e gli iperonimi dei synset della parola da disambiguare, raggiungendo un'accuratezza tra il 45% e il 53%.

Risultati dei test

Di seguito si riportano i risultati dei test richiesti presenti nella traccia dell'esercizio:

- risultato della disambiguazione della parola "**bank**" dato il contesto: "the bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities": l'algoritmo ha restituito correttamente **Synset('depository_financial_institution.n.01')**
- Risultato della disambiguazione della parola **"ash"** dato il contesto: "The house was burnt to ashes while the owner returned". L'algoritmo ha restituito correttamente **Synset('ash.n.01')**
- Risultato della disambiguazione della parola "ash" dato il contesto: "This table is made of ash wood". L'algoritmo di ha restituito correttamente **Synset('ash.n.02')**

In tutti i casi l'algoritmo risponde disambiguando correttamente la parola, dato il contesto in cui occorre. Senza considerare l'ottimizzazione con le stopword e le signature degli iponimi e degli iperonimi dei synset, in una precedente versione l'algoritmo restituiva per l'ultimo test il synset '**ash.n.01',** risultando meno accurato.

Semcor disambiguation

Per condurre una indagine più appropriata sull'accuratezza dell'algoritmo di lesk, abbiamo testato la nostra definizione del lesk su un sottoinsieme di frasi estratto casualmente dal corpus Semcor.
SemCor è un corpus in lingua inglese composto da 352 testi annotati semanticamente, con un totale di 37176 frasi, che provengono dal Brown corpus. Attualmente, Semcor rappresenta il più ampio dataset annotato a mano con i synset di wordnet.
Tutte le funzioni usate per manipolare il corpus, analizzare le frasi ed estrarre i sensi sono da riferirsi alla documentazione ufficiale della libreria **nltk**, disponibile al seguente link: https://www.nltk.org/_modules/nltk/corpus/reader/semcor.html

Per eseguire il testing sul corpus Semcor è stata definita la libreria python **"semcore.py".** Analizzando il contenuto di alcuni documenti del corpus, come prima cosa, è stato necessario definire una funzione che organizzasse la lettura e l'estrazione dei sensi annotati del corpus. I file del corpus sono documenti XML, nei quali un tag **<p pnum="1">** segna l'inizio di un paragrafo del documento e un tag **<s snum="1">** individua l'inizio di una frase all'interno del paragrafo.
Ad esempio, le seguenti righe individuano 3 frasi distinte:

- <wf cmd="ignore" pos="DT">The</wf>
- <wf cmd="done" pos="JJ" lemma="nuclear" wnsn="1" lexsn = " 3:00:00 ::" > nuclear</wf>
- <wf cmd="done" pos="NN" lemma="war" wnsn="2" lexsn = "1:26:00::" > war</wf>

Non tutte le frasi del corpus presentano lo stesso contenuto informativo. Ad esempio, le stopword come “The”, nella prima frase, non hanno associato un attributo **“lemma”** o un attributo **“wnsn”** che individua il rispettivo synset di wordnet. In generale, queste mancanze sono presenti sia per le parole di contenuto, sia per le parole di funzionalità.
Per eseguire correttamente la disambiguazione sono state considerate solo le frasi annotate con le indicazioni del POS e del riferimento a wordnet. La disambiguazione è stata eseguita solo sui sostantivi, riconoscibili dall’attributo **“pos=NN”.**
La funzione **“open_semcor”** esegue la lettura di un sottoinsieme di frasi, il cui numero è dato in input, su un documento di semcor passato per parametro. Affianco a questa, è stata definita la funzione **“open_random_semcor”** che esegue la lettura allo stesso modo della precedente ma sull’intero nucleo di documenti del corpus, restituendo un sottoinsieme di frasi scelte in modo casuale.
Una terza funzione **“read_nouns_from_semcor”** legge da una lista di frasi in input i lemmi associati ai sostantivi **(tag NN),** quindi restituisce una lista di terne:

- **lemma**: il lemma associato al sostantivo
- **synset**: l'indicazione del synset associato al lemma, preso dal corpus annotato
- **index**: indice della frase da cui è stato estratto il lemma

Una volta letti i sostantivi da disambiguare, l’operazione di disambiguazione viene eseguita dalla funzione **“word_sense_disambiguation”** che riceve in input una lista di lemmi di sostantivi, e una lista di frasi in cui essi occorrono, con il ruolo di contesto. La funzione invoca l'algoritmo di Lesk per ogni coppia (lemma,frase), quindi calcola l'accuratezza totale sull'insieme di frasi.

Un ultima funzione **“testing_WSD_on_semcor”** esegue le precedenti 3 operazioni, (lettura delle frasi, recupero dei sostantivi, disambiguazione) tante volte quanto vale il parametro iteration, dato in input. L’obiettivo è condurre un certo numero di test disambiguando i sostantivi recuperati da un insieme di frasi scelte in modo casuale dall’intero corpus semcor. Il parametro "sentence_number" indica il numero di frasi da estrarre. Il parametro "max_number_lemmas" indica il massimo numero di lemmi da disambiguare presi in modo casuale dalla lista di lemmi. Se questo valore eccede il numero di lemmi in una lista, viene selezionato un sottoinsieme casuale su tutta la lista.

Un programma di esempio

Ad uso dimostrativo e per condurre i vari testi è stato definito il file wordsense_disambiguation_main.py, il quale esegue un certo numero di test di disambiguazione di frasi estratte casualmente sull'intero corpus semcor. La sintassi per eseguire correttamente il programma è:
python wordsense_disambiguation_main.py **N_SENT MAX_LEMMAS N_ITERATION**

- dove, N_SENT = numero di frasi da testare
- dove, MAX_LEMMAS = massimo numero di sostantivi per blocco di N_SENT frasi scelte casualmente
- dove, N_ITERATION = numero di test da eseguire. Ad ogni test vengono selezionate in modo casuale N_SENT frasi

Risultati dei test

Come risultato dei test, di seguito viene riportato il log del programma "wordsense_disambiguation_main.py". Sono stati eseguiti 10 test di disambiguazione su blocchi di 50 frasi estratte casualmente dal corpus semcor. Nei test vengono selezionati un numero variabile di sostantivi da 1 a 10.

iterazione numero 1:
numero di lemmi random: 4
accuratezza test 25.0%

iterazione numero 2:
numero di lemmi random: 10
accuratezza test 40.0%

iterazione numero 3:
numero di lemmi random: 5
accuratezza test 80.0%

iterazione numero 4:
numero di lemmi random: 2
accuratezza test 50.0%

iterazione numero 5:
numero di lemmi random: 4
accuratezza test 50.0%

iterazione numero 6:
numero di lemmi random: 8
accuratezza test 37.5%

iterazione numero 7:
numero di lemmi random: 8
accuratezza test 62.5%

iterazione numero 8:
numero di lemmi random: 6
accuratezza test 66.67%

iterazione numero 9:
numero di lemmi random: 10
accuratezza test 60.0%

iterazione numero 10:
numero di lemmi random: 8
accuratezza test 62.5%

Accuratezza totale: 53.42%