Relazione
Esercitazione 1.1

Concept Similarity

Introduzione

La seguente esercitazione richiede di calcolare, tramite WordNet, la similarità massima tra coppie di termini raccolte nel file *WordSim353*.

Il valore di similarità ottenuto verrà successivamente confrontato, tramite indici di correlazione, con un *gold value*, ovvero il valore definito da un gruppo di utenti annotato nel file prima citato (W*ordSim353)*.

Per il calcolo della similarità sono state utilizzate tre metriche

- Wu & Palmer
- Shortest Path
- Leakcock & Chodorow

Mentre gli indici di correlazione presi in considerazione sono

- Pearson
- Spearman

NOTE:

Per calcolare la similarità fra due termini immaginiamo di prendere la massima similarità fra tutti i sensi del primo termine e tutti i sensi del secondo termine. L'ipotesi consiste nel fatto che i due termini funzionano come contesto di disambiguazione l'uno per l'altro.

Nella formula ci sono i concetti che appartengono ai synset associati ai termini w1 e w2.

$$\mathrm{sim}(w_1, w_2) = \max_{c_1 \in s(w_1), c_2 \in s(w_2)} [\mathrm{sim}(c_1, c_2)]$$

Struttura del codice

Per lo svolgimento di questa esercitazione il progetto è stato diviso nei seguenti file:

- **`metrics.py`** contiene l'implementazione delle tre metriche:
    - *Wu & Palmer*
    - *Shortest Path*
    - *Leacock & Chodorow*
- **`utils.py`** contiene le funzioni di utilità necessarie come metodi di supporto, per ogni metrica prima citata
- **`main.py`** legge e parsifica il file WordSim353.csv ed esegue un ciclo per ogni record del file calcolando le metriche ottenute, tramite gli indici di correlazione

Descrizione ed implementazione delle metriche

Wu & Palmer (WUP)

Descrizione e formulazione

Di seguito la formula della metrica

$$\mathrm{cs}(s_1, s_2) = \frac{2 \cdot depth(\mathrm{LCS})}{depth(s_1) + depth(s_2)}$$

Dove, *depth(x)* rappresenta il livello di profondità dell'albero a cui si trova quel determinato nodo.

LCS è il primo antenato in comune tra due sensi ovvero risalendo l'albero tassonomico della gerarchia dei nomi il primo nodo genitore comune tra i due sensi.

Implementazione

Il calcolo della similarità di WUP è definito nel metodo `wu_palmer (w1, w1)` che prende in input una coppia di termini, tale metodo 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 dei sensi). L'antenato comune ritornato dalla funzione è calcolato creando 2 insiemi contenenti rispettivamente gli iperonimi di `syns1` e di `syns2`.

Nello specifico vengono considerati gli iperonimi che sono in comune tra i due synset e vengono inseriti nella lista `common`. Successivamente viene ordinata la lista delle intersezioni di iperonimi in base alla loro profondità (cioè alla distanza dalla radice e l'iperonimo in questione). Infine viene ritornato il primo elemento di lcs (se lcs non è vuota) ovvero l'iperonimo comune ai due sensi come la minore max_depth()

```
def lcs(syns1, syns2):

    hp1 = get_hypernyms(syns1)
    hp2 = get_hypernyms(syns2)

    common = intersection(hp1, hp2)

    lcs = sorted(common, key=lambda hp: hp.max_depth(), reverse=True)

    return (None if len(lcs) == 0 else lcs[0])
```

Il risultato del calcolo della metrica di WUP 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.

Shortest Path (SP)

Descrizione e Formulazione

Di seguito la formula della metrica

$$\text{sim}_{path}(s_1, s_2) = 2 \cdot \text{depthMax} - len(s_1, s_2)$$

Per ogni versione specifica di WordNet, `depthMax` è un valore fissato.

La similarità tra i due sensi `(s1, s2)` è funzione del percorso più corto tra i due sensi `len(s1,s2)`

- se `len(s1,s2)` è 0, `sim_path(s1,s2)` è il massimo valore `2*depthMax`.
- se `len(s1,s2)` è `2* depthMax`, allora `sim_path(s1,s2)` è il valore minimo `0`.

Quindi il valore di `sim_path(s1,s2)` è compreso tra `0` e `2*depthMax.`

Implementazione

Per la versione corrente di WordNet la profondità massima è pari a 20. Ottenibile dalla seguente stringa di codice:

```
return max(max(len(path) for path in sense.hypernym_paths()) for sense in
wn.all_synsets())
```

Questa metrica si avvale della misura di distanza tra i synset dei nodi, la quale 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 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.

Si calcola l'LCS e si ritorna la somma della distanza tra s1 e LCS e s2 e LCS.

```
def shortest_path(w1, w2):

    min_length = DEPTH_MAX * 2
    for s1 in wordnet.synsets(w1):
        for s2 in wordnet.synsets(w2):
            len = utils.len_path(s1,s2)
            if len:
                if len < min_length:
                    min_length = len
    return 2 * DEPTH_MAX - min_length
```

Leacock Chodorow (LC)

Descrizione e Formulazione

Di seguito la formula della metrica

$$\mathrm{sim}_{LC}(s_1, s_2) = -\log \frac{len(s_1, s_2)}{2 \cdot \mathrm{depthMax}}$$

Quando `s1` e `s2` hanno lo stesso senso, `len(s1, s2) = 0`. A livello pratico, si aggiunge `1` sia a `len(s1, s2)` sia a `2*depthMax` in modo da evitare `log(0)`. Quindi il valore di `simLC(s1,s2)` è compreso nell'intervallo `(0, log(2 * depthMax + 1))`

Implementazione

Dati i sensi `s1` e `s2` si calcola `len = len_path(s1,s2)`

```
def leacock_chodorow(w1, w2):
    max_sim = 0
    for s1 in wordnet.synsets(w1):
        for s2 in wordnet.synsets(w2):
            len = utils.len_path(s1,s2)
            if len:
                if len > 0:
                    sim = -(log(len / (2 * DEPTH_MAX)))
                else:
                    sim = -(log(len + 1 / (2 * DEPTH_MAX + 1)))
            else:
                sim = 0
            if sim > max_sim:
                max_sim = sim
    return max_sim
```

La funzione ritorna il massimo valore di similarità fra ogni coppia di sensi

Risultati

Di seguito i risultati ottenuti dagli indici di Spearman e Pearson con le metriche implementate

Metrica (implementata)	Spearman	Pearson
Wu & Palmer	0.31788	0.2675
Shortest Path	0.22591	0.08635
Leakcock & Chodorow	0.22591	0.23804

A fini di comparazione illustro di seguito i valori ottenuti con le metriche built-in della libreria NLTK

Metrica (built-in)	Spearman	Pearson
Wu & Palmer	0.33825	0.29475
Shortest Path	-0.10506	-0.13089
Leakcock & Chodorow	0.30119	0.32598

Conclusioni

Analizzando nel dettaglio possiamo indicare che per quanto riguarda

- Wu & Palmer
    - I risultati di entrambi gli indici sono per lo più direttamente correlati, tuttavia, si tratta di una correlazione debole sia per le metriche implementate che per le metriche built-in di NLTK
- Shortest Path
    - Rispetto alla similarità precedente, la Shortest Path restituisce più risultati meno correlati, sebbene la correlazione rimanga molto bassa.
- Leakcock & Chodorow
    - Anche in quest'ultimo caso i due indici sono per lo più direttamente correlati. Sebbene le correlazioni di Pearson tra le precedenti due metriche sono diverse, i valori restituiti dalle correlazioni di Spearman per le due metriche di Shortest Path e Leacock Chodorow sono identici.

Sarebbe opportuno specificare che per alcune casistiche, la similarità ritornata in merito risulta pari a zero, un esempio è la coppia di termini

`Maradona, Football`

- `Synset Maradona : []`

- Synset **Football**: [Synset('football.n.01'), Synset('football.n.02')]

Questo è dovuto al fatto che il termine *Maradona* non abbia synset su WordNet. Il valore di similarità, perciò andrà ad influire sui valori degli indici di correlazione nonostante la coppia di termini in questione abbia parecchia similarità per un essere umano.

Anche per questa coppia di termini

Investor, Earning

- Synset **Investor**: [Synset('investor.n.01')]
- Synset **Earning**: [Synset('gain.v.08'), Synset('earn.v.02')]

Earning è un verbo mentre Investor è un sostantivo, per cui riceviamo questo errore

Computing the lch similarity requires Synset('investor.n.01') and Synset('gain.v.08') to have the same part of speech

Per concludere, come si nota dai risultati mostrati e descritti, tutte le misure di similarità sono basse (anche per le misure calcolate con le metriche built-in di NLTK), il che significa che queste sono ancora molto distanti dal ragionamento di un annotatore umano.

Relazione
Esercitazione 1.2

Word Sense Disambiguation

Introduzione

In questa esercitazione si è implementato l'algoritmo di Lesk in versione semplificata. Lo scopo di questo algoritmo è quello di fare Word Sense Disambiguation (WSD).

Vengono estratte randomicamente 50 frasi dal corpus annotato `SemCor` e viene applicata la disambiguazione su un sostantivo all'interno della frase che rispetti certe condizioni (descritte in seguito).

Al termite di `n_iterazioni` (10), il programma indica l'accuratezza con cui l'algoritmo ha disambiguato i termini polisemici e confronta il risultato con l'accuratezza della disambiguazione effettuata dall'algoritmo di Lesk di NLTK

NOTE:

È stato adottato un approccio `bag of words` per la disambiguazione di parole.

Struttura ed implementazione del codice

Il codice è diviso in 3 file:

- **`utils.py`** contiene metodi di utily tra cui
    - `process_corpus` metodo che prendi in input il numero di parole da estrare dal corpus. I termini vengono estratti randomicamente secondo questi requisiti
        - il termine deve essere diverso da quelli estratti precedentemente
        - il termine deve essere un sostantivo
        - il termine deve avere almeno più di un synset

Una volta estrapolati i termini viene ritornata una struttura costituita da

- Termine
- Synset del termine
- Frase che contiene il termine

- `process_stop_words` metodo che parsifica le stop-words prese dal file *stop_ words_ FULL.txt*

- `clean_sentence` metodo che esegue una pipeline di pulizia della frase passata in input. La pipeline consiste in

  - portare tutte le parole della frase in minuscolo
  - rimuovere le stop-word (richiamando il metodo prima descritto)
  - rimuovere la punteggiatura
  - rimuovere eventuali spazi vuoti
  - lemmatizzare le parole

- **`mylesk.py`**

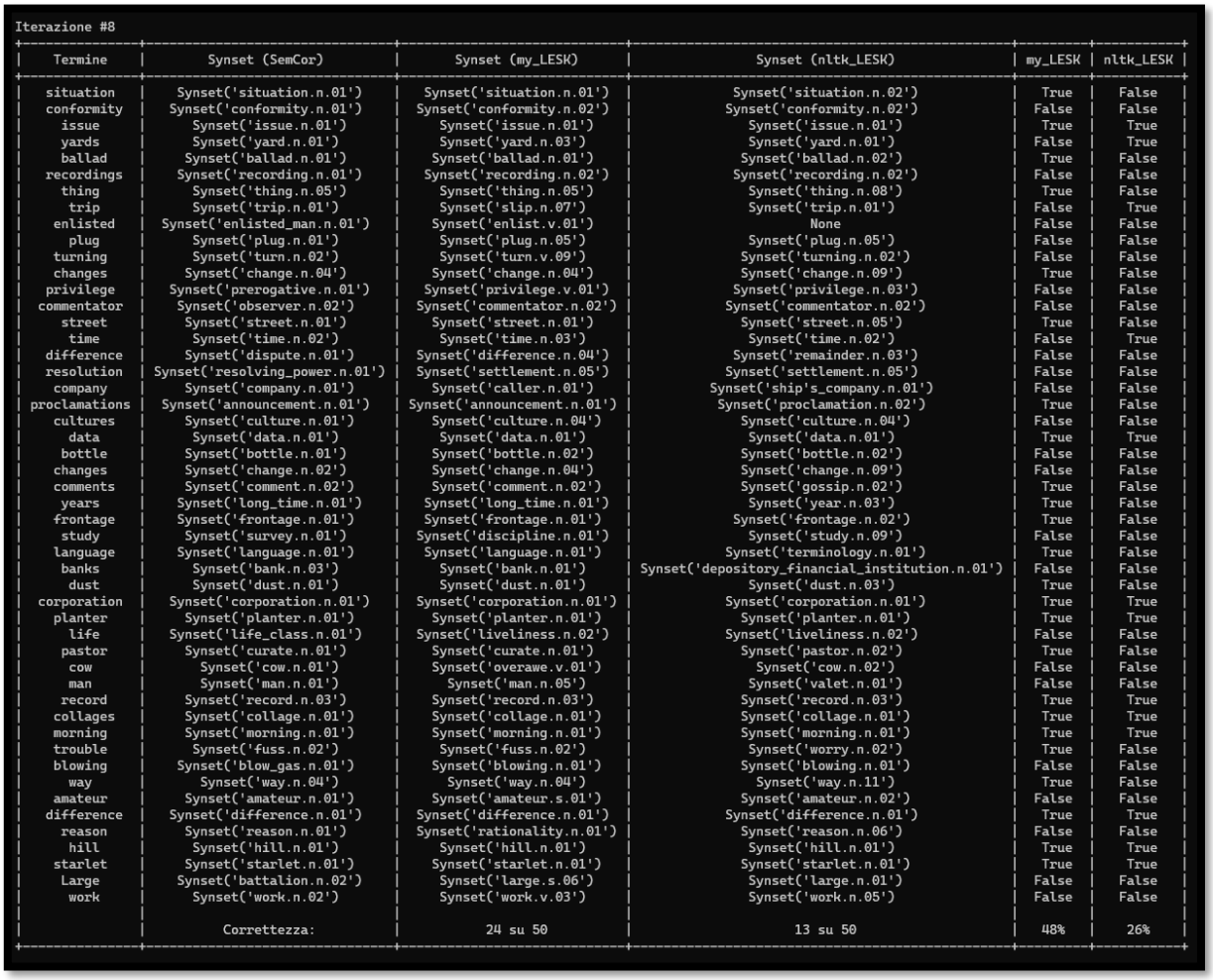
  contiene il metodo `my_lesk` che, come indica il nome, è un'implementazione del metodo originale di NLTK: al metodo viene dato in input la parola da disambiguare e la frase che contiene la parola, successivamente per ogni synset della parola, tramite un approccio bag of word viene calcolata la misura di overlap tra i termini nella frase passata in input e i termini presenti nelle informazioni del synset trattato. Al termine viene ritornato il synset con massimo overlap.

- **`main.py`**

  viene richiamato il metodo `process_corpus` (prima descritto) passando in input il numero di parole che si vogliono estrarre dal corpus annotato. Successivamente viene eseguito un ciclo di n. iterazioni (`N_ATTEMPTS`) dove per ogni iterazione viene richiamato l'algoritmo di Lesk implementato (`my_lesk.lesk`) sia quello di NLTK (`lesk`). I risultati vengono poi stampati in una tabella auto esplicativa.

Risultati

Di seguito una schermata di un'iterazione



Iterazione #8

Termine	Synset (SemCor)	Synset (my_LESK)	Synset (nltk_LESK)	my_LESK	nltk_LESK
situation	Synset('situation.n.01')	Synset('situation.n.01')	Synset('situation.n.02')	True	False
conformity	Synset('conformity.n.01')	Synset('conformity.n.02')	Synset('conformity.n.02')	False	False
issue	Synset('issue.n.01')	Synset('issue.n.01')	Synset('issue.n.01')	True	True
yards	Synset('yard.n.01')	Synset('yard.n.03')	Synset('yard.n.01')	False	True
ballad	Synset('ballad.n.01')	Synset('ballad.n.01')	Synset('ballad.n.02')	True	False
recordings	Synset('recording.n.01')	Synset('recording.n.02')	Synset('recording.n.02')	False	False
thing	Synset('thing.n.05')	Synset('thing.n.05')	Synset('thing.n.08')	True	False
trip	Synset('trip.n.01')	Synset('slip.n.07')	Synset('trip.n.01')	False	True
enlisted	Synset('enlisted_man.n.01')	Synset('enlist.v.01')	None	False	False
plug	Synset('plug.n.01')	Synset('plug.n.05')	Synset('plug.n.05')	False	False
turning	Synset('turn.n.02')	Synset('turn.v.09')	Synset('turning.n.02')	False	False
changes	Synset('change.n.04')	Synset('change.n.04')	Synset('change.n.09')	True	False
privilege	Synset('prerogative.n.01')	Synset('privilege.v.01')	Synset('privilege.n.03')	False	False
commentator	Synset('observer.n.02')	Synset('commentator.n.02')	Synset('commentator.n.02')	False	False
street	Synset('street.n.01')	Synset('street.n.01')	Synset('street.n.05')	True	False
time	Synset('time.n.02')	Synset('time.n.03')	Synset('time.n.02')	False	True
difference	Synset('dispute.n.01')	Synset('difference.n.04')	Synset('remainder.n.03')	False	False
resolution	Synset('resolving_power.n.01')	Synset('settlement.n.05')	Synset('settlement.n.05')	False	False
company	Synset('company.n.01')	Synset('caller.n.01')	Synset('ship's_company.n.01')	False	False
proclamations	Synset('announcement.n.01')	Synset('announcement.n.01')	Synset('proclamation.n.02')	True	False
cultures	Synset('culture.n.01')	Synset('culture.n.04')	Synset('culture.n.04')	False	False
data	Synset('data.n.01')	Synset('data.n.01')	Synset('data.n.01')	True	True
bottle	Synset('bottle.n.01')	Synset('bottle.n.02')	Synset('bottle.n.02')	False	False
changes	Synset('change.n.02')	Synset('change.n.04')	Synset('change.n.09')	False	False
comments	Synset('comment.n.02')	Synset('comment.n.02')	Synset('gossip.n.02')	True	False
years	Synset('long_time.n.01')	Synset('long_time.n.01')	Synset('year.n.03')	True	False
frontage	Synset('frontage.n.01')	Synset('frontage.n.01')	Synset('frontage.n.02')	True	False
study	Synset('survey.n.01')	Synset('discipline.n.01')	Synset('study.n.09')	False	False
language	Synset('language.n.01')	Synset('language.n.01')	Synset('terminology.n.01')	True	False
banks	Synset('bank.n.03')	Synset('bank.n.01')	Synset('depository_financial_institution.n.01')	False	False
dust	Synset('dust.n.01')	Synset('dust.n.01')	Synset('dust.n.03')	True	False
corporation	Synset('corporation.n.01')	Synset('corporation.n.01')	Synset('corporation.n.01')	True	True
planter	Synset('planter.n.01')	Synset('planter.n.01')	Synset('planter.n.01')	True	True
life	Synset('life_class.n.01')	Synset('liveliness.n.02')	Synset('liveliness.n.02')	False	False
pastor	Synset('curate.n.01')	Synset('curate.n.01')	Synset('pastor.n.02')	True	False
cow	Synset('cow.n.01')	Synset('overawe.v.01')	Synset('cow.n.02')	False	False
man	Synset('man.n.01')	Synset('man.n.05')	Synset('valet.n.01')	False	False
record	Synset('record.n.03')	Synset('record.n.03')	Synset('record.n.03')	True	True
collages	Synset('collage.n.01')	Synset('collage.n.01')	Synset('collage.n.01')	True	True
morning	Synset('morning.n.01')	Synset('morning.n.01')	Synset('morning.n.01')	True	True
trouble	Synset('fuss.n.02')	Synset('fuss.n.02')	Synset('worry.n.02')	True	False
blowing	Synset('blow_gas.n.01')	Synset('blowing.n.01')	Synset('blowing.n.01')	False	False
way	Synset('way.n.04')	Synset('way.n.04')	Synset('way.n.11')	True	False
amateur	Synset('amateur.n.01')	Synset('amateur.s.01')	Synset('amateur.n.02')	False	False
difference	Synset('difference.n.01')	Synset('difference.n.01')	Synset('difference.n.01')	True	True
reason	Synset('reason.n.01')	Synset('rationality.n.01')	Synset('reason.n.06')	False	False
hill	Synset('hill.n.01')	Synset('hill.n.01')	Synset('hill.n.01')	True	True
starlet	Synset('starlet.n.01')	Synset('starlet.n.01')	Synset('starlet.n.01')	True	True
Large	Synset('battalion.n.02')	Synset('large.s.06')	Synset('large.n.01')	False	False
work	Synset('work.n.02')	Synset('work.v.03')	Synset('work.n.05')	False	False
	Correttezza:	24 su 50	13 su 50	48%	26%

Di seguito il risultato finale (media sulle 10 iterazioni)

```
Accuratezza media 'my_LESK': 39.2%

Accuratezza media 'nltk_LESK': 19.8%
```

Conclusioni

L'accuratezza media su 10 esecuzioni si aggira intorno al `39%`

Una possibile spiegazione di tale accuratezza risiede nella limitata lunghezza di alcune frasi presenti nel corpus che, insieme all'approccio BOW (*bag of words*), non forniscono un contesto sufficiente a differenziare i sensi.

Inoltre, nel corpus, il POS tag dei termini risulta seppur lievemente differente in certi casi rispetto al POS tag definito da NLTK, questo fa sì che alcuni termini contrassegnati come `NN` sul corpus non vengano riconosciuti come tali (ma per esempio come `ADJ`, ecc..)