Nazareno Borello Marco Galatola a.a. 2019/2020

Relazione TLN - parte 3

Relazione TLN - parte 3
- Esercizio 1 (Overlapping)
- Esercizio 2 (Content to Form)
- Esercizio 3 (Hanks)
- Esercizio 4 (Text Segmentation)
- Esercizio 5 (Open Information Extraction)

Esercizio 1 (Overlapping)

L'obiettivo di questa esercitazione è il calcolo della similarità fra varie definizioni dello stesso concetto.

Le definizioni sono state raggruppate in array. Da ognuna sono state rimosse punteggiatura e *stopword* per poi applicare lo *stemming*.

Inserendo le parole ottenute in un insieme è stata costruita una *bag of words* per definizione.

Le bow associate ad ogni concetto sono poi comparate fra di loro secondo la seguente formula:

Il punteggio di un insieme di definizioni equivale alla media fra le similarità.

```
sim_scores = []
for d in defs:
    score = 0
    for i in range(len(d) - 1):
        for j in range(i + 1, len(d)):
            score += sim(d[i], d[j])
    sim_scores.append(score / (len(d) * (len(d) - 1) / 2))
```

I risultati sono stati aggregati secondo la natura del concetto.

	Generico	**Specifico**
Concreto	0.248775	0.158048
Astratto	0.087255	0.140196

Esercizio 2 (Content to Form)

L'obiettivo di questa esercitazione è l'estrazione di un synset Wordnet a partire da un insieme di definizioni.

Le definizioni di ogni concetto sono state raggruppate in array. Da ognuna sono state rimosse punteggiatura e *stopword* per poi applicare lo *stemming*.

Inserendo le parole ottenute in un insieme è stata costruita una *bag of words* per concetto.

Ogni bow è stata poi comparata con quella di vari synset, ottenuta partendo da definizione, esempi e lemmi di questi ultimi.

La misura di similarità tra due bow equivale a :

I synset vengono scelti tramite un'esplorazione *greedy* dell'albero di Wordnet partendo dalla radice, sfruttando un heap per mantenere i nodi in ordine di punteggio di similarità, la ricerca termina dopo 10000 step non miglioranti (numero ricavato tramite sperimentazione) od in caso si riveli esaustiva.

```
def content_to_form(definition):
    h = [] # heap
    i = 0 # not improving steps
    max_steps = 10000 # maximum not improving steps
    best_syn = wn.synset('entity.n.01')
    max_score = sim(definition, syn_set(best_syn))
    heappush(h, (-max_score, best_syn))

    while i < max_steps and h:
        (_, syn) = heappop(h)
        for hypo in syn.hyponyms():
            score = sim(definition, syn_set(hypo))
            heappush(h, (-score, hypo))
            if(score > max_score):
                best_syn = hypo
                max_score = score
                i = -1
        i += 1

    return (best_syn, max_score)
```

Di seguito sono riportati i risultati attesi e quelli ottenuti coi relativi punteggi di similarità.

Concetto atteso	Concetto ottenuto	SIM
Justice	Justice	0.75
Patience	Hard Times	1.0
Greed	Desire	1.0
Politics	Regulation	0.75
Food	Animal order	1.0
Radiator	Heater	0.75
Vehicle	Landing	0.86
Screw	Slice	0.83

Esercizio 3 (Hanks)

L'obiettivo di questa esercitazione è la creazione di cluster semantico relativo ad un verbo transitivo, la fonte selezionata è il Brown Corpus, il verbo scelto è *buy*.

Sono state selezionate dal corpus solo le frasi contenenti le parole *buy*, *buys* o *bought*.

Utilizzando la parsificazione a dipendenze di Spacy, sono state estratte le coppie, (soggetto, oggetto) relative al verbo, mantenendo solo quelle complete.

Gli elementi di ogni coppia sono stati disambiguati e tradotti in synset Wordnet utilizzando un'implementazione dell'algoritmo di Lesk.

```
def lesk(word, sentence): #bag of words approach at WD
    best_sense = []
    max_overlap = -1
    context = set(sentence.split())
    for s in wn.synsets(word):
        signature = set(s.definition().split())
        for e in s.examples():
            signature.update(e.split())
        overlap = len(context & signature)
        if overlap > max_overlap:
            best_sense = s
            max_overlap = overlap
    return best_sense
```

Ad ogni synset è stato assegnato un supersenso, noun.person è stato scelto come valore di default, per via del mancato riconoscimento dei nomi propri.

I risultati sono stati aggregati, la coppia (noun.person, noun.artifact) risulta dominante sulle altre.

Esercizio 4 (Text Segmentation)

L'obiettivo di questa esercitazione è la divisione automatica di un testo in paragrafi, la fonte selezionata è un articolo della Flat Earth Society: *Better and flatter earths*.

Il testo è stato diviso in frasi utilizzando l'apposito tokenizer presente di NLTK, sulle quali sono stati effettuati la rimozione delle *stopword* e lo *stemming*.

Tutte le parole rimanenti nel testo sono state inserite in un dizionario ed associate ad un indice per motivi d'efficienza.

Ad ogni frase è stato associato un count vector della lunghezza del dizionario.

Inizialmente il testo viene diviso in sezioni di 10 frasi, i punti di separazione vengono poi riaggiustati tramite una funzione, ad ogni sezione è associata la somma dei vettori delle frasi contenute.

```
improvement = True
while(improvement):
    improvement = False
    for i in range(len(separators)):
        siml = cos_sim(psents[separators[i] - 1], splits[i + 1])
        simr = cos_sim(psents[separators[i]], splits[i])
        if siml <= cos_sim(psents[separators[i] - 1], np.subtract(splits[i],
psents[separators[i] - 1])):
            siml = 0
        if simr <= cos_sim(psents[separators[i]], np.subtract(splits[i + 1],
psents[separators[i]])):
```

```
            simr = 0
        print(siml, simr)
        if siml != 0 or simr != 0:
            if(siml >= simr):
                splits[i] = np.subtract(splits[i], psents[separators[i] - 1])
                splits[i + 1] = np.add(splits[i + 1], psents[separators[i] - 1])
                separators[i] -= 1
            else:
                splits[i] = np.add(splits[i], psents[separators[i]])
                splits[i + 1] = np.subtract(splits[i + 1],
psents[separators[i]])
                separators[i] += 1
            improvement = True
```

Viene calcolata la similarità (cosine similarity) fra ogni frase liminale, la sezione cui appartiene e quella adiacente; le frasi sono poi spostate nella sezione a similarità maggiore.

Questo procedimento viene ripetuto finché si ottiene un miglioramento

Esercizio 5 (Open Information Extraction)

L'obiettivo di questa esercitazione è l'estrazione di informazioni da un testo sotto forma di triple, la fonte selezionata è la biografia di A. M. Turing.

Il testo è stato diviso in frasi utilizzando l'apposito tokenizer presente di NLTK, successivamente sono stati rimossi i link fra parentesi quadre.

Ognuna delle frasi ottenuta è stata parsificata secondo una struttura a dipendenze utilizzando Spacy.

Partendo dalla ROOT (il verbo principale) di ogni frase, considerata come l'elemento centrale della tripla, si espandono ricorsivamente gli elementi a sinistra ed a destra eliminando quelli superflui in base al tag assegnatogli (selezionati tramite sperimentazione).

```
def expand(tokens, left = False):
    sent = []
    taboo = ["advcl", "punct", "prep", "cc", "", "compound"] if left > 0 else
["advcl", "punct", "conj", "cc", ""]
    if tokens:
        for t in tokens:
            if t.dep_ not in taboo:
                sent += expand(t.lefts,True) + [t.text] + expand(t.rights)
    return sent
```

Le triple così ottenute sono salvate in un DataFrame Pandas, filtrabile via query.