Marco Ottina 795130
Davide Sgaramella 938632

A.A. 2019/2020

Università degli Studi di Torino – polo Scienze della Natura
Laurea Magistrale in Informatica
Corso: Tecnologie del linguaggio naturale
Parte di: Luigi Di Caro

Esercitazione 4

L'esercitazione consiste nell'effetuare la segmentazione di un documento, ossia la partizione di una sequenza di frasi in sottosequenze contigue. Si è scelto di non matenere paragrafi con cardinalità inferiore a tre (ossia avente solo una o due frasi), unendo tali paragrafi con quello ritenuto migliore tra quelli immediatamente circostanti, qualora esistano.

L'algoritmo è ispirato al *text tiling* ed è articolato in sotto-componenti. Di seguito sono illustrate le principali classi usate ed i relativi metodi. Un'anticipazione: l'esercitazione viene svolta istanziando una istanza della classe *DocumentSegmentator*, fornendo al costruttore la lista di frasi del documento, ed invocando il metodo d'istanza *document_segmentation*, fornendo il numero di paragrafi desiderato e previsto.

WordWeighted

Semplice classe che racchiude una parola, conta il numero di presenze di essa nel documento ed il peso accumulato. Fornisce il peso totale della parola, tramite la funzione *getTotalWeight,* il quale è calcolato con la forumula *term frequency*: *peso totale * 1 + log2(numero di presenze)*. Si ricorda che il *peso totale* è la somma dei pesi di tutte le occorrenze di una data parola nella *weighted_bag* di una frase.

CachePairwiseSentenceSimilarity

Classe che mantiene come cache i valori di similarità tra due frasi. In caso di assenza della coppia, ne calcola la similarità mediante una funzione fornita nel costruttore. Quest'ultima è di default la funzione *weighted_overlap*, della classe *DocumentSegmentator,* del tutto simile ad un'altra funzione analoga discussa nell'esercitazione 2 ma normalizzata sulla cardinalità dell'insieme più piccolo.

Paragraph

Classe che rappresenta astrattamente un paragrafo. Richiede un'istanza di *DocumentSegmentator* nel costruttore ed è definita dagli indici, inclusivi, di inizio e fine e relativi alla lista di frasi. Inoltre, definisce il campo *words_cooccurrence_in_bags_counter*, ossia una mappatura che associa ad ogni parola il numero di volte che compare nel paragrafo.
Tale campo è necessario nel calcolo della *coesione*, che avviene nella funzione *get_cohesion*: essa è il rapporto tra la somma del peso di tutte le parole che occorrono più di una volta e la dimensione del paragrafo stesso, espresso come numero di frasi. Il *peso* sopracitato è quello fornito dalla classe *WordWeighted*.
Inoltre, mantiene due puntatori: uno per il paragrafo precedente, l'altro per quello successivo. Infine, permette di aggiungere o rimuovere frasi agli estremi dello stesso e di "unire" una istanza con un'altra, mantenendo aggiornate le informazioni precedentemente riportate.

DocumentSegmenter

Questa è la classe pricipale, che realizza l'algoritmo di segmentazione del testo, ossia l'obiettivo dell'esercitazione. Per essere istanziata necessita della lista di frasi di cui è composto il documento. Mantiene tale lista, la lista delle parole utili e filtrate di ogni frase, il peso di ogni parola del documento, le opzioni necessarie alla ricerca dei *synset*, la cache di questi ultimi e poco altro ancora.

Di seguito si riportano le funzioni che realizzano l'algoritmo di segmentazione del testo:

- *document_segmentation*: realizza l'algoritmo di segmentazione del testo. Dapprima calcola i pesi di tutte le parole di una frase, invocando *get_weighted_word_map_for_sentence* ed utilizzando la classe *WordWeighted*. Poi calcola gli indici ove suddividere il documento invocando *doc_tiling* ed esegue la suddivisione, restituendo una lista di liste di frasi.
- *get_weighted_word_map_for_sentence*: invoca la funzione *weighted_bag_for_sentence* della libreria *synsetInfoExtraction* e ne memorizza i valori restituiti in apposite caches.
- *doc_tiling*: è la funzione più importante ed articolata di tutta l'esercitazione, composta da fasi di inizializzazione, esecuzione iterativa e processamento finale dei risultati intermedi.
  - Innazitutto, si definisce un paragrafo per ogni frase, istanziando la classe *Paragraph*. Segue poi la definizione della bozza di soluzione, ossia ad ogni paragrafo se ne associa un altro, tra quello precedente ed il successivo, avente maggiore similarità. Si impone al primo paragrafo l'associazione con il secondo ed all'ultimo col penultimo. Fatto ciò, si unisce ogni paragrafo con quello a cui è associato, formando paragrafi di almeno due frasi.
  - Completata l'inizializzazione, si procede all'algoritmo intermedio. In modo alternato, per ogni paragrafo, si verifica se la prima / ultima frase è più "idonea" in tale paragrafo o nel precedente / successivo. A tale fine, si calcola la *coesione* del paragrafo corrente e di quello precedente / successivo, sia iniziali sia provando a spostare tale frase nell'altro paragrafo (quello adiacente). In tale modo si verifica se mantenere o spostare il *break point*: se la somma delle coesioni dei due paragrafi citati fosse maggiore dopo che si spostasse il *break point* allora si conferma lo spostamento, altrimenti (ossia se la coesione fosse calata) si ripristina la situazione iniziale. Così facendo, si migliora l'approssimazione dei paragrafi, in quanto le frasi si addensano nei paragrafi molto coesi e quelli vuoti vengono eliminati.
  - Inoltre, si esegue una ultima operazione di pulizia: potrebbero essere generati dei paragrafi troppo “piccoli”: si è scelto *3* come cardinalità soglia minima. Ciascun paragrafo con una o due frasi è quindi unito ad un paragrafo contiguo. Per determinare il paragrafo contiguo più promettente, si calcola la *weighted similarity* e si sceglie quello con similarità maggiore.
  - Fatto ciò, si convertono i paragrafi in indici (i *break point*) utilizzati dalla funzione *document_segmentation* per segmentare il documento.

Risultati

Come risultati abbiamo mostrato dove sono stati trovati i break point, in questo modo abbiamo poi preso le frasi che compongono ogni paragrafo e mostrato anche la lunghezza delle finestre in questione. Così facendo si avranno delle finestre di diversa lunghezza/dimensione.
Sono stati analizzati quattro testi di diversa lunghezza: “cat_book”, “iceland”, “cambridge_analytica” e “brexit”.

- *“cat_book”*: tratta di una persona immaginaria che racconta del suo gatto, della passione per i libri e di qualche problema personale. I tre paragrafi prodotti, uno per argomento precedentemente citato, sono stati suddivisi abbastanza bene: vengono inclusi nel primo due frasi del secondo, il che è comprensibile osservando quante parole sono simili a quelle del primo paragrafo rispetto a quante lo siano del secondo.
- *“iceland”*: in questo semplice testo si parla dell’Islanda e del fenomeno dell’aurora boreale. Il testo descrive le origini del fenomeno e prosegue andando a descrivere come fare per vedere l’aurora. Il testo è stato suddiviso in 14 paragrafi su 77 frasi.
- *“cambridge_analytica”*: il testo in questione è un articolo del tabloid inglese “The Guardian” che descrive le dinamiche dello scandalo Cambridge Analytica dei primi mesi del 2018. Il testo è stato suddiviso in 16 paragrafi su 90 frasi.
- *“brexit”*: come per il testo precedente, in questo articolo di giornale si parla degli ultimi sviluppi sulla questione brexit e del rapporto tra UE e UK, in vista dell’imminente uscita del Regno Unito dall’Unione Europea. Il testo è stato suddiviso in 12 paragrafi su 44 frasi.

Mediamente, i paragrafi prodotti hanno una cardinalità variabile tra le 4 e le 6 frasi, con rari estremi di 3 e 8 frasi. Ciò può essere spiegato analizzando l'estrazione dell'informazione contenuta nelle frasi e nei paragrafi ed il modo in cui questi sono confrontati, ossia le funzioni di coesione e similarità. Se l'estrazione dell'informazione fosse più precisa, ricca e/o descrittiva ed i confronti fossero meno *naive*, allora le frasi potrebbero meglio agglomerarsi in paragrafi più corposi perchè verrebbe individuate più precisamente quanto le frasi sono collegate alle altre.

Codice condiviso

Una porzione consistente di codice è stata definita nel package *utilities* al fine di essere riciclato e di snellire il file sorgente dell'esercitazione, così da ridurre il “rumore” e focalizzarsi meglio sull'algoritmo e sulle funzioni e strutture d'appoggio. Molto del codice condiviso è stato descritto nella relazione sull'esercitazione 2 e nessun'altra funzionalità degna di nota è stata aggiunta nel codice condiviso.