Laboratorio di TLN parte terza

Giacometti Lorenzo[915752], Giovanale Matteo[926431], and Daniel Haures[919707]

Università degli Studi di Torino
Dipartimento di Informatica

Abstract. Implementazione dell'algoritmo di TextTiling per la segmentazione di testi in base al contenuto semantico.

1 Introduzione

In questo paper, presentiamo un esercizio di programmazione volto all' implementazione dell' algoritmo di TextTiling per segmentare il testo di alcuni documenti.

L' obiettivo è esplorare le capacità di TextTiling nel rilevare le transizioni tematiche e valutare la sua efficacia in diversi contesti documentali. Dopo una breve panoramica dell'algoritmo, descriviamo in dettaglio il processo di implementazione, includendo le scelte progettuali e implementative. Successivamente, discutiamo i risultati ottenuti, mediante l'applicazione dell' algoritmo, su un insieme di documenti di prova e forniamo un' analisi delle performance del nostro codice. Per l' implementazione dell'algoritmo di TextTiling ci siamo basati sul paper di Marti A. Hearst [1].

2 Implementazione

Il nostro programma prende in input una serie di documenti, ognuno di essi viene diviso secondo i 'tagli' suggeriti dal nostro algoritmo di TextTiling. Prima però di andare ad eseguire l' algoritmo vero e proprio, prendiamo ciascun documento e lo suddividiamo in blocchi tramite la funzione *extrack blocks(N_BLOCK,doc)*, dove ogni blocco avrà al suo interno esattamente N_BLOCK frasi.

Successivamente andiamo a calcolare un punteggio di similarità (lexical score) sui blocchi, prendendoli a due a due, tramite la funzione *lexical similarity fun()*.

2.1 lexical_similarity_fun(block1, block2)

Questa funzione, prima di fare la cosine similarity, rimuove le stopwords, segni di punteggiatura e numeri dai testi del blocco 1 e del blocco 2. Successivamente crea un dizionario contenente tutti i termini e le loro relative frequenze dei due blocchi. Quest'ultimo passaggio è fondamentale per calcolare lo score di similarità secondo la formula:

$$score(i) = \frac{\sum w_{t,b1} w_{t,b2}}{\sqrt{\sum w_{t,b1}^2 \sum w_{t,b2}^2}}$$

Per applicare questa formula abbiamo creato una lista contenente tutte le parole all'interno dei due blocchi. Iteriamo su ogni singolo termine della lista andando a misurare il match di quella parola all'interno dei due blocchi.

2.2 Variante Embeddings

Abbiamo creato anche una funzione alterativa per il calcolo del lexical score chiamata *semantic_ similarity_ fun(block1, block2)*. Per ogni frase contenuta dai due blocchi verrà calcolato un embedding attraverso l'utilizzo della API Sentence Transformer [2]. Sulle due matrici così ottenute verrà calcolato il prodotto scalare vettore-vettore (ossia embedding-embedding). La funzione ritornerà la similarità massima raggiungibile tra i due blocchi, restituendo il massimo valore di prodotto scalare tra due frasi appartenenti a blocchi diversi.

2.3 get_boundaries(similarities,block_len, SECTION_SENS)

Raccolte tutte le misure di similarità in una lista andiamo a calcolare i 'tagli' da effettuare nel nostro documento, previa azione di smoothing sugli score ottenuti. Ogni coppia di valori nella lista andrà a rappresentare un 'gap' tra due blocchi del testo.

Nella prima parte di *get_ boundaries()* andiamo a calcolare i cosiddetti 'depth score' per ogni singolo 'gap'. Essi sono dei punteggi che misurano il cambio di argomento tra i due lati del 'gap' dati i loro lexical score. La formula utilizzata è:

$$depthscore(a2) = (y_{a1} - y_{a2}) + (y_{a3} - y_{a2})$$

Successivamente andiamo a calcolare una misura di treshold, sopra la quale consideriamo il nostro 'depth score' come un 'taglio' nel documento. Questa divisione corrisponde ad un cambio di argomento tra i due blocchi.

Infine abbiamo implementato una funzione di nome *boundaries_ error()* che confronta i tagli che abbiamo ottenuto con quelli attesi e restituisce un punteggio che indica se il taglio coincide oppure di quanto si discosta da quello atteso.

3 Risultati

I risultati che più si avvicinano a quelli attesi li abbiamo ottenuti andando a considerare una singola frase all'interno di ogni blocco e mettendo il valore che regola il livello di smoothing del similarity score a 0,7 .

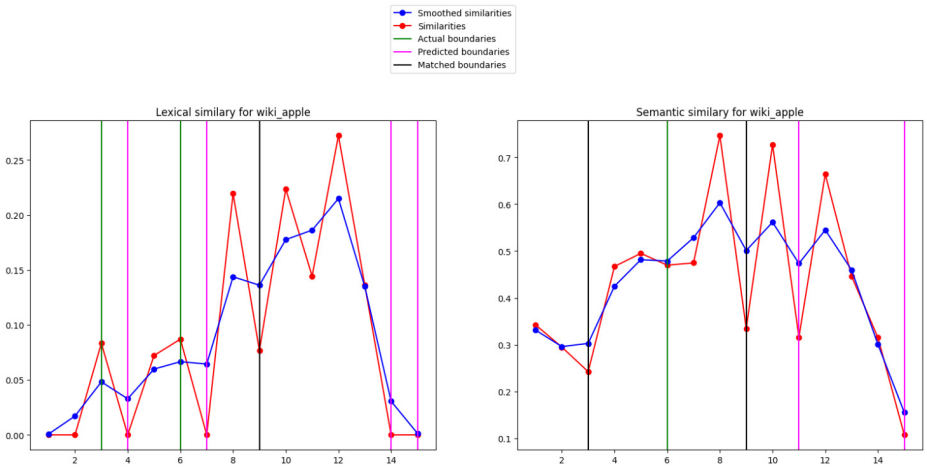

Fig. 1. risultati ottenuti relativi al documento 'wiki apple'

Come si vede nell'immagine soprastante (Fig. 1), abbiamo deciso di evidenziare in nero i tagli applicati correttamente, in viola quelli predetti e in verde quelli attesi.

Dai test fatti siamo giunti a conclusione che, i risultati migliori, sono stati ottenuti utilizzando gli embeddings e il relativo prodotto scalare, piuttosto che andando a calcolare il lexical score tramite la cosine similarity. Qua di seguito mostriamo una tabella contenente gli errori commessi dai due approcci su un totale di sei documenti:

Table 1. Accuracy del nostro programma

documenti	*L match error*	*S match error*	*L n* of section error*	*S n* of section error*
apple	2	3	2	1
paxton	2	3	13	7
paper	2	1	2	3
punk	3	5	2	0
I.T.	8	3	4	0
emotions	4	0	2	1
totale	21	15	25	12

References

1. Author, Hearst, Marti A.: Article title. Text Tiling: Segmenting Text into Multi-paragraph Subtopic Passages (1997)
2. Sentence Bert Transfotmer documentation page: https://sbert.net