Andrea Mattone 882956 andrea.mattone@edu.unito.it
Calogero Jerik Scozzaro 883080 calogero.scozzaro47@edu.unito.it

TLN - 3° parte del corso
RELAZIONE

Esercizio 1 - Defs

I quattro termini scelti sono: emotion, person, revenge e brick.
Inizialmente carichiamo tutte le definizioni eliminando le stop words.
Successivamente creiamo, per ogni termine, un dizionario con

- chiave: parola presente in una definizione di quel termine
- valore: numero di occorrenze di quella parola nelle definizioni di quel termine

Per ogni termine abbiamo creato una funzione calculate_overlap che restituirà lo score.
In questa funzione ordiniamo le parole del dizionario in ordine decrescente in base al valore e salviamo le 5 più frequenti. Per ognuna di queste 5, contiamo il numero di occorrenze nelle definizioni e lo sommiamo. Questo passaggio serve per contare una volta la parola anche se presente più volte in una definizione.
Lo score verrà poi normalizzato dividendo il valore trovato per il numero di definizioni (35) e il numero delle parole più frequenti utilizzato (5).
Di seguito è presente uno screen dei risultati ottenuti.

```
           Astratto   Concreto
Generico   0.28       0.36
Specifico  0.23       0.44
Valore medio per concreti: 0.4
Valore medio per astratti: 0.26
```

Esercizio 2: content2form

In questo esercizio facciamo la ricerca onomasiologica. Usando le parole contenute nelle definizioni e WordNet cerchiamo di trovare il termine di partenza.
Quindi usiamo le definizioni come input per andare su WordNet e trovare il synset giusto.
Diamo uno score ad ogni synset trovato e restituiamo i 5 synset con lo score più alto.

Per ogni termine abbiamo creato una funzione try_to_find_term nella quale cerchiamo gli iponimi delle 5 parole più frequenti nelle definizioni di quel termine.
Successivamente, per ogni lista di iponimi cerchiamo il synset con lo score più alto utilizzando una funzione che calcola l'overlap tra il gloss del synset e le nostre definizioni.
Di seguito è presente uno screen dei risultati ottenuti.

```
Emotion:
(Synset('passion.n.01'), Synset('world.n.08'), '', Synset('life.n.03'), '')
*******
Brick:
(Synset('commemorative.n.01'), Synset('transparent_substance.n.01'), Synset('house-raising.n.01'), '', Synset('outbuilding.n.01'))
*******
Person:
(Synset('world.n.08'), Synset('man.n.03'), '', Synset('man.n.03'), Synset('line-drive_single.n.01'))
*******
Revenge:
(Synset('umbrage.n.01'), Synset('affect.n.01'), Synset('alienation.n.04'), Synset('neutralization.n.02'), Synset('hate.n.01'))
```

In alcuni termini non sono presenti 5 synset. Questo perché alcune delle parole più frequenti nelle nostre definizioni non sono presenti in WordNet.

Esercizio 3: Hanks

Per lo svolgimento di questo esercizio abbiamo scelto il verbo “won”.
Inizialmente salviamo il dataset in una lista.
Successivamente utilizziamo la funzione dependency_finder() nella quale cerchiamo 200 occorrenze di won con 2 argomenti (subj e obj) e per ogni termine trovato salviamo il supersenso.
Inoltre, creiamo due dizionari (dict_nsubj e dict_dobj) nel quale salviamo le occorrenze di ciascun supersenso e un dizionario (dict_tot) nel quale salviamo le occorrenze delle coppie.
Di seguito è presente uno screen dei risultati ottenuti.

```
Frequenza nsbuj:
noun.person  111
noun.location  10
noun.substance  10
noun.group  6
noun.artifact  6
noun.communication  2
noun.time  1
noun.plant  1
noun.quantity  1
noun.body  1
noun.object  1
```

```
Frequenza dobj:
noun.communication  90
noun.event  18
noun.cognition  17
noun.group  4
noun.location  4
noun.possession  3
noun.artifact  3
noun.act  3
noun.person  2
noun.substance  1
noun.quantity  1
```

```
Frequenza coppie:
noun.person-noun.communication  85
noun.person-noun.cognition  17
noun.location-noun.event  7
noun.person-noun.location  4
noun.group-noun.event  4
noun.substance-noun.group  3
noun.person-noun.event  2
noun.substance-noun.communication  2
noun.substance-noun.event  2
noun.time-noun.possession  1
noun.person-noun.possession  1
noun.plant-noun.event  1
```

Esercizio 4: Segmentation

Per questo esercizio abbiamo utilizzato un dataset diviso in 3 topic:

- Da riga 1 a 20, il topic è l’ebola
- Da riga 21 a 30, il topic è il “Trump wall”
- Da riga 31 a 36, il topic è Napoleone

Inizialmente dividiamo il dataset in 3 segmenti di lunghezza uguale, quindi la segmentation sarà:

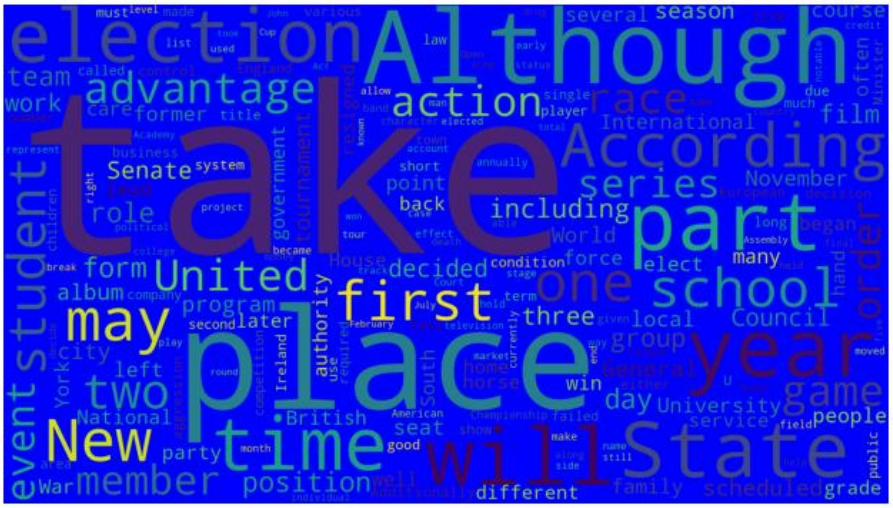
- Da riga 1 a 12
- Da riga 13 a 24
- Da riga 25 a 36

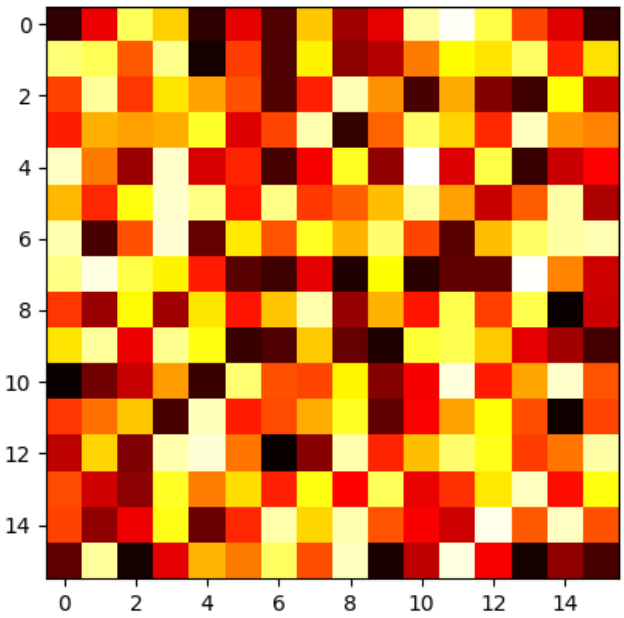
Successivamente utilizziamo la funzione calculate_segmentation().
In questa funzione è presente un ciclo che si ripete 1000 volte, nel quale tramite una funzione random (con valori possibili da 1 a 20) cerchiamo di spostare i “tagli”.
Per valutare i “tagli” è presente una funzione calculate_overlap() che calcola l’overlap tra i termini di due frasi successive presenti all’interno della stessa segmentation. Lo score della segmentation iniziale è 208.
Di seguito è presente uno screen con i risultati di 3 esecuzioni consecutive.
Si può notare come il programma riesca ad individuare bene il terzo topic, mentre fa un po’ di confusione sui primi due.

```
Segmentation iniziale: [12, 24, 36]
Score iniziale: 208
********************
Segmentation finale: [26, 31, 36]
Score finale: 437
PS C:\Users\Jerik\Desktop\I anno Magistrale\
rik/Desktop/I anno Magistrale/TLN/Di Caro/es
Segmentation iniziale: [12, 24, 36]
Score iniziale: 208
********************
Segmentation finale: [22, 31, 36]
Score finale: 391
PS C:\Users\Jerik\Desktop\I anno Magistrale\
rik/Desktop/I anno Magistrale/TLN/Di Caro/es
Segmentation iniziale: [12, 24, 36]
Score iniziale: 208
********************
Segmentation finale: [27, 31, 36]
Score finale: 436
```

Esercizio 5: TM/TV

Abbiamo svolto Topic Visualization.
Abbiamo utilizzato wordcloud e heatmap





Esercizio 6: Plagiarism detection

Inizialmente salviamo la data di creazione dei due file. Questo perché, se c'è un plagio, sicuramente sarà stato effettuato da chi ha creato il file più tardi.
Successivamente, tramite la funzione check_plagiarism() controlliamo se c'è un plagio.
In questa funzione viene calcolata il numero di variabili e funzioni con nome uguali, la size dei due file e l'owner dei due file e viene assegnato un punteggio di plagio.
Se questo punteggio supera una determinata soglia evidenzia una possibilità di plagio.

```
L'accusato di plagio è il file codeB.py
Possibilità di plagio: True
```

Esercizio 7: False friends

Utilizziamo come dataset di input un vettore nasari embedded dal quale estraiamo i termini.
Successivamente, per ogni termine cerchiamo delle parole simili come stringhe ma diverse come semantica. Per calcolare la semantic similarity utilizziamo la funzione shortest_path_nltk e salviamo i due termini solo se la loro max similarity è minore di una

soglia molto bassa (0.15). In questo modo salviamo solo i termini con significati molto diversi.

```
False friends:
{'million': 'mullion', 'quarto': 'quartz', 'hour': 'honour', 'octavo': 'octave', 'adenine': 'adenosine', 'desertion': 'exertion', 'slaughterhouse': 'slaug
htered', 'abbreviation': 'aviation', 'abdication': 'addiction', 'weaning': 'weakling', 'ablation': 'salvation', 'denial': 'genial', 'abortion': 'reabsorpt
ion', 'abridgement': 'debridement', 'repeal': 'reel', 'absolution': 'solution', 'academia': 'macadamia', 'academician': 'academia', 'academy': 'academia',
 'dialect': 'dialectic', 'injury': 'jury', 'skill': 'swill', 'pact': 'impact', 'ledger': 'leer', 'adept': 'cadet', 'acetate': 'activate', 'ketosis': 'kera
tosis', 'vinegar': 'vineyard', 'acorn': 'actor', 'acre-foot': 'barefoot'}
```

Esercizio 8: FCA

I concetti che abbiamo scelto sono: coccodrillo, gallina, cane e cinghiale.
Le proprietà che abbiamo scelto sono: pericoloso, addomesticato, edibile e mammifero.

```
c = Context.fromstring(
    '''
                |coccodrillo|gallina|cane|cinghiale |
pericoloso      |  x        |       |    |     x    |
addomesticato   |           |       |  x |          |
edibile         |           |  x    |    |     x    |
mammifero       |           |       |  x |     x    |
'''
)
```