Università degli Studi di Torino

FACOLTÀ DI INFORMATICA
Corso di Laurea Magistrale in Informatica

Tecnologie del linguaggio naturale

Relazione esercitazioni

Candidato: Iodice Francesco
Docente: Luigi Di Caro

Indice

Similarity

Consegna: dati 4 termini differenziati in base alla loro concretezza e specificità:

- building --> [Concreto][Generico]
- molecule --> [Concreto][Specifico]
- freedom --> [astratto][Generico]
- compassion --> [astratto][specifico]

E circa 20 definizioni diverse per ogni termine.

1. Caricamento dei dati sulle definizioni (file definizioni.xls o google doc presente su Moodle)

2. Preprocessing (su frequenza minima dei termini, stemming, etc. a vostra scelta)

3. Calcolo similarità tra definizioni (cardinalità dell'intersezione dei termini normalizzata su lunghezza minima tra le due, o varianti a scelta)

4. Aggregazione sulle due dimensioni (concretezza / specificità come da schema in basso)

5. Interpretazione dei risultati e scrittura di un piccolo report (da inserire nel vostro portfolio per l'esame

Implementazione

I dati sono salvati in una matrice dove le righe indicano i diversi concetti e le colonne le diverse definizioni per ogni concetto.
Sono state realizzate 2 strutture dati per poter eseguire calcoli statistici sulla matrice di dati:

- $\boldsymbol{sts\_all\_defs}$: $k \rightarrow [(t_1, f_1), ..]$ una mappa dove:
  - **Key**: k rappresenta il termine da descrivere
    - (Esempio k = **concreto_generico_building : building**)
  - **Value**: la lista di coppie $(\boldsymbol{t_1}, \boldsymbol{f_1})$ dove t_1 è un qualsiasi termine usato per descrivere k e f è il numero di occorrenze del termine t in ogni definizione di k.

- $\boldsymbol{topic\_single\_def}$: $k \rightarrow [d_1, d_2, \dots d_{20}]$ realizzata a partire da **sts_all_defs** una mappa dove:
  - **Key**: k rappresenta il termine da descrivere
  - **Value**: $\left[d_1 = (t_1, t_2 \dots t_x), d_2 = (t_1, t_2 .. t_j), \dots d_{20} = (t_1, t_2 .. t_k)\right]$ è una lista di liste di termini. In ogni elemento i-esimo ci sono i termini **rilevanti** usati nella definizione i-esima per descrivere k.
    I termini in ogni lista sono ordinati in base alla loro occorrenza su tutte le definizioni usate per descrivere uno stesso termine.
      - $(t_1, t_2, t_3 \dots)$ t_1 ha una maggiore frequenza di t_2 così come t_2 ha una maggiore frequenza t_3

Durante la creazione delle mappe viene effettuato preprocessing (lemmatizzazione + rimozione stopword) sui termini.

La similarità tra le definizioni di uno stesso termine $\boldsymbol{k}$ è come segue:

1. Da **topic_single_def** vengono estratte tutte le liste di parole associate al termine k.

2. Per ogni coppia di definizioni viene calcolato l'intersezione tra le 2 (i termini che condividono), indichiamo questo valore con $\mathbf{s} = \text{sim\_defs}(\text{def}_1, \text{def}_2)$. Se i 2 insiemi hanno cardinalità diversa essa viene normalizzata sulla lunghezza minima dei 2.
   I termini più rilevanti vengono mantenuti grazie all'ordinamento effettuato in precedenza.

$$sim\_defs(def_1, def_2) = 100 \frac{def_1 \cap def_2}{min\ (|def_1|, |def_2|)}$$

3. La similarità delle definizioni per il termine $\boldsymbol{k}$ è uguale alle medie delle similarità $\boldsymbol{s}$ per ogni coppia di definizioni che descrivono $\boldsymbol{k}$, se abbiamo *d* definizioni:

$$Sim\_term(k) = \frac{1}{d} \sum_{i=i}^{d} \sum_{j=i+1}^{d} sim\_defs(\text{def}_i, \text{def}_j)$$

Risultati

I risultati sono poi illustrati di seguito:

Risultati	Astratto	Concreto
Generico	7.18 %	19.5 %
Specifico	12.21 %	12.35 %

Di seguito una variante dei valori di similarità ottenuta effettuando stemming delle parole nelle definizioni invece che lemmatizzazzione.

Risultati	Astratto	Concreto
Generico	7.48 %	23.85 %
Specifico	7.96 %	13.72 %

Come potevamo aspettarci le percentuali di similarità sono maggiori nelle definizioni dei concetti concreti, in quanto si usano molto frequentemente le stesse parole o simili per descriverli.

Content to form

Consegna

Date 12 definizioni diverse per un singolo concetto inferirlo automaticamente.

1. Caricamento dei dati content-to-form
2. Preprocessing (a vostra scelta)
3. Utilizzo di WordNet come sense inventory, per inferire il concetto descritto dalle diverse definizioni
4. Definire ed implementare un algoritmo (efficace ma anche efficiente) di esplorazione dei sensi di WordNet, usando concetti di similarità (tra gloss e definizioni, esempi d'uso, rappresentazioni vettoriali, etc.)
    - Suggerimento A: sfruttare principi del genus-differentia
    - Suggerimento B: sfruttare tassonomia WordNet nell'esplorazione
    - Suggerimento C: pensare a meccanismi di backtracking

Implementazione

Di seguito una breve descrizione testuale dell'algoritmo implementato

***Input** : Lista definizioni per il termine w*
***Output**: w o un concetto semanticamente legato a w*

1. *Estrazione termini rilevanti e soggetti dalle definizioni*
2. *Estrazione dominio e contesto*
    - ***dominio**: possibili iperonimi usati per descrivere w*
    - ***contesto**: termini rilevanti presenti nelle definizioni*
3. *Calcolo delle frequenze dei domini e contesti in modo da operare unicamente su quelli più frequenti*
4. *Ricerca in Wordnet su ogni termine presente nel **dominio**, si itera su tutti i termini del dominio, dato un termine si guardano tutti i sensi associati al synsets di wordnet e per ogni senso si valutano tutti gli iponimi*
5. *Viene calcolata la similarità tra la definizione del synset e la stringa ottenuta dalla concatenazione di tutti i termini nel **contesto**. Il synset con il valore massimo di similarità viene resituito.*

Algoritmo di ricerca

1. Estrazione termini rilevanti
Da ogni definizione vengono estratti:

- **Soggetti**: effettuando Pos Tagging + Parsing su ogni definizione
- **Termini rilevanti**: tutti i termini presenti nelle definizioni eliminando eventuali stopword e simboli di punteggiatura.

2. Estrazione dominio e contesto
Vengono costruite 2 liste:

- $contex$ = i termini rilevanti della definizione
- $domain$ = soggetto relativi alla definizione corrente unito a tutti i domini (estratti da wordnet) legati ai termini presenti nell'insieme delle parole rilevanti.

I punti 1 e 2 vengono eseguiti su ogni definizione per il termine w. Gli insiemi $contex$ e $domain$ sono cumulativi, ad ogni iterazione su una definizione diversa vengono estesi con i nuovi termini rilevanti e i nuovi domini ottenuti dalla definizione corrente.

3. Calcolo delle frequenze
Date le liste contex e domain vengono calcolati 2 insiemi:

- $context$ = i contesti più frequenti sulla frequenza dei termini in $contex$
- $candidate\_genus$ = i domini più frequenti sulla frequenza dei termini in domain

Vengono mantenuti solo i domini e contesti più frequenti, in base ai seguenti parametri:

- $max\_term\_in\_context$
- max_genus

4 Ricerca in Wordnet
Viene costruita la stringa:

- $s\_context$ = la concatenazione dei vari lemmi di ogni parola nell'insieme $context$.

La ricerca è di tipo top-down, l'algoritmo itera su ogni termine w presente nell'insieme $candidate\_genus$, si valutano tutti i sensi associati a w e tutti gli iponimi di ogni senso.

5. Similarità
La valutazione di ogni senso avviene calcolando la similarità tra la definizione del senso e la stringa $s\_context$, la misura di similarità adottata utilizza è la cosin similarity tra vettori embedding relativi ai termini, offerta da *spaCy*.

- Lo score associato a ciascun synset è il valore della similarità.

Durante la ricerca vengono aggiornati 2 valori:

- $best\_score$ = punteggio da 0 a 1
- $best\_synset$ = synset relativo allo score

Dopo avere iterato su ogni termine nell'insieme $candidate\_genus$ viene ritornato il synset con lo score maggiore.

Risultati

Target term	**Obtained term**	**Definition term**
politics	Synset('morality.n.01')	concern with the distinction between good and evil or right and wrong; right or good conduct
patience	Synset('unresponsiveness.n.01')	the quality of being unresponsive; not reacting; as a quality of people, it is marked by a failure to respond quickly or with emotion to people or events
greed	Synset('longer.n.01')	a person with a strong desire for something
justice	Synset('law.n.02')	legal document setting forth rules governing a particular kind of activity
food	Synset('water.n.06')	a liquid necessary for the life of most animals and plants
screw	Synset('piezoelectricity.n.01')	electricity produced by mechanical pressure on certain crystals (notably quartz or Rochelle salt); alternatively, electrostatic stress produces a change in the linear dimensions of the crystal
vehicle	Synset('transportation.n.02')	the act of moving something from one location to another
radiator	Synset('transportation.n.02')	the act of moving something from one location to another

L'algoritmo riesce a trovare concetti molti simili a quelli target, come nel caso di:

- politics : Synset('morality.n.01')
- patience : Synset('unresponsiveness.n.01')
- justice : Synset('law.n.02')
- food : Synset('water.n.06')
- vehicle : Synset('transportation.n.02')

La similarità tra 2 termini si basa unicamente sulla definizione presente in wordnet del synset che sta valutando, di conseguenza se è scarsa di contenuto la valutazione è bassa e l'algoritmo non riesce a trovare il synset appropriato.

Nel file *result.txt* sono presenti informazioni aggiuntive sui risultati, quali:

- il contesto per ogni termine
- i vari candidati genus esplorati nella ricerca

Hanks theory

Consegna: Implementazione teoria di P. Hanks

1. Scegliere un verbo transitivo (min valenza = 2)
2. Recuperare da un corpus n istanze in cui esso viene usato
3. Effettuare parsing e disambiguazione
4. Usare i super sensi di WordNet sugli argomenti (subj e obj) del verbo scelto
5. Aggregare i risultati, calcolare le frequenze, stampare i cluster semantici ottenuti

   Un cluster semantico è inteso come combinazione dei semantic types (ad esempio coppie di sem types se valenza = 2)

Implementazione

Per l'esercitazione sono stati utilizzati i seguenti verbi:

- **build**
- **love**
- **eat**

Per recuperare n frasi in cui viene usato il verbo selezionato è stato utilizzato il corpus brown. Dato che la forma della parola relativa al verbo cambia in base alla coniugazione di quest'ultimo è stato necessario l'utilizzo del lemmatizzatore di Wordnet. Iterando su tutte le parole di una singola frase, per ogni parola viene calcolato il lemma e se è uguale alla forma base del verbo scelto la frase viene salvata.

Per ogni frase, con l'ausilio di spacy, viene effettuato il pos taggin + parsing, ottenendo i vari token collegati tramite dipendenze sintattiche. Iterando su tutti i token è possibile estrarre il **soggetto** e il **complemento oggetto** relativi al verbo della frase, in quanto sono token figli del token del verbo a cui sono legati.

Soggetto e Complemento oggetto vengono mappati con i relativi super sensi di Wordnet nel seguente modo:

- ***Nome proprio***: person
- ***Pronome*** : [I, you, she, he, we, they] : person oppure [It] : entity
- ***Parola generica***: viene utilizzato il WSD che data la frase e il termine generico tenta di restituire il senso del termine

I supersensi di **Soggetto** e **complemento oggetto** rappresentato rispettivamente i filler1 e filler2 dei verbi (in quanto tutti e 3 i verbi hanno valenza 2), le combinazioni dei tipi semantici sono salvati nei *semantic cluster*.

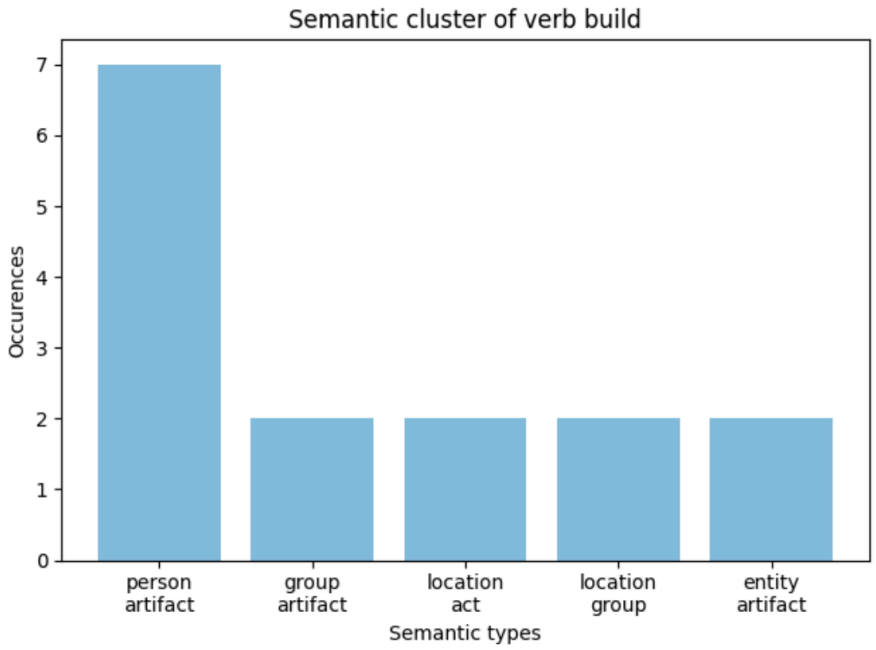
Semantic cluster contiene tutte le coppie **super_sense_subj:super_sense_dobj** estratte dalle frasi.

Vengono calcolate le frequenze delle coppie supersensi, ottenendo così per Filler1 e Filler2 i supersensi più frequenti.

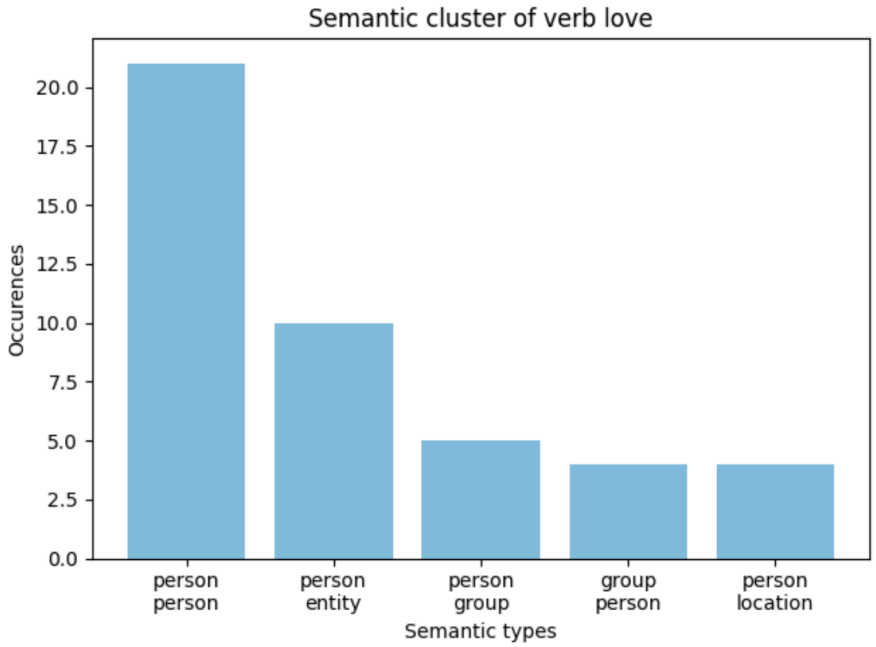
Risultati

I **cluster semantici** ottenuti sono visibili nei grafici sottostanti con le relative frequenze.

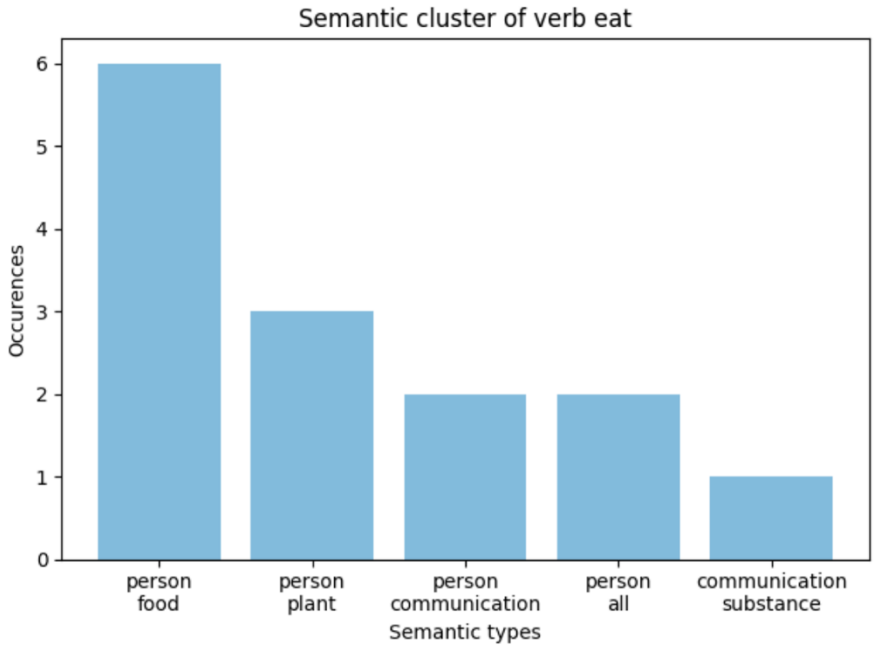
Semantic Type for verb BUILD



Semantic Type for verb LOVE



Semantic Type for verb EAT



Come si può vedere la coppia più frequente è quella più significativa per il significato più frequente del verbo:

- (person:person : 20) per il verbo amare
- (person:food : 7) per il verbo mangiare
- (person:artifact : 7) per il verbo costruire

Alcune coppie di semantic type, quella con frequenza minore, possono essere poco significative per il verbo in esame, ma questo può essere anche dovuto all'accuratezza del WSD che è pari al **68%**.

Text segmentation

Consegna**:** Ispirandosi al Text Tiling, implementare un algoritmo di segmentazione del testo. Usando informazioni come frequenze (globali, locali), co-occorrenze, risorse semantiche (WordNet, etc.), applicando step di preprocessing (as usual), etc.

Implementazione

Il task è stato realizzato eseguendo i seguenti step:

1. Windowing + Tokenizing
    - Trasformazione del testo in un insieme di N frasi.
    - Le frasi vengono raggruppate in finestre di lunghezza fisse, specificata dal parametro $size\_windows$, ottenendo così $N/size\_windows$ finestre della stessa lunghezza.
    - Ogni finestra viene 'tokenizzata' trasformando l'insieme di frasi al suo interno in una lista di parola rilevanti.

2. Cohesion ranking
    - Per ogni finestra i-esima viene valutata la similarità intragruppo.
        - La media fra la similarità tra la finestra $i-1$ e i e quella fra la finestra i e $i+1$

3. Search Break Point
    - Ricerca dei punti a bassa coesione circondanti da quelli ad alta coesione.

Saranno ora descritti i 3 step con maggiore dettaglio.

1. Windowing + Tokenizing
Il testo viene letto da file e trasformato in una lista di frasi, successivamente le frasi vengono raggruppate in finestre di lunghezza fissa e la cardinalità di ogni finestra è data dal parametro *size_windows*. Lo step successivo è quello di estrarre i termini rilevanti in ogni finestra filtrando stop word e rimuovendo i vari simboli di punteggiare.
La struttura dati risultati è una lista di finestra in cui in ogni elemento i-esimo sono presenti i termini rilevanti per la finestra i-esima.

2. Coesion ranking
Lo scopo di questo step è quello di calcolare la coesione di una finestra rispetto a quelle adiacenti (in quella precedente e in quella successiva), sulla base dei termini rilevanti presenti in ogni finestra.
Il valore di coesione di una finestra è dato da quanto le parole che occorrono nella finestra sono semanticamente simili alle parole che occorrono in quelle adiacenti, di conseguenza per ogni coppia di finestre adiacenti è stata calcolata la similarità semantica per coppia di parole presenti nelle finestre.

Date le finestre $A\ e\ B$ il valore di coesione è dato dalla seguente formula:

$$Cohesion(A,B) = \frac{10}{|A||B|} \sum_{i \in A} \sum_{j \in B} sim(w_i, w_j)$$

Dove:

- le due sommatorie iterano su ogni combinazione di parole presenti nelle finestre $A\ e\ B$
- $sim(a,b)$ è il valore di similarità fra parola a e b.
- Il prodotto delle cardinalità delle 2 finestre al denominatore serve come fattore di normalizzazione
- Il valore 10 al numeratore serve ad espandere il range dei valori

Come misura di similarità fra parole è stata utilizzata la **Weighted Overlap**: basata sulla versione **Lexical** di NASARI.

Il valore di similarità per la finestra i-esima viene calcolata come **media** dei seguenti valori:

- similarità tra la finestra i e i + 1
- similarità tra la finestra i e i – 1

Weighted Overlap

Il programma effettua parsing della versione Lexical di NASARI mantenendola in memoria per accedere alla rappresentazione vettoriale di una parola quando viene coinvolta nel calcolo della misura di similarità.

Per ogni finestra (corrente, precedente e successiva) vengono estratti i relativi vettori Nasari, ovvero una lista di vettori relativi a tutte le parole rilevanti presenti nella finestra.

Dati i vettori v1 e v2 viene calcolata l'intersezione tra i 2 e valutato il rank degli elementi comune, la formula è la seguente:

$$WO(v_1, v_2) = \frac{\sum_{q \in O} \left(rank(q, v_1) + rank(q, v_2) \right)^{-1}}{\sum_{i=1}^{|O|} (2i)^{-1}}$$

3. Search Break Point

Per determinare i punti in cui dividere il testo, in modo da determinare i paragrafi, si scorre la lista contenente i valori di coesione similarities. Una finestra è un buon candidato per un punto di split point se il suo valore di coesione è minore rispetto la media dei valori totali e se la somma delle similarità della finestra successiva e precedente è elevata.
Si cerca il miglior candidato, ovvero fra tutte le finestre con un valore di similarità basso (sotto la media) si prende quella la cui somma di similarità delle finestre adiacenti è massima. L'algoritmo procede iterativamente andando ad estrarre ad ogni iterazione il miglior minimo, la condizione di stop è che non ci sono più 'migliori minimi' da estrarre oppure è riuscito a trovare il numero target di split points.

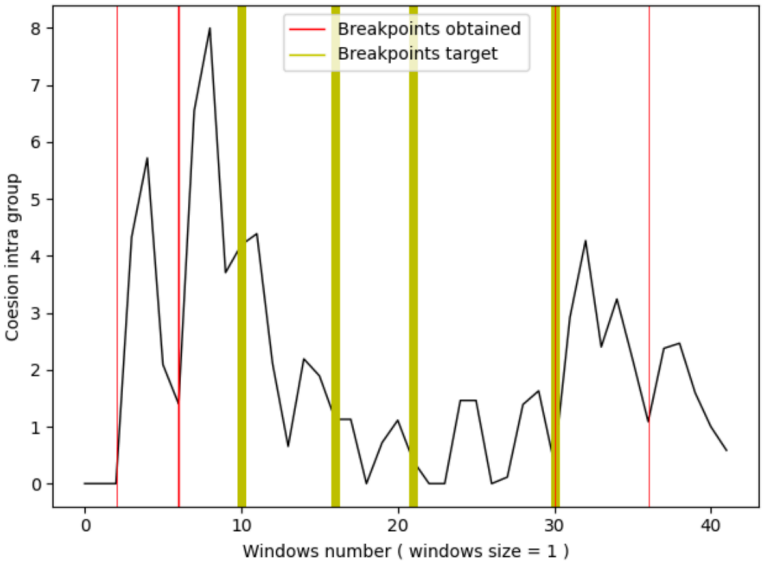
Risultati

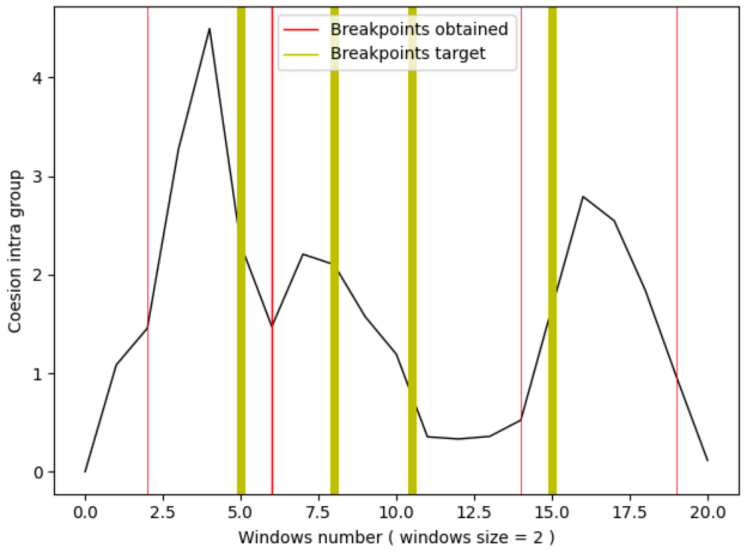
Il testo utilizzato è un breve biografia dell'attore Leonardo Di Caprio, presente in *utils/dicaprio_life.txt*.

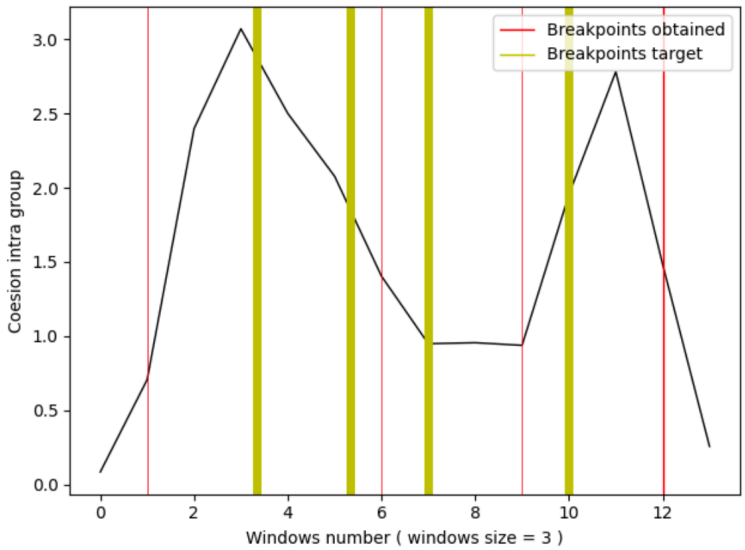
Sono stati effettuati diversi esperimenti variando la dimensione delle finestre. Nella variabile $break\_point\_sentence\_target$ ci sono gli indici delle frasi che indicano il cambio di discorso all'interno del testo.

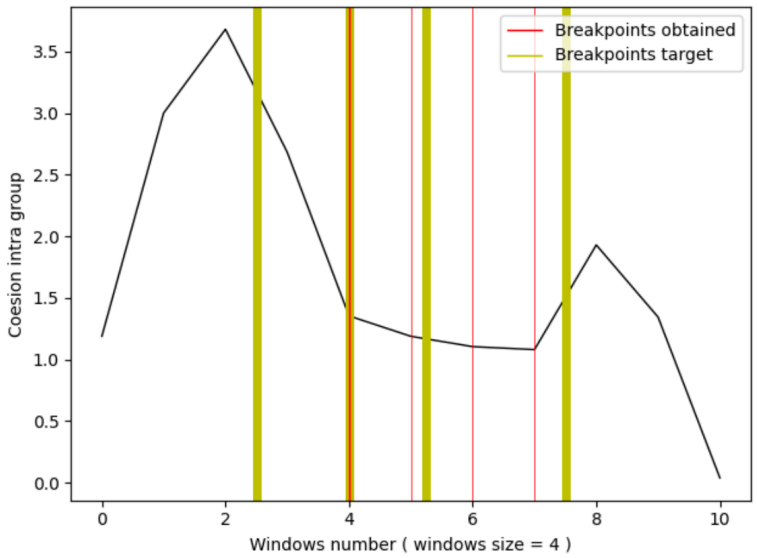
Di seguito sono indicati i risultati dei vari esperimenti:

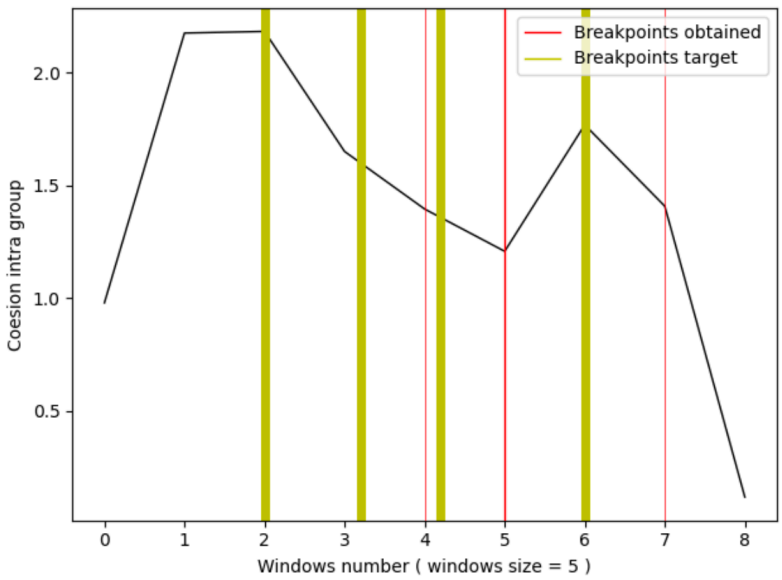
- **riga rossa: split points target**
- **riga gialla: split point ottenuto**






Breakpoints obtained
Breakpoints target
Coesion intra group
Windows number (windows size = 3)


Breakpoints obtained
Breakpoints target
Coesion intra group
Windows number (windows size = 4)


Breakpoints obtained
Breakpoints target
Coesion intra group
Windows number (windows size = 5)

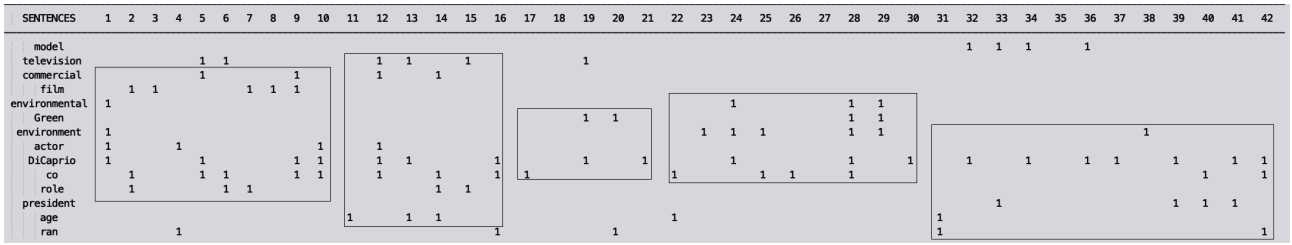
Durante l'esecuzione del programma gli indici nella
variabile $break\_point\_sentence\_target$ vengono divisi per il numero di frasi per finestra in modo da identificare a grandi linee in quale finestra è presente la frase che indica il cambio del discorso.

Di conseguenza i valori di $break\_point\_sentence\_target$ possono anche essere decimali, e se nel grafico la linea gialla è molto vicina a quella rossa il programma, in questa particolare configurazione delle finestre, non poteva arrivare ad una soluzione più corretta. (In quanto la frase che genera il salto del discorso è interna ad una finestra).

Come si può vedere dai grafici il parametro ottimale è $windows\_size = 4$, con questa configurazione il programma individua esattamente 3 break point su 4.

E' anche stato realizzato un metodo che dal testo costruisce una matrice dove sulle:

- **righe** : sono indicati i termini rilevanti (non stop word e non numeri) con occorrenze totale nel testo maggiore di 3
- **colonne** : indici delle frasi all'interno del testo
- **celle(i, j)**: occorrenza parola della riga i nella frase con indice j



Come si può vedere in figura, in questo testo i cambi di discorso non sono troppo evidenti in quanto le parole usate non variano molto da una finestra ad un'altra.

NOTA: per eseguire il programma è necessario che in *utils* ci sia la versione lexical di NASARI (in questa versione sono presenti 10 feature per ogni item) disponibile al seguente url:

- https://goo.gl/85BubW

Open information Extraction

Consegna : Implementazione di un sistema di OIE

Implementazione

Le triple sono estratte da frasi che contengo almeno un soggetto, un verbo e un complemento oggetto.

Il sistema implementato è in grado di estrarre delle triplette della forma **(Soggetto, Verbo, Oggetto)** da un insieme di frasi.

Data in input una frase il sistema esegue:

1. **Pos Tagging e Dependency Parsing** con l'ausilio di **spaCy** che per ogni token allega la rispettiva dipendenza sintattica (**dep_**)
2. Vengono scansionate le dipendenze trovate e costruiti gli argomenti della tripletta

La scelta che l'algoritmo deve fare è in quale dei 3 argomenti mappare i token della frase in analisi.

Mapping

Per poter fare questo è necessario categorizzare la **dep_** in modo da restringere le associazioni possibili tra **dep_** e argomento e delle triple.

I soggetti, complementi oggetti e oggetto diretto e le relazioni vengono riconosciuti in quanto la loro dipendenza sintattica ricade:

- [nsubj, nsubjpass] per i soggetti
- [dobj, obj] per i complementi oggetti e oggetto diretti
- ["ROOT", "adj", "attr", "agent", "amod"] per le relazioni
    - adj è stato inserito perchè se presente, legato al verbo apporta maggiore significativià delle triple.

Le relazioni e le entità possono essere costruite usando più di una parola, infatti il Soggetto, Predicato e Oggetto da inserire nelle triple sono realizzati come concatenazioni di più parole. Dopo aver individuato l'elemento centrale di uno degli argomenti della tripla vengono aggiunte informazioni aggiuntive, concatenando altri token successivi se la loro dipendenza sintattica appartenenza a questo insieme:

- ["compound", "prep", "conj", "mod"]

Dopo aver valutato il token x se appartiene ad uno dei 3 insiemi indicati sopra il suo testo viene salvato in una variabile temporale concatenando i testi dei token successivi, con il criterio sopra citato, fino a quando non si trovo un token appartenente ad un insieme diverso da quello a cui appartenga il token x.

Risultati

Sono state prese circa 15 frasi da Wikipedia di diversa complessità, come si può vedere se la frase è semplice il sistema riesce a mappare il suo significato in una tripla, ma quando la complessità aumenta le triple diventano poco consistenti o non riescono a mappare il significato della frase.

Sentence : Roma is the capital of Italy.

- ***(Roma, be capital, Italy)***

Sentence : Londinium was founded by the Romans.

- ***(Londinium, found by, Romans)***

Sentence : Italy is considered to be one of the world's most culturally and economically advanced countries.

- ***(Italy, consider one culturally, countries)***

Sentence : The City of London, London's ancient core − an area of just 1.12 square miles (2.9 km2) and colloquially known as the Square Mile − retains boundaries that follow closely its medieval limits.

- ***(City core that, ancient square retain medieval, London miles − boundaries limits)***

Sentence : The City of Westminster is also an Inner London borough holding city status.

- ***(City, be hold status, Westminster)***

Sentence : Greater London is governed by the Mayor of London and the London Assembly.

- ***(London, govern by, Mayor London)***

Sentence : London is located in the southeast of England.

- ***(London, locate, southeast England)***

Sentence : Westminster is located in London.

- ***(Westminster, locate, London)***

Sentence : London is the biggest city in Britain.

- ***(London, be big city, Britain)***

Sentence : London has a population of 7,172,036.

- ***(London, have, population 7,172,036)***

Sentence : Italy is a country consisting of a peninsula.

- ***(Italy, be country, peninsula)***

Sentence : Italy is located in south-central Europe.

- ***(Italy, locate south central, Europe)***

Sentence : The Roman Empire was among the most powerful economic, cultural, political and military forces in the world of its time.

- ***(Empire, be powerful economic, forces world time)***