Babelarity - Progetto del corso di Metodologie di Programmazione A.A. 2017/2018

prof. Roberto Navigli, dott. Federico Scozzafava

bozza - aggiornata il 19/07/2018

v1.1 in arancio - modifiche necessarie per permettere la correzione automatica
Aggiornamenti nella classe Junit in verde

Il progetto finale del corso di Metodologie di Programmazione 2017/2018 consiste nella realizzazione di un framework per il calcolo della similarità semantica tra documenti di testo.

Definizioni preliminari

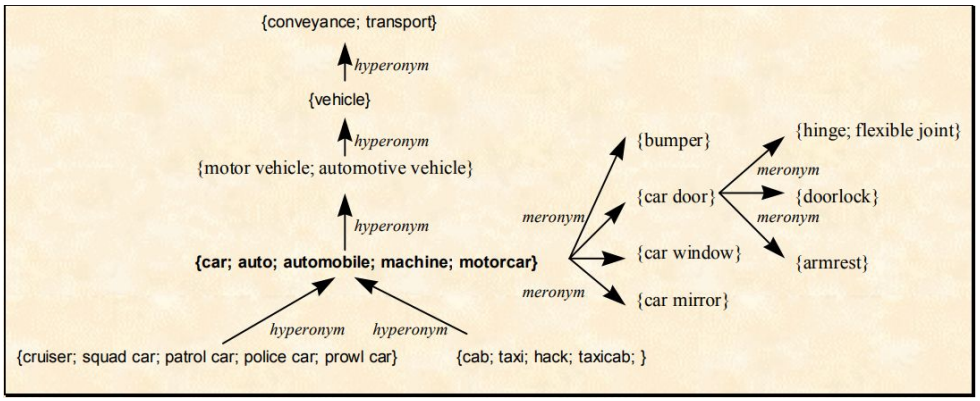
"**La similarità semantica** è una metrica definita su un insieme di documenti o termini tra i quali il concetto di distanza è basato sulla somiglianza tra i soli significati (o contenuto semantico), al contrario della similarità che può essere stimata considerando unicamente la rappresentazione sintattica (quindi non semantica) dei termini o dei documenti."
La similarità semantica può essere stimata utilizzando reti semantiche (o basi di conoscenza) che definiscono le relazioni tra concetti. Ad esempio "automobile" è simile a "bus", ma è anche correlato a "strada" e "guidare". *https://en.wikipedia.org/wiki/Semantic_similarity*

La similarità tra documenti (o distanza tra documenti) è uno dei temi centrali nel campo dell'Information Retrieval (il campo su cui si fondano i motori di ricerca). Le metriche di similarità tra documenti rivestono un ruolo chiave in applicazioni web come l'indicizzazione di pagine, clustering dei documenti ed estrazione automatica dei metadati.

"Una **rete semantica** è una forma di rappresentazione della conoscenza. È un grafo (orientato o non orientato) formato da vertici, che rappresentano concetti, e archi, che rappresentano relazioni semantiche tra i concetti. Le reti semantiche sono un tipo comune di dizionario leggibile da una macchina (dizionario elettronico)." *https://it.wikipedia.org/wiki/Rete_semantica*

BabelNet (http://babelnet.org) è una risorsa linguistica multilingua che aggrega informazioni lessicografiche ed enciclopediche provenienti da risorse eterogenee in diverse lingue in un'unica rete semantica. Le maggiori fonti di informazione della rete provengono dalla rete semantica WordNet (https://wordnet.princeton.edu/) e da Wikipedia (https://wikipedia.org).
BabelNet contiene informazioni riguardo concetti ed entità nominali per nomi, aggettivi, avverbi e verbi ed è costruita intorno al concetto di **synset (synonym set o insieme di sinonimi)**. Un synset è un insieme di parole con la stessa parte del discorso (part-of-speech o POS) che possono essere usate e sostituite in un certo contesto. Per esempio le parole { car, auto, automobile, machine, motorcar } formano un synset perché possono essere usate per esprimere lo stesso concetto. Un synset è spesso accompagnato da una descrizione (glossa): "4-wheeled; usually propelled by an internal combustion engine". Inoltre i synset possono essere legati tra

loro attraverso relazioni semantiche come iperonimia (tra concetti specifici e più generali), meronimia (tra la parte e il tutto), causa ecc. come illustrato sotto.



miniBabelNet rappresenta una versione ridotta e semplificata della rete semantica BabelNet limitata anche alla sola lingua inglese i cui nodi (synset) sono un sottoinsieme di quelli presenti in WordNet e le relazioni tra questi sono arricchite con archi provenienti da altre risorse presenti in BabelNet.

Il framework di similarità semantica è basato sulla **rete semantica miniBabelNet**. La rete semantica include informazioni e relazioni tra parole e concetti rappresentate nel seguente modo:

- Un **dizionario** che associa ad ogni parola in forma flessa il suo lemma, ovvero la sua forma base non declinata (la forma che si trova normalmente all'interno di un dizionario)
    - **lemmatization-en.txt** nel seguente formato:
      `forma flessa TAB(\t) lemma separati da TAB NEWLINE(\n).`
- Un **dizionario** che contiene tutti i synset uno per ogni riga con i sinonimi (lemmatizzazioni) che esprimono quel concetto; ogni riga del file è un nodo del grafo di BabelNet:
    - I sensi all'interno di miniBabelNet sono raggruppati in synset, ovvero insiemi di sinonimi che esprimono lo stesso concetto
    - Ogni synset è identificato attraverso un id univoco (ad es. bn:00000001n)
    - **dictionary.txt** nel seguente formato:

```
id_synset TAB(\t) lemma1 TAB(\t) lemma2 TAB(\t) ... TAB(\t) lemmaN  NEWLINE(\n)
```

- Un file contenente le **definizioni** per ogni synset
    - **glosses.txt** nel seguente formato:

```
id_synset TAB(\t) gloss1 TAB(\t) gloss2 TAB(\t) ... TAB(\t) glossN NEWLINE(\n)
```

- Un file contenente le **relazioni** (gli archi del grafo) tra i vari synset sotto forma di coppie
    - ogni coppia di synset è un arco nella rete semantica
    - **relations.txt** nel seguente formato:

```
synset_sorgente TAB(\t) synset_destinazione TAB(\t) nome_relazione_semplice
TAB(\t) nome_relazione_completo NEWLINE(\n)
```

Download miniBabelNet

Download Documents

Download Corpus

Struttura del progetto

Il progetto consiste nello sviluppo di:

1) un sistema di caricamento e memorizzazione dei documenti di testo
2) una misura di similarità lessicale tra parole nella rete miniBabelNet
3) una misura di similarità semantica tra concetti (synset) nella rete miniBabelNet
4) l'implementazione di una misura di similarità tra documenti

Sul gruppo facebook sarà rilasciato un file .zip contenente le interfacce da implementare insieme ad un piccolo test junit.

Nota bene:

Per ogni misura di similarità sono suggerite due metriche, una base ed una avanzata. È richiesto lo sviluppo di **UNA SOLA** metrica per ogni parte del progetto a totale discrezione dello studente.

Assumete che i file in input si trovino nella cartella **resources** alla radice del progetto. NON CABLATE PERCORSI ALL'INTERNO DEL CODICE (es. "c:\\casamia\\progetto_che_piace_a_me\\resources\\file.txt"), ma utilizzate solo percorsi locali impostati come variabili costanti statiche pubbliche, idealmente all'interno di interfacce (es. DICTIONARY_FILE = "resources/dictionary.txt"). Utilizzare / e non \\, per permettere l'accesso sia su sistemi linux che windows.

Modellazione del problema

Il sistema consente di calcolare la similarità tra **oggetti linguistici**, questi possono essere Parole, Synset o Documenti. La classe **MiniBabelNet** rappresenta l'implementazione della omonima rete semantica ed espone un metodo per calcolare la similarità tra oggetti linguistici ed implementa il pattern strategy per impostare le implementazioni di similarità tra i diversi tipi di oggetti linguistici.

Classi

Si implementi la classe **MiniBabelNet** che rappresenta l'omonima rete semantica. La classe deve implementare il **Singleton pattern** e deve essere **iterabile sui synset** contenuti nella rete semantica ed implementa l'interfaccia i seguenti metodi:

- **getSynsets**(String word): restituisce l'insieme di synset che contengono tra i loro sensi la parola in input
- **getSynset**(String id): restituisce il synset relativo all'id specificato
- **getLemmas**(String word): restituisce uno o più lemmi associati alla parola flessa fornita in input
- **getSynsetSummary(Synset s)**: Restituisce le informazioni inerenti al Synset fornito in input sotto forma di stringa.

    - Restituisce le informazioni inerenti al Synset fornito in input sotto forma di stringa:

  Il formato della stringa è il seguente:
  ID\tPOS\tLEMMI\tGLOSSE\tRELAZIONI
  Le componenti LEMMI, GLOSSE e RELAZIONI possono contenere più elementi, questi sono separati dal carattere ";"
  Le relazioni devono essere condificate nel seguente formato:
  TARGETSYNSET_RELATIONNAME es. bn:00081546n_has-kind

  es: bn:00047028n NOUN word;intelligence;news;tidings Information about recent and important events bn:0000001n_has-kind;bn:0000001n_is-a
- **setLexicalSimilarityStrategy()**: Imposta l'algoritmo di calcolo della similarità tra parole (di default, in fase di costruzione dell'oggetto viene impostato l'algoritmo implementato dallo studente).
- **setSemanticSimilarityStrategy()**: Imposta l'algoritmo di calcolo della similarità tra synset (di default, in fase di costruzione dell'oggetto viene impostato l'algoritmo implementato dallo studente).
- **setDocumentSimilarityStrategy()**: Imposta l'algoritmo di calcolo della similarità tra documenti (di default, in fase di costruzione dell'oggetto viene impostato l'algoritmo implementato dallo studente).
- computeSimilarity(): calcola e restituisce un double che rappresenta la similarità tra due oggetti linguistici (Synset, Documenti o parole)

Si implementi la classe **BabelSynset** che implementi l'interfaccia segnaposto **Synset** ed espone i seguenti metodi:

- **getID()**: restituisce l'id univoco del synset sotto forma di stringa. Gli identificativi dei synset seguono il formato bn:00000000n, dove l'ultimo carattere rappresenta la parte del discorso del concetto n(oun), v(erb), a(djective), (adve)r(b).
- **getPOS()**: restituisce la parte del discorso (Part-of-Speech) del synset (calcolabile a partire dall'ID del synset) scelta tra NOUN, ADV, ADJ, VERB. Utilizzare un'enumerazione.
- **getLemmas()**: restituisce l'insieme delle lessicalizzazioni di cui è composto il synset
- **getGlosses()**: restituisce le definizioni del synset

Si implementi la classe **CorpusManager**, responsabile del parsing, caricamento e salvataggio dei documenti. La classe deve essere **iterabile** sui documenti contenuti nel corpus, deve implementare il pattern singleton ed espone i seguenti metodi:

- **parseDocument**(): restituisce una nuova istanza di Document parsando un file di testo di cui è fornito il percorso in input.
    - Ogni documento fornito è così strutturato: nella prima linea è presente il titolo e l'ID del documento separati da TAB. Il resto del documento rappresenta il contenuto testuale.
- **saveDocument**(): salva su disco l'oggetto Document passato in input.
- **loadDocument**(): carica da disco l'oggetto Document identificato dal suo ID.

Si implementi la classe **Document** che implementa i seguenti metodi:

- **getTitle**(): restituisce il titolo del documento
- **getId():** restituisce l'Id del documento
- **getContent**(): restituisce il contenuto del documento sotto forma di stringa

Similarità Lessicale:

Fornire l'implementazione della classe **BabelLexicalSimilarity** che definisce il metodo per calcolare la similarità tra due parole:

- **computeSimilarity**(Word w1, Word w2): restituisce il valore di similarità sotto forma di double

Algoritmo baseline:

Il calcolo della similarità lessicale tra due parole utilizza la rete semantica **miniBabelNet** contando il numero di parole in comune tra le definizioni di tutti i synset in cui appare il lemma della prima parola e quelli della seconda. Ad esempio, per calcolare la similarità lessicale tra "word" e "sentence" si identificano le occorrenze di parole in comune (in grassetto qui sotto):

DEFINIZIONI DI "WORD"

A unit of language that native speakers can identify
In **linguistics**, a **word** is **the** smallest element that can be uttered in isolation **with** objective **or** practical meaning.
The smallest element that may be uttered in isolation **with** semantic **or** pragmatic content
Smallest **linguistic** element that may be uttered in isolation **with** semantic **or** pragmatic content
A distinct **unit of language** (sounds in speech **or** written **letters**) **with** a particular meaning, composed **of** one **or more** morphemes, **and** also **of** one **or more** phonemes that determine its sound pattern.
The smallest discrete **unit of** written **language with** a particular meaning, composed **of** one **or more letters or** symbols **and** one **or more** morphemes

DEFINIZIONI DI "SENTENCE"

A string **of words** satisfying **the** grammatical rules **of** a **language**
In non-functional **linguistics**, a sentence is a textual **unit** consisting **of** one **or more words** that are grammatically linked.
A grammatical **unit of language**.
Grammatical **unit**.
A grammatically complete series **of words** (consisting **of** a subject **and** predicate, even if one **or the** other is implied) that typically begins **with** a capital **letter and** ends **with** a full stop.
A grammatically complete series **of words** consisting **of** a subject **and** predicate, even if one **or the** other is implied, **and** typically beginning **with** a capital **letter and** ending **with** a full stop.
Grammatically complete series **of words** consisting **of** a subject **and** predicate.

e si calcola quindi il valore: numero di occorrenze in comune / (numero parole nelle definizioni 1 + numero parole nelle definizioni 2) = 104/(114+113) = 0.4581

Si consiglia di utilizzare un insieme di **parole stopword** da non considerare nel conteggio (es. articoli, preposizioni, ecc.).

Algoritmo avanzato:

Un approccio più avanzato consiste nel calcolare la similarità tra due parole sfruttando le informazioni lessicali contenute nel contesto in cui queste appaiono, utilizzando il corpus di documenti fornito.
La similarità tra due parole è definita attraverso una misura di similarità vettoriale: descriveremo prima un metodo per rappresentare una parola sotto forma di un vettore numerico, e successivamente descriveremo due metodi per il calcolo della similarità tra vettori.

Sia V il vocabolario delle parole univoche costruito a partire dal corpus di documenti, ovvero una mappa contenente tutti i lemmi unici (senza duplicati) che appaiono in ogni documento del corpus associati a un indice intero univoco. Ogni parola *w* è rappresentata attraverso un vettore di co-occorrenza, ovvero un istogramma lungo |V|, nel quale il valore della componente i-esima indica quanto la parola *w* tende ad apparire in un testo insieme alla parola *i*-esima nel vocabolario V.
Date due parole x e y, il valore di co-occorrenza è calcolato come segue:

$$pmi(x;y) \equiv log \frac{freq(x,y)}{freq(x) \cdot freq(y)}$$

dove freq(x,y) è il numero di volte che le parole x ed y appaiono insieme all'interno di un documento, mentre freq(x) e freq(y) è il numero di volte in cui le parole appaiono in assoluto all'interno del corpus.
Per calcolare il vettore di co-occorrenza di una parola w è quindi sufficiente calcolare il valore di co-occorrenza tra le coppie di parole (*w*, *w'*) per ogni *w'* nel vocabolario V.
La PMI (Pointwise Mutual Information) è una misura di associazione che quantifica la discrepanza tra la probabilità della coincidenza di due eventi (nel nostro caso la probabilità che le parole x e y co-occorrano) e la loro distribuzione individuale (la loro frequenza assoluta nel corpus).

Si consiglia di utilizzare un insieme di **parole stopword** da non considerare nel conteggio (es. articoli, preposizioni, ecc.).

La similarità tra due parole è infine calcolata misurando la distanza tra i relativi vettori contestuali utilizzando misure di similarità tra vettori come:

- Cosine similarity
- Weighted Overlap

Esempio:
Documento1 = 'A programming language is a formal computer language or constructed language designed to communicate instructions to a machine, particularly a computer.'

Documento2 = ‘C (/ˈsiː/, as in the letter c) is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations.’

Documento3 = ‘Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.’

Corpus = [Documento1, Documento2, Documento3]

V = [program, language, formal, computer, language, construct, design, communicate, instruction, machine, particular, general, purpose, imperative, support, structure, lexical, variable, scope, recursion, static, type, system, prevent, intend, operation, java, concurrent, class, object, implementation, dependency, possible]

pmi(java, language) = freq(java, language)/(freq(java)*freq(language)) = 1 / (1*3) = 1/3

vettore[java] = [language:pmi(java,language)=⅓, program:pmi(java, program)=1, object:⅓ ….]

Misura di similarità vettoriale:

Cosine similarity:

piano = (spartito=1, edificio=100, musica=10000, schermo=0, lettera=0)
musica = (spartito=1, edificio=0.5, musica=1, schermo=0, lettera=0)
cosineSimilarity(piano, musica) = (1*1 + 100*0.5 + 10000*1 + 0*0 + 0*0) / (sqrt(1^2+100^2+10000^2+0^2+0^2) * sqrt(1^2+0.5^2+1^2+0^2+0^2)) = 0.670033162496

La similarità del coseno, o cosine similarity, è una tecnica euristica per la misurazione della similitudine tra due vettori effettuata calcolando il coseno tra di loro, usata generalmente per il confronto di testi nel data mining e nell'analisi del testo (*Wikipedia*). Dati due vettori di attributi numerici, A e B, il livello di similarità tra di loro è espresso utilizzando la formula

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$$

Un altro modo di indicare la formula, del tutto equivalente, è:

$$\frac{\sum_{k=1}^{n} A(k)B(k)}{\sqrt{\sum_{k=1}^{n} A(k)^2}\sqrt{\sum_{k=1}^{n} B(k)^2}}$$

In altre parole al numeratore viene calcolata la somma del prodotto delle coppie di dimensioni A_i * B_i, per ogni dimensione i dei vettori, al denominatore si moltiplicano le radici delle somme dei quadrati A_i^2 di ogni dimensione i.

Weighted Overlap:

La misura di similarità vettoriale Weighted Overlap è così definita:

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

v1 e v2 sono due vettori, O è l'insieme di dimensioni in comune tra i due vettori, ovvero i sensi o le parole in comune nei vettori contestuali, e rank(q, v) è la posizione della dimensione q nel vettore v dove ogni componente è ordinata per valore.
In altre parole ogni vettore è inizialmente ordinato e ad ogni componente è associato un valore di rank che coincide con la sua posizione nel vettore ordinato. Successivamente si sommano i rank delle componenti in comune di entrambi i vettori e si divide per la somma di 2*i per i che va da 1 al numero di componenti in comune.

Esempio:
v1 = [casa:1, cane:3, martello:21, java:12, porta:13, garage:7]
v2 = [pollo:2, monitor:32, java:23, casa:7, cane:17]

ordiniamo i vettori per ricavare il rank di ogni componente:
v1_r = [martello:21, java:12, porta:13, garage:7, cane:3, casa:1]
v2_r = [monitor:32, java:23, cane:17, casa:7, pollo:2]

individuiamo le componenti in comune:
O = [java, casa, cane]

la Weighted Overlap tra v1 e v2 è quindi calcolata come segue:

WO(v1,v2) = (1/(2+2)+1(6+5)+1/(5+3)) / (1/2+1/4+1/6) = (¼+1/11+⅛) / 0.916 = 0.465 /0.916 = 0.501

Solitamente la Weighted Overlap funziona bene su vettori poco sparsi con un esiguo numero di componenti in comune, mentre la Cosine Similarity funziona meglio su vettori più sparsi.

Similarità Semantica

Fornire l'implementazione della classe **BabelSemanticSimilarity** che definisce il metodo per calcolare la similarità tra due synset:

- **computeSimilarity**(Synset s1, Synset s2): restituisce il valore di similarità sotto forma di double

La similarità semantica è definita tra due synset e sfrutta le informazioni della rete semantica miniBabelNet: la rete definisce le relazioni semantiche di vario tipo tra i vari concetti ed entità tra cui:

- is-a: iperonimia (‘car’ is-a ‘vehicle)
- has-kind: iponimia (‘vehicle’ has-kind ‘car’)
- part-of: meronimia (‘window’ part-of ‘building’)
- has-part: olonimia (‘building’ has-part ‘window’)

PATH:

Questa metrica di similarità consiste nel calcolo del **percorso di lunghezza minima** (se esiste) tra i due concetti all’interno della rete semantica. Un modo semplice e rapido per calcolare il cammino di lunghezza minima è attraverso una **visita in ampiezza** (BFS, https://en.wikipedia.org/wiki/Breadth-first_search) del grafo diretto a partire dal nodo sorgente che termina non appena il nodo destinazione è stato visitato.

Pseudocodice della procedura di visita BFS:

```
BFS (G, s)                    //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

      mark s as visited.
      while ( Q is not empty)
            //Removing that vertex from queue,whose neighbour will be visited now
            v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                         Q.enqueue( w )              //Stores w in Q to further visit its neighbour
                         mark w as visited.
```

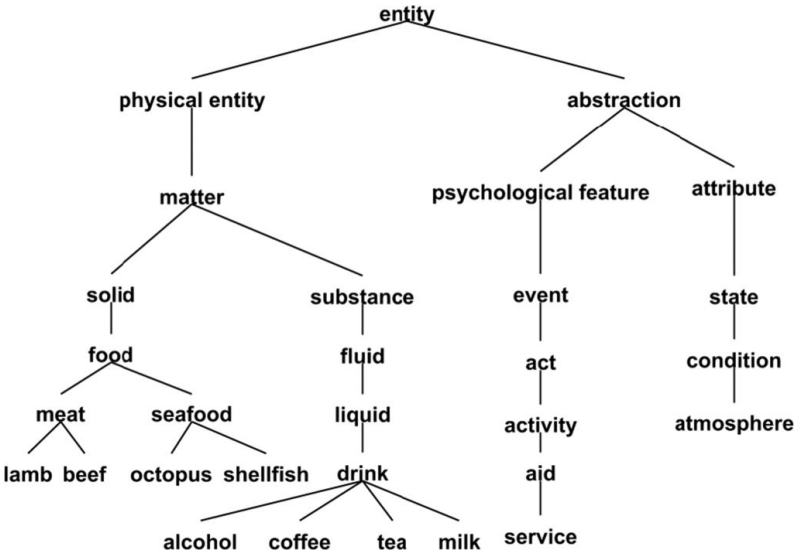
Dato il cammino tra due concetti s1 e s2, la **similarità** è calcolata con la seguente **formula**:

$$PATH(s1,\ s2)\ =\ 1\ /\ (path\_length(s1,\ s2) + 1)$$

Algoritmo avanzato:

LCH:

Questa metrica di similarità impiega la tassonomia degli iperonimi della rete semantica, ovvero l’albero delle relazioni is-a tra i concetti che ha come radice il concetto generale ‘Entity’. Un estratto dell'albero tassonomico, ogni arco rappresenta una relazione di iperonimia (is-a) tra concetti in miniBabelNet.



L'algoritmo LCH consiste nell'individuazione del Least Common Subsumer (LCS) di due concetti A e B, ovvero il concetto più specifico che sia antenato sia di A che B. Nell'esempio mostrato l'LCS tra 'seafood' e 'liquid' è rappresentato dal nodo 'matter'.
Il calcolo dell'LCS consiste in una semplice visita dell'albero 'a ritroso', partendo dal nodo sorgente attraversando solamente archi 'is-a' fino alla radice. Una volta calcolati i cammini nodo-radice di entrambi i synset si consideri (se esiste) il nodo a livello nell'intersezione dei due cammini che si colloca a livello più basso nella tassonomia.

Dati due synset s1 e s2 la similarità tra i concetti è misurata come segue:

$$LCH(s1,\ s2)\ =\ -Math.log\_e(LCS(s1,\ s2).length\ /\ (\ 2\ *\ max\_depth(pos)\ )\ )$$

dove max_depth(pos) è la profondità massima dell'albero. Il parametro pos indica che è consigliabile calcolare diversi valori di profondità massima per ogni POS dei nodi s1 e s2, definendo diversi valori di profondità massima per nomi, aggettivi avverbi e verbi.

Similarità tra documenti:

Fornire l'implementazione della classe **BabelDocumentSimilarity** che definisce il metodo per calcolare la similarità tra due documenti:

- **computeSimilarity**(Document d1, Document d2): restituisce il valore di similarità sotto forma di double

Bag of Words:

Un modo semplice e rapido per definire una misura di similarità tra due documenti consiste nel rappresentare ogni documento attraverso una Bag-of-Words (BoW), ovvero come un insieme (non ordinato) di parole.

Nella classificazione di documenti, la BoW è un vettore sparso del numero di occorrenze delle parole, che non è altro che un istogramma sparso sul vocabolario. (*Wikipedia*)
Per ogni documento è possibile quindi definire il vettore delle occorrenze delle parole normalizzato e calcolare la similarità utilizzando la Cosine Similarity o Weighted Overlap, analogamente a come descritto nell'algoritmo avanzato per la similarità tra parole.

Algoritmo avanzato:

Una rappresentazione più sofisticata di un documento è attraverso la definizione di un grafo semantico. Il grafo semantico di un documento è composto dai synset estratti dalle parole del testo connessi tra loro tramite le relazioni definite in miniBabelNet. Successivamente definiremo un algoritmo che determina la correlazione tra due grafi semantici.

Costruzione del grafo semantico:

- Per ogni parola del documento si considerino i synset associati all'interno di miniBabelNet
- Una volta identificati i nodi del grafo si procede con l'aggiunta degli archi:
    - se due nodi del grafo sono in relazione diretta si aggiunge un arco
    - per i restanti nodi non connessi effettuiamo una visita BFS a distanza massima 2 (l'esplorazione del grafo non scopre nodi distanti più di 2 archi)
    - se durante la ricerca visitiamo un nodo presente nel documento aggiungiamo un arco diretto tra quest'ultimo e il nodo sorgente.

RandomWalk:

Una volta costruito il grafo semantico per un documento andremo ad associare un punteggio di rilevanza ad ogni synset nel grafo applicando l'algoritmo RandomWalk.
In matematica Random Walk (o passeggiata aleatoria) è la formalizzazione dell'idea di prendere passi successivi in direzioni casuale. Matematicamente parlando, è il processo stocastico più semplice, il processo markoviano, la cui rappresentazione matematica più nota è costituita dal processo di Wiener. (*Wikipedia*)
L'algoritmo di Random Walk applicato su grafi è spesso utilizzato per approssimare l'algoritmo di PageRank in una rete, il famoso algoritmo brevettato ed impiegato da Google nel suo motore di ricerca.
Analogamente all'algoritmo di PageRank che calcola un valore di rilevanza per ogni documento o pagina all'interno del WorldWideWeb, il nostro algoritmo di RandomWalk eseguito sul grafo semantico del documento assegna uno score di rilevanza semantica ad ogni synset nella rete.

L'algoritmo modella una visita aleatoria nel grafo durante la quale, partendo da un nodo casuale del grafo, ad ogni step viene percorso un arco a caso tra quelli del nodo attuale. L'algoritmo comincia scegliendo un nodo casuale ed effettua un numero di passi nel grafo fissato. Ad ogni passo, inoltre, con probabilità α l'algoritmo può decidere di interrompere il cammino e ricominciare da un nodo casuale nel grafo, questa probabilità è chiamata

probabilità di restart. Una volta effettuate il numero di iterazioni impostate l'algoritmo termina e restituisce un vettore reale nella dimensione dei synset presenti in miniBabelNet ai quali assegna come valore il numero di volte che, durante la passeggiata aleatoria, un determinato synset è stato visitato.

Pseudocodice:

Input: graph G = (V,E);
starting node n;
restart probability α
maximum number of iteration k
Output: random walk vector $v \in R^n$, $n = |S|$ (S are the synset in miniBabelNet)

0. v = new int[|S|]
1. while k > 0:
2. rand = generate a random number between 0 and 1
3. if rand > α :
4. n = choose a random node in G
5. v[n]++
6. next_step = choose a random node from the n's neighborhood
7. n = next_step
8. k--
9. return v

Metrica di similarità:

Una volta estratti i vettori di RandomWalk per ogni documento (eventualmente normalizzati) è possibile misurare la similarità calcolando la Cosine Similarity o la Weighted Overlap tra i vettori come già spiegato nei paragrafi precedenti.

Interfacce e classe Junit

```
package it.uniroma1.lcl.babelarity;

import java.nio.file.Path;

public interface CorpusManager {
	static CorpusManager getInstance() {
		// Completare
		return null;
	}

	Document parseDocument(Path path);

	Document loadDocument(String id);

	void saveDocument(Document document);
}
```

```
package it.uniroma1.lcl.babelarity;

public interface Document extends LinguisticObject{
	String getId();
	String getTitle();
	String getContent();
}
```

```
package it.uniroma1.lcl.babelarity;

public interface LinguisticObject {
}
```

```
package it.uniroma1.lcl.babelarity;

import java.util.List;

public interface MiniBabelNet extends Iterable<Synset>{

	static MiniBabelNet getInstance() {
		// Completare
		return null;
	}

	List<Synset> getSynsets(String word);

	Synset getSynset(String id);

	List<String> getLemmas(String word);

	/**
	 * Restituisce le informazioni inerenti al Synset fornito in input sotto forma di stringa.
	 * Il formato della stringa è il seguente:
	 * ID\tPOS\tLEMMI\tGLOSSE\tRELAZIONI
	 * Le componenti LEMMI, GLOSSE e RELAZIONI possono contenere più elementi, questi sono separati dal carattere ";"
	 * Le relazioni devono essere condificate nel seguente formato:
	 * TARGETSYNSET_RELNAME   es. bn:00081546n_has-kind
	 *
	 * es: bn:00047028n	NOUN	word;intelligence;news;tidings	Information about recent and important events	bn:0000001n_has-kind;bn:0000001n_is-a
	 *
	 * @param s
	 * @return
	 */
	String getSynsetSummary(Synset s);

	double computeSimilarity(LinguisticObject o1, LinguisticObject o2);

}
```

```
package it.uniroma1.lcl.babelarity;

public interface Synset extends LinguisticObject{
}
```

```
---------------
```

```
package it.uniroma1.lcl.babelarity;

public interface Word extends LinguisticObject{
	static Word fromString(String s) {
		//Completare
		return null;
	}
}
```

Classe di test Junit:

```
package it.uniroma1.lcl.babelarity;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.api.TestInstance.Lifecycle;

@TestInstance(Lifecycle.PER_CLASS)
class BabelarityTest {

	private static final Path DOCUMENTS = Paths.get("resources/documents");
	private MiniBabelNet miniBabelNet;
	private CorpusManager documentManager;

	@BeforeAll
	public void setup() {
		try {
			miniBabelNet = MiniBabelNet.getInstance();
			documentManager = CorpusManager.getInstance();
		} catch (Exception e) {
			e.printStackTrace();
			System.out.println("NO");
		}
	}

	@Test
	public void testMiniBabelNet() {
		Synset synset = miniBabelNet.getSynset("bn:00081546n");
		String summary = miniBabelNet.getSynsetSummary(synset);
		String[] split = summary.split("\t");
		String id = split[0];
		String pos = split[1];
		String[] lemmi = split[2].split(";");
		String[] glosse = split[3].split(";");
		String[] relations = split[4].split(";");
		Arrays.sort(relations);
		assertTrue(lemmi[0].equals("word") && lemmi.length == 1);
		assertEquals(id, "bn:00081546n");
		assertEquals(pos, "NOUN");
		assertTrue(glosse.length == 6);
		assertTrue(relations.length == 749);
		assertEquals(relations[2], "bn:00000239n_has-kind");
	}

	@Test
	public void testDocumentManager() {
```

```
                Document document =
documentManager.parseDocument(DOCUMENTS.resolve("Cultural_tourism.txt"));
                assertEquals(document.getTitle(), "Cultural tourism");
                assertEquals(document.getContent().substring(0, 107),
                                "Cultural tourism (or culture tourism) is the subset of tourism
concerned with a country or region's culture");
                Document document1 =
documentManager.parseDocument(DOCUMENTS.resolve("programming_language.txt"));
                assertTrue(!document.getId().equals(document1.getId()));
                documentManager.saveDocument(document);
                assertEquals(document, documentManager.loadDocument(document.getId()));
        }

        @Test
        public void testLexicalSimilarityIdentity()
        {
                testSimilarityIdentity(Word.fromString("test"));
        }

        @Test
        public void testLexicalSimilarity1() {
                String word1 = "test";
                String word2 = "exam";
                String word3 = "pop";
                String word4 = "rock";
                similarityTest(Word.fromString(word1), Word.fromString(word2), Word.fromString(word3),
Word.fromString(word4));
        }

        @Test
        public void testLexicalSimilarity2() {
                String word1 = "port";
                String word2 = "ship";
                String word3 = "fear";
                String word4 = "emotion";
                similarityTest(Word.fromString(word1), Word.fromString(word2), Word.fromString(word3),
Word.fromString(word4));
        }

        @Test
        public void testSemanticSimilarityIdentity()
        {
                Synset synset1 = miniBabelNet.getSynset("bn:00036821n");
                testSimilarityIdentity(synset1);
        }

        @Test
        public void testSemanticSimilarity1() {
                Synset s1 = miniBabelNet.getSynset("bn:00034472n");
                Synset s2 = miniBabelNet.getSynset("bn:00015008n");
                Synset s3 = miniBabelNet.getSynset("bn:00081546n");
                Synset s4 = miniBabelNet.getSynset("bn:00070528n");
                similarityTest(s1, s2, s3, s4);
        }

        @Test
        public void testSemanticSimilarity2() {
                Synset s1 = miniBabelNet.getSynset("bn:00024712n");
                Synset s2 = miniBabelNet.getSynset("bn:00029345n");
                Synset s3 = miniBabelNet.getSynset("bn:00035023n");
                Synset s4 = miniBabelNet.getSynset("bn:00010605n");
                similarityTest(s1, s2, s3, s4);
        }

        @Test
        public void testDocumentSimilarity1() {
                Document d1 =
documentManager.parseDocument(DOCUMENTS.resolve("C_programming_language.txt"));
                Document d2 =
documentManager.parseDocument(DOCUMENTS.resolve("Java_programming_language.txt"));
```

```
                Document d3 =
documentManager.parseDocument(DOCUMENTS.resolve("Cristiano_Ronaldo.txt"));
                Document d4 = documentManager.parseDocument(DOCUMENTS.resolve("Thomas_Muller.txt"));
                similarityTest(d1, d2, d3, d4);
        }

        @Test
        public void testDocumentSimilarity2() {
                Document d1 = documentManager.parseDocument(DOCUMENTS.resolve("Eugenio_Montale.txt"));
                Document d2 = documentManager.parseDocument(DOCUMENTS.resolve("Umberto_Eco.txt"));
                Document d3 =
documentManager.parseDocument(DOCUMENTS.resolve("Tourism_in_the_Netherlands.txt"));
                Document d4 =
documentManager.parseDocument(DOCUMENTS.resolve("Cultural_tourism.txt"));
                similarityTest(d1, d2, d3, d4);
        }

        @Test
        public void testDocumentSimilarity3() {
                Document d1 =
documentManager.parseDocument(DOCUMENTS.resolve("Council_of_the_European_Union.txt"));
                Document d2 =
documentManager.parseDocument(DOCUMENTS.resolve("European_Union_law.txt"));
                Document d3 =
documentManager.parseDocument(DOCUMENTS.resolve("Java_virtual_machine.txt"));
                Document d4 =
documentManager.parseDocument(DOCUMENTS.resolve("Java_programming_language.txt"));
                similarityTest(d1, d2, d3, d4);
        }

        public void testSimilarityIdentity(LinguisticObject o1) {
                double sim0 = miniBabelNet.computeSimilarity(o1, o1);
                assertTrue(Double.compare(sim0, 1.0) == 0);
        }

        public void similarityTest(LinguisticObject o1, LinguisticObject o2, LinguisticObject o3,
LinguisticObject o4) {
                double sim1 = miniBabelNet.computeSimilarity(o1, o2);
                double sim2 = miniBabelNet.computeSimilarity(o3, o4);
                double sim3 = miniBabelNet.computeSimilarity(o1, o3);
                double sim4 = miniBabelNet.computeSimilarity(o2, o4);
                assertTrue(sim1 > sim3 && sim2 > sim4);
        }

}
```

NOTA BENE: la classe di test è sufficiente a ottenere 18 se l'output sarà ragionevole rispetto ai dati presenti nel file fornito in input. **Non è possibile modificare questa classe in alcun modo.**

Valutazione

Per ottenere la sufficienza il progetto deve eseguire correttamente la classe di test Junit fornita. Verranno resi disponibili nei prossimi giorni dei file di input con cui testare il progetto. Qualora l'output non fosse quello atteso, il progetto sarà valutato come

insufficiente. Nel caso in cui il test di correttezza venga superato, verranno valutati i seguenti elementi:

1. l'architettura, la modellazione a oggetti e l'organizzazione del codice in termini di correttezza ed estensibilità, soprattutto in relazione all'uso appropriato di naming (es. camelCase), incapsulamento e visibilità di campi e metodi, ereditarietà e polimorfismo.
2. documentazione del codice mediante javadoc (su tutte le intestazioni di classe e sui metodi richiesti da queste specifiche, inclusa la documentazione dei parametri), esportata da Eclipse.
3. l'implementazione corretta dei design pattern appropriati.
4. **l'uso di Java 8 (espressioni lambda e stream)**
5. **l'uso di Java 9 per la creazione di moduli appropriati (bonus +1 punto on top)**

Sono previsti quattro scaglioni di punteggio:

- **Fino a 20 punti:** implementazione di tutte le metriche base.
- **Fino a 25 punti:** implementazione di una metrica avanzata.
- **Fino a 30 punti:** implementazione di due metriche avanzate.
- **Fino a 35 punti:** implementazione di tre metriche avanzate.

Non è possibile utilizzare alcuna **libreria esterna** di supporto. Tutti i progetti con voto >= 30 saranno premiati con una maglietta di BabelNet.

Consegna

La consegna del progetto è totalmente svincolata dal sostenimento dello scritto, ovvero può essere effettuata in qualsiasi appello e non necessariamente lo stesso in cui si sostiene lo scritto. La consegna deve essere effettuata entro la mezzanotte di 5

giorni prima della data d'inizio della discussione per l'appello in questione compilando il seguente Google Form **FORM DI SOTTOMISSIONE** (ad esempio, se la discussione inizia il 25 dicembre <:o), la consegna deve essere effettuata entro le ore 24 del 20 dicembre).

Per la consegna del codice sorgente è richiesta la creazione e caricamento di sorgenti in un repository GitLab (https://gitlab.com/), indicando come nome del progetto `Babelarity_<cognome>.<matricola>` con **cognome minuscolo**.

Guida per la creazione e caricamento dei file su un progetto GitLab:

- Creare un nuovo repository attenendosi alla guida: https://docs.gitlab.com/ee/gitlab-basics/create-project.html
- Condividere il progetto con:
  - federico.scozzafava@gmail.com
  - navigli@di.uniroma1.it
  - **impostare i permessi di condivisione su 'developer'**
- Caricare il codice sul repository attenendosi alla guida:https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html

Tutte le classi e le interfacce devono trovarsi all'interno package **`it.uniroma1.lcl.babelarity`** (è possibile utilizzare sottopackage, ma non per le classi specificate sopra).

Non dovete consegnare i documenti forniti o i file relativi a miniBabelNet. Non sono ammessi percorsi assoluti cablati nel codice.

Plagio

Nel caso di plagio accertato, dal Web o da colleghi di corso, non sarà consentito consegnare nuovamente il progetto in questo A.A., il che implicherà dover sostenere nuovamente anche la prova scritta nell'anno seguente. Anche chi ha permesso il plagio (per esempio fornendo il proprio codice) e avesse già sostenuto l'esame, subirà conseguenze importanti.