

Introduzione al Natural Language Processing

Programmazione di Applicazioni Data Intensive

Laurea in Ingegneria e Scienze Informatiche
DISI – Università di Bologna, Cesena

Proff. Gianluca Moro, Roberto Pasolini
nome.cognome@unibo.it



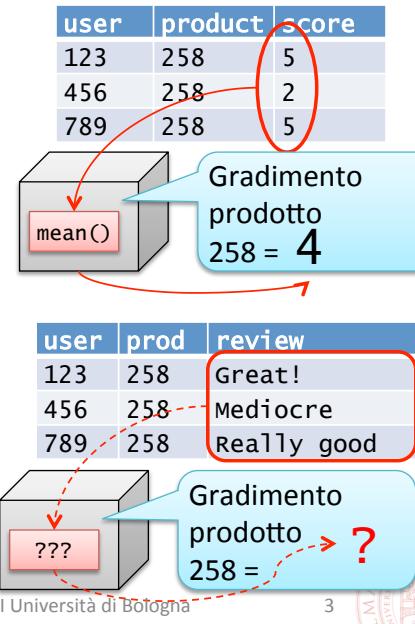
Outline

- Introduzione al Natural Language Processing
- Pre-processamento del testo
- Modello Bag of Words
- NLTK



Dati Strutturati e Destruzzurati

- **Dati strutturati:** sono provvisti di un modello (o schema) capace di descriverli e attribuirgli semantica
 - e.g. tabelle di tuple e modello relazionale
- I rating di user ai prodotti per effettuare recommendation (vedi lezioni preced.) sono un esempio di dati strutturati
 - e.g. utenti, prodotti e score, sono descritti da attributi di domini numerici secondo il modello relazionale
- **Dati destrutturati:** sono sprovvisti di un modello in grado di spiegarne formalmente la semantica
 - recensioni di prodotti in linguaggio naturale



Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

3



Dati non Strutturati: Testo

- Il tipo di dato non strutturato di maggior interesse per questo corso è il **testo in linguaggio naturale** (es. inglese, italiano ...)
 - Il testo segue **regole linguistiche** (lessico, sintassi, grammatica ...)
 - ma in generale è **ambiguo** e dipendente dal contesto; si presta a più interpretazioni (parole con più significati)
- I dati testuali destrutturati sono **ampiamente diffusi**
 - email, pagine web, post in social network, forum, blog, etc.
 - nel nostro caso di studio: nomi/descrizioni di prodotti e recensioni
- La conoscenza ricavabile da essi può avere **grande valore**
 - es., ricavare **l'opinione generale** da recensioni su prodotti, come il grado di soddisfazione verso un prodotto, servizio, un marchio ...
- Tecniche per strutturare il testo e impiegare algoritmi di analisi come per i dati strutturati
 - rappresentazioni del testo che riducano la perdita di informazione



Problematiche Tipiche nel NLP

- Nonostante un linguaggio naturale abbia sintassi e semantica definite, esistono molte difficoltà pratiche nell'interpretazione
- Nel linguaggio naturale esistono molte **ambiguità**
 - una parola può avere diversi significati e perfino svolgere diverse funzioni nella frase (nome, verbo, ...)
- Il testo da interpretare può contenere **errori**
 - refusi, tempi verbali errati, mancanza di punteggiatura, ...
- Esistono elementi di uso comune non previsti dalla lingua
 - uso intenso di abbreviazioni, emoticon :-), #hashtag, ...
- Ci sono aspetti particolarmente difficili da interpretare per un calcolatore
 - come si riconosce ironia e sarcasmo da un testo scritto?



Pre-processamento di un Testo

- Del testo è inizialmente disponibile come una **sequenza di caratteri** in forma **non strutturata**
- Un primo passaggio per l'analisi consiste nel **convertirlo** in una forma più facilmente **interpretabile dal calcolatore**
- Le tecniche di *pre-processing* estraggono dal testo una sequenza di “elementi di base” strutturati e significativi
 - questi possono includere parole, segni di punteggiatura, ...
 - un elemento può essere annotato (ad es. come nome, verbo, ...)
- L'insieme di tecniche “ideale” da usare dipende del contesto
 - ad es. le tecniche usate per il testo di un libro (frasi lunghe e ben scritte) sono in genere diverse da quelle usate per i tweet (frasi brevi e scritte di fretta con errori, uso di emoticon, ...)



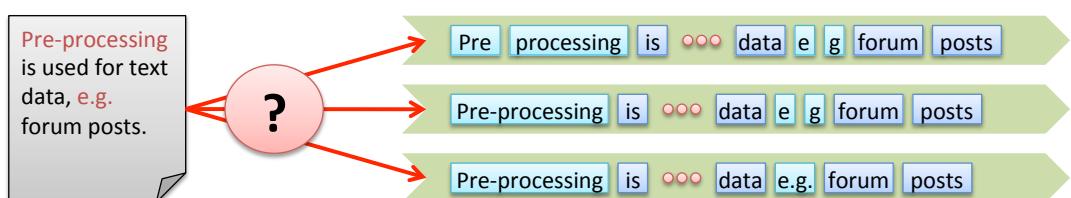
Segmentazione (*tokenization*)

- La prima operazione compiuta in genere sul testo è la sua **segmentazione in elementi sintattici di base (token)**
- L'operazione più comune consiste nello scomporre il testo nelle sue **singole parole (word tokenization)**
 - insieme alle parole possono essere previsti altri elementi, quali segni di punteggiatura, simboli, numeri, ecc.
- Testi non brevi possono anche essere dapprima **divisi in frasi (sentence tokenization)**, poi segmentate a loro volta in parole
- Anche se apparentemente semplice, la segmentazione in parole o frasi presenta delle complicazioni.
 - nella segmentazione di frasi bisogna distinguere ad es. se un punto è usato per **terminare una frase** oppure per **un'abbreviazione**



Segmentazione in Parole

- Le parole in una frase si possono separare semplicemente considerando spazi e segni di punteggiatura
- Esistono però casi particolari e ambiguità
 - due parole unite da un trattino potrebbero essere separate o unite
 - può essere desiderabile estrarre le singole parole da fusioni e contrazioni, ad esempio separare “isn’t” in “is” e “n’t”
- Per una segmentazione accurata è quindi utile integrare regole e modelli specifici della lingua analizzata



Part of Speech (POS)

- Le **part of speech (POS)** sono le **classi grammaticali** a cui ciascun elemento di una frase può appartenere
- Ad alto livello le POS sono **“nome”, “verbo”, “aggettivo”, ecc.**
 - si possono etichettare anche elementi diversi dalle parole, ad es. segni di punteggiatura
- A seconda delle esigenze, si possono fare ulteriori distinzioni
 - ad es. un nome può essere comune o proprio, singolare o plurale, ...
 - le distinzioni possono dipendere dalla lingua che si sta considerando
- Esistono diverse tassonomie di POS, da utilizzare a seconda delle esigenze
 - la più nota è quella usata dal *Penn Treebank*, il primo dataset di testo etichettato con POS di ampie dimensioni



Tipi di POS in Penn Treebank

(da <http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/Penn-Treebank-Tagset.pdf>)

1. CC	Coordinating conjunction	Si vedano ad es. le diverse classi utilizzate per nomi (N*) , verbi (V*) e aggettivi (J*)
2. CD	Cardinal number	
3. DT	Determiner	
4. EX	Existential <i>there</i>	
5. FW	Foreign word	
6. IN	Preposition or subordinating conjunction	
7. JJ	Adjective	
8. JJR	Adjective, comparative	
9. JJS	Adjective, superlative	
10. LS	List item marker	
11. MD	Modal	
12. NN	Noun, singular or mass	
13. NNS	Noun, plural	
14. NP	Proper noun, singular	
15. NPS	Proper noun, plural	
16. PDT	Predeterminer	24. SYM Symbol 25. TO <i>to</i> 26. UH Interjection 27. VB Verb, base form 28. VBD Verb, past tense 29. VBG Verb, gerund or present participle 30. VBN Verb, past participle 31. VBP Verb, non-3rd person singular present 32. VBZ Verb, 3rd person singular present 33. WDT Wh-determiner 34. WP Wh-pronoun 35. WP\$ Possessive wh-pronoun 36. WRB Wh-adverb
17. POS	Possessive ending	
18. PP	Personal pronoun	
19. PP\$	Possessive pronoun	
20. RB	Adverb	
21. RBR	Adverb, comparative	
22. RBS	Adverb, superlative	
23. RP	Particle	



Part-of-Speech Tagging

- Il **POS tagging** consiste nell'**etichettare ciascuna parola** estratta dalla segmentazione del testo col suo POS nella frase
- Questa operazione deve **tenere conto del contesto** di ogni parola, in quanto una stessa parola può avere diversi POS
 - ad es. “set” in inglese può essere un nome, un verbo o un aggettivo
- Il POS tagging può essere utile o meno a seconda dei passaggi successivi
 - utile ad esempio se si vogliono filtrare parole di tipo specifico, oppure se elaborazioni successive dipendono dai POS
 - non necessario ad es. se si vogliono estrarre le parole chiave del documento indipendentemente dalla loro posizione nella frase



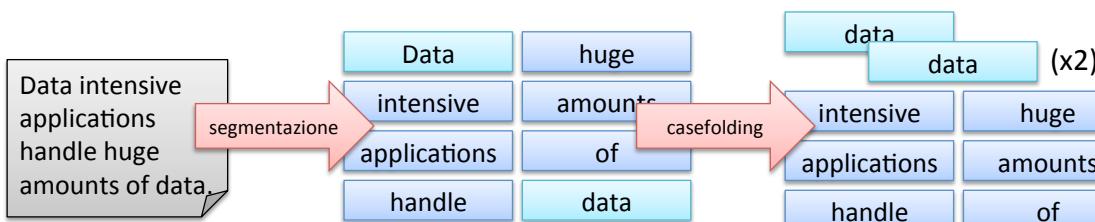
Filtrare la Sequenza di Parole

- La sequenza delle parole estratte va spesso “ripulita” prima di essere utilizzata nelle analisi successive
- Alcuni elementi estratti possono non essere necessari
 - ad es. punteggiatura ed eventuali tag HTML o simili
- Altri elementi possono essere “normalizzati” per eliminare differenze di forma ma non di sostanza
 - verbi coniugati in tempi diversi, parole con e senza maiuscola, refusi,
 - ...
- Per questo, dopo l'estrazione dei *token*, è spesso prevista una serie di passaggi per modificare la sequenza
 - spesso sostituendo o eliminando singoli elementi



CASEFOLDING

- Il **casefolding** consiste nel convertire tutte le parole **completamente in minuscolo**
 - oppure, equivalentemente, tutte in maiuscolo
- È un metodo basilare per **uniformare token** che, nonostante siano stringhe diverse, **rappresentano la stessa parola**
 - ad es., se in un testo cerco la stringa "data", in genere voglio trovare anche "Data" o "DATA"



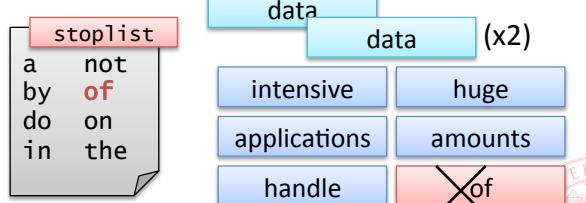
Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

15



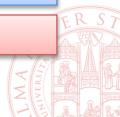
RIMOZIONE STOPWORD

- Alcune **parole** di un **documento**, indipendentemente dal loro ordine, sono **indicative dell'argomento** trattato
 - es.: "parole", "documento", "argomento"
- Altre parole, **ad esempio** articoli, preposizioni **e** congiunzioni, **non danno indicazioni sull'argomento**
 - es.: "ad", "e", "non"
- Queste parole, dette **stopword**, sono spesso **rimosse** dalla rappresentazione di ciascun documento
- Ciò si basa su una **lista predefinita** di **stopword** (*stoplist*)
 - la lista ovviamente dipende dalla lingua dei documenti
 - non esiste un'unica lista standard per ciascuna lingua



Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

16



Lemmatizzazione

- In generale le parole si trovano nei testi in **forma flessa** in base alla lingua (coniugata, declinata, ...)
 - verbo “trovare”: “trovano”, “trovasti”, “trovasse”, ...
 - aggettivo “flesso”: “flessi”, “flessa”, “flesse”
- Può essere utile **normalizzare** tutte le forme flesse di uno stesso nome, verbo, ecc. alla sua **forma base**
 - motivi simili al casefolding: dati più compatti, unione di termini simili
- La **lemmatizzazione** consiste nel sostituire ciascuna parola col suo **lemma**, ovvero la sua forma base presente nel dizionario
 - nomi al maschile singolare, verbi all’infinito, ...
- Un algoritmo esatto di lemmatizzazione è spesso complesso
 - è necessaria della conoscenza approfondita della lingua
 - è opportuno che le parole siano etichettate con la loro POS



Stemming

- Diversamente dalla lemmatizzazione, lo **stemming** ricava da ogni parola la sua **radice morfologica**
 - la radice può **non** essere una parola di senso compiuto (il lemma lo è)
- Lo stemming richiede generalmente **algoritmi più semplici** rispetto alla lemmatizzazione, risultando più efficiente
 - in genere rimuove suffissi (es. “-s”) o li sostituisce (es. “-ies”→“-i”)
- Lo stemming si usa come alternativa più efficiente alla lemmatizzazione per **unificare termini simili**
 - al contrario della lemmatizzazione, lo stemming unisce anche lemmi simili e potenzialmente correlati (es. nome e verbo con stessa radice)
 - lo stemming è però più approssimativo (può unire termini non correlati) ed il suo utilizzo può comportare una perdita di precisione
 - lo stemming e la sua efficacia dipendono dalla lingua



Full Text Search

- Le interrogazioni viste finora su database sono eseguite su dati strutturati con ottimizzazione dell'efficienza
- L'Information Retrieval si occupa della ricerca di testo in documenti, ossia in testo non strutturato
 - ciò richiede l'**estrazione e indicizzazione dei termini** in ogni documento
 - l'indicizzazione è preceduta da tecniche di pre-processing appena viste
- Se la query contiene più termini, occorre specificare **come si combinano ... devono comparire tutti ? devono essere vicini ?**
- Si possono voler individuare anche **termini simili** a quelli scritti esplicitamente (ad es. sinonimi)
- Per soddisfare al meglio la query dell'utente i documenti individuati devono essere elencati **a partire dal più rilevante**
- La ricerca deve essere più **efficiente** possibile



Full Text Search (FTS): Compromesso tra Esattezza e Completezza

- Da una parte, i documenti restituiti da una ricerca devono soddisfare il più possibile **il bisogno informativo** dell'utente
- Dall'altra, la ricerca deve **fornire quanti più documenti possibile** tra quelli disponibili che sono **rilevanti** per la query
- Date una o più query di test (con documenti rilevanti noti), è possibile misurare la bontà del sistema in termini di:
 - ***precision*** = percentuale di documenti rilevanti tra quelli restituiti
 - ***recall*** = percentuale di documenti rilevanti restituiti tra tutti quelli rilevanti nella collezione
- Tendenzialmente, cambiando i parametri di un sistema FTS per migliorare uno dei due aspetti, l'altro peggiora
 - es. se restituisco più documenti, aumentano sia quelli inerenti che non, ossia aumenta la recall e diminuisce la precision del risultato



Full Text Search: Modelli

- La disciplina dell'Information Retrieval ha elaborato diversi **modelli** per rappresentare e recuperare documenti
- Data un'interrogazione, il modello determina **quali** sono i documenti rilevanti e/o **quanto** ciascuno di essi lo è
 - Questo determina l'insieme di documenti presentati in risposta all'utente e il loro ordine, dal più rilevante a scendere
- I modelli proposti possono essere classificati in base
 - alle **teorie matematiche** su cui sono definiti: possono essere basati sulla teoria degli insiemi, sull'algebra (spazi vettoriali) o probabilistici
 - alla considerazione dell'**interdipendenza dei termini**: questa può essere ignorata, definita dal modello stesso ("intrinseca") sull'insieme di documenti o definita da una fonte esterna ("trascendente")



Modello Booleano Standard

- Nel **modello booleano**, i documenti sono semplicemente **divisi tra rilevanti e non** ad un'interrogazione
 - I documenti rilevanti sono restituiti **senza un ordine specifico**
- Ad un generico termine di ricerca viene associato un **insieme di documenti rilevanti**, quelli che contengono il termine
- I termini possono essere combinati con **operatori logici**, a cui corrispondono diverse **operazioni sugli insiemi**
 - A **AND** B (congiunzione logica): **intersezione** tra l'insieme di documenti rilevanti per A e quelli rilevanti per B
 - A **OR** B (disgiunzione logica): **unione** tra gli insiemi di documenti

Pro: concettualmente semplice, facile da implementare

Contro: restituisce i documenti **senza un ordine significativo** e spesso in quantità troppo alta o troppo bassa



Estrarre la Semantica di un Testo

- Le tecniche viste finora sono mirate ad interpretare la sintassi e la struttura del testo per ricavarne gli elementi di base
 - si ottiene una rappresentazione strutturata, ma di basso livello
- Dopo di queste, si possono applicare tecniche per interpretare il significato che il testo vuole comunicare
- Una comprensione completa ed esatta del significato di un testo è molto complessa per un calcolatore
 - è opportuno l'uso di tecniche che simulino il ragionamento umano, quali ad es. il *deep learning*
- D'altra parte la comprensione esatta è spesso non necessaria
 - ad es. per capire che due testi trattano lo stesso argomento è spesso sufficiente osservare che le parole più frequenti sono le stesse



Basi di Conoscenza Semantiche

- Per capire il significato di un testo, è necessario **conoscere le parole esistenti** e come queste siano in **relazione tra loro**
 - quali sono i significati possibili della parola X?
 - quali altre parole hanno un significato uguale o simile?
- Gli algoritmi per l'analisi testuale (soprattutto semantica) usano spesso delle **basi di conoscenza esterne** del linguaggio
- Tali basi di conoscenza sono costituite in genere da insiemi di parole con diverse informazioni associate



WordNet

- **WordNet** è tra i più noti database lessicali della lingua inglese
- Include più di 150.000 termini, divisi in 4 POS di base
 - nomi, verbi, aggettivi, avverbi
- I termini sono organizzati in oltre 100.000 *synset (synonyms set)*, insiemi di parole con un significato unico
 - ogni synset ha una descrizione testuale breve del significato (*gloss*)
 - uno stesso termine può essere presente in molteplici synset
- Tra i synset e tra i singoli termini in essi sono definite relazioni rispettivamente *semantiche* e *lessicali*

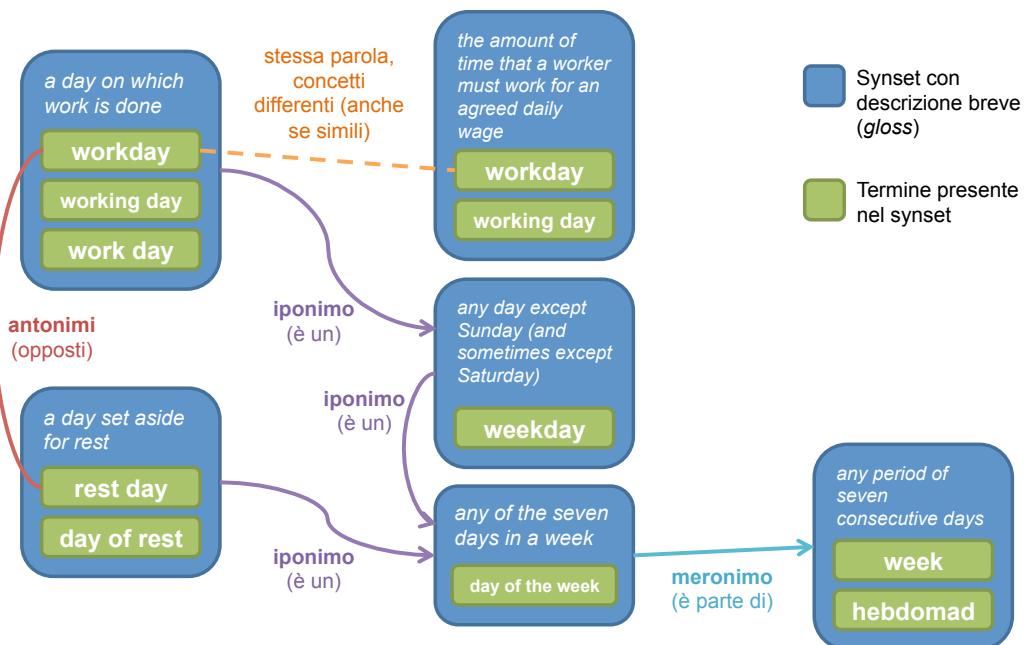


Principali Relazioni Semantiche in WordNet

- **Iponimia:** “essere un (tipo specifico di)” (detta anche *is-a*)
 - es.: “cane”, “gatto”, ... sono iponimi di “animale”
 - relazione opposta: **iperonimia** (“animale” è iperonimo di “cane”, ...)
 - (quasi) ogni synset ha un solo iperonimo, in modo da avere una tassonomia ad albero di nomi
- **Meronimia:** “essere parte/membro/sostanza di”
 - es.: “motore”, “ruota”, ... sono meronimi (parti) di “automobile”
 - relazione opposta: **olonomia** (*has-a*)
- **Implicazione (entailment):** un’azione ne comporta un’altra
 - es.: “mangiare” implica “masticare” e “ingoiare”
- **Antonimia:** “essere opposto di” (relazione lessicale)
 - es.: “uomo” e “donna” sono antonimi



Esempio: una Porzione di Grafo WordNet



Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

27



Word Sense Disambiguation

- La **Word Sense Disambiguation (WSD)** è la procedura che associa **ad ogni parola** in un testo **il suo significato** esatto
 - distinguere ad es. se “spina” in un testo si riferisce all’aculeo di un animale, ad un connettore o altro
 - La WSD richiede una base di conoscenza che associa ad ogni parola i significati possibili (ad es. WordNet)
- La disambiguazione avviene in genere in base al contesto di ciascuna parola, ovvero le parole vicine ad essa
 - ad esempio l’*algoritmo di Lesk* considera il significato la cui definizione ha il maggior numero di parole presenti vicino al termine nel testo
 - Def. Una **spina elettrica** è un connettore che può essere inserito in una **presa di corrente** complementare
 - e.g. la frase “la **spina** elettrica è inserita nella presa di corrente” ha più termini in comune con la def. precedente che con la def. successiva
 - Def. In botanica, elemento indurito, acuminato e pungente

Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

28



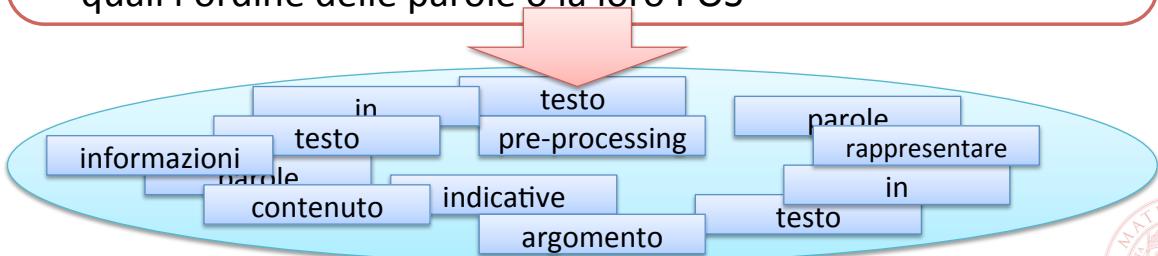
Named Entity Recognition

- Una **named entity** è una **specifica entità** a cui ci si può riferire per nome in un testo
 - può essere una persona (es. “il Presidente Obama”), un luogo (es. “Germania”), un’organizzazione (es. “ONU”), ...
 - sono a volte considerate named entity anche elementi come numeri e date (es. “25 dicembre”, “20€”, “quaranta per cento”, ...)
 - sono in genere tra gli **elementi maggiormente informativi** in un testo
- La **named entity recognition (NER)** consiste nell’**individuare** le named entity in un testo e **classificarle per tipo**
 - vanno definiti a priori i tipi di named entity da distinguere, quali “persona”, “luogo” e “organizzazione”
- Disponibili librerie anche in Python e.g. NLTK, SpaCy



Rappresentare il Contenuto di un Testo

- Nell’analisi di testi serve spesso **rappresentare il contenuto generale** di un documento in modo strutturato e compatto
 - ad es. per la classificazione automatica per categorie o per la ricerca di documenti simili tra loro o pertinenti ad un argomento dato
- Le **parole** contenute in un documento sono in genere **indicative del suo contenuto**
 - le parole sono estratte tramite le tecniche di pre-processing viste
- D’altra parte, non sono strettamente necessarie informazioni quali l’ordine delle parole o la loro POS



Rappresentazione Bag of Words

- Un ***Bag of Words*** (BOW) è una rappresentazione in forma di *multiset* delle parole contenute in un'unità di testo (un *documento*)
- Si tratta di un **elenco delle parole distinte** presenti nel testo, con associata a ciascuna il **numero di occorrenze** in esso
- Normalizzando le parole in fase di pre-processing (es. stemming), si riduce il numero di parole distinte, semplificando i passi successivi



parola	occorrenze
distinto	2
documento	1
elenco	1
forma	1
multiset	1
normalizzare	1
numero	2
ridurre	1
parola	3
testo	2



n-gram

- Gli *n-gram* sono sequenze di *n* parole presenti in un testo
 - *n*=2: *bigram*, *n*=3: *trigram*
- Gli *n-gram* si possono estrarre da un testo in aggiunta o sostituzione alle parole singole
- Possono rendere più completo il BOW di un testo, includendo **termini composti** di più parole
 - “Bag of Words”, “New York”, ...
- Sono però inclusi anche molti ***n-gram non significativi*** e il numero totale è molto alto

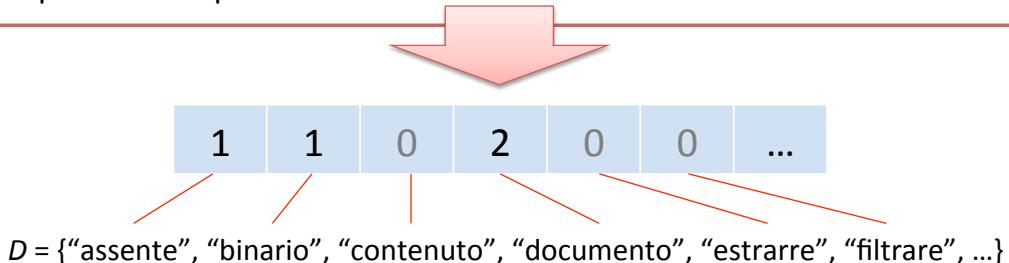


parola	occorrenze
n-gram	3
testo	3
termini	1
New	1
...	...
New York	1
parole presenti	1
termini composti	1
includendo termini	1
possono estrarre	1
...	...
...	...



Rappresentazione in Forma Vettoriale

- Si consideri di definire un insieme D (*dizionario*) di termini distinti che si potrebbero trovare in un documento di testo
- Il BOW di ciascun documento può essere rappresentato come un vettore di $|D|$ elementi
 - l' i -esimo valore è il numero di occorrenze dell' i -esimo termine
 - in alternativa si può considerare un vettore binario con 1 per le parole presenti e 0 per le assenti



Vector Space Model

- Il *Vector Space Model* prevede di rappresentare un insieme di N documenti come vettori in un medesimo spazio
- Lo spazio ha un numero di dimensioni D pari al numero di parole considerate nel dizionario condiviso
 - in genere tutte le parole distinte trovate nell'insieme di documenti
- L'insieme di documenti si può rappresentare in una *matrice termini-documenti* di dimensioni $D \times N$
 - l'elemento i,j indica quante volte la parola i appare nel documento j
 - in alcuni casi si usa invece una matrice *documenti-termini* $N \times D$, che è semplicemente la trasposta della matrice termini-documenti

	Antonio and Cleopatra	Giulio Cesare	Amleto	Otello
Antonio	157	73	0	0
Bruto	4	157	1	0
Cesare	232	227	2	1
Calpurnia	0	10	0	0
Cleopatra	57	0	0	0
mercy	2	0	5	5
worser	2	0	1	1



Term Weighting

- Ogni documento contiene termini **più o meno significativi** per caratterizzare il suo contenuto
 - le stopword e la punteggiatura sono esempi di elementi poco significativi e per questo spesso rimossi in pre-processing
- Il **numero di occorrenze** costituisce un'indicazione immediata del “peso” di un termine all'interno di un documento
- Esistono però diversi **schemi di term weighting** per ottenere **pesi più accurati** sulla base di altre informazioni
 - ad es. la presenza del termine su tutta la collezione di documenti



TF-IDF e Vector Space Model

- Nel VSM occorre determinare **come assegnare i valori (pesi)** a ciascun termine in ciascun vettore (documento o query)
- **tf** (*term frequency*) è il fattore **locale**, che pesa la rilevanza di ciascun termine in ciascuno dei **singoli documenti**
 - è il **numero di apparizioni** del termine nel documento
- **idf** (*inverse document frequency*) è il fattore **globale**, che pesa l'importanza di ciascun termine **nell'intera collezione**
 - più alto per termini che **compaiono in meno documenti**, in quanto più utili a **distinguere questi documenti** dagli altri
- il **tf-idf** per un termine t e un documento d è il prodotto dei logaritmi di **tf** e **idf**:
$$\text{tf.idf}(t, d) = \underbrace{\log(f(t, d))}_{\text{tf}} \cdot \underbrace{\log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)}_{\text{idf}}$$
 - log perché l'importanza dei termini non cresce linearmente con la loro frequenza



Normalizzazione dei Vettori

- I documenti di un insieme possono avere lunghezze differenti
- A documenti più lunghi corrispondono in genere pesi dei termini maggiori e di conseguenza vettori di norma maggiore
- Per garantire **pari peso a documenti di lunghezza diversa**, si usa **normalizzare i vettori** ottenuti in seguito al term weighting

$$w(t, d)_{\text{norm}} = \frac{w(t, d)}{\sqrt{\sum_{\tau \in D} w(\tau, d)^2}}$$

- La normalizzazione mantiene la **direzione** del vettore, che indica la **frequenza relativa** tra le parole e quindi **l'argomento**
 - ad es. se le parole *A* e *B* hanno peso 20 e 10 rispettivamente, nel vettore normalizzato *A* avrà comunque peso doppio rispetto a *B*



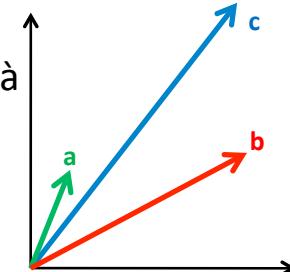
Similarità Coseno

- Rappresentando i documenti come vettori nel medesimo spazio, possiamo misurarne la similarità con diverse metriche
- Per confrontare coppie di documenti si usa spesso la **similarità coseno**, pari al **coseno dell'angolo tra i vettori**

$$\cos(a, b) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} = \frac{\sum_{i=1}^n a_i \cdot b_i}{\sqrt{\sum_{i=1}^n a_i^2} \cdot \sqrt{\sum_{i=1}^n b_i^2}}$$

Il vettore **a** è più simile a **c** di **b**, in quanto l'angolo formato è più piccolo

- è pari al **prodotto scalare dei vettori normalizzati**
- Per vettori con tutti valori positivi, la similarità coseno è sempre inclusa tra 0 e 1
 - 1 (angolo 0°): i vettori hanno la stessa direzione (se normalizzati sono uguali)
 - 0 (angolo 90°): i doc. non hanno parole in comune



Vector Space Model vs Modello Booleano

- In risposta ad una query, col VSM **si ottiene un ranking effettivo** dei documenti, **dal più rilevante a scendere**
 - La misura dell'output **non è limitata** come nel modello booleano, ma si può porre una **soglia sulla similarità** o fissare un **numero di risultati**
- Una query è rappresentata come un vettore con valori positivi in corrispondenza dei termini che appaiono in essa
- **Non è però possibile usare espressioni booleane** (AND e OR)
 - Ad es., dati due termini, non è in generale possibile dichiarare se si richiede che appaiano *entrambi* o *almeno uno* nel risultato
- Esistono modelli estesi che **combinano VSM e logica booleana**
 - Il *modello booleano esteso* prima recupera i documenti che soddisfano la proposizione booleana, poi il risultato è ordinato calcolando i rank dei doc
 - aumenta il potere espressivo del VSM con proposizioni booleane e.g. (**Harry AND Potter**) OR “**Philosopher's Stone**”



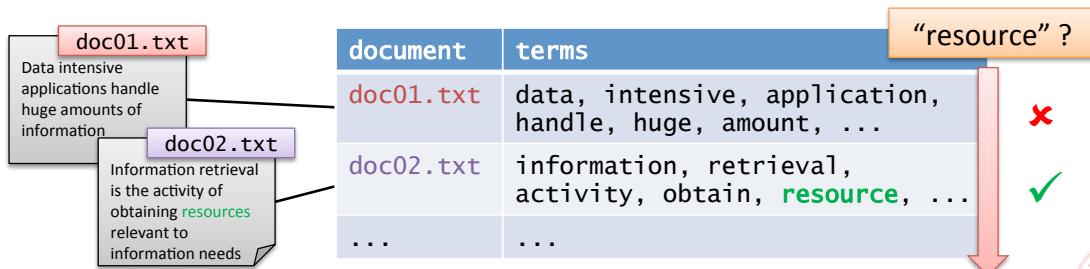
Modelli con Dipendenze tra Termini

- Nei modelli discussi finora, ciascun termine è considerato a se stante e **indipendente** da tutti gli altri
- In altri modelli (che non vedremo per ragioni di tempo) sono considerate esplicitamente **dipendenze reciproche tra termini**
 - Questo può servire ad es. a reperire documenti con termini **sinonimi** o comunque correlati a quelli dati esplicitamente dall'utente
- Nei casi più semplici le dipendenze sono *intrinseche*, dedotte statisticamente dalla collezione di documenti stessa
 - La ***Latent Semantic Analysis*** (LSA) traspone documenti e query in un nuovo spazio dove le dimensioni corrispondono idealmente a **concetti** semantici, ciascuno dato da un autovettore nello spazio originale
- In casi più avanzati, le dipendenze tra termini sono **trascendenti**, definite da una base di **conoscenza esterna**



Ricerca di un Termine

- Per ottenere la lista di documenti rilevanti ad una query, bisogna individuare in quali compaiono i termini ricercati
- La soluzione più facile è una **ricerca sequenziale su tutti i documenti**, verificando la rilevanza di ciascuno alla query
- Una simile ricerca richiede però **tempi lunghi** se i documenti sono in grande quantità
 - complessità **$O(N)$** , con N = numero di documenti su cui cercare



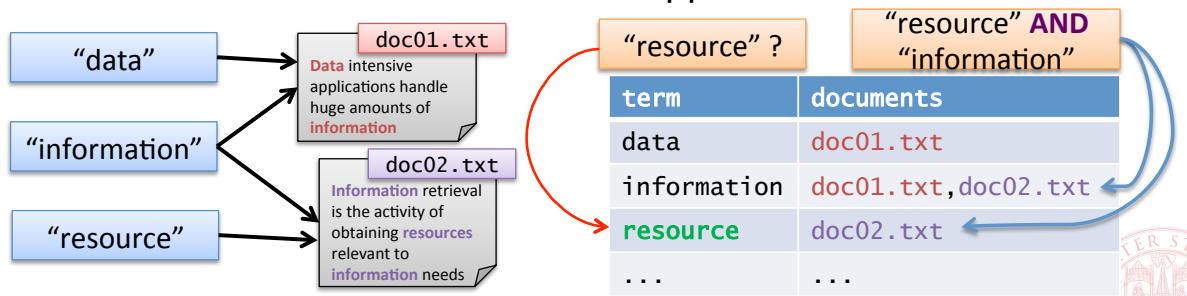
Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

42



Indicizzazione

- Per ricerche più efficienti si utilizzano degli **indici**
- Una struttura tipica è l'indice “inverso” (**inverted index**) che associa **ad ogni termine la lista di documenti** in cui compare
- Dato un termine, l'accesso avviene in un **tempo minimo e costante $O(1)$** alla lista di documenti
 - come accade ad es. per `HashMap.get(chiave)` in Java
- L'indice si costruisce e **mantiene aggiornato** associando un documento a ciascun termine che appare in esso



Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

43



Customer Satisfaction da Dati Testuali

- In un'attività di e-commerce, è importante sapere il **grado di soddisfazione dei clienti** in relazione ai singoli prodotti
- A questo fine, le loro **recensioni** sono un dato importante
- Se le recensioni hanno un **punteggio numerico** come nel nostro caso, è facile misurare il gradimento di un prodotto
- In altri casi, può essere necessario (o comunque utile) estrarre informazioni da **ciò che gli utenti scrivono**
 - Si possono estrarre informazioni da fonti di solo testo, ad es. post pubblici su forum e social network
- Vediamo in seguito un paio di metodi semplici per stimare la soddisfazione degli utenti dai loro messaggi di testo



Customer satisfaction: parole chiave

- Leggendo le recensioni, si possono spesso individuare **parole specifiche** che indicano la soddisfazione o meno dell'utente
 - ★ ★ ★ ★ ★ ***Capolavoro!*** *Un film assolutamente consigliato.*
 - ★ ★ ★ ★ ★ ***Film orribile,*** *non sono riuscito a finirlo.*
 - ★ ★ ★ ★ ★ ***Ottimo film,*** *nonostante alcuni attori mediocri.*
- Un metodo semplice per valutare una recensione è verificare la presenza di queste parole
- Una possibilità consiste nell'usare **liste predefinite** di parole e verificare quali di queste sono presenti
- Un altro approccio consiste nell'**analizzare automaticamente** recensioni etichettate per estrarne un **modello di conoscenza**



Customer satisfaction: conteggio delle parole chiave

- Possiamo creare una lista di termini usati per esprimere pareri **positivi** e una di termini per esprimere pareri **negativi**
- Estraendo le singole parole da un testo, possiamo contare quanti termini delle due liste sono presenti
- Confrontando i conteggi, possiamo avere una stima del grado di soddisfazione espresso nel testo

Parole positive

acclaimed decent
 adore excellent
 amazing fabulous
 awesome great
 best hilarious
 cheapest innovation

Parole negative

abominable deplorable
 annoying downer
 bore frustrating
 chaotic hate
 cheesy high-priced
 creepy imperfect



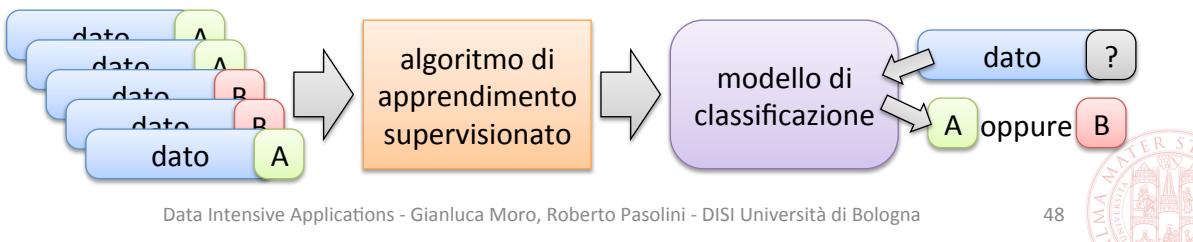
Customer satisfaction: estrazione automatica delle parole chiave

- La ricerca di parole chiave predefinite richiede la compilazione accurata delle liste di parole da cercare
- Queste liste sono laboriose da scrivere, è difficile che siano complete e sono da riscrivere per ciascuna lingua
- Inoltre, se si ha a che fare con diverse categorie di prodotti, esistono molti termini **specifici per ciascuna categoria**
 - aggettivi come “**thrilling**” e “**boring**” hanno senso per un libro o un film, ma non per un elettrodomestico o uno smartphone
 - “**small**” **positivo** per una fotocamera, **negativo** per una stanza d’albergo
- Sarebbe utile, avendo delle recensioni, **estrarre in automatico conoscenza** sulle parole usate nel contesto specifico
 - Usando recensioni **pre-etichettate come positive e negative**, si potrebbero cercare le parole chiave tra quelle ricorrenti



Machine learning e classificazione

- Le tecniche di **machine learning** permettono di **apprendere conoscenza** dai dati per **effettuare predizioni** su di essi
- Un problema comune consiste nel **classificare** i dati, ovvero dividerli tra due o più gruppi (**classi**) secondo criteri specifici
- Per automatizzare questa operazione si possono usare algoritmi di **apprendimento supervisionato**
 - L'algoritmo analizza un insieme di dati (*training set*) pre-etichettati con i rispettivi gruppi di appartenenza ed estrae un **modello di conoscenza**
 - Il modello è usato come classificatore per predire le classi di appartenenza di dati simili



Algoritmi di apprendimento supervisionato

- Esiste una grande varietà di algoritmi di apprendimento, che generano diversi tipi di modelli di conoscenza
 - modelli probabilistici** (es. *naïve Bayes*): calcola la classe di appartenenza più probabile in base alle caratteristiche dell'oggetto
 - k-nearest neighbor**: verifica quale sia la classe più ricorrente tra i k oggetti del training set con caratteristiche più simili a quello dato
 - ...e altri (alberi decisionali, **Support Vector Machines**, reti neurali, ...)
- È necessario definire le caratteristiche (**feature**) degli oggetti, tramite un'adeguata **rappresentazione strutturata**
- Una soluzione tipica è rappresentare ciascun oggetto come un **vettore** in un unico spazio multidimensionale
 - Ogni dimensione corrisponde ad una possibile *feature* degli oggetti
 - In questo modo è facile ad es. calcolare la similarità tra oggetti

Customer satisfaction tramite machine learning

- Col machine learning è possibile **estrarre un classificatore da recensioni pre-etichettate** come positive e negative
- Il modello ottenuto potrà essere usato per stimare la polarità di altre recensioni dello stesso contesto
 - L'algoritmo di classificazione individua i termini con maggiore
- Per trattare i testi, è necessario rappresentarli come vettori: possiamo usare il **Vector Space Model** visto in precedenza
 - La presenza di ciascun termine costituisce una *feature* dei testi
 - Per confrontare due testi tra loro si può usare la similarità coseno
 - A seconda della soluzione specifica, possiamo tenere conto o meno della specifica rilevanza delle parole all'interno dei testi (es. tf-idf)



Customer satisfaction: uso di informazione esterna

- Nel caso di un e-commerce, pareri sui prodotti sono dati dalle recensioni pubblicate dagli utenti del sito
- Tuttavia, sul Web si possono trovare molte altre informazioni, da cui si potrebbe valutare meglio la reputazione dell'oggetto
 - pareri più o meno sommari su social network
 - recensioni e discussioni approfondite su blog e forum specializzati
- Un problema importante è **reperire tutte le informazioni rilevanti** per ogni prodotto in modo sistematico
- Molti servizi online danno **l'accesso via API** alle informazioni
 - In questo modo altri servizi possono farne uso in modo automatico



Esempio di fonte di informazione esterna: Twitter

- **Twitter** è un servizio di social networking (o *microblogging*) con mezzo milione di utenti in tutto il mondo
 - Gli utenti possono pubblicare messaggi (*tweet*) di max. 140 caratteri, che possono essere etichettati per argomento tramite gli *hashtag*
- Twitter fornisce un [API per l'accesso ai tweet pubblicati](#)
 - L'API accetta interrogazioni, ciascuna costituita da una stringa di ricerca e da altri parametri opzionali (data pubblicazione, lingua, località, ...)
 - Ogni interrogazione restituisce un numero limitato di tweet, dei quali sono dati tutti i dettagli (data e ora, utente, numero di risposte, ...)
- Da una ricerca sugli ultimi tweet pubblicati, otteniamo [informazioni in tempo reale](#) su qualsiasi prodotto, da cui possiamo misurare il grado di soddisfazione verso di esso



Twitter: accesso via API

- Per usare le API Twitter da un'applicazione esterna, è necessario registrare l'applicazione stessa su Twitter
- Uno sviluppatore può accedere col suo account Twitter a apps.twitter.com per registrare le sue applicazioni
- Ad un'applicazione registrata sono assegnate delle chiavi d'accesso con cui autenticarsi all'API prima di eseguire richieste
- L'API si utilizza normalmente tramite HTTP: ad ogni richiesta corrisponde una risposta in formato strutturato (es. JSON)
 - ma esistono librerie per semplificare l'uso, vedi prossima slide...
- Twitter limita l'uso dell'API per ciascuna applicazione ad alcune centinaia di query all'ora



Customer satisfaction: spunti per l'approfondimento

- Abbiamo proposto due soluzioni molto semplici alla stima della customer satisfaction su un insieme di testi
 - Conteggio di parole con connotazione positiva/negativa note a priori
 - Uso di machine learning e VSM su recensioni pre-etichettate
- Questi approcci si dimostrano discretamente accurati su recensioni e pareri reali
 - Circa 78% di accuratezza su dataset di 20.000 recensioni dai nostri dati
- Esistono però anche soluzioni più avanzate che garantiscono in genere un'accuratezza maggiore nell'analisi dei testi
- Una limitazione importante è ad esempio considerare le singole parole ignorandone l'ordine e le frasi che formano
 - In questo caso le negazioni sono un problema: ad es. “**not bad**” viene erroneamente giudicato **negativo** per la presenza del termine “**bad**”
 - Altre difficoltà: confronti tra oggetti, frasi ipotetiche, sarcasmo, ...



Caso di Studio: Ricerca di Prodotti Medicali

- Azienda della Regione Emilia Romagna che distribuisce farmaci e prodotti medicali per ospedali
 - circa 100 mila prodotti, ciascuno rappresentato con una descrizione di testo destrutturato di circa 50 caratteri
 - testo brevissimo e qualitativamente scadente:
 - contiene abbreviazioni di medesimi termini difformi tra loro
 - unità di misura quantità pesi volumi immersi nel testo
 - termini distinti spesso concatenati come se fossero un'unica parola
 - marche e nomi di prodotti indistinguibili dal resto del testo
 - difficoltà anche solo per fare la tokenizzazione
- Problema
 - Realizzare un motore di ricerca con cui cercare prodotti rilevanti rispetto ad interrogazioni formulate come con motori di ricerca Web
 - L'interrogazione deve restituire una lista di prodotti ordinati per rilevanza



Ricerca di Prodotti Medicali: Soluzione

- Obiettivo: soddisfare interrogazioni a set di keyword restituendo prodotti ordinati per rilevanza
- data la minima quantità di testo disponibile e la bassa qualità, occorre trasformare il testo in trigram e unigram
- prima occorre eliminare simboli, punteggiatura (opzionalmente i numeri) gli url, separatori etc.
- si costruiscono due bag of word
 - Trigram x Parole e Parole x Prodotti
 - moltiplicandole si ottiene la matrice trigram x prodotti
- si applica tfidf
- l'interrogazione q con k termini riceve analogo preprocessing e diventa un vettore di trigram (come se fosse un prodotto)
- infine si cercano i prodotti simili a q con similarità coseno o jaccard superiore ad una soglia fissata



Ricerca di Prodotti Medicali: Soluzione

Discovering Product Similarity From Their Textual Descriptions

Choose a dataset:
partially pre-processed

Choose a term weighting:
Binary Frequency

Threshold similarity between 0 and 1:
0.6

Write your free text query:
abbaSSoS lingue sterile

SELECTED DATASET SUMMARY

```
<<TermDocumentMatrix (terms: 3433, documents : 1184)>>
Non-/sparse entries: 30791/4033881
Sparsity           : 99%
Maximal term length: 3
Weighting          : binary (bin)
```

ORIGINAL DATASET

	id_prodotto	descrizione_estar
1	141	ABBASSALINGUA IN LEGNO STERILE 2470
2	308	ACCU-CHEK FLEXLINK CANNULA 8MM *04626567001*
3	315	ACCU-CHEK INSIGHT FLEX CAN 8 *6541810001*

Showing 1 to 3 of 1,184 entries

Previous 1 2 3 4 5 ... 395 Next

SELECTING A PRODUCT **FREE TEXT SEARCHING**

Products of the Selected Dataset

	id_prodotto	descrizione_estar	provenienza	codice_estar
1	141	abbassalingua in legno sterile 2470	NO	38667
2	308	acchu-cheek flexlink cannula 8mm 04626567001	NO	68154
3	315	acchu-cheek insight flex can 8 6541810001	NO	68165

Showing 1 to 3 of 1,184 entries

Previous 1 2 3 4 5 ... 395 Next

Similar Products to the Selected Above

	similarity	id_prodotto	descrizione_estar	provenienza	codice_estar
1	0.977008420918394	144	ABBASSALINGUA STERILE M.U. IN LEGNO REF 2470	UN	50143749

Showing 1 to 1 of 1 entries

Previous 1 Next



NLTK

- **NLTK (Natural Language Toolkit)** è una libreria Python per il processamento di testo in linguaggio naturale
- Offre diversi algoritmi per l'interpretazione sintattica e semantica di testi in diverse lingue
 - è possibile utilizzare specifici algoritmi, impostarne i parametri e generare manualmente modelli per l'interpretazione del testo
 - in alternativa, per ogni task (es. segmentazione) sono offerte funzioni basate su parametri default e modelli preaddestrati per l'uso rapido
- NLTK permette di scaricare diversi set di dati testuali (*corpora*) e basi di conoscenza da utilizzare negli algoritmi



NLTK: Scaricare Dataset

- NLTK fornisce funzionalità generali per lo scaricamento e l'utilizzo di *risorse*
 - modelli e basi di conoscenza su cui si basano alcuni algoritmi
 - set di dati testuali (*corpora*) per addestrare e testare modelli
- Per scaricare una risorsa specifica, utilizzare la funzione `nltk.download` indicandone il nome
 - se non indicato nulla, è mostrata una GUI con le risorse scaricabili
- Le risorse sono salvate in una directory “`nltk_data`” nella propria home
- Nel seguito, riporteremo le chiamate a `download` necessarie per fare funzionare gli algoritmi discussi
 - è sufficiente eseguire `download` una sola volta per ciascuna



NLTK: Segmentazione in Parole

- Gli algoritmi di segmentazione sono nel package `tokenizer`
 - i token estratti sono restituiti come lista di stringhe
- `wordpunct_tokenizer` segmenta il testo in parole e punteggiatura basandosi su semplici espressioni regolari
`>>> text = "I wouldn't pay $2.50 for this."`
`>>> nltk.tokenize.wordpunct_tokenize(text)`
`['I', 'wouldn', "'", 't', 'pay', '$', '2', '.', '50', 'for', 'this', '.']`
- `word_tokenizer` utilizza invece un modello pre-addestrato su testo esistente per ottenere risultati più accurati

```
>>> nltk.download("punkt")
>>> nltk.tokenize.word_tokenize(text)
['I', 'would', "n't", 'pay', '$', '2.50', 'for',
'this', '.']
```

“wouldn’t” è scomposta correttamente in due parole

il numero decimale “2.50” è considerato un’entità unica

Data Intensive Applications - Gianluca Moro, Roberto Pasolini - DISI Università di Bologna

60



NLTK: Part-of-Speech Tagging

- Per eseguire il POS tagging tramite un algoritmo di default si usa la funzione `pos_tag`
 - l’input è una lista di token ottenuta dalla segmentazione
 - l’output è una lista di tuple (`token, tag`)
- Si può selezionare il set di POS da usare col parametro `tagset`
 - default: Penn Treebank (specifici per l’inglese)
 - “`universal`”: tag generici (NOUN, VERB, ADJ, ...)
- Per utilizzare `pos_tag` vanno scaricati i modelli relativi
`>>> nltk.download('averaged_perceptron_tagger')`
`>>> nltk.download('universal_tagset')`



NLTK: Esempio di POS Tagging

```
>>> text = "I wouldn't pay $2.50 for this."
>>> tokens = nltk.tokenize.word_tokenizer(text)

>>> nltk.pos_tag(tokens)
# con Penn Treebank
[('I', 'PRP'),
 ('would', 'MD'),
 ("n't", 'RB'),
 ('pay', 'VB'),
 ('$', '$'),
 ('2.50', 'CD'),
 ('for', 'IN'),
 ('this', 'DT'),
 ('.', '.')]

>>> nltk.pos_tag(tokens,
postag="universal")
[('I', 'PRON'),
 ('would', 'VERB'),
 ("n't", 'ADV'),
 ('pay', 'VERB'),
 ('$', '.'),
 ('2.50', 'NUM'),
 ('for', 'ADP'),
 ('this', 'DET'),
 ('.', '.')]
```



NLTK: Filtrare e Trasformare i Token

- Per le operazioni più semplici di trasformazione e filtraggio dei token, si possono utilizzare le normali funzionalità di Python
- Per rimuovere la punteggiatura e altri elementi non voluti, eliminarne i token dalla lista restituita dalla segmentazione
 - con POS tagging, si possono eliminare in base al tag
 - altrimenti si possono verificare ad es. con espressioni regolari
- Per il casefolding, usare il metodo `lower` delle stringhe
- Per la rimozione stopword, si può verificare la presenza delle parole in una lista fornita da NLTK

```
>>> nltk.download("stopwords")
>>> nltk.corpus.stopwords.words("english")
['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
 'ourselves', 'you', "you're", "you've", ...]
```



NLTK: Lemmatizzazione

- NLTK offre un modulo di lemmatizzazione basato su WordNet

```
>>> nltk.download("wordnet")
>>> wn1 = nltk.stem.WordNetLemmatizer()
```

- Per lemmatizzare un termine è necessario indicarne la POS

- “n” = nome, “v” = verbo, “a” = aggettivo

```
>>> wn1.lemmatize("cars", "n")
'car'
>>> wn1.lemmatize("mice", "n")
'mouse'
>>> wn1.lemmatize("ate", "v")
'eat'
```



NLTK: Stemming

- NLTK offre diversi algoritmi di stemming

- per lo stemming non è necessario indicare la POS della parola, in quanto si basa esclusivamente sulla morfologia

- L'esempio più noto è l'*algoritmo di Porter*

```
>>> ps = nltk.stem.PorterStemmer()
>>> ps.stem("jumping")
'jump'
>>> ps.stem("jumps")
'jump'
>>> ps.stem("strange")
'strang'
```



NLTK/WordNet: Ricerca di Synset

- Si può usare NLTK per cercare termini e relazioni in WordNet

```
>>> nltk.download("wordnet")
>>> from nltk.corpus import wordnet as wn
```
- Si possono cercare i **synset** in cui appare un termine dato
 - si può filtrare per POS col parametro opzionale **pos**
 - le POS sono “n”=nome, “v”=verbo, “a”=aggettivo, “r”=avverbio

```
>>> wn.synsets("car", pos="n")
[Synset('car.n.01'), Synset('car.n.02'), ...]
```
- Ogni oggetto **Synset** è identificato da una stringa
 $\text{"primoTermine.pos.numero"}$
- Si può ottenere un singolo **synset** data la stringa identificativa

```
>>> car = wn.synset("car.n.01")
```



NLTK/WordNet: Attributi dei Synset

- Dato un synset, se ne può ottenere la **definizione (gloss)**

```
>>> car.definition()
'a motor vehicle with four wheels; usually propelled
by an internal combustion engine'
```
- I **lemmi** (le parole contenute nel synset) possono essere
 ottenuti sia in forma di oggetti **Lemma** che di stringhe
 - ogni lemma è identificato dalla stringa "idSynset.parola"

```
>>> car.lemmas()
[Lemma('car.n.01.car'), Lemma('car.n.01.auto'),
 Lemma('car.n.01.automobile'), ...]
```

```
>>> car.lemma_names()
['car', 'auto', 'automobile', 'machine', 'motorcar']
```



NLTK/WordNet: Relazioni

- Gli oggetti `Synset` e `Lemma` offrono metodi per ottenere gli oggetti correlati in base alle relazioni definite in WordNet

```
>>> car.hypernyms()      # cos'è un'automobile?
[Synset('motor_vehicle.n.01')]

>>> car.hyponyms()      # che tipi di auto esistono?
[Synset('ambulance.n.01'),
 Synset('beach_wagon.n.01'),
 Synset('bus.n.04'), ...]

>>> car.part_meronyms() # che parti ha un'auto?
[Synset('accelerator.n.01'),
 Synset('air_bag.n.01'),
 Synset('auto_accessory.n.01'), ...]
```



NLTK: Word Sense Disambiguation

- La funzione `lesk` permette di ottenere il `Synset` più coerente con una determinata parola in una frase (lista di token)

– è possibile specificare una `pos` o provarle tutte

```
>>> p1 = "Lead is a very soft, malleable metal"
>>> p2 = "That actor plays the lead in the movie"
>>> word = "lead"
>>> s1 = nltk.wsd.lesk(nltk.word_tokenize(p1), word)
>>> s1.definition()
'a soft heavy toxic malleable metallic element; ...'
>>> s2 = nltk.wsd.lesk(nltk.word_tokenize(p2), word)
>>> s2.definition()
'an actor who plays a principal role'
```



scikit-learn: Estrazione Bag of Words

- Funzionalità per la rappresentazione di testi in forma BOW e Vector Space Model sono fornite da scikit-learn
- `CountVectorizer` converte documenti di testo in vettori basati sui semplici conteggi di parole
- `TfidfVectorizer` funziona in modo simile, ma calcola il tf.idf di ogni termine in ogni documento
- Entrambi supportano molti parametri opzionali
 - ad es. `min_df` indica un numero minimo di documenti in cui deve apparire un termine perché sia considerato

```
>>> from sklearn.feature_extraction.text
      import CountVectorizer
>>> vect = CountVectorizer(min_df=3)
```



scikit-learn: Costruzione del Dizionario

- Il dizionario di parole da considerare nei vettori inizialmente non è definito: va costruito in base ai documenti
- Per convertire dei documenti definendo il dizionario da usare in base ad essi, usare il metodo `fit_transform`
 - l'input è una lista con i documenti in forma di stringhe

```
>>> docs = ['the sky is blue',
          'sky is blue and sky is beautiful',
          'the beautiful sky is so blue',
          'i love blue cheese']
>>> dtm = vect.fit_transform(docs)
```

- Con `get_feature_names` si ottiene il dizionario estratto

```
>>> vect.get_feature_names()
['and', 'beautiful', 'blue', 'cheese', 'is', 'love',
 'sky', 'so', 'the']
```



scikit-learn: Matrice Documenti-Termini

- `fit_transform` restituisce la matrice documenti-termini dove ogni riga corrisponde al BOW di un documento
 - è una matrice *sparsa*, dove sono rappresentati in memoria solo i valori non nulli: usare `toarray` per convertirla ad un normale array NumPy
- ```
>>> dtm.toarray()
array([[0, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 0, 2, 0, 2, 0, 0],
 [0, 1, 1, 0, 1, 0, 1, 1, 1],
 [0, 0, 1, 1, 0, 1, 0, 0, 0]], dtype=int64)
```
- Questa matrice può essere usata come input per addestrare o testare un modello di scikit-learn (es. `LinearRegression`)
  - Per convertire in vettori altri documenti utilizzando lo stesso dizionario, usare il metodo `transform`



# scikit-learn: Estrazione BOW con NLTK

- Il pre-processing eseguito da scikit-learn di default su ciascun documento è basilare
  - divisione in token, casefolding, rimozione stopword (opzionale)
- Per estenderlo, si può specificare col parametro `tokenizer` una funzione arbitraria che converta un testo in lista di token
- Da questa funzione si possono impiegare gli algoritmi di NLTK per eseguire stemming, lemmatizzazione, filtro per POS, ecc.

```
es. con tokenizzazione, casefolding e stemming
ps = nltk.stem.PorterStemmer()
def my_tokenizer(text):
 return [ps.stem(token.lower()) for token
 in nltk.tokenize.word_tokenize(text)]
vect = TfidfVectorizer(tokenizer=my_tokenizer)
```



## scikit-learn: Parametri per Costruzione Dizionario

- Col parametro `stopwords` del vectorizer si può indicare una lista di parole da escludere dalle feature
  - con la stringa “english” si usa una lista di stopword inglesi predefinita
  - in alternativa si può passare una lista arbitraria, presa ad es. da NLTK
- Col parametro `ngram_range` si indica in una tupla i numeri minimo e massimo di parole da usare per estrarre n-gram
  - ad es. con `(1, 1)` (default) si estraggono solo parole singole, con `(1, 2)` parole singole e bigram, con `(3, 3)` solo trigram, ecc.

```
>>> vect = CountVectorizer(ngram_range=(2, 2))
>>> dtm = cv.fit_transform(docs)
>>> vect.get_feature_names()
['and sky', 'beautiful sky', 'blue and', ...]
```

