Tecnologie del linguaggio naturale

Appunti delle lezioni di
Paolo Alfano

Note Legali

Liberatoria, aggiornamenti, segnalazione errori:

Quest'opera viene pubblicata in formato elettronico senza alcuna garanzia di correttezza del suo contenuto. Il documento, nella sua interezza, è opera di Paolo Didier Alfano
e viene mantenuto e aggiornato dallo stesso, a cui possono essere inviate eventuali segnalazioni all'indirizzo *paul15193@hotmail.it*
Ultimo aggiornamento: 09 luglio 2018

Indice

Parte Prima:
Linguistica computazionale generale

26 / 02 /2018

1 Introduzione

Per cominciare andremo ad effettuare una veloce analisi del linguaggio naturale. Il linguaggio umano è discreto poiché costituito da fonemi mentre altri esseri viventi[1] sfruttano un linguaggio continuo. Inoltre il linguaggio umano è ricorsivo mentre quello di altri esseri viventi è formato da gesti atomici. Esiste inoltre una dipendenza dalla struttura e una forma di località del linguaggio.
Una prima forma di intelligenza ricercata nelle macchine e legata al linguaggio è quella mostrata nel *test di Turing*. Anche se in un certo senso una macchina che superi il test di Turing possa essere "intelligente", sono state sollevate diverse critiche a questo test da cui sono stati sviluppati diversi altri test. Ad esempio ad oggi vengono spesso utilizzati dei chat-bot e degli assistenti vocali personali presenti su quasi ogni smartphone.

Un aspetto importante da considerare -e lo faremo in modo più approfondito in seguito- è che il linguaggio è strutturato secondo *livelli di conoscenza*. Esistono sei livelli del linguaggio:

1. Fonetica e fonologia: lo studio della linguistica del suono
2. Morfologia: lo studio del significato dei componenti delle parole
3. Sintassi: lo studio delle relazioni strutturali tra le parole
4. Semantica: lo studio del significato
5. Pragmatica: come il linguaggio viene usato per raggiungere un goal
6. Discorso: lo studio delle unità linguistiche non atomiche

Ognuno di questi livelli ci fornisce qualche informazione riguardo le strutture linguistiche di una frase. Ogni struttura è un insieme su cui è definita una relazione

27 / 02 /2018

Quale è invece la differenza tra un linguaggio come il Java, il C++ e il nostro linguaggio naturale? La risposta è che il nostro linguaggio è *ambiguo*. Esistono frasi che noi sappiamo contestualizzare ma che in realtà potrebbero avere moltissimi significati. Le ambiguità si riscontrano a tutti i livelli strutturali di cui abbiamo parlato. E oltre all'ambiguità che proviene dai livelli indicati, in ogni lingua abbiamo altre ambiguità notevoli: ad esempio quella di usare un linguaggio informale, oppure l'utilizzo di locuzioni, neologismi...

[1] Ad esempio le api

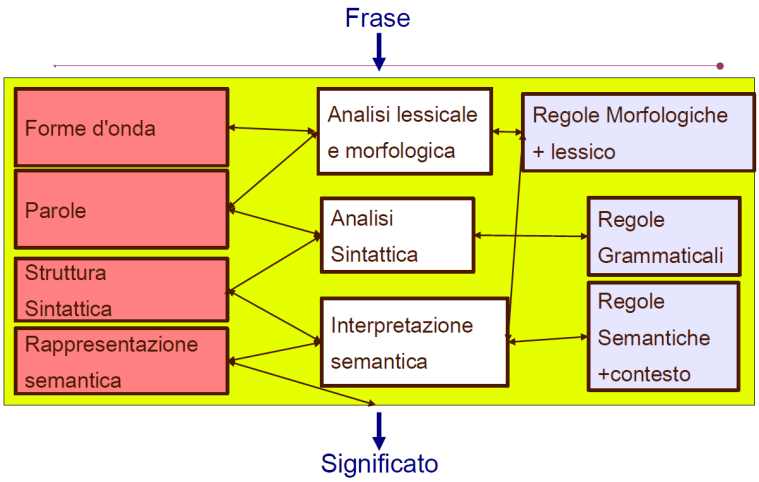


Figura 1: Processo di analisi di una frase

L'ultimo aspetto introduttivo da considerare riguarda lo stato dell'arte nel campo del natural language processing. Verso la fine degli anni '90 i programmi erano già in grado di leggere gli elaborati di centinaia di studenti e valutarli in modo indistinguibile da un umano. Ad oggi uno degli esempi più sorprendenti è Watson, un software sviluppato dalla IBM in grado di vincere giochi complessi come il quiz americano Jeopardy.
Comunque ci sono stati anche grandi sviluppi nell'ambito del riconoscimento del parlato e nella conversione da testo a parlato.

2 I livelli linguistici

Lo scopo di questa prima parte del corso è quello di sviluppare degli strumenti per l'elaborazione automatica del linguaggio naturale. Per questo dobbiamo capire come studiare una frase all'interno dei vari livelli di cui abbiamo parlato in precedenza. Questa operazione viene mostrata in Figura 1.

In questo capitolo percorreremo brevemente i livelli che analizzeremo nelle lezioni successive, per averne una prima visione intuitiva.
Prima di procedere nella discesa dei vari livelli dobbiamo capire che al momento esiste una forma di competizione nello studio di questi problemi tra AI a regole e AI statistica. Allo stesso tempo esiste anche una competizione tra linguistica computazionale e tecniche di deep learning. Per ogni problema esistono tanti modelli possibili: ad albero di decisione, logistic regression, SVM, reti neurali...
In realtà non è tanto il modello che andiamo ad utilizzare ad essere determinante, quanto invece le *features* che utilizziamo per descrivere il dominio. In questo corso faremo uso di feature che hanno un'evidenza cognitiva. Queste feature

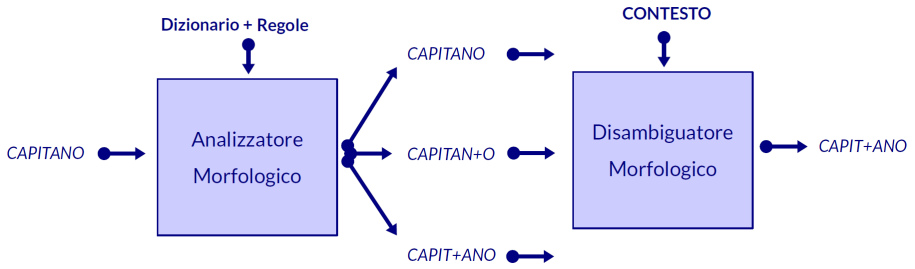


Figura 2: Analisi e disambiguazione morfologica

tendono a favorire l'approccio a regole ma possono essere usate in ogni sistema di machine learning.

2.1 Il livello morfologico e l'analisi lessicale

L'analisi lessicale parte ovviamente dal lessico che è fondato sul concetto di parola. Dunque cosa è una parola? Intuitivamente è una sequenza di caratteri delimitata da spazi o punteggiatura.
Ad ogni modo questa risposta non è del tutto sufficiente. Vediamolo con un esempio

Esempio 1 : in certi casi una parola ne esplica diverse e in altri casi si verifica il viceversa:
"passamela": che sta per "passa a me essa". Abbiamo una parola che ne esplica quattro.
"by the way": viceversa tre parole che nel loro insieme formano un unico concetto logico.

Un ulteriore problema è che in molte lingue abbiamo la presenza di suffissi che complica la questione

Esempio 2 : la parola "capitano" se cercata sul dizionario da diversi significati:
capitano (forma non declinabile)
capitan+o (nome o aggetivo o forma del verbo capitanare)
capit+ano (forma del verbo capitare)

Per tutti questi motivi solitamente viene utilizzato un analizzatore morfologico che, dato un token in ingresso fornisce diverse possibili analisi morfologiche in uscita. Quale sia la giusta analisi viene poi stabilito da un disambiguatore morfologico. L'intero procedimento viene mostrato in Figura 2

In questo contesto diventa importante la differenza tra operazione di *lemmatizzazione* e operazione di *stemming*, le due operazioni che permettono all'analizzatore morfologico di operare:

- Lemmatizzazione: riduzione dei vocaboli ad una forma standard. Ad esempio riportare tutti i verbi all'infinito

- Stemming: ridurre una parola alla propria radice

Ad oggi l'analisi morfologica -sebbene non sia un processo banale- è trattata abbastanza bene tramite una serie di librerie: Xerox Finite State Technology Tools, Morph-it!, TULE...

02 / 03 /2018

Visto che abbiamo parlato di analizzatore morfologico dovremmo chiederci, quali sono le possibili componenti di una frase? Ovvero, quali sono le *parti del discorso*?[2]
A scuola ci hanno insegnato che le parti del discorso sono: nomi, verbi, aggettivi, avverbi, preposizioni, articoli, pronomi, congiunzioni e interiezioni. Queste categorie possiedono due proprietà importanti, infatti sono

- Paradigmatiche: se andiamo a sostituire un elemento di una categoria con un altro elemento della categoria la frase è ancora valida
- Sintagmatiche: tra i vari vocaboli che si susseguono all'interno di una frase esiste una relazione sintagmatica che studieremo nel livello sintattico

Una distinzione interna che dobbiamo fare è che le parti del discorso possono essere *aperte* o *chiuse*

- Aperte: ogni giorno vengono aggiunti nuovi elementi a tali categorie. Ad esempio i nomi, i verbi, gli aggettivi e gli avverbi sono categorie aperte poiché molto spesso vengono inventate nuove parole come i neologismi e altro ancora
- Chiuse: le aggiunte alle categorie chiuse sono molto più rare, dunque i cambiamenti in queste categorie sono molto più lenti. Le classi chiuse vengono anche dette funzionali perché non fanno balenare in mente un immagine in modo diretto. Hanno invece un ruolo che è funzione di altre parti del discorso

Un'altra distinzione che possiamo fare è tra categoria *di contenuto* o *non di contenuto*. Infatti diremo che una parte del discorso è di contenuto se possiede una propria semantica. I nomi e i verbi ad esempio lo sono, gli articoli no perché non hanno un loro immediato significato.
Ad ogni modo le distinzioni viste finora non sono nette. Alcuni verbi come i modali[3] sono chiusi e non aperti. Uno schema generale è visibile in Figura 3

Inoltre il numero di parti del discorso non è fissato a nove come da noi sotto inteso fino ad ora. Alcuni studi ne hanno individuate dodici, altri ancora addirittura una trentina.
Comunque stabilire quanti parti del discorso abbiamo è importante e ci permette

[2] In inglese vengono dette *part of speech* da cui l'acronimo PoS
[3] Verbi che precisano la relazione tra il soggetto e il verbo che li segue: ad esempio "*Devo* studiare!"

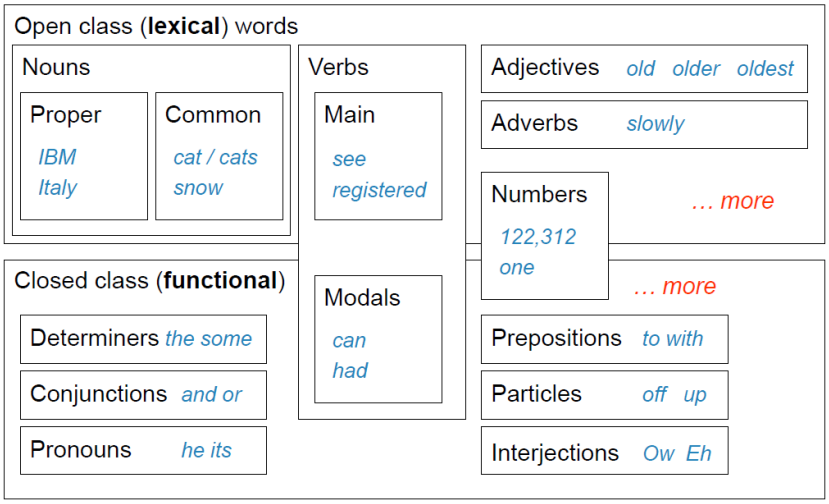


Figura 3: Parti del discorso aperte e chiuse

di capire in quante categorie potremmo suddividere gli elementi con i nostri strumenti automatici. L'operazione di assegnamento di una parola a una delle categorie viene detta *PoS tagging*.

2.2 Livello sintattico

Quando parliamo del livello sintattico ci riferiamo alle relazioni sintagmatiche nominate nelle pagine precedenti. Quando parliamo di sintassi esistono due tipi di strutture che possiamo individuare, la *consituency structure* e la *dependency structure*:

- Consituency structure: rappresenta dei raggruppamenti di unità linguistiche degne di nota
- Dependency structure: rappresenta le relazioni di dipendenza tra le parole

Mostriamo entrambe queste strutture in Figura 4

Cominciamo andando a considerare le consituency structure. Solitamente le frasi organizzano le parole in una serie di costituenti annidati. Ma come fare a capire se qualcosa è un costituente? Noam Chomsky propone un semplice test in cui un elemento è un costituente se è sostituibile con un altro elemento della stessa categoria e se a quel elemento ne posso concatenare un altro della stessa categoria senza invalidare la correttezza della frase. Vediamo un esempio

Esempio 1 : nella frase "Paolo ama Francesca", mostrata in Figura 4, la verbal phrase(VP) "ama Francesca" è un costituente perché al posto di essa

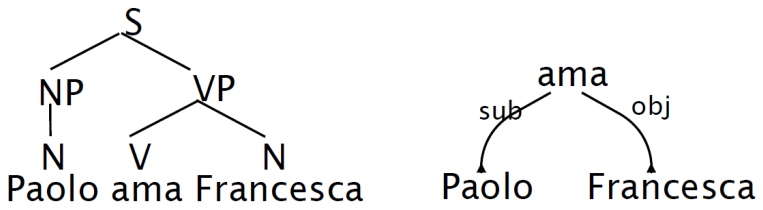


Figura 4: A sinistra una consituency structure, a destra una dependency structure

posso inserirvi un'altra VP come "odia Enrico" senza invalidare la correttezza della frase. Inoltre alla VP possiamo concatenarne un'altra senza invalidare la correttezza.

Una domanda interessante da porsi è: ma i costituenti sono una nostra completa invenzione o esistono in modo spontaneo? Un noto esperimento ad opera di Fodor-Bever sembra dimostrare che esistano indipendentemente dagli studi umani. L'esperimento consisteva nell'emettere un suono in corrispondenza dell'inizio di un nuovo costituente oppure all'inizio di parole casuali per comprendere se tale suono venisse percepito meglio in un caso o nell'altro. Questa teoria è stata confermata[4] dal fatto che i suoni venivano percepiti meglio in corrispondenza dell'inizio del nuovo costituente.

Per quanto riguarda le strutture di dipendenza esistono due componenti fondamentali: *head* che è la componente dominante e la *dependent* che è la parola dominata. La relazione tra head e dependent può essere di due tipi:

- Argomento: quando la parola dipendente è soltanto un argomento dell'head. Un esempio è "ama Francesca" in cui "Francesca" è solo argomento dell'head "ama".
- Modificatore: quando la dipendente va a modificare in qualche modo l'head. Un esempio è "corre velocemente" in cui la dipendente "velocemente" ci dice qualcosa in più sull'head "corre".

2.3 Livello semantico

Cosa succede quando un interlocutore comprende quello che gli stiamo dicendo? Sta eseguendo una trasformazione dal linguaggio naturale in una qualche forma di rappresentazione della conoscenza. Questo procedimento viene detto *interpretazione semantica*.
Se andiamo a definire come *lessema* la coppia (forma, significato) allora possiamo andare a definire due semantiche: la *semantica lessicale* che riguarda effettivamente il significato dei lessemi, e la *semantica formale*(o *composizionale*) che va ad analizzare la composizione dei vari lessemi.

[4] Anche se sono stati sollevati dei dubbi e alcune critiche a riguardo

Se cominciamo dalla semantica lessicale, sembra che sia molto importante andare a costruire i lessemi per studiare la semantica. Costruirli in modo automatico non è semplice perché se per esempio ad una certa forma assegniamo il significato contenuto nel dizionario per quel vocabolo abbiamo un problema di ricorsione circolare

Esempio 1 : cerchiamo di costruire il lessema per la parola "rosso". Dal dizionario abbiamo che

rosso=nome, il colore del sangue o di un rubino

Se però andiamo a prendere la definizione di "sangue" abbiamo che

sangue=nome, il liquido rosso che circola nel cuore nelle arterie e nelle vene degli animali

Notiamo che per definire rosso usiamo il sangue e per definire il sangue usiamo rosso. Questa situazione di ricorsione non chiusa è problematica

Questa situazione problematica si verifica perché stiamo cercando di definire le parole usando le parole.
Una soluzione potrebbe essere quella di stabilire che alcune parole sono *indefinibili* e andare a costruire i nostri lessemi partendo proprio da queste parole. Anche questo però è complicato perché non sarebbe facile stabilire quando una parola è indefinibile.
Questa situazione viene invece risolta andando a considerare le relazioni tra lessemi: omonimia, polisemia, sinonimia e iponimia. Vediamole tutte:

- Omonimia: quando due lessemi hanno stessa forma ortografica ma hanno significati molto diversi. Ad esempio "caccia" inteso come velivolo militare oppure come uccisione di animali selvatici
- Polisemia: quando due lessemi hanno stessa forma ortografica ed esprimono significati diversi. Ad esempio il fattore inteso come l'agricoltore oppure il membro di una moltiplicazione. La differenza con l'omonimia è che nella polisemia abbiamo una vera e propria estensione del significato a più di un vocabolo mentre nell'omonimia abbiamo una casuale ortografia identica.
- Sinonimia: quando due lessemi hanno diversa forma ortografica ma stesso significato
- Iponimia/iperonimia: quando tra i due lessemi esiste una struttura gerarchica. Ad esempio "automobile" è iponimo di "veicolo" mentre "veicolo" è iperonimo di "automobile"

Come dicevamo i vari lessemi vengono messi insieme mediante la semantica formale o composizionale. L'intuizione fondamentale è che la semantica di un

sintagma dipenda dalla semantica delle sue componenti. Vedremo che una delle soluzioni per studiare la semantica composizionale consiste nell'andare ad utilizzare il lambda calcolo come proposto da Montague nel 1974.

06 / 03 /2018

In tutto questo ricopre un ruolo importante il tipo di ragionamento che possiamo andare a fare. Abbiamo tre modi di ragionare:

- Deduzione: è il metodo più noto. Un esempio di deduzione è il sillogismo. A partire da un insieme di premesse vere ricaviamo delle conseguenze vere. In questo modo la deduzione non può commettere errori.
- Induzione: in modo opposto cerchiamo di ricavare delle informazioni a partire da quello che vediamo. In questo caso è possibile commettere errori.
- Abduzione: anche questo tipo di ragionamento non certo in cui partendo da una premessa certa e una premessa non certa, cerchiamo di ricavare una conclusione probabile.

L'ultimo tipo di semantica che andiamo a introdurre velocemente è la *semantica distribuzionale* con cui intuitivamente diciamo che il significato di una parola è legato alla distribuzione delle parole che lo circondano

Esempio 2 : consideriamo le tre frasi che seguono

> Una bottiglia di tesguino è sul tavolo
> A tutti piace il tesguino
> Il tesguino ti fa ubriacare
>
> Pur non sapendo cosa sia il tesguino(e infatti è una parola inventata), potremmo facilmente dedurre che sia una bevanda di natura alcolica.

Per gestire situazioni come questa, solitamente vengono usate delle *matrici di co-occorrenza*. Sia sulle righe che sulle colonne abbiamo delle parole e in posizione $[i, j]$ abbiamo un valore che indica quanto spesso la parola i compaia "vicino"[5] alla parola j.
L'utilizzo di questo approccio si è evoluto nel tempo: prima si utilizzavano vettori molto lunghi(decine di migliaia di elementi) e sparsi. Più di recente sono emersi dei vettori più corti(qualche centinaio di elementi) e densi. Questi vettori più corti sono più facili da utilizzare. Questa riduzione di dimensione, detta *embedding* è stata resa possibile dagli algoritmi di machine learning. Il più noto è skip-gram di Google ed è ispirato alle reti neurali. Infatti il procedimento viene detto *neural embedding*. Tramite questo algoritmo si ottengono dei vettori di vocaboli correlati fra loro. Ad esempio il vettore di lunghezza due collegato al

[5]Dove il concetto di vicino può essere definito a piacimento

vocabolo "Redmond", contiene le parole "Redmond Washington" e "Microsoft" a mostrare che vengono ritrovati dei vocaboli che hanno degli elementi in comune. Ancora più suggestivo è che tutti questi vettori sono in qualche modo collegati. Infatti è stato verificato che:

vector("king") - vector("man") + vector("woman") $\sim$ vector("queen")
vector("Paris") - vector("France") + vector("Italy") $\sim$ vector("Rome")

Ovvero se prendiamo il vettore relativo alla parola "king" e gli sottraiamo il vettore relativo alla parola "man" andandogli poi ad aggiungere quello relativo alla parola "woman" otteniamo un vettore molto simile a quello della parola queen.
Al momento non esistono applicazioni precise di quanto appena mostrato ma negli ultimi anni c'è stato un grande fermento per cercare di usare questi vettori per fare deduzione. Ovvero per cercare di verificare se da un vettore che esprima il concetto "Ogni uomo è mortale" unito al vettore "Socrate è un uomo" si possa ricavare il vettore "Socrate è mortale", anche se ad ora nessuno vi è riuscito.

2.4 Livello pragmatico

L'intuizione che sta dietro al livello pragmatico è la seguente: per interpretare correttamente una frase dobbiamo avere un certo insieme di informazioni.

Esempio 1 : se pronuncio la frase

> "Oggi mi trovo ad Alessandria"
>
> L'interpretazione di "io" (sottinteso) e "oggi" dipende da chi enuncia la frase e quando, rispettivamente. Abbiamo dunque bisogno di una rappresentazione del mondo che sia ricca.

In questo contesto sono importanti le *anafore*, ovvero dei sintagmi che si riferiscono a oggetti precedentemente menzionati. Le anafore devono essere gestite correttamente.
Nell'ambito della pragmatica hanno assunto sempre più importanza le reti neurali. Esistono infatti grandi collaborazioni tra studiosi di linguistica computazionale e studiosi di reti neurali. Infatti(secondo uno dei padri fondatori delle reti neurali, LeCun) ad oggi le reti sono in grado di fare apprendimento ma non ragionamento che per ora deve derivare dalla linguistica computazionale.

3 Analisi morfologica

Ricordiamo ancora una volta che lo scopo primo che cerchiamo di assolvere è di costruire un sistema di PoS tagging. Questo perché esistono diverse applicazioni di questo strumento: text-to-speach, permette di velocizzare i parser...

Attualmente i migliori algoritmi di PoS tagging ottengono percentuali di accuratezza intorno al 98% e considerando che gli algoritmi più semplici ottengono una percentuale del 90% non è per niente male. Un tipico esempio di algoritmo semplice associa ad un vocabolo il tag con cui è usato più di frequente.
Ricordando che i *tipi* sono tutte le categorie possibili e che i *token* sono le occorrenze, otteniamo statisticamente che circa il 10% dei tipi sono ambigui. Ma poiché questi tipi sono molto frequenti, coprono circa il 40% dei token.
Adesso andiamo a vedere i più comuni algoritmi per effettuare PoS tagging.Vedremo tre algoritmi di Pos tagging: *Rule-Based Tagger: ENGTWOL* (ENGlish TWO Level analysis), *Stochastic Tagger: HMM-based*, *Transformation-Based Tagger*(Brill).

3.1 ENGlish TWO Level analysis

Il primo algoritmo, ENGTWOL, è il più semplice ed intuitivamente effettua due analisi. Con una prima analisi morfologica assegna tutti i possibili tag alle parole. Con una seconda analisi va a rimuovere i tag impropri basandosi su un sistema di regole. Solitamente l'insieme delle regole è di dimensione elevata, formato da un migliaio di regole. Mostriamolo su di un esempio.

Esempio 1 : supponiamo di analizzare la frase "Pavlov had shown that salivation". Allora con la prima analisi otteniamo che

Pavlov PAVLOV N NOM SG PROPER
had HAVE V PAST VFIN SVO
had HAVE PCP2 SVO
shown SHOW PCP2 SVOO SVO SV
that ADV
taht PRON DEM SG
that DET CENTRAL DEM SG
that CS
salivation N NOM SG

A questo punto possiamo usare delle regole della forma

IF (and(+1 A/ADV/QUANT)(+2 SENT-LIM)(NOT -1 SVOC/A))
THEN eliminate non-ADV tags
ELSE eliminate ADV

Dove la condizione dello statement IF richiede che la parola successiva sia un aggettivo un avverbio o un quantificatore, che la parola ancora successiva sia un vincolo di frase e infine che la parola precedente non sia un verbo che permette l'uso di aggettivi come complementi.

3.2 Sequence classification

Il secondo algoritmo che viene utilizzato si chiama Stochastic Tagger: HMM-based. In questo caso l'output fornisce una visione probabilistica. L'intuizione è la seguente: supponendo di conoscere la distribuzione dei tag, dato un nuovo input vorremmo vedere con quale probabilità emerge un certo tag. Ovvero siamo interessati alla quantità che segue

$$\hat{t}_1^n = argmax_{t_1^n} P(t_1^n | w_1^n) \tag{1}$$

Ovvero stiamo cercando di massimizzare la probabilità che dato un insieme di n parole w_1^n si ottenga un certo vettore t_1^n.
Ovviamente si presuppone di avere a disposizione un corpus annotato con le varie probabilità delle singole parole. Ad ogni modo non è proprio la quantità appena esposta quella che andiamo a studiare. Infatti tramite la regola di Bayes otteniamo che:

$$\hat{t}_1^n = argmax_{t_1^n} P(t_1^n | w_1^n) = argmax_{t_1^n} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} = argmax_{t_1^n} P(w_1^n | t_1^n) P(t_1^n)$$

Dove il denominatore è stato eliminato visto che non contiene la quantità da massimizzare. Dunque abbiamo che

$$\hat{t}_1^n = argmax_{t_1^n} P(w_1^n | t_1^n) P(t_1^n) \tag{2}$$

In questa forma la quantità da massimizzare è più comoda perché composta da una likelihood e da una probabilità a priori.

09 / 03 /2018

In questa forma possiamo fare due approssimazioni molto utili

- $P(w_1^n | t_1^n) \approx \prod_i P(w_i | t_i)$
  Con questa prima approssimazione diciamo che la probabilità che da un certo insieme di tag si ottenga un certo insieme di parole è circa uguale al prodotto che da ognuna di quelle parole si ottenga quel tag
- $P(t_1^n) \approx \prod_i P(t_i | t_{i-1})$
  Con questa seconda approssimazione che viene detta *approssimazione markoviana* diciamo che il valore di un tag in una certa posizione dipende esclusivamente dal valore del tag precedente andando ad ignorare tutti i tag ancora precedenti e tutti i tag successivi

Dunque complessivamente

$$\hat{t}_1^n \approx \prod_i P(w_i | t_i) P(t_i | t_{i-1}) \tag{3}$$

Questa forma ci torna particolarmente utile perché entrambi i termini della produttoria sono facilmente calcolabili partendo da un training set, ovvero da un

corpus. Infatti:

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$
$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

Dove C è una funzione di conteggio. Per intenderci meglio, se ad esempio vogliamo sapere quale è la probabilità che ad un verbo t_{i-1} segua un nome t_i mi è sufficiente contare nel mio corpus quante volte ad un verbo segue un nome, e andare a dividere per il numero di verbi che compaiono nel corpus. Questa probabilità viene quindi calcolata in modo empirico ed è molto facile da calcolare. Dobbiamo comunque stare attenti al fatto che segue: esistono alcuni eventi che si verificano raramente nei testi. In un testo non particolarmente lungo potrebbe succedere che ad esempio un nome seguito da un nome non si presenta mai. Questo vuol forse dire che sia impossibile vedere un nome seguito da un nome? Ovviamente no e in certi casi dobbiamo tenerne di conto. Infatti se durante la fase di training assegniamo il valore 0 ad un certo evento perché non si è mai verificato, nel test set potremmo avere problemi perché probabilità pari a zero rischiano di annullare l'intera produttoria e questo sarebbe ovviamente sbagliato.
Il modello che utilizziamo in questo contesto è chiamato *Hidden Markov Model*(HMM) perché ad ogni passo lo stato che vogliamo individuare(ovvero il tag) è nascosto, non ci è noto. D'altronde viene anche detto di Markov perché ad ogni passo teniamo di conto soltanto del tag precedente.

Notiamo che esiste un parallelismo con gli automi a stati finiti in cui ad ogni tag associamo uno stato e ad ogni arco associamo la probabilità di transire da uno stato ad un altro. Notiamo che esisteranno poi due stati particolari S_i ed S_f che indicano rispettivamente l'inizio e la fine della frase da analizzare.

Nonostante sia molto facile calcolare la probabilità negli HMM, noi vogliamo prendere quella massima relativa ad un certo insieme di tag. Dunque tecnicamente dovremmo provare tutte le combinazioni di tag lungo una certa frase e prendere quella massima. Purtroppo questo metodo è impraticabile poiché di complessità esponenziale nel numero di tag. Preferiamo dunque usare delle tecniche di programmazione dinamica.
Considerata una sequenza di tag che termina ad un certo stato j con un certo tag T, la probabilità di quella sequenza di tag può essere spezzata in due parti: la probabilità che quella sia la miglior sequenza fino allo stato $j-1$ moltiplicata per la probabilità di transire dal tag dello stato $j-1$ al tag T.
Con questa tecnica permette di abbassare la complessità da esponenziale a quadratica nella lunghezza della frase da analizzare.
Questa tecnica viene infatti sfruttata nel'*algoritmo di Viterbi*. In questo algoritmo viene costruita una matrice che ha tante colonne quante sono le parole della frase da analizzare e ha tante righe quanti sono i tag possibili.

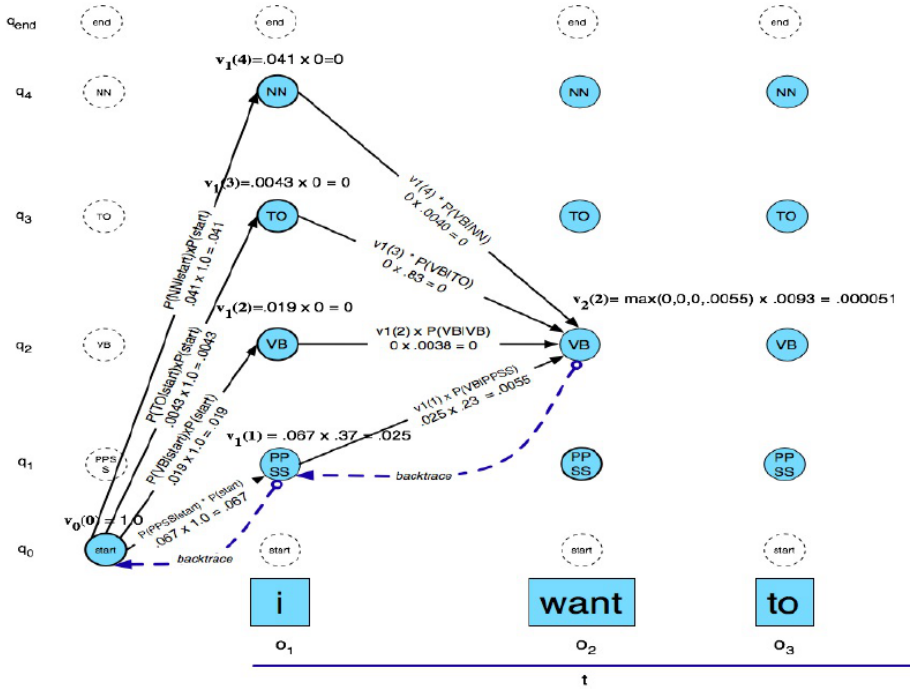


Figura 5: Evoluzione della matrice di Viterbi

Il valore associato ad ogni cella viene calcolato come segue:

$$v_t(j) = \max_{i=1}^{n} v_{t-1}(i) a_{ij} b_j(o_t) \tag{4}$$

ovvero prendiamo il massimo tra $v_{t-1}(i)$ ovvero quanto contenuto nelle celle della matrice nella colonna precedente(quindi la cella relativa al tag più probabile per la parola precedente) e lo andiamo a moltiplicare per la probabilità a_{ij} che indica la probabilità di transire da un tag relativo alla parola i al tag relativo alla parola j. Infine lo moltiplichiamo per la probabilità di emissione $b_j(o_t)$ (ma notiamo che non avendo il pedice in i questa quantità è al di fuori del massimo) che rappresenta la probabilità di osservare il simbolo o_t dato lo stato corrente j.
Ovviamente per gestire correttamente gli stati iniziali e finali dobbiamo anche avere a disposizione le probabilità $P(t|start)$ e $P(end|t)$ che indicano rispettivamente la probabilità di transire in un certo tag partendo dallo stato iniziale e la probabilità di transire allo stato finale da un certo tag.
Un esempio di applicazione dell'algoritmo di Viterbi viene mostrato in Figura 5

Comunque, tramite il modello di Markov andiamo ad effettuare delle approssimazioni del mondo reale. Potremmo però pensare di effettuare approssimazioni migliori andando a considerare ad esempio i due tag precedenti e non solo il precedente. In certi sistemi questo viene fatto ma presenta due svantaggi:

- Complessità: il modello da calcolare è più complesso, il costo computazionale aumenta
- Sparseness: i valori che andiamo a moltiplicare fra loro sono molto piccoli. Aggiungere ulteriori fattori comporta un avvicinamento della probabilità allo zero. Questo ci riconduce ai problemi di cui abbiamo già parlato per quanto riguarda probabilità prossime allo zero.

Per concludere possiamo effettuare un confronto tra l'Hidden Markov Model e il *Maximum Entropy Model*. Questo secondo modello viene detto *discriminativo* a differenza del primo che viene detto *generativo*. La differenza è che nei modelli generativi andiamo ad applicare la regola di Bayes mentre nei modelli discriminativi no. Infatti:

- Generativi: $\hat{T} = \arg\max_T P(T|W) = \arg\max_T P(W|T)P(T) = \arg\max_T \prod_i P(w_i|t_i) \prod_i P(t_i|t_{i-1})$
- Discriminativi: $\hat{T} = \arg\max_T P(T|W) = \arg\max_T \prod_i P(t_i|w_i, t_{i-1})$

3.3 Transformation Based Tagger

L'ultimo algoritmo che vediamo mette insieme metodi statistici e metodi a regole. Anche in questo caso abbiamo dunque bisogno di un corpus da analizzare. Intuitivamente abbiamo due fasi: in una prima fase associamo ad ogni parola il tag più probabile, dopo andiamo a modificare i tag assegnati tramite un insieme di regole. Per essere più precisi l'algoritmo effettua i seguenti passi

1. Assegna il tag più probabile ad ogni parola
2. Controlla ogni possibile trasformazione e seleziona quella che migliora maggiormente il tagging
3. Assegna i nuovi tag al corpus applicando le regole

I passi 2 e 3 vengono poi ripetuti fino a quando viene soddisfatto un certo criterio che potrebbe essere di accuratezza. Il risultato sono una serie di regole di trasformazione.
I vantaggi di questo approccio sono che le regole apprese sono compatte e intelligibili da un umano. Il problema principale di questo algoritmo è che è più lento di quello basato su HMM.

3.4 Ulteriori considerazioni

Esistono un paio di questioni che meritano la nostra attenzione al di là dell'algoritmo scelto per effettuare il PoS tagging. La prima è: come fare a gestire parole sconosciute che compaiono per la prima volta in un testo? Abbiamo diversi metodi per gestire questa situazione

- Assumere che siano nomi, visto che la maggior parte delle volte lo sono

- Assumere che le parole sconosciute seguano la stessa distribuzione del corpus e determinarle di conseguenza

- Usare informazioni morfologiche. Ad esempio nell'esperanto tutte le parole che terminano per "a" sono nomi

- Fare uso di un dizionario esterno che ci dia la risposta

- Assumere che la distribuzione delle parole sia uniforme, poco efficace

Un altro problema da gestire è quello della suddivisione dei dati. Su quale parte del corpus dovremmo effettuare training o test? In questo caso seguiamo le regole del machine learning: circa l'80% per il training, circa il 10% per la configurazione degli iper parametri e infine circa il 10% per il test.

12 / 03 /2018

4 Analisi sintattica

Andando a considerare il livello successivo, ovvero quello sintattico, il nostro scopo diventa quello di ottenere una struttura dati che rappresenti in qualche modo la sintassi della frase che stiamo interpretando.
Un primo aspetto da notare è che il nostro linguaggio può essere ricorsivo: ad esempio la frase "il gatto ha preso il topo che ha rubato il formaggio". Oppure la frase "Elena e Maria odiano il latte e il caffè rispettivamente" costringe il lettore a rileggere la frase per comprenderne pienamente il significato. Con l'analisi sintattica stiamo dunque cercando di cogliere la complessità al livello di forma. Queste intuizioni spinsero Noam Chomsky a proporre di trattare i linguaggi naturali come i linguaggi formali. Per questo motivo oggi vedremo la sintassi e le grammatiche generative, dopo studieremo la gerarchia di Chomsky e infine vedremo altre strutture e grammatiche di particolare interesse.

4.1 Competence: sintassi e grammatiche generative

Il primo errore da evitare è quello di confondere la sintassi con la semantica. La sintassi cerca di studiare il legame tra le parole prendendone in analisi la forma. La semantica invece va a considerarne il significato.

Esempio 1 : la frase "Paolo ama Francesca" e la frase "Francesca ama Paolo" possiedono lo stesso albero sintattico ma non hanno assolutamente lo stesso significato!

Stabilito ciò, possiamo dire che un'altra distinzione da fare subito è quella tra *competence* e *performance*. Questa distinzione venne teorizzata da Chomsky e ci dice che la competence è l'insieme delle caratteristiche intrinseche di una lingua mentre la performance riguarda le capacità di un reale ascoltatore di comprendere il significato dei concetti espressi in una lingua.

Della competence si occupano le grammatiche formali, mentre della performance se ne occupano gli algoritmi di parsing.

I primi studi relativi alle grammatiche sono quelli fatti da Emil Post e Alan Turing. Formalmente una grammatica generativa viene definita come una quadrupla

$$G = (\Sigma, V, S, P) \tag{5}$$

dove Σ è l'insieme dei simboli utilizzabili, V è l'insieme degli stati, $S \in V$ è lo stato iniziale mentre P è l'insieme delle regole di produzione costruite mettendo insieme i simboli e gli stati.
Le grammatiche generative vanno incontro a quanto a quanto teorizzerà Chomsky: studiano il linguaggio naturale come se fosse un linguaggio formale. L'albero di derivazione che possiamo ricavare da una grammatica studia la struttura sintattica della frase.
Esiste anche una particolare categoria di grammatica che viene dette *grammatica libera dal contesto*. Con questo intendiamo dire che è una grammatica formale in cui ogni regola sintattica è espressa sotto forma di derivazione di un simbolo a sinistra a partire da uno o più simboli a destra. Vediamo un esempio

Esempio 2 : una grammatica che abbia la seguente regola di produzione

$aaXbb \rightarrow aacbb$

Non è libera da contesto perché X è all'interno di un contesto.
Invece se la grammatica avesse la regola di produzione

$Y \rightarrow zzz$

in questo caso Y è libera dal contesto, dunque visto che tutte e regole di produzione lo sono, anche la grammatica lo è.

Un meccanismo semplice per capire se una grammatica è libera dal contesto consiste nel guardare il termine sinistro delle regole di produzione. In quel termine non vi devono essere letterali. Una proprietà molto importante delle grammatiche libere dal contesto è che un insieme di letterali può essere sempre e solo generato in un modo. Questo non è invece vero nel caso delle grammatiche ambigue

4.2 Competence: Gerarchia di Chomsky

Nell'ambito dei suoi studi, Chomsky propone una una gerarchia detta *gerarchia di Chomsky* che definisce alcune classi di grammatiche volte a categorizzare le grammatiche generative:

- Grammatiche di tipo 0 o grammatiche illimitate: includono tutte le grammatiche e hanno regole della forma $\alpha A \beta \rightarrow \alpha \gamma \beta$

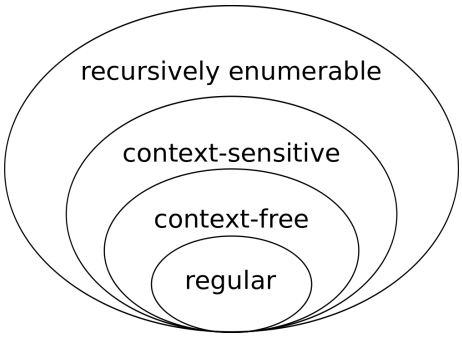


Figura 6: Gerarchia di Chomsky

- Grammatiche di tipo 1 o grammatiche dipendenti dal contesto: grammatiche più generali di quelle libere dal contesto. I linguaggi prodotti da queste grammatiche sono tutti quelli riconosciuti da una macchina di Turing non deterministica
- Grammatiche di tipo 2 o grammatiche libere dal contesto: quelle che abbiamo studiato nelle scorse pagine
- Grammatiche di tipo 3 o grammatiche regolari: generano i linguaggi regolari e che sono quelli riconosciuti dagli automi a stati finiti. Le produzioni sono della forma $A \rightarrow aB$

Mostriamo in Figura 6 la gerarchia appena descritta

Visto che Chomsky dimostra che i linguaggi naturali non possono essere regolari, teorizza che i linguaggi possano essere equivalenti a grammatiche libere dal contesto. Nel 1985 Shieber dimostra che questo non è vero. Ad esempio il tedesco-svizzero non è esprimibile tramite una grammatica libera dal contesto. Questo sembrerebbe portare a pensare che i linguaggi naturali siano prodotti da grammatiche dipendenti dal contesto.

4.3 Competence: altri linguaggi e grammatiche

Parallelamente a Shieber, nel 1985 Joshi propone tramite un nuovo insieme di linguaggi detto insieme dei*mildly context sensitive language* in cui ogni linguaggio possiede le seguenti quattro caratteristiche

- Include le grammatiche libere dal contesto
- Capace di comprendere le dipendenze annidate e incrociate di quei linguaggi come il tedesco, l'olandese...
- Il parsing viene effettuato in tempo polinomiale
- Proprietà di crescita costante: un linguaggi L viene detto in costante crescita se esiste una costante c_0 e un insieme finito di costanti C tale per cui

$\forall w \in L : |w| > c_0 \; \exists w' \in L : |w| = |w'| + c$
per un qualche $c \in C$
Di fatto stiamo dicendo che se abbiamo delle costruzioni lineari nella grammatica non è possibile che la lunghezza delle frasi cresca esponenzialmente

Le grammatiche appena descritte sono leggermente più generali di quelle libere dal contesto e meno generali di quelle dipendenti dal contesto. Possono produrre ad esempio la stringa $a^n b^n c^n$, che le grammatiche libere dal contesto non erano in grado di produrre. Le produzioni di queste grammatiche sono qualcosa della forma $CB \to f(C, B)$ dove f è una funzione lineare.

Negli stessi anni vengono proposte quattro grammatiche più generali di quelle libere dal contesto anche se è stato dimostrato che sono equivalenti fra loro. Noi ne vediamo velocemente due: la prima è la *Tree Adjoining Grammar* mentre la seconda è la *Combinatory Categorial Grammar*. Queste grammatiche non sono più quelle con le semplici produzioni. Perché a questo punto vorremmo utilizzare delle strutture elementari che siano in grado di esprimere le strutture elementari. La struttura elementare diventa quindi non più la stringa che avevamo nelle produzioni, bensì degli alberi multilivello.

- Tree Adjoining Grammar: proposta da Joshi, non abbiamo più la stringa ma gli alberi e su di essi facciamo due operazioni: la sostituzione che prende la foglia di un albero e la sostituisce con qualcos'altro. La seconda operazione è l'adjoining con cui andiamo a sostituire un nodo interno con un sottoalbero.
- Combinatory Categorial Grammar: partiamo da una serie di simboli che vengono combinati. Quindi partiamo dalle parole. L'approccio dunque è bottom-up visto che partiamo dagli oggetti finali della nostra grammatica.

13 / 03 /2018

4.4 Performance: anatomia di un parser

A questo punto, abbiamo visto quali sono gli elementi che possiamo usare per costruire un parser sintattico, possiamo concentrarci sull'individuare un algoritmo che vada ad individuare la struttura gerarchica delle sotto sequenze mediante un albero. Per farlo, studiamo quali sono gli elementi da analizzare in un parser. Dunque con "anatomia di un parser" intendiamo tener di conto della competence e di altri elementi:

1. Grammatica: la prima cosa da capire è con quale grammatica abbiamo a che fare. Potrebbe essere libera dal contesto, oppure Tree Adjoining...
   Spesso le grammatiche reali non sono libere dal contesto ma noi tendiamo a semplificare e a considerarle tali
2. Algoritmo: per determinare con maggiore precisione l'algoritmo dobbiamo determinare due aspetti

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$

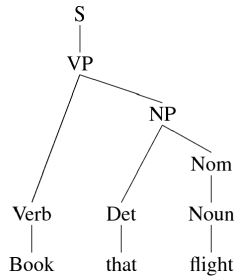


Figura 7: A sinistra la grammatica utilizzata, a destra l'albero di derivazione

- Strategia di ricerca: la strategia ci indica da dove partire ad analizzare la frase. Può essere top-down, bottom-up, oppure può anche partire dal centro della frase
- Organizzazione della memoria: alcune tecniche ci dicono come gestire la memoria. Ad esempio potremmo usare il backtracking o una tecnica di programmazione dinamica

3. Oracolo: l'oracolo è uno strumento che ci dice in quale "direzione" andare. Ad esempio se possiamo applicare più regole di produzione della nostra grammatica, quale sarebbe meglio utilizzare? Gli oracoli possono essere probabilistici, basati su regole...

Da ora in poi faremo l'approssimazione che tutte le grammatiche siano libere dal contesto.

Esempio 1 : supponiamo di voler costruire l'albero relativo alla frase "book that flight". Per farlo utilizziamo la grammatica mostrata nella parte sinistra di Figura 7

Come dicevamo supponiamo di avere a che fare con una grammatica libera dal contesto. Come strategia di ricerca decidiamo di utilizzare un approccio top-down e per gestire la memoria utilizziamo l'algoritmo BFS, quindi applichiamo le regole nell'ordine in cui le troviamo. Notiamo quindi che in questo caso l'oracolo non esiste, ovvero non abbiamo alcuna indicazione intelligente su quali regole applicare per prime.
In base a queste scelte, l'albero di derivazione che abbiamo è quello mostrato nella parte destra di Figura 7.

Possiamo notare che, purtroppo, per raggiungere una soluzione nell'esempio appena visto dobbiamo esplorare molti cammini "inutili". Questo approccio non va bene quando abbiamo a che fare con frasi vagamente più complicate di quella vista. In termini di memoria occupiamo troppo spazio e questo non va bene.
La prima soluzione che potrebbe venire in mente è di andare ad utilizzare un algoritmo in profondità. In questo caso il problema della memoria è certamente

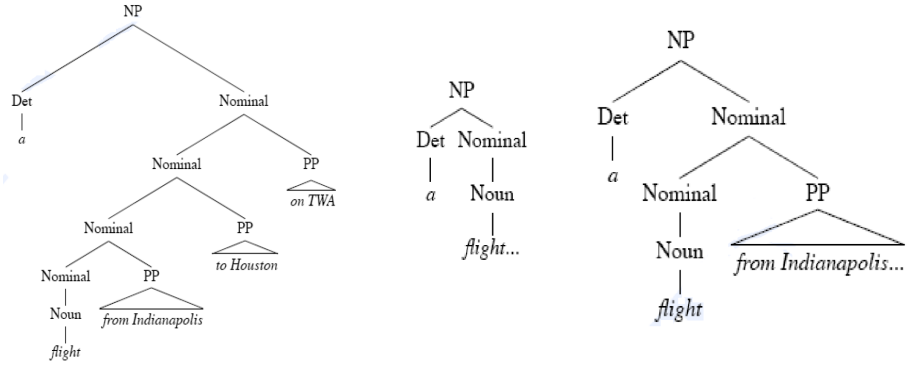


Figura 8: A sinistra l'albero completo, al centro il primo passo dell'algoritmo, a destra il secondo passo dell'algoritmo

risolto però introduciamo un nuovo problema, ovvero quello della *left recursion*. Se andiamo ad espandere sempre in profondità esiste il rischio di ritrovarsi in un loop perché magari una serie di regole di produzione danno luogo ad un ciclo. Ad esempio la regola $NP \rightarrow NP\ PP$ genera proprio questo problema. In questo caso accorgersi del ciclo è molto facile, ma nella realtà dove abbiamo a che fare con grammatiche a moltissime regole non possiamo controllare facilmente che questa situazione non si verifichi.
Un ulteriore problema dell'approccio in profondità è che per quanto sia vicino alla vera soluzione del problema, se finisce in un "vicolo cieco" butta via tutto e riparte da capo. Vediamolo con l'esempio che segue

Esempio 2 : supponiamo di voler costruire l'albero per la frase "a flight from Indianapolis to Houston on TWA". Mostriamo nella parte sinistra di Figura 8 l'albero che possiamo costruire utilizzando un approccio in profondità

Al centro dell'immagine mostriamo invece il primo "vicolo cieco" in cui l'algoritmo in profondità si ferma. Notiamo che quando l'algoritmo si ferma per la prima volta ricomincia da capo e trova un qualcosa di molto simile a quanto trovato prima, ma con l'aggiunta di una parte ulteriore, che mostriamo nella parte destra di Figura 8. Siamo dunque in una situazione di sotto problemi condivisi.

Il problema dell'algoritmo appena descritto è che ogni volta butta via una parte della soluzione ottimale per ricominciare da capo. Questo fatto mette in evidenza che in questo contesto le tecniche di programmazione dinamica potrebbero aiutarci, infatti fra poco vedremo un algoritmo di programmazione dinamica per generare l'albero di derivazione.
Notiamo che in questo contesto le tecniche di programmazione dinamica sono praticamente d'obbligo perché ad esempio per una frase di 15 parole possiamo creare 10^6 diversi alberi di parsing. Addirittura in certi casi neanche le tecniche di programmazione dinamica bastano per gestire problemi di dimensione maggiore.

La formazione di così tanti alberi è dovuta al fatto che abbiamo un alto livello di ambiguità nel linguaggio che deriva fondamentalmente da due cause:

- Ambiguità di attachment: si verifica quando un certo costituente può essere "attaccato" in più di un punto del nostro albero di parsing. Un esempio potrebbe essere il seguente: "One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know." Groucho Marx, Animal Crackers, 1930

- Ambiguità di coordinazione: si verifica quando diversi insiemi di frasi possono essere uniti dalla congiunzione "e". Ad esempio nella frase "televisioni e radio nazionali" cosa sono nazionali? le radio o le televisioni?

4.5 Performance: Cocke Kasami Younger(CKY)

L'algoritmo usato in questo caso assume di lavorare su grammatiche libere dal contesto, utilizzando un approccio bottom-up con gestione della memoria tramite programmazione dinamica. Per ora non faremo uso di un oracolo, ma useremo le regole nell'ordine in cui compaiono.
Un'assunzione dell'algoritmo CKY è che le regole di produzione siano nella forma normale di Chomsky, ovvero che siano del tipo:

$A \to BC$
$A \to d$

Ovvero abbiamo solo delle regole "da completare" e delle regole lessicali. Ad ogni modo l'assunzione non lede la generalità dell'algoritmo visto che ogni grammatica con l'opportuna aggiunta di regole può essere convertita in forma normale di Chomsky.
A livello pratico l'algoritmo CKY si occupa di riempire una tabella, che ha la forma di un albero, con dei termini opportuni. Mostriamo un esempio di tale tabella in Figura 9

L'intuizione che sta dietro all'algoritmo è la seguente: se abbiamo una regola di produzione della forma $A \to BC$ allora se A "domina" dalla parola i alla parola j della nostra frase, dovrà esistere un punto k tale per cui B "domina" dal punto i al punto k e tale per cui C "domina" dal punto k al punto j. L'idea è quindi di costruire dei cicli su questi indici e andare a salvare A in posizione $[i, j]$ della tabella, B in posizione $[i, k]$ e C in posizione $[k, j]$.
La complessità dell'algoritmo è $O(n^3)$. Ne mostriamo il codice in Figura 10

Notiamo come la complessità $O(n^3)$ dell'algoritmo derivi dai tre for annidati. Intuitivamente la prima riga del primo for si occupa di riempire il livello più basso della tabella. Nei due for interni andiamo invece a riempire i livelli superiori.

16 / 03 /2018

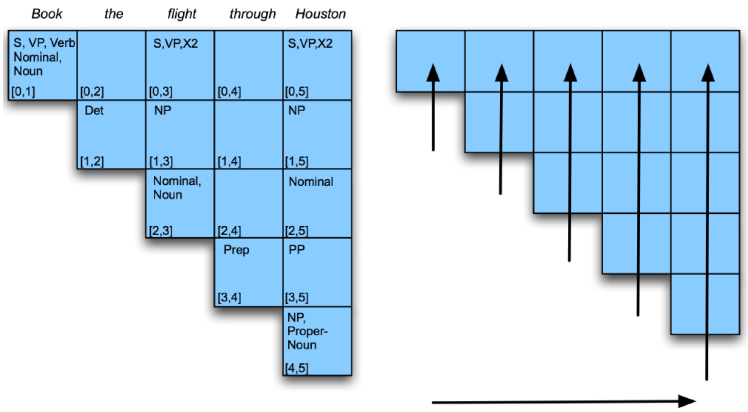


Figura 9: Tabella che viene progressivamente riempita dall'algoritmo CKY

function CKY-PARSE(*words*, *grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

Figura 10: Algoritmo di Cocke Younger Kasami

Cerchiamo di comprendere come funzioni l'algoritmo con l'esempio che segue

Esempio 1 : supponiamo di voler verificare se la frase "Paolo ama Francesca dolcemente" sia una frase valida della lingua italiana e quali siano le sue componenti. Innanzitutto sappiamo che il vettore words sarà:
$words = [Paolo, ama, Francesca, dolcemente]$.
Supponiamo di utilizzare la seguente grammatica:

$S \rightarrow NP\ VP$ $\qquad$ $VP \rightarrow VPADV$
$VP \rightarrow V\ NP$ $\qquad$ $NP \rightarrow Paolo \mid Francesca$
$V \rightarrow ama$ $\qquad$ $ADV \rightarrow dolcemente$

Mostriamo il risultato finale dell'esecuzione dell'algoritmo in Figura 11

Adesso cerchiamo di capire come abbiamo riempito la tabella. Di fatto

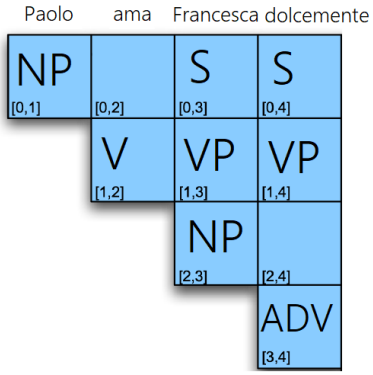


Figura 11: Risultato dell'esecuzione di CKY sull'esempio 1

la frase viene considerata valida se nella cella più in alto a destra(ovvero la radice dell'albero) troviamo S che indica che è una frase di senso compiuto. Per comprendere come abbia operato l'algoritmo scomponiamo l'esecuzione sui tre cicli for andando ad analizzarne gli indici

Primo for: j=1 Visto che $words[1] = Paolo$, la prima cosa da controllare in base all'algoritmo mostrato in Figura 10 è se esista una regola di produzione A della nostra grammatica tale che $A \rightarrow Paolo$, ovvero se $Paolo$ è termine destro della regola di produzione. Questa regola esiste ed è $NP \rightarrow Paolo$. Dunque andiamo a mettere in $[j-1, j] = [0, 1]$ il valore NP

Secondo for: i=-1 : in questo caso il for interno si ferma subito perché la condizione è già violata

Primo for: j=2 : visto che $words[2] = ama$ cerchiamo una regola tale che abbia a destra il termine ama. Questa regola esiste e il suo termine sinistro è V. Dunque andiamo a scrivere V in $[j-1, j] = [1, 2]$

Secondo for: i=0 :

Terzo for: k=1 : visto che $[i, k] = [0, 1] = NP$ e $[k, j] = [1, 2] = V$, andiamo a controllare se esiste una regola tale che il suo termine destro sia $NP\ V$. Questa regola non esiste e dunque la casella $[i, j] = [0, 2]$ resta vuota

Primo for: j=3 : controlliamo se esista una regola tale che il termine destro sia $Francesca$. Questa regola esiste, dunque andiamo a scrivere in $[2, 3]$ il termine NP

Secondo for: i=1 :

Terzo for: k=2 : visto che $[i, k] = [1, 2] = V$ e $[k, j] = [2, 3] = NP$, andiamo a controllare se esista una regola tale che il termine destro sia $V\ NP$. Questa regola esiste e dunque poniamo il suo termine sinistro VP in $[i, j] = [1, 3]$.

Secondo for: i=0 :

Terzo for: k=1 : come visto in precedenza, ci chiediamo se esista una regola tale che il suo termine destro sia $NP\ VP$. Questa regola esiste e dunque mettiamo S in $[0, 3]$.
Cosa ci sta dicendo il fatto che S sta in $[0, 3]$? Ci sta dicendo che la frase "Paolo ama Francesca" è una frase valida per l'italiano

Terzo for: k=2 : in questo caso non possiamo fare niente perché in $[i, k] = [0, 2]$ la cella è vuota.

Primo for: j=4 : controlliamo se esiste una regola il cui termine destro sia "dolcemente". Questa regola esiste e andiamo a porre in $[3, 4]$ il termine sinistro di tale regola, ovvero ADV

Secondo for: i=2 :

Terzo for: k=3 : esiste una regola tale che il termine destro sia $NP\ ADV$? No, quindi in $[2, 4]$ lasciamo lo spazio vuoto. Notiamo che questo intuitivamente ha molto senso perché di solito gli avverbi non modificano i nomi, bensì i verbi

Secondo for: i=1 :

Terzo for: k=2 : anche in questo caso non possiamo far niente perché in $[k, j] = [2, 4]$ non abbiamo niente

Terzo for: k=3 : esiste una regola il cui termine destro sia $VP\ ADV$? Sì e dunque scrivo VP in $[1, 4]$.

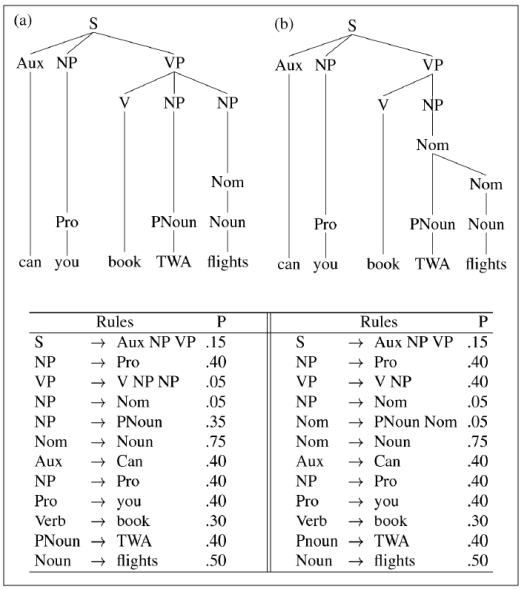
Secondo for: i=0 :

Terzo for: k=1 : esiste una regola il cui termine destro sia $NP\ VP$? Sì, dunque vado a mettere S in $[0, 4]$.

Potremmo proseguire ma l'ultimo passo è quello più importante perché mettiamo nella radice dell'albero il termine S. Questo ci mostra che la frase "Paolo ama Francesca dolcemente" è una frase valida in quanto possiamo costruire un albero per tale frase. Notiamo che se la frase fosse stata ambigua, avremmo potuto avere più di un S in posizione $[0, 4]$. Come possiamo individuare il giusto significato tra i vari possibili? A questo livello semplicemente non possiamo. Questo perché siamo ad un livello puramente sintattico e non semantico.

4.6 Performance: parsing probabilistico

Come avevamo detto, adesso possiamo provare a dare una versione di CKY probabilistica di modo che si possa dire con una certa probabilità se un certo albero di derivazione sia giusto. La differenza con la versione di CKY fornita nella sezione precedente è che semplicemente cambiamo l'oracolo. Se prima non lo avevamo adesso ne abbiamo uno probabilistico.
L'idea è che se associamo ad ogni regola di produzione una certa probabilità,



Rules			P
S	→	Aux NP VP	.15
NP	→	Pro	.40
VP	→	V NP NP	.05
NP	→	Nom	.05
NP	→	PNoun	.35
Nom	→	Noun	.75
Aux	→	Can	.40
NP	→	Pro	.40
Pro	→	you	.40
Verb	→	book	.30
PNoun	→	TWA	.40
Noun	→	flights	.50

Rules			P
S	→	Aux NP VP	.15
NP	→	Pro	.40
VP	→	V NP	.40
NP	→	Nom	.05
Nom	→	PNoun Nom	.05
Nom	→	Noun	.75
Aux	→	Can	.40
NP	→	Pro	.40
Pro	→	you	.40
Verb	→	book	.30
Pnoun	→	TWA	.40
Noun	→	flights	.50

Figura 12: Probabilità di ottenere i due alberi

possiamo poi associare ad ogni albero un certa probabilità data dal prodotto delle probabilità delle varie regole. Ovvero:

$$P(T,S) = \prod_{node \in T} P(rule(n)) \tag{6}$$

dove S è la frase e T è l'albero prodotto.
In Figura 12 mostriamo a titolo d'esempio le probabilità di ottenere due alberi.

Notiamo che a sinistra abbiamo i due alberi e subito sotto le regole di produzione a cui abbiamo associato una certa probabilità. Notiamo inoltre che la somma delle probabilità delle regole che hanno stesso termine sinistro è pari a 1. Sennò non avremmo una probabilità.
La domanda più importante a questo punto diventa: come possiamo stabilire le probabilità per le varie regole di produzione?
Solitamente andiamo ad utilizzare le frequenze delle regole all'interno di un corpus. Tali corpus vengono detti *tree bank*(TB). Ad oggi il più importante è il "Penn TB". Se prima i TB venivano scritti a mano, ad oggi si usano dei meccanismi semi automatici basati sulle tecniche di machine learning. In una prima fase usiamo gli algoritmi di learning e in una seconda rimuoviamo gli errori a mano.
A livello pratico, per calcolare la probabilità di una regola di produzione $\alpha \to \beta$ andiamo a determinare la quantità $P(\alpha \to \beta|\alpha)$. Per farlo prendiamo ogni albero del TB e calcoliamo:

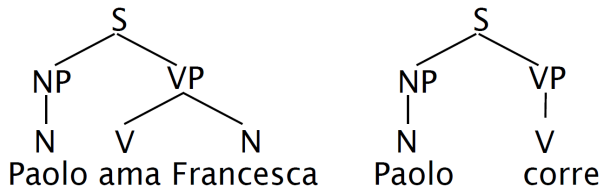


Figura 13: Tree bank formato da due soli alberi

$$P(\alpha \to \beta|\alpha) = \frac{C(\alpha \to \beta)}{C(\alpha)}$$

Dove $C(x)$ conta semplicemente le occorrenze di x.

Esempio 1 : consideriamo un semplice TB formato da due soli alberi. Quelli mostrati in Figura 13

In questo caso allora le probabilità sono:

$P(S \to NP\ VP) = 1$
$P(NP \to N) = 1$
$P(VP \to V\ N) = 0.5$ $\quad$ $P(VP \to V) = 0.5$
$P(N \to Paolo) = 0.67$ $\quad$ $P(N \to Francesca) = 0.33$
$P(V \to corre) = 0.5$ $\quad$ $P(V \to ama) = 0.5$

In base a quanto detto, possiamo andare a mostrare l'algoritmo CKY probabilistico.

function PROBABILISTIC-CKY(*words*,*grammar*) **returns** most probable parse and its probability

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  for all $\{ A \mid A \to words[j] \in grammar \}$
    $table[j-1, j, A] \leftarrow P(A \to words[j])$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{ A \mid A \to BC \in grammar,$
        and $table[i,k,B] > 0$ **and** $table[k,j,C] > 0 \}$
        if $(table[i,j,A] < P(A \to BC) \times table[i,k,B] \times table[k,j,C])$ **then**
          $table[i,j,A] \leftarrow P(A \to BC) \times table[i,k,B] \times table[k,j,C]$
          $back[i,j,A] \leftarrow \{k, B, C\}$
  return BUILD_TREE($back$[1, LENGTH(*words*), S]), $table$[1, LENGTH(*words*), S]

Figura 14: Algoritmo CKY probabilistico

In questo caso la complessità dell'algoritmo aumenta a $O(n^5)$ e nelle celle della tabella andiamo a mettere la probabilità associata.
L'ultimo problema che resta da risolvere è: come facciamo a valutare se un parser si "comporta bene"? Per farlo dobbiamo dare due misure, dette *preci-*

sione(precision) ed *esaurienza*(recall). Queste due misure ci dicono due cose diverse:

- Precision: prendo tutti i sottoalberi che ho individuato e confrontandoli con il TB valuto quale è la percentuale di sottoalberi corretti
- Recall: questa seconda misura vuole invece dirci quanti dei sottoalberi giusti ho individuato. Infatti potrei averne omesso qualcuno e questo non andrebbe bene

Ad ogni modo, in certi casi, a causa dell'ambiguità dobbiamo allegare alle foglie del nostro albero delle informazioni aggiuntive. Le grammatiche che fanno uso di informazioni aggiuntive vengono dette PCFG e fino a qualche anno fa rappresentavano lo stato dell'arte del parsing. L'idea è che nelle frasi esista una parola più importante delle altre. Ad esempio nella frase "Paolo ama Francesca" è il verbo ama a dare senso all'intera frase. O nella frase "il cane giallo" è il nome cane a dare un'immagine a chi legge, più che l'aggettivo giallo. Dunque l'intuizione è di individuare il termine più importante e riportare questa informazione ai livelli superiori. Ovviamente anche questa operazione viene fatta tramite probabilità. Questo approccio ha però un grave problema, vi sono troppe regole.

19 / 03 /2018

4.7 Altri tipi di parsing

Fino ad ora abbiamo supposto che le grammatiche fossero libere dal contesto. Questa ovviamente è un'approssimazione che funziona per tante strutture. Cosa succede se facciamo un ulteriore approssimazione dicendo che la grammatica utilizzata è regolare? In realtà usiamo diverse grammatiche in serie.
In questo contesto abbiamo il *chunck parsing*, ovvero un pezzo. Esiste una differenza forte tra chunck e sintagma. I sintagmi supportano la ricorsione, i chunck no! Dunque andiamo ad eliminare la ricorsione andando a costruire delle regole di produzione senza alcuna ricorsione.
Con un primo livello di grammatica regolare riconosco i vari chunk e dopo con un'altra grammatica li rimetto insieme per ricavarne delle informazioni.
Quale è il vantaggio di fare queste approssimazioni? Che la complessità temporale è lineare.

Un altro tipo di parsing è il *chunk parsing probabilistico*. Questo viene fatto andando a usare delle tecniche dette di IOB tagging, ovvero andando ad usare delle etichette *begin, inside, outside*

4.8 Parsing a dipendenze

Negli ultimi anni c'è stato un crescente interesse per quel che riguarda le grammatiche a dipendenze. Ovvero vogliamo andare ad individuare il soggetto, il complemento oggetto...

Nel 1959 Tesniere cerca di definire le dipendenze tra i vocaboli dicendo che tre i vicini esistono dei legami che vengono percepiti dalla mente. Queste dipendenze nel loro complesso danno senso alla frase. Un fatto importante da ricordare è che il rapporto tra i vari vocaboli non è paritario. Alcuni termini sono più importanti di altri.
Come riconoscere le varie parti, come ad esempio la testa(la parola più importante) e le sue dipendenze? Esistono dei test morfologici per capirlo basati su delle regole. Questi test sono stati sviluppati abbastanza di recente grazie all'approccio algoritmico di Hudson. Ad ogni modo esistono molti tipi di test: morfologico, sintattico... Esistono comunque dei fenomeni sintattici più complicati, ad esempio se abbiamo a che fare con verbi ausiliari, oppure causati dalla punteggiatura.
Prima di proseguire potremmo chiederci: come mai siamo così interessati al parsing a dipendenze? Perché ci forniscono molte informazioni in modo compatto e perché ad esempio ci permette di rispondere alla domanda "chi ha fatto cosa". E questo ci permette di avvicinarci alla semantica.

Quali approcci abbiamo per gestire questo problema? Il primo che vediamo è quello di *dynamic programming*. Questo primo metodo è simile a CKY probabilistico e bisogna dire che Eisner nel 1996 ha trovato un modo per passare da una complessità $O(n^5)$ a $O(n^3)$.
Un secondo approccio genera un minimum spanning tree tra i vari nodi che sono le parole. Le distanze tra le parole che ci permettono di individuare lo spanning tree vengono ricavati dal corpus, che ad esempio ci suggerirà che "Paolo" e "ama" stanno più vicini di "Paolo" e "Francesca".
Costruire lo spanning tree non è facilissimo ma sappiamo che esistono degli algoritmi standard per fare questa operazione.
Il terzo approccio prevede un parsing con una grammatica a costituenti e converto in un formato a dipendenze attraverso delle "tabelle di percolazioni"
Il quarto approccio viene detto *deterministic parsing* basato su scelte greedy guidati da classificatori di machine learning. Ad ogni modo ci concentriamo tra poco su questo modello.
L'ultimo approccio, detto *constraint satisfaction* va a eliminare le dipendenze che non soddisfano certi vincoli.

Come dicevamo ci concentriamo sul parser deterministico detto *E-MALT*. È un parser bottom up depth first proprio perché non deve fare backtracking visto che non sbaglia. L'oracolo utilizzato è probabilistico.
Nel caso del MALT parser la grammatica è "nascosta" nelle preferenze del parser. Ovvero usiamo il tree bank per apprendere un automa probabilistico e dunque la grammatica è implicita nei valori associati all'arco dell'automa. Dunque avremo ad esempio un arco tra "Paolo" e "ama" che ne indica la probabilità. L'altro punto determinante è che non c'è né backtracking né programmazione dinamica perché gli alberi che individua sono costruiti grazie all'oracolo probabilistico e dunque l'output dell'algoritmo è quello ritenuto più probabile. Negli approcci visti finora invece c'erano degli output oggettivamente non corretti.

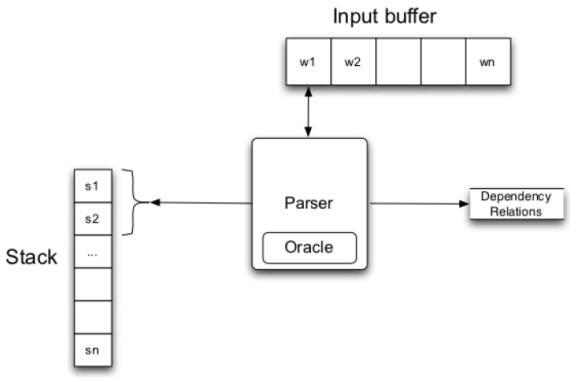


Figura 15: Struttura di un parser malt

Qui invece operiamo in termini di probabilità: dunque la frase "ama ama ama" non verrà considerata sbagliata, ma solo poco probabile!
Quale è la struttura di un parser malt? La mostriamo in Figura 15

L'oracolo è "nascosto" al centro. Abbiamo un buffer in input, uno stack legato all'automa su cui vado a scrivere. Infine abbiamo una struttura in output dove inserisco le relazioni. Ad oggi tutti i parser a dipendenze hanno questa struttura.
La versione che vediamo noi fa un'assunzione semplificativa. Ovvero che tutti gli alberi prodotti in output siano *proiettivi*. In tali alberi diciamo che se una certa parola i ne domina un'altra j, allora tutte le parole incluse nell'intervallo $[i, j]$ sono dominate da i.
Questa assunzione viene fatta perché le strutture non proiettive sono sempre molto più difficili da gestire. Dobbiamo comunque dire che esistono alcune lingue (olandese) che sono inerentemente non proiettive. In realtà anche alcuni elementi in italiano(il 2%-3% degli elementi lo sono). Per trattare le strutture non proiettive dovremmo aggiungere altre componenti al nostro automa e anche dal punto di vista probabilistico diventa più complicato.
Da un punto di vista degli stati, ogni stato è composto da tre cose

- Uno stack per le parole parzialmente analizzate
- Una lista contenente le rimanenti parole da analizzare
- Un insieme contenente le dipendenze create fino a quel punto dall'analisi

Abbiamo tre possibili transizioni(o comandi):
Lo shift prende la prossima parola e la inserisce in cima allo stack(push).
Il left crea una dipendenza (a,b) tra la prossima parola "a" della lista e la parola "b" sulla cima dello stack. Inoltre viene effettuata un'operazione di pop sullo stack.
L'operazione di right fa invece tre cose: crea una dipendenza (b,a) tra la prossima parola "a" della lista e la parola "b" sulla cima dello stack, rimuove "a"

dalla lista ed effettua il pop sullo stack andando a mettere la parola prelevata all'inizio della lista.
Notiamo che se abbiamo tre mosse possibili, non avremo determinismo! La scelta dell'azione da eseguiredeve essere fatta da un oracolo!
Vediamo come opera il parser tramite un esempio.

Esempio 1 : supponiamo che lo stato iniziale sia:

$$[[root], [I\ booked\ a\ morning\ flight], ()]$$

e che lo stato finale sia:

$$[[root], [], ((booked, I)(booked, flight)(flight, a)(flight, morning))]$$

Notiamo come nello stato finale abbiamo creato tutte le dipendenze.
Vediamo l'evoluzione della lista e dello stack in Tabella 1

Azione	Stato
/	$\{[root], [Ibookedamorningflight], []\}$
Shift	$\{[root, I], [bookedamorningflight], []\}$
Left	$\{[root], [bookedamorningflight], [(booked, I)]\}$
Shift	$\{[root, booked], [amorningflight], [(booked, I)]\}$
Shift	$\{[root, booked, a], [morningflight], [(booked, I)]\}$
Shift	$\{[root, booked, a, morning], [flight], [(booked, I)]\}$
Left	$\{[root, booked, a], [flight], [(booked, I), (flight, morning)]\}$
Left	$\{[root, booked], [flight], [(booked, I), (flight, morning), (flight, a)]\}$
Right	$\{[root], [booked], [(booked, I), (flight, morning), (flight, a), (booked, flight)]\}$
Right	$\{[], [root], [(booked, I), (flight, morning), (flight, a),$ $(booked, flight), (root, booked)]\}$
Shift	$\{[root], [], [(booked, I), (flight, morning), (flight, a),$ $(booked, flight), (root, booked)]\}$

Tabella 1: Esecuzione del parser E-Malt sull'esempio 1

Ovviamente notiamo che le azioni eseguite dal parser sono scelte dall'oracolo e questo è ovviamente determinante! L'oracolo può sbagliare. Anche l'algoritmo di parsing migliore del mondo ha un oracolo che su trenta parole una la sbaglia. Ad ogni modo è possibile rimediare a questi errori. Notiamo che comunque, anche se l'albero ottenuto con il parser malt non è quello corretto, viene comunque considerato "legale". Come dicevamo è infatti per questo motivo che non abbiamo il backtracking.

Abbiamo comunque due problemi da risolvere: gli operatori e le relazioni dovrebbero essere etichettate. In realtà questo non è un vero problema perché basta specificare per ogni azione(left, right, shift) una azione corrispondente alla relazione. Dunque avremo "ShiftSubj", "ShiftObj", "LeftSubj", "LeftObj"...
Ovviamente al crescere delle relazioni possibili cresce la difficoltà del problema

di learning da risolvere.
Il secondo problema è: quale operatore utilizzare ad ogni passo? Solitamente andiamo ad addestrare un classificatore di machine learning che scelga l'operatore. Notiamo però che il numero di stati è potenzialmente infinito visto che le azioni possiamo potenzialmente eseguirle infinite volte. Come apprendere riguardo un automa che ha infiniti stati? Per questo motivo non vengono usati gli stati ma le feature definite sullo stato. Ad esempio: lo stack, la lista, l'albero parziale, il PoS che è in posizione top dello stack...

20 / 03 /2018

Ovviamente l'uso di tecniche di learning mette in evidenza tre necessità importanti:

- Capire quali sono le feature linguisticamente significative: ad esempio quelle che avevamo indicato poco sopra
- Avere un corpus di riferimento: ad oggi esiste un dataset formato da 70 tree bank per oltre 50 lingue diverse a dipendenze
- Algoritmo di training: per ricostruire la sequenza di azioni eseguite (shift/left/right) partiamo dall'albero delle dipendenze(che ci viene fornito) e procediamo ad applicare le azioni affinché si verifichi il match tra ciò che l'azione fa e la dipendenza che troviamo nell'albero.

Da qui in poi è un problema di machine learning risolvibile mediante una funzione di scoring che stabilisce quale sia la migliore azione da applicare. Dunque l'apprendimento è locale ma utilizza delle feature determinate globalmente.

Un approccio diverso è quello basato su vincoli sviluppato presso l'Università di Torino. In questo caso abbiamo una grammatica a dipendenze, come strategia di ricerca usiamo un procedimento bottom-up depth first senza backtracking. L'oracolo è invece basato su regole.
Il parser opera come segue:

1. Chunking: tramite delle regole a priorità stabiliamo quali siano i legami tra vocaboli
2. Verb-SubCat: mettiamo insieme i vari chunk tramite delle sotto dipendenze.

Chiaramente questo sistema tende ad adattarsi al dominio su cui lo utilizziamo

5 Analisi semantica

Quando parliamo di interpretazione semantica ci riferiamo probabilmente a uno degli argomenti più complicati della linguistica. Questo perché la semantica è

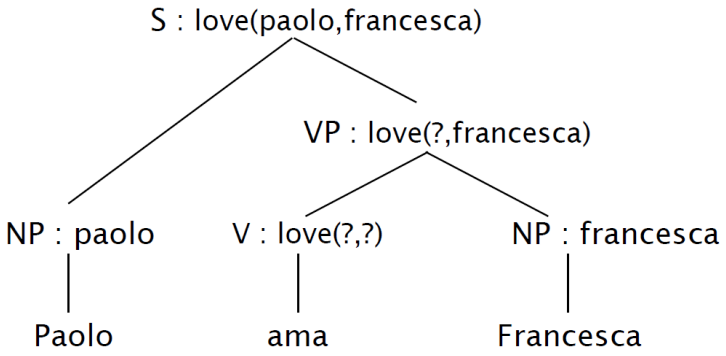


Figura 16: Albero dell'esempio 1 ottenuto mediante la logica del primo ordine

più difficile da incasellare in un sistema di regole.
I primi studi a riguardo sono quelli di Winograd nel '72. All'inizio degli anni '80 gli studi di Pereira e Warren portano allo sviluppo di CHAT-80, software capace di rispondere a domande più o meno complicate.

5.1 Logica del primo ordine

La domanda da porsi in ambito semantico è: come possiamo rappresentare il significato? Esistono molte risposte, ad esempio i database, oppure le logiche descrittive o modali. Alcuni studi hanno preso in analisi le logiche del primo ordine perché abbastanza espressive. Purtroppo le logiche del primo ordine danno una visione statica del mondo e dunque, come vedremo fra poco, la soluzione individuata si trova nel λ-calcolo.
Dando per scontati gli elementi basilari della logica del primo ordine, come dovremmo strutturare un generico algoritmo di semantica computazionale?

1. Effettuare il parsing della frase per ottenere un albero sintattico a costituenti(fatto durante l'analisi sintattica)
2. Cercare la semantica di ogni parola nel lessico
3. Costruire con un approccio bottom-up la semantica per ogni sintagma secondo il principio di Frege: "Il significato del tutto è determinato dal significato delle parti e dalla maniera in cui sono combinate"

Proviamo ad eseguire l'analisi semantica utilizzando la logica del primo ordine

Esempio 1 : supponiamo di avere la solita frase "Paolo ama Francesca". Andando ad applicare gli operatori della logica del primo ordine, otteniamo quanto mostrato in Figura 16

Purtroppo nel caso dell'esempio precedente rimangono irrisolte alcune questioni: come costruire il significato delle VP? "love(?,francesca)" oppure "love(francesca,?)"? Come possiamo specificare come combinare le parti? Come

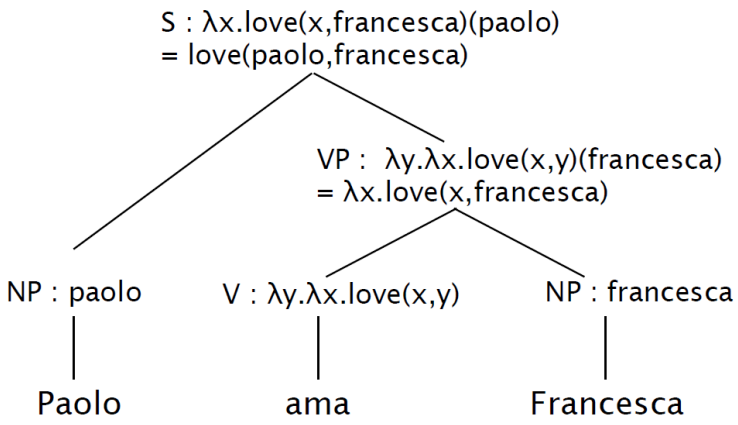


Figura 17: Albero ottenuto utilizzando le tecniche del λ-calcolo

rappresentare parti di formule? Ad esempio anche il predicato "love(?,?)" esula dalla logica del primo ordine stretta.

5.2 λ-calcolo

La vera svolta arriva mediante l'uso del λ-calcolo. Viene utilizzato un nuovo operatore λ per legare le variabili libere mediante l'operazione di *beta-reduction*. L'operazione di beta reduction viene effettuata mediante tre passaggi distinti. Partendo da un qualcosa del tipo:
$(\lambda\ x.love(x, mary))(john)$

1. Eliminiamo il λ e otteniamo $(love(x, mary))(john)$
2. Rimuoviamo l'argomento e otteniamo $love(x, mary)$
3. Rimpiazziamo le occorrenze della variabile legata dal λ con l'argomento in tutta la formula ottenendo $love(john, mary)$

Allora possiamo provare a riprendere l'esempio di prima e vedere come risolverlo correttamente

Esempio 1 : utilizzando il λ-calcolo il risultato che otteniamo è quello mostrato in Figura 17

Notiamo che in questo caso non abbiamo più "love(?,?)", bensì degli oggetti ben definiti grazie ai λ. Questo ci deriva dal fatto che utilizziamo una logica superiore a quella del primo ordine. Un aspetto importante da considerare è che anche se i livelli intermedi seguono le regole del λ-calcolo, nella radice troveremo sempre un elemento della logica del primo ordine, che è anche quello che ci interessa di più perché illustra il significato della frase.

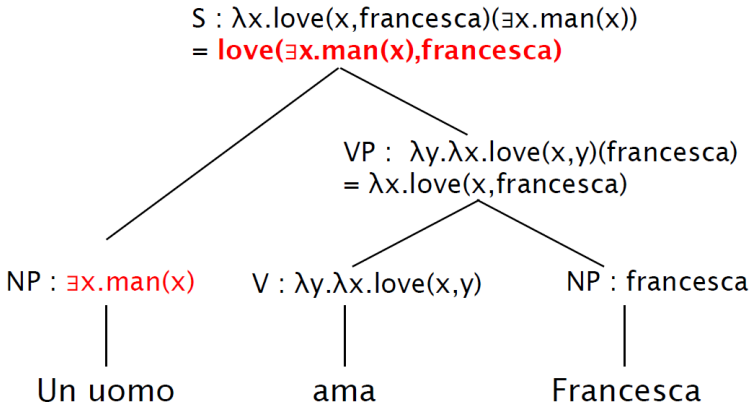


Figura 18: Semantica errata dell'articolo "un"

Un ulteriore aspetto da considerare è che la semantica che studiamo non tiene di conto dei tempi e di altri aspetti linguistici. In questo senso la semantica che studiamo(che viene detta di Montague) si concentra sulla struttura *predicato-argomento*

23 / 03 /2018

Uno dei problemi della semantica utilizzata assieme al λ-calcolo è che non è modulare. Ovvero quando vogliamo estenderlo con la semantica di nuove parti sintattiche, il procedimento non è semplice. Di fatto la semantica computazionale non scala bene. Vediamo cosa questo voglia dire cercando di aggiungere gli articoli.

Esempio 2 : supponiamo di voler studiare la semantica della frase "un uomo ama Francesca". Potremmo farlo come mostrato in Figura 18

Notiamo che purtroppo la frase ottenuta alla fine, nella radice dell'albero, non è una formula della logica del primo ordine in quanto ci aspetteremmo che i quantificatori stiano al di fuori del predicato.
Notiamo inoltre che $\exists x.man(x)$ significa "c'è un uomo" più che "un uomo".

Il problema verificatosi nell'esempio appena visto è dovuto ad una errata definizione dell'articolo. Per questo motivo introdurre nuove componenti sintattiche non è facile.
Il procedimento più semplice è quello di procedere al contrario inserendo nella radice quello che ci aspettiamo di ottenere e andando in seguito a scendere nell'albero.
Allora, riferendoci sempre alla frase "un uomo ama Francesca" cercheremo di partire da:

$\exists z(man(z) \wedge love(z, Francesca))$

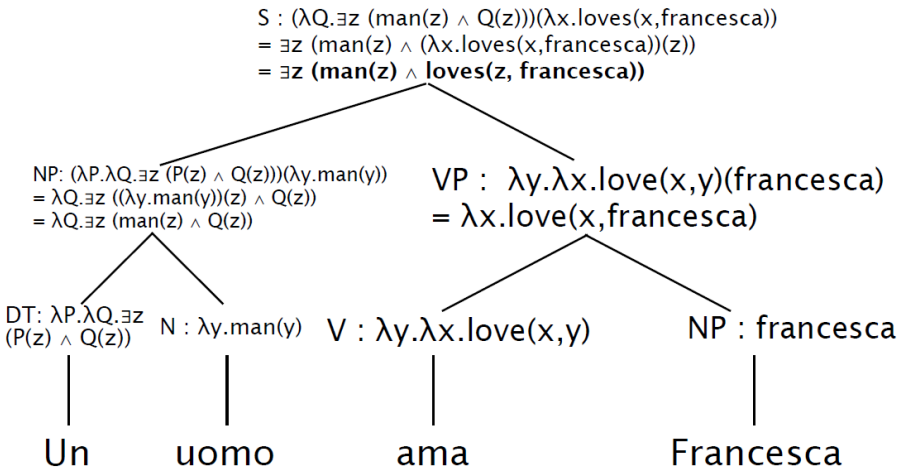


Figura 19: Semantica corretta dell'articolo "un"

Se permettiamo l'astrazione sui predicati otteniamo:

$(\lambda Q.\exists z(\lambda y.man(y)(z) \wedge Q(z)))(\lambda x.love(x, Francesca))$

Di fatto dopo una serie di passaggi come questo andiamo ad ottenere la semantica dell'articolo, che è:

$DT : \lambda P.\lambda Q.\exists z(P(z) \wedge Q(z))$

Notiamo come la semantica dell'articolo è assai più complessa di quella dei verbi o dei nomi. Questo è dovuto al fatto che nomi e verbi sono vocaboli "di contenuto" e dunque si spiegano abbastanza da soli indipendentemente dal contesto. Questo invece non si può dire per gli articoli invece, che dunque hanno una semantica più complicata.
L'ultimo passo da fare è di cambiare la semantica di composizione facendo diventare NP una funzione. A questo punto possiamo gestire con la nuova semantica la frase "un uomo ama Francesca". Mostriamo l'albero in Figura 19

Notiamo che otteniamo delle formule abbastanza lunghe ma, ad ogni modo uno dei vantaggi della semantica è la sistematicità della stessa. Non importa quanto complicata possa essere la semantica di un nuovo termine da aggiungere. Se procediamo al contrario come abbiamo fatto poco fa, possiamo sempre estenderla.
Un ultimo problema che è doveroso citare nell'ambito della semantica è quello della *scope ambiguity*. Una tipica situazione di scope ambiguity viene evidenziata dalla frase che segue:

"In questo paese, ogni 15 minuti una donna mette al mondo un bambino. È nostro compito trovare quella donna e fermarla"[Groucho Marx]

È ovvio che la donna a cui facciamo riferimento non è sempre la stessa, ma per come abbiamo definito la nostra semantica, siamo in grado di cogliere solo uno dei due significati. Non possiamo coglierli entrambi.

Ad ogni modo esistono altri approcci alla semantica di cui possiamo parlare velocemente: il primo è l'*Abstract Meaning Representation*(AMR), basato su una semantica un po' meno formale che va ad ignorare i quantificatori universali, le inflessioni morfologiche per i numeri e non distingue tra eventi reali ed ipotetici come eventi futuri o immaginati.
Il secondo approccio, detto di *Natural language understanding*(NLU), è quello utilizzato nei sistemi di dialogo come gli assistenti vocali, ad esempio Siri.
In questo caso la semantica "finge" che il linguaggio naturale non sia ricorsivo. Questo perché anche se il linguaggio è effettivamente ricorsivo, in realtà molto spesso nelle frasi come "Siri, segna un appuntamento per domani alle 12 con Francesca" non vi sia alcuna forma di ricorsione. Dunque il livello sintattico resta invariato, ma il livello semantico viene in qualche modo semplificato. Si preferisce utilizzare un approccio a regole detto *condition-action*. Ovvero l'assistente vocale si aspetta di ricevere alcune condizioni(sul tempo, sul luogo) e un'azione da eseguire. Se una di queste due componenti viene a mancare Siri ricorda all'utente di fornirgliela.

6 Generazione del linguaggio naturale

In accordo con quanto detto da McDonald nell 1992, con *natural language generation* intendiamo il processo di costruzione di un testo in linguaggio naturale per andare incontro a specifici obiettivi comunicativi.
Lo scopo generale dunque è quello di creare un sistema automatico per produrre testo in linguaggio naturale. L'input sarà una forma di rappresentazione dell'informazione proveniente da qualsiasi origine(dati da una macchina ad esempio) e l'output sarà un documento contenente spiegazioni o messaggi.
Potremmo dire che il natural language processing si compone di due parti diverse:

- Natural language understanding: con cui comprendiamo quello che ci viene detto
- Natural language generation: con cui produciamo qualcosa in output in base a quello che ci è stato detto

Solitamente possiamo o utilizzare dei template(ad esempio il comando di printf in C è un template) oppure possiamo costruire un vero e proprio sistema di NLG. I vantaggi dei template sono che sono facili e veloci da gestire, c'è poca formalizzazione e non facciamo uso di linguistica. I vantaggi del NLG sono che

è mantenibile e produce un output testuale di alta qualità.
Ad ogni modo esistono anche dei sistemi ibridi basati su entrambi gli approcci.
Vediamo un esempio di NLG

Esempio 1 : un programma di NLG abbastanza noto è "BabyTalk". L'idea è che, a fronte dei dati medici provenienti da dei macchinari, il sistema di NLG produca in automatico tre documenti diversi pensati ognuno per un soggetto diverso: uno per i medici, uno per le infermiere, uno per i familiari del paziente.
Per realizzarlo è stato chiesto ai medici come avrebbero scritto una serie di dati provenienti dai macchinari e in base a queste considerazioni vengono prodotti i documenti.

Negli ultimi anni grazie agli studi sui big data c'è un maggiore fermento nel campo dell'NLG perché a fronte di tutti questi dati, dobbiamo anche essere in grado di spiegarli. Ad esempio un'applicazione potrebbe essere di far comprendere come mai una rete neurale ha preso una certa decisione, visto che spesso questo tipo di tecnologie da una risposta ma senza spiegarne i motivi.
Di norma, le tecniche di NLG devono rispondere a sette compiti. Notiamo che a livello di astrazione facciamo un procedimento inverso a quello visto finora: partiamo da un primo livello che contiene il significato e andiamo sempre di più verso la struttura:

1. Content determination: al primo livello siamo molto vicini al significato e lontani dal linguaggio. A questo livello stabiliamo cosa vogliamo dire. Ovvero stabiliamo quali siano i messaggi provenienti dalla struttura dati. Questi messaggi sono basati su entità concetti e relazioni sul dominio

2. Discourse planning: a questo livello ci rendiamo conto che un testo non è solo una collezione casuale di frasi. Abbiamo invece una struttura. Introduciamo dunque due tipi di relazioni: concettuale e retorica

3. Sentence aggregation: per aumentare la fluenza dei messaggi andiamo a combinarli per ottenere frasi certamente più lunghe ma anche più naturali. Senza questa fase otterremmo una serie di frasi corrette ma spezzate, non legate fra loro e quindi poco naturali.

4. Lexicalisation: andiamo a stabilire quali parole utilizzare per esprimere i concetti e le relazioni tra di essi. Questo è il primo livello dipendente dal linguaggio che usiamo. Come dice la parola stessa, ci occupiamo del lessico.

5. Referring expression generation: andiamo a gestire alcune parole ben precise come i nomi propri e i pronomi. Generare le referring expressions determina quali parole usare per esprimere le entità del dominio in una maniera comprensibile all'utente. È infatti il momento in cui dobbiamo cominciare a riferirci non solo alle classi ma anche alle istanze delle stesse(nomi per l'appunto). Ad esempio "Alessandro Mazzei" e "Il professore che tiene la lezione di TLN" si riferiscono alla stessa persona

6. Syntactic and morphological realization: visto che ogni lingua ha una morfologia e una sintassi, utilizziamo le regole morfologiche e sintattiche. Ad esempio una regola morfologica è che in inglese per formare i verbi al passato dobbiamo aggiungere "ed". Una regola sintattica è che il soggeto va prima del verbo o che soggetto e verbo devono concordare sul numero.

7. Ortographic realization: usiamo infine le regole ortografiche per finalizzare la frase. Ad esempio andiamo a gestire le maiuscole e le minuscole, oppure la punteggiatura.

26 / 03 /2018

7 Traduzione automatica

Non appena è stato visto che le macchine potevano essere usate per avere a che fare con il linguaggio umano, il problema della traduzione automatica è diventato subito centrale.
Nel 1933 abbiamo Troyanskii e Artsrouni che in modo del tutto indipendente propongono dei per macchine meccaniche ai tempi in cui un calcolatore non esisteva ancora. I veri investimenti in questo campo arrivano alla fine degli anni '40 con l'inizio della guerra fredda e nel 1956 viene introdotto il termine di intelligenza artificiale.
Nel 1966 tramite un documento detto ALPAC report vengono analizzati i risultati ottenuti in quasi venti anni di ricerca e non soddisfano le aspettative. I fondi vengono ridotti e dopo il '66 sono in molti ad abbandonare il progetto.
Negli anni '90 riemerge un interesse per l'argomento e i primi veri traduttori automatici sono quelli che nascono nel 2000 grazie al web, detti *online machine translator*.
Nel 2007 Google introduce un nuovo approccio statistico che apporta qualche miglioramento ma il vero salto di qualità lo abbiamo nel 2016 quando sempre Google introduce un approccio basato su reti neurali.
Complessivamente non abbiamo ancora raggiunto un livello di traduzione automatica di alta qualità però questi strumenti a nostra disposizione possono darci un grande aiuto.

Uno strumento di grande importanza nell'ambito della traduzione automatica è il *triangolo di Vauquois* che cominciamo a mostrare in Figura 20

Vauquois ha cercato di schematizzare il procedimento di traduzione andando ad utilizzare la pipeline usata anche da noi ma non soltanto per la fase di analisi, ma anche per la fase di generazione. Ad esempio possiamo notare che abbiamo due semantiche perché la semantica dipende chiaramente dal linguaggio. Mostriamo ad esempio come ad ognuna delle fasi corrisponda una scomposizione o composizione degli elementi in modo simile a quello visto da noi. Questo fatto viene mostrato in Figura 21

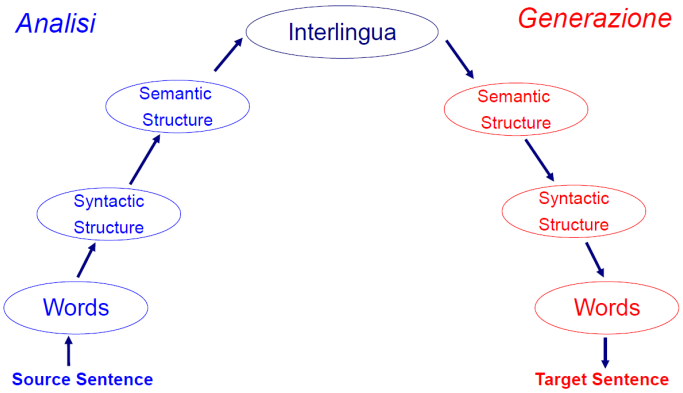


Figura 20: Analisi e generazione nel triangolo di Vauquois

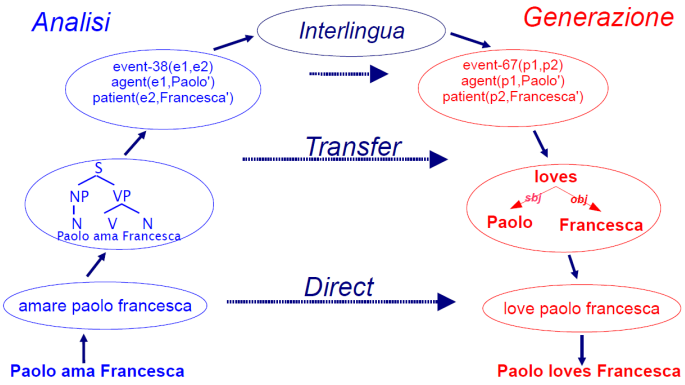


Figura 21: Componenti e tipi di traduzione nel triangolo di Vauquois

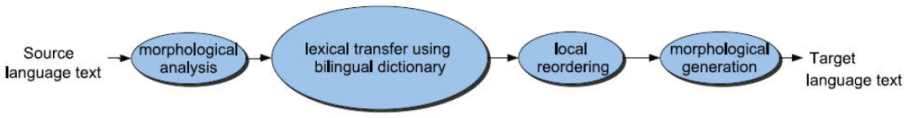


Figura 22: Procedimento di traduzione diretta

L'idea di utilizzare tre livelli sia in fase di analisi che in fase di generazione permette a Vauquois di formalizzare tre tipi di traduzione, sempre mostrati in Figura 21:

- Direct MT: la traduzione procede parola per parola dalla lingua di origine alla lingua d'arrivo. Mostriamo in Figura 22 l'intero procedimento di traduzione diretta

  Tramite le fasi mostrate andiamo prima ad effettuare l'analisi morfologica della frase nella lingua in ingresso. Dopo andiamo a tradurre termine per

termine tramite l'operazione di trasferimento lessicale. Poi andiamo ad effettuare un riordinamento e infine otteniamo mediamente la morfologia la frase nella lingua di uscita. Il vero problema in questa fase è quello di riordinamento locale che può essere molto complicato

- Transfer MT: prima deriviamo la struttura sintattica o parzialmente semantica della frase, e partendo da questa generiamo la frase nella lingua di arrivo. Possiamo scomporre il procedimento di traduzione transfer in tre fasi:

  1. La prima è di analisi in cui facciamo analisi morfologica, chuncking e costruiamo le dipendenze
  2. Nella seconda fase, di transfer, costruiamo la traduzione degli idiomi ed effettuiamo alcune operazioni di disambiguazione.
  3. Nell'ultima fase, detta di sintesi, andiamo ad utilizzare un dizionario bilingue per la traduzione dei termini e abbiamo a che fare con il riordinamento per poi andare ad eseguire una generazione morfologica.

  A livello pratico le regole di transfer vanno ad operare sugli alberi cambiandone la struttura e operando quel riordinamento che era complicato da fare nel caso della traduzione diretta. Per questo l'operazione di traduzione transfer può essere complessa.

- Interlanguage MT: andiamo a rappresentare il significato della frase nella lingua di origine in una lingua intermedia detta *interlingua*. A partire da questa andiamo a generare la frase nella lingua di arrivo. In questo caso l'analisi linguistica è completa e possiamo rappresentare la conoscenza e generare frasi. Ovviamente anche questo approccio porta con sé dei problemi. Cosa rappresentare in interlingua? Quale dovrebbe essere il giusto livello di dettaglio? Esistono lingue che possono esprimere in molti modi diversi una stessa parola o in cui alcuni concetti non hanno un corrispettivo in un'altra lingua. Ad esempio in giapponese abbiamo due parole distinte per il fratello minore e quello maggiore, mentre in italiano no!

Quale dei tre approcci descritti dovremmo utilizzare? Solitamente un procedimento di traduzione che coinvolga più livelli è più accurato e dunque va da sé che la traduzione di tipo interlanguage sia migliore di quella direct. Per questo motivo di solito si tende ad usare un approccio tanto più complesso quanto più sono lontane la lingua di partenza e quella di arrivo. Da italiano a napoletano è probabile che basti una traduzione direct, ma da italiano a giapponese avremo probabilmente bisogno della traduzione interlingua.

Un approccio diverso cui accenniamo è quello delle *word alignment* in cui andiamo a studiare la probabilità di allineamento di frasi e parole. Questo metodo è stato utilizzato ampiamente nello studio della stele di Rosetta ad esempio.

L'ultimo approccio di cui parliamo è quello basato su reti neurali. In realtà un modello basato su reti neurali era già stato proposto in precedenza, intorno agli anni '80. All'epoca mancavano però i dati e la tecnologia(ovvero le schede video). Ad ogni modo l'approccio basato su reti neurali in termini di accuratezza è certamente il migliore e per alcuni linguaggi non è neanche troppo lontano dai valori ottenuti da traduttori umani.

Parte Terza:
Significato,
semantica distribuzionale
e social media NLP

07 / 05 /2018

8 Complessità e composizionalità

8.1 Significato del significato

Per cominciare dobbiamo partire da alcune definizioni già incontrate in precedenza

- Lessico: insieme degli elementi linguistici a disposizione
- Sintassi: indica come gli elementi di un dizionario possono essere interconnessi tra loro per costruire frasi
- Semantica: interpretazione di una struttura lessico-sintattica
- Pragmatica: livello di significato più fine, che prescinde dall'uso del lessico e della sintassi, e può essere anche indipendente dal livello semantico. Prende in considerazione non solo il contenuto semantico ma anche informazioni non esplicitate che riguardano il discorso o un background psicologico, cognitivo, contestuale
- Comunicazione: il linguaggio nasce come strumento per condividere significati contenuti all'interno della nostra mente; serve quindi per comunicare
- Convenzione: poiché la comunicazione è un concetto ampio, sia verbale che non verbale, nel caso verbale si usa la convenzione di trasferire un contenuto semantico attraverso dei simboli.
- Granularità: tutta la struttura del linguaggio è più o meno profonda. L'approccio per studiarlo è una vista su questa struttura. Il contenuto semantico, cambia a seconda che ci concentriamo su una parola, una frase o un testo
- Cultura: una parola, una convenzione, cambia a seconda della cultura a cui si fa riferimento
- Senso comune: strettamente legato alla cultura. È il significato che diamo alle convenzioni, in maniera condivisa all'interno della cerchia culturale, viene usato per esprimere concetti diversi.
- Esperienza personale: dinamica che cambia il significato del significato
- Similarità: usata quasi in maniera automatica, è il meccanismo che ci permette di adeguarci a situazioni non previste e non note, e che ci permette di capire il significato di qualcosa di nuovo. È fondamentale nell'NLP perché spesso i calcolatori devono avere a che fare con situazioni non note a priori

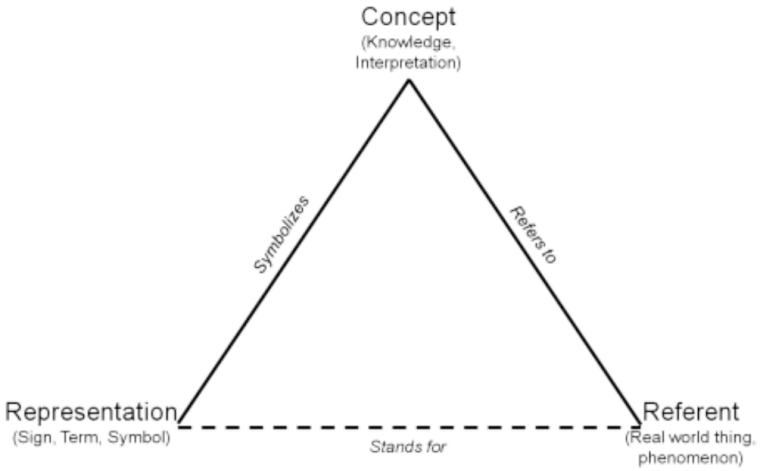


Figura 23: Struttura del triangolo semeiotico

Grazie a tutte queste definizioni abbiamo creato una *ontologia* di base: in questo senso non ci interessa del significato singolo, bensì del significato condiviso fra i concetti.
Per quanto riguarda il significato delle parole, detto anche *word meaning*, esistono diverse teorie:

- Basate su primitive: per rappresentare il significato di una parola dobbiamo frammentarlo in piccoli contenuti semantici di natura più atomica. Ad esempio per comprendere il significato di scrivania devo capire il concetto di tavolo, e ancora prima di struttura piana con fondamenta
- Basate su relazioni: il significato che sta dietro una parola nasce dalla relazione con altre parole. Una parola di per sé non ha significato se non impiegata in un contesto fatto di altre parole
- Basate su composizioni: non solo una parola prende significato quando inserita in un contesto lessicale, ma la composizionalità stessa delle parole produce significati nuovi e le parole stesse prendono un significato individuale. Quindi è la composizione tra parole che dà significato.

Un modello del significato molto utile in questo ambito è quello del *triangolo semeiotico* in cui qualsiasi concetto è rappresentabile attraverso il triangolo mostrato in Figura 23

Nel triangolo abbiamo: un *concetto* che è la rappresentazione che abbiamo in testa. Ad esempio è l'idea di gatto che abbiamo immagazzinato nel tempo all'interno della mente.
Abbiamo poi una *rappresentazione* che si basa sulle convenzioni e ci permette di comunicare il concetto. Ad esempio è la vera e propria parola "gatto".
Il referente è invece l'elemento reale nel mondo, ad esempio il gatto vero e

proprio.
Esistono alcuni aspetti interessanti del triangolo che dobbiamo citare

- Per comunicare abbiamo effettivamente bisogno della rappresentazione ma non del referente

- La rappresentazione è l'unico punto da cui può partire un algoritmo di intelligenza artificiale/NLP. L'intuizione è che un algoritmo dovrebbe partire dalla rappresentazione per cercare di coprire al massimo gli altri due vertici, cercando di comprendere al massimo il concetto

Al giorno d'oggi abbiamo anche un altro aspetto da considerare, ovvero il *multilingual word meaning*: l'abbattimento delle barriere linguistiche viene visto almeno in parte come una opportunità. Infatti attraverso un confronto dei testi diventa possibile comprendere le sfumature di significato tra le lingue.
Diverso è il caso delle lingue rare in cui, in assenza di un dizionario, tutto diventa più difficile.
Infine abbiamo anche il problema dei "concetti comunque diversi" secondo cui alcuni concetti saranno sempre diversi indipendentemente dalla traduzione che fornisce una sua sfumatura concettuale.
Infine dobbiamo anche considerare che alcuni concetti non sono verbalizzati, ad esempio "spaghettata" in inglese non esiste...

L'analisi semantica può anche essere fatta in modo molto diverso a seconda della granularità che decidiamo di utilizzare: perché ad esempio a livello di parola abbiamo la word sense disambiguation, se invece consideriamo i chunk abbiamo la multiword expression, a livello di documento abbiamo invece la summarization...
Consideriamo in maggior dettaglio la *word sense disambiguation*(WSD). Per essere effettuata dobbiamo avere a che fare con dei problemi di specificità, copertura e soggettività. Un esempio di strumento che svolge egregiamente questo compito è WordNet. Una variante della WSD è la *word sense induction* in cui non abbiamo un dizionario contenente tutti i possibili sensi di una parola. Il senso viene invece dedotto cercando all'interno di grandi quantità di testi. Dunque la valutazione è più complessa e si basa sul metodo della *pseudo-word*. Con questo metodo che si basa su tecniche di clusterizzazione andiamo a valutare quanto cambino i cluster delle parole se andiamo a sostituire coppie di parole casuali con la loro concatenazione. L'idea è di andare a costruire dei cluster che interpretino i concetti originari. Se i concetti vengono separati nettamente allora il procedimento è andato a buon fine. Purtroppo in certi casi invece potremmo trovare i concetti mescolati fra loro. In tal caso il procedimento non è andato a buon fine.

Facciamo invece una breve digressione sulla semantica lessicale. Ad oggi esistono molti dizionari elettronici come WordNet, BabelNet. Esistono anche delle risorse linguistico-cognitive dette *property norms*. Queste risorse descrivono delle proprietà che vengono in mente quando si descrive un concetto.

Ovviamente è un po' paradossale pensare di descrivere una parola mediante altre parole. Meglio cercare di definire dei concetti. Possiamo farlo andando a chiederci cosa inserire nella definizione di un concetto? Come valutiamo la bontà di una definizione? Notiamo inoltre che dare una definizione univoca è praticamente impossibile. In particolare se il concetto da definire è astratto.

14 / 05 /2018

8.2 Costruzione del significato

Lo scopo di questa sezione è di mostrare come si possa costruire il significato andando ad utilizzare più termini insieme.
Pustejovsky ha creato una teoria della semantica sviluppata su diverse strutture:

- Struttura argument: visione della parola come una funzione con arietà specificata. Fondamentalmente esprime una corrispondenza tra una parola e la struttura sintattica. In questo Pustejovsky si basa in parte su precedenti definizioni, come quella di Grimshaw.
- Struttura ad evento: identificare una parola mediante un evento associato, ad esempio uno stato, un processo, una transizione. Questa deinifizione è invece basata in parte su quella di Vendler.
- Struttura qualia: la definizione degli attributi essenziali di un oggetto mediante entità lessicali.
  Questa struttura è particolarmente importante perché ogni qualia esplica quattro aspetti del significato di una parola. Ognuno di questi aspetti è identificabile con un *ruolo*
  - Ruoli costitutivi: la relazione tra un oggetto e i suoi costituenti. Dunque materiali, pesi, componenti
  - Ruoli formali: caratteristiche che distinguono il concetto dagli altri all'interno di un dominio. Ad esempio orientazione, dimensione, colore, forma...
  - Ruoli telic: definiscono lo scopo e il comportamento del concetto
  - Ruoli agentive: relativi alla creazione del concetto, alla sua origine
- Struttura ad eredità: come la parola è relazionata globalmente agli altri concetti

La teoria appena esposta è completa ed esatta però ha il problema di essere difficile da implementare!

La seconda teoria cui ci dedichiamo è quella di Hanks: è più semplice da implementare rispetto a quella di Pustejovsky.
In questa teoria il verbo è la radice del significato, poiché non esistono espressioni senza verbo. Ad ogni verbo viene associata una *valenza*, ovvero il numero di

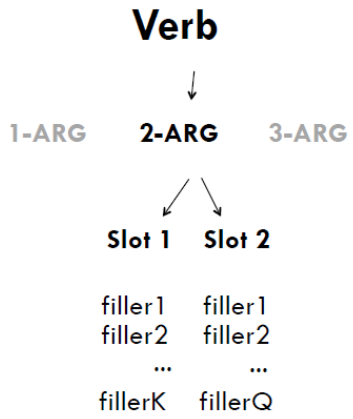


Figura 24: Argomenti, slot e filler nella struttura di Hank

argomenti necessari per il verbo. In base al numero di argomenti che un verbo richiede, in certi casi possiamo differenziarne il significato.
Una volta che, per un certo verbo, abbiamo selezionato una valenza ad n argomenti, dobbiamo andare a specificarli mediante un certo numero di *slot*. Dunque avremo n slot. Ogni slot può avere un certo numero di valori possibili che li riempiano. Tali valori vengono detti *filler*.
Inoltre ad ogni filler possiamo associare dei *tipi semantici* che rappresentano delle generalizzazioni concettuali strutturate come una gerarchia.
Mostriamo graficamente quanto appena detto in Figura 24

Dunque il significato di un verbo dipende dagli argomenti e dai tipi semantici associati.
Anche in questo modello abbiamo però dei problemi. Infatti dobbiamo definire con esattezza i tipi semantici e dunque a quale livello gerarchico dobbiamo fermarci? Ad esempio ci fermiamo al concetto di "persona" o introduciamo anche gli "animali" per poter raccogliere tutto sotto il concetto "esseri viventi"? Oppure potremmo avere dei termini che si riferiscono a un certo livello di generalizzazione ma in modo dipendente dal contesto.

Esempio 1 : se analizziamo la frase "lo studente va a scuola" dobbiamo capire quali proprietà attivare in relazione al tipo semantico. Ovvero, per lo studente dobbiamo usare un tipo semantico studente? O persona? Probabilmente "essere vivente" è troppo generico visto che non permette di attivare abbastanza proprietà specifiche per lo studente.

I problemi emersi con la teoria di Hanks vengono risolti andando a concentrarsi sulle proprietà. Questo viene fatto con una terza teoria detta *affordance linguistica*.
Il termine affordance viene utilizzato all'interno della teoria per specificare che un oggetto quando viene percepito fornisce dei suggerimenti su come utilizzarlo. Fornisce dunque delle possibilità d'uso, dette appunto affordance. Queste

possibilità le abbiamo anche se l'oggetto non lo abbiamo mai visto.
Questo concetto può anche essere esteso alla linguistica. Infatti in presenza di parole non note possiamo ricavarne il significato dal contesto in cui appare. Dunque il contesto crea delle proprietà semantiche.
Di fatto il significato che attribuiamo a una parola è dato dalle proprietà delle parole che gli associamo. Alcune proprietà possono essere conosciute e altre assunte.
Con questo approccio il potere espressivo cresce notevolmente perché possiamo comprendere parole che non avevo nel corpus iniziale andando a studiarne le proprietà.
Diventa importante stabilire dove e come prendere le proprietà. Solitamente andando a cogliere proprietà latenti derivanti dalla similarità, o andando ad effettuare estrazione da risorse linguistiche o tramite questionari, studi cognitivi...
Tutti questi dati possono essere presi da risorse linguistiche come annotazioni manuali, corpora, o ancora tramite strumenti di machine learning.
Ad ogni modo non abbiamo bisogno di grandi quantità di dati perché non esiste un vero e proprio corpus "completo" e anche perché il numero di proprietà utilizzate cresce secondo il logaritmo dei termini introdotti

15 / 05 /2018

9 Tipi di semantica

In questa sezione ci dedicheremo all'analisi di vari problemi: il text-mining, la semantica documentale e la sua visualizzazione, la semantica distribuzionale e infine parleremo di ontologie

9.1 Text mining

I problemi di *text mining* vengono posti per la prima volta per cercare di risolvere il problema del *question answering*: l'idea è che un utente umano ponga delle domande a cui il calcolatore deve rispondere in base alle informazioni che trova in un testo. Le domande possibili riguardano dei fatti, delle definizioni...In realtà il campo di ricerca è molto ampio visto che esistono anche dei settori di ricerca intesi a studiare una categorizzazione per le domande. Infatti l'idea è che ad ogni domanda siano associate diverse informazioni come le parole, la punteggiatura, i lemmi, i sinonimi...
Nell'ambito del NLP lo scopo diventa proprio quello di reperire in modo automatico queste informazioni. Dunque buona parte della ricerca è ad oggi quello che sta dietro al question answering. Ne sono esempi l'estrazione di iperonimi, la word sense disambiguation...
Fra le varie semantiche possibili abbiamo la semantica lessicale, formale, statistica e linguistico-distribuzionale. Oggi ci concentriamo sulla semantica statistica tramite il text mining.
Quando parliamo di text mining ci riferiamo ad un insieme di tecniche del data

mining applicate allo studio di testi.
Gli approcci di linguistica computazionale che abbiamo visto finora studiano i formalismi descrittivi del funzionamento del linguaggio naturale e utilizzano un approccio top-down.
Oggi invece ci occupiamo di statistica che studia qualitativamente e quantitativamente certi fenomeni mediante un approccio bottom up.
La differenza è che nell'approccio top-down della linguistica computazionale andiamo a costruire delle regole per comprendere l'espressione linguistica. Invece con l'approccio statistico andiamo ad estrarre le regole mediante evidenze statistiche.
Dal punto di vista statistico le parole sono dei token da analizzare e dunque non ci interessiamo dei caratteri che le compongono né della sintassi in generale.
In base a questa assunzione un documento può essere visto come un vettore di coppie $(parola, \#\text{occorrenze})$. Per crearlo procediamo come segue: ogni volta che troviamo una parola nuova la aggiungiamo al vettore segnando che la parola è occorsa per la prima volta. Ogni volta che incontriamo una parola già vista semplicemente incrementiamo la componente corrispondente nel vettore.
Ovviamente se facciamo questo procedimento separatamente per tanti documenti otteniamo una matrice dove ogni riga rappresenta un documento e ogni colonna rappresenta un possibile vocabolo. Un problema di questa matrice è che è abbastanza sparsa in senso vettoriale.
Perché costruire questa rappresentazione? Perché il vantaggio che ne ricaviamo è che diventa molto semplice misurare la similarità tra documenti mediante la *cosine similarity*:

$$sim = cos(\theta) = \frac{A \cdot B}{\| A \| \cdot \| B \|} \tag{7}$$

Notiamo infatti che ogni documento della matrice(dunque ogni riga) è di fatto un punto di $\mathbb{R}^n$ dove n è il numero di colonne della matrice. A questo punto diventa facile osservare che il valore tra due documenti(dunque tra due vettori) sarà tanto più alto quanto più i due vettori puntano nella stessa direzione.

Un approccio differente è quello che si concentra su altre due informazioni statistiche importanti, la *term frequency*(TF) e la *inverse document frequency*(IDF):

- Term frequency: indica quante volte un termine compare in un certo documento. È definita come:

$$tf_{i,j} = \begin{cases} 1 + log_2(f_{i,j}) & \text{con} f_{i,j} > 0 \\ 0 & \text{altrimenti} \end{cases} \tag{8}$$

  Notiamo che il valore di tf cresce seguendo il logaritmo perché se un termine compare mille o duemila volte non è così differente.

- Inverse document frequency: valuta quanto un termine è specifico per un documento. Dunque va a valutare in quanti documenti un certo termine

compare. È definito come:

$$idf_i = log_2(\frac{N}{n_i}) \tag{9}$$

dove dove N è il numero totale di documenti ed n_i è il numero di documenti in cui il termine i appare.
Notiamo che questa quantità serve per cercare di escludere tutte quelle parole (come gli articoli) che compaiono molte volte in un documento ma non sono specifiche del documento perché le possiamo trovare anche in ogni altro documento

Solitamente per valutare quanto una parola sia interessante viene utilizzato il prodotto delle due quantità appena descritte:

$$tf - idf = tf_{i,j} \cdot idf_i \tag{10}$$

Un altro approccio per determinare quanto due parole siano correlate in un testo consiste nel costruire una matrice di co-occorrenza che mostra per ogni parola quanto spesso un'altra parole le compare "vicino". Questo ci permette di implementare la nozione di "contesto" in cui una certa parola appare.

21 / 05 /2018

9.2 Applicazioni del text mining

Vediamo adesso alcune applicazioni dei concetti di text mining visti poc'anzi

1. Tag cloud: nelle tag cloud le parole figurano con una dimensione proporzionale alla loro frequenza. Possiamo includere nelle tag cloud l'informazione di co-occorrenza? Sì, magari andando a rappresentare le parole ad una distanza proporzionale alla loro co-occorrenza

2. Tag flakes: in questo caso viene fatto un primo ordinamento in base alla frequenza e successivamente andiamo a separare le dimensioni su una base semantica. Questo procedimento viene mostrato in Figura 25

   Alternativamente possiamo estrarre in modo automatico una gerarchia dei termini in cui sia rappresentata la reciproca similarità. Ovviamente per fare questo abbiamo bisogno di un buon algoritmo di costruzione della gerarchia. Mostriamo una costruzione di questo tipo in Figura 26

3. Document clustering: con un insieme di tecniche di apprendimento non supervisionato andiamo a raggruppare i documenti che hanno qualcosa in comune. Non esiste un "clustering perfetto" perché stiamo semplicemente ottimizzando una funzione e quindi tutto dipende da cosa vogliamo evidenziare e che funzione ottimizziamo. In certi casi le tecniche di clustering vengono utilizzate in cascata con altre. Ad esempio potremmo effettuare un procedimento di clustering e in seguito andare a creare una tag cloud per ogni cluster.

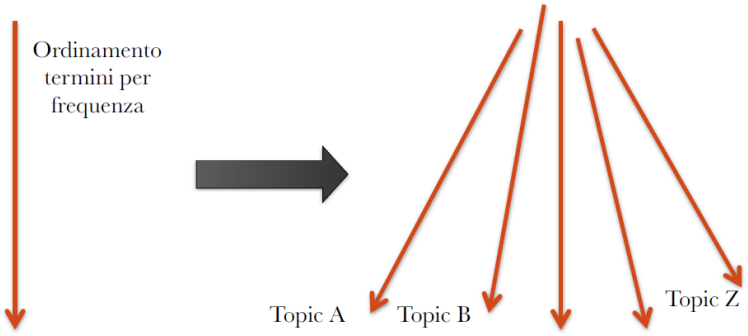


Figura 25: A sinistra il primo ordinamento per frequenza, a destra la suddivisione semantica

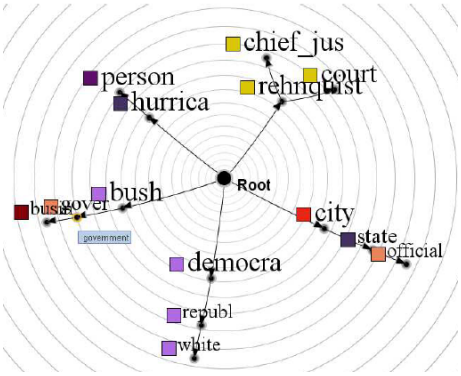


Figura 26: Rappresentazione gerarchica dei tag flakes

4. Document classification: come possiamo andare ad associare un documento ad una certa etichetta di una tassonomia? Possiamo farlo mediante delle tecniche di apprendimento non supervisionato.
   Una prima tecnica banale consiste nel prendere il vettore che rappresenta il documento e andare a verificare quale fra tutte le etichette della tassonomia compare più volte nel documento. Questa idea però è fin troppo semplice e fornisce risultati approssimativi.
   Una seconda tecnica che possiamo utilizzare consiste nell'andare a costruire una matrice $n \times n$ dove n è il numero di etichette della tassonomia. Inizialmente la matrice coincide con la matrice identità $n \times n$.
   Successivamente, ogni elemento della tassonomia cede una parte di "informazione" ai nodi con lui comunicanti. Dunque se un nodo x è collegato con un nodo y nella tassonomia, allora incrementeremo il valore della matrice in posizione $[pos(x)][pos(y)]$.
   La quantità di informazione trasferita viene pesata per un coefficiente stabilito a priori presente su ogni arco della tassonomia.

	world	Asia	Africa	America	Afghanistan	Iraq	China	Canada	US
$\vec{cv}_{world}$	0.450	0.169	0.141	0.158	0.018	0.018	0.018	0.021	0.021
$\vec{cv}_{Asia}$	0.052	0.469	0.006	0.006	0.156	0.156	0.156	0.0003	0.0003
$\vec{cv}_{Africa}$	0.100	0.012	0.873	0.012	0.0006	0.0006	0.0006	0.0007	0.0007
$\vec{cv}_{America}$	0.057	0.007	0.007	0.520	0.0003	0.0003	0.0003	0.204	0.204
$\vec{cv}_{Afghanistan}$	0.004	0.100	0.0002	0.0002	0.872	0.012	0.012	0	0
$\vec{cv}_{Iraq}$	0.004	0.100	0.0002	0.0002	0.012	0.872	0.012	0	0
$\vec{cv}_{China}$	0.004	0.100	0.0002	0.0002	0.012	0.012	0.872	0	0
$\vec{cv}_{Canada}$	0.006	0.0003	0.0003	0.165	0	0	0	0.806	0.023
$\vec{cv}_{US}$	0.006	0.0003	0.0003	0.165	0	0	0	0.023	0.806

Figura 27: Esempio di matrice ottenuta dopo un certo numero di iterazioni

Questo procedimento di trasferimento di informazione procede per un numero stabilito a priori di iterazioni. Alla fine otterremo una matrice simile a quella mostrata in Figura 27

A questo punto per associare un documento alla tassonomia andiamo a ridurre il vettore del documento alla stessa dimensione del numero di etichette. Solitamente in questo procedimento di riduzione manteniamo le componenti del vettore documento che coincidano con le etichette della tassonomia o che siano semanticamente correlate. Andiamo infine a calcolare la similarità del vettore documento ridotto con ogni vettore della tassonomia e associamo il documento all'etichetta il cui vettore presenta maggiore similarità

5. Document segmentation: in questo caso lo scopo è di individuare elementi diversi del discorso all'interno dei documenti per analizzare l'evoluzione di un tema, delle relazioni e altro ancora.
   Il metodo più famoso in questo campo è quello del *text tiling*.
   L'idea è di separare un testo in finestre di lunghezza fissata a priori. All'interno di ogni finestra viene calcolata una *coesione intra-gruppo*, una quantità che misura quanto le parole all'interno di una finestra sono legate fra loro. Che misura utilizzare come coesione intra-gruppo non è importante. Ne esistono diverse implementazioni, ad esempio basate sulla co-occorrenza. A questo punto andiamo a cercare i *break point*, ovvero quei punti in cui la coesione intra-gruppo si abbassa bruscamente. È molto probabile che in tali punti finisca una sezione del documento relativa ad un argomento e ne cominci una nuova. Mostriamo questa situazione in Figura 28

   Questo ci permette di abbandonare la lunghezza fissa delle finestre e andare ad individuare delle finestre in base ai break point.

6. Riassunto automatico di documenti: viene fatto solitamente in due modi

   - Estrattivo: dato il testo cerchiamo di attribuire un valore di *salience*, ovvero di importanza, alle frasi. Questo ci permette di evidenziare le frasi maggiormente significative e utilizzarle nel riassunto

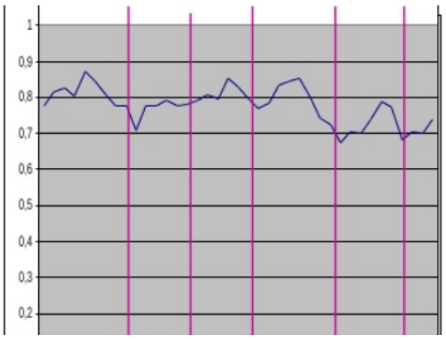


Figura 28: Sulle ordinate il valore di coesione intra-gruppo. Abbiamo evidenziato in viola i break point

- Astrattivo: metodi più complessi perché non vanno a selezionare frasi ma vanno a generare nuovi documenti e quindi il riassunto astrattivo prevede l'utilizzo di raffinate tecniche di natural language generation

7. Orienteering e browsing: partendo dall'idea che le persone non sempre sanno come cercare informazioni, lo scopo è di sviluppare dei meccanismi per aiutare gli utenti a navigare i topic sfruttando le relazioni tra i documenti e altro ancora. Ad esempio sfruttano relazioni latenti tra documenti e abilitando la navigazione guidata del contesto

8. Information retrieval: ultimo ma non meno importante, è in realtà lo scopo del text mining. Inizialmente si cercava un match tra parole chiave e documenti. Nel tempo si è evoluto andando a considerare anche link, snippet e altre informazioni.
   Spesso nell'information retrieval ritroviamo l'idea di *rete semantica*. La rete semantica viene a formarsi quando andiamo ad esplicitare i legami fra le tre componenti fondamentali: query, documenti e concettualizzazioni. Infatti mediante una query reperiamo dei documenti. Tali documenti sono relativi a dei concetti nella concettualizzazione. Questo crea un collegamento implicito tra query e concettualizzazione. Solitamente una rete semantica è qualcosa di simile a quanto mostrato in Figura 29

9.3 Semantica documentale e visualizzazione

Nell'ambito della semantica documentale esistono diversi modelli che possiamo utilizzare.
Il primo è detto *topic model*. È un modello statistico e probabilistico che analizza l'uso del linguaggio ed individua automaticamente gli argomenti in una collezione di testi. Il modello viene detto non supervisionato perché non necessita di avere a disposizione delle annotazioni manuali. Di fatto un topic è una lista pesata di parole estratta dal documento.

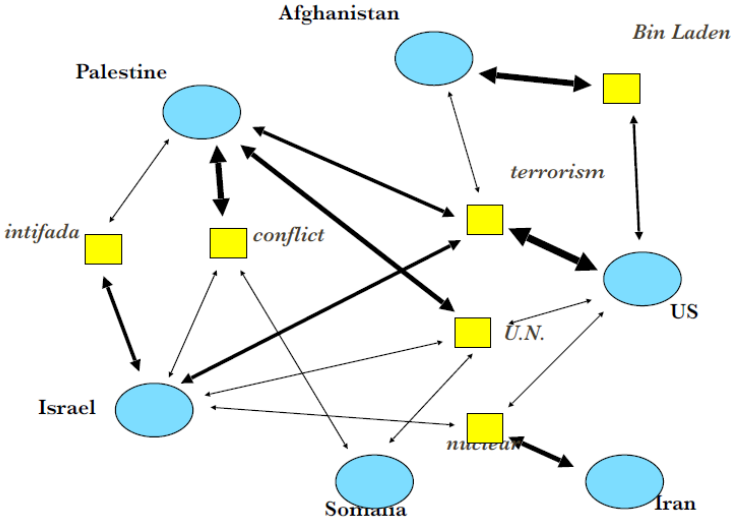


Figura 29: Esempio di rete semantica

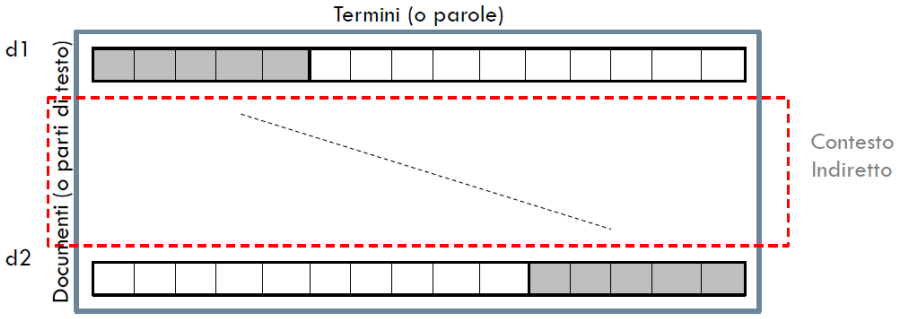


Figura 30: Rappresentazione originale prima della costruzione dell'SVD

Questo approccio porta con sé due problemi: non è sempre facile interpretare l'output del documento e inoltre i topic estratti non sono sempre utili.

Un approccio diverso è quello della *latent semantic analysis* che è invece basata su una fattorizzazione matriciale detta *singular value decomposition*(SVD) che da luogo a dei vettori meno sparsi.
Con questo approccio possiamo andare a catturare concetti latenti valutando delle possibili combinazioni lineari delle feature originali.

Esempio 1 : supponiamo di avere la situazione mostrata in Figura 30

> In questo caso, andando a calcolare la cosine similarity tra i documenti d1 e d2, otteniamo un valore pari a zero. Purtroppo in questo caso la misura

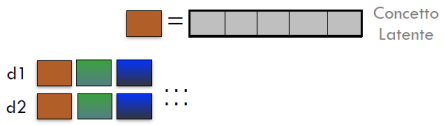


Figura 31: Rappresentazione dopo la costruzione dell'SVD

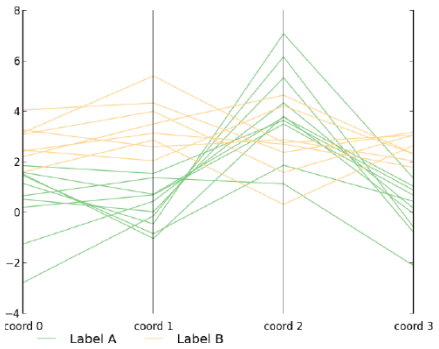
di similarità non tiene di conto del concetto latente e dunque dobbiamo farlo noi andando ad utilizzare il procedimento di costruzione dell'SVD.
Il procedimento di fattorizzazione genera due rappresentazioni diverse da quelle originali ma simili fra loro, come quelle mostrate in Figura 31

In questo caso la rappresentazione è più densa e rende i due documenti più simili fra loro.

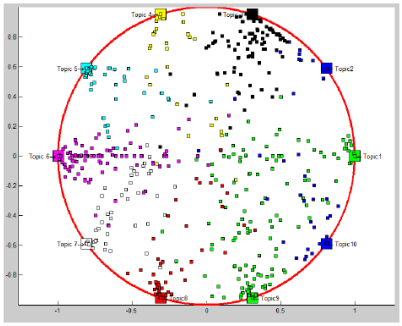
Purtroppo il modello appena visto non generalizza bene su documenti non ancora visti. Inoltre dopo la fattorizzazione potremmo avere dei valori negativi che sono difficili da interpretare. Per questo sono state create delle varianti ad esempio la non-negative matrix factorization(NNMF), oppure versioni probabilistiche come la pLSA che sfrutta la statistica bayesiana.
O ancora l'approccio *Dynamic Topic Modeling*(DTM) va a concentrarsi sull'evoluzione dei topic nel tempo a patto di avere un corpus annotato con valori temporali per i topic.

Un problema comune a tutto il text mining e alla semantica documentale è: come possiamo andare a rappresentare uno spazio a n dimensioni(quello dei termini) quando abbiamo soltanto un foglio bidimensionale?
Abbiamo diversi approcci che mostriamo in Figura 32

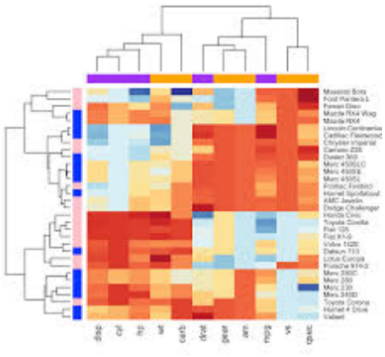
- Parallel coordinates: per ognuna delle coordinate(dei termini) abbiamo una barra che ci indica le occorrenze di quella coordinata in un certo documento. Questo approccio viene mostrato nella parte a) di Figura 32
- RadViz: in questo caso ogni termine rappresenta un punto di attrazione in un cerchio e i vari documenti si dispongono a una distanza da tali punti in modo proporzionale al numero di occorrenze di quel termine. Questo secondo approccio viene mostrato nella parte b) di Figura 32
- HeatMap: in questo caso i singoli valori contenuti nella matrice sono rappresentati da colori. Questo approccio viene mostrato nella parte c) di Figura 32
- Correlation circle: in questo caso i termini sono disposti lungo una circonferenza e collegati fra loro mediante delle rette che ne indicano la correlazione. Questa rappresentazione viene mostrata nella parte d) in Figura 32



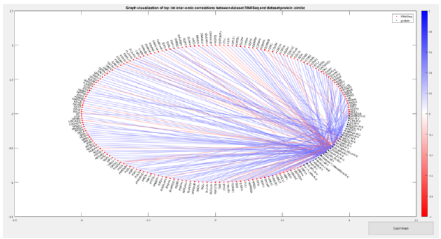
(a) Parallel coordinates



(b) RadViz



(c) HeatMap



(d) Correlation circle

Figura 32: Visualizzazione nella semantica documentale

9.4 Semantica distribuzionale

Potremmo dire che la semantica distribuzionale è una rivisitazione in chiave linguistica delle tecniche di text mining. Nel tempo la storia della semantica distribuzionale ha visto alcuni protagonisti

- Harris: nel 1954 per primo afferma che parole che occorrono in stessi contesti tendono ad avere significati simili
- Firth: similmente, nel 1957, afferma che una parola è caratterizzata da quelle che la accompagnano
- Furnas: nel 1983 sostiene che la congiunzione di varie parole ci permette di specificare più precisamente l'oggetto del discorso
- Deerwester: nel 1990 introduce i concetti latenti con cui si afferma che esista una struttura sottostante che viene parzialmente oscurata dalla scelta "casuale" delle parole che viene fatta per comporre un discorso
- Bley: nel 2003 introduce una visione probabilistica secondo cui il tema del documento influenza in modo probabilistico le parole utilizzate nel documento
- Turney: sempre nel 2003 impiega per la prima volta delle coppie di parole. L'intuizione è che se una certa coppia di parole (x, y) presenta gli stessi pattern della parola (w, z) allora possiamo costruire una sorta di "proporzione" nel significato delle due coppie

Dobbiamo anche dire che la semantica distribuzionale assume nomi diversi a seconda del contesto in cui viene utilizzata. Ad esempio nell'ambito dell'information retrieval viene nominata come "vector space model", nell'ambito delle scienze cognitive viene nominata come studio dei "conceptual spaces". O ancora nella teoria dei grafi viene studiata mediante le "matrice di adiacenza"...
Una prima domanda da porsi è la seguente: come mai fino ad ora abbiamo usato solo le matrici? E perché questo approccio funziona abbastanza bene?
Di fatto la risposta a entrambe le domande è che le matrici sono una approssimazione e questo ci semplifica la vita anche se porta anche a dei risultati meno precisi. Infatti con delle semplici matrici perdiamo ad esempio l'ordinamento delle parole, ma abbiamo solo le frequenze.
A questo riguardo le matrici si collocano tra due rappresentazioni:

- Rappresentazione puramente simbolica: facile da utilizzare ma di per sé povera di informazione
- Rappresentazione associazionistica/connessionistica: in questa seconda rappresentazione proveniente da studi di psicologia si afferma che la rappresentazione del significato nella nostra mente sia funzione di tutto il nostro vissuto e che quindi dovremmo sempre tenerne di conto. Sebbene questa rappresentazione sia carica di informazione è infattibile da implementare

Dunque le matrici si collocano fra queste due rappresentazioni facilitando la condivisione della conoscenza.
Infatti nella matrici l'idea è di andare porre sulle righe le entità basilari e sulle colonne le qualità. In questo senso possiamo applicare delle tecniche di tassellazione di Voronoi per dividere lo spazio rappresentato dalla matrice in regioni. Dunque il significato viene effettivamente rappresentato mediante una regione di uno spazio vettoriale costruito in base alle entità e alle qualità.

A questo punto, avendo assunto l'utilizzo delle matrici, possiamo andare ad utilizzare le tecniche note su questo tipo di dato: dunque le misure di similarità, le strategie di pesatura, le trasformazioni matriciali e le tecniche di clustering. Ad ogni modo per ottenere i migliori risultati esistono delle operazioni di preprocessing che dovremmo sempre effettuare

- Normalizzazione: procedimento lessicale, o sintattico, o morfologico che si traduce nell'esecuzione delle seguenti operazioni: tokenizzazione, stemming, lemmatizzazione

- Denormalizzazione: procedimento semantico attuato mediante alcune tecniche: con le *named entities* cerchiamo di estrarre nomi dal testo, ad esempio invece con i *semantic roles* andiamo a determinare i ruoli all'interno della frase. Tali ruoli solitamente sono "agent", "patient", "mean"(mezzo)...

Nell'ambito della semantica distribuzionale è importante il lavoro del già citato Peter Turney. Pubblicato nel 2010, cerca di mettere ordine sulle ricerche fatte fino a quel momento e fornisce una classificazione delle matrici nell'ambito della semantica distribuzionale. Secondo Turney ne esistono di tre tipi:

1. Term document matrix: su ogni riga abbiamo un documento e su ogni colonna un termine. Su questo tipo di matrici possiamo eseguire tutte le operazioni di similarità, di clusterig, di classificazione, di segmentazione e parzialmente di question answering

2. Term content matrix: generalizzano il tipo precedente di matrice e su ogni riga possiedono un contesto e su ogni colonna un termine. L'idea è che un contesto non deve essere necessariamente un documento ma può anche essere una frase, un paragrafo, una dipendenza sintattica...
   In questo caso le operazioni supportate sono la similarità, la costruzione di cluster, la classificazione delle parole, la generazione automatica di un thesaurus, la disambiguazione, l'etichettatura semantica dei ruoli...

3. Pair-pattern matrix: vera novità proposta da Turney. In queste matrici su ogni riga abbiamo coppie di parole mentre su ogni colonna abbiamo un pattern. I pattern possono essere qualcosa come relazioni. Ad esempio "X è risolto da Y", "X causa Y", "X è causato da Y" e altro ancora.
   Intanto potremmo chiederci come mai non usare triple di parole. Per due motivi: il primo è che il costo computazionale sarebbe eccessivo. Il secondo è che sarebbe anche inutile perché otterremmo matrici decisamente

troppo sparse.
Su questa matrice possiamo misurare similarità relazionali e similarità di pattern. Con questo secondo tipo di similarità andiamo fondamentalmente ad effettuare del clustering sui pattern.
Ad ogni modo l'utilizzo di questa matrice permette di cogliere una maggiore profondità semantica perché ad esempio permette di rispondere alla domanda "individua la lista di tutte le X tali che X causa il cancro".

A questo punto riprendiamo il concetto di similarità. Tale concetto è importante, tanto che spesso la semantica distribuzionale viene anche detta semantica della similarità.
Negli anni '50 Quine asserisce che la similarità è estremamente importante perché ci permette di mettere ordine mediante categorizzazione e ci fornisce degli stimoli. Infatti gli eventi futuri non saranno uguali a quelli passati ma in molti casi cercheremo delle similarità con gli eventi passati per cercare di capire come comportarci con le nuove situazioni che si presentano.
Abbiamo diverse definizioni di similarità. Intanto dobbiamo sempre fare attenzione che si tende facilmente a confondere -anche negli articoli specialistici- il termine "parola" con il termine "concetto".
Vediamo alcune definizioni di similarità

- Semantic similarity: si riferisce ai sinonimi o ai quasi sinonimi. Viene spesso utilizzata a sproposito

- Semantic relatedness: spesso chi parla di similarità semantica si riferisce invece alla semantic relatedness. Riguarda dei concetti che condividono proprietà. Il significato di questa quantità è molto generico tanto da essere spesso inutilizzabile

- Attributional similarity: è abbastanza un sinonimo della definizione precedente. Questo nome sarebbe più appropriato ma in letteratura si preferisce utilizzare il nome semantic relatedness.

- Taxonomical similarity: più facilmente misurabile perché riguarda quei concetti che condividono degli iperonimi

- Relational similarity: va a studiare la similarità tra coppie di parole

- Semantic association: va ad analizzare i concetti che co-occorrono frequentemente. È una misura diversa dalla semantic relatedness perché non prescinde dalle co-occorrenze ed è maggiormente orientata alla corpus analysis

Come avevamo detto in precedenza, uno dei problemi delle rappresentazioni matriciali è che perdiamo informazioni sull'ordine. La quantità di informazione persa da un algoritmo di information retrieval in assenza di un ordine delle parole è stimata nell'ordine del 20%.
Per porre rimedio a questa situazione sono state pensate alcune soluzioni: in certi casi facciamo uso di matrici pair pattern sensibili all'ordine delle parole.

Oppure vengono introdotti dei vettori ausiliari detti *vettori di ordinamento* volti a fornire informazioni aggiuntive sull'ordine.
Un altro problema è che spesso le matrici sono orientate al significato di singole parole. Spesso dobbiamo dunque usare delle combinazioni lineari dei vettori per cercare di lavorare sulla semantica composizionale.
Altre tecniche ancora, dette di *arricchimento matriciale* vanno ad aggiungere informazioni alla matrice stessa mediante delle risorse esterne esistenti.
Un'ultima direzione in cui la ricerca si è orientata è quella della *filaments of meaning in wordspace*. Secondo questa idea la semantica delle parole non è contenuta in spazi vettoriali di dimensione elevata, bensì in piccole porzioni di tale spazio. Dunque il significato di una parola dipende in realtà da una rappresentazione ristretta.

25 / 05 /2018

9.5 Ontology learning

Per cercare di risolvere il problema dell'apprendimento di una ontologia solitamente procediamo mediante degli strumenti automatici.
Una prima definizione di questo problema viene data da Philipp Cimiano il quale afferma che l'ontology learning assomiglia a un procedimento di reverse engineering: data una conoscenza di un certo dominio, con la sua rappresentazione e la sua scrittura, si vuole cercare di ritornare alla concettualizzazione di partenza.
Notiamo che esiste una similarità con quanto visto nel triangolo semeiotico.
Quando dobbiamo risolvere questo problema dobbiamo gestire due questioni

- La conoscenza del mondo non è codificata e dunque non è uguale per tutti. Potremmo avere visioni del mondo molto diverse fra loro
- Non tutto quello che sappiamo del dominio viene effettivamente utilizzato

Ad ogni modo l'ontology learning si differenzia da alcune altre tecniche:

- Ontology population: in questo caso, supponendo di avere già una propria ontologia a disposizione, andiamo a popolarla inserendovi delle istanze di ogni etichetta
- Ontology annotation: data una ontologia già costruita e partendo da una base documentale, andiamo a taggare il testo con delle informazioni concettuali
- Ontology enrichment: data l'ontologia e una base documentale, possiamo andare a trovare istanze degli elementi dell'ontologia e dire qualcosa di più riguardo l'ontologia stessa. Infatti nel momento in cui notiamo la che un certo termine si presenta con una frequenza elevata andiamo a ristrutturare l'ontologia sia a livello di concetti(aggiungendo nodi), sia a livello di relazioni(aggiungendo archi). Un aspetto interessante da considerare è che possiamo trovare nuovi concetti non visti in precedenza ristrutturando l'ontologia

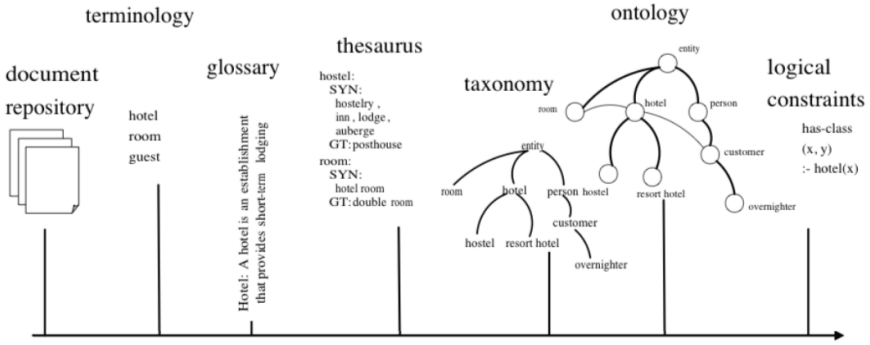


Figura 33: Possibili livelli di un'ontologia

Visto che le ontologie riguardano la formalizzazione della conoscenza, potremmo chiederci: ma quanto sono davvero formali? In Figura 33 mostriamo una serie di rappresentazioni dei dati in cui il grado di formalizzazione cresce da sinistra verso destra. Infatti la minor formalizzazione possibile è quella del testo non strutturato, a cui segue quella della terminologia in cui abbiamo una serie di termini espressi per dominio di interesse, a cui segue la rappresentazione a thesaurus che fornisce relazioni tra le parole. Ancora più formale è la tassonomia che fornisce una gerarchizzazione dei concetti. A questo punto abbiamo proprio l'ontologia che fornisce una gerarchia ben strutturata che include diversi tipi di relazioni, regole e assiomi.

Nel procedimento di ontology learning abbiamo diversi task: estrazione di termini, estrazione di sinonimi, estrazione di concetti, estrazione di relazioni, popolamento(mediante named entity recognition e infromation extraction) e infine induzione gerarchica di concetti.
Noi ci concentriamo sull'ultimo task che può essere portato a compimento mediante tre paradigmi:

- Pattern lessico sintattici: in questo caso andiamo a definire dei pattern che abbiamo trovato nel testo e che indicano una relazione semantica. Nel tempo abbiamo cominciato ad apprendere questi pattern in modo automatico partendo da un training set con migliaia di frasi che esprimono ognuna una sola relazione semantica
- Distributional hypothesis: un approccio di semantica distribuzionale in cui se prendiamo due coppie di parole tali per cui gli elementi di ogni coppia sono a distanza simile tra di loro, allora è possibile che abbiamo lo stesso tipo di relazione
- Nozione di sussunzione: insieme di tecniche che permettono di costruire delle gerarchie

Per cercare di portare a compimento questi task abbiamo di fatto tre metodologie:

1. Natural language processing: la prima cosa da fare è di strutturare i PoS che sono solitamente indicativi per la semantica.
   Dopo dobbiamo eseguire delle operazioni di preprocessing come la tokenizzazione, lo stemming/lemmatizzazione, il PoS tagging, named entity recognition...
   Dopo dobbiamo eseguire una analisi sintattica con un sistema a regole basato su alberi di parsing.
   A seguito dell'analisi sintattica possiamo fare una analisi semantica mediante vector space model, distribuzioni di probabilità e tecniche di rilevanza dei termini come tf, tf-idf...
   Infine possiamo usare risorse linguistiche esterne a supporto come WordNet, BabelNet, FrameNet...

2. Formal concept analysis: in questo caso gli elementi fondamentali sono tre. Abbiamo gli *oggetti* che sono l'equivalente dei concetti, gli *attributi* che corrispondono agli oggetti come le feature corrispondevano ai concetti, e infine l'*incidenza* che rappresenta il fatto che un certo oggetto possieda o meno un certo attributo. Questa relazione tra oggetti e attributi viene espressa mediante la *matrice di context analysis*. Tale matrice presenta in ogni riga un oggetto, e in ogni colonna un attributo.
   Sugli elementi di questa matrice possiamo applicare due operatori

   - $\uparrow$(up): l'operatore up viene applicato sulle righe della matrice(dunque agli oggetti) e ci fornisce gli attributi che quell'oggetto possiede
   - $\downarrow$(down): l'operatore down viene applicato sulle colonne(dunque sugli attributi) e ci dice quali oggetti possiedono quell'attributo

   Cerchiamo di comprendere meglio il funzionamento della matrice e degli operatori con l'esempio che segue

   Esempio 1 : supponiamo di avere la matrice mostrata in Tabella 2

	A1	A2	A3	A4	A5
O1	x	x			x
O2			x	x	
O3		x			
O4		x		x	x
O5	x				x
O6		x			x

Tabella 2: Tabella FCA relativa all'esempio 1

   In questo caso $\uparrow (O_4) = \{A_2, A_4, A_5\}$.
   Invece $\downarrow (A_1, A_5) = \{O_1, O_5\}$
   In base a questa matrice è sempre possibile costruire un reticolo in cui consideriamo da prima l'insieme degli elementi che possiedono nessuno attributo, poi gli insiemi degli elementi che possiedono uno

soltanto dei degli attributi, poi due attributi...
In base al reticolo possiamo poi strutturare una gerarchi di oggetti più o meno generali di altri. Infatti considereremo più generali tutti quegli oggetti con meno attributi.

3. Machine learning: volendo utilizzare le tecniche di machine learning possiamo adottare approcci sia supervisionati, come i classificatori bayesiani, gli alberi di decisione, le SVM, le reti neurali...
   Oppure alternativamente possiamo anche usare dei metodi non supervisionati come il clusterig che può essere naturalmente divisivo o agglomerativo

Un discorso a parte merita l'ambito della *open information extraction*. In questo caso abbiamo un insieme di tecniche che nascono dalla necessità di estrarre efficientemente grandi quantità di informazioni da corpora di grandi dimensioni. Solitamente lo scopo è di estrarre delle triple della forma
$(arg1, verbal\ phrase, arg2)$
Vincolarsi a una rappresentazione tanto semplice rischia di far ottenere dati rumorosi che però è un rischio da correre nel caso in cui si richieda una elevata efficienza per la risoluzione efficiente di un problema complesso come il question asnwering su larga scala.
Un vantaggio dell'approccio OIE è che diventa facile apporre dei vincoli logici per richiedere dati più specifici, ad esempio richiedere che la data di nascita di un personaggio che stiamo cercando sia successiva ad un certo anno.
Solitamente il procedimento segue due fasi: una prima fase relativa al PoS tagging, una seconda fase relativa all'estrazione degli elementi delle triple che può essere eseguito in base a WSD, dipendenze sintattiche e altro ancora. Infine esiste una terza fase di pulizia.
Le problematiche nell'ambito OIE sono molteplici: intanto non esiste un approccio rigoroso e unico che potrebbe portare a tralasciare talune o talaltre parti. Esistono anche molti metodi di estrazione diversi fra loro che sono difficili da comparare. Infatti non esiste una forma di gold standard a cui affidarsi. Infine è stato mostrato che le triple non funzionano molto bene in contesti reali.

28 / 05 /2018

10 NLP e social media

Per capire come il natural language processing interagisca con i social media, dobbiamo intanto capire con che tipi di dato abbiamo a che fare.
Visto che ci concentriamo su twitter, analizziamo testi 280 caratteri spesso incompleti e dipendenti dal contesto. Sono spesso non corretti dal punto di vista grammaticale, fanno uso di abbreviazioni, link, emoticon, hashtag e altri elementi. L'essere umano è comunque in grado di cogliere il significato ma un parser sintattico no!
In questo ambito dunque, i task da portare a termine sono diversi: dobbiamo

individuare delle strutture dedicate che permettano di cogliere al meglio il significato dei tweets. Dobbiamo anche capire cosa farne di tutti quegli elementi che in realtà sono notizie, tale ambito viene detto news analytics. Abbiamo poi i procedimenti di opinion mining e sentiment analysis che anche se in certi contesti vengono usati come sinonimi in realtà non lo sono. Infatti solitamente l'output di un processo di opinion mining è tendenzialmente un insieme di opinioni più o meno oggettive. Invece nel caso della sentiment analysis abbiamo a che fare con output classificati come positivi/negativi. Abbiamo anche l'operazione di scraping che va a recuperare le informazioni da twitter(o da un qualunque altro sito internet) per effettuarne delle analisi.
Ad oggi i principali ambiti di ricerca riguardano

- Scraping: determinare il valore commerciale e sociale dei dati
- Pulizia dei dati: come ripulire i dati di twitter che come abbiamo visto presentano diversi problemi
- Protezione dei dati
- Processamento dell'informazione: come trattare dati multilingue, errori, il fatto che ogni social network possieda un proprio bacino di utenti diverso da quello di un altro social per età, grado medio di istruzione...
- Visualizzazione dei dati

L'analisi approfondita di questi dati è importante perché i social media possono essere visti come dei *sensori sociali*. Questo può essere fatto andando a comprendere cosa sia un *termine emergente*. In base al termine possiamo costruirgli intorno un *topic* ovvero un insieme di parole minimale che esplicita e disambigua l'argomento della discussione.
Dobbiamo intanto marcare una differenza importante tra termini emergenti e termini caldi. I primi sono quelli che in un breve periodo di tempo presentano uno scostamento forte dal numero di volte che vengono utilizzati mediamente. I termini caldi invece, pur essendo utilizzati tanto, lo sono sempre e dunque non sono così interessanti. Ad esempio un picco di utilizzi della parola earthquake indica che probabilmente si è verificato un terremoto. Invece la parola computer pur essendo utilizzata mediamente molto di più non indica alcun evento particolare proprio perché è usata così spesso. Inoltre anche se un termine presenta effettivamente un picco, dobbiamo considerare se quel picco si presenta periodicamente allora non è emergente. Ad esempio la parola morning presenta effettivamente un picco nelle ore del mattino. Ma questo picco, presentandosi tutti i giorni ci permette di capire che tale parola non sia emergente!
Dunque per ogni parola in un tweet $\vec{tw}_j$ avremo una serie di pesi associati $\vec{tw}_j = \{w_{j,1}, w_{j,2}, \ldots, w_{j,v}\}$, dove:

$$w_{j,x} = 0.5 + 0.5 \cdot \frac{tf_{j,x}}{tf_j^{max}} \tag{11}$$

Un altro aspetto che dobbiamo considerare in twitter è quanto impatto hanno le informazioni pubblicate da un utente o da un altro. È chiaro che i tweets di Obama avranno una maggiore risonanza di quelli di Di Caro, dunque dobbiamo tenerne di conto andando a fornire un certo valore di *autorità* agli autori in base al numero di connessioni che hanno con gli altri.
Per comprendere complessivamente l'importanza di una parola, possiamo usare la seguente metafora: "Con nutrimento abbondante (tweet che la riportano), il suo ciclo di vita è prolungato. Altrimenti, la parola muore in quanto il nutrimento diventa insufficiente".
Dunque calcoliamo il *nutrimento* come segue:

$$nutr_k^t = \sum_{tw_j \in TW_k^t} w_{k,j} \cdot auth(user(tw_j)) \tag{12}$$

dunque sfruttiamo anche il concetto di autorità cui accennavamo prima.
Una volta che abbiamo individuato i termini emergenti possiamo andare a determinare i topic correlati andando ad analizzare l'intervallo in cui i termini sono stati riportati sulla rete, e analizziamo la relazione semantica esistente tra tutte le keyword. Dunque ci interessiamo alla co-occorrenza fra i termini. Andiamo anche a calcolare la frequenza dei singoli termini.

Una applicazione di quanto detto finora sta nell'estrazione automatica del contenuto affettivo nei documenti testuali. Al momento la maggior parte dei sistemi riesce a catturare soltanto una differenza a grana grossa, come sentimenti positivi/negativi. È difficile determinare quale tipo di emozione sia associato ad un tweet(tristezza, rabbia...).
Al livello semplice positivo/negativo non abbiamo molta informazione e dunque solitamente utilizziamo delle tecniche dette di *aspect based analysis* in cui cerchiamo di catturare delle relazioni precise tra ciò che viene valutato e la connotazione emotiva associata. Questo viene fatto utilizzando degli oggetti da valutare che siano inseriti all'interno di una ontologia fornita e popolata mediante una tecnica di ontology population.
Per effettuare queste operazioni possiamo utilizzare diversi strumenti:

- Opinion mining ML: dove ML sta per markup language. Questo strumento ci permette di specificare l'ontologia e l'attribuzione dei sentimenti. Inoltre mediante dei TAGS possiamo distinguere maggiormente tra elementi soggettivi e oggettivi, domande, suggerimenti...

- SentiVis: strumento che mediante una analisi della struttura sintattica(parsing a dipendenze) va a propagare i valori di sentiment all'interno della struttura sintattica. Dunque se un nodo interno dell'albero presenta un certo valore (ad esempio +0.7 che indica un sentimento positivo), tale valore può essere propagato alle foglie comportando una differente visualizzazione dei risultati