# Modelli A Stati Finiti Per Il Linguaggio Naturale

A cura di De Pascalis Roberto

## Introduzione

- Agli inizi del Natural Language processing si credeva che i modelli probabilistici a stati finiti potessero descrivere il sistema linguistico umano.
- Noam Chomsky dimostrò nel suo *Syntactic Structures* che tali modelli non descrivevano correttamente tale sistema.
- Tuttavia esistono caratteristiche del linguaggio naturale che possono essere modellate attraverso modelli probabilistici a stati finiti.

## Grammatiche Generative

Una grammatica generativa è :

- Un sistema combinatorio discreto
- Un codice per tradurre l'ordine delle parole in combinazioni di pensieri

Infatti l'insieme delle parole viene permutato, combinato e prodotto per creare un insieme particolare (l'enunciato)

Esempio : "Cane morde uomo", "Uomo morde cane" sono diverse combinazioni con diverso significato

## Le catene di Markov

- Si pensò di descrivere gli enunciati come catene di parole.
- Allo scopo di rendere il modello più realistico venne introdotto il concetto di probabilità, per descrivere le transizioni tra sequenze di parole.
- Si usarono, quindi, le catene di Markov. Fu elaborata un' enorme quantità di testi per stimare delle tavole di "probabilità di transizione".

## “English is not a finite state language”

Chomsky individuò i tre problemi di tale struttura :

- un enunciato non è una sequenza di parole tenute insieme da regole di probabilità di transizione.
- Le sequenze costruite attraverso tali tavole di probabilità forniscono enunciati lontani dall’essere ben formati.
- Nomi, verbi, aggettivi, ecc. non vengono collegati in una lunga catena, ma inseriti in uno schema precostituito che assegna ad ogni parola un posto preciso.

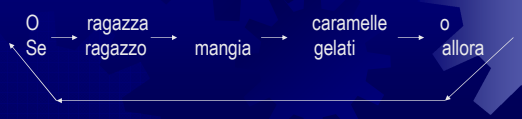
## “English is not a finite state language”

- “Incolori idee verdi dormono furiosamente”.
- Non c’è nessuna relazione tra grammaticalità e frequenza di una sequenza di parole.
- Quando si studia una lingua si impara come mettere in ordine le parole, senza, però, registrare quale di esse ne segua un’altra.

## “English is not a finite state language”

Alcuni enunciati non possono essere prodotti da una catena di parole.

Esempio
“O la ragazza mangia gelati, o mangia caramelle”
“Se la ragazza mangia gelati
allora il ragazzo mangia caramelle”




## “English is not a finite state language”

Un sistema a catene di parole soffre di amnesia: quando viene raggiunto il paradigma *o* / *allora*, non ha modo di ricordare se all’inizio c’era *o* / *se*.

Un linguaggio naturale può avere enunciati del tipo:

“Se…o..o..allora..”

E’ impossibile costruire una catena di Markov che tenga traccia di un numero rilevante di queste dipendenze.

## “English is not a finite state language”

Chomsky dedusse che il linguaggio naturale non può essere descritto né da un automa a stati finiti e né da una grammatica context free.

Questo portò all’abbandono degli automi in quanto non erano un modello soddisfacente del sistema linguistico umano.

## Nuove strade per gli automi a stati finiti

Negli ultimi anni è stata rivelata la loro applicabilità nel campo della morfologia e della fonetica:

- “Morphotactics”
  ossia studio della sintassi dei morfemi
- Alternanza morfologica
  ossia studio dei cambiamenti dei morfemi in base al contesto
- Alternanza fonologica

## Nuove strade per gli automi a stati finiti

- “Morphotactics” ossia costruzione delle parole
  esempio
  I morfemi non possono combinarsi liberamente :
  *piti-less-ness* è corretta
  *piti-ness-less* non lo è
- Alternanza morfologica
  *die* si trasforma in *dy* nel contesto di *dying*

## Approccio alla morfologia con gli automi a stati finiti

La morfologia è un ramo della linguistica che studia la formazione delle parole.

Ovvero la distinzione tra forme superficiali e le loro analisi, chiamate lemmi.

Esempio:
il lemma per la forma superficiale inglese di “bigger”, è big+adj+comp, che sta ad indicare che “bigger” è il comparativo dell’aggettivo “big”.

## Approccio alla morfologia con gli automi a stati finiti

Attraverso tale approccio, si cercano di individuare delle relazioni tra il lemma e la sua forma superficiale.
Tali relazioni sono chiamate regolari.

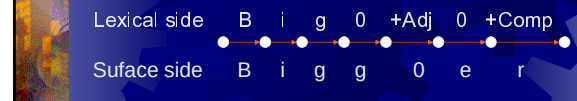

## Approccio alla morfologia con gli automi a stati finiti

trasduttore : in questo caso è un automa a stati finiti che fornisce la forma lessicale.

forma lessicale : Il mappaggio tra forma superficiale ed il suo lemma.

## Approccio alla morfologia con gli automi a stati finiti

Esempio ( 0 è la stringa vuota )




Il trasduttore trasforma *bigger* in *big+Adj+Comp, ossia Bigger* è il comparativo dell'aggettivo *big*.

Più compatto : b i g 0:g +Adj:0 0:e +Comp:r

## Approccio alla morfologia con gli automi a stati finiti

I trasduttori a stati finiti sono bi-direzionali :

una forma superficiale produce un lemma e viceversa.

Lo stesso transduttore può essere utilizzato
- sia per analisi
  ( surface input, o upward application)
- sia per generazione
  ( lexical input, o downward application)

## Approccio alla morfologia con gli automi a stati finiti

- Le espressioni regolari coprono un sottoinsieme del linguaggio naturale.
- E' vantaggiosa la descrizione di sottolinguaggi con espressioni regolari. (nomi, indirizzi, date, ecc.)
- Le espressioni regolari possono essere compilate in automi a stati finiti.
- Gli automi possono esse minimizzati, sequenzializzati e compressi per ridurne grandezza.

## Espressioni regolari

Le espressioni regolari denotano :
- Insiemi di stringhe.
- Insiemi di coppie di stringhe.

Terminologia:
- Un linguaggio è un insieme di stringhe.
- Una relazione è un insieme di coppie di stringhe.

I termini "linguaggio regolare" e "relazione regolare" si riferiscono ad insiemi che possono essere descritti da espressioni regolari.

## Espressioni regolari

Espressioni atomiche unarie:

- A,B.. Variabili
- a,b… Simboli
- 0 Stringa vuota
- ? Simbolo jolly

Espressioni atomiche binarie:

- a:b denota la relazione tra le stringhe a, b
- a:a relazione di identità, denotabile con a

## Espressioni regolari

I linguaggi sono rappresentati da semplici automi.

Le relazioni sono rappresentate da trasduttori.

In a:b a denota il *Linguaggio inferiore*
b denota il *Linguaggio superiore*

Un trasduttore che realizza una relazione regolare può essere applicato per mappare una stringa nel linguaggio superiore nella corrispondente nel linguaggio inferiore.

## Espressioni regolari

Espressioni composte: (operatori)

- [] Raggruppamento espressioni
- A|B Unione
- A B Concatenazione
- (A) Opzionalità ( eq. [A | 0] )
- A+ Una o più concatenazioni di A
- A* Stella di Kleene (eq. (A*) )

Espressioni e linguaggi regolari sono chiusi per unione e concatenazione.

## Espressioni regolari

- ~A Complemento
- \A Complemento di termine
- A & B Intersezione
- A – B Sottrazione

Linguaggi regolari sono chiusi rispetto ~, & e –.
Relazioni regolari non lo sono.

## Espressioni regolari

I seguenti operatori sono utilizzati per costruire relazioni regolari:

- A .x. B Prodotto Cartesiano (tra espressioni reg.)
- A .o. B Composizione (tra relazioni regolari)

Nota : [a .x. b] = a:b
La composizione fra due relazioni regolari è chiusa.
es. x:y .o. y:z = x:z

## Operatori per le espressioni regolari

Aggiungiamo un livello di astrazione alla sintassi delle espressioni regolari: (A,B,L,R sono ling. reg.)

- Operatore di contenimento
  $A =$_{def}$ [?* A ?*]
- Operatore di restrizione
  A=>L _ R =$_{def}$ [ ~[ [~ [?*L] A ?*] | [?* A ~[R ?*] ] ?* ]

## Operatori per le espressioni regolari

Esempi :

- Contenimento
  $[a|b] denota tutte le stringhe che contengono una a od una b da qualche parte.
- Restrizione
  a=>b_c contiene tutte le stringhe dove a è incluso nella stringa b…c. es. bac è corretta, ab non lo è.
  L'espressione a:0=>b_c non è corretta.
  a=>_.#. Permette a solo alla fine di una stringa.

## Operatori per le espressioni regolari

- Operatore di rimpiazzo (senza vincoli)
  A ->B $=_{def}$ [ [~$[A - 0] [A .x. B] ]* ~$[A - 0 ] ]
- Operatore di marcatura (senza vincoli)
  A->B…C
- Operatore di rimpiazzo (left to right longest match)
  A @-> B
- Operatore di marcatura (left to right longest match)
  A @-> B…C

## Operatori per le espressioni regolari

- Gli operatori di contenimento e restrizione restituiscono un insieme di stringhe.
- Gli operatori di rimpiazzo e marcatura restituiscono delle relazioni regolari.
- Rimpiazzo e marcatura sono ambigue in mancanza di restrizioni.

## Operatori per le espressioni regolari

Esempi :

- Rimpiazzo ( senza vincoli )
  A->B denota una relazione regolare

[ab | b | ba | aba] -> x è l'insieme delle relazioni :

| a b a | a b a | a b a | a b a |
|---|---|---|---|
| x a | a x a | a x | x |

## Operatori per le espressioni regolari

- Marcatura ( senza vincoli )

  a|aa -> “[“…”]”

  mappa la stringa del linguaggio superiore aaa in tre differenti stringhe del linguaggio inferiore :

  a a a  a a a  a a a
  - - -  - ----  ---- -
  [a][a][a]  [a] [a a]  [a a] [a]

## Operatori per le espressioni regolari

- Marcatura ( senza vincoli )

  a|e|i|o|u -> “[“…”]” , b|c|d…|z->”(“…”)”

  Posso farla anche in parallelo

  g e l a t o
  (g)[e](l)[a](t)[o]

  Ha marcato le vocali e le consonanti.

## Operatori per le espressioni regolari

- Rimpiazzo ( left to right longest match )

  [ab | b | ba | aba] @-> x  restituisce l’unica relazione:

  a b a
  ---
  x

  Sceglie il più grande pattern a partire da sinistra.

## Operatori per le espressioni regolari

- Marcatura ( left to right longest match )

  Vogliamo marcare i sintagmi nominali del tipo (d) a* n+, dove d è il determinante, a un aggettivo, ed n un sostantivo.

  (d) a* n+ @->”[“…”]”

  mappa dannvann in [dann]v[ann]. dann e ann sono i più grandi patterns che corrispondono alla parte sinistra di @->.

## Il sottolinguaggio delle date

Un automa a stati finiti non è in grado di descrivere il linguaggio naturale, ma è possibile farlo per sottoinsiemi di esso.

Da un espressione regolare che definisce la sintassi delle date, possiamo derivare un trasduttore a stati finiti che le riconosca e le marchi.

Il trasduttore contiene il parser.

## Il sottolinguaggio delle date

Le seguenti sono istanze valide del linguaggio delle date :

Domenica
11 Agosto
Domenica, 11 Agosto
11 Agosto, 1996
Domenica, 11 Agosto, 1996

## Il sottolinguaggio delle date

1to9 = 1 | 2 |…| 9
0to9 = “0” | 1to9
Giorno = Lunedì |…| Domenica
Mese = Gennaio |…| Dicembre
Data = 1to9 | [1 | 2] 0to9 | 3 [“0” | 1]
Anno = 1to9 (0to9 (0to9 (0to9)))
Date = Giorno | (Giorno “, “) Data “ “ Mese (“, “ Anno)

Tutte le date da 1 Gennaio , 1 a 31 Dicembre, 9999

## Il sottolinguaggio delle date

Date @-> “[“…”]”

↓ Compilazione

trasduttore con 23 stati e 321 archi.

Esempio

Oggi è [Martedì, 11 Aprile 1976]. Ieri era [Lunedì]. Tra un mese sarà l’[11 Maggio].

## Il sottolinguaggio delle date

Estendiamo la grammatica per esprimere gli anni bisestili :

Pari = “0” | 2 |…| 8
Dispari = 1 | 3 |…| 9
N = 1to9 0to9*
Div4 = [((N) Pari) [“0” | 4 | 8]] | [(N) Dispari [2 | 6]]
Bisestile = Div4 – [[N – Div4] “0” “0”]

## Il sottolinguaggio delle date

- Introduciamo delle restrizioni e dei contenimenti:

LeapDays =
Febbraio “ “ 2 9 “, “ => _ Bisestile .#.
MaxDaysInAMonth =
~$[ Febbraio “ “ 3 “0” |
[ Febbraio | Aprile | Giugno | Settembre | Novembre ] “ “ 3 1]
WeekDayDates =
Modella l’associazione tra Data e Giorno (es. 19 Giugno 2002 è Mercoledì)

## Il sottolinguaggio delle date

Per completare la definizione del linguaggio delle “date valide”, introduciamo l’ultimo vincolo :

DateValide =
Date & MaxDaysInMonth & LeapDays & WeekDayDates

Manca ora l’espressione che verrà compilata :

DateValide @-> “[VD “ … “]
Date - DateValide @-> “[ID “ … “]

## Il sottolinguaggio delle date

Il trasduttore che ne viene fuori riconosce e marca le date valide (VD) e quelle non valide (ID).

Esempio

Oggi è [VD Mercoledì, 19 Giugno, 2002], ma non è [ID Mercoledì, 20 Giugno, 2002].

Nota :l’operatore @-> sceglie sempre la stringa più grande, anche se essa contiene sottostringhe valide.