Special Topics in Computer Science

NLP in a Nutshell

CS492B Spring Semester 2009

Jong C. Park
Computer Science Department
Korea Advanced Institute of Science and Technology

PART-OF-SPEECH TAGGING USING STOCHASTIC TECHNIQUES

The Noisy Channel Model

- Presentation
  - Assumptions
    - Each word is known.
    - Each word has a finite set of possible tags.
  - When a word has more than one possible tag, statistical methods enable us to determine the optimal sequence of part-of-speech tags *T* = *t*1, *t*2, *t*3, … , *tn*, given a sequence of words *W* = *w*1, *w*2, *w*3, … , *wn*.

The Noisy Channel Model

The Noisy Channel Model

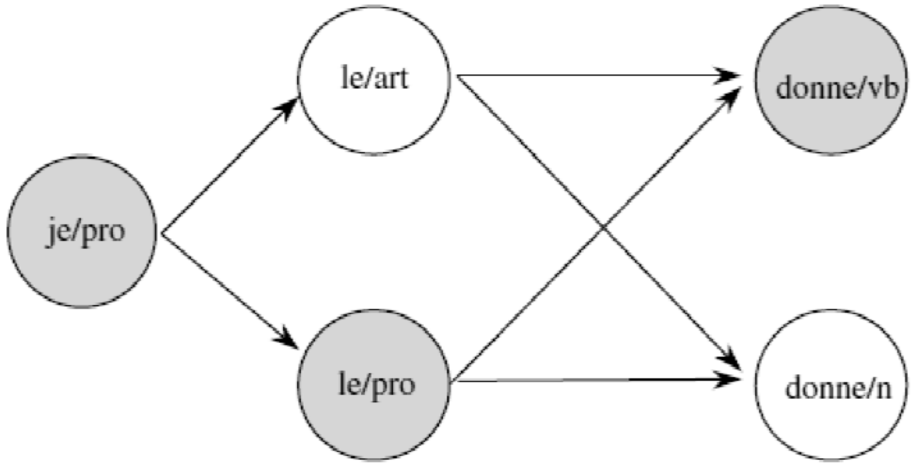
- The *N*-gram Approximation
  - Statistics on a sequence of any length are impossible to obtain.
  - We make approximations on *P(T)* and *P(W|T)* to make the estimation tractable.
  - A product of trigrams usually approximates the complete part-of-speech sequence.

The Noisy Channel Model

The Noisy Channel Model

- Tagging a Sentence
  - Example: *Je le donne* 'I give it'

Fig. 7.1. Possible sequences of part-of-speech tags, where *pro denotes a pronoun, art an article, n* a noun, and *vb a* verb*.*



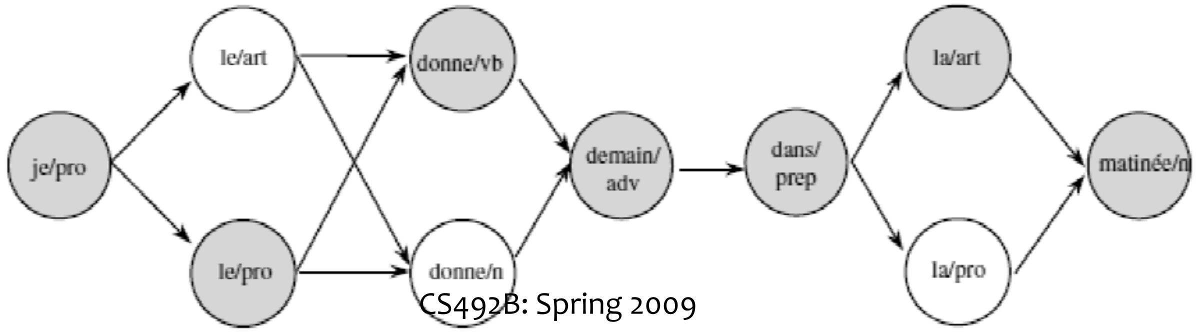
The Noisy Channel Model

The Noisy Channel Model

- The Viterbi Algorithm: An Intuitive Presentation
  - The noisy channel model is not efficient.
    - speed & memory
    - why?
      - – maintains nonoptimal paths for all the intermediate nodes

The Noisy Channel Model

Fig. 7.2. The search space, where *adv denotes an adverb and prep a preposition; the other tags* are as given in Fig. 7.1.



CS492B: Spring 2009

Remaining Issues

- Markov Models
  - Markov Chains
  - Hidden Markov Models
  - Three Fundamental Algorithms to Solve Problems with HMMs
  - The Forward Procedure
  - Viterbi Algorithm
  - The Backward Procedure
  - The Forward-Backward Algorithm

Remaining Issues

- Tagging with Decision Trees
- Unknown Words
- An Application of the Noisy Channel Model: Spell Checking

Remaining Issues

- A Second Application: Language Models for Machine Translation
  - Parallel Corpora
  - Alignment
  - Translation

Table 7.7. Parallel texts from the Swiss federal law on milk transportation.

German	French	Italian
Art. 35 Milchtransport	**Art. 35 Transport du lait**	**Art. 35 Trasporto del latte**
1 Die Milch ist schonend und hygienisch in den Verarbeitungsbetrieb zu transportieren. Das Transportfahrzeug ist stets sauber zu halten. Zusammen mit der Milch dürfen keine Tiere und milchfremde Gegenstände transportiert werden, welche die Qualität der Milch beeinträchtigen können.	1 Le lait doit être transporté jusqu’à l’entreprise de transformation avec ménagement et conformément aux normes d’hygiène. Le véhicule de transport doit être toujours propre. Il ne doit transporter avec le lait aucun animal ou objet susceptible d’en altérer la qualité.	1 Il latte va trasportato verso l’azienda di trasformazione in modo accurato e igienico. Il veicolo adibito al trasporto va mantenuto pulito. Con il latte non possono essere trasportati animali e oggetti estranei, che potrebbero pregiudicarne la qualità.
2 Wird Milch ausserhalb des Hofes zum Abtransport bereitgestellt, so ist sie zu beaufsichtigen.	2 Si le lait destiné à être transporté est déposé hors de la ferme, il doit être placé sous surveillance.	2 Se viene collocato fuori dall’azienda in vista del trasporto, il latte deve essere sorvegliato.
3 Milchpipelines sind nach den Anweisungen des Herstellers zu reinigen und zu unterhalten.	3 Les lactoducs des exploitations d’estivage doivent être nettoyés et entretenus conformément aux instructions du fabricant.	3 I lattodotti vanno puliti e sottoposti a manutenzione secondo le indicazioni del fabbricante.

Fig. 7.10. Alignment. After Brown et al. (1993).

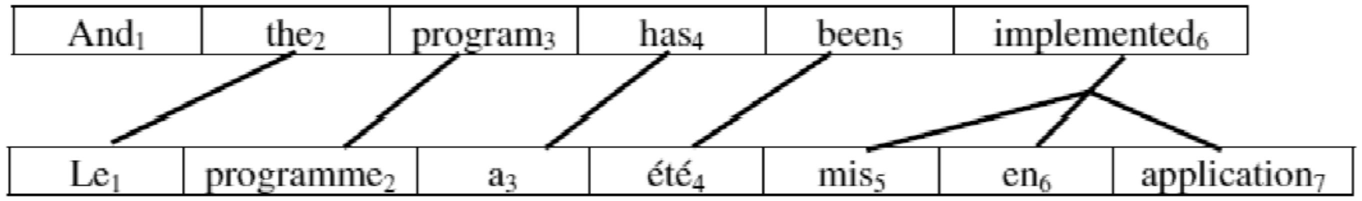
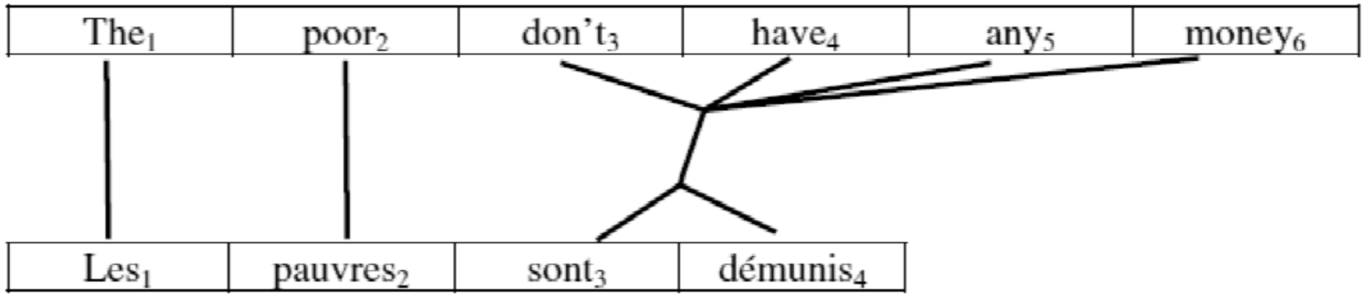


Fig. 7.11. A general alignment. After Brown et al. (1993).



INTRODUCTION TO PROLOG

LISTS, OPERATORS AND ARITHMETIC

Lists, Operators and Arithmetic

- Representation of lists
- Some operations on lists
- Operator notation
- Arithmetic

Representation of lists

- Note
  - All structured objects in Prolog are trees.
- Example lists
  - [anna, tennis, tom, skiing]
  - .(anna, .(tennis, .(tom, .(skiing, [ ]))))

```
?- List1 = [a,b,c], List2 = .(a, .(b, .(c, [ ]))).
List1 = [a,b,c], List2 = [a,b,c].

[a,b,c] = [a|[b,c]] = [a,b|[c]] = [a,b,c|[ ]]
```

Representation of lists

- A list is a data structure that is either empty or consists of two parts: a head and a tail.
  - The tail itself has to be a list.
- Lists are handled in Prolog as a special case of binary trees.

Some operations on lists

- Membership
- Concatenation
- Adding an item
- Deleting an item
- Sublist
- Permutations
- Problems

Some operations on lists

- MEMBERSHIP
  - The membership relation

    member(X,L)
  - Intended behavior

    ```
    ?- member(b,[a,b,c]).
    yes
    ?- member(b,[a,[b,c]]).
    no
    ?- member([b,c],[a,[b,c]]).
    yes
    ```

Some operations on lists

- Observation
  - X is a member of L if either:
    - X is the head of L, or
    - X is a member of the tail of L.
- Sample program

```
member(X,[X|Tail]).
member(X,[Head|Tail]) :-
          member(X,Tail).
```

Some operations on lists

- CONCATENATION
  - The concatenation relation

    conc(L1,L2,L3)
  - Intended behavior

```
?- conc([a,b],[c,d],[a,b,c,d]).
yes
?- conc([a,b],[c,d],[a,b,a,c,d]).
no
```

Some operations on lists

- Observation
  - If the first argument is the empty list then the second and the third arguments must be the same list.

```
conc([ ], L, L).
```

  - Otherwise the first argument has a head and a tail and must look like [X|L1].

```
conc([X|L1],L2,[X|L3]) :-
    conc(L1,L2,L3).
```

Some operations on lists

- Decomposition
  - We can use the conc program to decompose a given list into two lists.

```
?- conc(L1,L2,[a,b,c]).
L1 = [ ]
L2 = [a,b,c];
L1 = [a]
L2 = [b,c];
L1 = [a,b]
L2 = [c];
L1 = [a,b,c]
L2 = [ ]
```

Some operations on lists

- Pattern matching
  - We can use the program to look for a certain pattern in a list.

```
?- conc(Before,[may|After],
    [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec]).
Before = [jan,feb,mar,apr]
After = [jun,jul,aug,sep,oct,nov,dec]
?- conc(_,[Month1,may,Month2|_],
        [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec]).
Month1 = apr
Month2 = jun
```

Some operations on lists

- What are these for?

```
?- L1 = [a,b,z,z,c,z,z,z,d,e],
     conc(L2,[z,z,z|_],L1).
```

```
member(X,L) :-
     conc(L1,[X|L2],L).
```

Some operations on lists

- ADDING AN ITEM
  - Very simple!

```
add(X,L,[X|L]).
```

Some operations on lists

- DELETION
  - The deletion relation

```
del(X,L,L1)
```

  - Sample program

```
del(X,[X|Tail],Tail).
del(X,[Y|Tail],[Y|Tail1]) :-
  del(X,Tail,Tail1).
```

Some operations on lists

- The deletion program is non-deterministic.

```
?- del(a,[a,b,a,a],L).
L = [b,a,a];
L = [a,b,a];
L = [a,b,a];
no
```

Some operations on lists

- The deletion program can be used in the inverse direction.

```
?- del(a,L,[1,2,3]).
L = [a,1,2,3];
L = [1,a,2,3];
L = [1,2,a,3];
L = [1,2,3,a];
no
```

- Example uses of del(X,L,L1):

```
insert(X, List, BiggerList) :-
   del(X, BiggerList, List).
member2(X, List) :-
   del(X, List, _).
```

Some operations on lists

- SUBLIST
  - Intended behavior

```
?- sublist([c,d,e],[a,b,c,d,e,f]).
yes
?- sublist([c,e],[a,b,c,d,e,f]).
no
```

  - Sample program

```
sublist(S,L) :-
   conc(L1,L2,L),
   conc(S,L3,L2).
```

Some operations on lists

- The sublist relation can be used to find all sublists of a given list.

```
?- sublist(S,[a,b,c]).
S = [ ];
S = [a];
S = [a,b];
S = [a,b,c];
S = [ ];
...
```

Some operations on lists

- PERMUTATIONS
  - Intended behavior

```
?- permutation([a,b,c],P).
P = [a,b,c];
P = [a,c,b];
P = [b,a,c];
...
```

Some operations on lists

- Observation
  - If the first list is empty then the second list must also be empty.
  - If the first list is not empty then it has the form [X|L], and a permutation of such a list can be constructed by first permuting L for L1 and then inserting X at any position into L1.

Some operations on lists

- Sample program

```
permutation([ ],[ ]).
permutation([X|L],P) :-
   permutation(L,L1),
   insert(X,L1,P).


permutation2([ ],[ ]).
permutation2(L,[X|P]) :-
   del(X,L,L1),
   permutation2(L1,P).
```

Problems

- Complete the following programs

```
last(Item,List) :-
   conc(_,[Item],List).

reverse([ ], [ ]).
reverse([First|Rest],Reversed) :-
   reverse(Rest,ReversedRest),
   conc(ReversedRest,[First],Reversed).
```

There are many other ways to reverse a given list.

Operator notation

- Motivation
  - Example
    - 2*a+b*c
    - Is it +(*(2,a),*(b,c)) or *(*(2,+(a,b)),c)?
- Precedence
  - The operator with the highest precedence is understood as the principal functor of the term.
  - Which is higher: + or *?

Operator notation

- A programmer can define his or her own operators.

```
peter has information.
floor supports table.

has(peter,information).
supports(floor,table).
```

Operator notation

- Directives
  - :- op(600,xfx,has).
    - 'has' is defined as an operator.
    - its precedence is 600.
    - its type is 'xfx', a kind of infix operator.
  - The operator names are atoms.
  - The range is fixed, e.g. between 1 and 1200.

Operator notation

- Operator types
  - infix operators of three types
    - xfx, xfy, yfx
  - prefix operators of two types
    - fx, fy
  - postfix operators of two types
    - xf, yf

Operator notation

- Precedence of argument
  - If an argument is enclosed in parentheses or it is an unstructured object then its precedence is 0.
  - If an argument is a structure then its precedence is equal to the precedence of its principal functor.

Operator notation

- 'x' and 'y'
  - 'x' represents an argument whose precedence must be strictly lower than that of the operator.
  - 'y' represents an argument whose precedence is lower or equal to that of the operator.
- Example
  - What is the type of '-'?
  - Is it a - b - c as (a - b) - c, or as a - (b - c)?
    - yfx if (a-b)-c only
    - yfy if both
    - xfy if a-(b-c)
    - xfx if neither

Operator notation

- Another example
  - What is the type of 'not'?
    - Is not not p allowed?
    - fy if yes
    - fx if no
- Predefined operators
  - Figure 3.8
- Example
  - ~(A&B) <===> ~A v ~B

Arithmetic

- A subset of the predefined operators can be used for basic arithmetic operations.
  - +, -, *, /, **, //, mod

```
?- X = 1 + 2.
X = 1 + 2.
?- X is 1 + 2.
X = 3.
```

Arithmetic

- Arithmetic is also used for comparison.

```
?- 277*37 > 10000.
yes
?- born(Name,Year),
    Year >= 1980,
    Year =< 1990.
```

value comparison

>, <, >=, =<, =:=, =\=

Arithmetic

- Sample interaction

```
?- 1 + 2 =:= 2 + 1.
yes
?- 1 + 2 = 2 + 1.
no
?- 1 + A = B + 2.
A = 2
B = 1
```

Arithmetic

- Example use of arithmetic operations
  - Greatest Common Divisor (GCD)
    - If X and Y are equal then D is equal to X.
    - If X<Y then D is equal to the gcd of X and the difference Y-X.
    - If Y<X then do the same as above with X and Y interchanged.

```
gcd(X,X,X).
gcd(X,Y,D) :- X<Y, Y1 is Y-X, gcd(X,Y1,D).
gcd(X,Y,D) :- Y<X, gcd(Y,X,D).
```

Arithmetic

- Example use of arithmetic operations
  - Counting items in a list: length(List,N)
    - If the list is empty then its length is 0.
    - If the list is not empty then List = [Head|Tail]; so its length is equal to 1 plus the length of Tail.

```
length([ ], 0).
length([_|Tail],N) :-
        length(Tail,N1),
        N is 1+N1.
```