CS4442/9542b: Artificial Intelligence II
Prof. Olga Veksler

Lecture 10

NLP: Part of Speech Tagging (POS)

Many slides from: L. Kosseim, M. Hearst, K. McCoy, Yair Halevi

Outline

- What is POS and POS tagging
- Why we need POS tagging
- Different Approaches to POS
  1. rule-based tagging
  2. statistical tagging

What is a Part of Speech ?

- Words that somehow ‘behave’ alike:
  - Appear in similar contexts
  - Perform similar functions in sentences
  - Undergo similar transformations
- Terminology
  - POS (part-of-speech tag) are also called grammatical tag, grammatical category, syntactic class, word class

Substitution Test

- Two words belong to the same part of speech if replacing one with another does not change the grammaticality of a sentence.
  - The {sad, big, fat, green, …} dog is barking.

How many word classes are there?

- A basic set:
  - N(oun), V(erb), Adj(ective), Adv(erb), Prep(osition), Det(erminer), Aux(ilaries), Part(icle), Conj(unction)
- A simple division: open/content vs. closed/function
  - Open: N, V, Adj, Adv
    - New members are added frequently
  - Closed: Prep, Det, Aux, Part, Conj, Num
    - New members are added rarely
- Many subclasses, e.g.
  - eats/V ⇒ eat/VB, eat/VBP, eats/VBZ, ate/VBD, eaten/VBN, eating/VBG, ...

POS tagging

- Goal: assign the right part of speech tag (noun, verb, …) to words in a text

  "The/AT representative/NN put/VBD chairs/NNS on/IN the/AT table/NN."
- What set of parts of speech do we use?
  - There are various standard tagsets to choose from; some have a lot more tags than others
  - The choice of tagset is based on the application
  - Accurate tagging can be done with even large tagsets

What does Tagging do?

1. Collapses distinctions
   - E.g., all personal pronouns tagged as **PRP**
   - Lexical identity may be completely discarded

2. Introduces distinctions (by reducing ambiguity)
   - E.g., "deal" tagged with **NN** or **VB**

Why do POS Tagging?

- **word sense disambiguiaton (semantics)**
  - Limits the range of meanings, "deal" as noun vs. "deal" as verb
- **speech recognition / synthesis (better accuracy)**
  - how to recognize/pronounce a word
  - *CONtent*/noun VS *conTENT*/adj
- **stemming**
  - which morphological affixes the word can take
  - adverb - *ly* = noun (*friendly* - *ly* = friend)
- **question answering**
  - analyzing a query to understand what type of entity the user is looking for and how it is related to other noun phrases mentioned in the question
- **partial parsing/chunking**
  - to find noun phrases/verb phrases
- **information extraction**
  - tagging and partial parsing help identify useful terms and relationships between them

Tag sets

- Different tag sets, depends on the purpose of the application
  - 45 tags in Penn Treebank
  - 62 tags in CLAWS with BNC corpus
  - 79 tags in Church (1991)
  - 87 tags in Brown corpus
  - 147 tags in C7 tagset
  - 258 tags in Tzoukermann and Radev (1995)

Penn Treebank

- First syntactically annotated corpus
- 1 million words from Wall Street Journal
- Part of speech tags and syntax trees

Important Penn Treebank tags

- 45 tags

IN	preposition or subordinating conjunct.
JJ	adjective or numeral, ordinal
JJR	adjective, comparative
NN	noun, common, singular or mass
NNP	noun, proper, singular
NNS	noun, common, plural
TO	"to" as preposition or infinitive marker
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle or gerund
VBN	verb, past participle
VBP	verb, present tense, not 3rd p. singular
VBZ	verb, present tense, 3rd p. singular
...	

Verb inflection tags

VBP	base present	*take*
VB	infinitive	*take*
VBD	past	*took*
VBG	present participle	*taking*
VBN	past participle	*taken*
VBZ	present 3sg	*takes*
MD	modal	*can, would*

The entire Penn Treebank tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Slide modified from Massimo Poesio's

Terminology

- Suppose we have text "The cat decided to jump on the couch to play with another cat"
- Terminology
  - Word type
    - Distinct words in the text (vocabulary), the text above has 10 word types: "the cat decided to jump on couch play with another"
  - Word token
    - any word occurring in the text
    - The text above has 13 word tokens

Distribution of Tags

- Parts of speech follow the usual frequency-based distributional behavior
  - Most word types have only one part of speech
  - Of the rest, most have two
  - A small number of word types have lots of parts of speech
  - Unfortunately, the word types with lots of parts of speech occur with high frequency (and words that occur most frequently tend to have multiple tags)

Most word types are not ambiguous but...

- but most word types are rare…
- Brown corpus (Francis&Kucera, 1982):
  - 11.5% word types are ambiguous (>1 tag)
  - 40% word tokens are ambiguous (>1 tag)

	num. word types	
Unambiguous (1 tag)	**35 340**	
Ambiguous (>1 tag)	**4 100**	
2 tags	3760	
3 tags	264	
4 tags	61	
5 tags	12	
6 tags	2	
7 tags	1	*"still"*

Why is Tagging Hard?

- Tagging is a type of disambiguation
- Examples:
  1. Book/**VB** that/**DT** flight/**NN**
     - "book" can also be **NN**
     - Can I read a ***book*** on this flight?
  2. Does/**VBZ** that/**DT** flight/**NN** serve/**VB** dinner/**NN** ?
     - "that" can also be a **complementizer**
     - My travel agent said ***that*** there would be a meal on this flight.

Potential Sources of Disambiguation

1. **Lexical information:**
   - look up all possible POS for a word in a dictionary
   - *"table"*: {noun, verb} but not a {adj, prep,…}
   - *"rose"*: {noun, adj, verb} but not {prep, ...}
2. **Syntagmatic information:**
   - some tag sequences are more probable than others:
   - **DET** + **N** occur frequently but **DET**+**V** never occurs
   - **ART**+**ADJ**+**N** is more probable than **ART**+**ADJ**+**VB**
   - We can find the syntagmatic information
     - by talking to the experts
     - or, better, from training corups

Syntagmatic information from Corpus

- For a is a sequence of tags $t_1, t_2, .., t_k$ compute $P(t_1, t_2, .., t_k)$, which will tell us how likely this tag sequence is
  - we have done something very similar before, i.e. we computed probability of a sequence of words
  - will make similar approximations as before,

$$P(t_n | t_1, t_2, .., t_{n-1}) = P(t_n | t_{n-k} \ldots t_{n-1})$$

  - in fact, for computational efficiency, the assumption will be

$$P(t_n | t_1, t_2, .., t_{n-1}) = P(t_n | t_{n-1})$$

Techniques to POS tagging

1. rule-based tagging
   - uses hand-written rules
2. statistical tagging
   - uses probabilities computed from training corpus
     - Charniak
     - Markov Model based

Rule-based POS Tagging

- Step 1: Assign each word with all possible tags
  - use dictionary
- Step 2: Use if-then rules to identify the correct tag in context (disambiguation rules)

Rule-based POS Tagging: Sample rules

- **N-IP rule**:
  A tag N (noun) cannot be followed by a tag IP (interrogative pronoun)
  ... man who …
  - man: {N}
  - who: {RP, IP} --> {RP} relative pronoun
- **ART-V rule**:
  A tag ART (article) cannot be followed by a tag V (verb)
  ...the book…
  - the: {ART}
  - book: {N, V} --> {N}

Rule-based Tagger

- using only syntagmatic patterns
  - Green & Rubin (1971)
  - accuracy of 77%
- In addition, it is very time consuming to come up with the rules and need an expert in English to come up with the rules

Statistical POSTagger: Charniak 1993

- Simplest statistical tagger
- Use corpus to calculate most probable tag for each *word*
  - that is the one maximizing

    count(*word* has tag *t*)/count(*word*)
- Charniak tagging assigns POS tag to each word separately
- Given a *word* to tag,
  1. for each possible tag *t* for this *word*, compute

     count(*word* has tag *t*)
  2. choose tag *t* that maximizes the above

Statistical POSTagger: Charniak 1993

- Accuracy of 90%
  - contrast with 77% accuracy of the rule-based tagger!
  - Another evidence of the power of statistical methods over rule-based meothds
  - MUCH better than rule based, but not very good...
    - 1 mistake every 10 words
  - funny fact: every word will have only one POS assigned to it (book will always be assigned the noun tag)
- This tagger is used mostly as baseline for evaluation
- How do we improve it?
  - tag of a word should depend on tags of other words around it, i.e. have to take “context” in the account
  - in other words, some sequence of tags are much more likely than others

Statistical Tagger: Markov Model Based

- Suppose we want to tag sentence of words $w_1\, w_2 \dots\, w_n$
- Let $t_1\, t_2 \dots\, t_n$ be a possible sequence of tags corresponding to the sentence above
  - That is t_i is a tag for word w_i
- Let $w_{1,n}$ be shorthand for sentence $w_1\, w_2 \dots\, w_n$ and $t_{1,n}$ be shorthand for its tagging $t_1\, t_2 \dots\, t_n$
- We want to find the “best” tagging $t_{1,n}$ out of all possible taggings
- We have 2 sources of information in our corpus:
  1. given that the previous word tag is t_i, we can find how likely the tag of the next word is t_{i+1} , namely $P(t_{i+1}|t_i)$
  2. we can find how likely is each word for each tag, namely $P(w_i|t_i)$
     - tells us how likely part of speech t_i will “generate” a word w_i
     - For example, if we know that that tag of the word is t_i = **noun**, what is the probability of the word to be “book”
     - P(book|verb) > P(book|noun), because there are many more nouns than verbs

Statistical Tagger: Markov Model Based

- Suppose we are given tag assignment t_{1n} for sentence w_{1n}
- Using Bayes law:

$$P(t_{1,n} \mid w_{1,n}) = \frac{P(w_{1,n} \mid t_{1,n})P(t_{1,n})}{P(w_{1,n})}$$

- This says:

$$P\left(\boxed{\begin{array}{c}\textbf{word\_1}\\ \textbf{has tag\_1}\end{array}} \quad \boxed{\begin{array}{c}\textbf{word\_2}\\ \textbf{has tag\_2}\end{array}} \cdots\cdots \boxed{\begin{array}{c}\textbf{word\_n}\\ \textbf{has tag\_n}\end{array}}\right) =$$

$$= \frac{P\left(\boxed{\begin{array}{c}\textbf{tag\_1 ``gives''}\\ \textbf{word\_1}\end{array}} \cdots \boxed{\begin{array}{c}\textbf{tag\_n ``gives''}\\ \textbf{word\_n}\end{array}}\right) P\left(\boxed{\textbf{tag\_1}} \cdots \boxed{\textbf{tag\_n}}\right)}{P\left(\boxed{\textbf{word\_1}} \cdots \boxed{\textbf{word\_n}}\right)}$$

Statistical Tagger: Markov Model Based

$$P(t_{1,n} \mid w_{1,n}) = \frac{P(w_{1,n} \mid t_{1,n})P(t_{1,n})}{P(w_{1,n})}$$

- We will make two simplifying assumptions

1. Given a POS tag, probability of a word is independent of the tags of other words in a sentence:

$$P(w_{1,n} \mid t_{1,n}) = \prod_{i=1}^{n} P(w_i \mid t_i)$$

$$P\left(\boxed{\begin{array}{c}\textbf{tag\_1 ``gives''}\\ \textbf{word\_1}\end{array}} \cdots \boxed{\begin{array}{c}\textbf{tag\_n ``gives''}\\ \textbf{word\_n}\end{array}}\right) =$$

$$= P\left(\boxed{\begin{array}{c}\textbf{tag\_1 ``gives''}\\ \textbf{word\_1}\end{array}}\right) \cdots P\left(\boxed{\begin{array}{c}\textbf{tag\_n ``gives''}\\ \textbf{word\_n}\end{array}}\right)$$

- $P(w_i \mid t_k)$ can be approximated from the tagged corpus as: $C(w_i$ has tag $t_k)/C(t_k)$, that is how many times word_i has tag_k divided by how often tag_k occurs in the corpus

Statistical Tagger: Markov Model Based

$$P(t_{1,n} \mid w_{1,n}) = \frac{P(w_{1,n} \mid t_{1,n})P(t_{1,n})}{P(w_{1,n})}$$

2. Each tag is only dependent only on one previous tag:

$$P(t_{1,n}) = \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

$$P(\boxed{\text{tag\_1}} \;\ldots\; \boxed{\text{tag\_n}}) =$$

$$= \boldsymbol{P}(\boxed{\text{tag\_1}} \mid \boxed{\text{tag\_0}}) \cdots P(\boxed{\text{tag\_n}} / \boxed{\text{tag\_(n-1)}})$$

- this is **Markov** assumption that we have seen before in language modeling
- Recall that P(tag 1| tag2) can be approximated by C(tag2 tag1)/C(tag2)
- Here P(tag 1| tag 0) stands for P(tag 1) and is approximated by C(tag1)/(number of tokens in the training corpus)

Statistical Tagger: Markov Model Based

- Using these 2 simpifications, we get

$$P(t_{1,n} \mid w_{1,n}) = \frac{\prod_{i=1}^{n} P(w_i \mid t_i)P(t_i \mid t_{i-1})}{P(w_{1,n})}$$

- Given a possible sequence of tags $t_{1,n}$ for sentence $w_{1,n}$, we can evaluate how good this tag/sequence assignment is using $P(t_{1,n} \mid w_{1,n})$
- Algorithm: go over all possible tag assignments and choose the tag assignment which gives highest $P(t_{1,n} \mid w_{1,n})$
  - Notice that $P(w_{1,n})$ **does not** effect maximization so we do not have to compute it

Statistical Tagger: Markov Model Based

- Algorithm: given sentence $w_{1,n}$ go over all possible tag assignments $t_{1,n}$ and compute

$$\boxed{\prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})}$$

- Choose final tagging $t_{1,n}$ which maximizes
- Efficiency
  - For each word w_i, try only the tags given by the dictionary (lexical information)
  - Ex: for "fly", possible tags are NOUN, VERB and also ADJECTIVE (meaning "keen" or "artful", mainly in England)

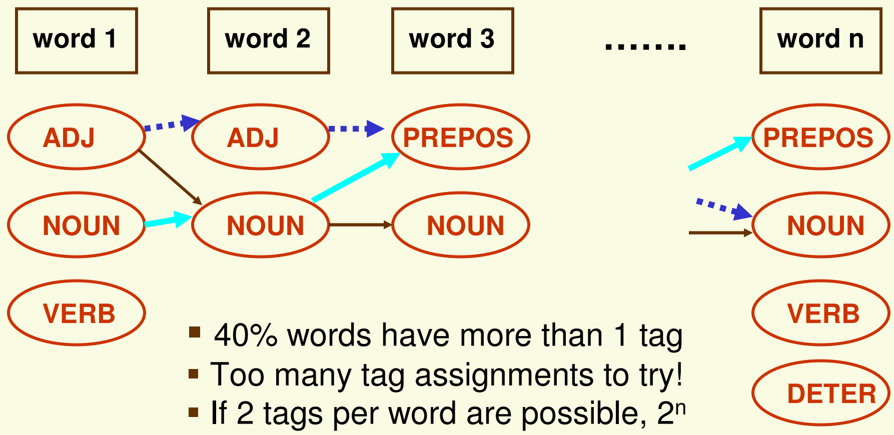
Statistical Tagger: Markov Model Based

- Side note: Markov tagger becomes Charniak's tagger if tags are assumed independent, that is if $P(t_i|t_{i-1}) = P(t_i)$, then

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

Statistical Tagger: Markov Model Based

- Algorithm: given sentence $w_{1,n}$ go over all possible tag assignments $t_{1,n}$ and compute $\prod_{i=1}^{n} P(w_i|\ t_i)P(t_i|t_{i-1})$



- 40% words have more than 1 tag
- Too many tag assignments to try!
- If 2 tags per word are possible, 2^n tag assignments are possible
  - exhaustive search is exponential

MM based Tagger: Viterbi Algorithm

- Fortunately there is a very useful algorithm (Viterbi)
- If there are k tags per word and n words, can find best tagging in time k^2n
  - There are k^n different tag assignments possible
- First, to avoid floating point underflows, take logarithm of

$$\prod_{i=1}^{n} P(w_i / t_i)P(t_i / t_{i-1})$$

$$log\left[\prod_{i=1}^{n} P(w_i / t_i)P(t_i / t_{i-1})\right] = \sum_{i=1}^{n}(log\,P(w_i / t_i) + log\,P(t_i / t_{i-1}))$$

- Now we want to maximize:

$$\sum_{i=1}^{n}[log\,P(w_i / t_i) + log\,P(t_i / t_{i-1})] = \sum_{i=1}^{n}\underbrace{log\,P(w_i / t_i)}_{\text{how likely word } w_i \text{ is for tag } t_i} + \sum_{i=1}^{n}\underbrace{log\,P(t_i / t_{i-1})}_{\text{how likely tag } t_i \text{ to follow tag } t_{i-1}}$$

MM based Tagger: Viterbi Algorithm

- Maximizing:

$$\sum_{i=1}^{n} log\ P(w_i \mid t_i) + \sum_{i=1}^{n} log\ P(t_i \mid t_{i-1})$$

- Is equivalent to minimizing

$$-\sum_{i=1}^{n} log\ P(w_i \mid t_i) - \sum_{i=1}^{n} log\ P(t_i \mid t_{i-1})$$

- It is more convenient to minimize the expression above

MM based Tagger: Viterbi Algorithm

- So we need to find a sequence of tags $t_1, t_2, \ldots, t_n$ to minimize:

$$-\sum_{i=1}^{n} log\ P(w_i \mid t_i) - \sum_{i=1}^{n} log\ P(t_i \mid t_{i-1})$$

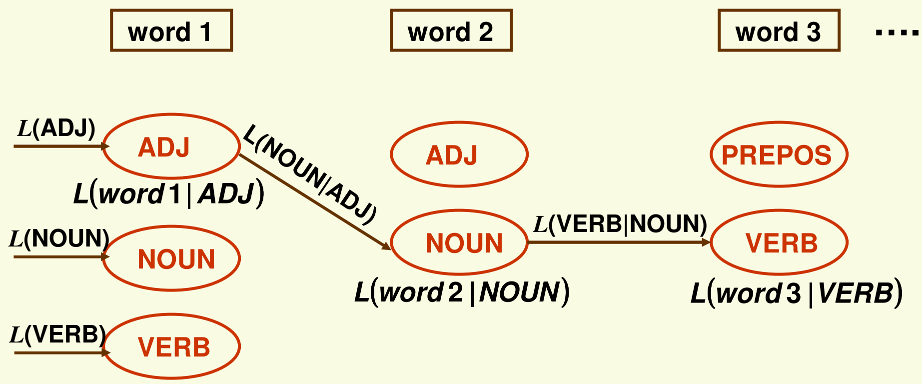
- To simplify notation, will write
  - $L(w|t)$ instead of $-log[P(w|t)]$
  - $L(t_i|t_{i-1})$ instead of $-log[P(t_i|t_{i-1})]$

- In new notation, we need to find a sequence of tags $t_1, t_2, \ldots, t_n$ to minimize:

$$\sum_{i=1}^{n} \left[L(w_i \mid t_i) + L(t_i \mid t_{i-1}) \right]$$

MM based Tagger: Viterbi Algorithm

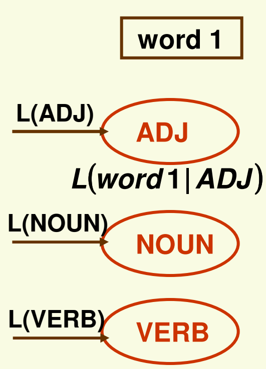
$$\boxed{\sum_{i=1}^{n}\left[L(w_i \mid t_i)+L(t_i \mid t_{i-1})\right]}$$



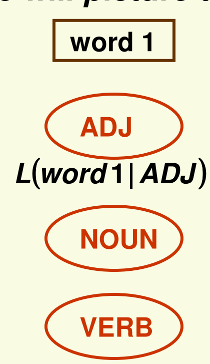
MM based Tagger: Viterbi Algorithm

- Let's change notation slightly for the first word
  - L(word1|tag) will stand for -log[P(word1|tag)] - log [P(tag)]
  - This will simplify the pictures:

instead of this:



we will picture this:

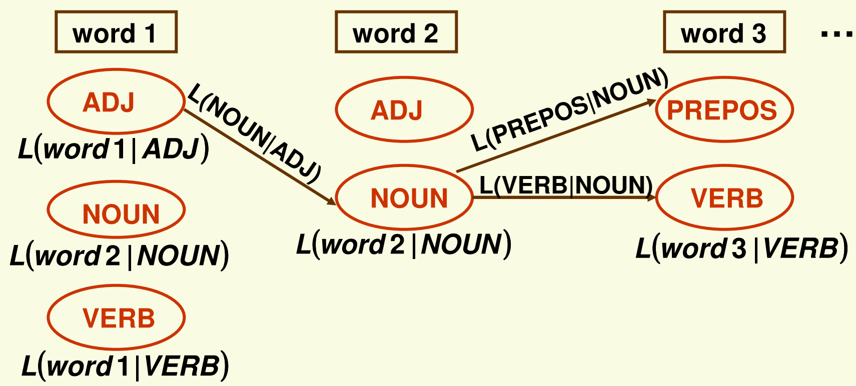


$$\boxed{\sum_{i=1}^{n}\log P(w_i \mid t_i)+\sum_{i=2}^{n}\log P(t_i \mid t_{i-1})}$$

MM based Tagger: Viterbi Algorithm

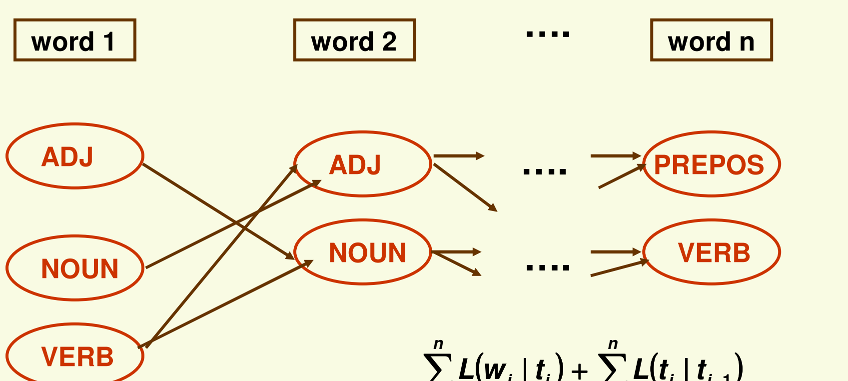
- Each node has cost L(word|tag)
- Each link between nodes has cost L(tag 1| tag 2)
- Cost of path, summing up node costs and edge costs is:

$$\sum_{i=1}^{n} L(w_i \mid t_i) + \sum_{i=2}^{n} L(t_i \mid t_{i-1})$$



MM based Tagger: Viterbi Algorithm

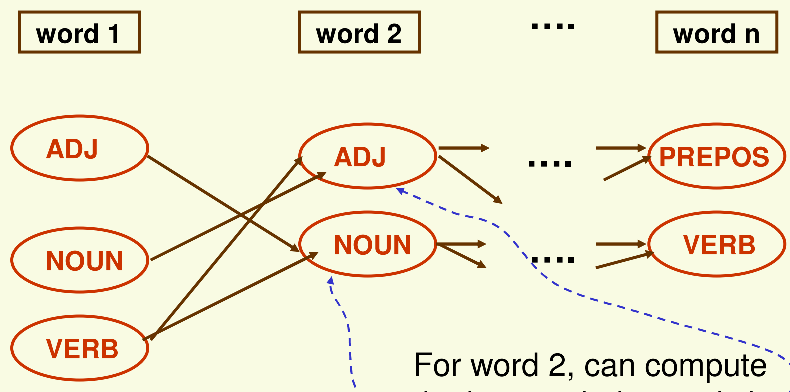
- So we need to find the path with smallest cost that starts at some node corresponding to word 1 and ends at some node corresponding to word n



$$\sum_{i=1}^{n} \underbrace{L(w_i \mid t_i)}_{\text{sits on nodes}} + \sum_{i=2}^{n} \underbrace{L(t_i \mid t_{i-1})}_{\text{sits on edges}}$$

MM based Tagger: Viterbi Algorithm

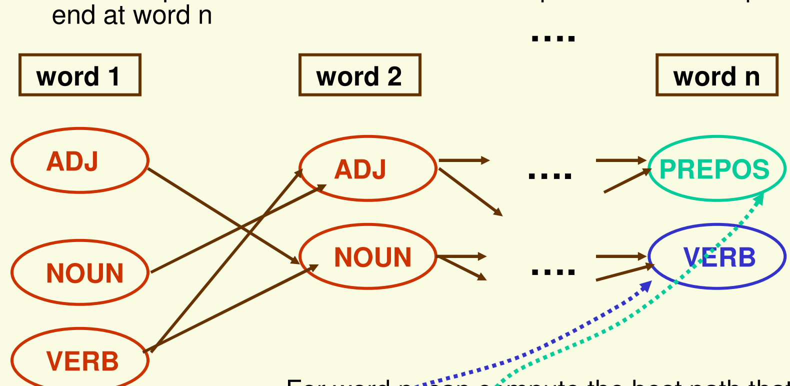
- Idea: for every node corresponding to word ***i***, we can efficiently find the best (smallest cost) path that ends at it (and starts at any node corresponding to word 1)



For word 2, can compute the best path that ends here and here

MM based Tagger: Viterbi Algorithm

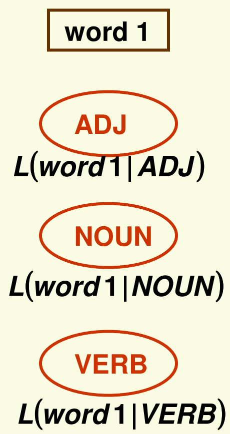
- First compute the best path that ends at any node for word 1
- Then compute the best path that ends at any node for word 2
- …..
- Finally compute the best path that ends at any node for word n
  - The best path overall is the smallest cost path out of those paths that end at word n



For word n, can compute the best path that ends here and here. Take the smallest cost path out of these two as the final best path

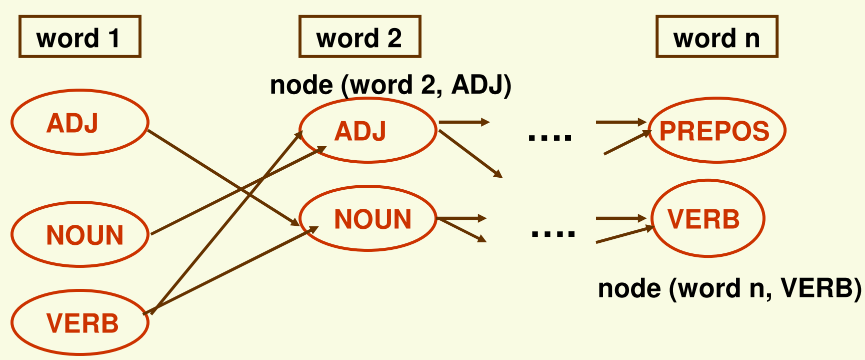
MM based Tagger: Viterbi Algorithm

- First compute the best path that ends at any node for word 1
- Trivial, since the path has just 1 node
- C(w1,tag) = L(w1|tag), holds the cost of the best path ending at (w1,tag)
- P(w1,tag) = null, holds the parent node on the best path ending at (w1,tag)



MM based Tagger: Viterbi Algorithm

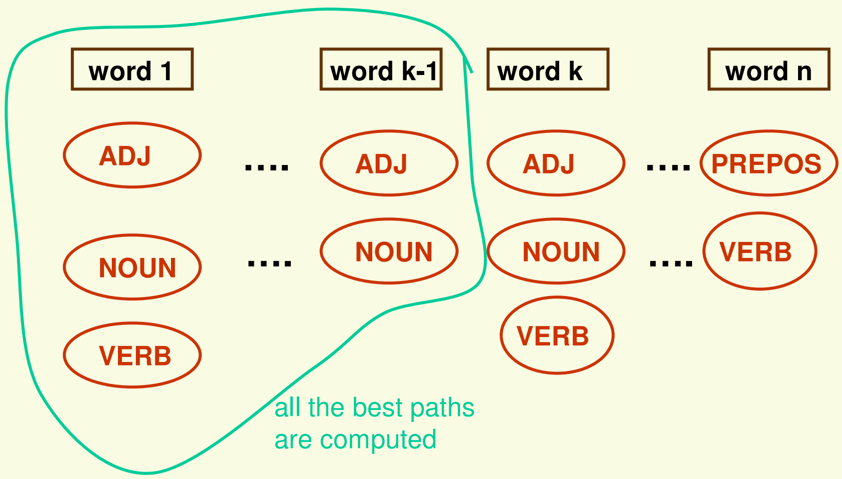
- In general, any node is specified by the word and the tag (word i,tag)



- Let C(word k,tag) stand for the cost of the best path starting at any node for word 1 and ending at node (word k, tag)
- Let P(word k,tag) stand for the parent on the best cost path starting at any node for word 1 and ending at node (word k, tag). Note that the parent must be the node (word k-1, tag')
- After all C(w,t) are computed, the best cost path overall is given by the minimum over all t of C(word n, t)

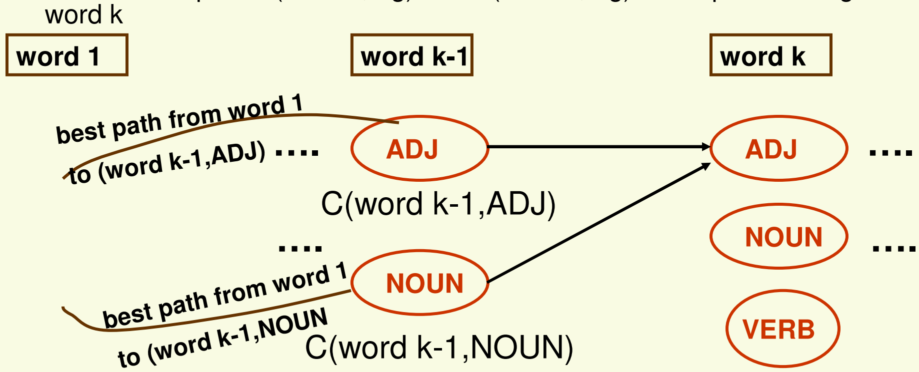
MM based Tagger: Viterbi Algorithm

- We saw that for all possible values of tag, computing C(word 1,tag) and P(word 1,tag) is trivial
- Suppose we have computed C(word i,tag) and P(word i,tag) for all tags and all i < k



MM based Tagger: Viterbi Algorithm

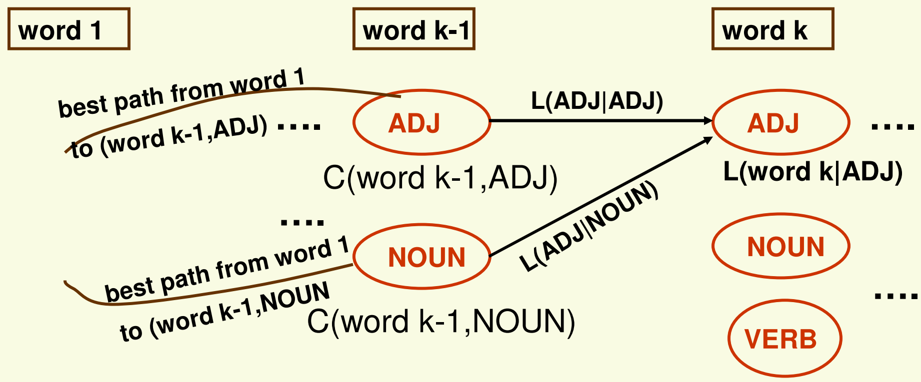
- Suppose we have computed C(word i,tag) for all tags and all i < k
- Need to compute C(word k,tag) and P(word k,tag) for all possible tags of word k



- Consider node (word k, ADJ). Let P be the best path from the word 1 to node (word k, ADJ). Path P will go either through either
  - node (word k-1,ADJ). In this case P follows the best path from word 1 to node (word k-1, ADJ)
  - or through node (word k-1,NOUN). In this case P follows the best path from word 1 to node (word k-1, NOUN)
  - we using property that a subpath of the best path is a best path itself

MM based Tagger: Viterbi Algorithm

- Therefore C(word k, ADJ) is the smaller of 2 quantities
  1. C(word k-1,ADJ)+L(ADJ|ADJ)+L(word k-1|ADJ)
     - In this case, P(word k, ADJ) = (word k-1, ADJ)
  2. C(word k-1,NOUN)+L(ADJ|NOUN)+L(word k-1|ADJ)
     - In this case, P(word k, ADJ) = (word k-1, NOUN)



MM based Tagger: Viterbi Algorithm

- In general, C(word k, tag) is computed as follows:

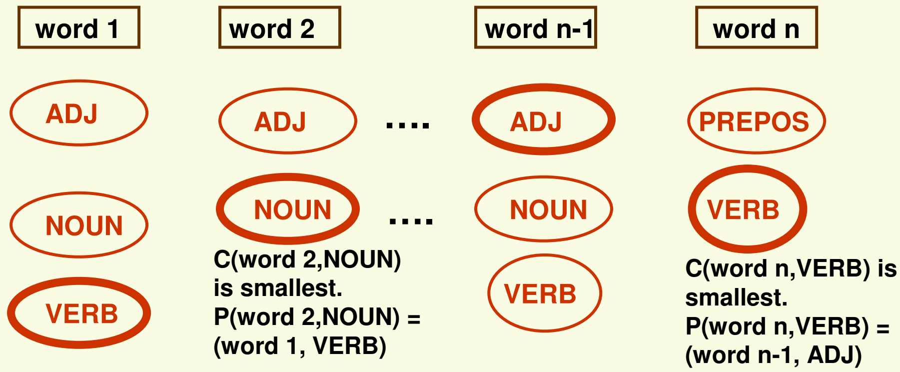
$$\boldsymbol{C(word\ k, tag) =}$$

$$= \underbrace{\min_{t \in T(word\ k-1)}}_{\text{search over all tags } t \text{ for word k-1}} \{\overbrace{C(word\ k-1, t)}^{\text{cost of best path from first word to node (word k-1, t)}} + \underbrace{L(tag \mid t)}_{\text{cost of going between nodes (word k-1, t) and (word k,tag)}}\} + \overbrace{L(word\ k \mid tag)}^{\text{cost of going through node (word k, tag)}}$$

- P(word k, tag) = (word k-1, t*) where t* is the tag for word k-1 minimizing the expression above

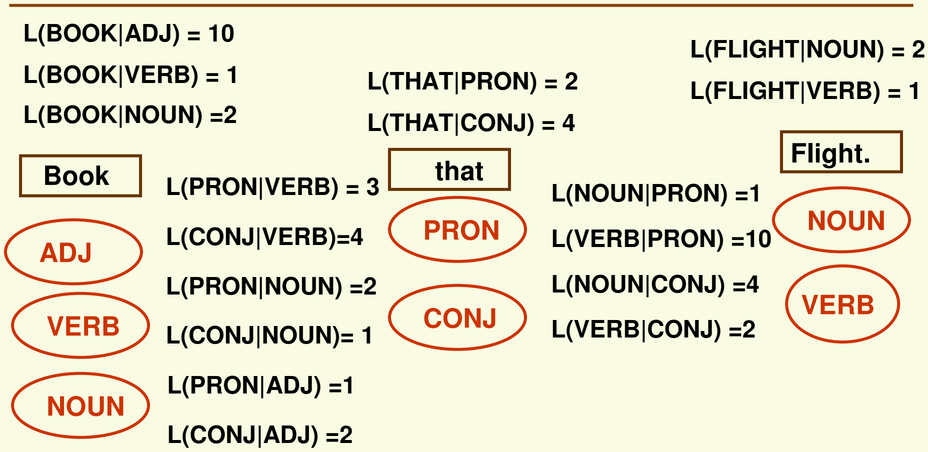
MM based Tagger: Viterbi Algorithm

- After we computed C(word i,t) for all ***i*** and t, the best cost path is found as the maximum of C(word n,t) over all tags ***t*** that word n can have
- The parents on the path can be traced back using the computed P(word i,t)



- Final tagging is: VERB NOUN … ADJ VERB

MM based Tagger: Small Example



- Notice that I made these log probabilities up

MM based Tagger: Small Example

L(BOOK|ADJ) = 10

L(BOOK|VERB) = 1

L(BOOK|NOUN) = 2

Book

ADJ

VERB

NOUN

- Iteration 1:
  - C(book,adj) = 10, P(book,adj) = null
  - C(book,verb) = 1, P(book,verb) = null
  - C(book,noun) = 2, P(book,noun) = null

MM based Tagger: Small Example

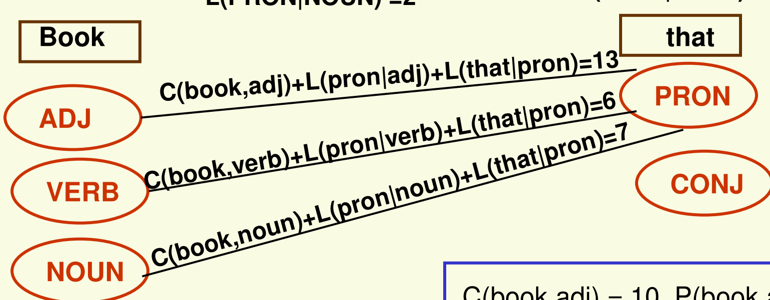
L(PRON|ADJ) =1

L(PRON|VERB) = 3

L(PRON|NOUN) =2

L(THAT|PRON) = 2

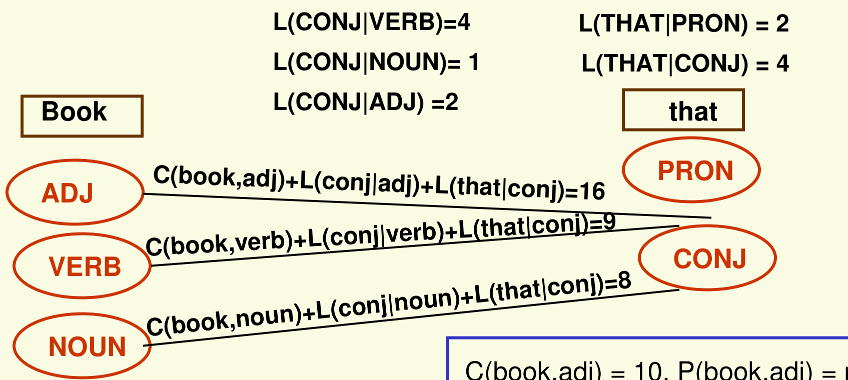
L(THAT|CONJ) = 4



C(book,adj) = 10, P(book,adj) = null
C(book,verb) = 1, P(book,verb) = null
C(book,noun) = 3, P(book,noun) = null

- Iteration 2:
  - C(that, pron) = 6, P(that, pron) = (book,verb)

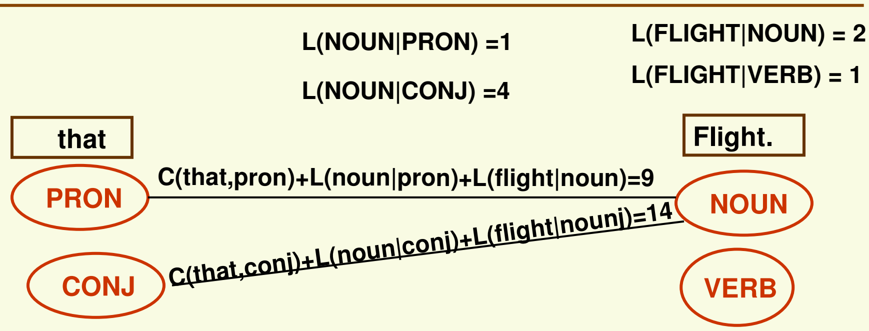
MM based Tagger: Small Example



C(book,adj) = 10, P(book,adj) = null
C(book,verb) = 1, P(book,verb) = null
C(book,noun) = 3, P(book,noun) = null

- Iteration 2:
  - C(that, pron) = 6, P(that, pron) = (book,verb)
  - C(that, conj) = 8, P(that, conj) = (book,noun)

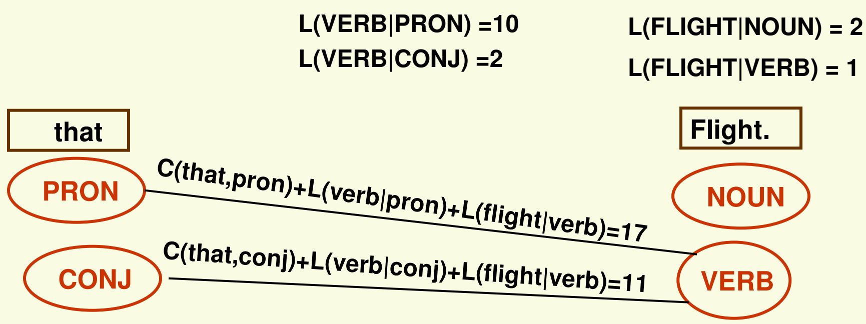
MM based Tagger: Small Example



C(book,adj) = 10, P(book,adj) = null
C(book,verb) = 1, P(book,verb) = null
C(book,noun) = 3, P(book,noun) = null
C(that, pron) = 6, P(that, pron) = (book,verb)
C(that, conj) = 8, P(that, conj) = (book,noun)

- Iteration 3:
  - C(flight, noun) = 9, P(flight, noun) = (that,pron)

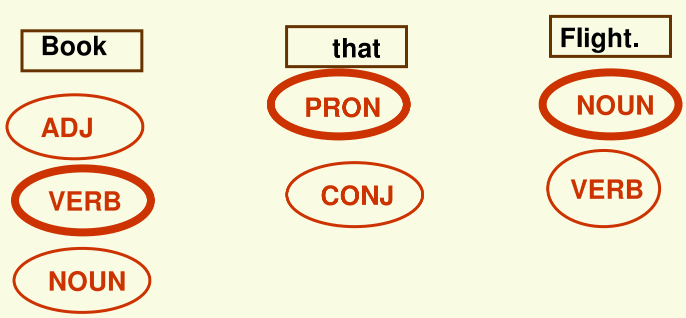
MM based Tagger: Small Example



C(book,adj) = 10, P(book,adj) = null
C(book,verb) = 1, P(book,verb) = null
C(book,noun) = 3, P(book,noun) = null
C(that, pron) = 6, P(that, pron) = (book,verb)
C(that, conj) = 8, P(that, conj) = (book,noun)

- Iteration 3:
  - C(flight, noun) = 9, P(flight, noun) = (that,pron)
  - C(flight, verb) = 11, P(flight, verb) = (that,conj)

MM based Tagger: Small Example



C(book,adj) = 10, P(book,adj) = null
C(book,verb) = 1, P(book,verb) = null
C(book,noun) = 3, P(book,noun) = null
C(that, pron) = 6, P(that, pron) = (book,verb)
C(that, conj) = 8, P(that, conj) = (book,noun)
C(flight, noun) = 9, P(flight, noun) = (that,pron)
C(flight, verb) = 11, P(flight, verb) = (that,pron)

Final Tagging is: Book<verb> that <pron> flight<noun>

Viterbi Algorithm

```
for each t ∈Tags(word 1) do
      C(word 1, t) = L(word 1 | t), P(word 1, t) = null
 for i ← 2 to n do
    for each t ∈ Tag(word i)  do
         C(word i,t) = -∝
         for each t' ∈ Tag(word i - 1)  do
                nextCost =  C(word i -1,t')+L(t|t')+L(word i|t)
                if nextCost < cost(word i,t ) do
                        C(word i,t ) =nextCost
                        P(word i,t) = t'
```

Note: Tags(word i) is the set of all possible tags for word i

Unknown Words

- Simplest method: assume an unknown word could belong to any tag; unknown words are assigned the distribution over POS over the whole lexicon
  - P(verb|"karumbula")=P(noun|"karumbula")=P(adjective|"karumbula")=…. etc
- Some tags are more common than others
  - for example a new word can be most likely a verb, a noun etc. but not a preposition or an article
- Use features of the word (morphological and other cues, for example words ending in *–ed* are likely to be past tense forms or past participles)

Tagging Accuracy

- Ranges from 96%-97%
- Depends on:
  - Amount of training data available
  - The tag set
  - Difference between training corpus and dictionary and the corpus of application
  - Unknown words in the corpus of application
- A change in any of these factors can have a dramatic effect on tagging accuracy – often much more stronger than the choice of tagging method