

# Part-of-Speech Tagging

Berlin Chen 2005

## References:

1. *Speech and Language Processing*, chapter 8
2. *Foundations of Statistical Natural Language Processing*, chapter 10

# Review

- Tagging (part-of-speech tagging)
  - The process of assigning (labeling) a part-of-speech or other lexical class marker to each word in a sentence (or a corpus)
    - Decide whether each word is a noun, verb, adjective, or whatever

The/AT representative/NN put/VBD chairs/NNS on/IN the/AT table/NN ✓

Or

The/AT representative/JJ put/NN chairs/VBZ on/IN the/AT table/NN

- An intermediate layer of representation of syntactic structure
  - When compared with syntactic parsing
- Above 96% accuracy for most successful approaches

Tagging can be viewed as a kind of syntactic disambiguation

# Introduction

- Parts-of-speech
  - Known as POS, word classes, lexical tags, morphology classes
- Tag sets
  - Penn Treebank : 45 word classes used (Francis, 1979)
    - Penn Treebank is a parsed corpus
  - Brown corpus: 87 word classes used (Marcus et al., 1993)
  - ....

The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

# The Penn Treebank POS Tag Set

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

# Disambiguation

- Resolve the ambiguities and chose the proper tag for the context
- Most English words are unambiguous (have only one tag) but many of the most common words are ambiguous
  - E.g.: “can” can be a (an auxiliary) verb or a noun
  - E.g.: statistics of Brown corpus

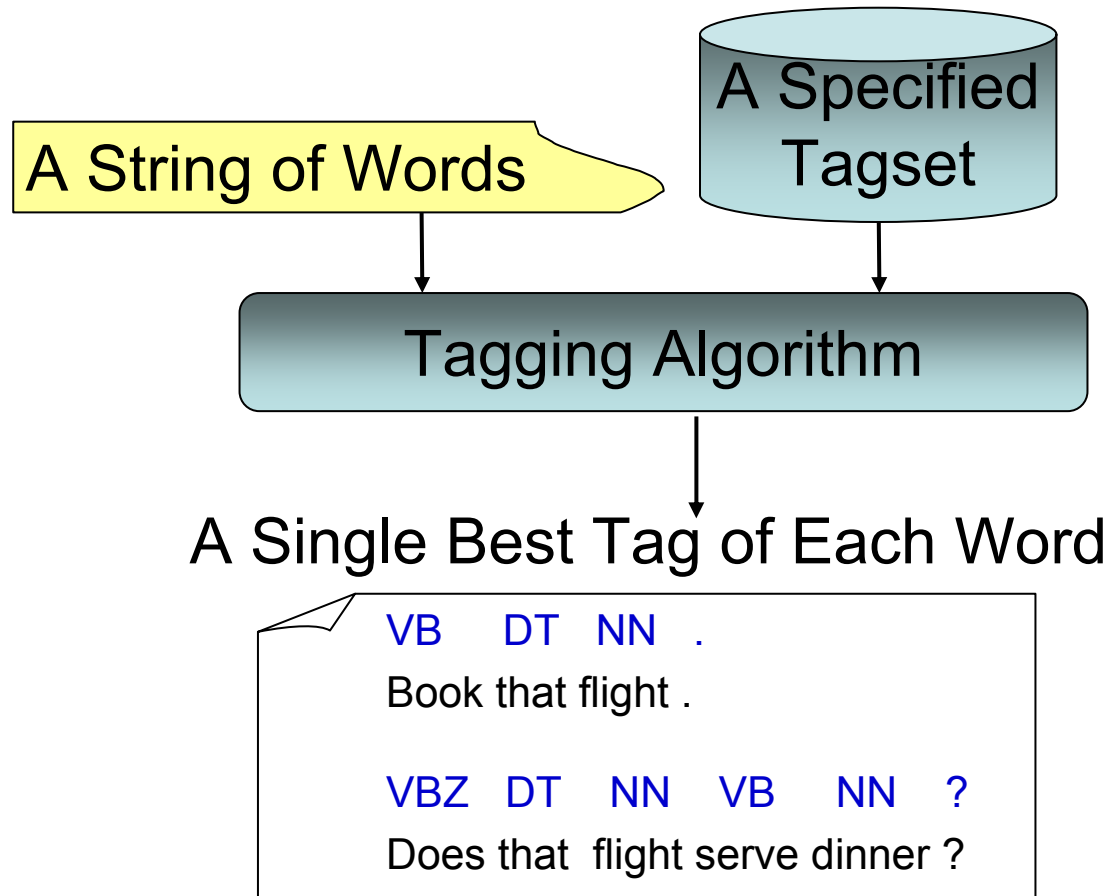
<b>Unambiguous (1 tag)</b>	<b>35,340</b>	
<b>Ambiguous (2–7 tags)</b>	<b>4,100</b>	
2 tags	3,760	
3 tags	264	
4 tags	61	
5 tags	12	
6 tags	2	
7 tags	1	(“still”)

- 11.5% word types are ambiguous

- But 40% tokens are ambiguous (However, the probabilities of tags associated a word are not equal → many ambiguous tokens are easy to disambiguate)

$$P(t_1|w) \neq P(t_2|w) \neq \dots$$

# Process of POS Tagging



## Two information sources used:

- Syntagmatic information (looking at information about tag sequences)
- Lexical information (predicting a tag based on the word concerned)

# POS Tagging Algorithms

## Fall into One of Two Classes

- Rule-based Tagger
  - Involve a large database of handcrafted disambiguation rules
    - E.g. a rule specifies that **an ambiguous word is a noun rather than a verb if it follows a determiner**
    - ENGTWOL: a simple rule-based tagger based on the **constraint grammar** architecture
- Stochastic/Probabilistic Tagger
  - Also called model-based tagger
  - Use a training corpus to compute the probability of a given word having a given context
  - E.g.: the HMM tagger chooses the best tag for a given word  
(**maximize the product of word likelihood and tag sequence probability**)

“a new play”

$P(\text{NN}|\text{JJ}) \approx 0.45$

$P(\text{VBP}|\text{JJ}) \approx 0.0005$

# POS Tagging Algorithms (cont.)

- Transformation-based/Brill Tagger
  - A hybrid approach
  - Like rule-based approach, determine the tag of an ambiguous word based on rules
  - Like stochastic approach, the **rules are automatically induced from previous tagged training corpus** with the machine learning technique
    - Supervised learning



# Rule-based POS Tagging

- Two-stage architecture
  - **First stage:** Use a dictionary to assign each word a list of potential **parts-of-speech**
  - **Second stage:** Use large lists of hand-written disambiguation rules to winnow down this list to a single part-of-speech for each word

Pavlov	had shown that salivation ...	An example for
Pavlov	<b>PAVLOV N NOM SG PROPER</b>	The ENGTOWL tagger
had	<b>HAVE V PAST VFIN SVO</b> (preterit) HAVE PCP2 SVO (past participle)	
shown	<b>SHOW PCP2 SVOO SVO SV</b>	
that	ADV PRON DEM SG DET CENTRAL DEM SG <b>CS</b> (complementizer)	A set of 1,100 constraints can be applied to the input sentence
salivation	<b>N NOM SG</b>	

# Rule-based POS Tagging (cont.)

- Simple lexical entries in the ENGTWOL lexicon

Word	POS	Additional POS features
smaller	ADJ	COMPARATIVE
entire	ADJ	ABSOLUTE ATTRIBUTIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog's	N	GENITIVE SG
furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	IMPERATIVE VFIN
show	V	PRESENT -SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

past participle

# Rule-based POS Tagging (cont.)

## ADVERBIAL-THAT RULE

**Given input:** "that"

**if**

(+1 A/ADV/QUANT); /\* if next word is adj, adverb, or quantifier \*/

(+2 SENT-LIM); /\* and following which is a sentence boundary, \*/

(NOT -1 SVOC/A); /\* and the previous word is not a verb like \*/

/\* 'consider' which allows adjs as object complements \*/

**then** eliminate non-ADV tags

**else** eliminate ADV tag

Example:

It isn't **that** odd!

ADV

A

I consider **that** odd.

Compliment

NUM

# HMM-based Tagging

- Also called *Maximum Likelihood Tagging*
  - Pick the most-likely tag for a word
- For a given sentence or words sequence , an HMM tagger chooses the tag sequence that maximizes the following probability

For a word at position  $i$  :

$$\text{tag}_i = \arg \max_j P(\text{word}_i | \text{tag}_j) \cdot P(\text{tag}_j | \text{previous } n - 1 \text{ tags})$$

word/lexical likelihood      tag sequence probability

N-gram HMM tagger

# HMM-based Tagging (cont.)

For a word  $w_i$  at position  $i$ , follow Bayes' theorem :

$$\begin{aligned} t_i &= \arg \max_j P(t_j | w_i, t_{i-1}, t_{i-2}, \dots, t_1) \\ &= \arg \max_j \frac{P(w_i, t_j | t_{i-1}, t_{i-2}, \dots, t_1)}{P(w_i | t_{i-1}, t_{i-2}, \dots, t_1)} \\ &= \arg \max_j P(w_i, t_j | t_{i-1}, t_{i-2}, \dots, t_1) \\ &= \arg \max_j P(w_i | t_j, t_{i-1}, t_{i-2}, \dots, t_1) P(t_j | t_{i-1}, t_{i-2}, \dots, t_1) \\ &\approx \arg \max_j P(w_i | t_j) P(t_j | t_{i-1}, t_{i-2}, \dots, t_{i-n+1}) \end{aligned}$$

# HMM-based Tagging (cont.)

- Assumptions made here
  - Words are independent of each other
    - A word's identity only depends on its tag
  - “Limited Horizon” and “Time Invariant” (“Stationary”)
    - **Limited Horizon**: a word's tag only depends on the previous few tags (limited horizon) and the dependency does not change over time (time invariance)
    - **Time Invariant**: the tag dependency won't change as tag sequence appears different positions of a sentence

Do not model long-distance relationships well !  
- e.g., Wh-extraction,...

# HMM-based Tagging (cont.)

- Apply bigram-HMM tagger to choose the best tag for a given word
  - Choose the tag  $t_i$  for word  $w_i$  that is most probable given the previous tag  $t_{i-1}$  and current word  $w_i$

$$t_i = \arg \max_j P(t_j | t_{i-1}, w_i)$$

- Through some simplifying Markov assumptions

$$t_i = \arg \max_j P(t_j | t_{i-1}) P(w_i | t_j)$$

tag sequence probability

word/lexical likelihood

# HMM-based Tagging (cont.)

- Apply bigram-HMM tagger to choose the best tag for a given word

$$t_i = \arg \max_j P(t_j | t_{i-1}, w_i)$$

$$= \arg \max_j \frac{P(t_j, w_i | t_{i-1})}{P(w_i | t_{i-1})}$$

The same for all tags

$$= \arg \max_j P(t_j, w_i | t_{i-1})$$

$$= \arg \max_j P(w_i | t_{i-1}, t_j) P(t_j | t_{i-1})$$

The probability of a word only depends on its tag

$$= \arg \max_j P(w_i | t_j) P(t_j | t_{i-1}) = \arg \max_j P(t_j | t_{i-1}) P(w_i | t_j)$$




# HMM-based Tagging (cont.)

- Example: Choose the best tag for a given word

Secretariat/**NNP** is /**VBZ** expected/**VBN** to/**TO** race/**VB** tomorrow/**NN**

to/ <b>TO</b> race/ <b>???</b>	0.34	0.00003	$P(\text{VB} \text{TO}) P(\text{race} \text{VB})=0.00001$
	0.021	0.00041	$P(\text{NN} \text{TO}) P(\text{race} \text{NN})=0.000007$

Pretend that the previous word has already tagged



# HMM-based Tagging (cont.)

- Apply bigram-HMM tagger to choose the best sequence of tags for a given sentence

$$\begin{aligned}
 \hat{T} &= \arg \max_T P(T | W) \\
 &= \arg \max_T \frac{P(T)P(W | T)}{P(W)} \\
 &= \arg \max_T P(T)P(W | T) \\
 &= \arg \max_{t_1, t_2, \dots, t_n} P(t_1, t_2, \dots, t_n)P(w_1, w_1, \dots, w_n | t_1, t_2, \dots, t_n) \\
 &= \arg \max_{t_1, t_2, \dots, t_n} \prod_{i=1}^n [P(t_i | t_1, t_2, \dots, t_{i-1})P(w_i | t_1, t_2, \dots, t_n)] \\
 &= \arg \max_{t_1, t_2, \dots, t_n} \prod_{i=1}^n [P(t_i | t_{i-m+1}, t_{i-m+2}, \dots, t_{i-1})P(w_i | t_i)]
 \end{aligned}$$

Assumptions:

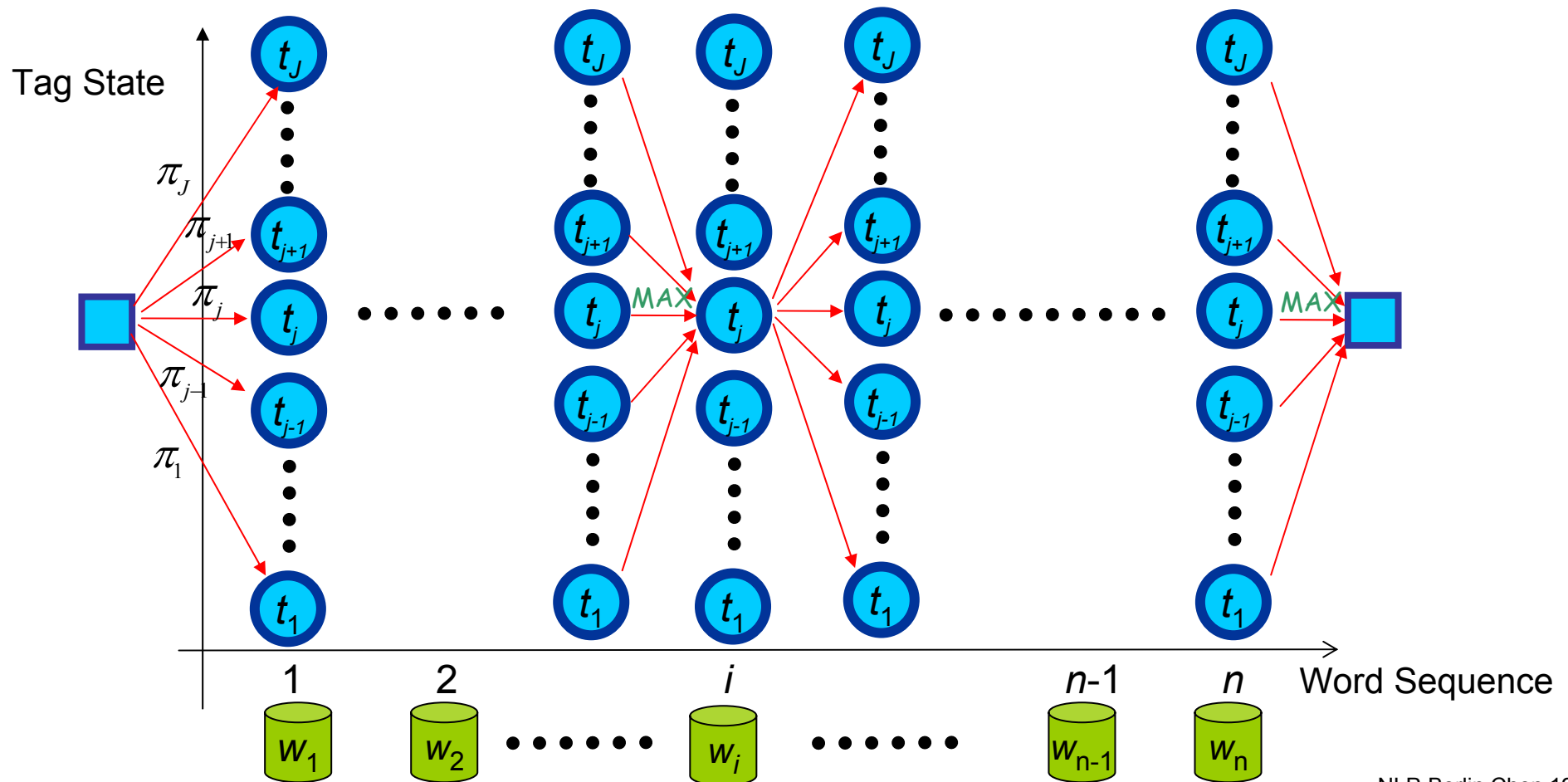
- words are independent of each other
- a word's identity only depends on its tag

Tag M-gram assumption

The probability of a word only depends on its tag

# HMM-based Tagging (cont.)

- The Viterbi algorithm for the bigram-HMM tagger
  - States: distinct tags
  - Observations: input word generated by each state



# HMM-based Tagging (cont.)

- The Viterbi algorithm for the bigram-HMM tagger

1. Initialization  $\delta_1(j) = \pi_j P(w_1 | t_j)$ ,  $1 \leq j \leq J$  ,  $\pi_j = P(t_j)$

2. Induction  $\delta_i(j) = \left[ \max_k \delta_{i-1}(k) P(t_j | t_k) \right] P(w_i | t_j)$ ,  $2 \leq i \leq n$ ,  $1 \leq j \leq J$

$$\psi_i(j) = \operatorname{argmax}_{1 \leq k \leq J} \left[ \delta_{i-1}(k) P(t_j | t_k) \right]$$

3. Termination  $X_n = \operatorname{argmax}_{1 \leq j \leq J} \delta_n(j)$

for  $i := n-1$  to 1 step -1 do

$$X_i = \psi_i(X_{i+1})$$

end

# HMM-based Tagging (cont.)

- Apply trigram-HMM tagger to choose the best sequence of tags for a given sentence
  - **When trigram model is used**

$$\hat{T} = \arg \max_{t_1, t_2, \dots, t_n} \left[ P(t_1)P(t_2|t_1) \prod_{i=3}^n P(t_i|t_{i-2}, t_{i-1}) \right] \left[ \prod_{i=1}^n P(w_i|t_i) \right]$$

- **Maximum likelihood estimation** based on the relative frequencies observed in the pre-tagged training corpus (labeled data)

$$P_{ML}(t_i|t_{i-2}, t_{i-1}) = \frac{c(t_{i-2}t_{i-1}t_i)}{\sum_j c(t_{i-2}t_{i-1}t_j)}$$

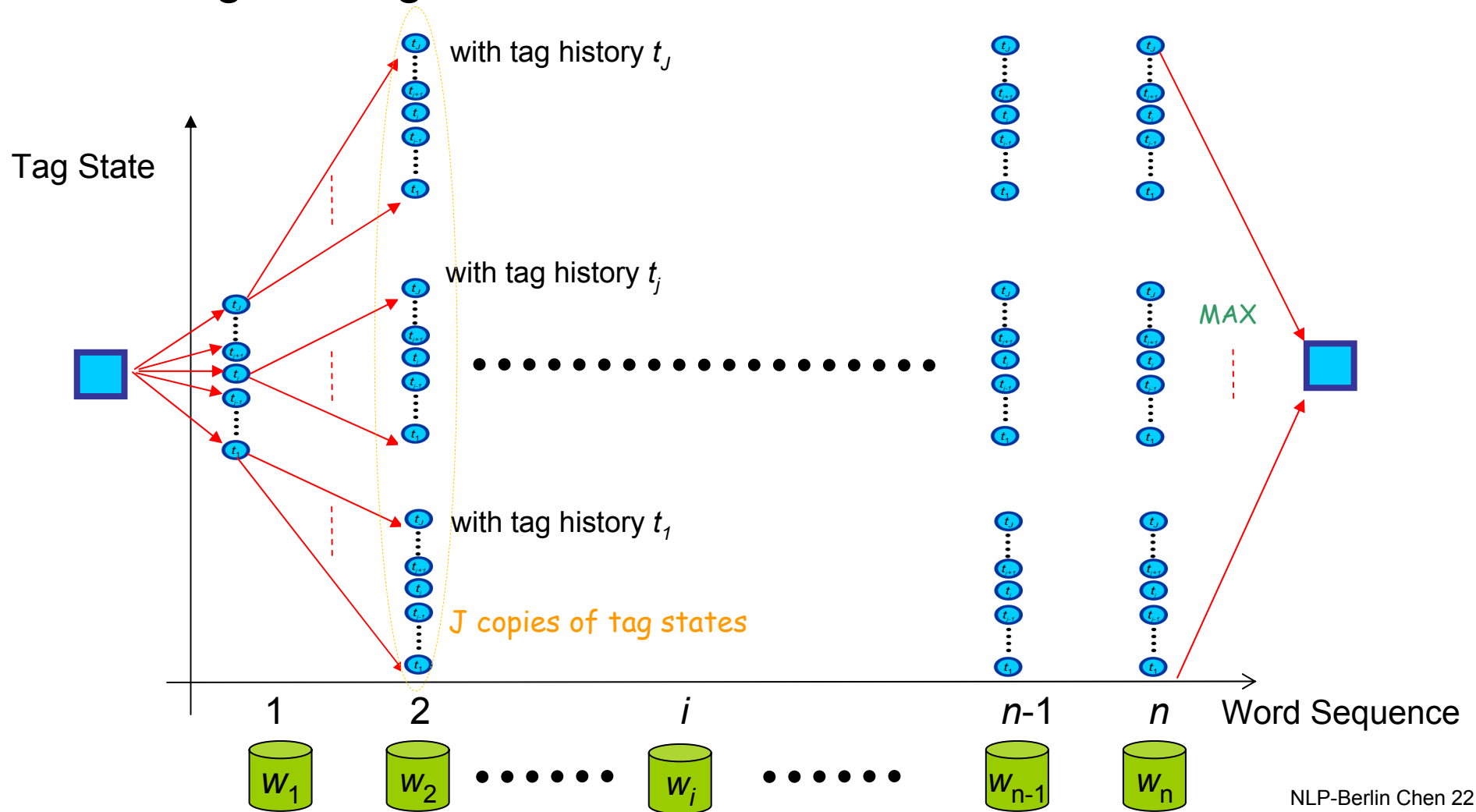
$$P_{ML}(w_i|t_i) = \frac{c(w_i, t_i)}{\sum_j c(w_j, t_i)}$$

Smoothing or linear interpolation are needed !

$$P_{smoothed}(t_i|t_{i-2}, t_{i-1}) = \alpha \cdot P_{ML}(t_i|t_{i-2}, t_{i-1}) + \beta \cdot P_{ML}(t_i|t_{i-1}) + (1 - \alpha - \beta) \cdot P_{ML}(t_i)$$

# HMM-based Tagging (cont.)

- Apply trigram-HMM tagger to choose the best sequence of tags for a given sentence



# HMM-based Tagging (cont.)

First tag	Second tag					
	AT	BEZ	IN	NN	VB	PERIOD
AT	0	0	0	48636	0	19
BEZ	1973	0	426	187	0	38
IN	43322	0	1325	17314	0	185
NN	1067	3720	42470	11773	614	21392
VB	6072	42	4758	1476	129	1522
PERIOD	8016	75	4656	1329	954	0

$$P(t_i|t_{i-1}) = \frac{c(t_{i-1}t_i)}{\sum_j c(t_{i-1}t_j)}$$

Table 10.3 Idealized counts of some tag transitions in the Brown Corpus. For example, NN occurs 48636 times after AT.

	AT	BEZ	IN	NN	VB	PERIOD
<i>bear</i>	0	0	0	10	43	0
<i>is</i>	0	10065	0	0	0	0
<i>move</i>	0	0	0	36	133	0
<i>on</i>	0	0	5484	0	0	0
<i>president</i>	0	0	0	382	0	0
<i>progress</i>	0	0	0	108	4	0
<i>the</i>	69016	0	0	0	0	0
.	0	0	0	0	0	48809

$$P(w_i|t_i) = \frac{c(w_i, t_i)}{\sum_j c(w_j, t_i)}$$

Table 10.4 Idealized counts for the tags that some words occur with in the Brown Corpus. For example, 36 occurrences of *move* are with the tag NN.

- Probability smoothing of  $P(t_i|t_{i-1})$  and  $P(w_i|t_i)$  is necessary

# HMM-based Tagging (cont.)

- Probability re-estimation based on unlabeled data
  - EM (Expectation-Maximization) algorithm is applied
    - Start with a **dictionary** that lists which tags can be assigned to which words
      - » word likelihood function can be estimated  $P(w_i|t_i)$
      - » tag transition probabilities set to be equal  $P(t_i|t_{i-1})$
    - EM algorithm learns (re-estimates) the word likelihood function for each tag and the tag transition probabilities
- However, a tagger trained on hand-tagged data worked better than one trained via EM
  - Treat the model as a **Markov Model** in training but treat them as a **Hidden Markov Model** in tagging

Secretariat/**NNP** is /**VBZ** expected/**VBN** to/**TO** race/**VB** tomorrow/**NN**



# Transformation-based Tagging

- Also called Brill tagging
  - An instance of Transformation-Based Learning (TBL)
- **Motive**
  - Like the **rule-based approach**, TBL is based on rules that specify what tags should be assigned to what word
  - Like the **stochastic approach**, rules are automatically induced from the data by the machine learning technique
- Note that TBL is a supervised learning technique
  - It assumes a pre-tagged training corpus

# Transformation-based Tagging (cont.)

- How the TBL rules are learned
  - Three major stages
    1. Label every word with its most-likely tag using a set of tagging rules (*use the broadest rules at first*)
    2. Examine every possible transformation (rewrite rule), and select the one that results in the most improved tagging (*supervised! should compare to the pre-tagged corpus*)
    3. Re-tag the data according this rule
  - The above three stages are repeated until some stopping criterion is reached
    - Such as insufficient improvement over the previous pass
  - *An ordered list of transformations (rules) can be finally obtained*

# Transformation-based Tagging (cont.)

- Example

- ①  $P(\text{NN}|\text{race})=0.98$   
 $P(\text{VB}|\text{race})=0.02$  } So, race will be initially coded as NN  
(label every word with its most-likely tag)



(a). is/VBZ expected/VBN to/To race/NN tomorrow/NN

(b). the/DT race/NN for/IN outer/JJ space/NN

② Refer to the correct tag  
Information of each word,  
and find the tag of race  
in (a) is wrong



- ③ Learn/pick a most suitable transformation rule: (by examining every possible transformation)

Change NN to VB while the previous tag is TO

**Rewrite rule:** expected/VBN to/To race/NN → expected/VBN to/To race/VB

# Transformation-based Tagging (cont.)

- Templates (abstracted transformations)
  - The set of possible transformations may be infinite
  - Should limit the set of transformations
  - The design of a small set of templates (abstracted transformations) is needed

E.g., a strange rule like:

transform NN to VB if the previous word was "IBM" and the word "the" occurs between 17 and 158 words before that

# Transformation-based Tagging (cont.)

- Possible templates (abstracted transformations)

The preceding (following) word is tagged **z**.  
 The word two before (after) is tagged **z**.  
 One of the two preceding (following) words is tagged **z**.  
 One of the three preceding (following) words is tagged **z**.  
 The preceding word is tagged **z** and the following word is tagged **w**.  
 The preceding (following) word is tagged **z** and the word two before (after) is tagged **w**.

Brill's templates.  
 Each begins with

"Change tag **a** to tag **b** when ...."

Schema	$t_{i-3}$	$t_{i-2}$	$t_{i-1}$	$t_i$	$t_{i+1}$	$t_{i+1}$	$t_{i+3}$
1				*			
2				*			
3				*			
4				*			
5				*			
6				*			
7				*			
8				*			
9				*			

**Table 10.7** Triggering environments in Brill's transformation-based tagger. Examples: Line 5 refers to the triggering environment "Tag  $t^j$  occurs in one of the three previous positions"; Line 9 refers to the triggering environment "Tag  $t^j$  occurs two positions earlier and tag  $t^k$  occurs in the following position."

# Transformation-based Tagging (cont.)

- Learned transformations

Verb, 3sg, past tense

Modal verbs (should, can,...)

#	Change tags		Condition	Example
	From	To		
1	NN	VB	Previous tag is TO	to/TO race/NN → VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not reply/NN → VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

Rules learned by Brill's original tagger

Verb, past participle

Verb, 3sg, Present

**Table 10.7** Triggering environments in Brill's transformation-based tagger. Examples: Line 5 refers to the triggering environment "Tag  $t^j$  occurs in one of the three previous positions"; Line 9 refers to the triggering environment "Tag  $t^j$  occurs two positions earlier and tag  $t^k$  occurs in the following position."

Source tag	Target tag	Triggering environment
NN	VB	previous tag is TO
VBP	VB	one of the previous three tags is MD
JJR	RBR	next tag is JJ <b>more valuable player</b>
VBP	VB	one of the previous two words is <i>n't</i>

Constraints for tags

Constraints for words

**Table 10.8** Examples of some transformations learned in transformation-based tagging.

# Transformation-based Tagging (cont.)

- Reference for tags used in the previous slide

Tag	Part Of Speech
AT	article
BEZ	the word <i>is</i>
IN	preposition
JJ	adjective
JJR	comparative adjective
MD	modal
NN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
PERIOD	. : ? !
PN	personal pronoun
RB	adverb
RBR	comparative adverb
TO	the word <i>to</i>
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle, gerund
VBN	verb, past participle
VBP	verb, non-3rd person singular present
VBZ	verb, 3rd singular present
WDT	<i>wh-</i> determiner ( <i>what, which</i> )

Table 10.1 Some part-of-speech tags frequently used for tagging English

# Transformation-based Tagging (cont.)

- Algorithm

```

function TBL(corpus) returns transforms-queue
  INITIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
  until end condition is met do
    templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
    best-transform ← GET-BEST-TRANSFORM(corpus, templates)
    APPLY-TRANSFORM(best-transform, corpus)
    ENQUEUE(best-transform-rule, transforms-queue)
  end
  return(transforms-queue)
  
```

append to the rule list

```

function GET-BEST-TRANSFORM(corpus, templates) returns transform
  for each template in templates
    (instance, score) ← GET-BEST-INSTANCE(corpus, template)
    if (score > best-transform.score) then best-transform ← (instance, score)
  return(best-transform)
  
```

```

function GET-BEST-INSTANCE(corpus, template) returns transform
  for from-tag ← from tag-1 to tag-n do
    for to-tag ← from tag-1 to tag-n do
      for pos ← from 1 to corpus-size do
        if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
          num-good-transforms(current-tag(pos-1))++
        elseif (correct-tag(pos) == from-tag && current-tag(pos) == from-tag)
          num-bad-transforms(current-tag(pos-1))++
        end
      best-Z ← ARGMAXt(num-good-transforms(t) - num-bad-transforms(t))
      if (num-good-transforms(best-Z) - num-bad-transforms(best-Z)
          > best-instance.Z) then
        best-instance ← "Change tag from from-tag to to-tag"
        if previous tag is best-Z
      end
    end
  return(best-instance)
  
```

for all combinations of tags

traverse corpus

X

Y

Z

score

Check if it is better than the best instance achieved in previous iterations

```

procedure APPLY-TRANSFORM(transform, corpus)
  for pos ← from 1 to corpus-size do
    if (current-tag(pos) == best-rule-from
        && (current-tag(pos-1) == best-rule-prev))
      current-tag(pos) = best-rule-to
  
```

The GET\_BEST\_INSTANCE procedure in the example algorithm is "Change tag from X to Y if the previous tag is Z".



# Homework-2: Tagging (I)

- Trace the Transformation-based Tagging Algorithm
  - Outline how the algorithm can be executed (depict a flowchart) and make your own remarks on the procedures of the algorithm
  - Analyze the associated time and space complexities of it
  - Draw your conclusions or findings
- Due 3/22

# Multiple Tags and Multi-part Words

- Multiple tags
  - A word is ambiguous between multiple tags and it is impossible or very difficult to disambiguate, so multiple tags is allowed, e.g.
    - adjective versus preterite versus past participle (JJ/VBD/VBN)
    - adjective versus noun as prenominal modifier (JJ/NN)
- Multi-part words
  - Certain words are split or some adjacent words are treated as a single word

would/MD n't/RB      Children/NNS 's/POS      treated as separate words

in terms of (in/I131 terms/I132 of/I133)      treated as a single word

# Tagging of Unknown Words

- Unknown words are a major problem for taggers
  - Different accuracy of taggers over different corpora is often determined by the proportion of unknown words
- How to guess the part of speech of unknown words?
  - Simplest unknown-word algorithm
  - Slightly more complex algorithm
  - Most-powerful unknown-word algorithm

# Tagging of Unknown Words (cont.)

- Simplest unknown-word algorithm
  - Pretend that each unknown word is ambiguous among all possible tags, with equal probability
    - Lose/ignore lexical information for unknown words
  - Must rely solely on the contextual POS-trigram (syntagmatic information) to suggest the proper tag

$$\hat{T} = \arg \max_{t_1, t_2, \dots, t_n} \left[ P(t_1)P(t_2|t_1) \prod_{i=3}^n P(t_i|t_{i-2}, t_{i-1}) \right] \left[ \prod_{i=1}^n P(w_i|t_i) \right]$$

- Slightly more complex algorithm
  - Based on the idea that the probability distribution of tags over unknown words is very similar to the distribution of tags over words that occurred only once (singletons) in a training set
  - The likelihood for an unknown word is determined by the **average of the distribution over all singleton in the training set** (similar to Good-Turing? )

Nouns or Verbs  $P(w_i|t_i)$ ?

# Tagging of Unknown Words (cont.)

- Most-powerful unknown-word algorithm
  - Hand-designed features
    - The information about how the word is spelled (inflectional and derivational features), e.g.:
      - Words end with s (→ plural nouns)
      - Words end with ed (→ past participles)
    - The information of word capitalization (initial or non-initial) and hyphenation

$$P(w_i | t_i) = p(\text{unknown - word} | t_i) \cdot p(\text{capital} | t_i) \cdot p(\text{endings/hyph} | t_i)$$

Assumption: independence between features

- Features induced by machine learning
  - E.g.: TBL algorithm uses templates to induce useful English inflectional and derivational features and hyphenation

The first N letters of the word  
The last N letters of the word

# Tagging of Unknown Words (cont.)

Feature	Value	NNP	NN	NNS	VBG	VBZ
unknown word	yes	0.05	0.02	0.02	0.005	0.005
	no	0.95	0.98	0.98	0.995	0.995
capitalized	yes	0.95	0.10	0.10	0.005	0.005
	no	0.05	0.90	0.90	0.995	0.995
ending	-s	0.05	0.01	0.98	0.00	0.99
	-ing	0.01	0.01	0.00	1.00	0.00
	-tion	0.05	0.10	0.00	0.00	0.00
	other	0.89	0.88	0.02	0.00	0.01

**Table 10.5** Table of probabilities for dealing with unknown words in tagging. For example,  $P(\text{unknown word} = \text{yes} | \text{NNP}) = 0.05$  and  $P(\text{ending} = \text{-ing} | \text{VBG}) = 1.0$ .

# Evaluation of Taggers

- Compare the tagged results with a human labeled **Gold Standard** test set in percentages of correction
  - Most tagging algorithms have an accuracy of around 96~97% for the sample tagsets like the Penn Treebank set
  - Upper bound (ceiling) and lower bound (baseline)
    - *Ceiling*: is achieved by seeing how well humans do on the task
      - A 3~4% margin of error
    - *Baseline*: is achieved by using the unigram most-like tags for each word
      - 90~91% accuracy can be attained

# Error Analysis

- Confusion matrix

	IN	JJ	NN	NNP	RB	VBD	VCN
IN	-	.2			.7		
JJ	.2	-	3.3	2.1	1.7	.2	2.7
NN		8.7	-				.2
NNP	.2	3.3	4.1	-	.2		
RB	2.2	2.0	.5		-		
VBD		.3	.5			-	4.4
VCN		2.8				2.6	-

(%)

- Major problems facing current taggers
  - NN (noun) versus NNP (proper noun) and JJ (adjective)
  - RP (particle) versus RB (adverb) versus JJ
  - VBD (past tense verb) versus VCN (past participle verb) versus JJ



# Applications of POS Tagging

- Tell what words are likely to occur in a word's vicinity
  - E.g. the vicinity of the possessive or person pronouns
- Tell the pronunciation of a word
  - DIScount (noun) and disCOUNT (verb) ...
- Advanced ASR language models
  - Word-class N-grams
- Partial parsing
  - A simplest one: find the noun phrases (names) or other phrases in a sentence

# Applications of POS Tagging (cont.)

- Information retrieval

- Word stemming
- Help select out nouns or important words from a doc
- Phrase-level information

*United, States, of, America* → "United States of America"  
*secondary, education* → "secondary education"

- Phrase normalization

*Book publishing, publishing of books*

- Information extraction

- Semantic tags or categories

# Applications of POS Tagging (cont.)

- Question Answering
  - Answer a user query that is formulated in the form of a question by return an appropriate noun phrase such as a location, a person, or a date
    - E.g. "Who killed President Kennedy?"

In summary, the role of taggers appears to be a fast lightweight component that gives sufficient information for many applications

- But not always a desirable preprocessing stage for all applications
- Many probabilistic parsers are now good enough !

# Class-based N-grams

- Use the lexical tag/category/class information to augment the  $N$ -gram models

$$P(w_n | w_{n-N+1}^{n-1}) = P(w_n | c_n) P(c_n | c_{n-N+1}^{n-1})$$

prob. of a word given the tag

prob. of a tag given the previous  $N-1$  tags

- Maximum likelihood estimation

$$P(w_i | c_j) = \frac{C(w)}{C(c)}$$
$$P(c_j | c_k) = \frac{C(c_k c_j)}{\sum_l C(c_l c_j)}$$

Constraints: a word may only belong to one lexical category

# Named-Entity Extraction

- Named entities (NE) include
  - Proper nouns as names for persons, locations, organizations, artifacts and so on
  - Temporal expressions such as “Oct. 10 2003” or “1:40 p.m.”
  - Numerical quantities such as “fifty dollars” or “thirty percent”
- Temporal expressions and Numerical quantities can be easily modeled and extracted by rules
- The personal/location/organization are much more difficult to identified
  - E.g., “White House” can be either an organization or a location name in different context

# Named-Entity Extraction (cont.)

- NE has its origin from the Message Understanding Conferences (MUC) sponsored by U.S. DARPA program
  - Began in the 1990's
  - Aimed at extraction of information from text documents
  - Extended to many other languages and spoken documents (mainly broadcast news)
- Common approaches to NE
  - Rule-based approach
  - Model-based approach
  - Combined approach

# Named-Entity Extraction (cont.)

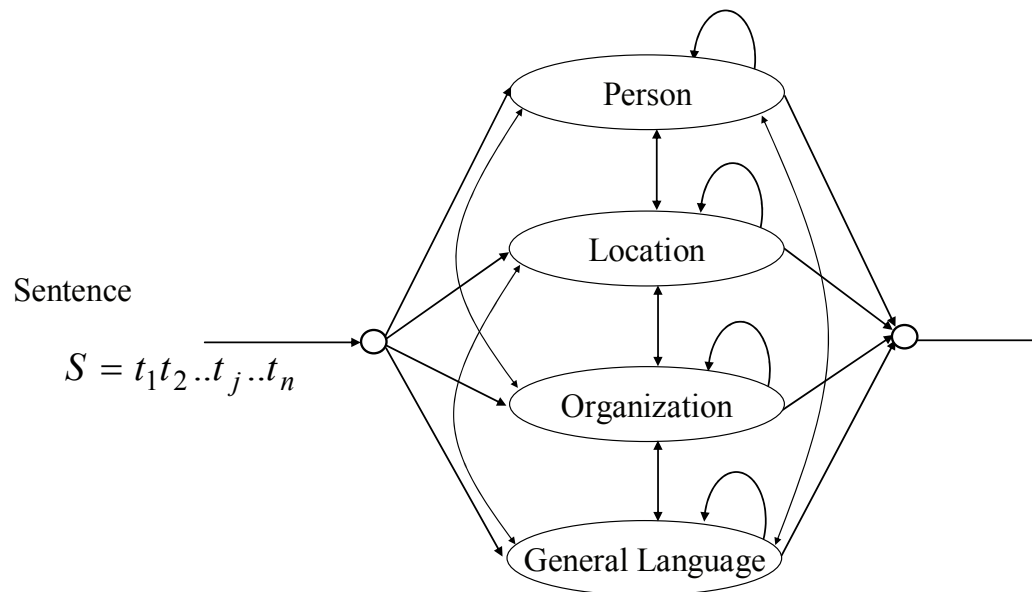
- Rule-based Approach
  - Employ various kinds of rules to identified named-entities
  - E.g.,
    - A cue-word “Co.” possibly indicates the existence of a company name in the span of its predecessor words
    - A cue-word “Mr.” possibly indicates the existence of a personal name in the span of its successor words
  - However, the rules may become very complicated when we wish to cover all different possibilities
    - Time-consuming and difficult to handcraft all the rules
    - Especially when the task domain becomes more general, or when new sources of documents are being handled

# Named-Entity Extraction (cont.)

- Model-based Approach

- The goal is usually to find the sequence of named entity labels (personal name, location name, etc.),  $E = n_1 n_2 \dots n_j \dots n_n$ , for a sentence,  $S = t_1 t_2 \dots t_j \dots t_n$ , which maximizes the probability  $P(E|S)$

- E.g., HMM is probably the best typical representative model used in this category





# Named-Entity Extraction (cont.)

- In HMM,
  - One state modeling each type of the named entities (person, location, organization)
  - One state modeling other words in the general language (non-named-entity words)
  - Possible transitions from states to states
  - Each state is characterized by a bi- or trigram language model
  - Viterbi search to find the most likely state sequence, or named entity label sequence  $E$  , for the input sentence, and the segment of consecutive words in the same named entity state is taken as a named entity

# Named-Entity Extraction (cont.)

- Combined approach
  - E.g., Maximum entropy (ME) method
    - Many different linguistic and statistical features, such as part-of-speech (POS) information, rule-based knowledge, term frequencies, etc., can all be represented and integrated in this method
    - It was shown that very promising results can be obtained with this method

# Named-Entity Extraction (cont.)

- Handling out-of-vocabulary (OOV) or unknown words
  - E.g., HMM
    - Divide the training data into two parts during training
      - In each half, every segment of terms or words that does not appear in the other half is marked as “Unknown”, such that the probabilities for both known and unknown words occurring in the respective named-entity states can be properly estimated
    - During testing, any segment of terms that is not seen before can thus be labeled “Unknown,” and the Viterbi algorithm can be carried out to give the desired results

# Named-Entity Extraction (cont.)

- Handling out-of-vocabulary (OOV) or unknown words for spoken docs
  - Out-of-vocabulary (OOV) problem is raised due to the limitation in the vocabulary size of speech recognizer
  - OOV words will be misrecognized as other in-vocabulary words
    - Lose their true semantic meanings
- Tackle this problem using SR & IR techniques
  - In SR (speech recognition)
    - Spoken docs are transcribed using a recognizer implemented with a lexical network modeling both word- and subword-level (phone or syllable)  $n$ -gram LM constraints
      - The speech portions corresponding to OOV words may be properly decoded into sequences of subword units

# Named-Entity Extraction (cont.)

- Tackle this problem using SR & IR techniques (cont.)
  - The subword  $n$ -gram LM is trained by the text segments corresponding to the low-frequency words not included in the vocabulary of the recognizer
  - In IR (Information Retrieval)
    - A retrieval process was performed using each spoken doc itself as a query to retrieve relevant docs from a temporal/topical homogeneous reference text collection
    - The indexing terms adopted here can be either word-level features, subword-level features, or both of them

# Named-Entity Extraction (cont.)

- Tackle this problem using SR & IR techniques (cont.)
  - Once the top-ranked text documents are selected, each decoded subword sequence within the spoken document, that are corresponding to a possible OOV word, can be used to match every possible text segments or word sequences within the top-ranked text documents
  - The text segment or word sequence within the top-ranked text docs that has the maximum combined score of phonetic similarity to the OOV word and relative frequency in the relevant text docs can thus be used to replace the decoded subword sequence of the spoken doc

$$\max_w \sum_{d \in D_r} P(e_{oov} | w) \cdot P(w | d) \cdot P(d | q_s)$$

The diagram shows the equation  $\max_w \sum_{d \in D_r} P(e_{oov} | w) \cdot P(w | d) \cdot P(d | q_s)$  with four red arrows pointing from descriptive text below to variables in the equation:

- An arrow points from "phone/syllable sequence of the OOV words" to  $e_{oov}$ .
- An arrow points from "word in the top-ranked relevant text doc set" to  $w$ .
- An arrow points from "doc belonging to the top-ranked relevant text doc set" to  $d$ .
- An arrow points from "spoken doc" to  $q_s$ .