ECE8813
Statistical Natural Language Processing

Lectures 14-15: Finite State Grammar and Part-of-Speech Tagging

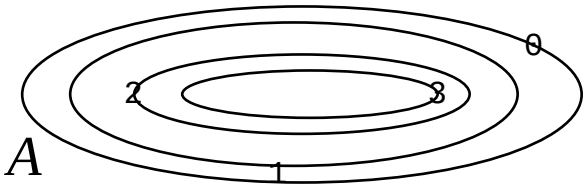
Chin-Hui Lee
School of ECE, Georgia Tech
Atlanta, GA 30332, USA
chl@ece.gatech.edu

Looking Ahead

- PoS Tagging (Lab4)
- Vector Space Representation
- Clustering (Lab 5)
- Text Categorization and IR (Lab 6)
- Probabilistic Context Free Grammar (???)
- Probabilistic Parsing (???)
- Alignment and Machine Trasnlation (???)
- Project Presentation (week before Finals)

Formal Grammar Specification

- Grammar *G={A, I, S, D}* and Language *L(G)*
  - G is defined by an alphabet set *A*, an intermediate set *I*, a root symbol *S*, and a set of derivation (production) rules *D*
  - *L(G)* is the language of the set of sentences generated by *G*
- Type of String Grammars
  - *Type 0: free* or *unrestricted*



$$D=\{\theta\rightarrow\psi\}\quad \theta\in I\quad \psi\in I\cup A$$

  - *Type 1: context-sensitive*

$$D=\{\alpha\theta\beta\rightarrow\alpha\psi\beta\}\quad \theta\in I\quad \psi\in I\cup A\quad \alpha,\beta:\text{string}$$

  - *Type 2: context-free*

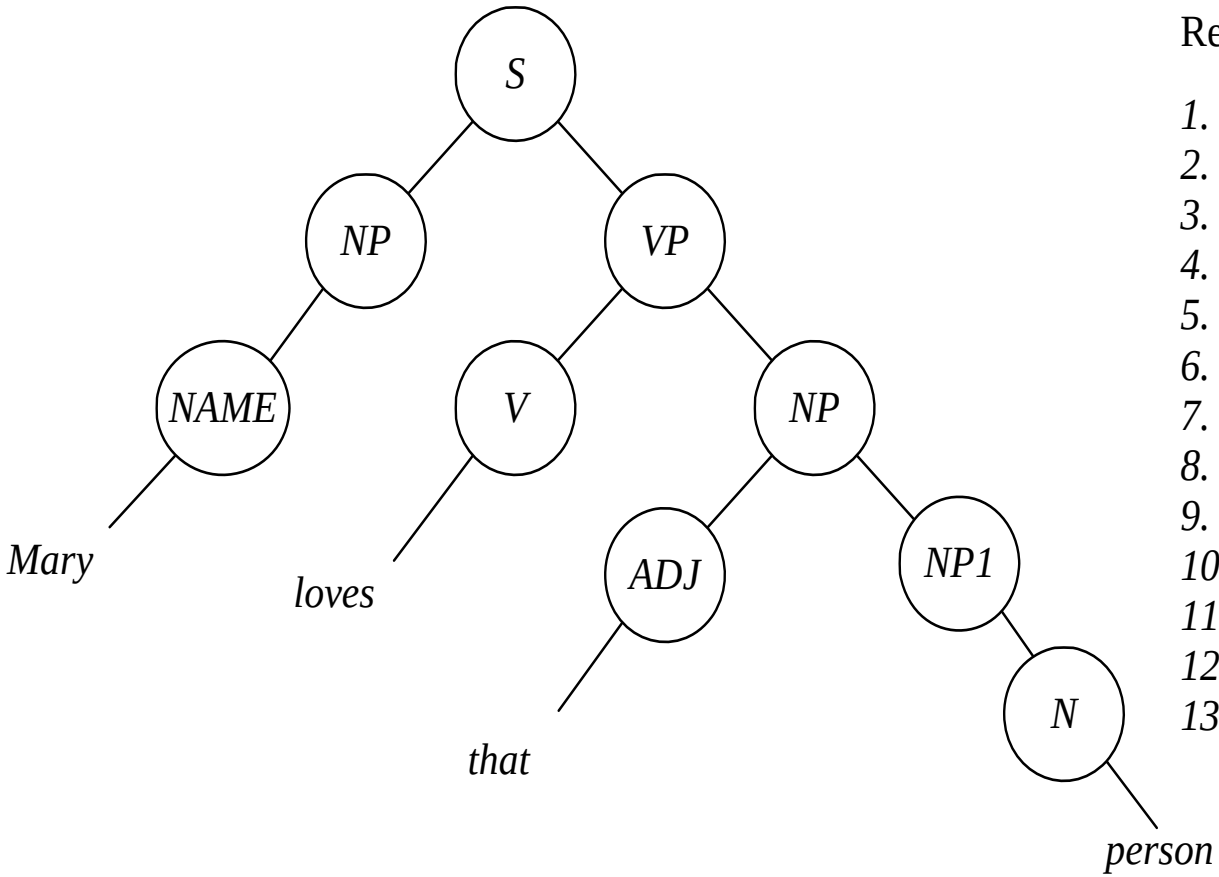
$$D=\{\alpha\rightarrow\beta\gamma,\alpha\rightarrow z\}\quad \alpha,\beta,\gamma\in I\quad z\in A$$

  - *Type 3: finite state* or *regular*

$$D=\{\alpha\rightarrow z\beta,\alpha\rightarrow z\}\quad \alpha,\beta\in I\quad z\in A$$

- *Chomsky Normal Form* (CNF)
  - A context-free language can be replaced by another language in CNF

Derivation Sequence: An Example

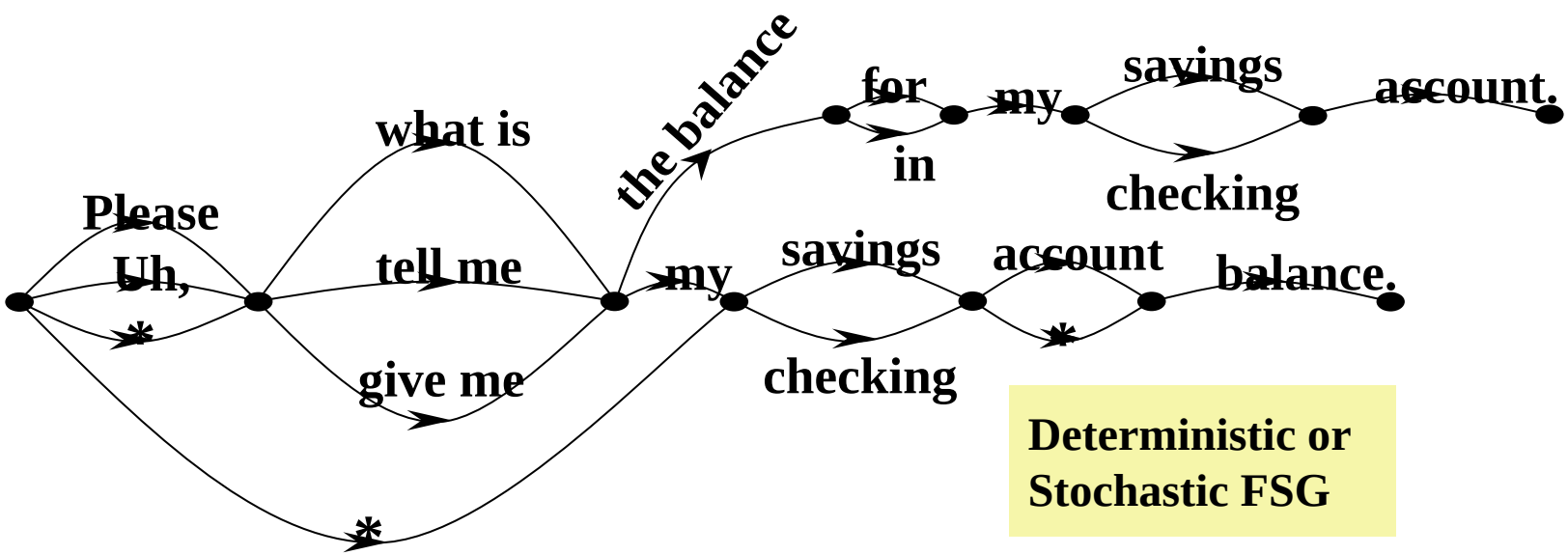


Rewrite Rules:

1. $S \rightarrow NP\ VP$
2. $VP \rightarrow V\ NP$
3. $VP \rightarrow AUX\ VP$
4. $NP \rightarrow ART\ NP1$
5. $NP \rightarrow ADJ\ NP1$
6. $NP1 \rightarrow ADJ\ NP1$
7. $NP1 \rightarrow N$
8. $NP \rightarrow NAME$
9. $NP \rightarrow PRON$
10. $NAME \rightarrow Mary$
11. $V \rightarrow loves$
12. $ADJ \rightarrow that$
13. $N \rightarrow person$

FSG: Specified by Terminal Symbols

- Word → word sequence → beyond
- Syntax model: a huge network to represent all possible and valid word sequences (e.g. |V|=60K)
  - Finite state network (FSN) approximation of word constraints
  - Deterministic or stochastic finite state grammar (FSG)



FSG: Specified by Non-Terminal Symbols

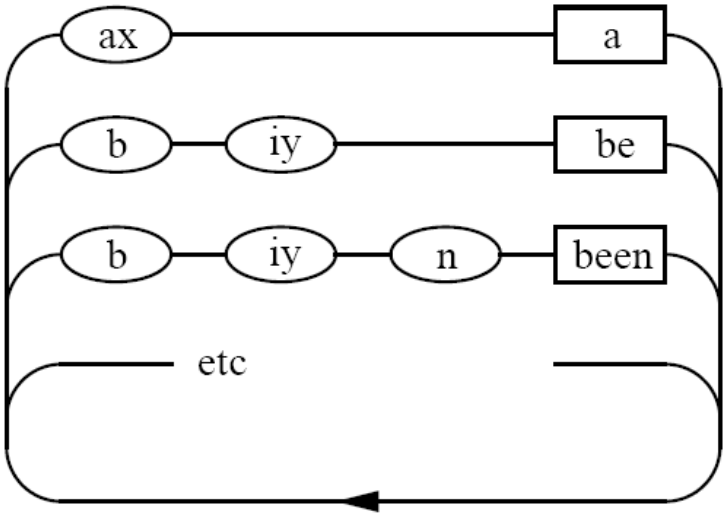
- How to pronounce a six digit sequence?
  - Alphabet terminal set: *A* = {*one, two, … , ten, eleven, … , twenty, … , ninety, hundred, thousand*}
  - Non-terminal (intermediate) set: *I* = {*digit6, digit3, digit2, digit1, teens, tys*}

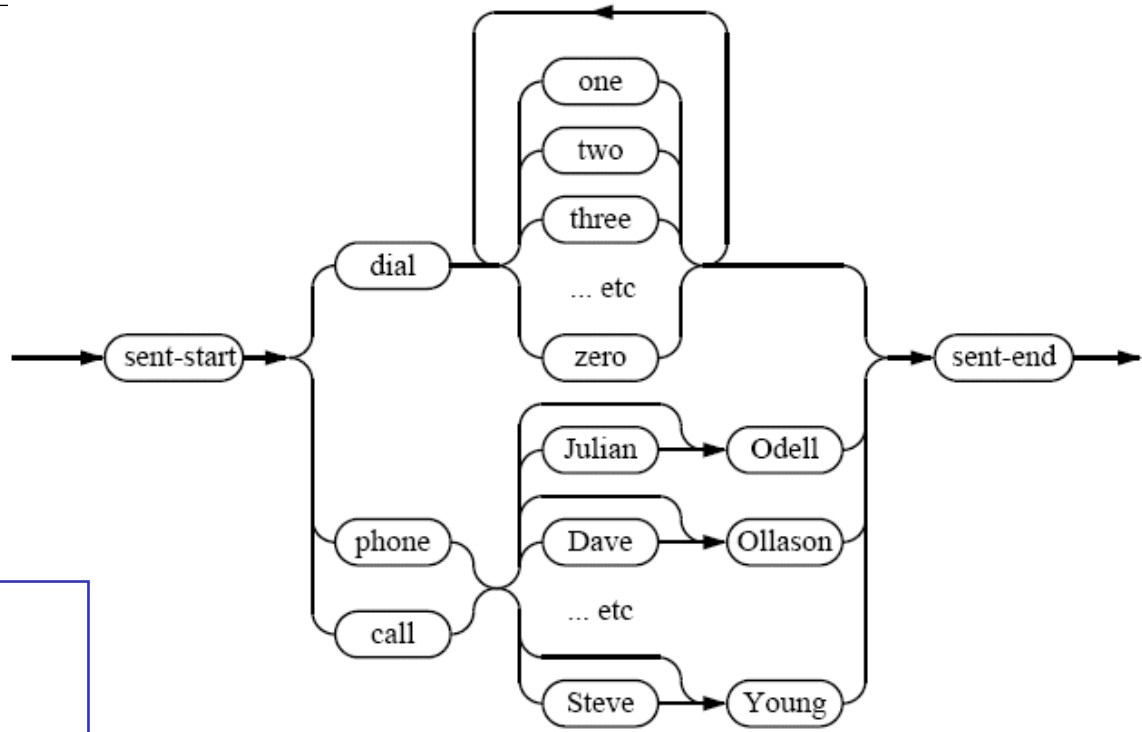
FSG Derivation Rules with Non-Terminals)

- 8 Rewrite (Derivation) rules with root *S* = *digit6*

$$D = \left\{ \begin{array}{c} digit\ 6 \rightarrow digit\ 3 \quad thousand \quad digit\ 3 \\ digit\ 6 \rightarrow digit\ 3 \quad thousand \quad OR \quad digit\ 3 \\ digit\ 3 \rightarrow digit\ 1 \quad hundred \quad digit\ 2 \\ digit\ 3 \rightarrow digit\ 1 \quad hundred \quad OR \quad digit\ 2 \\ digit\ 2 \rightarrow teens \quad OR \quad tys \quad OR \quad tys \quad digit\ 1 \quad OR \quad digit\ 1 \\ digit\ 1 \rightarrow one \quad OR \quad two \quad OR \quad \ldots \quad OR \quad nine \\ teens \rightarrow ten \quad OR \quad eleven \quad OR \quad \ldots \quad OR \quad nineteen \\ tys \rightarrow twenty \quad OR \quad thirty \quad OR \quad \ldots \quad OR \quad ninety \end{array} \right\}$$

Other Examples of Grammar Network





Word-loop grammar:

- For all possible sentences.
- Each branch represents a word in vocabulary
- May add transition proba-bilities from language models

Grammar for Voice Dialing

Part of Speech Tagging

- The problem of selecting the most likely sequence of lexical categories for the words in a sentence is known as **part-of-speech tagging**
  - Tagging is a case of limited syntactic disambiguation: many words have more than one syntactic category
  - Tagging has limited scope: we just fix the syntactic categories of words and do not do a complete parse
- Tagging is a limited but useful application.
  - Partial (shallow) parsing
  - Information extraction
  - Information retrieval
  - Question answering
  - Input to statistical parser
  - Input to a regular parser (reduces the running time)

Penn Treebank Tag Set

1. CC	Coordinating conjunction	19. PP$	Possessive pronoun
2. CD	Cardinal number	20. RB	Adverb
3. DT	Determiner	21. RBR	Comparative adverb
4 EX	Existential there	22. RBS	Superlative Adverb
5. FW	Foreign word	23. RP	Particle
6. IN	Preposition / subord. conj	24. SYM	Symbol (math or scientific)
7. JJ	Adjective	25. TO	to
8. JJR	Comparative adjective	26. UH	Interjection
9. JJS	Superlative adjective	27. VB	Verb, base form
10. LS	List item marker	28. VBD	Verb, past tense
11. MD	Modal	29. VBG	Verb, gerund/pres. participle
12. NN	Noun, singular or mass	30. VBN	Verb, past participle
13. NNS	Noun, plural	31. VBP	Verb, non-3s, present
14. NNP	Proper noun, singular	32. VBZ	Verb, 3s, present
15. NNPS	Proper noun, plural	33. WDT	Wh-determiner
16. PDT	Predeterminer	34. WP	Wh-pronoun
17. POS	Possessive ending	35. WPZ	Possessive wh-pronoun
18. PRP	Personal pronoun	36. WRB	Wh-adverb

Tagging Example

- Given an input: *The representative put chairs on the table*
- Determine a single tag for each word in the sentence given the Penn Treebank tag set
- *The*/DT *representative*/NN *put*/VBD *chairs*/NNS *on*/IN *the*/DT *table*/NN *./*
- Most words have more than one tag, so in tagging we try to assign the most likely tag to each word in a sentence

Baseline Tagging Accuracy

- In general, if you always select the most likely part of speech for a word in the training data, it is easy to obtain a 90% success rate largely because approximately 50% of the words in a typical corpus are not ambiguous. Hence, this is a baseline against which a good model must measure itself.
- It is possible to do better than 90% accuracy by using more information from the sentence a word appears in. For example, if a word follows a determiner, it is not likely that it is a verb.

Information Sources

- Syntagmatic: look at the tags assigned to nearby words; some combinations are highly likely while others are highly unlikely or impossible
  - DT JJ NN
  - DT JJ VBP
- Lexical: look at the word itself. (90% accuracy just by picking the most likely tag for each word)
  - *Verb* is more likely to be a noun than a verb

Early Approaches

- Two stage process:
  - Lookup the word in the dictionary and assign each word the list of possible tags
  - Use a set of handwritten rules to select tag given
    - word forms: context and current position
    - tags (individual or sets): context and current position
    - Example: $DT_{eq,-1,Tag}$ Þ NN
- Tagging by Parsing: Assign all tags, and let the parsing process keep the tags that are allowed. Mixed results

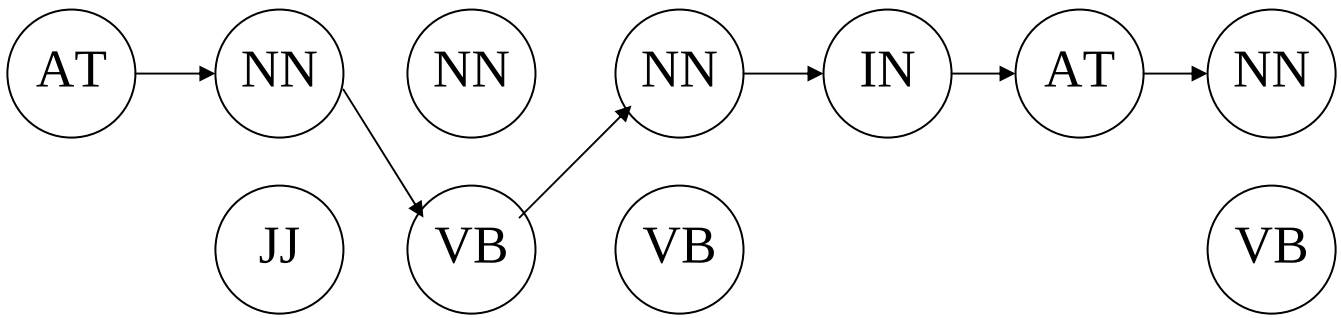
Statistical Approaches (Overview)

- Probabilistic:
  - HMM
    - Merialdo and many more (XLT)
  - Maximum Entropy
    - DellaPietra et al., Ratnaparkhi, and others
  - Neural Networks
- Rule-based:
  - TBL (Transformation Based Learning)
    - Brill's tagger
  - Decision Trees
  - Example-based (memory-based with K-nearest neighbors)
    - Daelemans, Zavrel, others
- Feature-based (inflective languages)
- Classifier Combination (Brill)

Problem Mapping of POS Tagging

- Finite state network (FSN) representation
  - State (node) space: the set of tags
  - Arc: tag transition (probabilities)
  - State output: tag-specific word probabilities
  - State-sequence: tag sequence
- An example:

The representative put chairs on the table.



Statistical POS Tagging

$$\hat{T} = \arg\max_{T \in \Psi} P(T \mid W)$$
$$= \arg\max_{T \in \Psi} P(W \mid T) P(T)$$

P(W|T): tag-specific word LM

P(T): tag language model

- Bigram tag language model approximation

$$P(T) = P(t_1^Q) \approx \prod_{q=1}^{Q} P(t_q \mid t_{q-1}) \quad P(t_1 \mid t_0) = 1$$

- Localized tag-specific language model

$$P(W \mid T) = P(w_1^Q \mid t_1^Q) \approx \prod_{q=1}^{Q} P(w_q \mid t_1^n) \approx \prod_{q=1}^{Q} P(w_q \mid t_q)$$

- Overall approximation

$$\hat{t}_1^Q = \arg\max_T P(W \mid T) P(T) \approx \arg\max_{t_1^Q} \prod_{q=1}^{Q} P(w_q \mid t_q) P(t_q \mid t_{q-1})$$

Notation

w_i: word at position *i*

t_i: tag assigned to word at position *i*

$w_{i,i+m}$: words at positions *i* though *i+m*

w^l: l^{th} word in the lexicon

t^j: j^{th} tag in the tag set

C(*): the number of occurrences of * in the training data

N: number of tags in the tag set

M: number of words in the lexicon

n: sentence length

A Tagging HMM Model

- *N* states corresponding to tags, or (n-1)-tuples of tags if n-gram tag model is used: t_1, t_2, …,t_N
- *M* word observations
- *B*: *N* × *M* matrix; for each state, there is a probability density function for each word.
  - $b_{ij}=\{W=w^j \mid T = t^i\}$
  - output (words) emitted by states
- *A*: *N* × *N* matrix; transition probability distribution (between tags)
- Π: *N*-dim vector; initial sate probability distribution (where do we start?)

The Equations

- The goal is to assign lexical tags to a sequence of words $w_{1,n}$ such that the probability of the assignment is maximized.
- $w_{1,n}$ is a sequence of words (in a sentence of length n) to assign a sequence of lexical tags $t_{1,n}$ that maximizes $P(t_{1,n} \mid w_{1,n})$.
- Use $\text{argmax}_{t_{1,n}} P(t_{1,n} \mid w_{1,n}) =$

  $\text{argmax}_{t_{1,n}} [P(t_{1,n}) \times P(w_{1,n} \mid t_{1,n})] / P(w_{1,n})$
- Since the goal is to find the $t_{1,n}$ that maximizes the probability, the denominator can be ignored:

  $\text{argmax}_{t_{1,n}} P(t_{1,n}) \times P(w_{1,n} \mid t_{1,n})$

The Equations (Cont.)

- Two probabilities to calculate:
  - $P(t_{1,n}) = P_{i=1..n}\, p(t_i|t_1,\ldots,t_{i-1})$
  - $P(w_{1,n} \mid t_{1,n}) = P_{i=1..n}\, p(w_i|w_1,\ldots,w_{i-1},t_1,\ldots,t_i)$
- Too many parameters to estimate.
- Approximations based on the following assumptions:
  - Tag depends on limited history:
    - $p(t_i|t_1,\ldots,t_{i-1}) \Rightarrow p(t_i|t_{i-n+1},\ldots,t_{i-1})$
  - Words do not depend on the context, only on the current tag:
    - $p(w_i|w_1,\ldots,w_{i-1},t_1,\ldots,t_i) \Rightarrow p(w_i|t_i)$

The Equations (Cont.)

- Consider $P(t_{1,n})$. It can be approximated by using probabilities that are **based on a limited number of prior tags**. Recall that an n-gram model looks at the probability of a tag for the current word given the previous n-1 tags
  - A **bigram** model looks at pairs of tags: $P(t_i|t_{i-1})$.
  - A **trigram** model looks at triples of tags: $P(t_i|t_{i-2},t_{i-1})$.
- Using bigrams: $P(t_{1,n}) \approx \prod_{i=1,n} P(t_i|t_{i-1})$
- To account for the beginning of the sentence, we can create a pseudo-tag ø at position 0 as the value of t_0. For example:

$$P(\text{ART N V N}) \approx P(\text{ART}|ø) \times P(\text{N}|\text{ART}) \times P(\text{V}|\text{N}) \times P(\text{N}|\text{V})$$

The Equations (Cont.)

- Consider $P(w_{1,n}|t_{1,n})$. If we assume that a word appears with a tag independently of the words of the surrounding tags, then we obtain the following: $P(w_{1,n}|t_{1,n}) \approx \prod_{i=1,n} P(w_i|t_i)$
- Combining the two pieces, we obtain the following equation: $\prod_{i=1,n} P(t_i|t_{i-1}) \times P(w_i|t_i)$
- The advantage of this equation is that the probabilities are easily obtained from a corpus labeled with tags

A Markov Chain Model for PoS Sequences

- Markov Chain Model for language modeling
  - Each tag is a Markov state, a total *N* states (vocabulary size)
  - A set of tag transition probability
- Given any a sentence: convert it to tag sequences (WordNet)

  W = "I want to fly from Atlanta to Toronto tomorrow."

  T="PN VB TO VB IN NN IN NN RB."
- 1st-order Markov model: *N*N* conditional probabilities

  $Pr(T) = p(PN|begin)*p(VB|PN)*p(TO|VB)*p(VB|TO)$ …
- 2nd-order Markov model: *N*N*N*

  $Pr(T) = p(PN|begin)*p(VB|PN,begin)*p(TO|VB,PN)*p(VB|TO,VB)*$ …
- *N*-th model: a large number of probabilities to be estimated

Supervised Learning (with Annotated Data)

- Use MLE

  $p(w_i|t_i) = c_{wt}(t_i,w_i) / c_t(t_i)$

  $p(t_i|t_{i-n+1},\ldots,t_{i-1}) = c_{tn}(t_{i-n+1},\ldots,t_{i-1},t_i) / c_{t(n-1)}(t_{i-n+1},\ldots,t_{i-1})$

- Smooth (both!)

  $p(w_i|t_i)$: "Add 1" for all possible tag,word pairs using a predefined dictionary (thus some 0 kept!)

  $p(t_i|t_{i-n+1},\ldots,t_{i-1})$: linear interpolation: e.g. for trigram model:

  $$p'_l(t_i|t_{i-2},t_{i-1}) = l_3\, p(t_i|t_{i-2},t_{i-1}) + l_2\, p(t_i|t_{i-1}) + l_1\, p(t_i) + l_0 / |V_T|$$

Estimating Probabilities

- To estimate the needed probabilities, we must use large samples of training data (independent of the test data)
- Derive from a training corpus the ratio of the total number of occurrences of an event to the total number of occurrences of all events of that type and use it as the probability. This simple ratio is known as the **maximum likelihood estimator** (MLE). For example, if we see 100 occurrences of the word "foo" in the corpus, with 60 as nouns, 20 as verbs, and 20 as adjectives, then an MLE of Pr(N|foo) is 0.6

Estimating Probabilities (Cont.)

- The accuracy grows as the amount of data grows
- Unfortunately, there are many estimates needed and sparse data for less likely events presents a problem. For example, in the Brown corpus, some 40,000 words appear 5 time or less, and so part of speech estimates for those words could be quite inaccurate
- In the worst case, a low frequency word does not appear in the corpus at all. These rare events are common in NLP applications
- We have already investigated methods that address the problem of zero probabilities while retaining the relative likelihoods for occurring items, namely smoothing

Obtaining Probabilities

Syntagmatic Probabilities:

For all tags t^j do

  For all tags t^k do

    $P(t^k \mid t^j)=C(t^j, t^k)/C(t^j)$

  End

End

Lexical Probabilities:

For all tags t^j do

  For all words w^l do

    $P(w^l \mid t^j)=C(w^l, t^j)/C(t^j)$

  End

End

Estimation of Syntagmatic Probabilities

- The syntagmatic bigrams, $P(t_i|t_{i-1})$, are estimated by counting the number of times each pair of tags occurs compared to individual tag counts. For example, the probability that a V follows an N is estimated as follows:

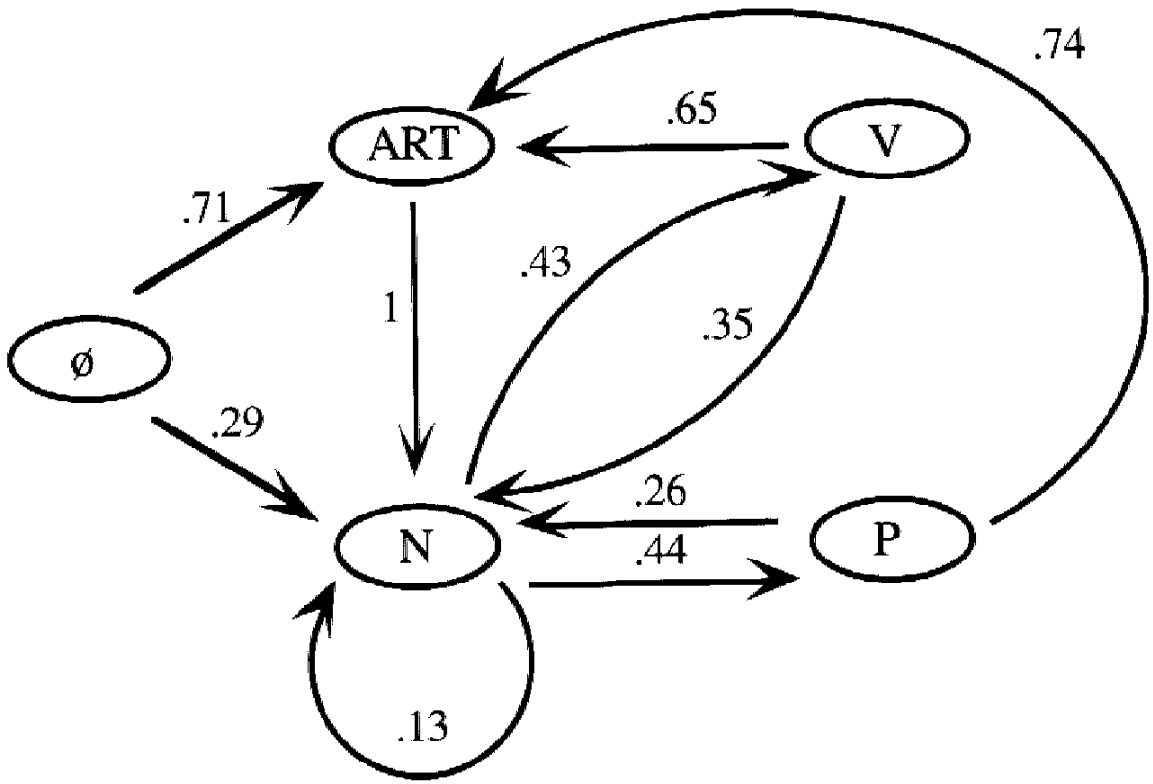
  $P(t_i=V|t_{i-1}=N) = C(N @ POS_{i-1} \ \& \ V @ POS_i) \mid C(N @ POS_{i-1})$

- The bigram probabilities for an artificially generated corpus of sentences containing only four tags (i.e., N, V, Art, and P) are shown on the next slide. There are 1,988 words in the corpus: 833 N, 300 V, 558 Art, and 307 P. To deal with sparse data in our example, any bigram not listed will assume a token probability of .0001

Syntagmatic Probability Estimates

Category	Count at i	Pair	Count at i, i+l	Bigram	Estimate
ø	300	ø, ART	213	P(ART|ø,)	.71
ø	300	ø, N	87	P(N|ø,)	.29
ART	558	ART, N	558	P(N|ART)	1
N	833	N, V	358	P(V|N)	.43
N	833	N, N	108	P(N|N)	.13
N	833	N, P	366	P(P|N)	.44
V	300	V, N	75	P(N|V)	.35
V	300	V, ART	194	P(ART|V)	.65
P	307	P, ART	226	P(ART|P)	.74
P	307	P, N	81	P(N|P)	.26

State (Tag) Transition Probabilities



Estimation of Lexical Probabilities

- $P(w_i|t_i)$ can be estimated by simply counting the number of occurrences of each word by tag and dividing by the no. of occurrences of the tag. For example,

  P(the|ART) = C(# times *the* is an ART)/ C(# times an ART occurs)

- The table on the next slide gives some counts of the number of co-occurrences of each word and tag

Word/Tag Counts

	N	V	ART	P	TOTAL
flies	21	23	0	0	44
fruit	49	5	1	0	55
like	10	30	0	21	61
a	1	0	201	0	202
the	1	0	300	2	303
flower	53	15	0	0	68
flowers	42	16	0	0	58
birds	64	1	0	0	65
others	592	210	56	284	1142
TOTAL	833	300	558	307	1998

Lexical Probability Estimates (Cont.)

The table below gives the lexical probabilities which are needed for our example:

P(the|ART)	.54	P(a|ART)	.360
P(flies|N)	.025	P(a|N)	.001
P(flies|V)	.076	P(flower|N)	.063
P(like|V)	.1	P(flower|V)	.05
P(like|P)	.068	P(birds|N)	.076
P(like|N)	.012		

Note that P(the|ART) = .54 is quite different than P(ART|the) = 300|303

The HMM Tagger

- The Markov Chain on slide 21 can be extended to include the lexical output probabilities (i.e., P(w|t)). For example, node N in the network would be associated with a probability table that indicates for each word how likely that word is to be selected if we randomly selected an N
- It is no longer trivial to compute the probability of a sequence of words from the network. However, if we are given a particular sequence, the probability that it generates a particular output is easily computed by multiplying together:
  1. the probabilities on the path
  2. the probability of each output

The Probability of a State Sequence

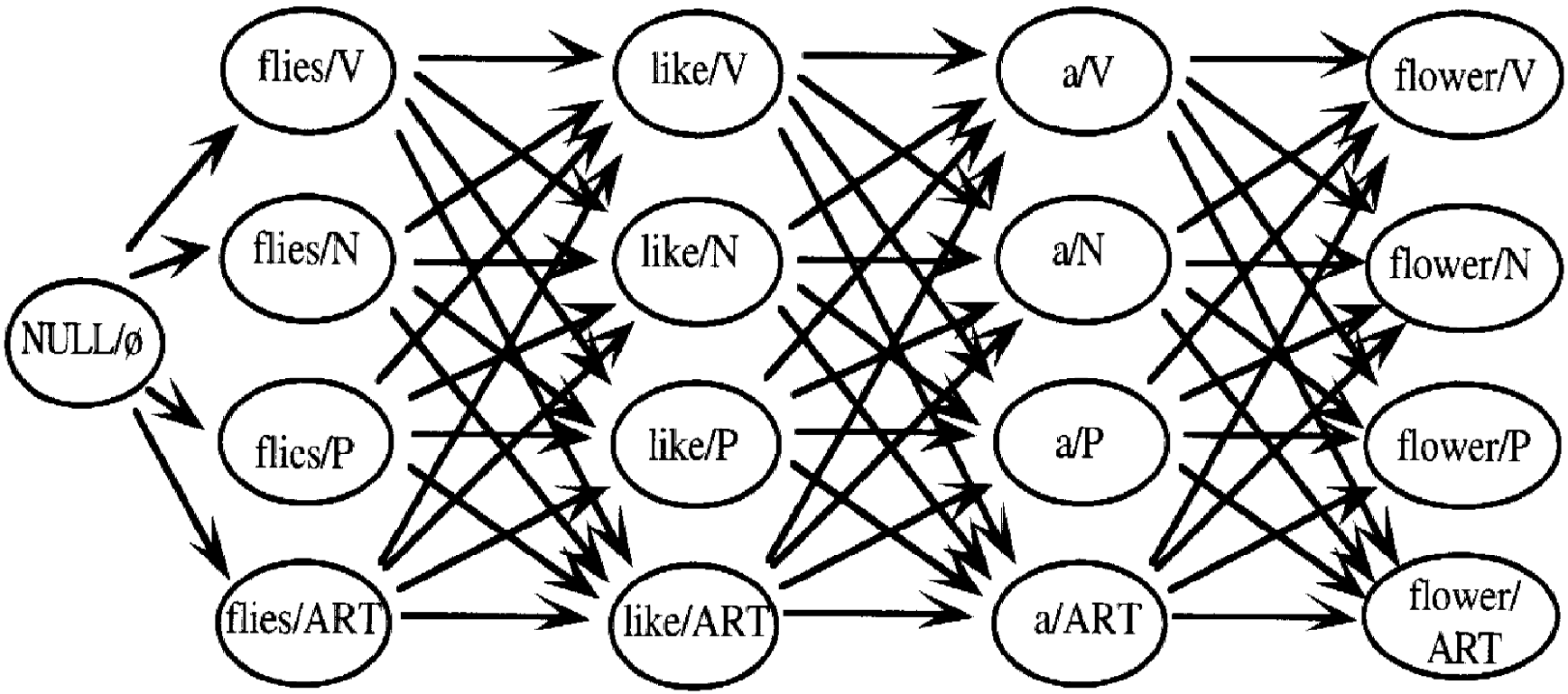
- For example, the probability that the sequence: N V ART N generates the output "Flies like a flower" is computed by determining:
  - Probability that the output is "Flies like a flower" by using the lexical output probabilities: P(flies|N) × P(like|V) × P(a|Art) × P(flower|N)=.025*.1*.36*.063= $5.4*10^{-5}$
  - P(N V ART N) given the Markov model: P(N V ART N) = P(N|Ø) × P(V|N) ×P(ART|V) ×P(N|ART)= .29 × .43 × .65 × 1 = .081
  - Multiply the above two numbers together to give the likelihood that the HMM would generate the sequence

Efficient Tagging

- How to find the most likely sequence of tags for a sequence of words uses the key insight that you don't need to enumerate all possible sequences. In fact, sequences that end in the same tag can be collapsed together since the next tag depends only on the current tag in the sequence
- Given the contextual and lexical estimates, we can use the Viterbi algorithm to avoid using the brute force method, which for N tags and T words examines N^T sequences

An Example

For "Flies like a flower", there are four words and four possible tags, giving 256 sequences depicted below. In a brute force method, all of them would be examined.



Important Insight

- To find the most likely sequence, we can sweep forward though the words, one at a time, finding the most likely sequence for each ending tag. That is, you first find the four best sequences for the two words "flies like", one ending in V, one in N, one in Art, and one in P
- This information is then used to find the four best sequences for the three words "flies like a", each ending in a different tag. The process continues until all of the words are accounted for
- This is the way the **Viterbi algorithm**, which is given on the next page, works. If there are n words to be tagged and N tags, it has a running time of $O(KnN^2)$ given a constant K

Viterbi Notation

- To track the probability of the best sequence leading to each possible tag at each position, the algorithm uses δ, an N×n array, where N is the number of tags and n is the number of words in the sentence. $\delta_t(t^i)$ records the probability of the best sequence up to position *t* that ends with the tag, t^i
- To record the actual best sequence, it suffices to record only the one preceding tag for each tag and position. Hence, another array γ, an N×n array, is used. $\gamma_t(t^i)$ indicates for the tag t^i in position *t* which tag at position *t-1* is in the best sequence

Viterbi Algorithm

- Given the word sequence $w_{1,n}$, the lexical tags $t^{1,N}$, the lexical probabilities $P(w_t|t_t)$, and the bigram probabilities $P(t^i|t^j)$, find the most likely sequence of lexical tags for the word sequence

Initialization Step:

For i= 1 to N do // For all tag states $t^{1,N}$

$\delta_1(t^i) = P(w_1|t^i) \times P(t^i|\text{ø})$

$\gamma_1(t^i) = 0$ // Starting point

Viterbi Algorithm

Iteration Step:

For f=2 to n // next word index

  For i= 1 to N // tag states $t^{1,N}$

$\delta_f(t^i) = \max_{j=1,N} (\delta_{f-1}(t^j) \times P(t^i \mid t^j)) \times P(w_f \mid t^i))$

$\gamma_f(t^i) = \text{argmax}_{j=1,N} (\delta_{f-1}(t^j) \times P(t^i \mid t^j)) \times P(w_f \mid t^i))$ //index that gave max

Sequence Identification Step:

$X_n = \text{argmax}_{j=1,N}\, \delta_n(t^j)$ // Get the best ending tag state for w_n

For i = n-1 to 1 do // Get the rest

  $X_i = \gamma_{i+1}(X^{i+1})$ // Use the back pointer from subsequent state

$P(X_1,\ldots, X_n) = \max_{j=1,N} \delta_n(t^j)$

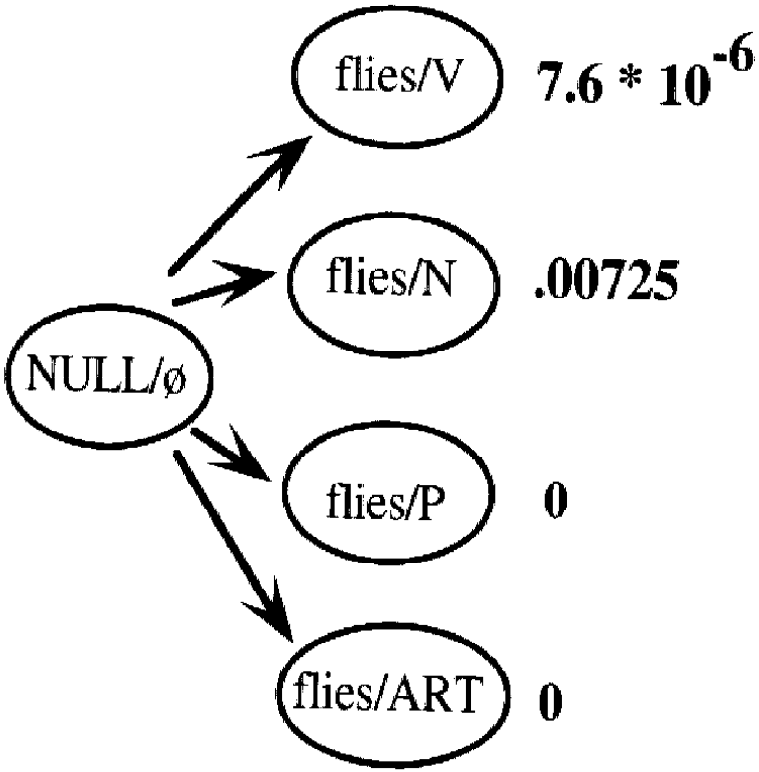
Example

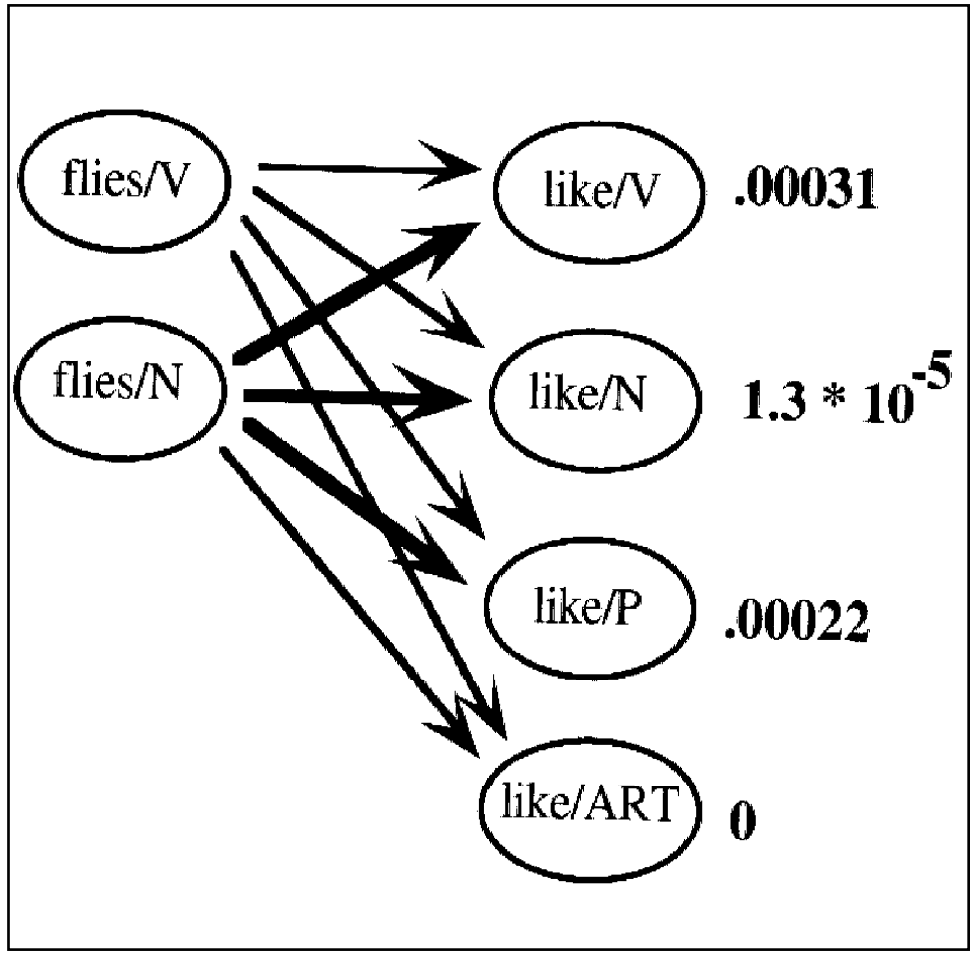
- Consider an example which uses the transition and lexical probabilities, assuming that any transition probability which is not given in the figure on slide 21 receives a probability of .0001. Consider how the algorithm works with the sentence, *Flies like a flower*The initialization phase uses the formula:

  $\delta_1(t^i) = P(\text{Flies} \mid t^i) \times P(t^i \mid \text{ø})$ where t^i ranges over four tags

- Because of the lexical probabilities, only entries for an N and V are greater than 0. Thus the most likely sequences are shown on the left-hand side of the figure on the next slide ($\delta_1(V)=.076\times.0001= .0000076$ and $\delta_1(N)=.29\times.025=.00725$)

Example (Cont.)





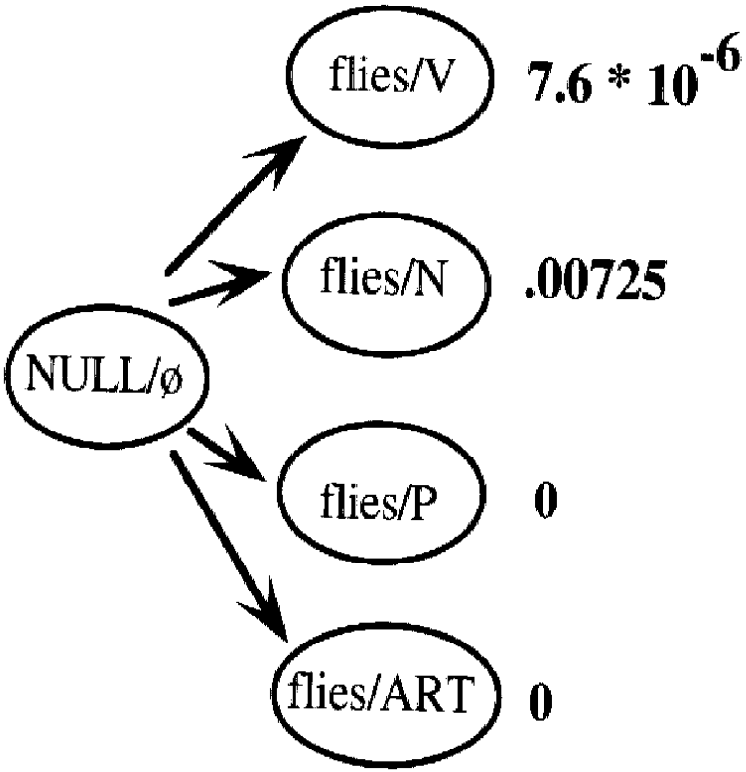
Example (Cont.)

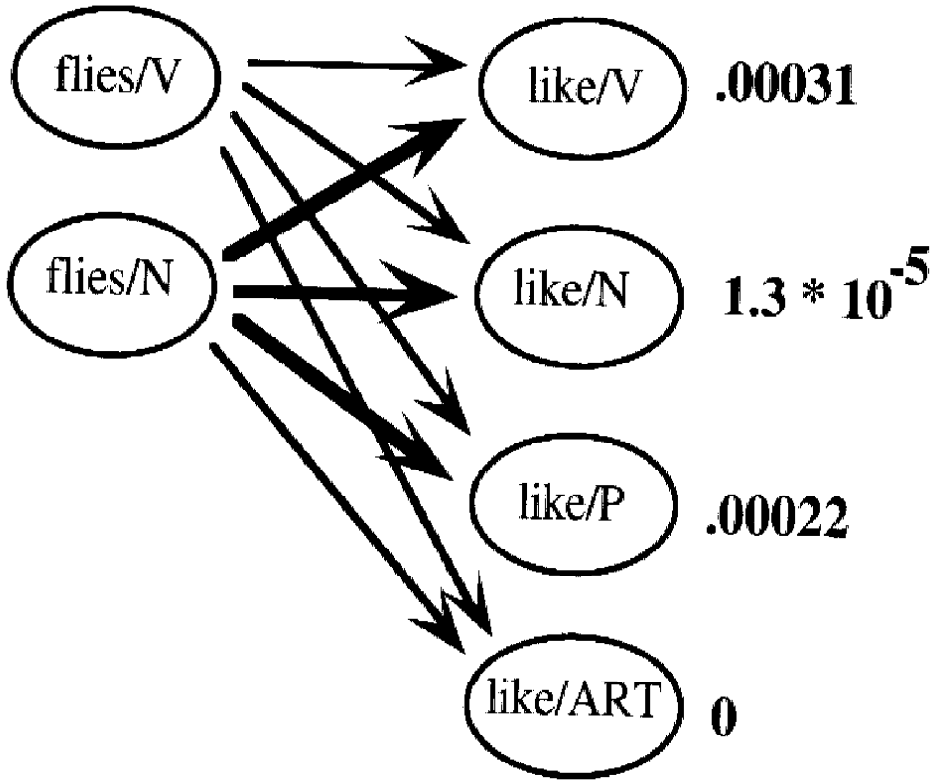
- The first iteration of the second phase of the algorithm is shown on right-hand side of the next figure. It extends the sequences one word at a time, keeping track of the best sequence found so far to each tag. For example:

$$\delta_2(V) = \max(\delta_1(N) \times P(V|N), \delta_1(V) \times P(V|V)) \times P(\text{like}|V)$$
$$= \max(.00725 \times .43, 7.6 \times 10^{-6} \times .0001) \times .1$$
$$= 3.12 \times 10^{-4}$$

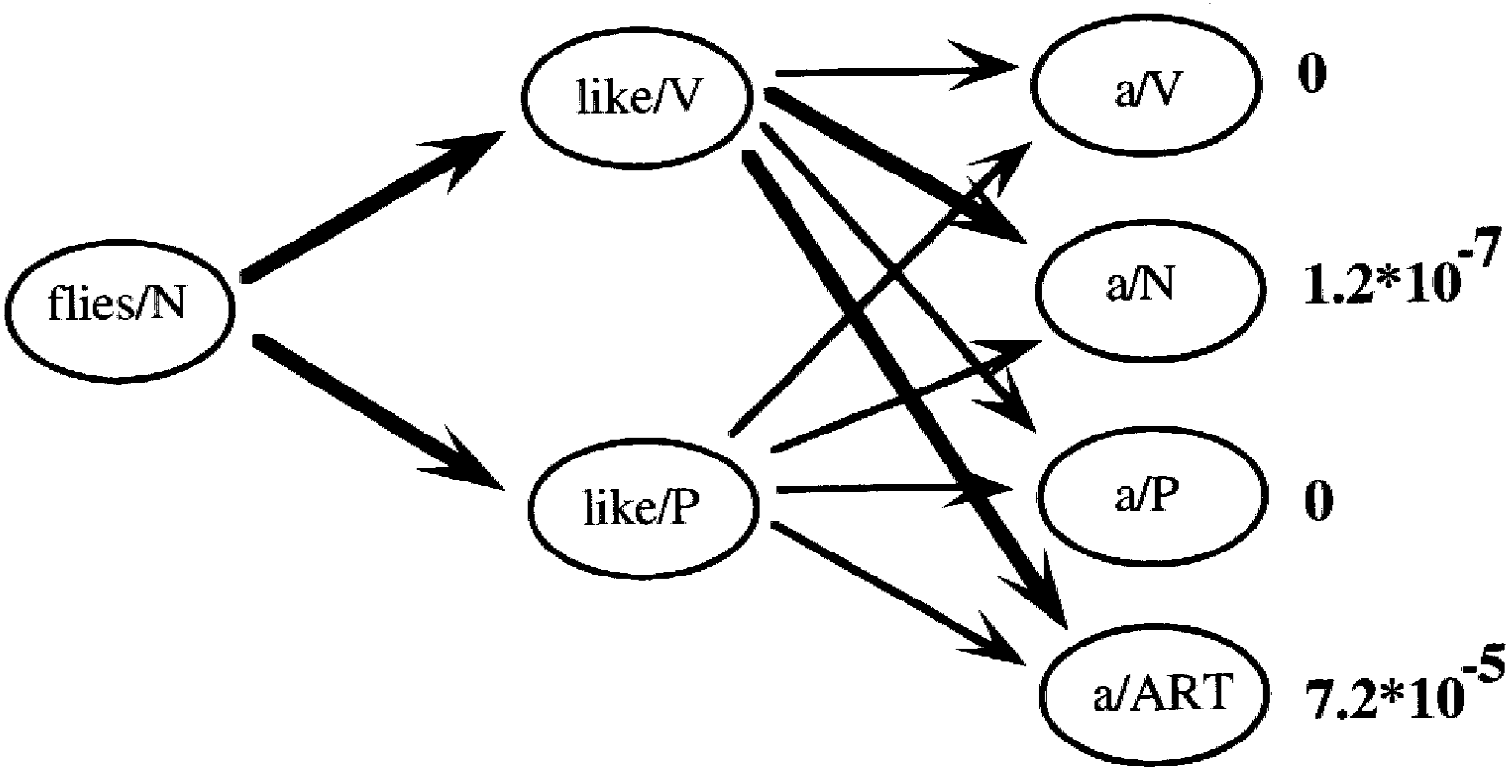
- Note that the heavy arrows indicate the best sequence leading up to each node

Example (Cont.)

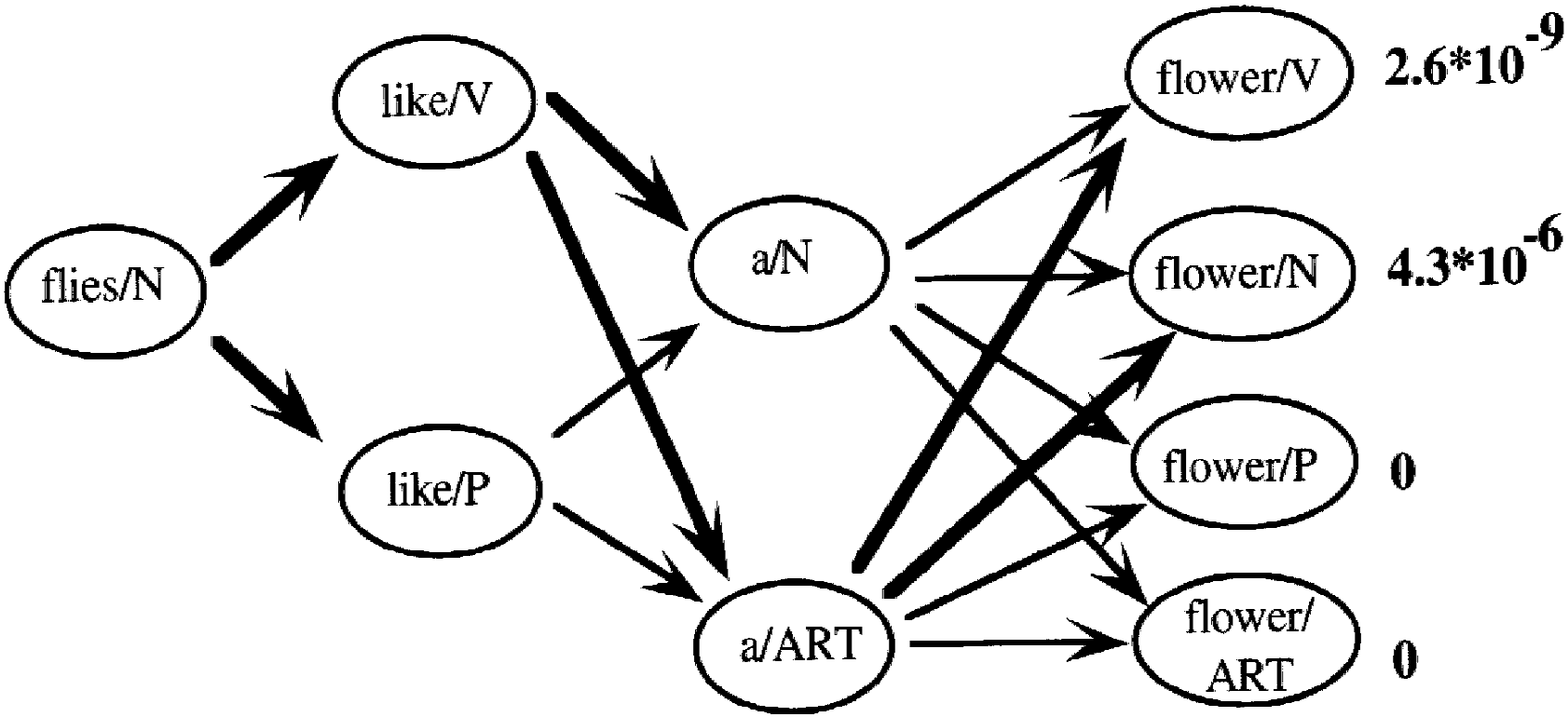




Second Iteration Step



Final Iteration



Finishing Up

- The highest probability sequence ends in flower|N. It is simple to trace back from this node to get the full sequence of N V ART N
- This approach can yield 95% or better accuracy when using trigram context models

HMM Tagger Issues

- Unknown Words: Words that do not appear in the training corpus (or dictionary) need to be handled by the tagger for robustness
  - Assume that the word can have any tag with a distribution obtained from the entire corpus
  - Limit to open class words
  - Use Morphology (word affixes)
  - Capitalization, hyphenation, affixes

HMM Issues

- Traditional Trigram Taggers:
  - A becomes an N×N×N matrix: a_{ijk} is the probability that t^k follows a history of t^i t^j
- Smoothing: important especially for lexical probabilities
  - Add-one method (using a predefined dictionary, so with 0s)
- Linear interpolation of lower order models allows us to cope with the fact that certain contexts (e.g., the comma) make longer histories less desirable
- Variable Memory Markov Models: history depends on the sequence instead of a fixed weighted sum
- Can add context to lexical probabilities: $P(w_i|t_{i-1},t_i)$
  - Smoothing becomes more important than ever in this case

HMM Issues

- Decoding can occur in a right-to-left manner instead of a left-to-right manner (Church 1988)
  - $P(t_{1,n})=P(t_n)\times P(t_{n-1,n}|t_n)\times P(t_{n-2,n-1}|t_{n-1})\times \ldots \times P(t_{1,2}|t_2)$
- The Viterbi algorithm maximizes the probability of the state sequence, i.e., $P(t_{1,n} \mid w_{1,n})$ maximized); however, one might also maximize $P(t_i|w_{1,n})$ for all i (which amounts to summing over different tag sequences). It can produce incoherent sequences, so this is usually not done

HMM Issues

- Manning and Schütze call the tagger we just presented a Visible Markov Model (VMM) to distinguish it from taggers that are trained using Baum-Welch due to the fact that there is no labeled training data
- In general, it is better to use supervised training without using Baum-Welch Reestimation if there is a sufficiently large labeled corpus available

Transformation-Based Tagging

- Exploit wider range of lexical and syntactic regularities
- Condition the tags on preceding words not just preceding tags
- Use more context than bigram or trigram

Transformations

- A transformation consists of two parts, a triggering environment and a rewrite rule.

• Examples of some transformations learned in transformation-based tagging

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
VBP	VB	one of the previous two words is n't

Learning Algorithm

- The learning algorithm selects the best transformations and determines their order of application
- Initially tag each word with its most frequent tag
- Iteratively we choose the transformation that reduces the error rate most
- We stop when there is no transformation left that reduces the error rate by more than a pre-specified threshold

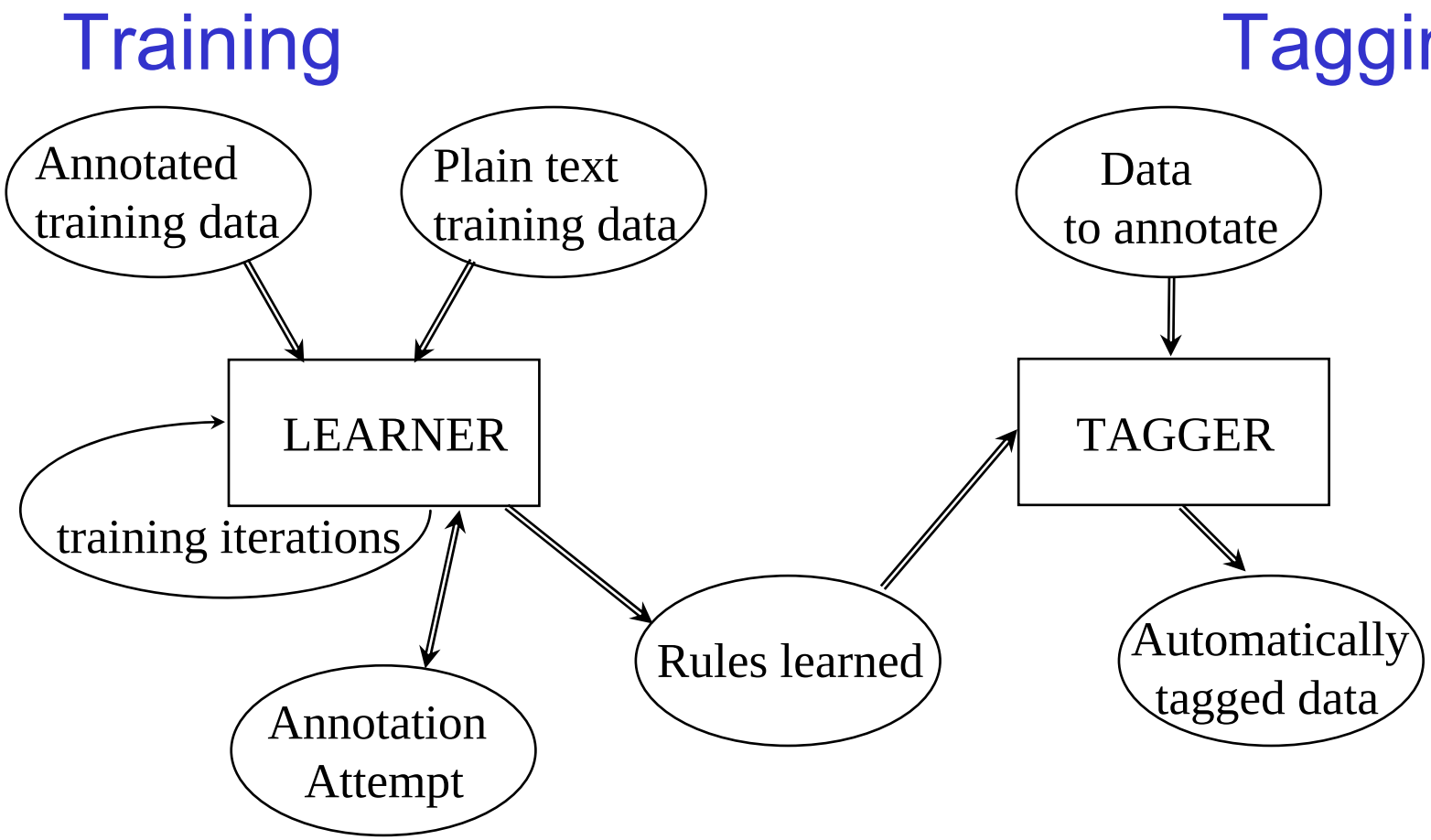
Unknown Words

- "OOV" words (out-of-vocabulary)
  - we do not have list of possible tags for them
  - and we certainly have no output probabilities
- Solutions:
  - try all tags (uniform distribution)
  - try open-class tags (uniform, unigram distribution)
  - try to "guess" possible tags (based on suffix/ending) - use different output distribution based on the ending (and/or other factors, such as capitalization)

Transformation-Based Learning (TBL)

- TBL tagging (Brill 1995) encodes complex interdependencies between words and tags by selecting and ordering transformations that move an imperfectly tagged sequence to one that has fewer errors
- TBL taggers have two components:
  - A specification of which error correcting transformations are admissible
  - A learning algorithm
- TBL facts:
  - Input: tagged corpus (ground truth) and dictionary
  - Initially tag each word in some way (using the dictionary)
  - Lists of transformations to improve the accuracy

The General Scheme



TBL Tagger

- ***Not*** a source channel view
- ***Not*** even a probabilistic model (no “numbers” used when tagging a text after a model is developed)
- It is however statistical in the sense that:
  - uses training data (combination of supervised [manually annotated data available] and unsupervised [plain text, large volume] training)
  - learning [rules]
  - criterion: accuracy (that’s what we are interested in in the end, after all!)
- Can be converted to a Finite State Transducer to gain tagging speed (Roche and Schabes, 1995)

Transformations

- A transformation consists of a triggering environment and a rewrite rule (much like a production system)
- A rewrite rule transforms a tag in context into another tag, e.g., $t_1 \rightarrow t_2$ is a rewrite rule meaning that tag 1 is replaced by tag 2
- Some of the triggering environments that Brill found important are depicted on the next slide. For example t^j occurs in one of the prior two positions. Can involve both tags and words
- Example, if one of the three previous tags is MD then VBP$\rightarrow$VB

Morphology-Triggered Rules

- Morphology (tagging environment): *change tag at position i from a to b given a condition, specified by a pool of rules (templates)*

w_{i-3}	w_{i-2}	w_{i-1}	w_i	w_{i+1}	w_{i+2}	w_{i+3}
t_{i-3}	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}	t_{i+3}

- Example:
  - w_i has suffix -ied
  - w_i has prefix re-

Notation

- C_k refers to the tagging of the corpus at iteration k
- $E(C_k)$ is the number of words mistagged in corpus C_k
- The stopping condition threshold is denoted $\in$
- The error is measured with respect to a tagged corpus (the ground truth)

TBL Learning Algorithm

- C_0 is a corpus with each word tagged (e.g., with its most frequent tag)
- For k=0 step 1 do

  v = transformation u_i that minimizes $E(u_i(C_k))$

  If $(E(C_k) - E(v(C_k))) < \in$ then break

  $C_{k+1} = v(C_k)$

  $\tau_{k+1} = v$
- Output $\tau_1, \tau_2, \tau_3 \ldots, \tau_k$

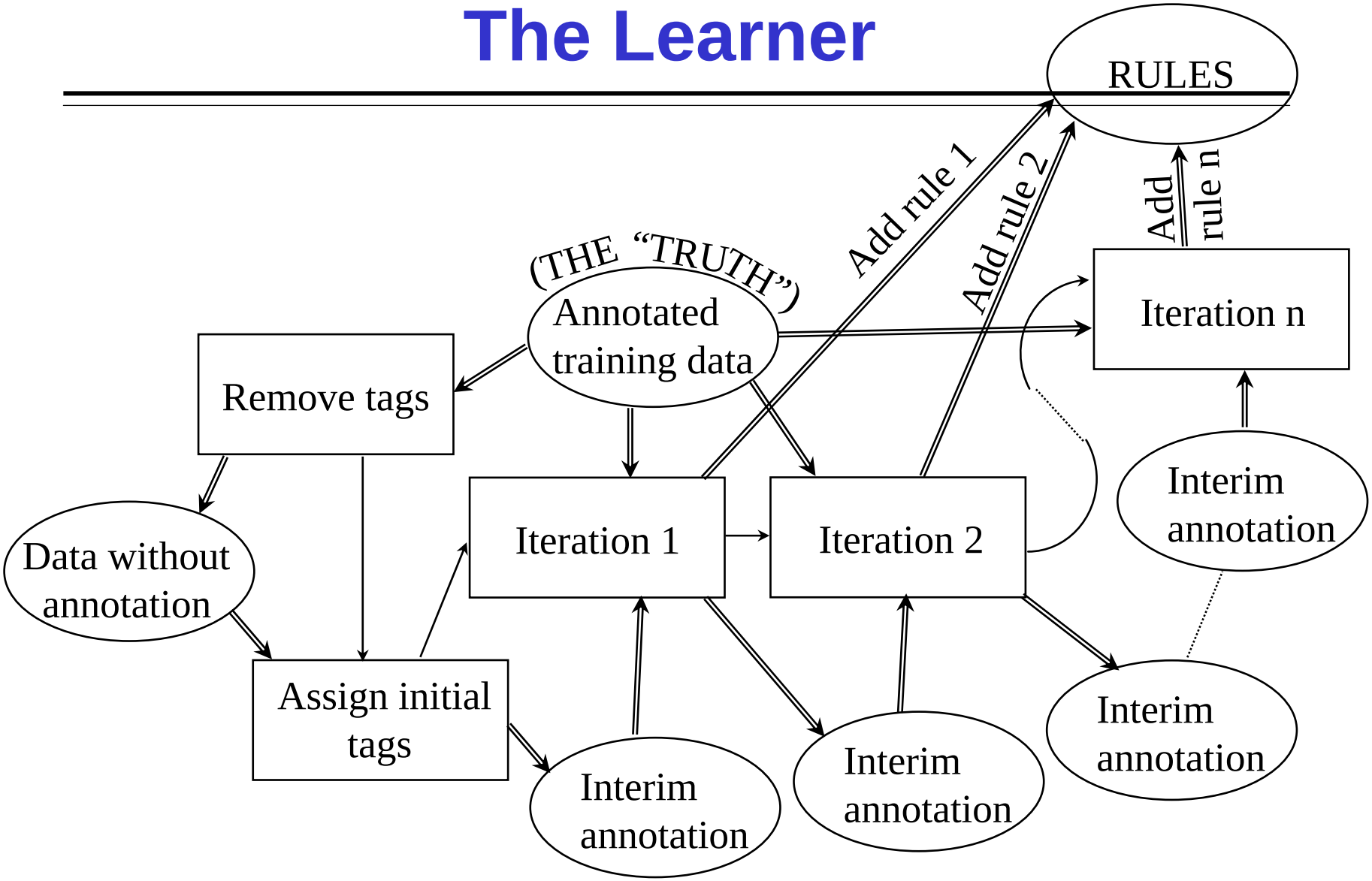
The Initial Assignment of Tags

- Several possible ways to initialize:
  - random
  - most frequent word class (NN)
  - the most frequent tag for a given word form
  - use an HMM tagger for the initial assignment
- The algorithm is not particularly sensitive to the initial assignments

The I/O of an Iteration

- Input (iteration i):
  - Intermediate data (initial or the result of previous iteration)
  - The TRUTH (the annotated training data)
  - [pool of possible rules]
- Output:
  - One rule $r_{selected(i)}$ to enhance the set of rules learned so far
  - Intermediate data (input data transformed by the rule learned in this iteration, $r_{selected(i)}$)

The Learner


RULES
Add rule 1
Add rule 2
Add rule n
(THE "TRUTH")
Annotated training data
Iteration n
Remove tags
Interim annotation
Data without annotation
Iteration 1
Iteration 2
Assign initial tags
Interim annotation
Interim annotation
Interim annotation

The Error Criterion

- Error rate:
  - beginning of an iteration: some error rate E_{in}
  - each possible rule r, when applied at every data position:
    - makes an improvement somewhere in the data ($c_{improved}(r)$)
    - makes it worse at some places ($c_{worsened}(r)$)
    - and, of course, does not touch the remaining data
- Rule contribution to the reduction of the errors:
  - $contrib(r) = c_{improved}(r) - c_{worsened}(r)$
- Rule selection at iteration i:
  - $r_{selected(i)} = argmax_r\ contrib(r)$
- New error rate: $E_{out} = E_{in} - contrib(r_{selected(i)})$

The Stopping Criterion

- When to stop:
  - no improvement can be made
    - contrib(r) = 0
  - or improvement too small
    - contrib(r) < Threshold
- Setting a reasonable threshold is advisable, although not subject to overfitting.
- Heldout?
  - remove rules that degrade performance on H

Rule Application

Two possibilities:

1. immediate consequences (left-to-right):

   data: DT NN VBP NN VBP NN...

   rule: NN => NNS / preceded by NN VBP

   apply rule at position 4:

   DT NN VBP NN VBP NN...

   DT NN VBP NNS VBP NN...

   Now rule does not apply at position 6 (not NN VBP).

2. delayed ("fixed input"):

   use original input for context

   the above rule then applies twice

The Tagger

Input:

untagged data

rules (S) learned by the learner

Tagging:

use the same initialization as the learner did

for i = 1..n (n: the number of learned rules)

apply the rule i to the intermediate data, changing (some) tags

end

the last intermediate data is the output

Modifications

- Unsupervised Modification:
  - Use only unambiguous words for evaluation criterion
  - Works extremely well for English (95.6% accuracy)
  - Does not work for languages with few unambiguous words
- *N*-best Modification
  - Can update TBL tagger to tag some words with multiple tags; however, there is no assessment of how likely each tag is

Factors Affecting Tagging Accuracy

- The amount of training data available (more is always better)
- The tag set: larger tag sets make the task harder (more ambiguity)
- Differences between training (and dictionary) and corpus of application
- Unknown words: the more OOV words, the lower the accuracy

Examples of Tagging Errors

- Tag with JJ instead of NN: an executive order
- Tag with RB instead of JJ: more important issues
- Tag with RP rather than IN: He ran up the staircase.
- Tag with VBN rather than VBD: loan needed to meet…
- Tag with VBD rather than VBN: loan needed to meet…

Confusion Matrix

Corr/Ass Tags	DT	IN	JJ	NN	RB	RP	VB	VBING	Other
DT	99.4	0.3			0.3				0
IN	0.4	97.5			1.5	0.5			0.1
JJ		0.1	93.9	1.8	0.9		0.1	0.4	2.8
NN			2.2	95.5			0.2	0.4	1.7
RB	0.2	2.4	2.2	0.6	93.2	1.2			0.2
RP		24.7		1.1	12.6	61.5			0.1
VB			0.3	1.4			96		2.3
VBING			2.5	4.4				93	0.1

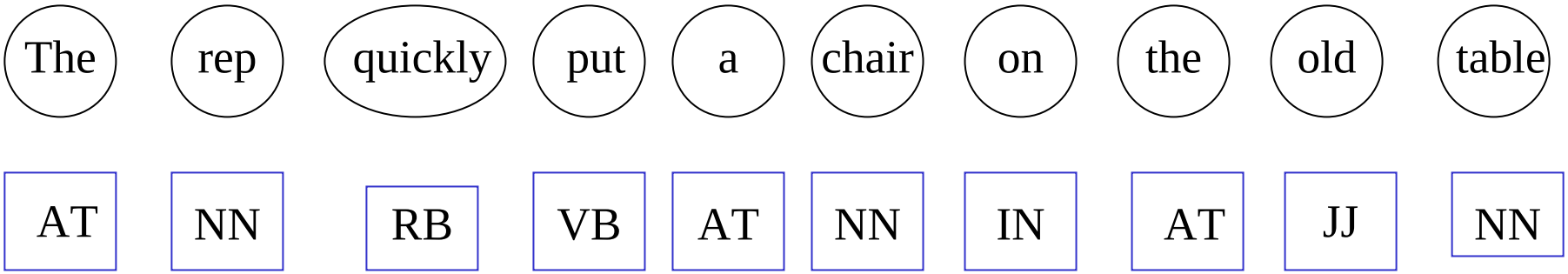
Tagger Evaluation

- Test data (S) previously unseen (during training)
  - Change test data often if at all possible! (“feedback cheating”)
  - Error-rate based (tag versus sentence levels)
- Formally:
  - Out(w) = set of output “items” for an input “item” w
  - True(w) = single correct output (annotation) for w
  - Errors(S) = $S_{i=1..|S|}d(Out(w_i) \neq True(w_i))$
  - Correct(S) = $S_{i=1..|S|}d(True(w_i) \in Out(w_i))$
  - Generated(S) = $S_{i=1..|S|}|Out(w_i)|$

Evaluation Metrics

- Accuracy: Single output (each word gets a single tag)
  - Error rate: Err(S) = Errors(S) / |S|
  - Accuracy: Acc(S) = 1 - (Errors(S) / |S|) = 1 - Err(S)
- What if multiple (or no) output?
  - Recall: R(S) = Correct(S) / |S|
    - proportion target tags the system got
  - Precision: P(S) = Correct(S) / Generated(S)
    - proportion it got right
  - Combination: F measure: F = 1 / (a/P + (1-a)/R)
    - a is a weight given to precision vs. recall; for a=.5, F = 2PR/(R+P)

Part-of-Speech Tagging of Sentences



- Table 10.1 shows a list of example tags
- Download: http://nlp.stanford.edu/software/tagger.shtml
- Markov model taggers
- Hidden Markov model taggers
- Transformation-based learning of Tags
  - Decision trees
  - Probabilistic models
  - Finite state transducers

Applications of Tagging

- Partial Parsing: syntactic analysis
- Information Extraction: tagging and partial parsing help identify useful terms and relationships between them
- Information Retrieval: noun phrase recognition and query-document matching based on meaningful units rather than individual terms
- 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

Summary

- Classes
  - Part-of-speech tagging (today)
  - Text representation: latent semantic indexing (next)
- Note
  - Project summary due on 2/24, let's start our discussion
  - Project plan finalize on 3/3 (presentation on 4/16)
  - Lab4 assigned on 2/28 and due on 3/12
  - Midterm on 3/12
  - Take-home Final (given 4/24, due on midnight 4/26)
- Reading Assignments
  - Manning and Schutze, Chapters 10 and 12