

Natural Language Processing and Machine Translation

Parsing, Sequence Labelling and Parts-of-Speech Tagging

Abhishek Koirala

M.Sc. in Informatics and
Intelligent Systems
Engineering

Language Structure and Meaning

Our purpose is to know how meaning is mapped onto what language structures

- [**Thing** The dog] is [**Place** in the garden]
- [**Thing** The dog] is [**Property** fierce]
- [**Action** [**Thing** The dog] is chasing [**Thing** the cat]]
- [**State** [**Thing** The dog] was sitting [**Place** in the garden] [**Time** yesterday]]

Word Categories: Traditional parts of speech

Noun	Names of things	boy, cat, truth
Verb	Action or state	become, hit
Pronoun	Used for noun	I, you, we
Adverb	Modifies V, Adj, Adv	sadly, very
Adjective	Modifies noun	happy, clever
Conjunction	Joins things	and, but, while
Preposition	Relation of N	to, from, into
Interjection	An outcry	ouch, oh, alas, psst

Constituency

Group of words may behave as a single unit of phrase , called a **constituent**

Sentence have parts, some of which appear to have subparts. **Constituents** are those grouping of words that go together.

I hit the man with an axe.

I hit [the man with an axe].

I hit [the man] with an axe.

I talked to a man using a phone.

I talked to [a man using a phone].

I talked to [a man] using a phone.

Constituent Phrases

For constituents, we usually name them as phrases based on the word that heads the constituent:

<i>the man from Amherst</i>	is a Noun Phrase (NP) because the head man is a noun
<i>extremely clever</i>	is an Adjective Phrase (AP) because the head clever is an adjective
<i>down the river</i>	is a Prepositional Phrase (PP) because the head down is a preposition
<i>killed the rabbit</i>	is a Verb Phrase (VP) because the head killed is a verb

Note that a word is a constituent (a little one). Sometimes words also act as phrases. In:

Joe grew potatoes.

Joe and *potatoes* are both nouns and noun phrases.

Compare with:

The man from Amherst grew *beautiful russet potatoes*.

We say *Joe* counts as a noun phrase because it appears in a place that a larger noun phrase could have been.

Constituent Phrases

1. They appear in similar environments (before a verb)

Kermit the frog comes on stage

They come to Massachusetts every summer

December twenty-sixth comes after Christmas

The reason he is running for president comes out only now.

But not each individual word in the constituent

**The comes out... *is comes out... *for comes out...*

2. The constituent can be placed in a number of different locations

Constituent = Prepositional phrase: *On December twenty-sixth*

On December twenty-sixth I'd like to fly to Florida.

I'd like to fly on December twenty-sixth to Florida.

I'd like to fly to Florida on December twenty-sixth.

But not split apart

**On December I'd like to fly twenty-sixth to Florida.*

**On I'd like to fly December twenty-sixth to Florida.*

The most common way of modelling constituency is using Context Free Grammars (CFG)

Context Free Grammar (CFG)

$$G = \langle T, N, S, R \rangle$$

- T is set of terminals (lexicon)
- N is set of non-terminals For NLP, we usually distinguish out a set $P \subset N$ of *preterminals* which always rewrite as terminals.
- S is start symbol (one of the nonterminals)
- R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (may be empty).
- A grammar G generates a language L .

Context Free Grammar (CFG)

$$G = \langle T, N, S, R \rangle$$

$$T = \{that, this, a, the, man, book, flight, meal, include, read, does\}$$

$$N = \{S, NP, NOM, VP, Det, Noun, Verb, Aux\}$$

$$S = S$$

$$R = \{$$

$$S \rightarrow NP \ VP$$

$$Det \rightarrow that \mid this \mid a \mid the$$

$$S \rightarrow Aux \ NP \ VP$$

$$Noun \rightarrow book \mid flight \mid meal \mid man$$

$$S \rightarrow VP$$

$$Verb \rightarrow book \mid include \mid read$$

$$NP \rightarrow Det \ NOM$$

$$Aux \rightarrow does$$

$$NOM \rightarrow Noun$$

$$NOM \rightarrow Noun \ NOM$$

$$VP \rightarrow Verb$$

$$VP \rightarrow Verb \ NP$$

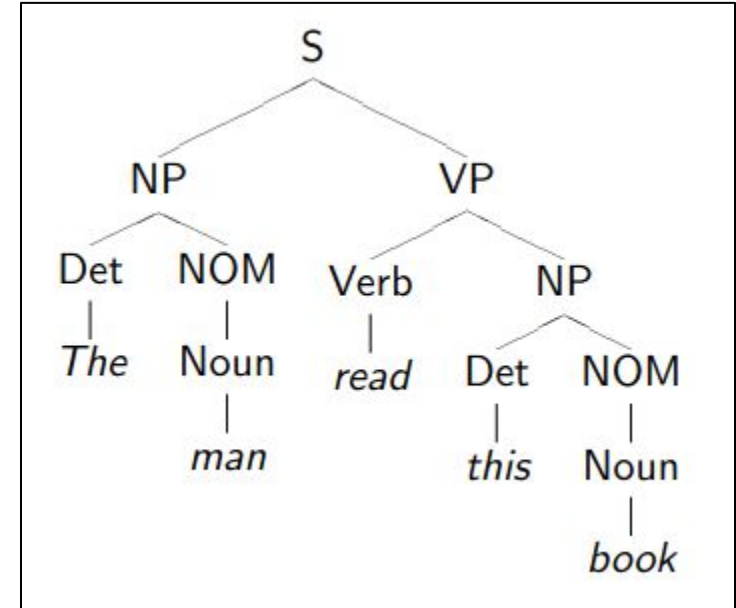
}

Parsing

- Our purpose is to run the grammar backwards to find the structure
- Parsing can be viewed as a search problem
- We search through the legal rewritings of the grammar. We want to find all structures matching an input string of words
- Two types:
 - Top - down
 - Bottom - up

Context Free Grammar (CFG)

$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid a \mid the$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid man$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid read$
$NP \rightarrow Det NOM$	$Aux \rightarrow does$
$NOM \rightarrow Noun$	
$NOM \rightarrow Noun NOM$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	



Top-Down Parsing

Top down parsing is goal directed

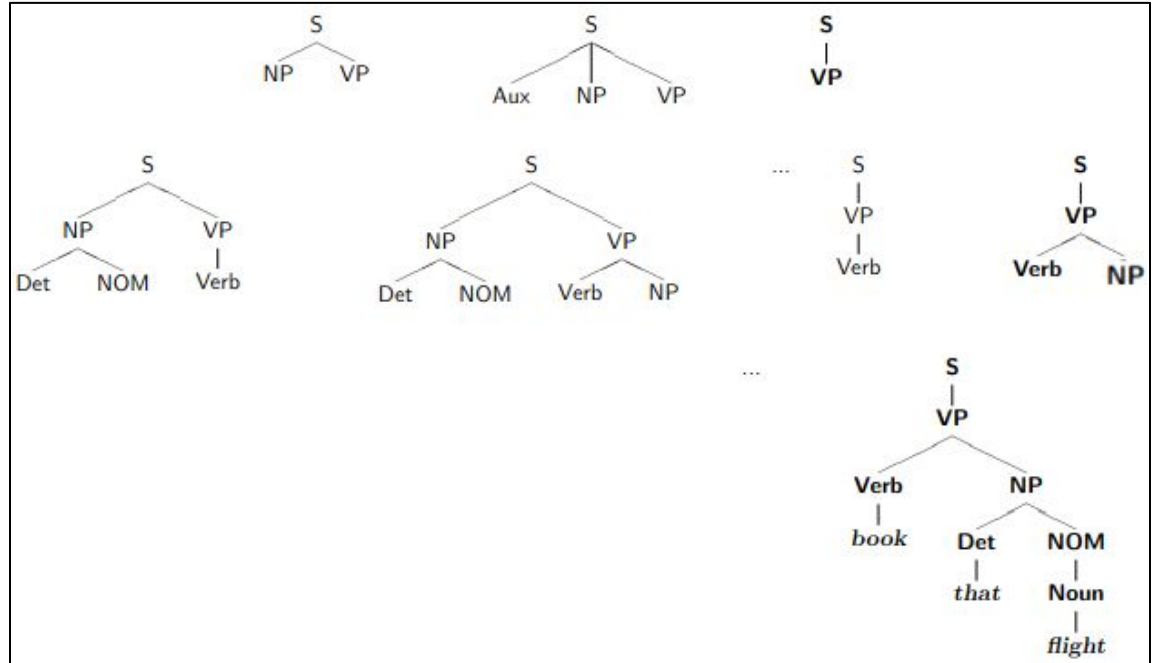
- A top-down parser starts with a list of constituents to be built
- It rewrites the goals in the goal list by matching one against the LHS of the grammar rules
- and expanding it with the RHS
- attempting to match the sentence to be derived

If a goal can be rewritten in several ways, then there is a choice of which rule to apply (search problem)

Top-Down Parsing

book that flight

$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid a \mid the$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid man$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid read$
$NP \rightarrow Det NOM$	$Aux \rightarrow does$
$NOM \rightarrow Noun$	
$NOM \rightarrow Noun NOM$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	



Bottom-up Parsing

Bottom up parsing is goal directed

- Initial goal list of a bottom-up parser is the string to be parsed
- If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule
- Parsing is finished when the goal list contains just the start symbol

If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)

The standard presentation is as shift-reduce parsing.

Shift-Reduce Parser

Start with the sentence to be parsed in an input buffer

- a “shift” action corresponds to pushing the next input symbol from the buffer onto the stack
- a “reduce” action occurs when we have a rule’s RHS on top of the stack. To perform the reduction, we pop the rule’s RHS off the stack and replace with the terminal on the LHS of the corresponding rule

If you end up with only the START symbol on the stack, then success !!

- If you don’t, and no “shift” or “reduce” actions are possible, backtrack

Bottom-up parsing example

$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid a \mid the$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid man$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid read$
$NP \rightarrow Det NOM$	$Aux \rightarrow does$
$NOM \rightarrow Noun$	
$NOM \rightarrow Noun NOM$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	

Book that flight.

How do you parse this using shift reduce approach ?

CKY Parser

- (Cocke - Kasami - Younger) CKY Parser
- Bottom-up parser
- Dynamic programming approach
- Presumes a CFG in Chomsky Normal Form

Rules are all either $A \rightarrow B C$ or $A \rightarrow a$
(with A, B, C nonterminals and a a terminal)

function CKY (word w , grammar P) returns table

for $i \leftarrow$ from 1 to $\text{LENGTH}(w)$ do

$\text{table}[i-1, i] \leftarrow \{A \mid A \rightarrow w_i \in P\}$

for $j \leftarrow$ from 2 to $\text{LENGTH}(w)$ do

 for $i \leftarrow$ from $j-2$ down to 0 do

 for $k \leftarrow i + 1$ to $j - 1$ do

$\text{table}[i,j] \leftarrow \text{table}[i,j] \cup \{A \mid A \rightarrow BC \in P,$
 $B \in \text{table}[i,k], C \in \text{table}[k,j]\}$

If the start symbol $S \in \text{table}[0,n]$ then $w \in L(G)$

CKY Parser

Grammar rules

$S \rightarrow NP VP$
 $NP \rightarrow Det N$
 $VP \rightarrow V NP$
 $V \rightarrow \text{includes}$
 $Det \rightarrow \text{the}$
 $Det \rightarrow \text{a}$
 $N \rightarrow \text{meal}$
 $N \rightarrow \text{flight}$

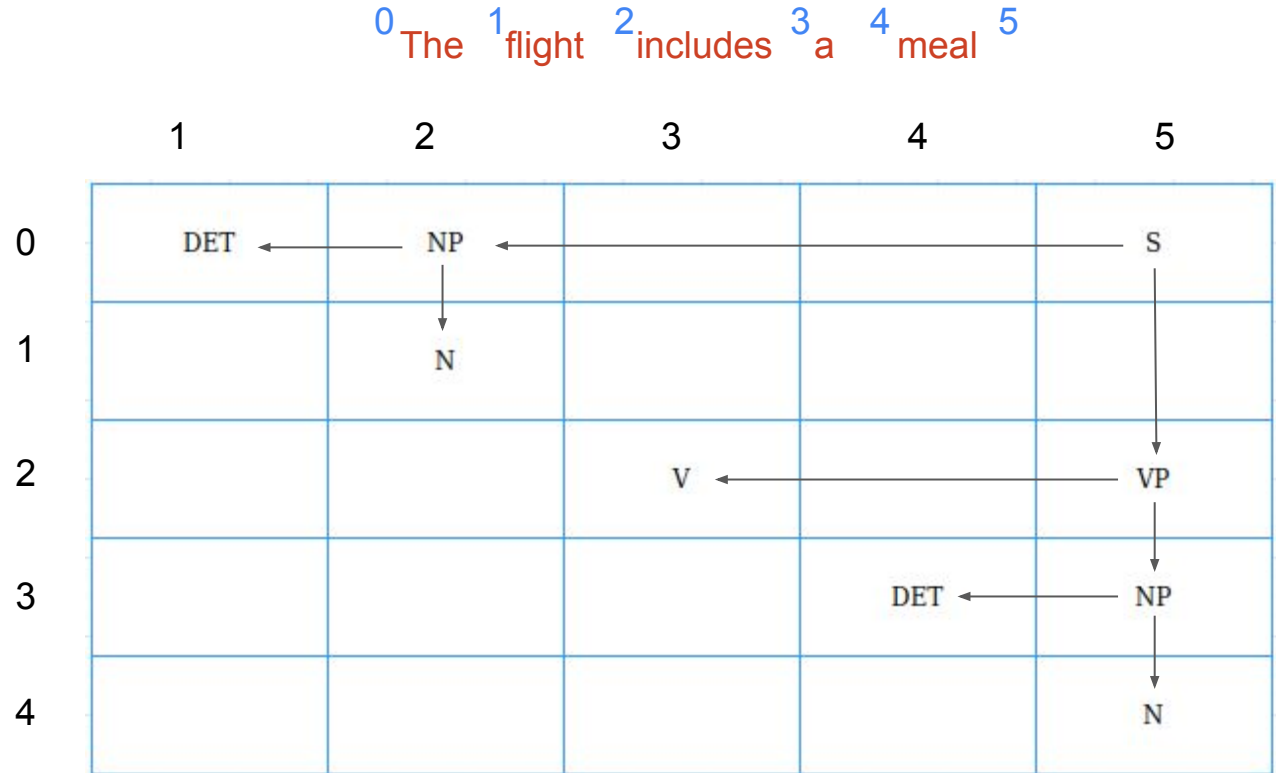
0 The 1 flight 2 includes 3 a 4 meal 5

	1	2	3	4	5
0					
1					
2					
3					
4					

CKY Parser

Grammar rules

$S \rightarrow NP VP$
 $NP \rightarrow Det N$
 $VP \rightarrow V NP$
 $V \rightarrow \text{includes}$
 $Det \rightarrow \text{the}$
 $Det \rightarrow \text{a}$
 $N \rightarrow \text{meal}$
 $N \rightarrow \text{flight}$



General Principles:

- A clever hybrid *Bottom-Up* and *Top-Down* approach
- *Bottom-Up* parsing completely guided by *Top-Down* predictions
- Maintains sets of “dotted” grammar rules that:
 - Reflect what the parser has “seen” so far
 - Explicitly predict the rules and constituents that will combine into a complete parse
- Similar to Chart Parsing - partial analyses can be shared
- Time Complexity $O(n^3)$, but better on particular sub-classes
- Developed prior to Chart Parsing, first efficient parsing algorithm for general context-free grammars.

Earley Parser

Three Main Operations:

- **Predictor:** If state $[A \rightarrow X_1 \dots \bullet C \dots X_m, j] \in S_i$ then for every rule of the form $C \rightarrow Y_1 \dots Y_k$, add to S_i the state $[C \rightarrow \bullet Y_1 \dots Y_k, i]$
- **Completer:** If state $[A \rightarrow X_1 \dots X_m \bullet, j] \in S_i$ then for every state in S_j of form $[B \rightarrow X_1 \dots \bullet A \dots X_k, l]$, add to S_i the state $[B \rightarrow X_1 \dots A \bullet \dots X_k, l]$
- **Scanner:** If state $[A \rightarrow X_1 \dots \bullet a \dots X_m, j] \in S_i$ and the next input word is $x_{i+1} = a$, then add to S_{i+1} the state $[A \rightarrow X_1 \dots a \bullet \dots X_m, j]$

The Earley Recognition Algorithm

The Main Algorithm: parsing input $x = x_1 \dots x_n$

1. $S_0 = \{[S' \rightarrow \bullet S \$, 0]\}$
2. For $0 \leq i \leq n$ do:
 Process each item $s \in S_i$ in order by applying to it the *single* applicable operation among:
 - (a) Predictor (adds new items to S_i)
 - (b) Completer (adds new items to S_i)
 - (c) Scanner (adds new items to S_{i+1})
3. If $S_{i+1} = \phi$, *Reject* the input
4. If $i = n$ and $S_{n+1} = \{[S' \rightarrow S \$\bullet, 0]\}$ then *Accept* the input

Earley Parser

$S \rightarrow NP VP$
 $NP \rightarrow art adj n$
 $NP \rightarrow art n$
 $NP \rightarrow adj n$
 $VP \rightarrow aux VP$
 $VP \rightarrow V NP$

The large can can hold the water

art adj n aux v art n \$

Input X : **art** adj n aux v art n \$

So : [$S' \rightarrow . S \$, 0$]
 [$S \rightarrow . NP VP , 0$]
 [$NP \rightarrow . art adj n , 0$]
 [$NP \rightarrow . art n , 0$]
 [$NP \rightarrow . adj n , 0$]

Earley Parser

$S \rightarrow NP VP$
 $NP \rightarrow art \ adj \ n$
 $NP \rightarrow art \ n$
 $NP \rightarrow adj \ n$
 $VP \rightarrow aux \ VP$
 $VP \rightarrow V \ NP$

Input X : **art** adj n aux v art n \$

So : [$S' \rightarrow . S \$, 0$]
 [$S \rightarrow . NP VP, 0$]
 [$NP \rightarrow . art \ adj \ n, 0$]
 [$NP \rightarrow . art \ n, 0$]
 [$NP \rightarrow . adj \ n, 0$]

S1 : [$NP \rightarrow art \ . adj \ n, 0$]
 [$NP \rightarrow art \ . n, 0$]

Input X : art **adj** n aux v art n \$

Input X : art **adj** n aux v art n \$

S2 : [$NP \rightarrow art \ adj \ . n, 0$]

Input X : art adj **n** aux v art n \$

S3 : [$NP \rightarrow art \ adj \ n . , 0$]
 [$S \rightarrow NP \ . VP, 0$]
 [$VP \rightarrow . aux \ VP, 3$]
 [$VP \rightarrow V \ NP, 3$]

Earley Parser

$S \rightarrow NP VP$
 $NP \rightarrow art adj n$
 $NP \rightarrow art n$
 $NP \rightarrow adj n$
 $VP \rightarrow aux VP$
 $VP \rightarrow V NP$

Input X : art adj n **aux** v art n \$

S4 : [$VP \rightarrow aux .VP$, 3]
[$VP \rightarrow .aux VP$, 4]
[$VP \rightarrow .V NP$, 4]

Input X : art adj n aux v **v** art n \$

S5 : [$VP \rightarrow V .NP$, 4]
[$NP \rightarrow .art adj n$, 5]
[$NP \rightarrow .art n$, 5]
[$NP \rightarrow .adj n$, 5]

Input X : art adj n aux v **art** n \$

S6 : [$NP \rightarrow art .adj n$, 5]
[$NP \rightarrow art .n$, 5]

Input X : art adj n aux v art **n** \$

S7 : [$NP \rightarrow art n.$, 5]
[$VP \rightarrow V NP.$, 4]
[$VP \rightarrow aux VP.$, 3]
[$S \rightarrow NP VP.$, 0]
[$S' \rightarrow S .\$$, 0]

S8 : [$S' \rightarrow S \$.$, 0]

Describe in your own words the role of **predictor**,
completer and **scanner** in Earley Parser?

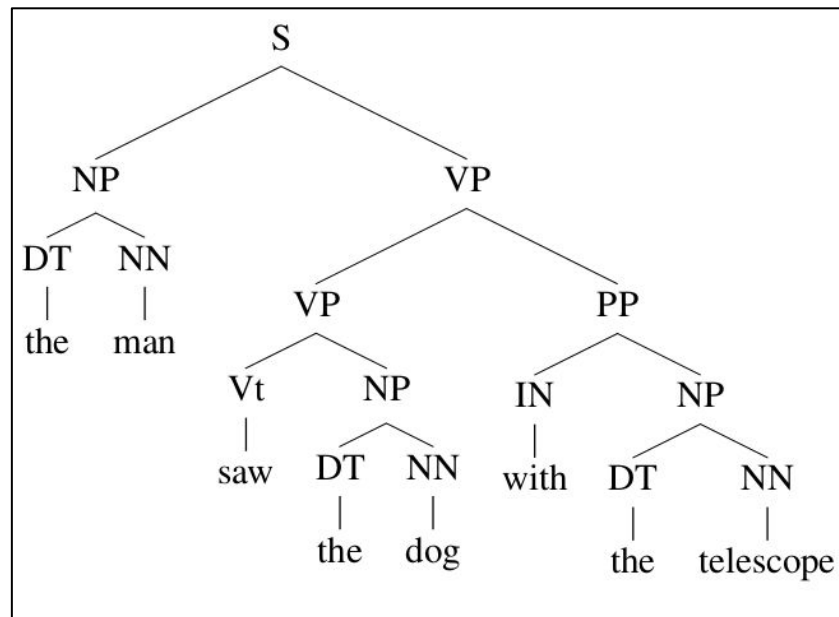
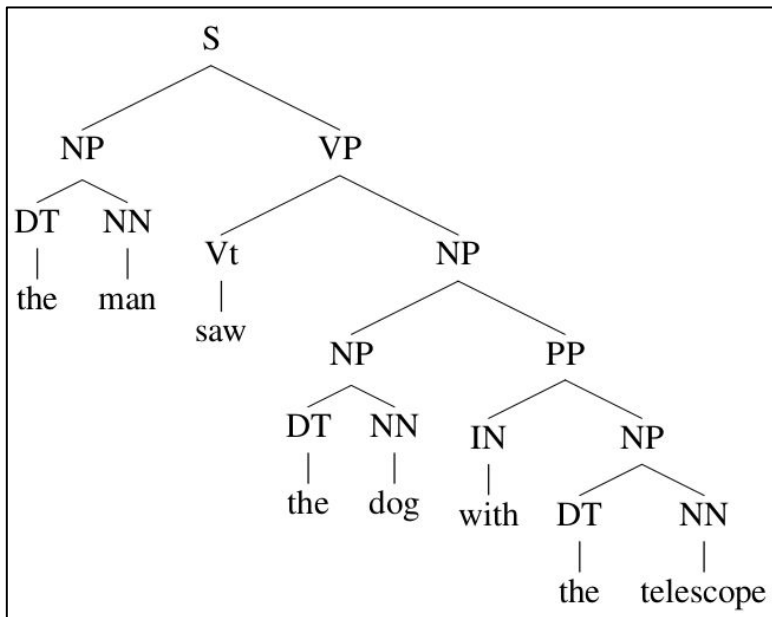
Partial Parsing

- Many language processing tasks do not require complex, complete parse tree for all inputs. For these tasks, a **partial parse** or a **shallow parse** of input sentences may be sufficient. Eg. Information extraction systems
- One such kind of partial parsing is known as **chunking**. **Chunking** is the process of identifying and classifying the flat, non-overlapping segments of a sentence that constitute the basic non recursive phrases corresponding to the major content-word parts-of-speech: noun phrases, verb phrases, adjective phrases and prepositional phrases.

[*NP* The morning flight] [*PP* from] [*NP* Denver] [*VP* has arrived.]

- The bracketing notion defines the two fundamental tasks that are involved in chunking:
 - Segmenting
 - Labelling

Ambiguity in Context Free Grammar



Probabilistic Context-Free Grammars (PCFG)

A PCFG consists of

- A Context Free Grammar $G = (N, \Sigma, S, R)$
- A parameter $q(\alpha \rightarrow \beta)$

for each rule $\alpha \rightarrow \beta \in R$. The parameter $q(\alpha \rightarrow \beta)$ can be interpreted as the conditional probability of choosing rule $\alpha \rightarrow \beta$ in a left-most derivation, given that the non-terminal being expanded is α . For any $X \in N$, we have the constraint

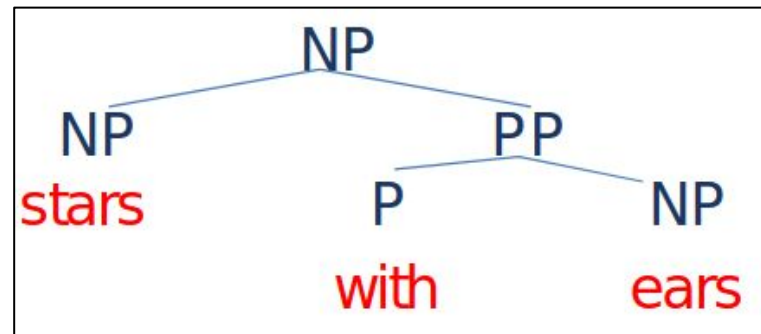
$$\sum_{\alpha \rightarrow \beta \in R: \alpha = X} q(\alpha \rightarrow \beta) = 1$$

In addition we have $q(\alpha \rightarrow \beta) \geq 0$ for any $\alpha \rightarrow \beta \in R$.

Probabilistic Context-Free Grammars (PCFG)

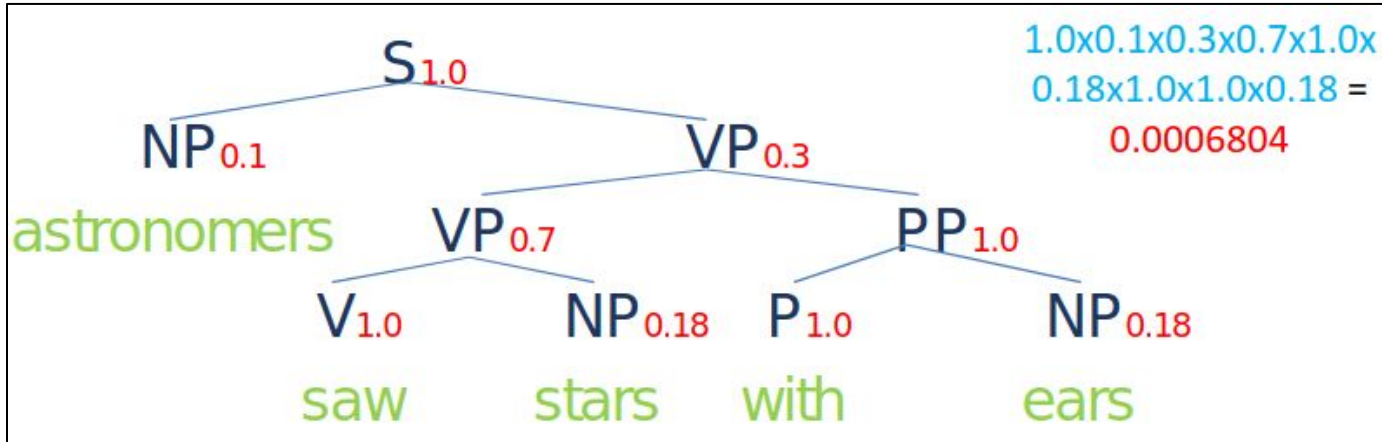
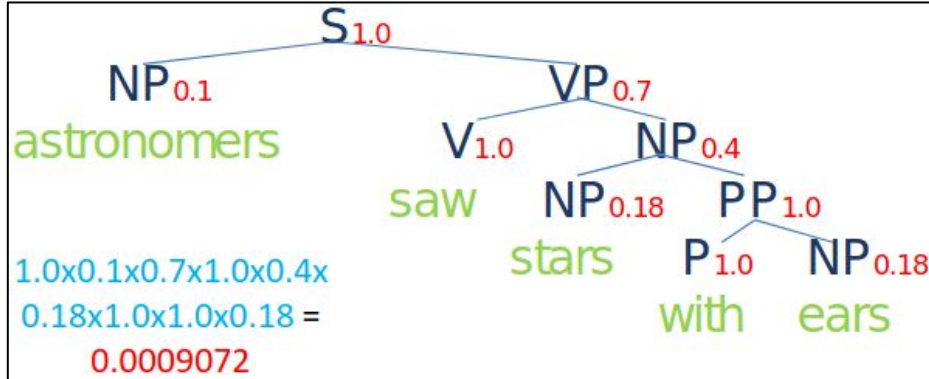
S → NP VP	1.0
PP → P NP	1.0
VP → V NP	0.7
VP → VP PP	0.3
P → with	1.0
V → saw	1.0

NP → NP PP	0.4
NP → astronomers	0.1
NP → ears	0.18
NP → saw	0.04
NP → stars	0.18
NP → telescopes	0.1



$$\begin{aligned} &P(NP \rightarrow NP PP, NP \rightarrow stars, PP \rightarrow P NP, P \rightarrow with, NP \rightarrow ears) \\ &= P(S \rightarrow NP PP) \times P(NP \rightarrow stars | NP \rightarrow NP PP) \\ &\times P(PP \rightarrow P NP | NP \rightarrow NP PP, NP \rightarrow stars) \\ &\times P(P \rightarrow with | PP \rightarrow P NP, NP \rightarrow NP PP, NP \rightarrow stars) \\ &\times P(NP \rightarrow ears | P \rightarrow with, PP \rightarrow P NP, NP \rightarrow NP PP, NP \rightarrow stars) \end{aligned}$$

Probabilistic Context-Free Grammars (PCFG)

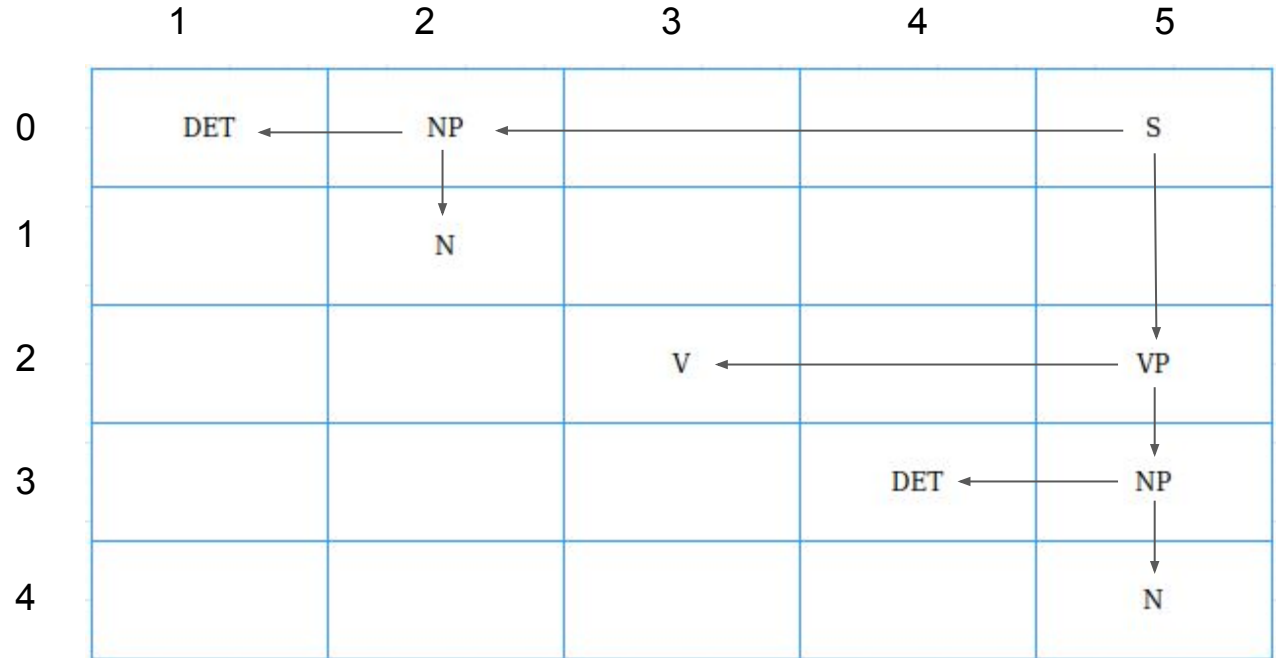


CKY Parser (PCFG)

Grammar rules

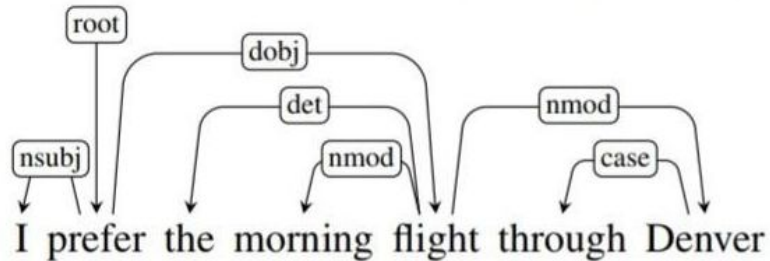
$S \rightarrow NP VP$ [0.8]
 $NP \rightarrow Det N$ [0.3]
 $VP \rightarrow V NP$ [0.2]
 $V \rightarrow \text{includes}$ [0.05]
 $Det \rightarrow \text{the}$ [0.4]
 $Det \rightarrow \text{a}$ [0.4]
 $N \rightarrow \text{meal}$ [0.01]
 $N \rightarrow \text{flight}$ [0.02]

0 The 1 flight 2 includes 3 a 4 meal 5



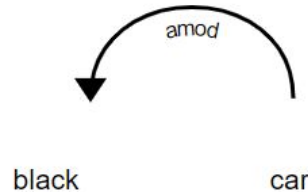
Dependency parsing

- Process of examining dependencies between phrases of a sentence in order to determine its grammatical structure
- The mechanism is based on the concept that there is a direct link between every linguistic unit of a sentence. These links are termed dependencies



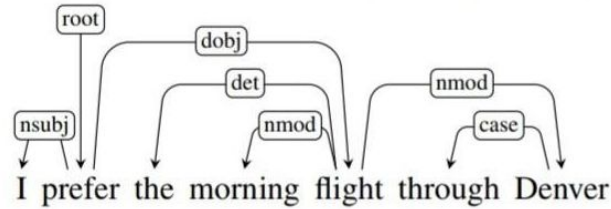
Dependency parsing

- Each relationship has one **head** and a **dependent** that modifies the **head**
- Each relationship is labelled according to the nature of the dependency between the **head** and the **dependent**.



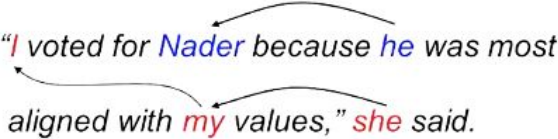
- Here a relationship exists between car and black because black modifies the meaning of car. Here, **car** acts as the **head** and **black** is the **dependent** of the head. The **nature of relationship** here is **amod** which stands for “**Adjectival Modifier**”. Find all the universal dependency relations [here](#)

Dependency parsing



- **Head-Dependent:** In the arrows representing relationship, the origin word is the Head and the destination word is the Dependent
 - Eg. (*I*, '*nsubj*', *prefer*) → '*prefer*' is the Head and "*I*" is the dependent
- **Root:** Word which is the root of our parse tree.
- **Grammar Functions and Arcs:** Tags between each Head-Dependent pair is a grammar function determining the relation between the head and the dependent. The arrowhead carrying the tag is called an arc

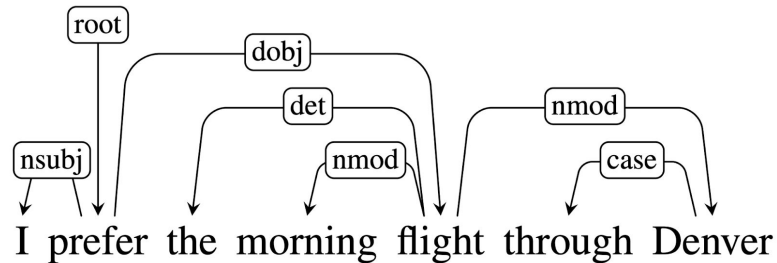
Why we need to learn sentence structure?

- Humans need to work out what *depends* what
 - Thus, similarly, a model needs to understand sentence structure in order to interpret language correctly
 - E.g., **coreference resolution** 

"I voted for Nader because he was most aligned with my values," she said.
- Human language is **ambiguous** by nature
 - Knowing these ambiguities allow us to watch out for these potential problems in our model

Building a parser

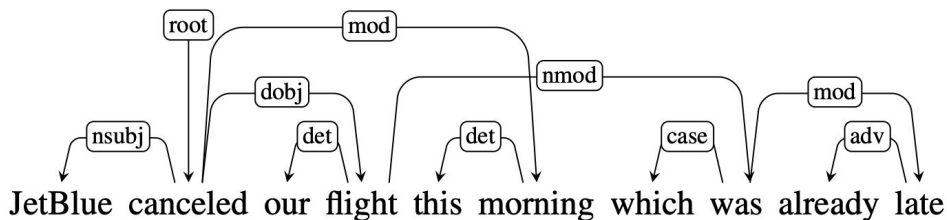
- A sentence is **parsed** by choosing for each word, what other (including ROOT) it is a dependent of
- To make the problem easier to solve, we impose some **constraints**:
 - a. There is a single designated root node that has no incoming arcs
 - b. With the exception of the root node, each vertex has exactly one incoming arc
 - c. There is a unique path from the root node to each vertex in V
 - d. Don't want cycles, e.g., A→B, B→A



Projectivity

Note that some dependency parsing algorithms are **projective**:

Projectivity: An arc from a head to a dependent is said to be projective **if there is a path from the head to every word that lies between the head and the dependent in the sentence**. A dependency tree is then said to be projective if all the arcs that make it up are projective.



In this example, the arc from **flight** to its modifier was is non-projective since there is no path from **flight** to the intervening words **this** and **morning**. A dependency tree is projective if it can be drawn with **no crossing edges**. Here there is no way to link flight to its dependent was without crossing the arc that links morning to its head.

Transition based Dependency Parsing

- It is a kind of shift reduce parsing originally developed for parsing programming languages
- Three transition operations:
 - **LEFTARC**: Assert a head dependent relation between the word at the top of the stack and the word directly below it; remove the lower word from the stack
 - **RIGHTARC**: Assert a head-dependent relation between the second word in the stack and the word at the top; remove the word at the top of the stack
 - **SHIFT**: Remove the word at the front of the input buffer and push it onto the stack

Transition based Dependency Parsing

Algorithm

```
function DEPENDENCYPARSE(words) returns dependency tree  
  
state  $\leftarrow$  {[root], [words], [] } ; initial configuration  
while state not final  
    t  $\leftarrow$  ORACLE(state)      ; choose a transition operator to apply  
    state  $\leftarrow$  APPLY(t, state) ; apply it, creating a new state  
return state
```

How do parser know which operator to select?

- ⇒ The parser consults an oracle for advice on which operator to choose.
- ⇒ The oracle could be created using machine learning techniques and information on POS of words

Dependency Parsing Summary

- Dependency parsing is hard because of many possible **ambiguities** and **paths**
- **Dependency treebanks** provide a useful ground truth we can based on
- Two main approaches:
 - **Transition-based**: simple but does not work with non-projectivity
 - **Graph-based**: works with non-projectivity but very very slow...
- Common way:
 - Use transition-based parser and then fix all non-projective paths manually
- UAS and LAS are two common metrics for dependency parsing

Sequence Labelling as Classification

- Goal of sequence labelling is to assign tags to words, or more generally, to assign discrete labels to discrete elements in a sequence.

They can fish

- A dictionary of coarse-grained pos tags might include **NOUN** as the only valid tag for *they*, but both **NOUN** and **VERB** as potential tags for *can* and *fish*

Which tag is accurate? How do we solve this

Sequence Labelling as Classification

- One way to solve such problem is by turning it into a classification problem
- Let $f((\mathbf{w}, m), y)$ indicate the feature function for tag y at position m in the sequence $\mathbf{w} = (w_1, w_2, \dots, w_M)$.
- A simple tagging model would have a single base feature:

$$f((\mathbf{w} = \text{they can fish}, m = 1), N) = (\text{they}, N)$$

$$f((\mathbf{w} = \text{they can fish}, m = 2), V) = (\text{can}, V)$$

$$f((\mathbf{w} = \text{they can fish}, m = 3), V) = (\text{fish}, V).$$

3 arguments at input

- a) Sentence to be tagged
- b) Proposed tag
- c) Index of token to which the tag is applied

Sequence Labelling as Classification

- The feature function would return a single feature: a tuple containing the word to be tagged and the tag that has been assigned.
- Look at the entire corpus, each of such feature would be assigned a weight, which can be learned from a labelled dataset using a classification algorithm or can be defined directly
- But this approach would not work for the problem we saw before as we must also rely on context
- We can build context into the feature set by incorporating the surrounding words as additional features

$$\begin{aligned}f((\mathbf{w} = \textit{they can fish}, 1), \mathbf{N}) &= \{(w_m = \textit{they}, y_m = \mathbf{N}), \\ &\quad (w_{m-1} = \square, y_m = \mathbf{N}), \\ &\quad (w_{m+1} = \textit{can}, y_m = \mathbf{N})\} \\ f((\mathbf{w} = \textit{they can fish}, 2), \mathbf{V}) &= \{(w_m = \textit{can}, y_m = \mathbf{V}), \\ &\quad (w_{m-1} = \textit{they}, y_m = \mathbf{V}), \\ &\quad (w_{m+1} = \textit{fish}, y_m = \mathbf{V})\} \\ f((\mathbf{w} = \textit{they can fish}, 3), \mathbf{V}) &= \{(w_m = \textit{fish}, y_m = \mathbf{V}), \\ &\quad (w_{m-1} = \textit{can}, y_m = \mathbf{V}), \\ &\quad (w_{m+1} = \blacksquare, y_m = \mathbf{V})\}.\end{aligned}$$

Parts of Speech Tagging

- Sequence labelling task
- Process of marking up a word in a text (corpus) as corresponding to a particular part of speech (syntactic tag), based on its definition and context.
- Determining POS Tags is much more complicated than simply mapping words to their tags
- Consider the word **back**

- Earnings growth took a **back/JJ** seat.
- A small building in the **back/NN**.
- A clear majority of senators **back/VBP** at the bill.

Parts of Speech Tagging

- POS Tagging, also called **grammatical tagging**, is the process of marking up a word in a text(corpus) as corresponding to a particular part of speech, based on both its definition and its context
- POS Tagging is a prerequisite to simplify a lot of different problems
 - Text-to-speech Conversion

```
import nltk
from nltk import word_tokenize
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
text = word_tokenize("They refuse to permit us to obtain the refuse permit")
nltk.pos_tag(text)
```

```
[('They', 'PRP'),
 ('refuse', 'VBP'),
 ('to', 'TO'),
 ('permit', 'VB'),
 ('us', 'PRP'),
 ('to', 'TO'),
 ('obtain', 'VB'),
 ('the', 'DT'),
 ('refuse', 'NN'),
 ('permit', 'NN')]
```

The word **refuse** is being used twice in this sentence and has two different meanings here. *refUSE* (/rə'fyooz/) is a verb meaning “deny,” while *REFuse* (/ˈref,yooos/) is a noun meaning “trash” (that is, they are not

Parts of Speech Tagging

POS Tagging can also be useful in applications such as

- Word Sense Disambiguation
- Question Answering
- Machine Translation

Rule based POS Taggers

- Typical rule based approaches use contextual information to assign tags to unknown or ambiguous words.
- Use dictionary or lexicon for getting possible tags for tagging each word
- Ambiguity is certain here too. If the word has more than one possible tag, the rule-based taggers use hand written rules to identify correct tag.
- Disambiguation can also be performed in rule based tagging by analyzing the linguistic feature of a word along with its preceding as well as following words.

Example of a rule:

If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective.

Rule based POS Taggers

- Two stages of rule based taggers
 - **First Stage:** Uses a dictionary to assign each word a list of potential parts-of-speech
 - **Second Stage:** Uses a large lists of handwritten disambiguation rules to sort down the list to a single part-of-speech for each word
- Properties of rule based taggers
 - These taggers are knowledge driven taggers
 - The rules are built manually
 - The information is coded in the form of rules
 - The number of rules are limited. Hence, there is always a chance of dealing with unseen cases

Stochastic Taggers

- Any model which somehow incorporates frequency or probability may be properly labelled stochastic
- The simplest stochastic tagger applies the following approaches for POS tagging
 - **Word Frequency approach** → In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.
 - **Tag Sequence probabilities** → It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

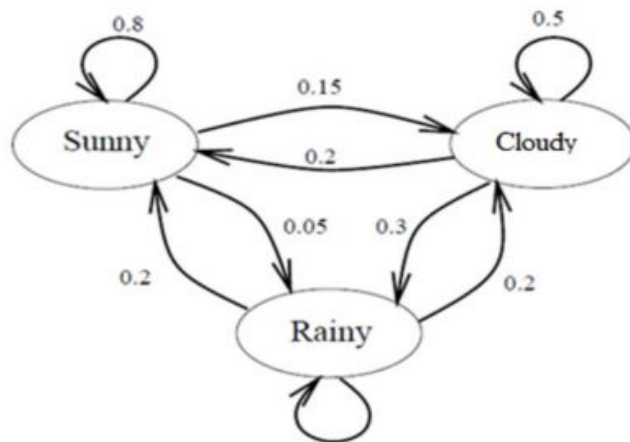
Hidden Markov Model POS Tagging

Before that, let's revise **Markov Model**

We have already discussed **Markov Model** in our previous chapter. Let's see how much you remember it.

Here's a hint:
$$P(q_1, \dots, q_n) = \prod_{i=1}^n P(q_i | q_{i-1})$$

- Exercise 1: Given that today is Sunny, what's the probability that tomorrow is Sunny and the next day Rainy?
- Exercise 2: Assume that yesterday's weather was Rainy, and today is Cloudy, what is the probability that tomorrow will be Sunny?



Hidden Markov Model POS Tagging

Basic Understanding of HMM model

- To model any problem using a HMM, we need a set of observations and a set of possible states. The states in an HMM are hidden
- In the POS Tagging problem, the **observations** are the words themselves in the given sequence. As for the **states**, which are hidden, these would be the POS tags for the words
- **Transition probabilities** would be something like $P(\text{VP} \mid \text{NP})$ ie, what is the probability of the current word having a tag of Verb Phrase given that the previous tag was a Noun Phrase
- **Emission probabilities** would be $P(\text{john} \mid \text{NP})$ or $P(\text{will} \mid \text{VP})$ ie, what is the probability that the word given a tag is a Noun Phrase

Hidden Markov Model POS Tagging

Viterbi Algorithm

Training corpus

<S>	Martin	Justin	can	watch	Will	<E>
<S>	Spot	will	watch	Martin	<E>	
<S>	Will	Justin	spot	Martin	<E>	
<S>	Martin	will	pat	Spot	<E>	

Training Corpus with Correct POS Tag:

<S>	Martin	Justin	can	watch	Will	<E>
	N	N	M	V	N	
<S>	Spot	will	watch	Martin	<E>	
	N	M	V	N		
<S>	Will	Justin	spot	Martin	<E>	
	M	N	V	N		
<S>	Martin	will	pat	Spot	<E>	
	N	M	V	N		

N: Noun
M: Modal verb
V: Verb

Hidden Markov Model POS Tagging

Viterbi Algorithm

Creating Emission Probabilities

	N	M	V
Martin	4/9	0	0
Justin	2/9	0	0
Will	1/9	3/4	0
Spot	2/9	0	1/4
Can	0	1/4	0
Watch	0	0	2/4
Pat	0	0	1/4
Σ	9	4	4

Training Corpus with Correct POS Tag:

<S>	Martin	Justin	can	watch	Will	<E>
	N	N	M	V	N	
<S>	Spot	will	watch	Martin	<E>	
	N	M	V	N		
<S>	Will	Justin	spot	Martin	<E>	
	M	N	V	N		
<S>	Martin	will	pat	Spot	<E>	
	N	M	V	N		

N: Noun
M: Modal verb
V: Verb

Hidden Markov Model POS Tagging

Viterbi Algorithm

Creating State Transition Probabilities

	N	M	V	<E>
<S>	3/4	1/4	0	0
N	1/9	1/3	1/9	4/9
M	1/4	0	3/4	0
V	1	0	0	0

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

Training Corpus with Correct POS Tag:

<S>	Martin	Justin	can	watch	Will	<E>
	N	N	M	V	N	
<S>	Spot	will	watch	Martin	<E>	
	N	M	V	N		
<S>	Will	Justin	spot	Martin	<E>	
	M	N	V	N		
<S>	Martin	will	pat	Spot	<E>	
	N	M	V	N		

N: Noun
M: Modal verb
V: Verb

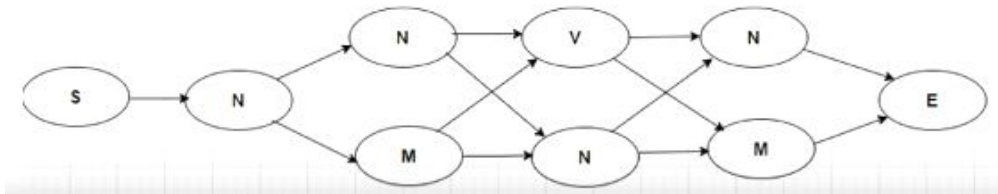
Hidden Markov Model POS Tagging

Viterbi Algorithm

Test Data

<S> Justin will spot Will <E>

Justin	will	Spot	Will
N	N	V	N
	M	N	M
Total ways : $1 \times 2 \times 2 \times 2 = 8$			



What is the correct POS Tags for the test data sequence?

Transformation Based Learning

- Example of rule-based machine learning
- Can be used for many classification tasks
- Has been applied to a wide variety of NLP tasks
 - POS tagging
 - Word Sense Disambiguation
- Consists of two phases:
 - Training phase (typically applied once):
 - Rules are learnt
 - Application phase (typically applied many times):
 - The rules are applied in the order they were learnt

POS Tagging with Transformation based Learning

- Initial state
 - Known words → words found in lexicon → tagged with their most frequent tag
 - Unknown words → tagged with most frequent tag in training corpus, depending on the first letter(capital or not) of the word in question
- Lexical tagging
 - The unknown words are tagged in isolation, based on their morphology and their immediate neighbour
- Contextual tagging
 - All words are tagged in context

POS Tagging with Transformation based Learning

An Example

The horse raced past the barn fell.

The/DT horse/NN raced/VBN past/IN the/DT
barn/NN fell/VBD ./.

- 1) Tag every word with most likely tag and score

The/DT horse/NN raced/VBD past/NN the/DT
barn/NN fell/VBD ./.

- 2) For each template, try every instantiation (e.g.
Change VBN to VBD when the preceding word is
tagged NN, add rule to ruleset, retag corpus, and
score

- 3) Stop when no transformation improves score
4) Result: set of transformation rules which can be
applied to new, untagged data (after initializing
with most common tag)
....What problems will this process run into?

POS Tagging with Transformation based Learning

Advantages

- Learning small set of simple rules are enough for tagging
- Development as well as debugging is very easy because the learned rules are easy to understand
- Complexity in tagging is reduced because there is interlacing of machine-learned and human-generated rules
- Much faster than Markov-model tagger

Disadvantages

- Does not provide tag probabilities
- Training time is very long on large corpora

Combining Taggers

- Can we obtain better taggers/models to obtain better result?
- (Brill and Wu, 1998; Halteren et al., 2001 published an observation that, although different taggers have different performances, they usually produce different errors
- Based on this encouraging observation, we can benefit from using more than one tagger in such a way that each individual tagger deals with the cases where it is the best.
- One way of combining tagger is using the output of one of the systems as input to the next system.
- Tapanainen and Voutilainen (1994) adapted this idea where a rule based system first reduces the ambiguities in the initial tags of the words as much as possible and then an HMM-based tagger arrives at the final decision.
- Rules can resolve only some of the ambiguities but with a very high correctness and the stochastic tagger resolves all ambiguities but with a lower accuracy

Combining Taggers

- Another approach (Clark et al. (2003)) investigates the effect of co-training where two taggers are iteratively retrained on each other's output.
- The tagger should be sufficiently different for co-training to be effective.
- This approach is suitable in cases when there is a small amount of annotated corpora
- Beginning from a seed set (annotated sentences). Both of the taggers (T1 and T2) are trained initially.
- Then the taggers are used to tag a set of unannotated sentences.
- The output of T1 is added to the seed set and used to retrain T2; likewise, the output of T2 is added to the seed set of retrain T1.
- Process is repeated using a new set of unannotated sentences at each iteration.

