Unit 4

Syntax Level Analysis

Dr. Madhura Vyawahare

4.	**Syntax Level Analysis** Parts of Speech (POS) tagging, approaches to POS tagging, concept of chunking and chinking, syntax parsing, Named Entity Recognition (NER), approaches to NER	**05**

POS tagging using nltk

```
# Importing the NLTK library
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag


# Sample text
text = "NLTK is a powerful library for natural language processing."


# Performing PoS tagging
pos_tags = pos_tag(words)


# Displaying the PoS tagged result in separate lines
print("Original Text:")
print(text)


print("\nPoS Tagging Result:")
for word, pos_tag in pos_tags:
    print(f"{word}: {pos_tag}")
```

```
Original Text:
NLTK is a powerful library for natural language
processing.
PoS Tagging Result:
NLTK: NNP
is: VBZ
a: DT
powerful: JJ
library: NN
for: IN
natural: JJ
language: NN
processing: NN
```

POS tagging using spacy

```
!pip install spacy
!python -m spacy download en_core_web_sm
```

```
#importing libraries
import spacy


# Load the English language model
nlp = spacy.load("en_core_web_sm")


# Sample text
text = "SpaCy is a popular natural language processing library."


# Process the text with SpaCy
doc = nlp(text)


# Display the PoS tagged result
print("Original Text: ", text)
print("PoS Tagging Result:")
for token in doc:
    print(f"{token.text}: {token.pos_}")
```

```
Original Text: SpaCy is a popular natural language
processing library.
PoS Tagging Result:
SpaCy: PROPN
is: AUX
a: DET
popular: ADJ
natural: ADJ
language: NOUN
processing: NOUN
library: NOUN
.: PUNCT
```

Syntax Analysis

- Syntax analysis or parsing is the **important phase of NLP.**
- The purpose of this phase is to draw exact meaning or dictionary meaning from the text.
- Syntax refers to the arrangement of words in a sentence such that they make grammatical sense.
- In NLP, syntactic analysis is used to assess how the natural language aligns with the grammatical rules.
- Computer algorithms are used to apply grammatical rules to a group of words and derive meaning from them.
- Syntactic analysis helps us understand the **roles played by different words** in a body of text.
- Consider **"Innocent peacefully children sleep little"** vs **"Innocent little children sleep peacefully"**

Part Of Speech tagging (POS)

- The part of speech tagging is a process of assigning corresponding part of speech like **noun, verb, pronoun, preposition, adverb, conjunction, participle, adjective, article, particles to each word in a sentence.**
- It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)).
- The tag in case of it is a part of speech tag, and signifies whether the word is a **noun, adjective, verb, and so on.**
- Automatic assignment of descriptors to the given tokens is called **Tagging.**

Book/VB (Verb, base form) **that/DT**(Determiner) **flight/NN** (Noun, singular)

POS Tagging example: English

Sentence: "She is a beautiful woman."

- **POS:** She - Pronoun, is - Verb, a - Determiner, beautiful - Adjective, woman - Noun

Sentence: "I will eat a delicious apple."

- **POS:** I - Pronoun, will - Verb, eat - Verb, a - Determiner, delicious - Adjective, apple - Noun

Examples of ambiguities in POS tagging

- The **attack/NN** was brutal.
- King was planning to **attack/VB** neighbouring states.
- Tigers usually **attack/VBP** their prey in a group.

- On Sunday, I read two **books/NNS.**
- During winter season, he **books/VBZ** a flight ticket to avail discount.
- They will **book/VBP** a flight on Sunday.

Tag set for English

- Parts of speech can be divided into two broad supercategories:

  Closed class types and

  Open class types.

- **Closed classes** are those that have relatively fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English.

- By contrast nouns and verbs are **open classes** because new nouns and verbs are continually coined or borrowed from other languages.
  - **open classes --** noun, verb, adjective, adverb
  - **closed classes --** (prepositions: on, under, over, near, by), (determiners: a, an, the), (pronouns: she, he, I), (conjunctions: For, And, Nor, But, Or, Yet, So) (participles:verb form that functions as an adjective)

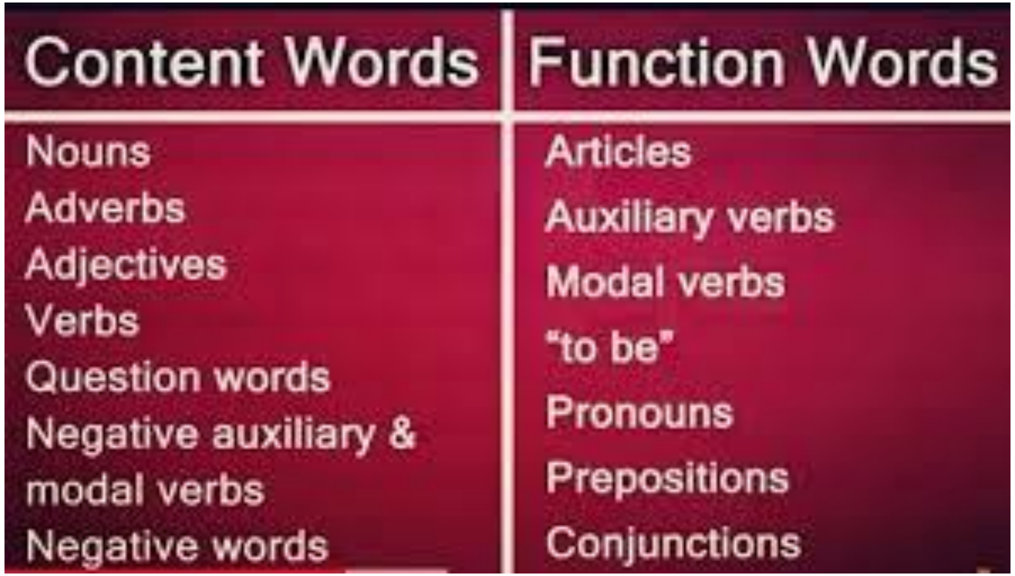
Function words and Content words

Content words

- Words that have meaning
- They are words we would look up in a dictionary, such as "lamp," "computer," "table."
- New content words are constantly added to the English language; old content words constantly leave the language as they become obsolete.
- Therefore, we refer to content words as **open** class

Function words

- words that exist to explain or create grammatical or structural relationships into which the content words may fit
- Words like "of," "the," "to," they have little meaning on their own
- They are much fewer in number and generally do not change as English adds and omits content word
- Therefore, we refer to function words as a **closed** list



Content Words	Function Words
Nouns	Articles
Adverbs	Auxiliary verbs
Adjectives	Modal verbs
Verbs	"to be"
Question words	Pronouns
Negative auxiliary & modal verbs	Prepositions
Negative words	Conjunctions

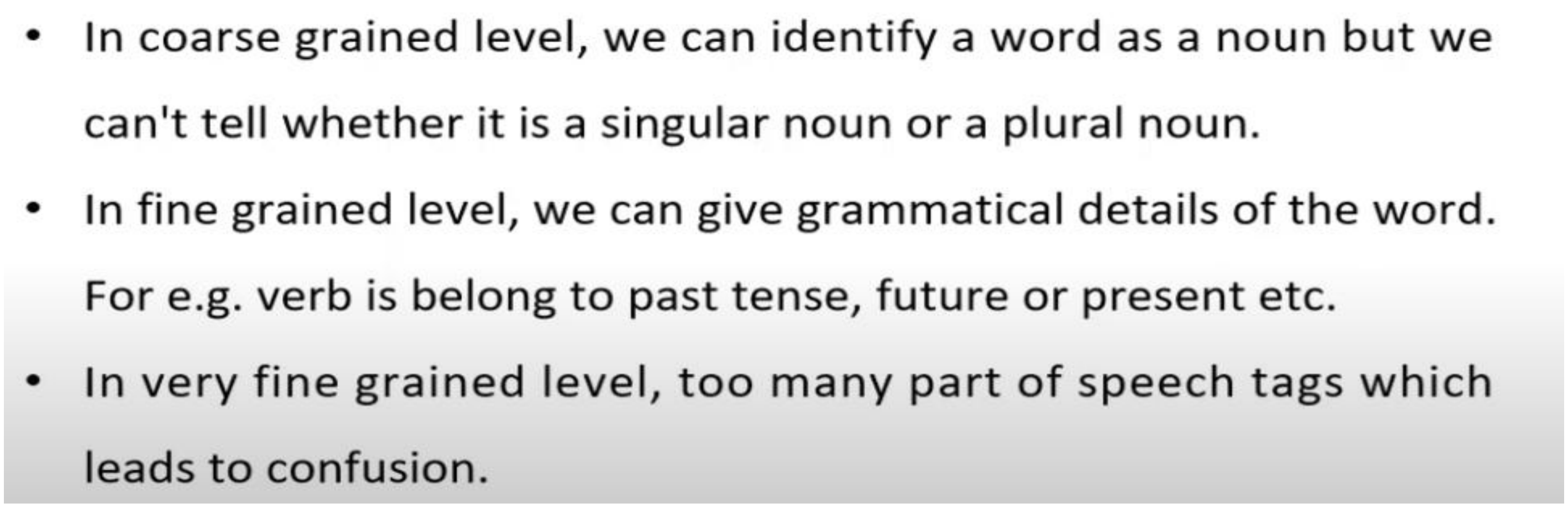
Tag set for English

- Tagset

1. [N] Nouns
2. [V] Verbs
3. [PR] Pronouns
4. [JJ] Adjectives
5. [RB] Adverbs
6. [PP] Postpositions
7. [PL] Participles
8. [QT] Quantifiers
9. [RP] Particles
10. [PU] Punctuations

- Vary in number of tags: a dozen to over 200.
- Size of tag sets depends on l**anguage, objectives and purpose**

POS tagging- Level of Detail



- In coarse grained level, we can identify a word as a noun but we can't tell whether it is a singular noun or a plural noun.
- In fine grained level, we can give grammatical details of the word. For e.g. verb is belong to past tense, future or present etc.
- In very fine grained level, too many part of speech tags which leads to confusion.

Steps involved in POS tagging

1. **Collect a dataset of annotated text:** This dataset will be used to train and test the POS tagger. The text should be annotated with the correct POS tags for each word.
2. **Preprocess the text:** This may include tasks such as tokenization (splitting the text into individual words), lowercasing, and removing punctuation.
3. **Divide the dataset into training and testing sets:** The training set will be used to train the POS tagger, and the testing set will be used to evaluate its performance.
4. **Train the POS tagger:** This may involve building a statistical model, such as a hidden Markov model (HMM), or defining a set of rules for a rule-based or transformation-based tagger. The model or rules will be trained on the annotated text in the training set.
5. **Test the POS tagger:** Use the trained model or rules to predict the POS tags of the words in the testing set. Compare the predicted tags to the true tags and calculate metrics such as precision and recall to evaluate the performance of the tagger.
6. **Fine-tune the POS tagger:** If the performance of the tagger is not satisfactory, adjust the model or rules and repeat the training and testing process until the desired level of accuracy is achieved.
7. **Use the POS tagger:** Once the tagger is trained and tested, it can be used to perform POS tagging on new, unseen text. This may involve preprocessing the text and inputting it into the trained model or applying the rules to the text. The output will be the predicted POS tags for each word in the text.

POS tagging using nltk

```
# Importing the NLTK library
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag


# Sample text
text = "NLTK is a powerful library for natural language processing."


# Performing PoS tagging
pos_tags = pos_tag(words)


# Displaying the PoS tagged result in separate lines
print("Original Text:")
print(text)


print("\nPoS Tagging Result:")
for word, pos_tag in pos_tags:
    print(f"{word}: {pos_tag}")
```

```
Original Text:
NLTK is a powerful library for natural language
processing.
PoS Tagging Result:
NLTK: NNP
is: VBZ
a: DT
powerful: JJ
library: NN
for: IN
natural: JJ
language: NN
processing: NN
```

POS tagging using spacy

```
!pip install spacy
!python -m spacy download en_core_web_sm
```

```
#importing libraries
import spacy


# Load the English language model
nlp = spacy.load("en_core_web_sm")


# Sample text
text = "SpaCy is a popular natural language processing library."


# Process the text with SpaCy
doc = nlp(text)


# Display the PoS tagged result
print("Original Text: ", text)
print("PoS Tagging Result:")
for token in doc:
    print(f"{token.text}: {token.pos_}")
```

```
Original Text: SpaCy is a popular natural language
processing library.
PoS Tagging Result:
SpaCy: PROPN
is: AUX
a: DET
popular: ADJ
natural: ADJ
language: NOUN
processing: NOUN
library: NOUN
.: PUNCT
```

Penn Treebank Tagset

The **Penn Treebank Tagset** is a **set of Part-of-Speech (POS) tags** developed for the **Penn Treebank Project** at the University of Pennsylvania.

- It's widely used in **Natural Language Processing (NLP)** for labeling words with their grammatical categories.
- Provides **45 POS tags** to cover English words (nouns, verbs, adjectives, adverbs, etc.).
- The **Penn Treebank Project** (by Univ. of Pennsylvania) originally defined **45 POS tags**.
- Sometimes you see **36 tags** in NLP tutorials or tools → that's a **simplified version** (collapsed categories).
- More fine-grained than the **basic 8 parts of speech** → helps machines understand grammar more accurately.
- The **Penn Treebank tagset,** has been applied to the Brown corpus and a number of other corpora.
- The Penn Treebank tagset was selected from the original **87-tag tagset for the Brown corpus**.

Penn Treebank Tagset

Full Penn Treebank Tagset = 45 tags

- Includes fine distinctions like:
  - VB, VBD, VBG, VBN, VBP, VBZ (different verb forms)
  - NN, NNS, NNP, NNPS (singular/plural + common/proper nouns)
  - PRP, PRP$, WP, WP$ (different pronoun types)
- Simplified / Universal Tagset = 36 tags
  - Merges some categories to make tagging easier.
  - Example: All verb forms → just VERB.

Penn Treebank (45 tags):

- “runs” → VBZ
- “running” → VBG
- “ran” → VBD

Simplified Tagset (36 tags):

- “runs”, “running”, “ran” → all simply **VERB**

Penn Treebank Tagset

Nouns

- **NN** → noun, singular (*cat, table*)
- **NNS** → noun, plural (*cats, tables*)
- **NNP** → proper noun, singular (*Alice, London*)
- **NNPS** → proper noun, plural (*Americans, Europeans*)

Pronouns

- **PRP** → personal pronoun (*he, she, it, they*)
- **PRP$** → possessive pronoun (*my, your, their*)
- **WP** → wh-pronoun (*who, what, whom*)
- **WP$** → possessive wh-pronoun (*whose*)

Verbs

- **VB** → verb, base form (*run, eat*)
- **VBD** → verb, past tense (*ran, ate*)
- **VBG** → verb, gerund/present participle (*running, eating*)
- **VBN** → verb, past participle (*eaten, driven*)
- **VBP** → verb, non-3rd person singular present (*run, eat*)
- **VBZ** → verb, 3rd person singular present (*runs, eats*)

Adjectives

- **JJ** → adjective (*happy, tall*)
- **JJR** → adjective, comparative (*happier, taller*)
- **JJS** → adjective, superlative (*happiest, tallest*)

Adverbs

- **RB** → adverb (*quickly, very*)
- **RBR** → comparative adverb (*faster, better*)
- **RBS** → superlative adverb (*fastest, best*)

Determiners / Articles

- **DT** → determiner (*a, an, the, this, those*)
- **PDT** → predeterminer (*all, both, half*)
- **WDT** → wh-determiner (*which, that*)

Conjunctions

- **CC** → coordinating conjunction (*and, but, or*)
- **IN** → subordinating conjunction/preposition (*in, on, because, if*)

Others

- **TO** → the word *to* (*to go, to eat*)
- **MD** → modal verb (*can, must, should*)
- **EX** → existential *there* (*there is, there are*)
- **FW** → foreign word
- **UH** → interjection (*oh!, wow!*)
- **SYM** → symbol (*$, %, +*)
- **CD** → cardinal number (*one, two, 100*)

Numbe	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer

Number	Tag	Description
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

POS tagging can be challenging due to several factors:

1. **Ambiguity:** Many words can have multiple meanings or parts of speech, making it difficult to determine the correct tag. For example, the word "play" can be a noun, verb, or adjective.
2. **Contextual Dependence:** The correct POS tag for a word often depends on the surrounding context. Words that are nouns in one sentence might be verbs in another.
   - Noun: *She is reading a* ***book***
   - Verb: *Please* ***book*** *a table for dinner.*
3. **Language Complexity:** Languages with complex grammatical structures, such as English, can have intricate rules for POS tagging.
4. **Morphological Complexity:** Languages with rich morphology, such as English and German, have many word forms that can affect POS tagging.
5. **Idioms and Colloquialisms:** These expressions can deviate from standard grammatical rules, making POS tagging more challenging.

Why is Tagging Hard?

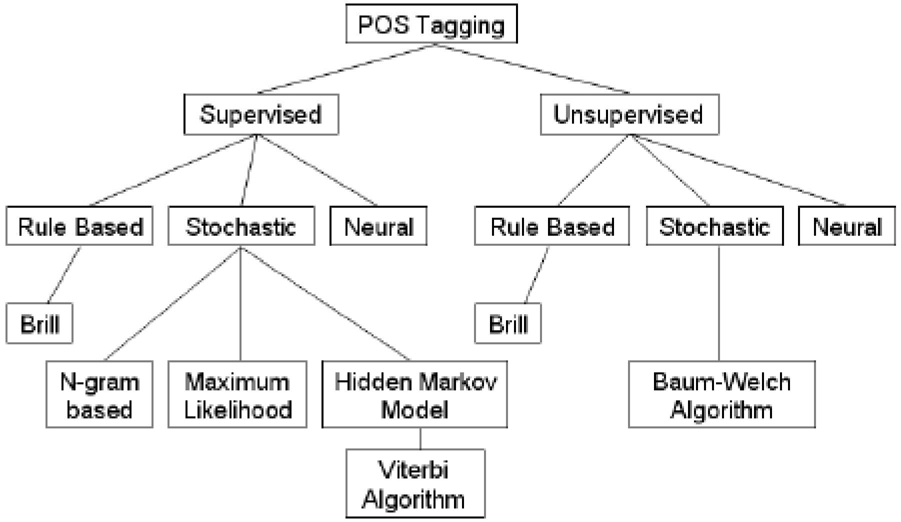
- Example
  1. Book/VB that/DT flight/NN
  2. Does/VBZ that/DT flight/NN serve/VB dinner/NN
- Tagging is a t**ype of disambiguation**
- Book can be NN or VB
- Can I read a book on this flight?
- That can be a DT or complementizer

The Problem

- Words often have more than one word class: **this**
- **This** is a nice day = PRP (Personal pronoun)
- **This** day is nice = DT
- You can go **this** far = RB (Adverb)

Classification of POS tagging approaches

- **Rule based POS tagging**
- **Stochastic POS tagging**
- **Transformation based Tagging**
- **Deep Learning Based**



Techniques for POS tagging

1. **Rule-based POS tagging**: The rule-based POS tagging models apply a set of handwritten rules and use contextual information to assign POS tags to words. These rules are often known as context frame rules.
2. **Transformation Based Tagging:** The transformation-based approaches use a pre-defined set of handcrafted rules as well as automatically induced rules that are generated during training.
3. **Stochastic (Probabilistic) POS tagging:** POS tags based on the probability of a particular tag sequence occurring.
4. **Deep learning models**: Various Deep learning models have been used for POS tagging such as Meta-BiLSTM which have shown an impressive accuracy of around 97 percent.

Rule based POS tagging

- One of the **oldest techniques** of tagging
- Rule-based taggers use **dictionary or lexicon** for getting possible tags for tagging each word.
- If the word has more than one possible tag, then rule-based taggers **use hand-written rules t**o identify the correct tag.
- Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with **its preceding as well as following words**.
- For example, suppose if the preceding word of a word is article then word must be a noun.
- If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective.
- Word ending with “ed” or “ing” must be assigned to a verb

Rule based POS tagging

- Basic Idea:
  - Assign all possible tags to words
  - Remove tags according to set of rules.
  - Typically more than 1000 hand-written rules, but may be machine-learned.

Rules

- Here is an example of how a rule-based POS tagger might work:

1. Define a set of rules for assigning POS tags to words. For example:

- If the word ends in "-tion," assign the tag "noun."
- If the word ends in "-ment," assign the tag "noun."
- If the word is all uppercase, assign the tag "proper noun."
- If the word is a verb ending in "-ing," assign the tag "verb."

2. Iterate through the words in the text and apply the rules to each word in turn.
3. For example:

- "Nation" would be tagged as "noun" based on the first rule.
- "Investment" would be tagged as "noun" based on the second rule.
- "UNITED" would be tagged as "proper noun" based on the third rule.
- "Running" would be tagged as "verb" based on the fourth rule.

3. Output the POS tags for each word in the text.

Rule based POS tagging : Stage 1

First Stage: In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.

FOR each word
   Get all possible parts of speech using a morphological analysis algorithm

Example

			NN		
			RB		
	VBN		JJ		VB
PRP	VBD	TO	VB	DT	NN
She	**promised**	**to**	**back**	**the**	**bill**

VBD past tense

VBN Past Participle

Rule based POS tagging : Stage 2

In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

Apply rules to remove possibilities

Example Rule:

IF VBD is an option and VBN|VBD follows "<start>PRP"
THEN Eliminate VBN

			NN		
			RB		
	VBN		JJ	VB	
PRP	VBD	TO	VB	DT	NN
She	**promised**	**to**	**back**	**the**	**bill**

Example: Verb vs. Noun (Context of Sentence)

Sentence: *She runs fast.*

Step 1: Assigning Possible Tags

- "She" could be a **PRP** (pronoun).
- "runs" could be a **VB** (verb) or **NNS** (noun, plural).
- "fast" could be an **RB** (adverb) or **JJ** (adjective).

Step 2: Applying Rules

Rule Applied:

- **If** the word is a verb (**VB**) and the next word is an adverb (**RB**),
  Then eliminate noun tags (**NNS**) for the word.
- In this case, "runs" is followed by "fast" (an adverb), so "runs" is tagged as **VB** (verb), and "fast" is tagged as **RB** (adverb).

Final Tags:

- "She" → **PRP** (pronoun)
- "runs" → **VB** (verb)
- "fast" → **RB** (adverb)

Example 2: Determiners and Nouns

Sentence: *The dog is running.*

Step 1: Assigning Possible Tags

- "The" is tagged as **DT** (determiner).
- "dog" could be a **NN** (noun, singular) or **VB** (verb).
- "is" is tagged as **VBZ** (verb, third-person singular).
- "running" is tagged as **VBG** (verb, gerund/present participle).

Step 2: Applying Rules

Rule Applied:

- **If** the word is a determiner (**DT**) and is followed by a noun (**NN**, **NNS**),
  Then eliminate verb tags for the following word.
- Since "dog" is a singular noun and follows the determiner "The," the verb tags for "dog" are eliminated, and it is tagged as **NN** (noun).

Final Tags:

- "The" → **DT** (determiner)
- "dog" → **NN** (noun)
- "is" → **VBZ** (verb, third-person singular)
- "running" → **VBG** (verb, gerund)

Example 3: Verb Forms with Infinitives

Sentence: *She promised to leave.*

Step 1: Assigning Possible Tags

- "She" is tagged as **PRP** (pronoun).
- "promised" is tagged as **VBD** (verb, past tense).
- "to" is tagged as **TO** (infinitive marker).
- "leave" is tagged as **VB** (verb, base form).

Step 2: Applying Rules

Rule Applied:

- **If** the word is a verb (**VBD**) and is followed by "to" (infinitive marker),
  Then the verb must be in its base form (**VB**).
- Since "promised" is followed by "to" (which marks an infinitive), "leave" must be tagged as **VB** (verb, base form).

Final Tags:

- "She" → **PRP** (pronoun)
- "promised" → **VBD** (verb, past tense)
- "to" → **TO** (infinitive marker)
- "leave" → **VB** (verb, base form)

Example 4: Pronouns and Verb Agreement

Sentence: *They play soccer.*

Step 1: Assigning Possible Tags

- "They" is tagged as **PRP** (pronoun).
- "play" could be a **VB** (verb) or **NNS** (noun, plural).
- "soccer" is tagged as **NN** (noun, singular).

Step 2: Applying Rules

Rule Applied:

- **If** the word is a pronoun (**PRP**) and is followed by a verb **(VB**),
  Then ensure that the verb agrees with the subject in number (plural subject "They" → plural verb).
- Since "They" is plural, the verb "play" should be tagged as **VB** (verb) to match the plural subject.

Final Tags:

- "They" → **PRP** (pronoun)
- "play" → **VB** (verb)
- "soccer" → **NN** (noun)

Example 5: Modal Verbs (type of auxiliary verb that express possibility, ability, necessity, or intent)

Sentence: *She can swim.*

Step 1: Assigning Possible Tags

- "She" is tagged as **PRP** (pronoun).
- "can" is tagged as **MD** (modal verb).
- "swim" is tagged as **VB** (verb, base form).

Step 2: Applying Rules

Rule Applied:

- **If** the word is a modal verb (**MD**) and the following word is a verb,
  Then the following verb must be in its base form (**VB**).
- Since "can" is a modal verb, "swim" must be tagged as **VB** (base form).

Final Tags:

- "She" → **PRP** (pronoun)
- "can" → **MD** (modal verb)
- "swim" → **VB** (verb, base form)

Example 6: Prepositions vs. infinitive marker

Sentence: *She walked to the park.*

Step 1: Assigning Possible Tags

- "She" is tagged as **PRP** (pronoun).
- "walked" is tagged as **VBD** (verb, past tense).
- "to" could be tagged as **TO** (infinitive marker) or **IN** (preposition).
- "the" is tagged as **DT** (determiner).
- "park" is tagged as **NN** (noun, singular).

Step 2: Applying Rules

Rule Applied:

- **If** the word "to" is followed by a noun phrase (e.g., a determiner followed by a noun),
  Then tag "to" as a preposition (**IN**).
- Since "to" is followed by "the park" (a noun phrase), it is correctly tagged as a preposition (**IN**).

Final Tags:

- "She" → **PRP** (pronoun)
- "walked" → **VBD** (verb, past tense)
- "to" → **IN** (preposition)
- "the" → **DT** (determiner)
- "park" → **NN** (noun)

Rule-Based POS Tagging

- **Advantages:**
  - Precise and accurate for well-defined languages and domains.
  - High Interpretability (Transparent Rules)
    - Every tagging decision is based on clear linguistic rules.
    - Easy to explain why a word got a particular tag.
  - No Large Training Data Required
  - **Deterministic & Consistent**: Same input always gives the same output.
- **Disadvantages:**
  - Time-consuming to create and maintain, inflexible for new languages or domains.
  - Hard coded rules required
  - Need to refer dictionary every time
  - Language experts are needed in case of disambiguation

Stochastic Tagging

- The use of probabilities in tags is quite old. **Stochastic taggers use probabilistic and statistical information to assign tags to words.**
- The model that includes **frequency or probability** (statistics) can be called **stochastic.**
- These taggers might use '**tag sequence probabilities**', '**word frequency measurements**' or a combination of both.
- The tag encountered most frequently in the training set is the one assigned to an ambiguous instance of that word (word frequency measurements).

Stochastic Tagging

- A stochastic approach includes **frequency, probability or statistics.**
- The simplest stochastic approach finds out the most frequently used tag for a specific word in the annotated training data and uses this information to tag that word in the unannotated text.
- The key idea is that the probability of a word belonging to a particular POS tag depends on the context, such as the surrounding words and the overall sequence of tags.

Common Stochastic Tagging Models:

1. **N-gram Models:**
   - Predict the POS tag of a word based on the n previous words in the sentence.
   - Simple but often effective, especially for short-range dependencies.
2. **Hidden Markov Models (HMMs):**
   - It’s called “Markov” because the model assumes the future depends only on the current state, not the full history.
   - It’s called “Hidden” because the states are not directly observed — only their effects (observations) are visible.
   - define transitions between states and how states emit observations
3. **Conditional Random Fields (CRFs):**
   - Undirected graphical models that can **capture dependencies between POS tags and words** in a more flexible way than HMMs.
   - Consider the **entire sentence context** and avoid the independence assumption of HMMs.

How Stochastic Tagging Works:

1. **Training:** A model is trained on a large annotated corpus, where each **word is labeled with its correct POS tag.**
2. **Tagging:** Given a new sentence, the model **calculates probability of each word belonging to different POS tags based on the training data & the context**.
3. **Assignment:** The model assigns the POS tag to each word that has the **highest probability**.
4. There would be no probability for the words that **do not exist in the corpus.**

Stochastic Tagging

Stochastic tagger applies the following approaches for POS tagging −

1. **Word Frequency Approach**

- The stochastic taggers disambiguate the words based on the **probability that a word occurs with a particular tag**.
- 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.

2. Tag Sequence Probabilities

- 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.

Key Components of an HMM for POS Tagging:

- **States:** Represent the possible POS tags (e.g., noun, verb, adjective).
- **Observations:** Represent the words in the sentence.
- **Transition Probabilities:** Define the probability of transitioning from one state (POS tag) to another.
- **Emission Probabilities:** Define the probability of a word being emitted from a particular state (POS tag).

The HMM Tagging Process

The HMM Tagger applies **dynamic programming** (specifically the **Viterbi algorithm**) to find the most likely sequence of tags for a given sequence of words.

Step 1: Training the Model:

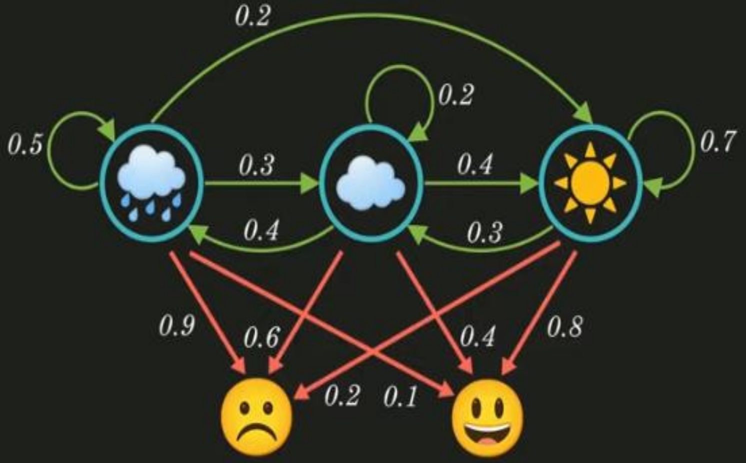
- The first step is to **train the HMM** using a labeled training corpus. This training process calculates the **transition probabilities** and **emission probabilities**.

Transition probabilities (A) represent the probability of moving from one POS tag to another in a sentence. For example, the probability of a **noun (NN)** being followed by a **verb (VB)** might be 0.3.

Emission probabilities (B) represent the probability of a word being tagged with a particular POS.

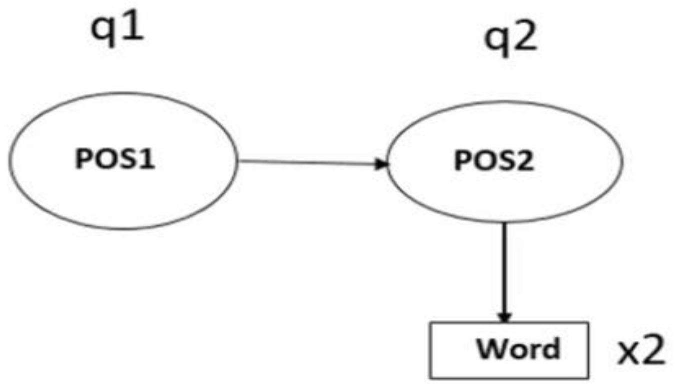
Step 2: Using the Viterbi Algorithm: Once the model is trained, the **Viterbi algorithm** is used to assign the best sequence of tags for a new sentence.

The algorithm selects the tag that maximizes the combined probability of the word-tag emissions and the tag sequence transitions.



$$\begin{array}{c|ccc} & \text{🌧} & \text{☁} & \text{☀} \\ \hline \text{🌧} & 0.5 & 0.3 & 0.2 \\ \text{☁} & 0.4 & 0.2 & 0.4 \\ \text{☀} & 0.0 & 0.3 & 0.7 \end{array} \qquad \begin{array}{cc} \text{☹} & \text{😃} \\ \hline 0.9 & 0.1 \\ 0.6 & 0.4 \\ 0.2 & 0.8 \end{array}$$

Viterbi Algorithm



q1 and q2: States (class)

x2: word emitted from q2 state

$$P(POS2|word, POS1) = P(word|POS2) * P(POS2|POS1)$$

Bayes Rule **Bigram**

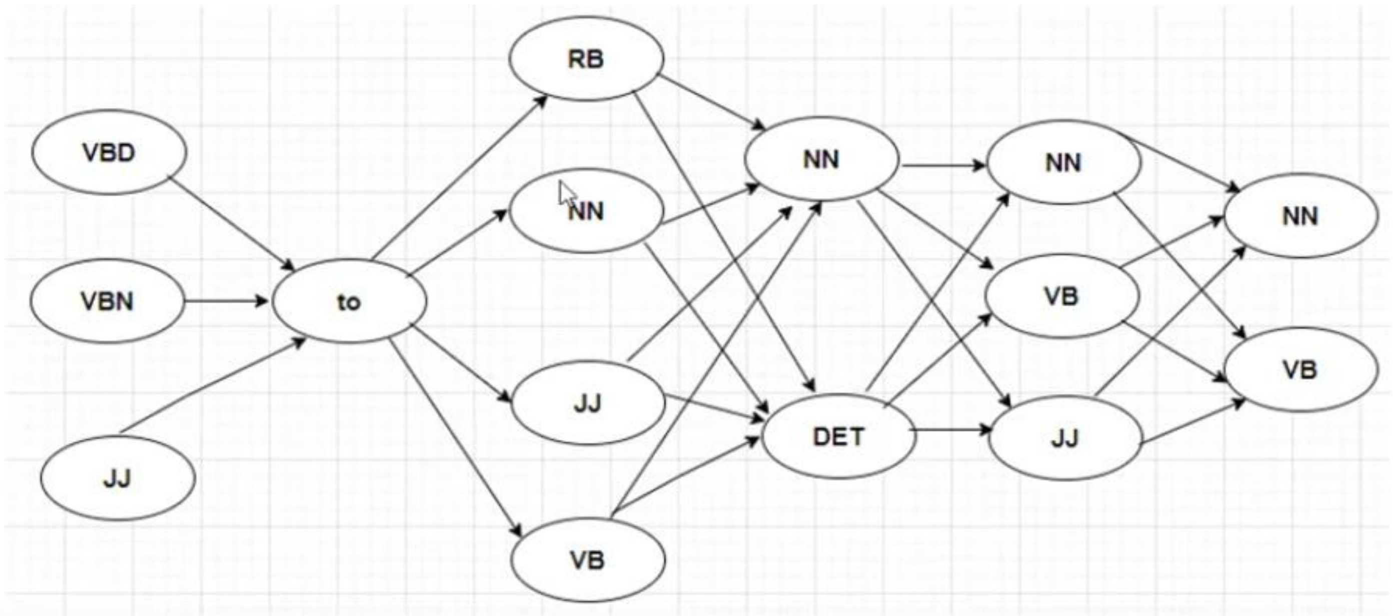
Bayes Rule data obtained from Emission Probability matrix while Bigram data is obtained from State Transition Matrix

Need of Viterbi Decoding Algorithm:

- It is based on dynamic programming.
- Dynamic programming breakdowns the problems into sub-problems.
- It saves the result for future purposes therefore no not need to compute the result again.

Statement is : **Assured to back the silver coin**

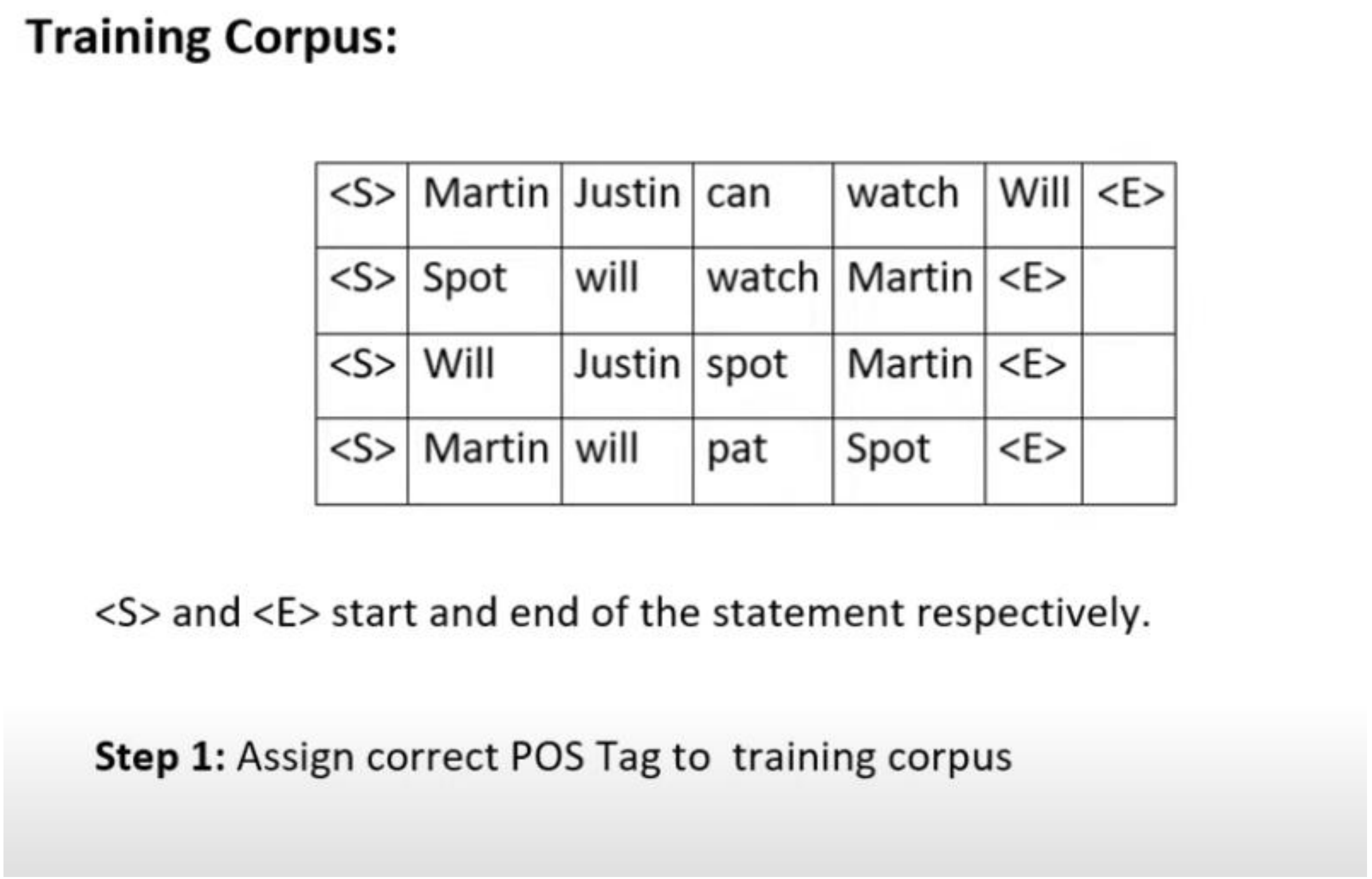
assured	**to**	**back**	**the**	**silver**	**coin**
W1	**W2**	**W3**	**W4**	**W5**	**W6**
VBD: Past Tense	To: to	RB: Adverb	NN: Noun	NN: Noun	NN: Noun
VBN: Participle		NN: Noun	Determiner	JJ: Adjective	VB: Verb
JJ: Adjective		JJ: Adjective		VB: Verb	
		VB: Verb			
(3)	(1)	(4)	(2)	(3)	(2)
Total number of ways are: 3×1×4×2×3×2= 144 possibilities are there.					



Total number of ways are: 3×1×4×2×3×2= 144 possibilities are there.

Viterbi algorithm is used to find efficient path.

Training Corpus



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>	

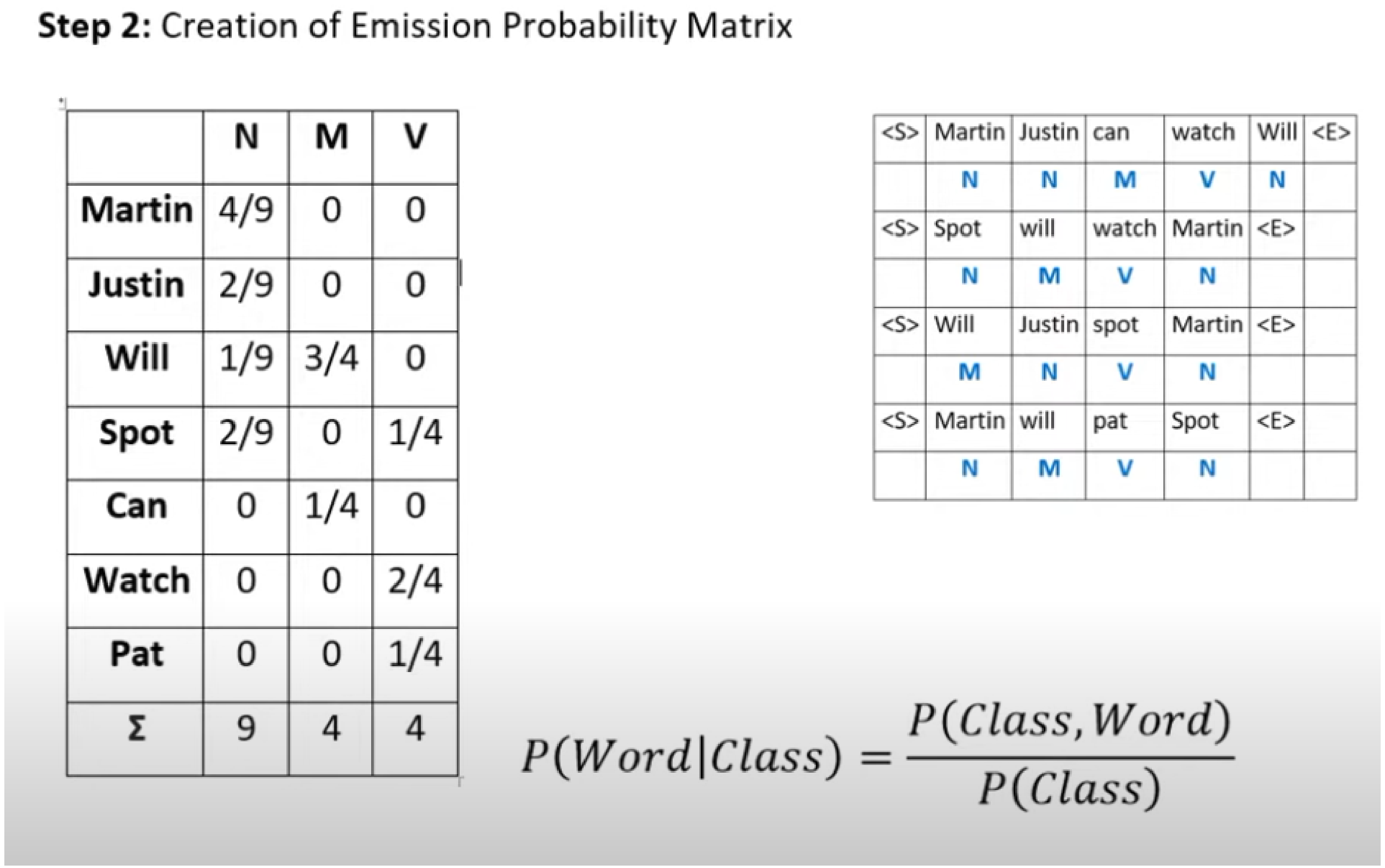
<S> and <E> start and end of the statement respectively.

Step 1: Assign correct POS Tag to training corpus

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



Step 2: Creation of Emission Probability Matrix

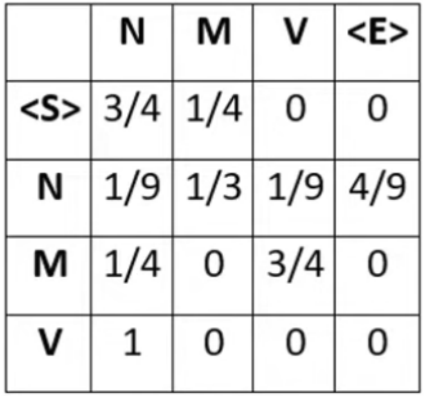
	N	M	V
Martin	4/9	0	0
Justin	2/9	0	0
Will	1/9	3/4	0
Spot	2/9	0	1/4
Can	0	1/4	0
Watch	0	0	2/4
Pat	0	0	1/4
Σ	9	4	4

<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		

$$P(Word|Class) = \frac{P(Class, Word)}{P(Class)}$$

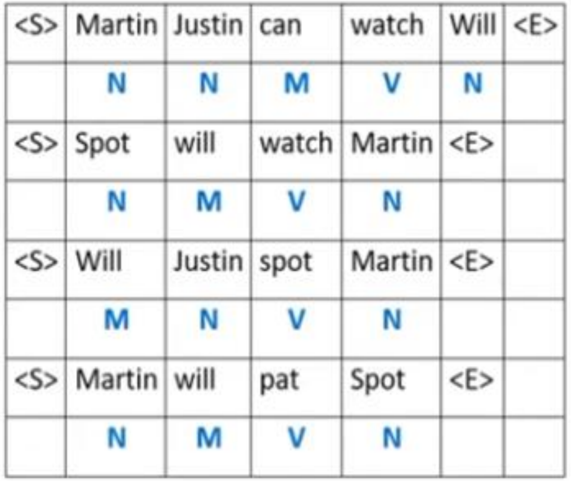
Emission probabilities represent the probability of a word being tagged with a particular POS.

Step 3: Creation of State Transition Probability Matrix



	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

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



<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		

represent the probability of moving from one POS tag to another in a sentence. For example, the probability of a **noun (N)** being followed by a **verb (V)** etc.

Test Data

<S> Justin will spot Will <E>

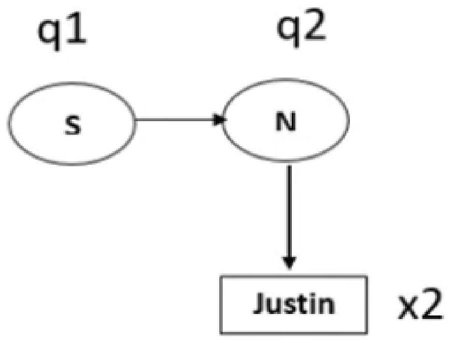
Justin	will	Spot	Will
N	N	V	N
	M	N	M
Total ways : 1×2×2×2=8			

$$\begin{tikzcd}
 & & N \arrow[r] \arrow[rdd] & V \arrow[r] \arrow[rdd] & N \arrow[rd] & \\
S \arrow[r] & N \arrow[ru] \arrow[rd] & & & & E \\
 & & M \arrow[r] \arrow[ruu] & N \arrow[r] \arrow[ruu] & M \arrow[ru] &
\end{tikzcd}$$

$$P(POS2|word, POS1) = P(word|POS2) * P(POS2|POS1)$$

Bayes Rule **Bigram**

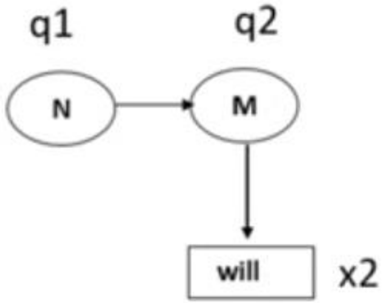
Justin



$$P(N|Justin, < S >) = P(Justin|N) * P(Noun| < S >)$$

$$P(N|Justin, < S >) = \frac{2}{9} \times \frac{3}{4} = \frac{1}{6}$$

Will as Modal Verb

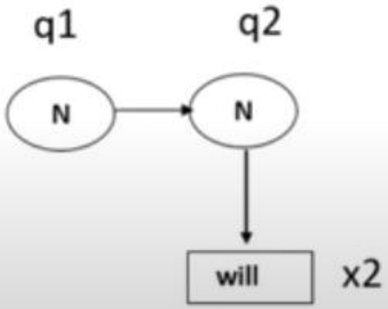


$$P(M|will,N) = P(will|M) * P(M|N)$$

$$P(M|will,N) = \frac{3}{4} \times \frac{1}{3} = \frac{1}{4}$$

$$P(M|will,N) = \frac{1}{4} \times \frac{1}{6} = \frac{1}{24}$$

Will as Noun



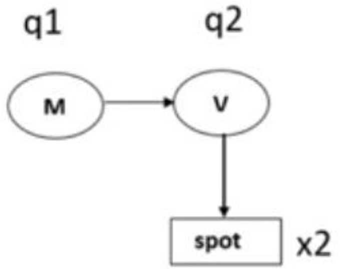
$$P(N|will,N) = P(will|N) * P(N|N)$$

$$P(N|will,N) = \frac{1}{9} \times \frac{1}{9} = \frac{1}{81}$$

$$P(N|will,N) = \frac{1}{81} \times \frac{1}{6} = \frac{1}{486}$$

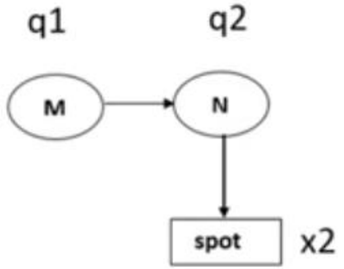
$\frac{1}{24} > \frac{1}{486}$ $P(M) > P(N)$ **Result: will as Modal Verb**

spot as Verb



$$P(v|spot,M) = P(spot|v) * P(v|M)$$
$$P(v|spot,M) = \frac{1}{4} \times \frac{3}{4} = \frac{3}{16}$$
$$P(v|spot,M) = \frac{3}{16} \times \frac{1}{24} = \frac{1}{128}$$

spot as Noun

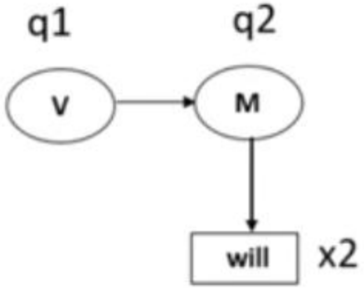


$$P(N|spot,M) = P(spot|N) * P(N|M)$$
$$P(N|spot,M) = \frac{2}{9} \times \frac{1}{4} = \frac{1}{18}$$
$$P(N|spot,M) = \frac{1}{18} \times \frac{1}{24} = \frac{1}{432}$$

$$\frac{1}{128} > \frac{1}{432} \; P(V) > P(N)$$

Result: spot as verb

will as modal Verb

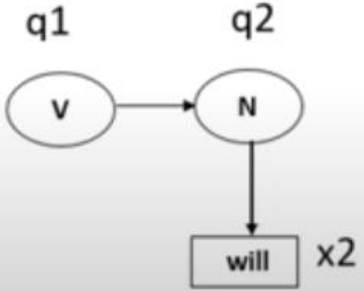


$$P(M|will,V) = P(will|M) * P(M|V)$$

$$P(M|will,V) = \frac{3}{4} \times \frac{1}{1000} = \frac{3}{4000}$$

$$P(M|will,V) = \frac{3}{4000} \times \frac{1}{128} = \frac{1}{512000}$$

will as Noun



$$P(N|will,V) = P(will|N) * P(N|V)$$

$$P(N|will,V) = \frac{1}{9} \times \frac{1}{1} = \frac{1}{9}$$

$$P(N|will,V) = \frac{1}{9} \times \frac{1}{128} = \frac{1}{1152}$$

Result: will as noun

Probability of will as Noun is more.

Result of POS tagging after applying HMM model

<S>	Justin	will	spot	Will	<E>
	N	M	V	N	

Example

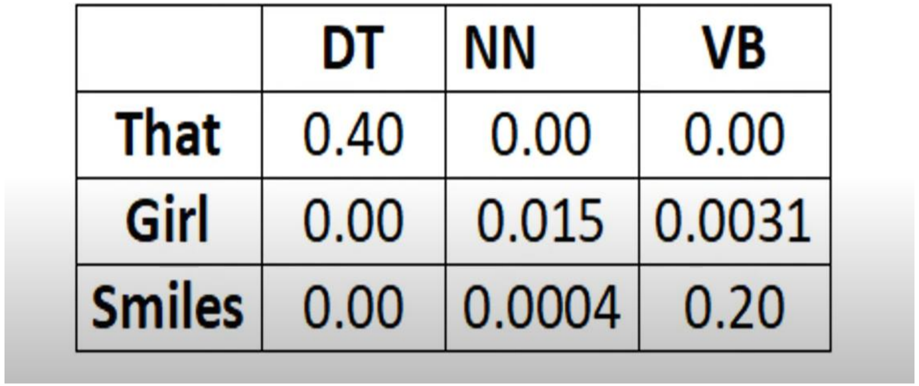
Given:

Emission Probability Matrix

State Transition Probability Matrix

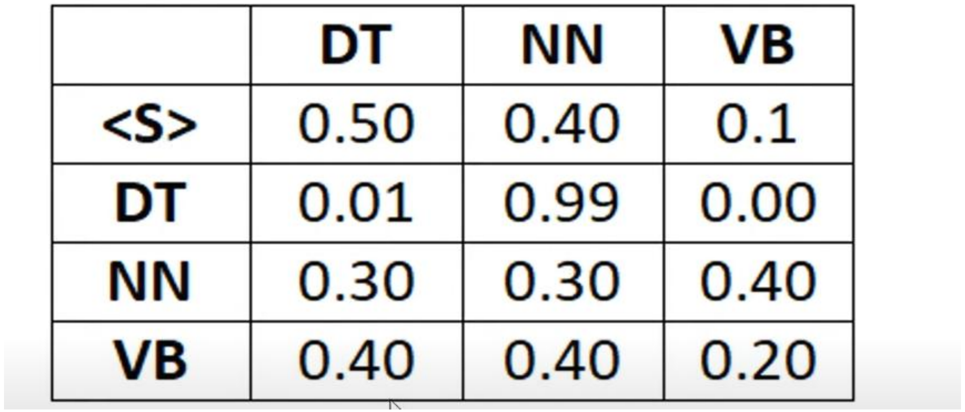
Test Data : <s> That girl smiles </s>

Solve- Emission Probability Matrix



	DT	NN	VB
That	0.40	0.00	0.00
Girl	0.00	0.015	0.0031
Smiles	0.00	0.0004	0.20

State Transition Probability Matrix



	DT	NN	VB
<S>	0.50	0.40	0.1
DT	0.01	0.99	0.00
NN	0.30	0.30	0.40
VB	0.40	0.40	0.20

Solution:

$$P(DT|that, <S>) = P(that|DT) \times P(DT|<S>)$$

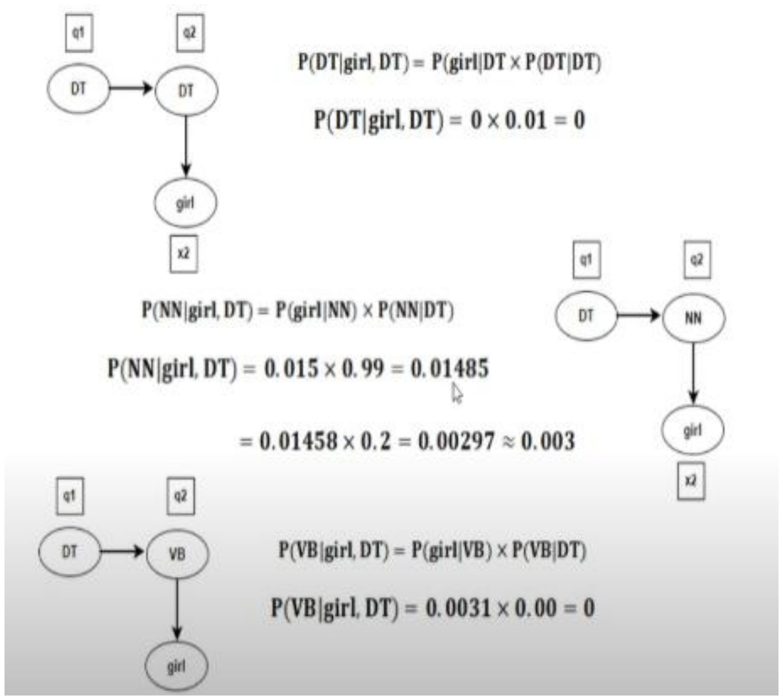
$$P(DT|that, <S>) = 0.40 \times 0.50 = 0.2$$

$$P(NN|that, <S>) = P(that|NN) \times P(NN|<S>)$$

$$P(NN|that, <S>) = 0.00 \times 0.40 = 0$$

$$P(VB|that, <S>) = P(that|VB) \times P(VB|<S>)$$

$$P(VB|that, <S>) = 0.00 \times 0.1 = 0$$



$P(DT|girl, DT) = P(girl|DT \times P(DT|DT)$

$P(DT|girl, DT) = 0 \times 0.01 = 0$

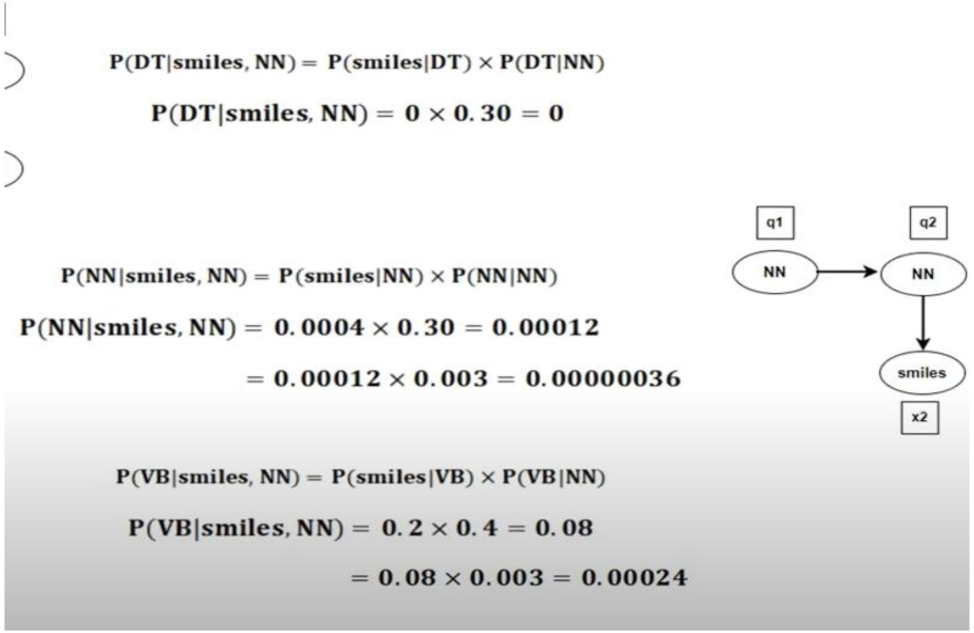
$P(NN|girl, DT) = P(girl|NN) \times P(NN|DT)$

$P(NN|girl, DT) = 0.015 \times 0.99 = 0.01485$

$= 0.01458 \times 0.2 = 0.00297 \approx 0.003$

$P(VB|girl, DT) = P(girl|VB) \times P(VB|DT)$

$P(VB|girl, DT) = 0.0031 \times 0.00 = 0$



$P(DT|smiles, NN) = P(smiles|DT) \times P(DT|NN)$

$P(DT|smiles, NN) = 0 \times 0.30 = 0$

$P(NN|smiles, NN) = P(smiles|NN) \times P(NN|NN)$

$P(NN|smiles, NN) = 0.0004 \times 0.30 = 0.00012$

$= 0.00012 \times 0.003 = 0.00000036$

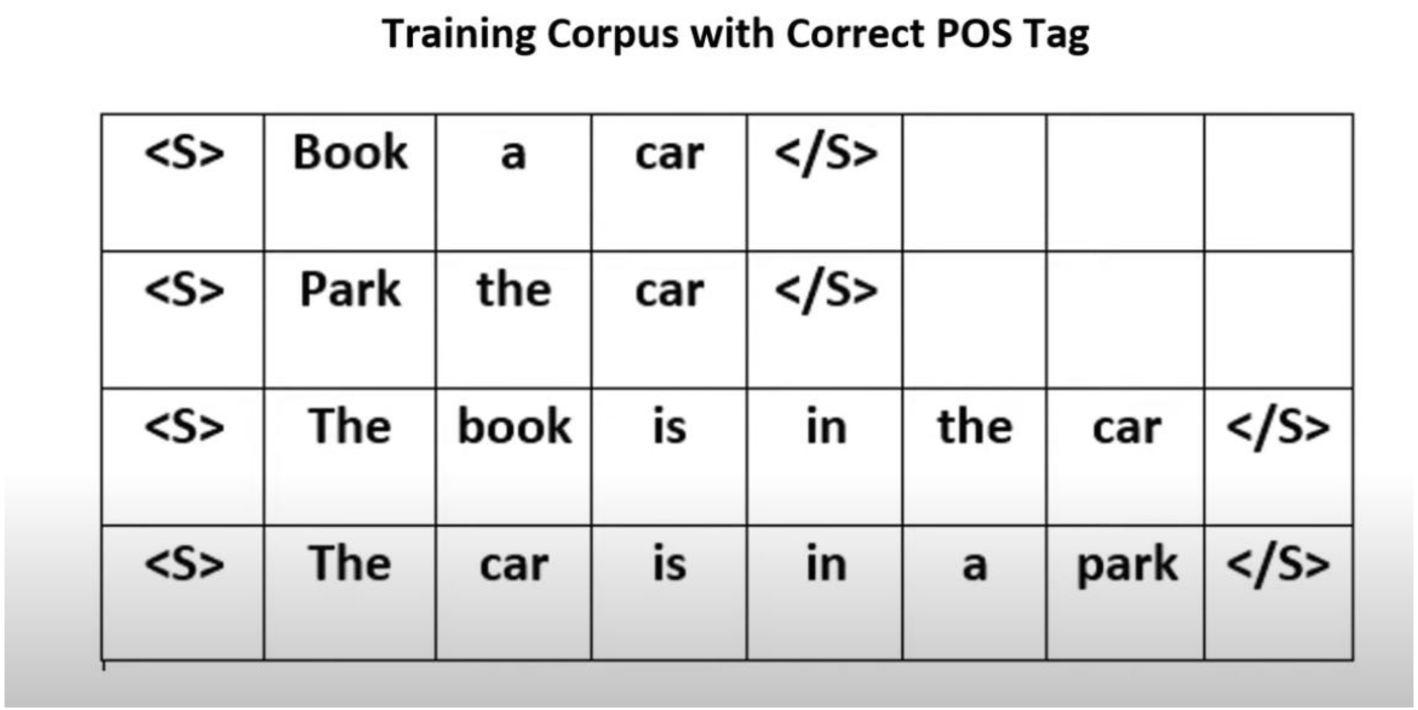
$P(VB|smiles, NN) = P(smiles|VB) \times P(VB|NN)$

$P(VB|smiles, NN) = 0.2 \times 0.4 = 0.08$

$= 0.08 \times 0.003 = 0.00024$

Smile is a verb

Girl is a Noun



Training Corpus with Correct POS Tag

<S>	Book	a	car	</S>			
<S>	Park	the	car	</S>			
<S>	The	book	is	in	the	car	</S>
<S>	The	car	is	in	a	park	</S>

Test Set: The Park is a Book

Step 1: Assign correct POS Tag to training corpus

<S>	Book	a	car	</S>			
	Verb	Det.	Noun				
<S>	Park	the	car	</S>			
	Verb	Det.	Noun				
<S>	The	book	is	in	the	car	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	
<S>	The	car	is	in	a	park	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	

POS Tags

Verb

Det

Noun

Preposition

Step 2: Creation of Emission Probability Matrix

	Verb (4)	Det. (6)	Noun (6)	Prep. (2)
Book	1/4	0	1/6	0
a	0	2/6	0	0
car	0	0	4/6	0
the	0	4/6	0	0
park	1/4	0	1/6	0
is	2/4	0	0	0
in	0	0	0	2/2
Σ	4	6	6	2

<S>	Book	a	car	</S>			
	Verb	Det.	Noun				
<S>	Park	the	car	</S>			
	Verb	Det.	Noun				
<S>	The	book	is	in	the	car	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	
<S>	The	car	is	in	a	park	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	

Step 3: Creation of State Transition Probability Matrix

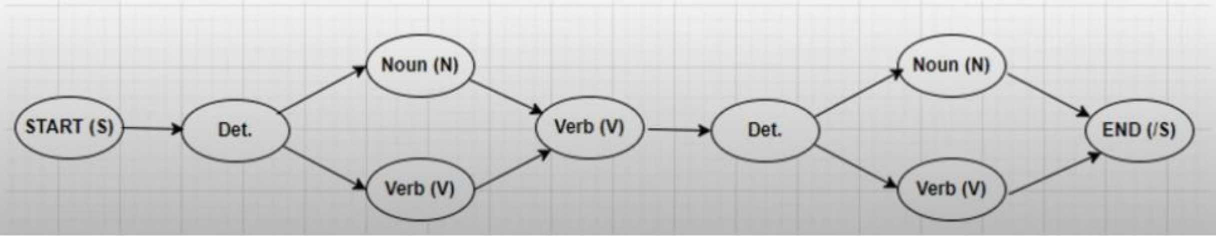
	Noun	Verb	Det.	Prep.	</S>
<S>	0	2/4	2/4	0	0
Noun	0	2/6	0	0	4/6
Verb	0	0	2/4	2/4	0
Det.	6/6	0	0	0	0
Prep.	0	0	2/2	0	0

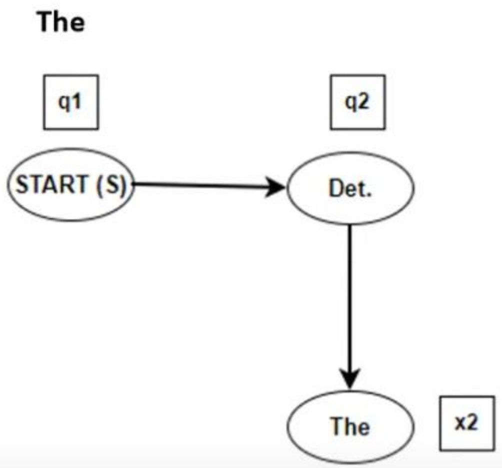
<S>	Book	a	car	</S>			
	Verb	Det.	Noun				
<S>	Park	the	car	</S>			
	Verb	Det.	Noun				
<S>	The	book	is	in	the	car	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	
<S>	The	car	is	in	a	park	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	

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

Test Data <S>The park is a book </S>

The	park	is	a	book
Det.	Noun	Verb	Det.	Noun
	Verb			Verb
1×2×1×1×2=4 ways are there				

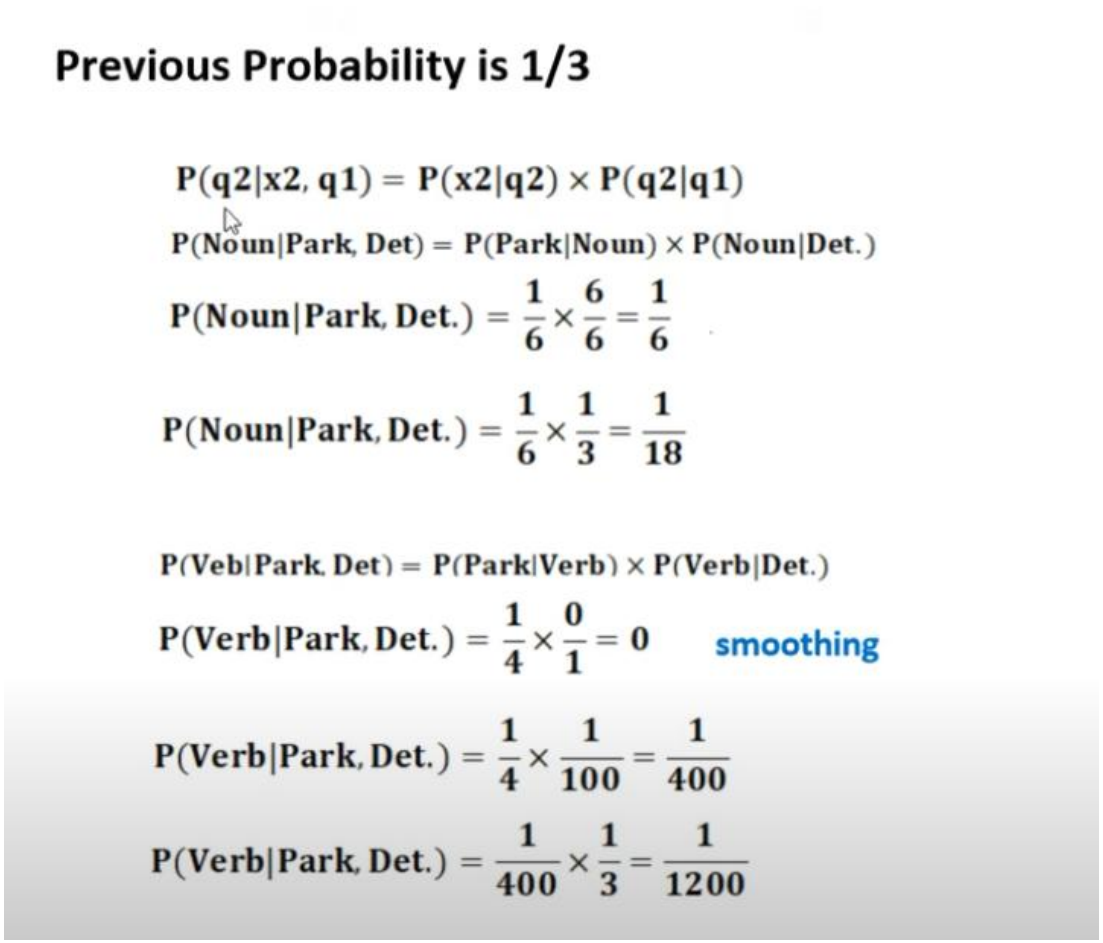




$$P(q2|x2, q1) = P(x2|q2) * P(q2|q1)$$

$$P(Det|The, <S>) = P(The|Det) * P(Det|<S>)$$

$$P(Det|The, <S>) = \frac{4}{6} \times \frac{2}{4} = \frac{1}{3}$$



Previous Probability is 1/3

$$P(q2|x2, q1) = P(x2|q2) \times P(q2|q1)$$

$$P(\text{Noun}|\text{Park, Det}) = P(\text{Park}|\text{Noun}) \times P(\text{Noun}|\text{Det.})$$

$$P(\text{Noun}|\text{Park, Det.}) = \frac{1}{6} \times \frac{6}{6} = \frac{1}{6}$$

$$P(\text{Noun}|\text{Park, Det.}) = \frac{1}{6} \times \frac{1}{3} = \frac{1}{18}$$

$$P(\text{Veb}|\text{Park, Det}) = P(\text{Park}|\text{Verb}) \times P(\text{Verb}|\text{Det.})$$

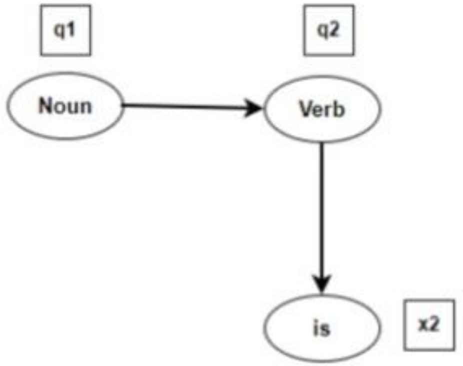
$$P(\text{Verb}|\text{Park, Det.}) = \frac{1}{4} \times \frac{0}{1} = 0$$ smoothing

$$P(\text{Verb}|\text{Park, Det.}) = \frac{1}{4} \times \frac{1}{100} = \frac{1}{400}$$

$$P(\text{Verb}|\text{Park, Det.}) = \frac{1}{400} \times \frac{1}{3} = \frac{1}{1200}$$

Park is a Noun

is as Verb



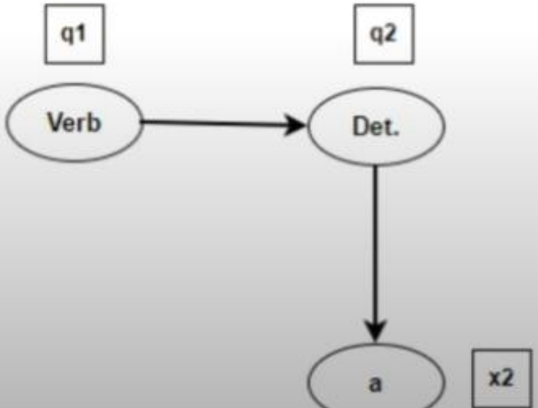
Previous Probability is 1/18

$$P(\text{Verb}|\text{is}, \text{Noun}) = p(\text{is}|\text{Verb}) \times P(\text{Verb}|\text{Noun})$$

$$P(\text{Verb}|\text{is}, \text{Noun}) = \frac{2}{4} \times \frac{2}{6} = \frac{1}{6}$$

$$= \frac{1}{6} \times \frac{1}{18} = \frac{1}{108}$$

a as Det



Previous Probability is 1/108

$$P(\text{Det.}|\text{a}, \text{Verb}) = P(\text{a}|\text{Det.}) \times P(\text{Det.}|\text{Verb})$$

$$P(\text{Det}|\text{a}, \text{Verb}) = \frac{2}{6} \times \frac{2}{4} = \frac{1}{6}$$

$$P(\text{Det.}|\text{a}, \text{Verb}) = \frac{1}{6} \times \frac{1}{108} = \frac{1}{648}$$

Previous Probability is 1/648

$$P(\text{Noun}|\text{book, Det.}) = P(\text{book}|\text{Noun}) \times P(\text{Noun}|\text{Det})$$

$$P(\text{Noun}|\text{book, Det.}) = \frac{1}{6} \times \frac{6}{6} = \frac{1}{6}$$

$$P(\text{Noun}|\text{book, Det.}) = \frac{1}{6} \times \frac{1}{648} = \frac{1}{3888}$$

$$P(\text{Noun}|\text{Verb, Det.}) = P(\text{book}|\text{Verb}) \times P(\text{Verb}|\text{Det})$$

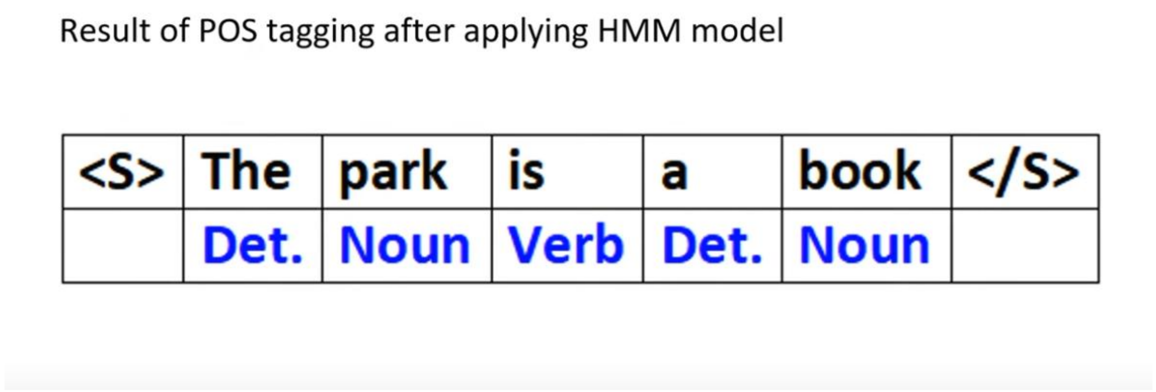
$$P(\text{Verb}|\text{book, Det.}) = \frac{1}{4} \times 0 = 0$$

$$P(\text{Noun}|\text{book, Det.}) = \frac{1}{4} \times \frac{1}{100} = \frac{1}{400}$$

$$P(\text{Noun}|\text{book, Det.}) = \frac{1}{400} \times \frac{1}{648} = \frac{1}{259200}$$

$$\frac{1}{3888} > \frac{1}{259200}$$

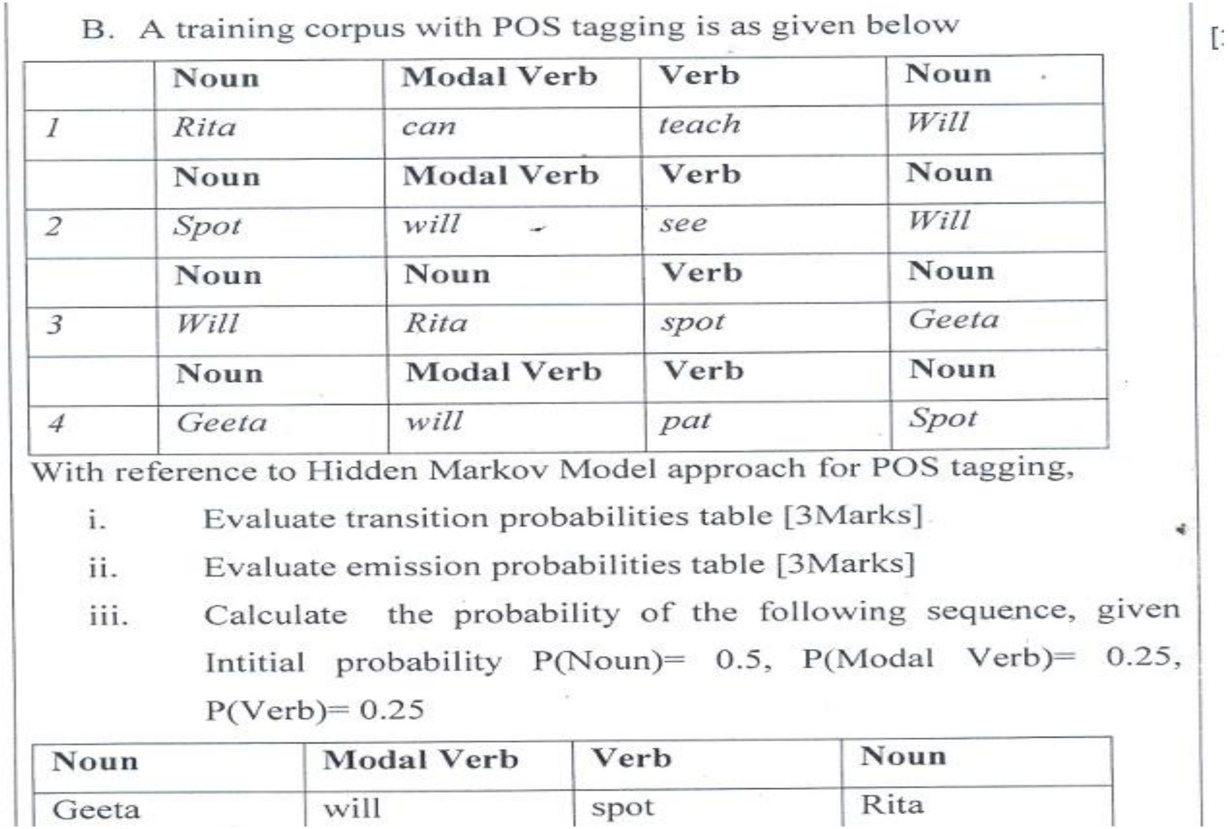
Result: book as Noun



Result of POS tagging after applying HMM model

<S>	The	park	is	a	book	</S>
	Det.	Noun	Verb	Det.	Noun	

Example 3



B. A training corpus with POS tagging is as given below

	Noun	**Modal Verb**	**Verb**	**Noun**
1	*Rita*	*can*	*teach*	*Will*
	Noun	**Modal Verb**	**Verb**	**Noun**
2	*Spot*	*will*	*see*	*Will*
	Noun	**Noun**	**Verb**	**Noun**
3	*Will*	*Rita*	*spot*	*Geeta*
	Noun	**Modal Verb**	**Verb**	**Noun**
4	*Geeta*	*will*	*pat*	*Spot*

With reference to Hidden Markov Model approach for POS tagging,

i. Evaluate transition probabilities table [3Marks]

ii. Evaluate emission probabilities table [3Marks]

iii. Calculate the probability of the following sequence, given Intitial probability P(Noun)= 0.5, P(Modal Verb)= 0.25, P(Verb)= 0.25

Noun	**Modal Verb**	**Verb**	**Noun**
Geeta	will	spot	Rita

[

Transformation-Based Tagging

- Transformation based tagging is also called **Brill tagging.** It is a rule-based algorithm for automatic tagging of POS to the given text.
- TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.
- **Combination of Rule-based and stochastic tagging**
- Like rule-based because rules are used to specify tags in a certain environment
- Like stochastic approach because machine learning is used—with tagged corpus as input. It applies hand-crafted or automatically learned rules (transformations) to correct mistakes in a systematic way.
- Input: tagged corpus

  dictionary (with most frequent tags)

  Usually constructed from the tagged corpus

Working of Transformation Based Learning (TBL)

steps to understand the working of TBL −

- **Start with the solution** − The TBL usually starts with some solution to the problem and works in cycles.
- **Most beneficial transformation chosen** − In each cycle, TBL will choose the most beneficial transformation.
- **Apply to the problem** − The transformation chosen in the last step will be applied to the problem.

The algorithm will stop when the selected transformation in step 2 will not add either more value or there are no more transformations to be selected.

- **Here’s how the process works:**

1. **Initialization**: The algorithm begins by assigning a preliminary tag to each word in the text. This can be done using a **dictionary lookup** or by assigning the **most common POS tag to each word**.
2. **Transformation Rules**: The system then applies a series of transformation rules to improve the accuracy of the initial tagging. These **rules are learned automatically from a tagged corpus**. Each rule specifies a condition under which a word's tag should be changed (e.g., "If a word is tagged as a noun and is preceded by a determiner, change its tag to an adjective"). Example
3. **Iterative Refinement**: The tagging process iteratively applies the transformation rules to the text, refining the tags with each pass. The rules are applied in a sequence determined by their effectiveness in reducing tagging errors.
4. **Final Output**: After several iterations, the tagging stabilizes, producing a final set of tags that should be more accurate than the initial ones.

Example

"If a word is tagged as a noun and is preceded by a determiner, change its tag to an adjective"

Sentence: **"The school bus arrived."**

- Original POS tags:
  - The (DT)
  - school (NN)
  - bus (NN)
- Rule applied: "school" is a noun (NN) preceded by a determiner (DT → "The"), so we re-tag it as an adjective (JJ).
- Modified POS tags:
  - The (DT)
  - school (JJ)
  - bus (NN)

Example

“Change NN to VB when previous tag is TO”

Is expected to rain tomorrow

is/VBZ

expected/VBN

to/TO

rain/NN

tomorrow/NN

After applying rule:

is/VBZ

expected/VBN

to/TO

rain/VB

tomorrow/NN

Algorithm

Step 1: Label every word with most likely tag (from dictionary)

Step 2: Check every possible transformation & select one which most improves tagging

Step 3: Re-tag corpus applying the rules

Repeat 2-3 until some criterion is reached, e.g., X% correct with respect to training corpus

RESULT: Sequence of transformation rules

Transformation rules make changes to tags

Example Sentence:
"The quick brown fox jumps over the lazy dog."
Step 1: Initial Tagging
Assume we start with an initial tagging where each word is assigned its most common POS tag:
- **The** (Determiner, DT)
- **quick** (Noun, NN)
- **brown** (Noun, NN)
- **fox** (Noun, NN)
- **jumps** (Verb, VBZ)
- **over** (Preposition, IN)
- **the** (Determiner, DT)
- **lazy** (Noun, NN)
- **dog** (Noun, NN)

Step 2: Apply Transformation Rules
Now, let's apply some transformation rules that were learned from a tagged corpus to correct the initial tagging.
Rule 1: Change "NN" (Noun) to "JJ" (Adjective) when it appears before another noun.
- Apply this rule to **quick** and **brown**:
    - **quick**: NN → JJ (because it appears before "fox" which is a noun)
    - **brown**: NN → JJ (because it appears before "fox" which is a noun)
- Sentence after Rule 1:
    - **The** (DT) **quick** (JJ) **brown** (JJ) **fox** (NN) **jumps** (VBZ) **over** (IN) **the** (DT) **lazy** (NN) **dog** (NN)

Rule 2: Change "NN" (Noun) to "JJ" (Adjective) when it appears before another adjective or noun and the previous word is a determiner (DT).
- Apply this rule to **lazy**:
    - **lazy**: NN → JJ (because it appears before "dog" which is a noun and is preceded by the determiner "the")
- Sentence after Rule 2:
    - **The** (DT) **quick** (JJ) **brown** (JJ) **fox** (NN) **jumps** (VBZ) **over** (IN) **the** (DT) **lazy** (JJ) **dog** (NN)

Step 3: Final Output
After applying these transformation rules, the POS tagging for the sentence is corrected:
- **The** (DT) **quick** (JJ) **brown** (JJ) **fox** (NN) **jumps** (VBZ) **over** (IN) **the** (DT) **lazy** (JJ) **dog** (NN)

Advantages of Transformation-based Learning (TBL)

- We learn small set of simple rules and these rules are enough for tagging.
- Development as well as debugging is very easy in TBL because the learned rules are easy to understand.
- Complexity in tagging is reduced because in TBL there is interlacing of machine learned and human-generated rules.
- Transformation-based tagger is **much faster than** Markov-model tagger.

Disadvantages of Transformation-based Learning (TBL)

- Transformation-based learning (TBL) does not provide tag probabilities.
- In TBL, the training **time is very long especially on large corpora.**

Applications of POS tagging

- There are several real-life applications of part-of-speech (POS) tagging in natural language processing (NLP):

1. **Information extraction**
2. **Named entity recognition**
3. **Text classification**
4. **Machine translation**
5. **Natural language generation**

Challenges in POS tagging

- Some common challenges in part-of-speech (POS) tagging include:
- **Ambiguity**
- **Out-of-vocabulary (OOV) words**
- **Complex grammatical structures**
- **Lack of annotated training data**
- **Inconsistencies in annotated data**

Chunking in NLP

- Chunking is the task of grouping words into meaningful phrases (chunks)
- Chunks: nouns, verbs, adjectives, and adverbs, into larger units known as chunks
- **Chunking stops at phrase level**
- common type of chunking: **noun phrase (NP) chunking**: such as "the cat," "a book," or "my friend."
- Another type of chunking: **verb phrase (VP) chunking**, such as "ate breakfast," "is running," or "will sing."

Example

- Chunking involves first identifying the parts of speech in a sentence using part-of-speech (POS) tagging.
- Once the parts of speech have been identified, the chunks are created by grouping together certain types of words based on their POS tags

Sentence:

"The quick brown fox jumps over the lazy dog."

- POS tags:
    - The/DT quick/JJ brown/JJ fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN
- Chunking (phrases):
    - [NP The quick brown fox]
    - [VP jumps]
    - [PP over](preposition)
    - [NP the lazy dog]

So the sentence is segmented into **chunks** like NP, VP, PP.

Example

- Sentence: The cat sat on the mat.
- POS tags: The/DT cat/NN sat/VBD on/IN the/DT mat/NN.
- Chunks:
- **Noun Phrase: The cat**
- **Verb Phrase: sat**
- **Prepositional Phrase: on the mat**

Applications

Named Entity Recognition (NER)

Text Summarization

Chinking

- **Chinking** is the **opposite of chunking**.
- Instead of **including** a sequence inside a chunk, you **exclude** some tokens (cut them out) from an existing chunk.
- In other words:
  - **Chunking rule** = define what to include.
  - **Chinking rule** = define what to remove from chunks.
- use: to exclude specific types of words. **ie. prepositions, conjunctions, or determiners**, from the chunks
- Chinking is typically used in combination with chunking

Example

Consider the sentence: "The quick brown fox jumps over the lazy dog."

1. **Chunking**:
   1. You might create a rule to chunk noun phrases (NP): [("The quick brown fox", "NP"), ("the lazy dog", "NP")].
2. **Chinking**:
   1. If you want to remove any determiner (DT) from the noun phrases, you can apply a chinking rule that excludes determiners from the chunks.
   2. After chinking, the noun phrases might become: [("quick brown fox", "NP"), ("lazy dog", "NP")].

Use Cases:

- **Refining Chunk Boundaries**
- **Complex Phrase Extraction**

Chinking using RegexpParser

- import nltk
- from nltk.chunk import RegexpParser
- from nltk import pos_tag, word_tokenize

- sentence = "The quick brown fox jumps over the lazy dog."
- tokens = word_tokenize(sentence)
- tagged = pos_tag(tokens)

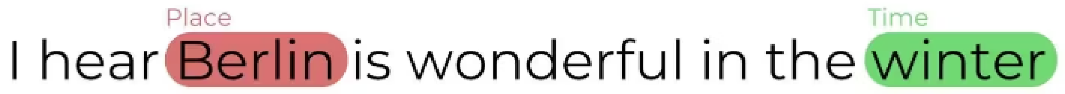
- grammar = r"""
-   NP: {<DT>?<JJ>*<NN>}   # Chunk determiners, adjectives, and nouns as NP
-   }<DT>{   # Chink out determiners
- """
- cp = RegexpParser(grammar)
- result = cp.parse(tagged)
- result.draw()

Why Chunking and Chinking is important in NLP

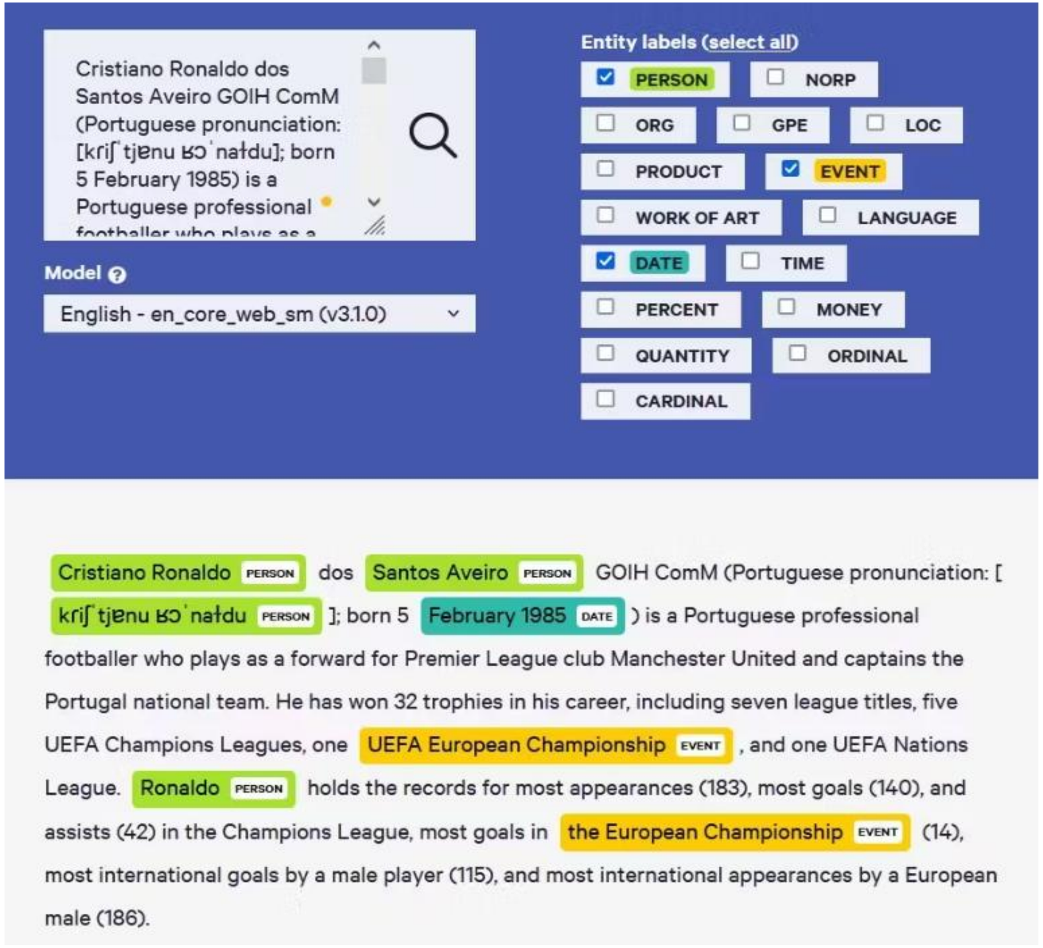
- Allow to **extract meaningful information** from text data.
- To **analyze patterns and relationships** within the text and **extract relevant insights**.
- For example, in a large corpus of news articles, we could use chunking to extract noun phrases from the headlines and analyze the frequency of different types of nouns to identify the most common topics.
- Similarly, in a customer feedback dataset, we could use chunking and chinking to extract relevant phrases and sentiments related to a particular product or service.

Named Entity Recognition

- Named entity recognition (NER) is a form of natural language processing (NLP) that involves **extracting and identifying essential information** from text.
- The information that is extracted and categorized is called **entity**.
- NER essentially extracts and categorizes the detected entity into a **predetermined category**.
- The category can be generic like such as the **names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. or a custom category depending on the use case such as Healthcare Terms, Programming Language, etc.**
- For example, an NER model detects “football” as an entity in a paragraph and classifies it into the category of sports.



Named Entity Visualizer



Cristiano Ronaldo dos Santos Aveiro GOIH ComM (Portuguese pronunciation: [kriʃˈtjɐnu ʁɔˈnaɫdu]; born 5 February 1985) is a Portuguese professional footballer who plays as a

Model

English - en_core_web_sm (v3.1.0)

Entity labels (select all)

[✓ PERSON] NORP ORG GPE LOC PRODUCT [✓ EVENT] WORK OF ART LANGUAGE [✓ DATE] TIME PERCENT MONEY QUANTITY ORDINAL CARDINAL

Cristiano Ronaldo PERSON dos Santos Aveiro PERSON GOIH ComM (Portuguese pronunciation: [kriʃˈtjɐnu ʁɔˈnaɫdu PERSON]; born 5 February 1985 DATE) is a Portuguese professional footballer who plays as a forward for Premier League club Manchester United and captains the Portugal national team. He has won 32 trophies in his career, including seven league titles, five UEFA Champions Leagues, one UEFA European Championship EVENT , and one UEFA Nations League. Ronaldo PERSON holds the records for most appearances (183), most goals (140), and assists (42) in the Champions League, most goals in the European Championship EVENT (14), most international goals by a male player (115), and most international appearances by a European male (186).

EXAMPLES for Not Named Entity and a Named entity

- Hotel & Taj Hotel
- Flower & Rose Flower
- Beach & Kovalam Beach
- Airport & Indira Gandhi International airport
- The School & Good Shepherd School
- Prime Minister & Mr. Manmohan Singh

Example

Barack Obama was born in Hawaii in 1961. He later became the President of the United States.

NER Output:

- **Barack Obama → PERSON**
- **Hawaii → LOCATION**
- **1961 → DATE**
- **President → TITLE** (sometimes)
- **United States → GPE (country)**

NER works **beyond POS tagging and chunking** → it tries to find **real-world named entities**

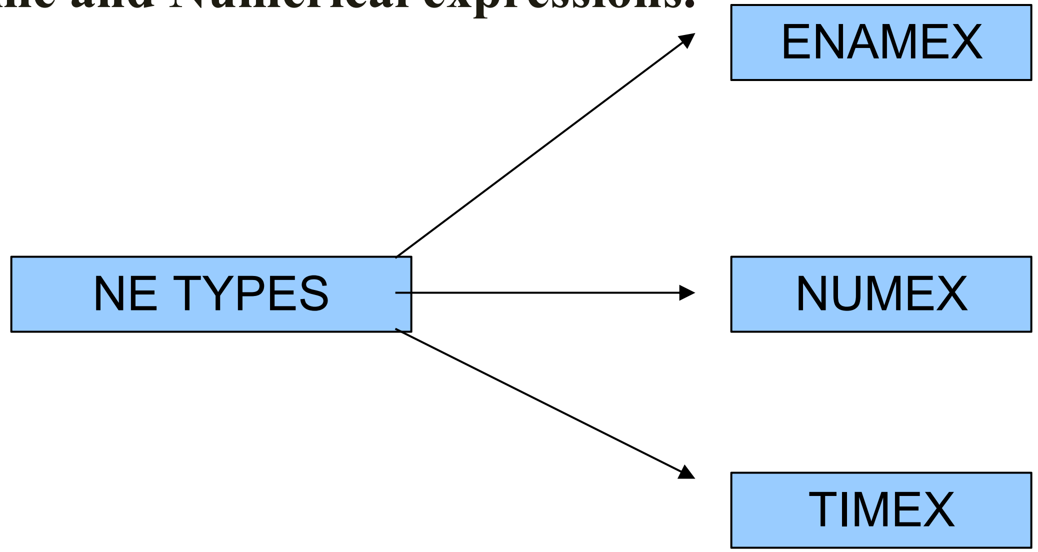
How is named entity recognition used?

- **Customer support**
- **Healthcare**
- **Search engines**

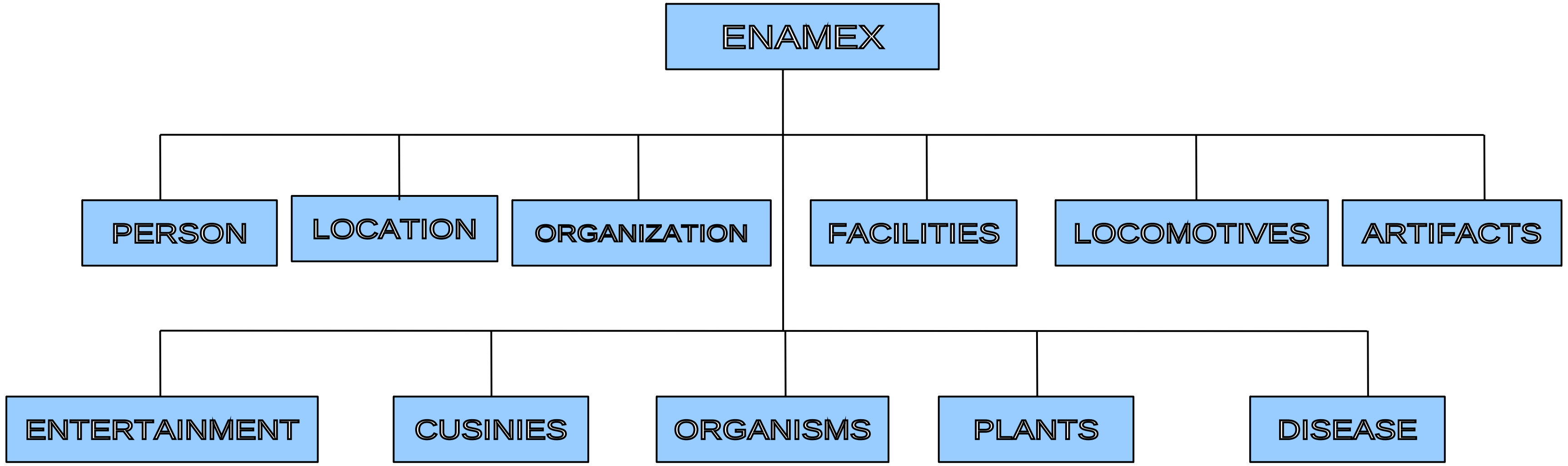
NE Types

The Named entity hierarchy is divided into three major classes Entity

Name, Time and Numerical expressions.



Entity Types

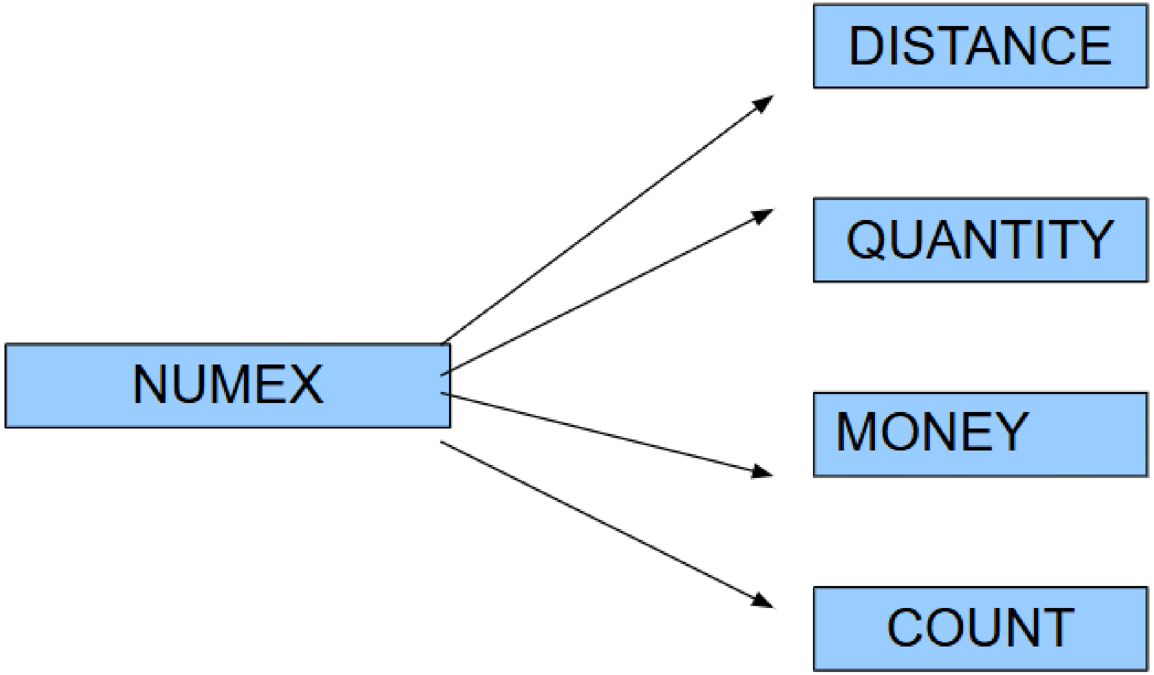


Reference Slides are provided for examples of each category

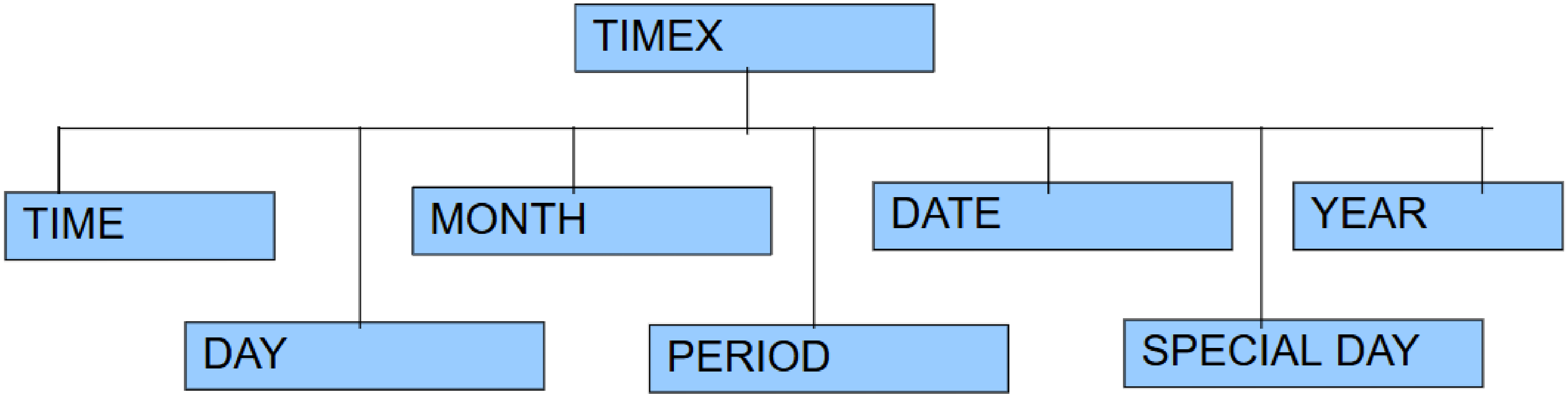
Entity Name Types

- **Persons** are entities limited to humans. A person may be a single individual or a group. Individual refer to names of each individual person. Group refers to set of individual
- **Location** entities are limited to geographical entities such as geographical areas like names of countries, cities, continents and landmasses, bodies of water, and geological formations.
- **Organization** entities are limited to corporations, agencies, and other groups of people defined by an established organizational structure
  - Example: [Sita]$_{\text{PERSON}}$ is working at [HCL]$_{\text{ORGANIZATION}}$, which is in [Seawoods]$_{\text{LOCATION}}$

Numerical Expressions



Time Expressions



Some problems in identifying NE

- **Variation of NEs.**
  - Manmohan Singh, Manmohan, Dr. Manmohan Singh
- **Ambiguity of NE types:**
  - 1945 (date vs. time)
  - Washington (location vs. person)
  - May (person vs. month)
  - Tata (person vs. organization)

Dictionary Based NER

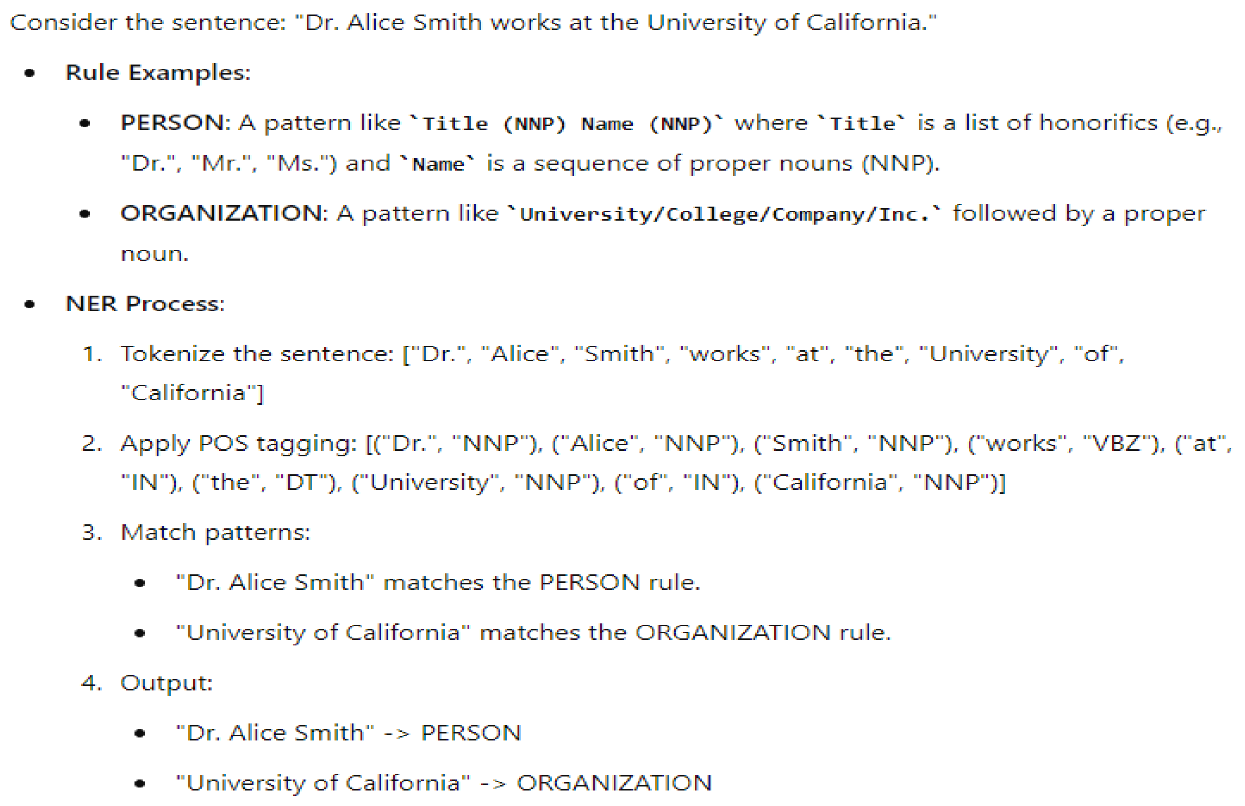
- Dictionary-based Named Entity Recognition (NER) is a method used to identify and classify named entities (like people, organizations, locations, etc.) in text by matching them against a predefined list or dictionary of entities.
- A dictionary is created, containing lists of named entities categorized by type (e.g., people, places, organizations).
- If a token or sequence of tokens matches an entry in the dictionary, it is tagged as a named entity of the corresponding type.

Example

Given the sentence: "John Doe works at OpenAI in San Francisco."

- **Dictionary Entries:**
    - PERSON: ["John Doe", "Alice Smith", ...]
    - ORGANIZATION: ["OpenAI", "Google", ...]
    - LOCATION: ["San Francisco", "New York", ...]
- **NER Process:**
    1. Tokenize the sentence: ["John Doe", "works", "at", "OpenAI", "in", "San Francisco"]
    2. Match tokens with the dictionary:
        - "John Doe" matches PERSON
        - "OpenAI" matches ORGANIZATION
        - "San Francisco" matches LOCATION
    3. Output:
        - "John Doe" -> PERSON
        - "OpenAI" -> ORGANIZATION
        - "San Francisco" -> LOCATION

Rule-based Named Entity Recognition (NER) is a method for identifying and classifying named entities in text based on a set of handcrafted linguistic rules.

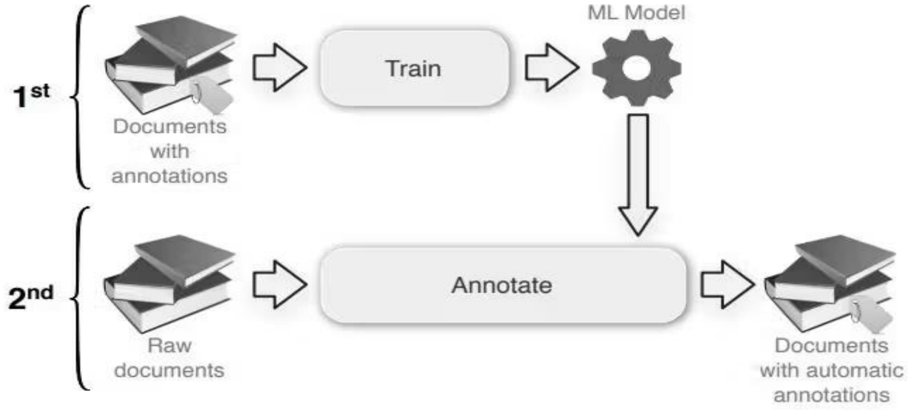


Consider the sentence: "Dr. Alice Smith works at the University of California."

- **Rule Examples:**
  - **PERSON:** A pattern like ``` `Title (NNP) Name (NNP)` ``` where ``` `Title` ``` is a list of honorifics (e.g., "Dr.", "Mr.", "Ms.") and ``` `Name` ``` is a sequence of proper nouns (NNP).
  - **ORGANIZATION:** A pattern like ``` `University/College/Company/Inc.` ``` followed by a proper noun.
- **NER Process:**
  1. Tokenize the sentence: ["Dr.", "Alice", "Smith", "works", "at", "the", "University", "of", "California"]
  2. Apply POS tagging: [("Dr.", "NNP"), ("Alice", "NNP"), ("Smith", "NNP"), ("works", "VBZ"), ("at", "IN"), ("the", "DT"), ("University", "NNP"), ("of", "IN"), ("California", "NNP")]
  3. Match patterns:
     - "Dr. Alice Smith" matches the PERSON rule.
     - "University of California" matches the ORGANIZATION rule.
  4. Output:
     - "Dr. Alice Smith" -> PERSON
     - "University of California" -> ORGANIZATION

Machine Learning Based NER

- utilizes statistical models
- **learn patterns from annotated training** data


1st
Documents with annotations
Train
ML Model
2nd
Raw documents
Annotate
Documents with automatic annotations

How Machine Learning-based NER Works:

a) Data Preparation:

- **Training Data**
- **Features Extraction**: word embeddings, part-of-speech tags, capitalization, context windows, character n-grams, prefixes, suffixes, and more.

b) Model Training:

- **Supervised Learning**: Common algorithms include Conditional Random Fields (CRF), Support Vector Machines (SVM), Hidden Markov Models (HMM), or deep learning models like Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Transformers.

Prediction:
- The trained model is applied to new, unlabeled text.
- The model assigns entity labels to words or phrases based on the patterns it learned during training.

Post-Processing:
- The output may be refined by applying additional rules or heuristics to correct common errors,
- handle overlapping entities, or improve classification accuracy.

Syntax Parsing (Analysis) in NLP

- The syntax analyzer **takes input data (text) and converts it into a structural representation** after verifying it for valid syntax using formal grammar.

Syntactic Parsing

- **Parsing --** converting a flat input sentence into a hierarchical structure that corresponds to the units of meaning in the sentence.
- There are different parsing formalisms and algorithms.
- **CFGs are** in the center of many of the parsing mechanisms. But they are complemented by some additional features that make the formalism more suitable to handle natural languages.

Top-Down Parsing

- **A top-down parser** is a type of syntax parser that **starts from the root** (or highest level) of the parse tree and works its way down **to the leaves** (the actual input words or tokens).
- It uses a grammar (typically in the form of production rules) to try to match a string of words with a valid sentence structure from the highest level (the start symbol) and recursively decomposes it into substructures, until it reaches the terminal symbols (the words of the sentence).

Key Characteristics of Top-Down Parsing:

1. **Start from the top**: It starts with the start symbol of the grammar (e.g., S for sentence).
2. **Recursive decomposition**: The parser recursively tries to break down the sentence into its subcomponents (constituents).
3. **Predictive**: It predicts what kind of structure to expect, trying to match parts of the sentence with grammar rules.

Working of a Top-Down Parser:

1. **Start with the start symbol**:
2. **Choose a production rule**:
3. **Expand recursively**:
4. **Match terminals**:
5. **Backtrack if necessary**:
6. **Termination**:

Top-Down Parsing: Example of CFG

1. $E \rightarrow E + T \mid T$
2. $T \rightarrow T * F \mid F$
3. $F \rightarrow (E) \mid id$

Derive a string id + id * id

Example of Top-Down Parsing:

Consider the following simple sentence:
"The cat sleeps"

We have a grammar with the following rules:

1. **S → NP VP** (A sentence consists of a noun phrase and a verb phrase)
2. **NP → Det N** (A noun phrase consists of a determiner and a noun)
3. **VP → V** (A verb phrase consists of a verb)
4. **Det → "The"** (A determiner is the word "The")
5. **N → "cat"** (A noun is the word "cat")
6. **V → "sleeps"** (A verb is the word "sleeps")

Step-by-Step Top-Down Parsing:

1. **Start with S** (Sentence):
   The parser begins with the start symbol S, which needs to be expanded into a **noun phrase (NP)** and a **verb phrase (VP)**.

   S → NP VP

2. Expand NP (Noun Phrase):

Now, the parser needs to expand NP. According to the grammar, an NP is a determiner (Det) followed by a noun (N).

NP → Det N

3. Expand Det (Determiner):

The parser then looks for a determiner. According to the grammar, Det → "The". It checks the input sentence and matches "The."

Det → "The"

4. Expand N (Noun):

The parser now needs to expand N (Noun). According to the grammar, N → "cat". It checks the next word in the sentence and matches "cat."

N → "cat"

5. Expand VP (Verb Phrase):

Now, the parser turns to the verb phrase VP. According to the grammar, VP → V. It needs to find a verb in the input.

VP → V

6. Expand V (Verb):

The parser now checks for the verb. According to the grammar, V → "sleeps". It matches the next word in the sentence, which is "sleeps."

V → "sleeps"

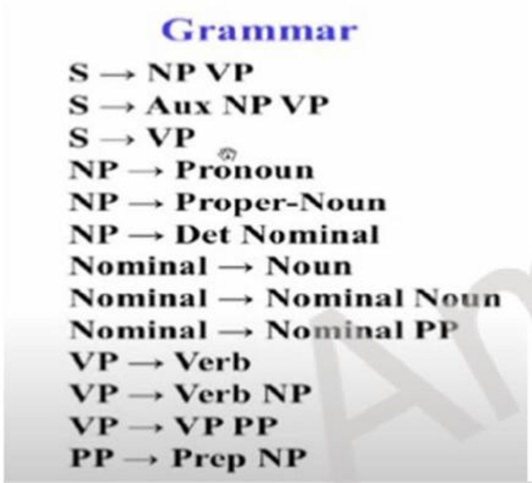
7. Complete the Parse:

At this point, all the components of the sentence have been successfully matched with the grammar. The sentence has been fully parsed.

```
          S
        /   \
      NP     VP
     /  \     |
  Det    N    V
   |     |    |
 "The" "cat" "sleeps"
```

Example

- A restaurant serves Dosas



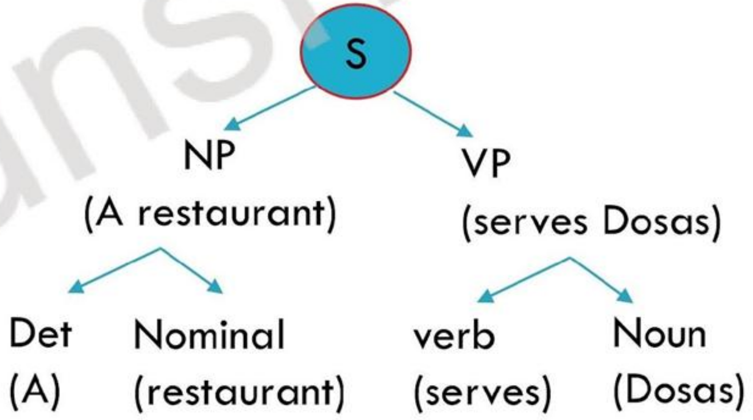
Grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

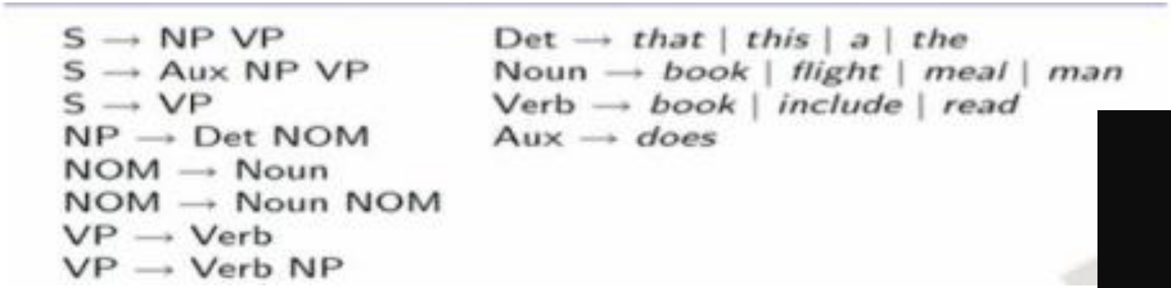
Det: A

Noun: restaurant, dosas

Verb: serves

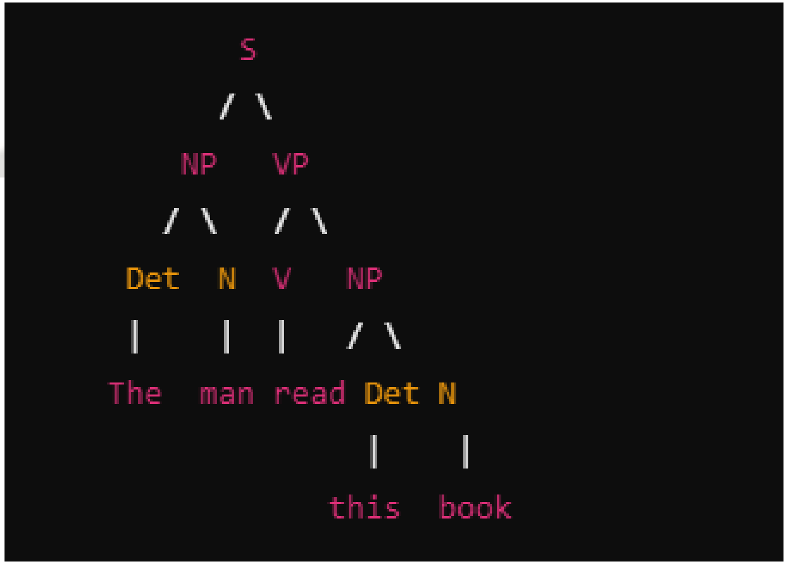


Developing parse tree using Top down approach



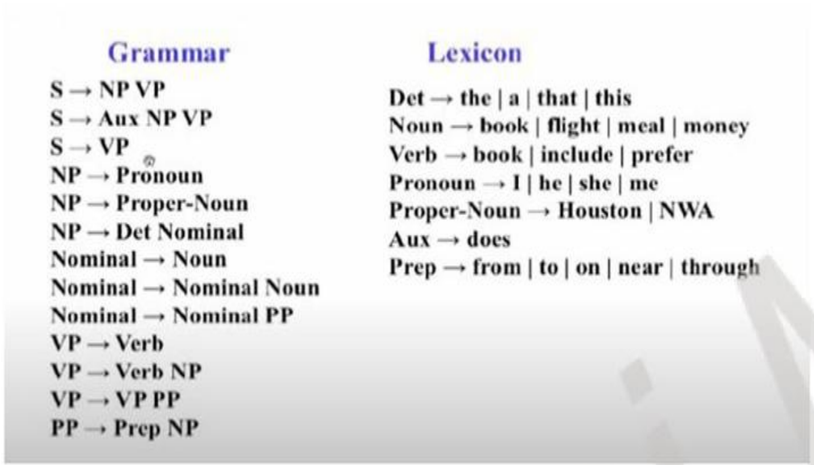
S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

The man read this book



```
           S
          / \
       NP    VP
      / \   / \
   Det   N  V   NP
    |    |  |   / \
   The  man read Det N
                  |   |
                 this  book
```

Developing parse tree using Top down approach



Grammar

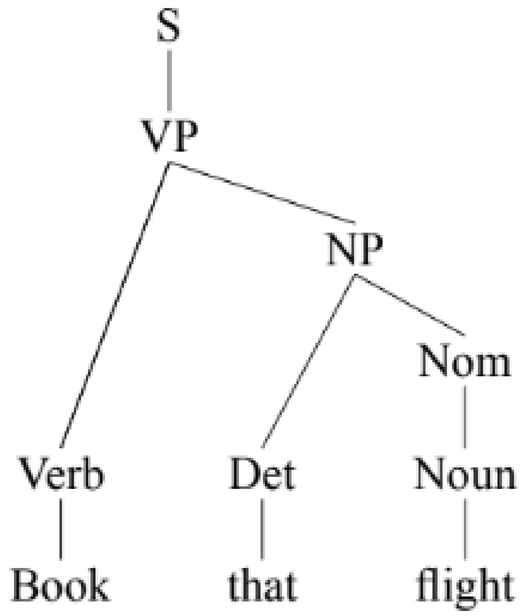
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

Lexicon

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

- Book that flight

- Consider the sentence : Book that flight



Bottom-Up Parsing

- Works in the opposite direction compared to a top-down parser.
- It begins with the input tokens (the words of the sentence) and attempts to build up the parse tree by combining words into larger and larger constituents until it reaches the start symbol.
- The parser attempts to **reduce** the input tokens into higher-level non-terminals (such as noun phrases, verb phrases, etc.), starting from the individual words and gradually constructing the full parse tree.

Working of a Bottom-Up Parser:

1. Start with the input symbols
2. Identify possible reductions
3. Apply reduction
4. Continue reducing
5. Termination

Example of Bottom-Up Parsing:

Consider the same grammar and sentence we used for top-down parsing:

Grammar:

1. **S** → **NP VP** (A sentence consists of a noun phrase and a verb phrase)
2. **NP** → **Det N** (A noun phrase consists of a determiner and a noun)
3. **VP** → **V** (A verb phrase consists of a verb)
4. **Det** → **"The"** (A determiner is the word "The")
5. **N** → **"cat"** (A noun is the word "cat")
6. **V** → **"sleeps"** (A verb is the word "sleeps")

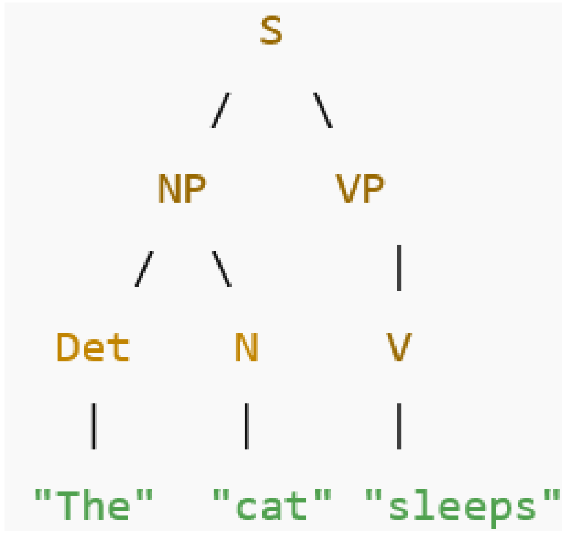
Sentence: "The cat sleeps"

Step-by-Step Bottom-Up Parsing:

1. **Start with the input tokens**:
   The input tokens are the words of the sentences

   "The", "cat", "sleeps"

   These are the terminal symbols. At this point, the parser has no constituents or higher-level symbols.



```
          S
         / \
       NP   VP
      / \    |
   Det   N   V
    |    |   |
 "The" "cat" "sleeps"
```

2. Identify possible reductions:
The parser looks at the input and checks if any sequences of adjacent words can be reduced to a non-terminal based on the grammar.

- The first sequence, "The", matches the rule **Det** → **"The"**. The parser reduces "The" to Det. Now, the input becomes:
- **Det, "cat", "sleeps"**
- The next sequence, "cat", matches the rule N → "cat". The parser reduces "cat" to N. Now, the input becomes:
- **Det, N, "sleeps"**

3. Further reductions:

Now, the parser looks at "Det N" (which is "The cat"). According to the rule **NP** → **Det N**, it can reduce "The cat" to NP. The input becomes: NP, "sleeps"

4. Reduce VP:

Next, the parser looks at "sleeps" and recognizes it as a verb, matching the rule V → "sleeps". The parser reduces "sleeps" to V. Now, the input becomes: NP, V

5. Complete the sentence:

Finally, the parser looks at the sequence "NP V" (which is "The cat sleeps") and recognizes that it matches the rule S → NP VP. The parser reduces "NP V" to S, completing the parse.

The final input is: S

At this point, the parser has successfully reduced the entire input to the

start symbol S, which indicates that the sentence is valid according to the grammar.

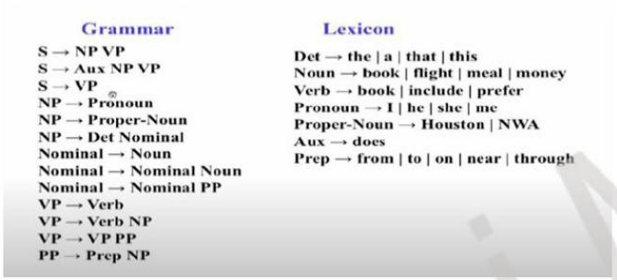
Advantages of Bottom-Up Parsing:

1. **Efficient for some grammars**: Bottom-up parsers are often more efficient than top-down parsers, especially when dealing with grammars **that are not left-recursive**.
2. **Handles ambiguity better**: Bottom-up parsing can handle certain types of ambiguities (like multiple possible productions) more gracefully, because it starts with the actual input and builds upwards, avoiding unnecessary backtracking.

Disadvantages of Bottom-Up Parsing:

1. **Complexity in implementation**: Bottom-up parsers, especially those that handle large grammars, are typically more complex to implement compared to top-down parsers.
2. **Memory-intensive**: Bottom-up parsers may require more memory to store intermediate results and constituent structures as they work through the input tokens.
3. **Inefficient for certain grammars**: In cases where the grammar requires multiple reductions before reaching the start symbol, the parser may explore many potential paths, which can be inefficient.

Developing parse tree using Bottom Up approach



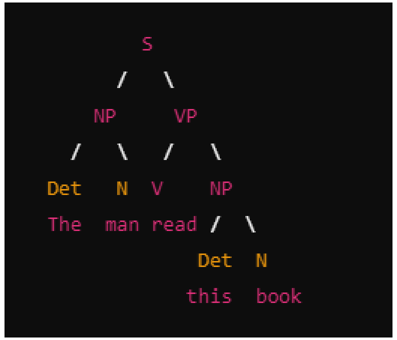
Grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

Lexicon

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

- The man read this book



```
          S
        /   \
      NP     VP
    /   \   /   \
  Det    N  V     NP
  The  man read /  \
                 Det  N
                this  book
```

Top-Down VS. Bottom-Up

Top-down

- Only searches for trees that can be answers
- But suggests trees that are not consistent with the words
- Guarantees that tree starts with S as root
- Does not guarantee that tree will match input words

Bottom-up

- Only forms trees consistent with the words
- Suggest trees that make no sense globally
- Guarantees that tree matches input words
- Does not guarantee that parse tree will lead to S as a root

The End