

Natural Language Processing (TYAIML)

Unit-4

Syntax Analysis:

Syntax Analysis, also known as Parsing, is the process of analysing a string of words (a sentence) to determine its grammatical structure according to a formal grammar.

The core idea is to uncover the hierarchical structure of the sentence—how words group together into phrases, and how those phrases relate to one another to form a complete, meaningful sentence.

Analogy: If words are the bricks, syntax is the architectural blueprint that shows how to arrange them to build a stable sentence (the house).

It checks whether a sequence of words is arranged in a way that follows the rules of grammar.

It helps machines understand “how words relate to each other” in a sentence.

Sentence: “*The boy plays football.*”

- Subject → *The boy*
- Verb → *plays*
- Object → *football*

Why is it Important?

Syntax is crucial for understanding the meaning of a sentence. The order of words and their grammatical relationships dramatically change meaning.

- **Example:** "The dog chased the cat" vs. "The cat chased the dog."
 - The words are identical, but the syntactic structure (who is the subject and who is the object) completely reverses the meaning.

Goals of Syntax Analysis

1. To determine if a sentence is **grammatically correct**.
2. To identify the **syntactic structure** (subject, object, predicate, etc.).
3. To produce a **parse tree** or **dependency tree** showing relationships between words.

Approaches in Syntax Analysis

There are two main approaches:

1. **Constituency-based Parsing**
 - Breaks the sentence into sub-phrases (*constituents*).
 - Represented by a **Parse Tree** (phrase structure tree).

- Example:


```
(S
        (NP The boy)
        (VP plays (NP football)))
```

2. Dependency-based Parsing

- Focuses on the **relations between words**.
- Represented by a **Dependency Tree**.
- Example:
 - *plays* → root
 - *boy* → subject of *plays*
 - *football* → object of *plays*

Techniques Used

- **Rule-Based Parsing** (using CFG – Context-Free Grammar).
- **Statistical Parsing** (probabilistic models, e.g., PCFG).
- **Neural Parsing** (modern NLP, using deep learning and transformers).

Applications of Syntax Analysis

- Grammar checking (e.g., Grammarly).
- Machine Translation (Google Translate).
- Question Answering systems.
- Chatbots & Virtual Assistants.
- Information Extraction.

Key Concepts and Terminology

1. **Context-Free Grammar (CFG)**: The most common formal system used to model syntax. A CFG consists of:
 - Terminals: The actual words (lexical items) of the language (e.g., the, cat, sat).
 - Non-Terminals: Symbols that represent syntactic categories (e.g., S for sentence, NP for noun phrase, VP for verb phrase).
 - Production Rules: Rules that define how non-terminals can be expanded into other non-terminals or terminals.
 - $S \rightarrow NP\ VP$
 - $NP \rightarrow Det\ N$
 - $VP \rightarrow V\ NP$

- Start Symbol: Usually S (Sentence).
2. Parse Tree: The visual output of syntax analysis. It's a tree structure that represents the hierarchical syntactic structure of a sentence based on the grammar.
- Leaves: The words of the sentence (terminals).
 - Root: The start symbol (e.g., S).
 - Internal Nodes: Non-terminal symbols.
3. Constituents: Groups of words that function as a single unit within the hierarchical structure (e.g., Noun Phrase NP, Verb Phrase VP, Prepositional Phrase PP).

A Simple Example

Sentence: "The cat sat on the mat." Let's define a very simple CFG: text

$S \rightarrow NP\ VP$, $NP \rightarrow Det\ N$, $VP \rightarrow V\ PP$, $VP \rightarrow V$, $PP \rightarrow P\ NP$.

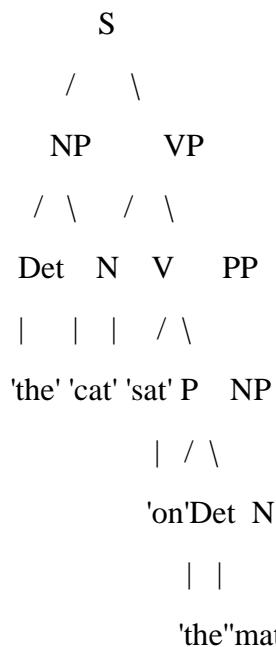
$Det \rightarrow 'the'$,

$N \rightarrow 'cat' \mid 'mat'$,

$V \rightarrow 'sat'$,

$P \rightarrow 'on'$

The resulting **parse tree** would look like this:



This tree shows us that:

- The subject is the noun phrase (NP) "The cat".
- The predicate is the verb phrase (VP) "sat on the mat", which itself contains a verb (V) "sat" and a prepositional phrase (PP) "on the mat".

Types of Parsing

There are two primary strategies for parsing:

1. Top-Down Parsing

- How it works: Starts from the root node (S) and applies grammar rules forward until it generates the surface sentence (the leaves). It's a goal-driven approach: "What rules do I apply to get from S to the words I see?"
- Analogy: You start with the goal of building a house (S), then decide you need walls and a roof (NP VP), then you break down the walls into bricks and mortar (Det N), and so on, until you match the available materials (the words).

2. Bottom-Up Parsing

- How it works: Starts from the words (leaves) and applies grammar rules backward to reduce them to higher-level non-terminals until it reaches the root node (S). It's a data-driven approach: "Given these words, what rules can I use to group them into larger and larger chunks until I form a sentence?"
- Analogy: You start with a pile of bricks, wood, and pipes (the words). You group the bricks into a wall (NP), the wood into a roof frame (VP), and then you combine the wall and roof into a house (S).

Tagset for English (Penn Treebank):

- A very popular part-of-speech tagset is from the University of Pennsylvania.
- It is called the Penn Treebank (PTB) and contains 45 part-of-speech tags.
- It is fairly standardized for English.
- Popular NLP tools such as Stanford CoreNLP and spaCy use this tag set.
- The POS tags are small and compact, consisting of 2–4 capital characters.

Syntactic Ambiguity:

The most significant challenge in syntax analysis is that natural language is full of ambiguity. A single sentence can often have multiple valid syntactic interpretations, leading to different meanings.

Classic Example: "I saw the man with the telescope."

This sentence has (at least) two parse trees:

1. The PP "with the telescope" modifies "the man":

I saw [the man] [with the telescope].

- Meaning: The man was holding the telescope when I saw him.

2. The PP "with the telescope" modifies "saw":

I saw [the man] [with the telescope].

- Meaning: I used the telescope to see the man.

A parser will often generate multiple parse trees for such sentences. Disambiguating between them requires more than just syntax—it requires semantic analysis (does "saw with" make sense?) and real-world context (is a telescope something you're more likely to use or see someone holding?).

Modern Approaches: From Rules to Data

Historically, parsers relied on hand-crafted CFG rules (like in the example above). This is difficult to scale and maintain. Modern NLP almost exclusively uses statistical or neural parsers.

1. Statistical Parsers:

- Use a treebank (a large, hand-annotated corpus of parse trees, like the Penn Treebank) as training data.
- The grammar rules are learned automatically from this data.
- Each grammar rule is associated with a probability (e.g., how likely is it that a VP expands to V NP vs. just V?).
- The parser's job is to find the most probable parse tree for a given sentence. The CYK algorithm is a classic algorithm for this. The Stanford Parser is a famous example.

2. Neural Parsers:

- Use deep learning models (e.g., Transformers, LSTMs) to predict the structure of a sentence directly from its words.
- They learn dense vector representations (embeddings) for words and their contexts, which help resolve ambiguities much more effectively.
- They often treat parsing as a sequence labeling task or use attention mechanisms to decide which words are related (who is the head of which word).
- Models like the BERT-based SyntaxGym or Benepar (BERT-based Neural Parsing) represent the state-of-the-art, achieving human-level performance on standard benchmarks.

POS Tagging:

- Parts of speech are also known as word classes or lexical categories.
- Part-of-speech (POS) tagging is the process of labeling words in a text with their corresponding parts of speech in natural language processing (NLP).
- It helps algorithms understand the grammatical structure and meaning of a text.

- Each word in a text is labeled with its corresponding part of speech which include nouns, verbs, adjectives, and other grammatical categories.
- The collection of tags used for a particular task is known as a tagset.
- Provides perceptions into the syntactic structure of the text, aiding in understanding word relationships, disambiguating word meanings, and facilitating various linguistic and computational analyses of textual data.
- Example:
 - Sentence: *I eat apples*
 - Tags: **I/PRONOUN, eat/VERB, apples/NOUN**
 - The challenge: many words are **ambiguous** (e.g., *book* can be a **noun** or a **verb**).
 - We use **probabilities** to choose the **most likely tag sequence** → that's where **HMM** comes in.

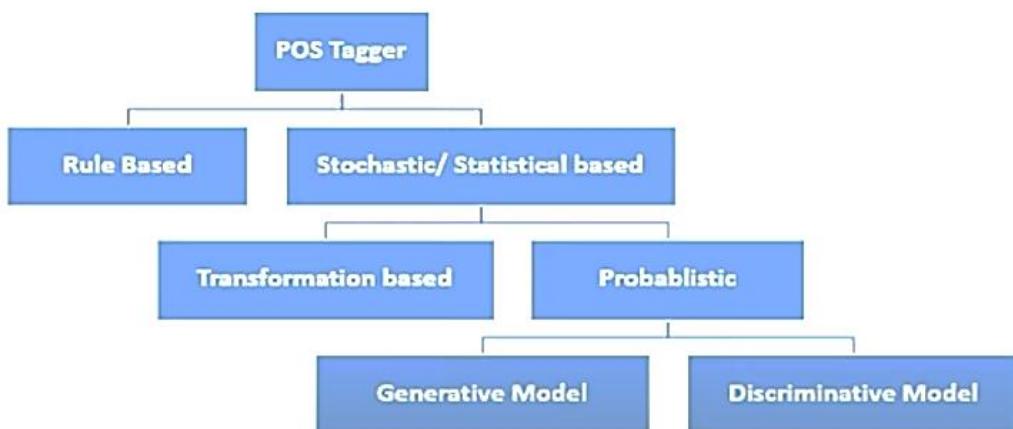
Advantages of Part-Of-Speech (POS) Tagging:

- 1. Improves Natural Language Understanding**
 - Helps machines understand the grammatical structure of sentences.
 - Provides context for interpreting word meanings.
- 2. Word Sense Disambiguation**
 - Resolves ambiguity of words that have multiple meanings.
 - Example: *book* (noun: "a novel") vs. *book* (verb: "to reserve").
- 3. Foundation for NLP Applications**
 - Essential preprocessing step for tasks like:
- 4. Supports Parsing and Grammar Checking**
 - Makes syntactic parsing more accurate.
 - Helps in grammar correction tools.
- 5. Enhances Text-to-Speech Systems**
 - Correct stress and intonation rely on POS information.
 - Example: *object* (noun vs. verb) affects pronunciation.
- 6. Improves Search and Information Extraction**
 - Enables smarter search engines by identifying key terms.
 - Helps extract subject–verb–object relations from text.
- 7. Supports Machine Learning Models**
 - Provides features for models in text classification, sentiment analysis, and summarization.

Limitations of Part-Of-Speech (POS) Tagging:

- 1. Ambiguity in Words**
 - Many words belong to multiple POS categories depending on context.
 - Example: "*flies*" (noun: insects, verb: action).
- 2. Dependency on Context**
 - POS tags often depend on surrounding words, which makes tagging harder in complex sentences.
- 3. Domain Sensitivity**
 - Taggers trained on one domain (e.g., news articles) may not perform well on another (e.g., medical text, social media).
- 4. Limited Vocabulary Handling**
 - Struggles with unknown words (Out-of-Vocabulary problem).
 - Especially problematic for new terms, slang, or technical jargon.
- 5. Errors in Ambiguous Constructions**
 - Sentences with complex or unusual structures can mislead POS taggers.
 - Example: "Visiting relatives can be boring" (Is "visiting" a verb or adjective?).
- 6. Language-Specific Challenges**
 - Morphologically rich languages are harder to tag compared to English.
- 7. Propagation of Errors**
 - Errors in POS tagging negatively affect downstream NLP tasks.
- 8. Resource Intensive**
 - High-accuracy taggers (like neural models) require large annotated corpora and computational power.

POS Tagger Classification:



Rule Based Tagging

- It involves assigning words their respective parts of speech using predetermined rules, contrasting with machine learning-based POS tagging that requires training on annotated text corpora.
- In a rule-based system, POS tags are assigned based on specific word characteristics and contextual cues.
- Like, a rule-based POS tagger could designate the “noun” tag to words ending in “-tion” or “-ment,” recognizing common noun-forming suffixes
- It assigns tags directly based on predefined rules:
- Rule: Assign the POS tag “noun” to words ending in “-tion” or “-ment.”
- Text: “The presentation highlighted the key achievements of the project’s development.”

Rule based Tags:

- “The” – Determiner (DET)
- “presentation” – Noun (N)
- “highlighted” – Verb (V)
- “the” – Determiner (DET)
- “key” – Adjective (ADJ)
- “achievements” – Noun (N)
- “of” – Preposition (PREP)
- “the” – Determiner (DET)
- “project’s” – Noun (N)
- “development” – Noun (N)

Advantages of Rule-Based Tagging in NLP

1. **High Accuracy in Specific Domains**
 - Works well in **restricted or domain-specific text** (like medical, legal, or scientific corpora) where vocabulary and grammar are predictable.
2. **Interpretability & Transparency**
 - Rules are **human-readable** and easy to understand, unlike black-box ML models.
 - Linguists can trace why a tag was assigned.
3. **No Large Training Data Required**
 - Unlike statistical or neural taggers, rule-based systems **don’t need massive annotated corpora**.

- Useful in **low-resource languages**.

4. **Deterministic Output**

- Same input → always same output.
- Makes it **reliable and consistent**, especially in controlled settings.

5. **Customizability**

- Rules can be tailored for **specific grammar, style, or application needs**.
- For example, handling special tokens in social media text.

6. **Good for Rare or Complex Structures**

- Can capture **linguistic exceptions and irregularities** that statistical models may miss.
- Example: differentiating "lead" (verb) vs. "lead" (metal) using context rules.

7. **Low Computational Cost**

- Simple rule-checking is often faster than training and running heavy ML models.

Limitations of Rule-Based Tagging in NLP

1. **Ambiguity Handling is Weak**

- Words with multiple possible tags (e.g., "book" → noun or verb) are hard to resolve using fixed rules.
- Context is often ignored or oversimplified.

2. **Time-Consuming to Build & Maintain**

- Requires **linguistic experts** to write hundreds/thousands of handcrafted rules.
- Updating rules for new domains or languages is difficult.

3. **Poor Scalability**

- Works for **small datasets** or specific domains, but fails for large, diverse corpora.
- Rule sets grow complex and inconsistent as vocabulary expands.

4. **Language Dependency**

- Each language needs a separate rule set, making it hard to generalize.
- Not suitable for **multilingual applications**.

5. **Lack of Robustness**

- Fails on **noisy text** (social media, slang, spelling errors, informal writing).
- Rules often break when encountering unseen words or structures.

6. **Limited Adaptability**

- Cannot **learn automatically** from data.

- Needs manual modification for new text styles, domains, or contexts.

7. Lower Accuracy Compared to Statistical/ML Approaches

- Machine Learning or Deep Learning taggers (e.g., HMMs, CRFs, BiLSTMs, Transformers) usually outperform rule-based systems in accuracy and generalization.

Statistical / Stochastic Tagging

- Utilize probabilistic models
- Uses trained algorithms to predict tags probabilistically
- CRFs (conditional random fields) and Hidden Markov Models (HMMs) are popular models for statistical point-of-sale classification.
- The algorithm estimates the chance of observing a specific tag given the current word and its context by learning from labeled samples during training
- Two approaches are used –
 - a. Transformation Based Tagger
 - b. Probabilistic Based Tagger

a) Transformation Based Tagger

- This tagger is based on concept of Transformation-Based Learning (TBL) approach.
- TBL uses supervised learning.
- There is assumption of pre-tagged training corpus.
- It combines idea of the rule-based and stochastic taggers.
- Like the rule based taggers, TBL is based on rules that specify what tags should be assigned to what words, but like the stochastic taggers, TBL is a machine learning technique, in which rules are automatically induced from the data.
- Label the training set with most frequent tags
e.g. **The can was rusted.**

The/DT can/MD was/VBD rusted/VBD

Add transformation rules to reduce training mistakes.

The/DT can/NN was/VBD rusted/VBN

Explanation: ‘can’ can be noun as well as modal verb also.

It uses machine learning as well as grammar rules.

1. Modal verb is never preceded by determiner.
2. Also in corpus (training data) possibility of ‘determiner model verb’ pair is zero.

Therefore *can* is replaced as Noun tag.

Explanation: ‘rusted’ can be VBD as well as VBN also.

1. In grammar rule, VBN is preceded by VBD.
2. Also in corpus, possibility of “VBD-VBD” pair is negligible.

Statement:

Race is **incorrectly** tagged in the following statement:

is/VBZ expected/VBN to/TO race/NN tomorrow/NN

In the second case this race is **correctly** tagged as an NN:

the/DT race/NN for/IN outer/JJ space/NN

For example, in the Brown corpus, race is most likely to be a noun:

$$P(\text{NN}|\text{race}) = .98 \quad P(\text{VB}|\text{race}) = .02$$

Therefore race is labelled as NN because its occurrence (probability) is more.

Brill's tagger learned a rule that applies exactly to this mistagging of race:

Change NN to VB when the previous tag is TO.

This rule would change race/NN to race/VB in exactly the following situation, since it is preceded by to/TO:

/VBN to/TO race/NN expected/VBN to/TO race/VB

b) Probabilistic:

Approach is to "pick the most-likely tag for this word".

Two approaches are there whether you generate the data from the class or the class from the data.

Problem statement: Data available of the form [d, c]

where d is observations and c is hidden classes.

Two types of Model are there

- i) Generative Model
- ii) Discriminative Model

1) Generative Model

- In generative model, we assume that class is there and it generates data.
- For e.g. class is given (subjects), and all words are generated from subject.
- For e.g. Part-of-Speech Tagging: POS tags are there and words are generated from that.
- Here flow is downward (Classes generate data).

Example are Hidden Markov Model, Naive Bayes Classifier



- For e.g. POS tag noun (class) generates data like girl, boy, man, woman, water etc.
- Statement is ‘The girl is intelligent.’ In this case, determiner class generates the, NN class generates girl, VBZ generates is and JJ generates adjective.



Advantages of Hidden Markov Model (HMM) in POS Tagging

1. Statistical Foundation:

HMM provides a strong probabilistic framework for modeling the sequence of words and their corresponding tags.

2. Handles Ambiguity Well:

HMM can effectively handle words with multiple possible tags (e.g., *book* as noun or verb) by choosing the tag sequence with the highest probability.

3. Sequence Context:

Unlike simple rule-based systems, HMM considers the *context* of neighboring words and tags, improving accuracy.

4. Automatic Learning:

Model parameters (transition and emission probabilities) can be automatically learned from large annotated corpora.

5. Efficient Algorithms:

Algorithms like the **Viterbi algorithm** enable efficient decoding to find the most likely tag sequence for a given sentence.

6. Language Independence:

Once trained, HMM can be applied to different languages with minimal modifications, given suitable training data.

7. Scalability:

HMMs can handle large datasets and long sequences efficiently.

Applications of HMM in POS Tagging

1. Part-of-Speech Tagging:

The main application — assigning grammatical categories (like noun, verb, adjective) to words in a sentence.

2. Speech Recognition:

Predicting phoneme or word sequences based on acoustic observations using similar probabilistic transitions.

3. Named Entity Recognition (NER):

Identifying proper names, locations, and organizations using sequential tagging similar to POS tagging.

4. Machine Translation:

Helps in aligning and tagging source language words to their equivalent parts of speech in the target language.

5. Information Extraction:

Identifying patterns like names, dates, or events in text using sequential models.

6. Spelling and Grammar Correction:

Predicting the most probable word or tag sequence helps identify incorrect word usage.

7. Text-to-Speech Systems:

POS tags obtained from HMM models help in selecting proper pronunciation and intonation.

Limitations of Hidden Markov Model (HMM) in POS Tagging

- Strong independence assumptions (each word depends only on its tag)

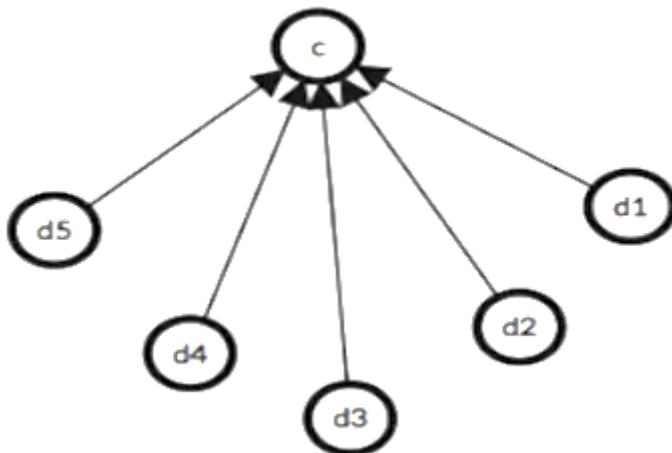
- Needs large tagged corpus for accurate probabilities
- Cannot easily use rich contextual features

2) Discriminative Model:

In the discriminative model, data is there and you assume that hidden state is generated from the data. Here flow is upward. (that data identifies class)

For e.g. keywords of the subjects are given (data) and from the keywords we determine subject. Keywords are data and subject is class. Class is hidden. By seeing nature of data, we can assign probability of the particular class.

Example are Maximum entropy model, Conditional Random field.



For e.g. For every word, various features are given with respect to previous words and future words. From that features, we can decide POS tag for that word. That is flow is upward. For a given word, using features of neighbouring word, we determine class or tag.



Advantages of Maximum Entropy Model in POS Tagging

1. Flexibility in Feature Usage:

- The Maximum Entropy (MaxEnt) model allows the use of diverse and overlapping features (e.g., word prefixes, suffixes, capitalization, previous tags, etc.), making it adaptable to complex linguistic contexts.

2. No Independence Assumption:

- Unlike models such as Naïve Bayes, MaxEnt does not assume independence between features, allowing it to model real-world dependencies in language data more accurately.

3. Probabilistic Framework:

- It provides probability distributions over possible tags, which makes it easier to interpret and combine with other models or post-processing steps.

4. Handles Sparse Data Well:

- Through regularization, MaxEnt can manage sparse or incomplete data without overfitting.

5. Domain Adaptability:

- Since features can be customized easily, the model can adapt well to new domains or languages by redefining relevant features.

Applications of Maximum Entropy Model in POS Tagging:

1. Part-of-Speech (POS) Tagging:

- Assigning the correct grammatical category (noun, verb, adjective, etc.) to each word in a sentence based on its context.

2. Named Entity Recognition (NER):

- Identifying proper names like persons, organizations, or locations, often as an extension of POS tagging.

3. Word Sense Disambiguation:

- Determining the correct meaning of a word with multiple senses depending on the context.

4. Syntactic Parsing:

- Providing probabilistic information about possible parses of a sentence by integrating POS probabilities.

5. Information Extraction:

- Used in extracting structured information (e.g., dates, names, relationships) from unstructured text.

A **Statistical or Stochastic Tagger** is a type of part-of-speech (POS) tagger that uses statistical methods to assign grammatical tags to words in a sentence. Unlike rule-based taggers, which rely on predefined rules, statistical taggers learn from a large corpus of labeled data, allowing them to predict the most likely tag for a word based on context.

Key Concepts:

- **Statistical Models:** These taggers use probability to determine the likelihood of a word belonging to a particular part of speech.
- **Training Data:** The system is trained on a large annotated corpus, meaning a collection of texts where each word is already tagged with its correct part of speech.
- **Contextual Analysis:** The decision to tag a word is influenced by its surrounding words, making it context-dependent. This is important because some words can serve different parts of speech depending on context (e.g., "run" can be a noun or a verb).

Example:

Consider the sentence: "**She enjoys reading books.**"

1. Step 1: Input the Sentence

The sentence is broken down into individual words: ["She", "enjoys", "reading", "books"].

2. Step 2: Calculate Probabilities

The statistical tagger uses a model (such as Hidden Markov Models or Conditional Random Fields) trained on a large corpus to determine the most probable tag for each word based on its context. For example:

- "She" is likely tagged as a **Pronoun (PRP)**.
- "enjoys" is most likely tagged as a **Verb (VBZ)** because it follows a subject (She).
- "reading" is a bit tricky, but based on the surrounding words, it is likely to be tagged as a **Gerund (VBG)** (a verb acting as a noun).
- "books" is tagged as a **Noun (NNS)** because it is plural and follows a verb.

3. Step 3: Output the Tags

After analyzing the probabilities, the tagger assigns the appropriate tags:

- "She" → Pronoun (PRP)
- "enjoys" → Verb (VBZ)
- "reading" → Gerund (VBG)
- "books" → Noun (NNS)

Advantages:

- **Accuracy:** These taggers tend to be more accurate than rule-based taggers because they can learn from a vast amount of data and adapt to various sentence structures.
- **Adaptability:** They can work on different languages and domain-specific texts, as long as there is enough training data.

Disadvantages:

- **Training Data Requirement:** The accuracy of the tagger heavily depends on the quantity and quality of the training data.
- **Handling Ambiguity:** Words that have multiple possible POS tags (like "run") can be challenging, but the context usually helps resolve these ambiguities.

Hidden Markov Model (HMM) in POS Tagging

A **Hidden Markov Model (HMM)** is a **statistical model** that represents a system as a sequence of **hidden states** producing **observable outputs**.

In POS tagging:

- **Hidden states** = POS tags
- **Observed states** = Words in the sentence

We don't see the **tags** directly (they're "hidden"), but we observe the **words**.

Goal in HMM Tagging:

Given a sequence of words, find the **most probable tag sequence**.

HMM Assumptions:

- Each tag depends only on the previous tag.
- Each word depends only on its tag.

Parameters:

- Transition Probability: Probability of a tag following another tag.
- Emission Probability: Probability of a word given a tag.
- Initial Probability: Probability of starting with a tag

Decoding: Viterbi Algorithm

We use the **Viterbi algorithm** to find the **most likely tag sequence** efficiently.

It uses **dynamic programming** to compute the **maximum probability path** through the sequence of tags.

Advantages:

- Simple and effective
- Based on solid probability theory
- Handles ambiguity statistically
- Works well for well-tagged training corpora

Limitations:

- Assumes independence (ignores long-range context)

- Needs large **tagged** data
- Struggles with **out-of-vocabulary** words
- Cannot use arbitrary features (like suffixes, capitalization)

Applications:

- POS tagging
- Named Entity Recognition
- Speech recognition
- Bioinformatics (gene sequence tagging)

Illustration 1:

Apply Hidden Markov model for the below data and find out the appropriate POS tags for the test data:

Training Data:

<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>	

Test Data:

<S> Justin will spot Will <E>

Training Data:

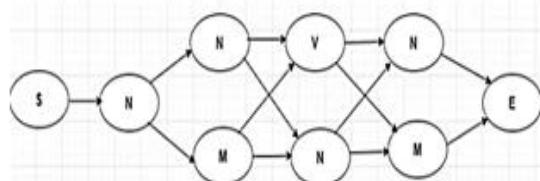
<S>	Martin	Justin	can	watch	Will	<E>
<S>	Spot	will	watch	Martin	<E>	
<S>	Will	Justin	spot	Martin	<E>	
<S>	Martin	will	pat	Spot	<E>	

Test Data:

<S> Justin will spot Will <E>

Justin	will	Spot	Will
N	N	V	N
M	N	N	M

Total ways : $1 \times 2 \times 2 \times 2 = 8$



Step 1: Assign correct POS tag:

<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>	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		

Step 2: Creation of State Transition Probability Matrix:

<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	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

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

C = Class

Step 3: Creation of Emission Probability Matrix:

<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	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

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

Test Data: <S> Justin will spot Will <E>

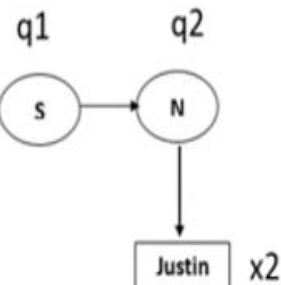
Justin	will	Spot	Will
N	N	V	N
	M	N	M

Justin:

q1 and q2: States (class)

x2: word emitted from q2 state

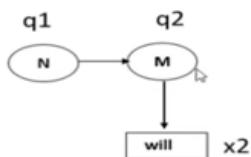
$$P(q2|x2, q1) = P(x2|q2) * P(q2|q1)$$



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

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

Will as Modal Verb



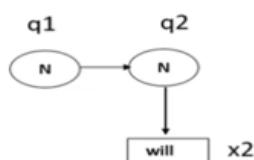
$$P(M|will, N) = P(will|M) * P(M|N)$$

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

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

Justin	will	Spot	Will
N	N	V	N
	M	N	M

Will as Noun



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

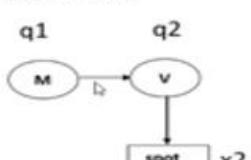
$$P(N|will, N) = \frac{1}{9} \times \frac{1}{9} = \frac{1}{81}$$

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

$$\frac{1}{24} > \frac{1}{486} \quad P(M) > P(N)$$

Result: will as Modal Verb

spot as Verb



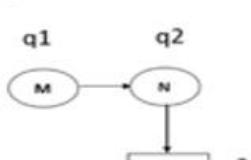
$$P(v|spot, M) = P(spot|v) * P(v|M)$$

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

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

Justin	will	Spot	Will
N	N	V	N
	M	N	M

spot as Noun



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

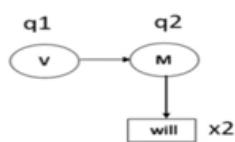
$$P(N|spot, M) = \frac{2}{9} \times \frac{1}{4} = \frac{1}{18}$$

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

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

Result: spot as verb

will as modal Verb

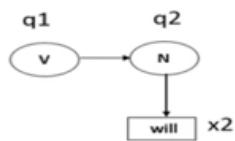


$$P(M|will, V) = P(will|M) * P(M|V)$$

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

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

will as Noun



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

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

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

$$\frac{1}{128} > \frac{1}{432}$$

$$P(V) > P(N)$$

Result: will as noun

Result of POS tagging after applying HMM model

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

Illustration 2:

Apply Hidden Markov model for the below data and find out the appropriate POS tags for the test data:

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

<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 Data

The	park	is	a	book
Det.	Noun	Verb	Det.	Noun
Verb				Verb

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

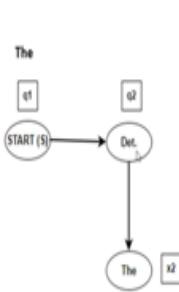
<S>	Book	a	car	</S>			
	Verb	Det.	Noun				
<S>	Park	the	car	</S>			
	Verb	Det.	Noun				
<S>	The	book	is	in	the	car	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	
<S>	The	car	is	in	a	park	</S>
	Det.	Noun	Verb	Prep.	Det.	Noun	

	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

Emission 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

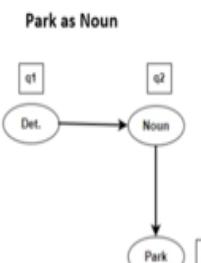
State Transition Probability Matrix



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

$$P(\text{Det}|\text{The}, <\text{S}>) = P(\text{The}|\text{Det}) * P(\text{Det}|<\text{S}>)$$

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



Previous Probability is 1/3

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

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

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

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

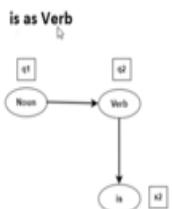
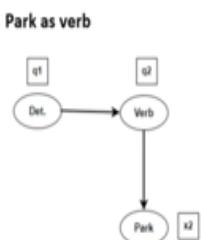
$$P(\text{Verb}|\text{Park}, \text{Det.}) = P(\text{Park}|\text{Verb}) * P(\text{Verb}|\text{Det.})$$

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

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

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

$$\frac{1}{18} > \frac{1}{1200} \quad \text{Result: park as noun}$$

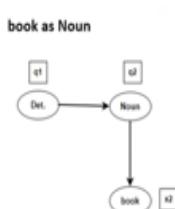


Previous Probability is 1/18

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

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

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

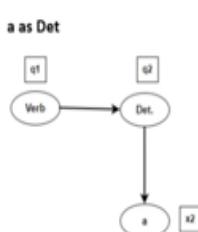


Previous Probability is 1/648

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

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

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

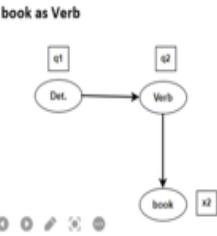


Previous Probability is 1/108

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

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

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



Previous Probability is 1/648

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

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

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

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

$$\frac{1}{3888} > \frac{1}{259200} \quad \text{Result: book as Noun}$$

Result of POS tagging after applying HMM model

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

Dr. Varsha Patil

Illustration 3:

Apply Hidden Markov model for the below data and find out the appropriate POS tags for the test data:

Emission Probability Matrix

	Time	Flies	like	an	arrow
VB	0.1	0.2	0.2	0	0
NN	0.1	0.1	0	0	0.1
IN	0	0	0.25	0	0
DT	0	0	0	0.5	0

State Transition Probability Matrix

	VB	NN	IN	DT	<E>
<S>	0.2	0.8	0	0	0
VB	0	0.3	0.2	0.5	0
NN	0.4	0.5	0.1	0	0
IN	0	0.75	0	0.25	0
DT	0	1	0	0	0

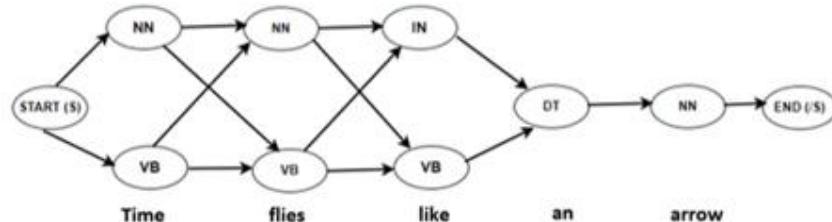
Test data <s> Time flies like an arrow</s>

Time	flies	like	an	arrow
NN	NN	IN	DT	NN
VB	VB	NN		

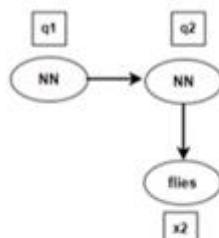
Test Data <s> Time flies like an arraow </s>

Time	flies	like	an	arrow
NN	NN	IN	DT	NN
VB	VB	NN		

$2 \times 2 \times 2 \times 1 \times 1 = 8$ ways

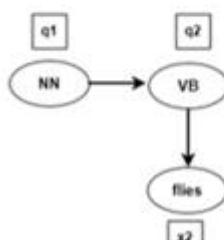


Previous Probability is 0.08



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

$$P(NN|flies, NN) = 0.1 \times 0.5 = 0.05 \\ = 0.05 \times 0.08 = 0.0040$$



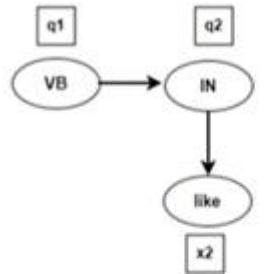
$$P(VB|flies, NN) = P(flies|VB) \times P(VB|NN)$$

$$P(VB|flies, NN) = 0.2 \times 0.4 = 0.08 \\ = 0.08 \times 0.08 = 0.0064$$

$$0.0064 > 0.0040$$

Result: flies is verb

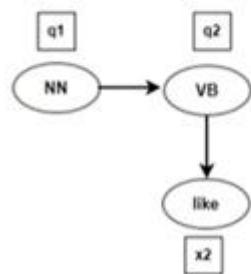
Previous Probability is 0.0064



$$P(\text{IN}|\text{like}, \text{VB}) = P(\text{like}|\text{IN}) \times P(\text{IN}|\text{VB})$$

$$P(\text{IN}|\text{like}, \text{VB}) = 0.25 \times 0.2 = 0.05$$

$$= 0.05 \times 0.0064 = 0.00032$$



$$P(\text{VB}|\text{like}, \text{VB}) = P(\text{like}|\text{VB}) \times P(\text{VB}|\text{VB})$$

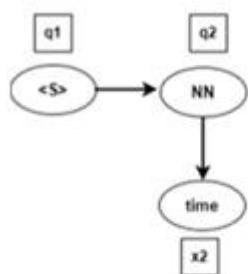
$$P(\text{IN}|\text{like}, \text{VB}) = 0.2 \times 0 = 0$$

$$= 0.2 \times 0.01 = 0.002$$

$$= 0.002 \times 0.0064 = 0.0000128$$

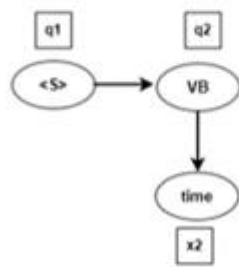
Result: like is preposition

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



$$P(\text{NN}|\text{time}, <\text{S}>) = P(\text{time}|\text{NN}) \times P(\text{NN}|<\text{S}>)$$

$$P(\text{NN}|\text{time}, <\text{S}>) = 0.1 \times 0.8 = 0.08$$



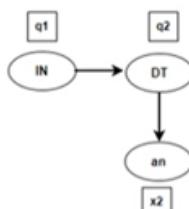
$$P(\text{VB}|\text{time}, <\text{S}>) = P(\text{time}|\text{VB}) \times P(\text{VB}|<\text{S}>)$$

$$P(\text{VB}|\text{time}, <\text{S}>) = 0.1 \times 0.2 = 0.02$$

↳

$$0.08 > 0.02$$

Previous Probability is 0.00032



$$P(\text{DT}|a, \text{IN}) = P(a|\text{DT}) \times P(\text{DT}|\text{IN})$$

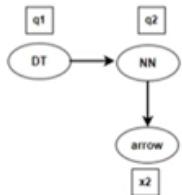
$$P(\text{DT}|a, \text{IN}) = 0.5 \times 0.25 = 0.125$$

$$= 0.125 \times 0.00032 = 0.00004$$

Result: an is DT

Previous Probability is 0.00004

<S>	Time	flies	like	an	arrow	</S>
	NN	VB	IN	DT	NN	



$$P(\text{NN}|\text{arrow}, \text{DT}) = P(\text{arrow}|\text{NN}) \times P(\text{NN}|\text{DT})$$

$$= 0.1 \times 1 = 0.1$$

$$= 0.1 \times 0.00004 = 0.00004$$

Result: arrow is noun

Illustration 4:

Apply Hidden Markov model for the below data and find out the appropriate POS tags for the test data:

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

Emission 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

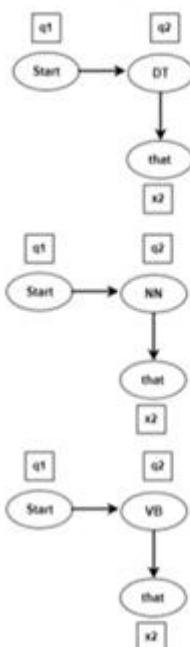
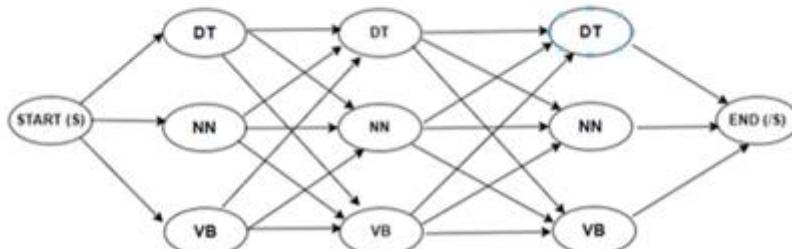
State Transition Probability Matrix

Test Data <S> That girl smiles </S>

That	Girl	Smiles
DT	DT	DT
NN	NN	NN
VB	VB	VB
3x3x3=27 ways		

Test Data <S> That girl smiles </S>

That	Girl	Smiles
DT	DT	DT
NN	NN	NN
VB	VB	VB
3x3x3=27 ways		



$$P(q_2|x_2, q_1) = P(x_2|q_2) \times P(q_2|q_1)$$

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

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

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

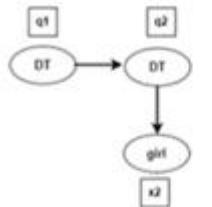
$$P(NN|that, < S >) = 0.00 \times 0.40 = 0$$

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

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

Probability of "that" word is large for DT Part of Speech Tag

Previous Probability is 0.2



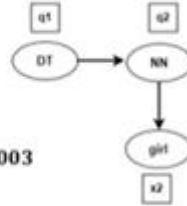
$$P(DT|girl, DT) = P(girl|DT) \times P(DT|DT)$$

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

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

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

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

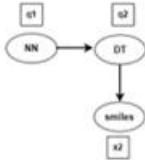


$$P(VB|girl, DT) = P(girl|VB) \times P(VB|DT)$$

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

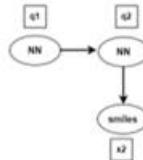
Result: Girl is noun

Previous Probability is 0.003



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

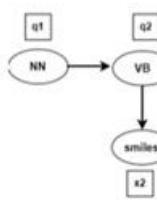
$$P(DT|smiles, NN) = 0 \times 0.30 = 0$$



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

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

$$= 0.00012 \times 0.003 = 0.0000036$$



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

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

$$= 0.08 \times 0.003 = 0.00024$$

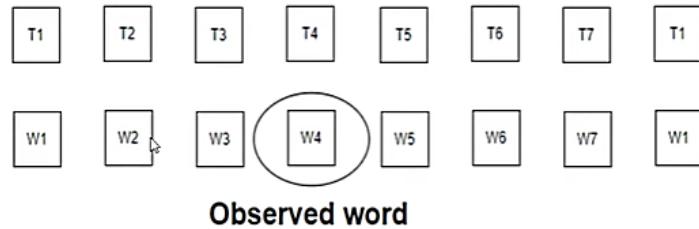
<S>	The	girl	smiles	</S>	
	Det.	Noun	Verb		

Result: Smiles is verb

Maximum Entropy Model (MEM or MaxEnt) in POS Tagging:

- MEM does POS tagging for a given sentence by combining several heterogeneous features in a probabilistic framework.
- It is based on the Principle of Maximum Entropy, which states that of all models that match training data, the one with the highest entropy should be chosen.
- It belongs to the family of classifiers known as the exponential or log-linear classifiers.
- It works by extracting some set of features from the input, combining them linearly (multiplying each feature by a weight and then adding them up), and then using this sum as an exponent.

$$p(c|x) = \frac{1}{Z} \exp \left(\sum_i w_i f_i \right)$$



Advantages

- Flexibility in Feature Usage
- No Independence Assumption
- Probabilistic Framework
- Automatic Learning
- Handles Sparse Data Well
- Domain Adaptability

Limitations:

- Computational Complexity
- Feature Engineering Requirement
- Memory Usage
- Data Sparsity
- Overfitting Risk
- Limited Long-Range Dependency Modeling
- Outperformed by Modern Models

Applications

- Part-of-Speech Tagging
- Named Entity Recognition (NER)
- Word Sense Disambiguation
- Syntactic Parsing
- Information Extraction
- Information Extraction

Illustration 1:

Apply Maximum Entropy model for the given features and compute the appropriate tag.
Assume all features having same weight equal to 1. Beam size = 2.

The	complex	record
Det.	Verb	Verb
Noun	Adj.	Noun

Feature
F1: T_{i-1} =Det and T_i =Adj
F2: T_{i-1} =Noun and T_i =Verb
F3: T_{i-1} =Adj. And T_i =Noun
F4: W_{i-1} =the and T_i =Adj
F5: W_{i-1} =the & W_{i+1} =record and T_i =Adj
F6: W_{i-1} =complex and T_i =Noun
F7: W_{i+1} =complex and T_i =Det
F8: W_{i-1} =NULL and T_i =Noun

Context/Features:

POS Tags	T_{i-1}	T_i	
Word	W_{i-1}	W_i	W_{i+1}

Assumption: All features having same weight which is equal to 1.
Beam Size=2

Features are taken as below. Observed word(W_i). Its previous word (W_{i-1}) and POS Tag (T_i and T_{i-1}). Sentence is: "The complex record"

Feature	The (Det.)	The (Noun)	Complex (Verb)	Complex (Adj.)	Record (Verb)	Record (Noun)
F1: T_{i-1} =Det and T_i =Adj				1		
F2: T_{i-1} =Noun and T_i =Verb				1		
F3: T_{i-1} =Adj. And T_i =Noun						1
F4: W_{i-1} =the and T_i =Adj				1		
F5: W_{i-1} =the & W_{i+1} =record and T_i =Adj				1		
F6: W_{i-1} =complex and T_i =Noun						1
F7: W_{i+1} =complex and T_i =Det	1					
F8: W_{i-1} =NULL and T_i =Noun		1				

Stentence is: "The complex record"

The	complex	record
Det.	Verb	Verb
Noun	Adj.	Noun

Feature	The (Det.)	The (Noun)	Complex (Verb)	Complex (Adj.)	Record (Verb)	Record (Noun)
F1:T _{i-1} =Det and T _i =Adj				1		
F2:T _{i-1} =Noun and T _i =Verb			1			
F3:T _{i-1} =Adj. And T _i =Noun						1
F4: W _{i-1} =the and T _i =Adj					1	
F5:W _{i-1} =the & W _{i-2} =record and T _i =Adj					1	
F6:W _{i-1} =complex and T _i =Noun						1
F7:W _{i-1} =complex and T _i =Det	1					
F8:W _{i-1} =NULL and T _i =Noun		1				

The Word

The (Det) F7 feature is present

The(Noun) F8 feature is present

$$p(\text{Det}|\text{The}) = \frac{\exp(1 * 1)}{\exp(1 * 1) + \exp(1 * 1)} = 0.5$$

$$p(\text{Noun}|\text{The}) = \frac{\exp(1 * 1)}{\exp(1 * 1) + \exp(1 * 1)} = 0.5$$

The(Det), The (Noun) 0.5 probability

Stentence is: "The complex record"

Prvious word The(Det), The (Noun) 0.5 probability

Observed Word is **complex** Word

The	complex	record
Det.	Verb	Verb
Noun	Adj.	Noun

Feature	The (Det.)	The (Noun)	Complex (Verb)	Complex (Adj.)	Record (Verb)	Record (Noun)
F1:T _{i-1} =Det and T _i =Adj				1		
F2:T _{i-1} =Noun and T _i =Verb			1			
F3:T _{i-1} =Adj. And T _i =Noun						1
F4: W _{i-1} =the and T _i =Adj				1		
F5:W _{i-1} =the & W _{i-2} =record and T _i =Adj				1		
F6:W _{i-1} =complex and T _i =Noun					1	
F7:W _{i-1} =complex and T _i =Det	1					
F8:W _{i-1} =NULL and T _i =Noun		1				

First Word: The	Second Word:Complex	Answer
Det	Verb	0.055
Det	Adj.	0.455
Noun	Verb	0.165
Noun	Adj.	0.330

First Word: The	Second Word:Complex	Answer
Det	Verb	0.055
Det	Adj.	0.455
Noun	Verb	0.165
Noun	Adj.	0.330

For The(Det) complex(Verb) No feature is present
complex (Adj.) F1,F4,F5 feature is present

$$p(\text{Verb}|\text{complex}) = \frac{\exp(0)}{[\exp(1 * 1) + \exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.11 \\ = 0.11 \times 0.5 = 0.055$$

$$p(\text{Adj.}|\text{complex}) = \frac{\exp(1 * 1) + \exp(1 * 1) + \exp(1 * 1)}{[\exp(1 * 1) + \exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.89 \\ = 0.89 \times 0.5 = 0.455$$

For The(Noun) complex(Verb) F2 feature is present
complex (Adj.) F4,F5 feature is present

$$p(\text{Verb}|\text{complex}) = \frac{\exp(1)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(1)} = 0.33 \\ = 0.33 \times 0.5 = 0.165$$

$$p(\text{Adj.}|\text{complex}) = \frac{\exp(1 * 1) + \exp(1 * 1)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(1)} = 0.66 \\ = 0.66 \times 0.5 = 0.33$$

For record : record(Verb) No feature is present

record (Noun) F3, F6 feature is present

First Word: The	Second Word:Complex	Answer
Det	Adj.	0.455
Noun	Adj.	0.330

First Word: The	Second Word:Complex	Answer
Det	Adj.	0.455

Feature	The (Det.)	The (Noun)	Complex (Verb)	Complex (Adj.)	Record (Verb)	Record (Noun)
F1:T _{i-1} =Det and T _i =Adj					1	
F2:T _{i-1} =Noun and T _i =Verb			1			
F3:T _{i-1} =Adj. And T _i =Noun						1
F4: W _{i-1} =the and T _i =Adj				1		
F5:W _{i-1} =the & W _{i-2} =record and T _i =Adj				1		
F6:W _{i-1} =complex and T _i =Noun					1	
F7:W _{i-1} =complex and T _i =Det	1					
F8:W _{i-1} =NULL and T _i =Noun		1				

$$p(\text{Verb}|\text{record}) = \frac{\exp(0)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.16 \\ = 0.16 \times 0.455 = 0.0728$$

$$p(\text{Noun}|\text{record}) = \frac{\exp(1 * 1) + \exp(1 * 1)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.85 \\ = 0.85 \times 0.455 = 0.387$$

First Word: The	Second Word:Complex	Answer
Det	Verb	0.055
Det	Adj.	0.455
Noun	Verb	0.165
Noun	Adj.	0.330

Beam Size=2

First Word: The	Second Word:Complex	Answer
Noun	Adj.	0.330

For record : record(Verb) No feature is present

First Word: The	Second Word:Complex	Answer
Det	Adj.	0.455
Noun	Adj.	0.330

record (Noun) F3, F6 feature is present

Feature	The (Det.)	The (Noun)	Complex (Verb)	Complex (Adj.)	Record (Verb)	Record (Noun)
F1:T _{i-1} =Det and T _i =Adj				1		
F2:T _{i-1} =Noun and T _i =Verb			1			
F3:T _{i-1} =Adj. And T _i =Noun					1	
F4: W _{i-1} =the and T _i =Adj				1		
F5:W _{i-1} =the & W _{i-2} =record and T _i =Adj				1		
F6:W _{i-1} =complex and T _i =Noun					1	
F7:W _{i-1} =complex and T _i =Det	1					
F8:W _{i-1} =NULL and T _i =Noun		1				

$$p(\text{Verb}|\text{record}) = \frac{\exp(0)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.16 \\ = 0.16 \times 0.330 = 0.0528$$

$$p(\text{Noun}|\text{record}) = \frac{\exp(1 * 1) + \exp(1 * 1)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.85 \\ = 0.85 \times 0.330 = 0.281$$

Sentence is: "The complex record"

The	complex	record
Det.	Verb	Verb
Noun	Adj.	Noun

First Word The	Second Word Complex	Third Word Record	Answer
Det	Adj. (0.455)	Verb	0.0728
Det	Adj. (0.455)	Noun	0.387
Noun	Adj. (0.330)	Verb	0.0528
Noun	Adj. (0.330)	Noun	0.281

The	complex	record
Det.	Adj	Noun

Illustration 2:

Consider the sentence "The Dog barked." Apply Maximum Entropy model for the given features and compute the appropriate tag for the word "braked". Assume two possible tags are VERB or NOUN.

Feature	Description	Weight
f1	Current word is "barked" and tag is VERB	1.5
f2	Current word is "barked" and tag is NOUN	0.5
f3	Previous word is "dog" and tag is VERB	0.7
f4	Word ends in "ed" and tag is VERB	1
f5	Word length > 5 and tag is NOUN	0.4

For tag = VERB:

Feature

Fires?

Value

f1	YES	1
f2	NO	0
f3	YES (previous word is "dog")	1
f4	YES ("barked" ends in "ed")	1
f5	NO (tag ≠ NOUN)	0

For tag = NOUN:

Feature

Fires?

Value

f1	NO	0
f2	YES	1
f3	NO (tag ≠ VERB)	0
f4	NO (tag ≠ VERB)	0
f5	YES (length > 5)	1

Feature	Description	Weight	VERB	NOUN
f1	Current word is "barked" and tag is VERB	1.5	1	0
f2	Current word is "barked" and tag is NOUN	0.5	0	1
f3	Previous word is "dog" and tag is VERB	0.7	1	0
f4	Word ends in "ed" and tag is VERB	1	1	0
f5	Word length > 5 and tag is NOUN	0.4	0	1

Compute Score

For tag = VERB :

$$\text{score}_{\text{VERB}} = 1.5 \cdot 1 + 0.7 \cdot 1 + 1.0 \cdot 1 = 3.2$$

For tag = NOUN :

$$\text{score}_{\text{NOUN}} = 0.5 \cdot 1 + 0.4 \cdot 1 = 0.9$$

Feature	Description	Weight	VERB	NOUN
f1	Current word is "barked" and tag is VERB	1.5	1	0
f2	Current word is "barked" and tag is NOUN	0.5	0	1
f3	Previous word is "dog" and tag is VERB	0.7	1	0
f4	Word ends in "ed" and tag is VERB	1	1	0
f5	Word length > 5 and tag is NOUN	0.4	0	1

as $P(\text{VERB}) > P(\text{NOUN})$

Barked is Verb

$$Z = \exp(3.2) + \exp(0.9) \approx 24.53 + 2.46 = 26.99$$

$$P(\text{VERB}) = \frac{24.53}{26.99} \approx 0.91$$

$$P(\text{NOUN}) = \frac{2.46}{26.99} \approx 0.09$$

Conditional Random Field (CRF) in POS Tagging

- A CRF is a type of probabilistic graphical model used for structured prediction.
- Unlike models that predict each label independently, CRFs model the entire sequence of labels **jointly**, considering the context and dependencies between neighboring labels.
- This is very useful in POS tagging because the tag of one word often depends on the tags of previous and next words.

How CRF Works in POS Tagging:

- The goal: Given a sentence (sequence of words) predict the most likely sequence of POS tags.
- CRF models the conditional probability directly.
- It defines a global conditional probability over the entire label sequence, which takes into account:

- **Features of the current word** (e.g., the word itself, its suffixes, prefixes, capitalization)
- **Transition features** — how likely a certain tag follows another (e.g., adjective followed by noun is common)
- **Contextual features** from neighboring words and tags

The goal: Given a sentence (sequence of words) $X = (x_1, x_2, \dots, x_n)$, predict the most likely sequence of POS tags $Y = (y_1, y_2, \dots, y_n)$.

CRF models the conditional probability $P(Y|X)$ directly.

Advantages of CRF in POS Tagging

- **Captures dependencies** between neighboring tags (e.g., "DT" usually followed by "NN").
- Can incorporate **rich and overlapping features** of the input words.
- Avoids the independence assumptions that simpler models (like Hidden Markov Models) make.
- Performs better on ambiguous words because it looks at context and transitions.

Brief Mathematical Formulation

The probability of a label sequence Y given input X is:

$$P(Y|X) = \frac{1}{Z(X)} \exp \left(\sum_{t=1}^n \sum_k \lambda_k f_k(y_{t-1}, y_t, X, t) \right)$$

- f_k are feature functions (e.g., "Is current word capitalized?", "Is previous tag DT and current tag NN?").
- λ_k are learned weights for those features.
- $Z(X)$ is a normalization factor ensuring probabilities sum to 1.

Parser:

- A parser is a program that generates a parse tree for the given string, if the string is generated from the underlying grammar.
- It takes input data (text) and converts it into a structural representation after verifying it for valid syntax using formal grammar.
- Parsing helps in determining the syntactic structure of sentences by detecting parts of speech, phrases, and grammatical relationships between words.

Generation of Parse Tree:

$S \rightarrow aABe, A \rightarrow Ab | a, B \rightarrow d$

String: aabcde

Top down approach:

```

S
/\ \
a A B e
/\ \
A b c d
S ⇒ aABe
⇒ aAbcBe
⇒ aabcBe
⇒ aabcde

```

Decision: Which production to use. (Left most Derivation)

Bottom up approach:

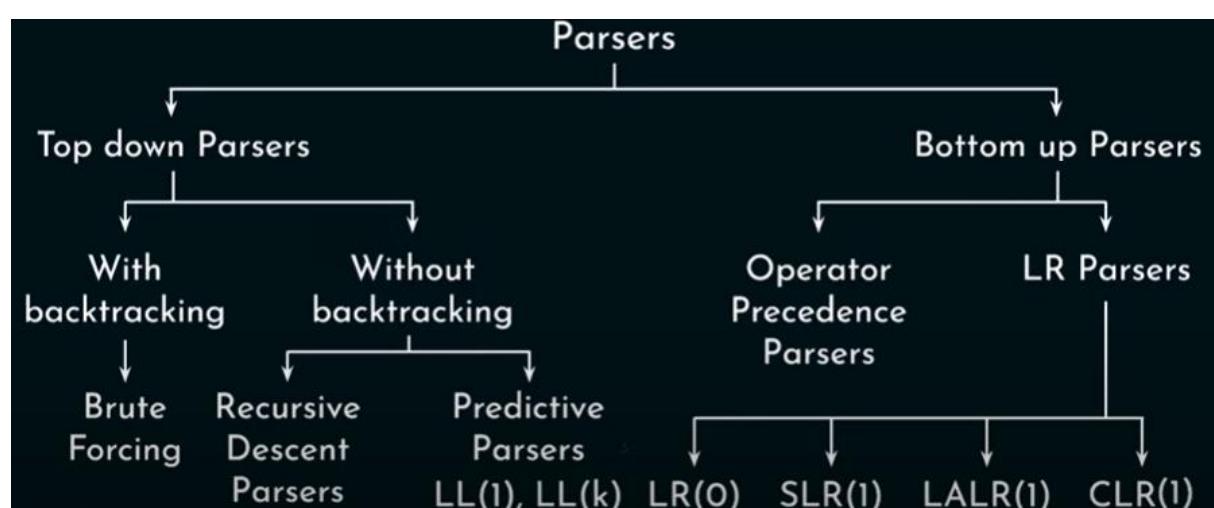
```

S
/\ \
a A B e
/\ / \
a b c d
S ⇒ aABe
⇒ aAde
⇒ aAbcde
⇒ aabcde

```

Decision: When to reduce. (Right most Derivation) - In reverse.

Classification of Parsers:



The figure presents a hierarchical classification of **parsers**, which are programs used in compilers to analyze the structure of strings based on grammar rules. Parsers are broadly divided into two main categories:

1. **Top-down Parsers**
2. **Bottom-up Parsers**

1. Top-down Parsers

These parsers start constructing the parse tree from the root (start symbol) and work down to the leaves (input symbols).

- **With backtracking:**
 - These parsers try a production rule and if it fails, they backtrack and try another rule.
 - Example: **Brute Forcing** — tries all possibilities until one fits.
- **Without backtracking:**
 - These parsers do not need to go back once a decision is made.
 - Further divided into:
 - **Recursive Descent Parsers:** Use recursive procedures for each non-terminal to parse input.
 - **Predictive Parsers:** Use lookahead symbols to decide which production to use, avoiding backtracking.
 - Examples: **LL(1), LL(k)** parsers (where LL means Left-to-right scanning of input and Leftmost derivation, and k is the number of lookahead tokens).

2. Bottom-up Parsers

These parsers start from the leaves (input symbols) and work up to the root (start symbol), effectively reducing the string to the start symbol by applying grammar rules in reverse.

- **Operator Precedence Parsers:**
 - Use precedence rules between operators to decide parsing.
- **LR Parsers (Left-to-right scanning of input and Rightmost derivation in reverse):**
 - Powerful class of bottom-up parsers that can handle a large class of grammars.
 - Subtypes include:
 - **LR(0):** No lookahead used.
 - **SLR(1) (Simple LR):** Uses one symbol lookahead with simple follow sets.
 - **LALR(1) (Look-Ahead LR):** More powerful than SLR(1), merges states to reduce parser size.
 - **CLR(1) (Canonical LR):** Most powerful LR parser with full context and lookahead.
