১) POS Tagging কী?

আমরা যখন কোনো বাক্য পড়ি, তখন স্বাভাবিকভাবেই বুঝতে পারি—
কোন শব্দটা নাম, কোনটা কাজ, কোনটা গুণ বোঝাচ্ছে।

কিন্তু কম্পিউটার নিজে থেকে এটা বুঝতে পারে না।
কম্পিউটারকে এই জিনিসটা শেখানোর জন্য যে প্রক্রিয়াটা ব্যবহার করা হয়, সেটাই **POS Tagging**।

POS Tagging-এর সংজ্ঞা

POS (Part of Speech) Tagging হলো এমন একটি প্রক্রিয়া যেখানে একটি বাক্যের প্রতিটি শব্দকে তার ব্যাকরণগত ভূমিকা অনুযায়ী আলাদা করে চিহ্নিত করা হয়।

সহজভাবে বললে:

> একটি শব্দ বাক্যে কী কাজ করছে—এই তথ্য বের করাই POS Tagging।

সাধারণ কিছু **POS Tag**

- Noun → নাম বোঝায়
- Verb → কাজ বোঝায়
- Adjective → গুণ বা বৈশিষ্ট্য বোঝায়
- Adverb → কাজ কীভাবে হচ্ছে বোঝায়
- Pronoun → নামের বদলে ব্যবহৃত হয়
- Determiner → কোনো নামকে নির্দিষ্ট করে
- Preposition → শব্দগুলোর মধ্যে সম্পর্ক বোঝায়

উদাহরণ

বাক্য: **“I love NLP”**

- I → Pronoun
- love → Verb
- NLP → Noun / Proper Noun

এখানে প্রতিটি শব্দের দায়িত্ব আলাদা। POS Tagging এই দায়িত্বগুলো কম্পিউটারকে বুঝিয়ে দেয়।

---

২) **POS Tagging** কেন দরকার?

NLP-এ কাজ করার সময় শুধু শব্দ জানা যথেষ্ট নয়।
একই শব্দ ভিন্ন জায়গায় ভিন্ন কাজ করতে পারে।

উদাহরণ:

- “book a ticket” → book = Verb
- “read a book” → book = Noun

যদি আমরা না জানি শব্দটা noun না verb, তাহলে বাক্যের অর্থই ভুল হয়ে যেতে পারে।

এই সমস্যার সমাধান করে POS Tagging।

POS Tagging দরকার কারণ**:**

1. বাক্যের গঠন (structure) বোঝা যায়
2. কোন শব্দটা গুরুত্বপূর্ণ সেটা ধরা যায়
3. শব্দের সঠিক অর্থ বুঝতে সাহায্য করে
4. NLP pipeline-এর পরের কাজগুলোর ভিত্তি তৈরি করে

---

৩) **POS Tagging-**এর ব্যবহার **(Applications)**

এখন চলো দেখি বাস্তবে এটা কোথায় কাজে লাগে।

৩.১ **Named Entity Recognition (NER)**

NER-এর কাজ হলো বাক্য থেকে মানুষ, জায়গা, কোম্পানি ইত্যাদি বের করা।

বেশিরভাগ সময় এই entity গুলো **Proper Noun** হয়।
POS Tag থাকলে সিস্টেম সহজেই বুঝতে পারে—
“এই শব্দটা সম্ভবত কোনো নাম।”

তাই NER-এর আগে POS Tagging করলে কাজ অনেক সহজ হয়।

৩.২ **Question Answering System**

Question Answering system-কে বুঝতে হয়—

- কে কাজটা করছে
- কী কাজ হচ্ছে
- কার ওপর কাজটা হচ্ছে

POS Tag থাকলে:

- subject
- verb
- object

এই অংশগুলো আলাদা করে ধরা যায়।
ফলে প্রশ্ন বুঝে সঠিক উত্তর দেওয়ার ক্ষমতা বাড়ে।

৩.৩ **Word Sense Disambiguation (WSD)**

একই শব্দ ভিন্ন বাক্যে ভিন্ন অর্থ দিতে পারে।

উদাহরণ: **left**

- “He left the room” → left = চলে গেছে (Verb)
- “Turn left” → left = দিক (Noun)

POS Tag দেখে সিস্টেম বুঝতে পারে—
এখানে শব্দটা verb না noun।
এভাবেই সঠিক অর্থ নির্ধারণ করা যায়।

৩.৪ **Chatbot**

Chatbot-কে user-এর বাক্য বিশ্লেষণ করে বুঝতে হয়:

- user কী চাইছে
- কোন শব্দগুলো intent বোঝাচ্ছে
- কোনগুলো entity

POS Tagging বাক্যকে structured করে তোলে।
ফলে intent detection এবং response generation আরও accurate হয়।

৩) **SpaCy** দিয়ে **POS Tagging (Practical)**

এখন পর্যন্ত আমরা বুঝেছি POS Tagging কী।
এখন প্রশ্ন হলো—বাস্তবে **Python** দিয়ে এটা করি কীভাবে**?**

এর জন্য সবচেয়ে জনপ্রিয় লাইব্রেরিগুলোর একটা হলো **SpaCy**।

SpaCy কেন ব্যবহার করবো**?**

- খুব fast
- production-ready
- POS tagging, dependency parsing সব built-in
- আলাদা করে model train না করেও কাজ করা যায়

৩.১ **SpaCy** দিয়ে **POS Tagging: Basic Code**

ধরি আমাদের একটা sentence আছে:

> “Apple is looking at buying a startup”

কোড

```
import spacy


# English model load

nlp = spacy.load("en_core_web_sm")


text = "Apple is looking at buying a startup"

doc = nlp(text)


for token in doc:

    print(token.text, token.pos_, token.tag_)
```

এই কোডে কী হচ্ছে?

1. `nlp(text)`
   → sentence প্রসেস হয়ে token এ ভাগ হচ্ছে
2. `token.text`
   → শব্দটা কী
3. `token.pos_`
   → শব্দটার বড় ক্যাটাগরি
4. `token.tag_`
   → শব্দটার ডিটেইল **grammatical info**

৩.১ Coarse-grained vs Fine-grained Tag

SpaCy দুই ধরনের POS tag দেয়।

১) Coarse-grained Tag (`pos_`)

এটা খুব general label দেয়।

উদাহরণ:

- NOUN
- VERB
- ADJ
- ADV
- PROPN

এটা দিয়ে বোঝা যায় শব্দটা মোটামুটি কোন category-এর।

---

২) Fine-grained Tag (`tag_`)

এটা অনেক বেশি detail দেয়।

উদাহরণ:

- NN → singular noun
- NNS → plural noun
- VBD → past tense verb
- VBG → verb (ing form)

অর্থাৎ:

- tense কী
- singular না plural
- verb-এর form কী

এসব hint পাওয়া যায়।

সহজভাবে মনে রাখার ট্রিক

- `pos_` → বড় ছবি
- `tag_` → ভিতরের ডিটেইল

৩.২ SpaCy দিয়ে Visual Representation (displacy)

শুধু tag দেখলে অনেক সময় বোঝা কঠিন হয়।
SpaCy এজন্য **visual** ভাবে **sentence structure** দেখাতে পারে।

কোড

```
from spacy import displacy


displacy.render(doc, style="dep", jupyter=True)
```

এখানে কী দেখা যায়?

- কোন শব্দ subject
- কোনটা verb
- কোনটা object
- কোন শব্দ কাকে depend করছে

এইটা শেখার সময় খুব কাজে লাগে, বিশেষ করে:

- sentence structure বুঝতে
- dependency grammar শিখতে

---

৪) **POS Tagging** ভিতরে কীভাবে কাজ করে: **HMM (Concept)**

এখন খুব গুরুত্বপূর্ণ একটা কথা।

SpaCy দিয়ে tag পাওয়া সহজ,
কিন্তু ভিতরে আসলে কী লজিক কাজ করে?

ক্লাসিক একটা পদ্ধতি হলো **Hidden Markov Model (HMM)**।

৪.১ **HMM-**এর মূল ধারণা

HMM-এ দুইটা জিনিস থাকে:

১) **Observed Data (**যেটা আমরা দেখি**)**

- sentence-এর শব্দগুলো

উদাহরণ:

I love NLP

এই শব্দগুলো আমরা চোখে দেখি।

---

২) **Hidden State (**যেটা আমরা দেখি না**)**

- আসল POS tag sequence

উদাহরণ:

PRON VERB NOUN

এই tag গুলো লুকানো থাকে।
আমাদের কাজ হলো:

> শব্দ দেওয়া আছে → সবচেয়ে সম্ভাব্য tag বের করা

এইটাই HMM-এর goal।

৪.২ **HMM-**এ দুইটা **Probability** খুব গুরুত্বপূর্ণ

(ক) Emission Probability

একটা tag থেকে একটা শব্দ আসার সম্ভাবনা।

ফর্মুলা:

P(word | tag)

মানে:

> যদি tag = VERB হয়, তাহলে এই word আসার সম্ভাবনা কত?

উদাহরণ:

- P("go" | VERB) → বেশি
- P("go" | NOUN) → কম
- P("Google" | PROPN) → বেশি

এইটা শেখা হয় training data থেকে।

(খ) Transition Probability

একটা tag-এর পরে আরেকটা tag আসার সম্ভাবনা।

ফর্মুলা:

P(tag_i | tag_(i-1))

উদাহরণ:

- DET → NOUN (খুব common)
- NOUN → VERB (common)
- DET → VERB (rare)

---

৫) বড় সমস্যা: সব **tag combination brute-force** করা অসম্ভব

ধরি তোমার কাছে একটা বাক্য আছে।

- শব্দ সংখ্যা = **n**
- প্রতিটা শব্দের জন্য সম্ভাব্য POS tag সংখ্যা = **k**

তাহলে একটা বাক্যের জন্য সম্ভাব্য tag sequence কত?

কেন **(k^n)?**

কারণ:

- ১ম শব্দের জন্য kটা choice
- ২য় শব্দের জন্য kটা choice
- ...
- nতম শব্দের জন্য kটা choice

সবগুলো মিলিয়ে: k x k x k x …….k = k^n

এটাই combinational explosion।

বাস্তব উদাহরণ দিয়ে বোঝি

ধরি:

- বাক্যে ১০টা শব্দ (n = 10)
- প্রতিটা শব্দ গড়ে ১২টা **tag** হতে পারে (k = 12)

তাহলে: 12 ^{ 10 }

এটা কত বড়?

12^2 = 144
12^3 = 1728
12^4 = 20736
12^5 = 248832
12^6 = 2985984
12^7 = 35831808
12^8 = 429981696
12^9 = 5159780352
12^10 = 61917364224

মানে প্রায় ৬১ বিলিয়ন সম্ভাব্য sequence।

এখন ভাবো, brute-force করলে:

- প্রতিটা sequence-এর probability বের করতে হবে
- তারপর সর্বোচ্চটা pick করতে হবে

এটা বাস্তবে খুব সময়সাপেক্ষ, অনেক ক্ষেত্রে অসম্ভব।

Brute-force কী?

Brute-force মানে হলো:

> কোনো সমস্যার সমাধান বের করার জন্য
> সব সম্ভাব্য উপায় এক এক করে ট্রাই করা,
> তারপর যেটা সবচেয়ে ভালো, সেটা বেছে নেওয়া।

এখানে কোনো smart shortcut থাকে না।
সব কিছু চেক করাই হলো brute-force।

Brute-force POS Tagging-এ মানে কী?

POS tagging-এর ক্ষেত্রে brute-force মানে:

1. একটা বাক্য নাও
2. প্রতিটা শব্দের জন্য যতগুলো possible POS tag আছে, সব ধরো
3. সেগুলো দিয়ে সব সম্ভাব্য **tag sequence** বানাও
4. প্রতিটা sequence-এর probability হিসাব করো

5. যেই sequence-এর probability সবচেয়ে বেশি, সেটাকে final answer ধরো

এখন একটা ছোট উদাহরণ দিয়ে বুঝি

ধরি বাক্য:

```
I love NLP
```

ধরি সম্ভাব্য tag:

- I → {PRON}
- love → {NOUN, VERB}
- NLP → {NOUN, PROPN}

এখন brute-force কী করবে?

Step 1: সব **possible sequence** বানানো

Sequence 1:

```
PRON NOUN NOUN
```

Sequence 2:

```
PRON NOUN PROPN
```

Sequence 3:

```
PRON VERB NOUN
```

Sequence 4:

```
PRON VERB PROPN
```

মোট 4টা sequence।

Step 2: প্রতিটা **sequence-**এর **probability** হিসাব করা

ধরি probability =
(emission probability) × (transition probability)

Brute-force এক এক করে হিসাব করবে:

- PRON → NOUN → NOUN
- PRON → NOUN → PROPN
- PRON → VERB → NOUN
- PRON → VERB → PROPN

তারপর compare করবে।

Step 3: যেটা সবচেয়ে বড়, সেটাই **answer**

এখানে brute-force ঠিকই answer দেবে।
কিন্তু এটা খুব ছোট বাক্য।

সমস্যা কোথায়?

এখন ভাবো বাস্তব sentence:

- শব্দ = 10টা
- প্রতিটা শব্দে average tag = 10টা

তাহলে:

$10^{10} = 10,000,000,000$

মানে ১০ বিলিয়ন sequence।

Brute-force-কে:

- ১০ বিলিয়ন sequence বানাতে হবে
- ১০ বিলিয়ন probability হিসাব করতে হবে

এটা বাস্তবে অসম্ভব।

তাহলে এখানে **brute-force** “কিভাবে কাজ করত”?

POS tagging-এ brute-force কাজ করত এইভাবে:

1. প্রতিটা শব্দের সব tag নাও
2. Cartesian product করে সব tag sequence বানাও
3. প্রতিটা sequence-এর জন্য:
    - emission probability multiply
    - transition probability multiply
4. সব sequence-এর মধ্যে max probability খুঁজে বের করো

Conceptually ঠিক, কিন্তু practically অকার্যকর।

৬) সমাধান: **Viterbi Algorithm (Efficient Optimization)**

Viterbi হলো এমন একটা algorithm যা brute-force না করে best tag sequence বের করে দেয়।

Viterbi-এর মূল আইডিয়া **(Intuition)**

তুমি যদি brute-force করো, তুমি বলছ:
“আমি সব রাস্তা দিয়ে হেঁটে দেখি, শেষে কোনটা best ছিল দেখি।”

Viterbi বলে:
“সব রাস্তা দেখা লাগবে না।
প্রতিটা ধাপে আমি শুধু সবচেয়ে সম্ভাব্য পথগুলোর best version রেখে দেই।”

এটা সম্ভব হয় **Dynamic Programming (DP)** দিয়ে।

Viterbi আসলে কী **store** করে**?**

বাক্যের প্রতিটা word position i তে, প্রতিটা tag t এর জন্য দুইটা জিনিস রাখে:

১**) Best score/probability**

“এই position i তে tag t এ এসে পৌঁছানোর মধ্যে সবচেয়ে বেশি probability কত?”

এটা একটা টেবিলের মতো (DP table)।

২**) Backpointer**

“এই best probability-টা আসছে আগের কোন tag থেকে?”

মানে: current tag-এর আগে কোন tag ছিল, যেখান থেকে আসলে best হলো।

Why this reduces computation?

Brute-force-এ তুমি সম্ভাব্য সব sequence চেক করো: (k^n)

Viterbi-এ তুমি প্রতি position-এ:

- প্রতিটা tag (k) এর জন্য
- আগের সব tag (k) compare করো

তাই মোট কাজ প্রায়: n x k^2

এটা exponential না, polynomial।

উদাহরণ:

n=10, k=12 হলে brute-force: 12^10 (৬১ বিলিয়ন)
Viterbi: 10 × 12^2 = 10 × 144 = 1440 steps (আনুমানিক)

এখানে পার্থক্য বিশাল।

৬.২ **Backtracking (**কীভাবে **final sequence** বের হয়**)**

Viterbi কাজ করে দুই ধাপে:

ধাপ ১**: Forward pass (DP fill)**

শুরু থেকে শেষ পর্যন্ত যায়।
প্রতিটা cell-এ:

- best probability রাখে
- backpointer রাখে

ধাপ ২**: Backtracking**

শেষ শব্দে গিয়ে:

- যেই tag-এর probability সবচেয়ে বড়, সেটাকে final tag ধরে

তারপর backpointer ধরে ধরে উল্টো পথে হাঁটে:

- শেষ word-এর tag থেকে
- আগের word-এর tag
- আগের word-এর tag
  এভাবে পুরো sequence বের হয়।

মানে Viterbi “best path” বের করে, কারণ সে path বানানোর সময়ই “best previous” track করে রেখেছিল।

---

৭) হাতে-কলমে উদাহরণ: “Will Will Google Sakha ”

এটা খুব ভালো উদাহরণ, কারণ এখানে একই শব্দ ভিন্নভাবে ব্যবহার হতে পারে।

বাক্য:
“Will Will Google Sakha”

এখানে ambiguity আছে।

সম্ভাব্য **meaning**

- প্রথম “Will” হতে পারে person name (Proper Noun / PROPN)
- দ্বিতীয় “Will” আবার নামও হতে পারে, বা modal verbও হতে পারে (AUX/MD)
- “Google” হতে পারে company name (PROPN) বা verb (to google)
- “Sakha ” সাধারণত proper noun

এখন প্রশ্ন:
কম্পিউটার কীভাবে ঠিক করবে কোনটা কোনটা?

এখানে **HMM** কী দেয়?

HMM দুইটা probability দেয়:

১) Emission: P(word | tag)

মানে “এই tag হলে এই word আসার chance কত?”

উদাহরণ:

- P("Will" | PROPN) বেশি (কারণ নাম Will আছে)
- P("Will" | AUX) ও সম্ভব (কারণ will modal verb)
- P("Google" | PROPN) বেশি
- P("Google" | VERB) ও সম্ভব

২) Transition: P(tag_i | tag_{i-1})

মানে “একটা tag-এর পরে আরেকটা tag আসার chance কত?”

উদাহরণ:

- PROPN → PROPN (নাম নাম পাশাপাশি আসতে পারে: Will Will)
- AUX → VERB (will go, will eat টাইপ)
- DET → NOUN (the dog)

তাহলে **final decision** কীভাবে হয়?

Viterbi বলে:
“আমি emission আর transition দুইটাই multiply করে দেখব, কোন sequence সবচেয়ে বেশি probable।”

ধরি দুইটা candidate sequence:

Candidate A:

Will/PROPN Will/PROPN Google/PROPN Sakha/PROPN
মানে: ৪টা নাম (সব proper noun)

এটা plausible, কারণ এটা একটা list/নামও হতে পারে।

Candidate B:

Will/AUX Will/VERB Google/NOUN Sakha/PROPN
এটা অনেক কম natural। কারণ will (AUX) এর পরে সাধারণত base verb লাগে, “Will” verb হিসেবে weird।

তাই transition probability A-এর ক্ষেত্রে বেশি natural হতে পারে।

Viterbi এইগুলা হিসাব করে সবচেয়ে probable path রাখে এবং শেষে সেটাই output দেয়।

8) Classical vs Modern POS Tagging Approach

তুমি এখন Classical part জানো (HMM + Viterbi)।
আরও দুইটা approach আছে, যেগুলো জানা থাকলে picture complete হয়।

(ক) Rule-based POS Tagging

- Grammar rule লিখে tag assign করা
- যেমন:
    - যদি শব্দ “the” হয় → DET
    - যদি শব্দের শেষে “-ly” থাকে → ADV

সমস্যা:

- Rule লিখতে অনেক সময় লাগে
- Language variation handle করতে পারে না

এজন্য আজকাল কম ব্যবহার হয়।

(খ) ML-based POS Tagging

- Feature-based classifier ব্যবহার করা হয়
- Feature যেমন:
    - word itself
    - previous word
    - suffix/prefix
    - previous tag

উদাহরণ:

- CRF (Conditional Random Field)

HMM থেকে বেশি accurate, কিন্তু feature engineering লাগে।

(গ) Deep Learning-based POS Tagging

- LSTM / BiLSTM / Transformer ব্যবহার হয়
- Word embedding দিয়ে context শেখে

আজকাল:

- SpaCy
- BERT-based tagger
  এই category-তে পড়ে।

এখানে আলাদা করে transition probability লিখতে হয় না—model নিজেই শিখে নেয়।

---

9) POS Tagging-এর Limitations (খুব গুরুত্বপূর্ণ)

POS Tagging perfect না। এটা জানাও জরুরি।

(ক) **Context** খুব **complex** হলে ভুল হয়

উদাহরণ:

- “Visiting relatives can be annoying”

এখানে “visiting” noun না verb—এটা ambiguous।

(খ) **Informal / Social Media Text**

- spelling ভুল
- slang
- emoji
  এইগুলো POS tagging-এ সমস্যা করে।

(গ) **Low-resource language**

- যেসব ভাষায় labelled data কম (যেমন বাংলা)
- সেখানে POS tagging তুলনামূলক কঠিন।

---

10) **POS Tagging** কোথায় দাঁড়ায় **NLP Pipeline-**এ

POS tagging কখনো একা কাজ করে না।

সাধারণ pipeline:

1. Text cleaning
2. Tokenization
3. POS Tagging
4. Chunking / Parsing
5. NER
6. Downstream task (QA, chatbot, classification)

মানে POS tagging হলো foundation layer।