corpus

- A **corpus** simply means a **collection of texts or sentences** stored in a computer for language study or NLP.
- If I collect 10,000 English sentences from newspapers — that becomes an **English corpus.**
  If I collect 5,000 tweets in Marathi — that's a **Marathi corpus.**
- **So corpus means *data* for the computer to learn language**.

- **So corpus = textbook for a computer**

Types of Corpora

- **Monolingual corpus** – one language (only English).
- **Parallel corpus** – two languages side by side (English–Hindi).
- **Annotated corpus** – corpus with extra info like word types or meanings.
- We use corpus to train models — like translation, spell check, chatbot, or sentiment analysis.

What is Annotation

- Annotation means **adding labels or information** to the text so the computer understands it better.
- Sentence → Ravi plays football.
  After annotation → Ravi/Name plays/Verb football/Noun
- So we *mark* each word with its type — that's annotation

Why We Do It

- Computers don’t understand words — they understand labels.
  When we tag words, the computer can learn patterns like we do in grammar.
- Example:
- “After ‘to’, mostly a verb comes — the computer learns this by looking at tags.”

Types of Annotations

- **POS Tagging** → marks word type (noun, verb, etc.)
- **NER (Named Entity)** → marks names, places, dates
- **Sentiment Tag** → marks feelings (positive / negative)
- Example:
  Pune/Location is/Verb beautiful/Adjective city/Noun.

How Annotation Happens

1. Sometimes humans tag the text (manual)

2. sometimes tools like SpaCy or NLTK do it automatically.

Task-Specific Training Corpora

- Different NLP tasks need **different types of data (corpora)** for training.
  That's called a **task-specific corpus** — a corpus made for one particular job.
- Example:
  For grammar → POS tagging corpus
  For emotion detection → Sentiment corpus
  For translation → Parallel corpus

Why We Need It

- Just like we study different subjects from different books,
  computers also need different datasets for different NLP tasks.”
- Example:
  A chatbot needs “question–answer” data,
  but a translation app needs “English–Hindi” sentence pairs.

One-Line Summary

Task-Specific Corpus = data prepared for one NLP task like tagging, translation, or sentiment analysis.

Statistical Techniques in NLP

- Statistical techniques help the computer **learn patterns** from text —
  like which word usually follows which, or what a word means in context.

Sentence: *I want to eat* ___

- The computer guesses **"food"** because it has seen it many times in that pattern.
- So it **uses probability**, not grammar rules.

Why They Are Needed

- Language has too many rules and exceptions. Statistical models don't memorize — they **learn from data** and make smart guesses
- **Example:**
- Word *"bank"* — could mean *river bank* or *money bank*.
  The model checks the surrounding words (context) and picks the most likely meaning.

Important Statistical Models

Model	Simple Meaning	Example Use
HMM (Hidden Markov Model)	Predicts next tag or word based on previous one	POS Tagging, Speech Recognition
MEMM (Maximum Entropy Markov Model)	Uses extra clues like capitalization or nearby words	NER, Tagging
CRF (Conditional Random Field)	Looks at the whole sentence before deciding	NER, Chunking
SVM (Support Vector Machine)	Divides data into classes	Spam vs. Not Spam, Sentiment

Example-HMM

- “I eat fish”

We have:

- **Words (observed):** `I`, `eat`, `fish`
- **Tags (hidden):** `Pronoun (PRON)`, `Verb (V)`, `Noun (N)`

Our job:

Find which tag sequence (PRON → V → N, or something else) has the **highest probability**.

📜 Probabilities Given

1️⃣ Start Probabilities

Tag	Start Probability
PRON	0.5
V	0.1
N	0.4

2️⃣ Transition Probabilities (from → to)

From → To	PRON	V	N
PRON	0.1	0.8	0.1
V	0.2	0.2	0.6
N	0.1	0.1	0.8

3 Emission Probabilities

Word	P(word | PRON)	P(word | V)	P(word | N)
I	0.9	0.05	0.05
eat	0.05	0.9	0.05
fish	0.1	0.1	0.8

Step 1: For the first word → "I"

We calculate for each tag separately:

Case 1: Tag = PRON

$$P(PRON, I) = P(\text{start as PRON}) \times P(I|PRON)$$

$$= 0.5 \times 0.9 = 0.45$$

Case 2: Tag = V

$$P(V, I) = 0.1 \times 0.05 = 0.005$$

Case 3: Tag = N

$$P(N, I) = 0.4 \times 0.05 = 0.02$$

✅ So, for the word "I", the most probable tag is **PRON (0.45)**.

Step 2: For the second word → "eat"

Now we consider **transitions** from each possible previous tag to a new tag.

We'll multiply:

$$P(\text{previous tag, previous word}) \times P(\text{transition}) \times P(\text{emission})$$

Let's calculate for each possible case.

Case 1: Previous tag PRON → Current tag V

$$= 0.45 \times P(V|PRON) \times P(eat|V)$$

$$= 0.45 \times 0.8 \times 0.9 = 0.324$$

Case 2: Previous tag PRON → Current tag N

$$= 0.45 \times 0.1 \times 0.05 = 0.00225$$

Case 3: Previous tag N → Current tag V

$$= 0.02 \times 0.1 \times 0.9 = 0.0018$$

Case 4: Previous tag V → Current tag V

$$= 0.005 \times 0.2 \times 0.9 = 0.0009$$

✅ Highest is **0.324 (PRON → V)**

So now we know the best path till now is:

So if we had 3 tags, we must check **3 × 3 = 9 possible transitions**:

PRON→PRON, PRON→V, PRON→N,

V→PRON, V→V, V→N,

N→PRON, N→V, N→N.

So why **Case 4 (V → V)**?

> "eat" might also be a **Verb**, and maybe the **previous word "I"** was wrongly guessed as a **Verb** (not a Pronoun).

n HMM, **we cannot discard any path too early** —

we must calculate every path's probability, because sometimes a lower-probability start can lead to a high

probability overall sentence later.

Step 3: For the third word → "fish"

Now, our previous best tag is **V**, and current word is "fish".

We again apply:

$$P(previous) \times P(transition) \times P(emission)$$

Case 1: V → N

$$= 0.324 \times 0.6 \times 0.8 = 0.1555$$

Case 2: V → V

$$= 0.324 \times 0.2 \times 0.1 = 0.00648$$

✅ Highest is **0.1555 (V → N)**

So our final path is:

```scss
I (PRON) → eat (V) → fish (N)
```

> "In Step 3, we don't have to check all combinations again.
> Because we already found that 'eat' is most likely a Verb,
> we only continue transitions starting from that Verb.
> So we check how likely the next word 'fish' is to be a Noun or another Verb.
> That's why we only had two cases — V→N and V→V."

In Hidden Markov Models, the **final probability** of a sentence with a particular tag sequence is:

$$P(\text{words}, \text{tags}) = P(\text{start tag}) \times P(\text{word}_1|\text{tag}_1) \times P(\text{tag}_2|\text{tag}_1) \times P(\text{word}_2|\text{tag}_2) \times P(\text{tag}_3|\text{tag}_2) \times P(\text{word}_3|\text{tag}_3)$$

Component	Probability	Explanation
P(start as PRON)	0.5	Sentence starts with pronoun
P(I	PRON)	0.9
P(V	PRON)	0.8
P(eat	V)	0.9
P(N	V)	0.6
P(fish	N)	0.8

✳️ Step 3: Multiply all of them

$$P = 0.5 \times 0.9 \times 0.8 \times 0.9 \times 0.6 \times 0.8$$

Now calculate step-by-step:

Step	Operation	Result
1	0.5 × 0.9	0.45
2	0.45 × 0.8	0.36
3	0.36 × 0.9	0.324
4	0.324 × 0.6	0.1944
5	0.1944 × 0.8	**0.15552**

✅ **Final Probability = 0.1555 (approx)**

We simply multiply all key probabilities in this best sequence:

$$P = 0.5 \times 0.9 \times 0.8 \times 0.9 \times 0.6 \times 0.8$$

Let's compute step-by-step:

1. $0.5 \times 0.9 = 0.45$
2. $0.45 \times 0.8 = 0.36$
3. $0.36 \times 0.9 = 0.324$
4. $0.324 \times 0.6 = 0.1944$
5. $0.1944 \times 0.8 = 0.1555$

✅ Final Probability = **0.1555**

This is the probability of the entire sequence:

> (PRON, V, N) for the sentence "I eat fish".

🌸 Example 2: "She sings well"

We want to predict **Part-of-Speech (POS)** tags using an **HMM**.

🧩 Step 1: Define what we have

Word	Hidden Tag (we need to find)
She	?
sings	?
well	?

Possible POS tags (states):

- **PRON** → Pronoun
- **V** → Verb
- **ADV** → Adverb

1 Start probabilities

Tag	Probability
PRON	0.6
V	0.2
ADV	0.2

👉 A sentence usually starts with a **Pronoun**, so it has the highest chance.

---

2 Transition probabilities

From → To	PRON	V	ADV
PRON	0.1	0.8	0.1
V	0.1	0.2	0.7
ADV	0.3	0.4	0.3

👉 Example: After a **Pronoun**, a **Verb** (0.8) usually comes — like "She sings."

---

3 Emission probabilities

Word	P(word | PRON)	P(word | V)	P(word | ADV)
She	0.9	0.05	0.05
sings	0.05	0.9	0.05
well	0.1	0.1	0.8

What is a **Sentence**?

A **sentence** is simply a **sequence of words.**

For example:

> **Sentence:** "I like mango"

It has **3 words** — I, like, mango.

NLP tries to **find how likely this sentence is**, using probabilities of words.

What is a **Unigram**?

Unigram means **single word** (1 word at a time).
It checks the probability of each word **independently** — not based on what comes before it.

🥖 Formula:

$$P(sentence) = P(w1) \times P(w2) \times P(w3)...$$

🎹 Example:

> Sentence: "I like mango"

$$P = P(I) \times P(like) \times P(mango)$$

Let's assume:

- P(I) = 0.3
- P(like) = 0.2
- P(mango) = 0.4

Then:

$$P(sentence) = 0.3 \times 0.2 \times 0.4 = 0.024$$

✅ So Unigram treats all words **independent** (doesn't care about word order).

What is a **Bigram**?

Bigram means **pair of words (2 words together).**

It looks at how often one word **follows another.**

🖊️ Formula:

$$P(sentence) = P(w1) \times P(w2|w1) \times P(w3|w2)$$

🎞️ Example:

> Sentence: "I like mango"

$$P = P(I) \times P(like|I) \times P(mango|like)$$

This means:

- P(I) = how often sentence starts with "I"
- P(like|I) = how often "like" follows "I"
- P(mango|like) = how often "mango" follows "like"

So **Bigram model understands sequence and context** better than Unigram.

What is a **Trigram**?

Trigram means **three words together** — it looks at how likely a word appears **given the two previous words**.

🖊️ Formula:

$$P(sentence) = P(w1) \times P(w2|w1) \times P(w3|w1, w2)$$

🎑 Example:

> Sentence: "I like mango"

$$P = P(I) \times P(like|I) \times P(mango|I, like)$$

So the probability of "mango" depends on **both** the previous words — "I like".

✅ Trigram captures **more context** and gives more accurate predictions than Bigram.

Comparison Table

Model	Meaning	Looks at	Example	Comment
Unigram	1 word	Individual words	P(I), P(like), P(mango)	No context
Bigram	2 words	Previous word	P(like	I), P(mango
Trigram	3 words	2 previous words	P(mango	I, like)
Sentence Probability	Full sentence	Product of all probabilities	P(I)×P(like	I)×P(mango

Why We Use These in NLP

Use	Explanation
Predict next word	"I like ___" → likely "mango", "apples", etc.
Speech recognition	To choose most probable sentence
Grammar check	Finds unnatural word order
Machine translation	Helps choose correct word sequence