Lecture 9

What is POS Tagging?

POS tagging is the process of labeling each word in a sentence with its part of speech—think nouns, verbs, adjectives, etc. It's a key step in Natural Language Processing (NLP) to understand sentence structure and meaning. For example, in "The cat runs," POS tagging identifies "The" as a determiner, "cat" as a noun, and "runs" as a verb.

- **Why It Matters:** Helps machines grasp grammar, disambiguate words (e.g., "lead" as a noun vs. verb), and prep data for tasks like translation or sentiment analysis.
- **How It's Done:** Algorithms (often trained models) use context and rules to assign tags from a tagset, like the Penn Treebank (common in English NLP).

Penn Treebank Tagset (Examples)

Here's a quick look at some common POS tags you'll see:

- **NN:** Noun, singular (e.g., "cat")
- **NNS:** Noun, plural (e.g., "cats")
- **VB:** Verb, base form (e.g., "run")
- **VBD:** Verb, past tense (e.g., "ran")
- **JJ:** Adjective (e.g., "quick")
- **DT:** Determiner (e.g., "the")
- **IN:** Preposition (e.g., "on")
- Full list: ~36 tags, but these are the biggies.

Examples of POS Tagging

Let's see it in action with some sentences:

1. **Sentence:** "The quick brown fox jumps the lazy dog."
    - **Tagged:**
        - The/DT quick/JJ brown/JJ fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN
    - **Breakdown:**
        - "The" (DT) sets up nouns, "quick" and "brown" (JJ) describe "fox" (NN), "jumps" (VBZ) is the action, etc.
2. **Sentence:** "She runs fast."
    - **Tagged:**
        - She/PRP runs/VBZ fast/RB
    - **Breakdown:**

        - “She” (PRP) is a pronoun, “runs” (VBZ) is a present-tense verb, “fast” (RB) is an adverb.
3. **Sentence:** “I saw a saw.”
    - **Tagged:**
        - I/PRP saw/VBD a/DT saw/NN
    - **Breakdown:**
        - “saw” as VBD (past tense of “see”) vs. NN (noun, the tool)—context decides!

How It Works (Simplified)

- **Manual Rules:** Early systems used grammar rules (e.g., “DT before NN”), but they struggled with ambiguity.
- **Modern Approach:** Machine learning models (e.g., Hidden Markov Models, neural nets) trained on tagged datasets (like Penn Treebank) predict tags based on word patterns and context.
- **Tools:** Libraries like NLTK, spaCy, or Stanford NLP handle this for us—no need to reinvent the wheel!

Tagging is difficult because some words can represent more than one part of speech at different times. They are **Ambiguous**. Let's look at the following example:

- The whole team played **well**. [adverb]
- You are doing **well** for yourself. [adjective]
- **Well**, this assignment took me forever to complete. [interjection]
- The **well** is dry. [noun]
- Tears were beginning to **well** in her eyes. [verb]

Transition counts

- The first dictionary is the `transition_counts` dictionary which computes the number of times each tag happened next to another tag.

This dictionary will be used to compute

Transition counts

- The first dictionary is the `transition_counts` dictionary which computes the number of times each tag happened next to another tag.

This dictionary will be used to compute:

$$P(t_i|t_{i-1})$$

This is the probability of a tag at position i given the tag at position $i-1$.

In order for you to compute equation 1, you will create a `transition_counts` dictionary where

- The keys are `(prev_tag, tag)`
- The values are the number of times those two tags appeared in that order.

Emission counts

The second dictionary you will compute is the `emission_counts` dictionary. This dictionary will be used to compute:

$$P(w_i|t_i)$$

In other words, you will use it to compute the probability of a word given its tag.

In order for you to compute equation 2, you will create an `emission_counts` dictionary where

- The keys are `(tag, word)`
- The values are the number of times that pair showed up in your training set.

```
transition examples:
(('--s--', 'IN'), 5050)
(('IN', 'DT'), 32364)
(('DT', 'NNP'), 9044)

emission examples:
(('DT', 'any'), 721)
(('NN', 'decrease'), 7)
(('NN', 'insider-trading'), 5)

ambiguous word example:
('RB', 'back') 304
('VB', 'back') 20
('RP', 'back') 84
('JJ', 'back') 25
('NN', 'back') 29
('VBP', 'back') 4
```

Markov chains

A **Markov Chain** is a mathematical model for systems that transition between states over time, where the next state depends *only* on the current state—not the past. It's all about memoryless predictions, called the **Markov Property**.

Key Ideas

- **States:** Distinct conditions (e.g., "sunny," "rainy" for weather).
- **Transitions:** Probabilities of moving from one state to another (e.g., sunny → rainy: 0.3).
- **No Memory:** What happened two steps ago doesn't matter—only where you are now counts.

Example: Weather Prediction

- **States:** Sunny (S), Rainy (R).
- **Transition Probabilities:**
    - S → S: 0.8 (80% chance it stays sunny)
    - S → R: 0.2 (20% chance it turns rainy)
    - R → S: 0.4 (40% chance rain clears up)
    - R → R: 0.6 (60% chance it stays rainy)

| | S | R |

|---|---|---|

| S | 0.8 | 0.2 |

| R | 0.4 | 0.6 |

Scenario: It's sunny today (Day 1). What's tomorrow (Day 2)?

- 80% chance sunny, 20% rainy—simple roll of the dice based on current state.

Why not learn something ?

adverb adverb verb noun punctuation mark, sentence closer

Part of speech tags:

lexical term	tag	example
noun	NN	something, nothing
verb	VB	learn, study
determiner	DT	the, a
w-adverb	WRB	why, where

Why not learn something ?

WRB RB VB NN .

You can use Markov chains to identify the probability of the next word. For example below, you can see that the most likely word after a verb is a noun.

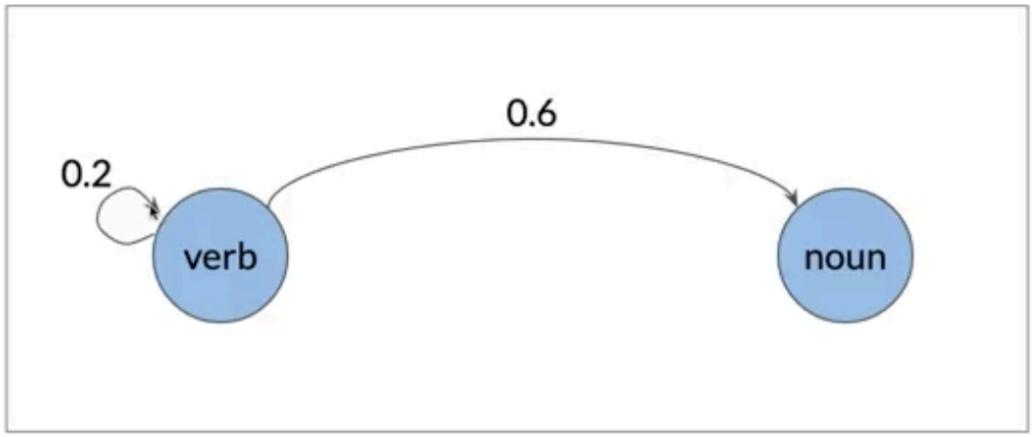
Why not learn swimming?

verb noun

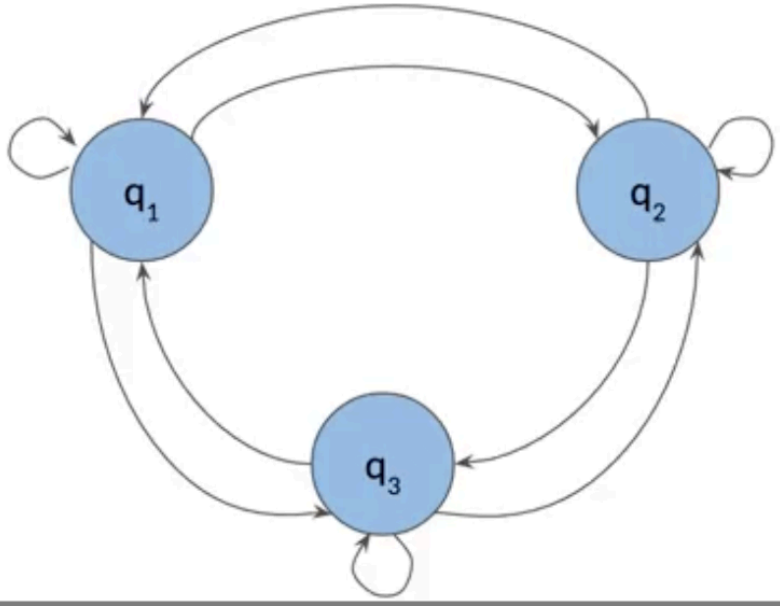
Why not learn swim?

verb verb

To properly model the probabilities we need to identify the probabilities of the POS tags and for the words.



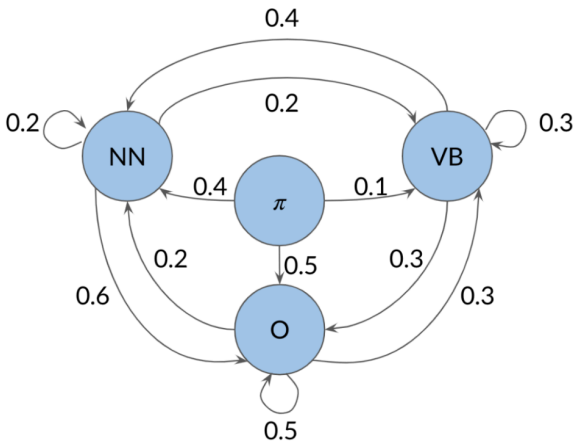
Markov Chains



$$Q = \{q_1, q_2, q_3\}$$

The circles of the graph represent the states of your model. A state refers to a certain condition of the present moment. You can think of these as the POS tags of the current word.

Transition prob

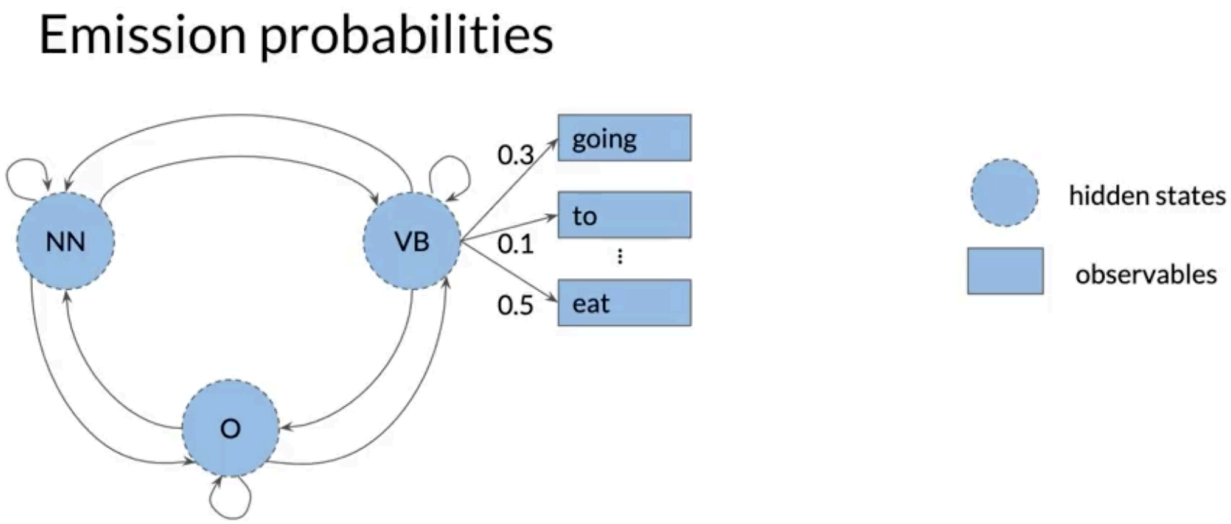


$A =$

	NN	VB	O
π (initial)	0.4	0.1	0.5
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

In the diagram above, the blue circles correspond to the part of speech tags, and the arrows correspond to the transition probabilities from one part of speech to another. You can populate the table on the right from the diagram on the left. The first row in your A matrix corresponds to the initial distribution among all the states. According to the table, the sentence has a 40% chance to start as a noun, 10% chance to start with a verb, and a 50% chance to start with another part of the speech tag.

Emission prob



$B =$

	going	to	eat	...
NN (noun)	0.5	0.1	0.02	
VB (verb)	0.3	0.1	0.5	
O (other)	0.3	0.5	0.68	

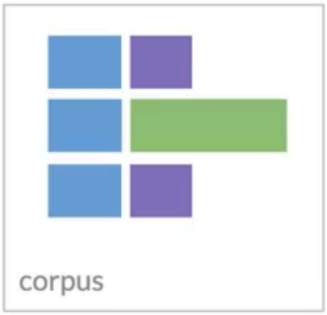
$$\sum_{j=1}^{V} b_{ij} = 1$$

The Hidden Markov model also has additional probabilities known as emission probabilities. These describe the transition from the hidden states of your Hidden Markov model, which are parts of speech seen here as circles for noun, verb, and the other, to the observables or the words of your corpus shown here inside rectangles. Here, for example, are the observables for the hidden states VB, which are the words; going, to, eat. The emission probability from the hidden states, verb to the observable, eat, is 0.5. This means when the model is currently at the hidden state for a verb, there is a 50 percent chance that the observable the model will emit is the word, eat.

States — Transition matrix — Emission matrix

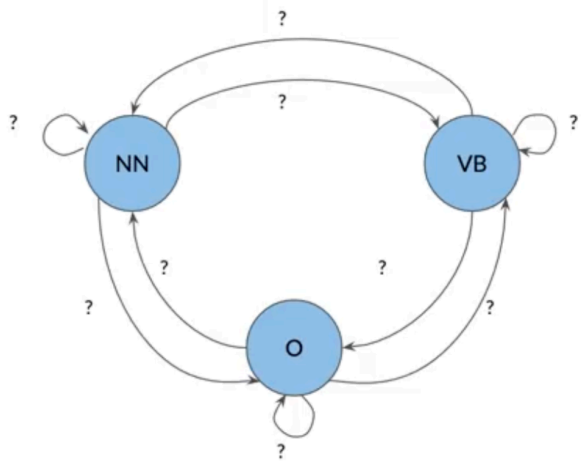
$$Q = \{q_1, \ldots, q_N\} \quad A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \cdots & a_{N+1,N} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & \cdots & b_{1V} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NV} \end{pmatrix}$$

Calculate Transition Prob?



transition probability: ■ + ■ = ⅔

Transition probabilities



1. Count occurrences of tag pairs

$$C(t_{i-1}, t_i)$$

2. Calculate probabilities using the counts

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)}$$

Prepare a corpus

Populating the transition matrix

$A =$

	NN	VB	O
π	1		
NN (noun)	0		
VB (verb)	0		
O (other)	6		

<s> in a station of the metro
<s> the apparition of these faces in the crowd :
<s> petals on a wet , black bough .

Ezra Pound – 1913

$A =$

	NN	VB	O
π	1	0	2
NN (noun)	0	0	6
VB (verb)	0	0	0
O (other)	6	0	

<s> in a station of the metro
<s> the apparition of these faces in the crowd :
<s> petals on a wet , black bough .

Ezra Pound – 1913

$A =$

	NN	VB	O
π	1	0	2
NN (noun)	0	0	6
VB (verb)	0	0	0
O (other)	6	0	8

<s> in a station of the metro
<s> the apparition of these faces in the crowd :
<s> petals on a wet , black bough .

Ezra Pound – 1913

In the table above, you can see that green corresponds to nouns (NN), purple corresponds to verbs (VB), and blue corresponds to other (O). Orange (π) corresponds to the initial state. The numbers inside the matrix correspond to the number of times a part of speech tag shows up right after another one.

To go from O to NN or in other words to calculate $P(NN|O)$ you have to compute the following:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)}$$

$A =$

	NN	VB	O	
π	1	0	2	3
NN	0	0	6	6
VB	0	0	0	0
O	6	0	8	14

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)}$$

$A =$

	NN	VB	O	
π	1+ε	0+ε	2+ε	3+3*ε
NN	0+ε	0+ε	6+ε	6+3*ε
VB	0+ε	0+ε	0+ε	0+3*ε
O	6+ε	0+ε	8+ε	14+3*ε

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \epsilon}{\sum_{j=1}^{N} C(t_{i-1}, t_j) + N * \epsilon}$$

Viterbi Algorithm

https://www.youtube.com/watch?v=IqXdjdOgXPM

What's the Viterbi Algorithm?

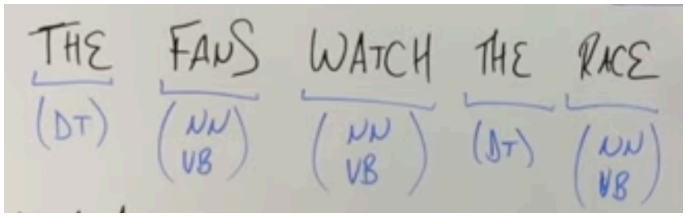
- **Goal:** Find the most probable sequence of hidden states in an HMM given observations.
- **How:** Uses dynamic programming to avoid brute-forcing all possible state sequences—computes the best path efficiently.
- **Context:** For POS tagging, states are tags (e.g., NN, VB), observations are words (e.g., "cat," "runs").

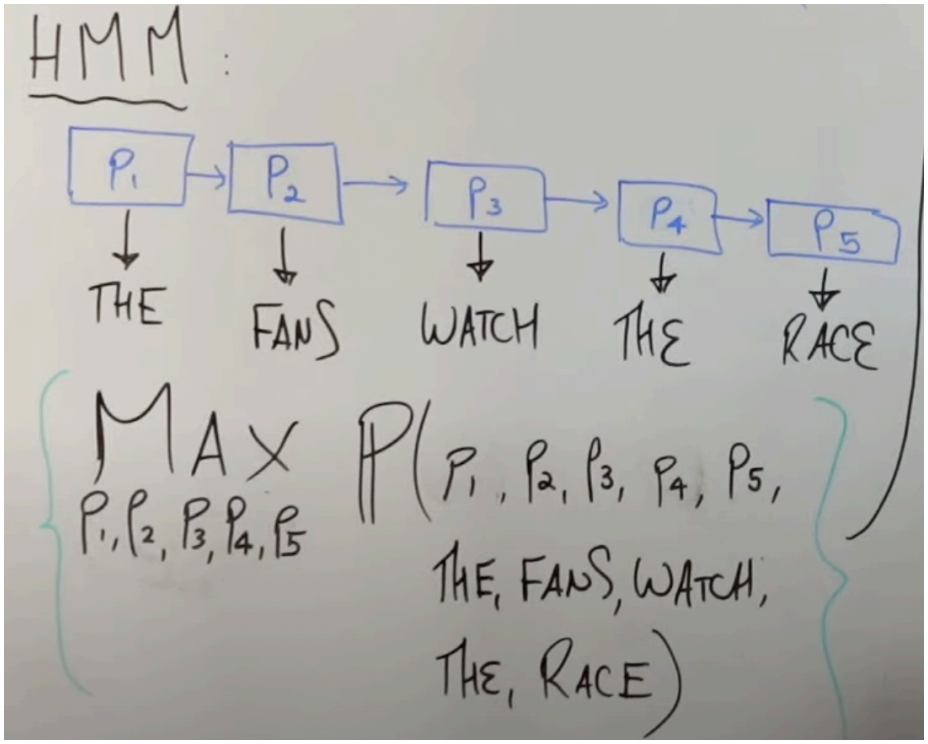
HMM Components

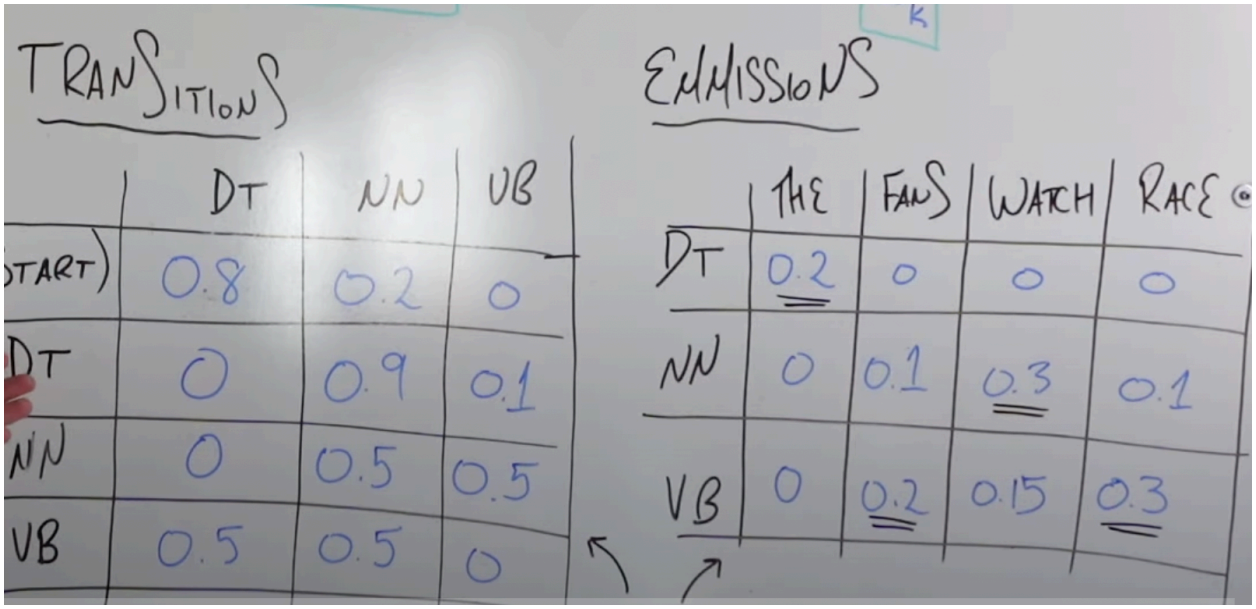
1. **States:** Hidden tags (e.g., NN, VB, DT).
2. **Observations:** Words in a sentence.
3. **Probabilities:**
   - **Transition:** $P(s_i|s_{i-1})$ —probability of moving between states (e.g., DT → NN).
   - **Emission:** $P(w_i|s_i)$ —probability of a word given a state (e.g., "cat" | NN).
   - **Initial:** $P(s_1)$ —probability of starting in a state.

Viterbi Steps

1. **Initialization:** Set up probabilities for the first observation.
2. **Recursion:** Compute the best path to each state at each step, tracking backpointers.
3. **Termination:** Pick the highest-probability final state.
4. **Backtracking:** Trace back to get the state sequence.







Applications

1. Speech Recognition:

- **Direct Application of HMM:** Models the process of spoken words. Hidden states represent the actual sounds (phonemes) being uttered, which we don't directly observe. The observed data is the acoustic signal captured by a microphone.
- **How Viterbi is Used:** Given a sequence of acoustic signals, the Viterbi algorithm finds the most likely sequence of phonemes that produced those sounds, thus helping to determine the spoken words.

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

- **Direct Application of HMM:** Helps determine the grammatical category of each word in a sentence. Hidden states represent the POS tags (like noun, verb, adjective), which are not explicitly given. The observed data is the sequence of words in the sentence.
- **How Viterbi is Used:** For a given sentence, the Viterbi algorithm finds the most likely sequence of POS tags that would have generated that sentence, based on the probabilities of tag sequences and words being associated with tags.

3. Named Entity Recognition (NER):

- **Direct Application of HMM:** Identifies and categorizes named entities (like people, organizations, locations) in text. Hidden states represent the entity types (or "not an entity"), which are not directly labeled. The observed data is the sequence of words in the text.
- **How Viterbi is Used:** Given a sentence, the Viterbi algorithm finds the most likely sequence of entity tags that corresponds to the words in the sentence, thus identifying and classifying the named entities.

Limitations

Scenario:

We want to tag the following sentence:
"She saw a man with a telescope."

Understanding the Two Interpretations:

1. **First Interpretation (Man has the Telescope):**
    - **"She saw [a man with a telescope]."**
    - Here, **"with a telescope"** is describing **the man**.

    - This means that **the man** is the one **holding or carrying the telescope**.
    - In this case, **"with"** is directly linked to **"man"**, forming a noun phrase **"a man with a telescope"**.

2. **Second Interpretation (She used the Telescope):**
    - **"[She saw a man] with a telescope."**
    - Here, **"with a telescope"** is describing **how she saw the man**.
    - This means that **she used a telescope** to **look at the man**.
    - In this case, **"with"** is not related to **"man"** but rather to the **verb "saw"** – it tells **how** she performed the action of **seeing**.

HMM Tagging issues:

- HMM **fails to link "with" to the verb "saw"** because it does not look at the **whole sentence context**.
- It tags **"with"** as a **preposition** linked to **"man"**, even when the correct interpretation is that **"with"** indicates the **manner of seeing**.

Models like **BiLSTM** and **Transformers** do not have this limitation. They can **understand the entire sentence context** and correctly associate **"with"** either with **"man"** or **"saw"** based on the **semantic structure** of the sentence.

1. Markov Assumption Limitation:

HMMs assume that the POS tag of the current word depends only on the previous word's tag. However, in the sentence:

- The word **"bank"** could be tagged as a **noun** (financial institution) or **verb** (to bank on something).
- The correct tag depends on the context provided by the **preceding word "The"** (indicating a noun).
- However, HMM might incorrectly predict the tag if the previous word was not considered correctly or if it does not consider longer dependencies.

2. Emission Independence Limitation:

HMMs assume that the observed word only depends on the current hidden state (POS tag).

- In our sentence, **"close"** can be a **verb** (to shut) or an **adjective** (near).
- The correct interpretation depends on the context (e.g., the presence of a modal verb "will"), but HMM does not inherently capture this relationship.

3. Limited Contextual Understanding:

HMM struggles when the correct tag depends on **long-range context**.

- In a sentence like **"He decided to bank on his skills."**, the word **"bank"** is a verb.
- The correct interpretation requires understanding the phrase **"bank on"**, but HMM would only look at the previous word and may miss this multi-word expression.

4. Inflexibility with Complex Features:

Modern deep learning models use **word embeddings** to capture semantic meaning.

- The words **"bank"** (financial institution) and **"riverbank"** are semantically different but appear similar to HMMs, which use simple probability distributions rather than embeddings.

5. Scalability Issues:

If we have a large corpus with many POS tags, the number of states (NNN) becomes large, increasing the computational complexity of the **Viterbi algorithm**.

- In real-time applications like speech recognition, this quadratic complexity can cause latency, making HMM impractical for fast processing.

6. Data Sparsity Issue:

If the training data does not contain enough instances of the word **"bank"** used as a verb, the HMM might not learn this correctly, leading to poor tagging accuracy.

7. Over-simplistic Assumptions:

HMMs assume that transition and emission probabilities remain constant. However:

- In **conversational speech**, the way words are pronounced may change based on the speaker's mood or the context, but HMM treats them as static probabilities.

Modern Alternatives:

To overcome these limitations, models like **CRFs**, **LSTMs**, and **Transformers** are used:

- **LSTMs** can remember long-range dependencies, making them suitable for sentences with complex contexts.

- **Transformers** use attention mechanisms to capture relationships between distant words effectively.

- **CRFs** do not assume independence between states, making them more flexible for structured predictions.

Advantages of N-gram and HMM Models:

- **Simplicity and Ease of Understanding:** N-gram models are conceptually straightforward and relatively easy to implement. The core idea of counting word sequences is intuitive.
- **Computational Efficiency (for smaller n):** Training and using n-gram models, especially for small values of *n* (like bigrams or trigrams), can be computationally efficient, especially with large datasets. Counting word sequences is a relatively fast process.
- **Effectiveness for Local Dependencies:** N-grams excel at capturing short-range dependencies between words. They are good at predicting the next word based on the immediate preceding words, which is useful for tasks like:
    - **Language Modeling:** Predicting the next word in a sequence (e.g., in autocomplete or predictive text).
    - **Speech Recognition:** Helping to disambiguate words based on the surrounding context.
    - **Spelling Correction:** Identifying likely correct words based on common word sequences.
- **Direct Modeling of Observed Sequences:** N-grams directly model the probability of sequences of observed words, which can be beneficial when the task primarily relies on the surface-level patterns of the language.

Advantages of Hidden Markov Models (HMMs):

- **Modeling Hidden Structure:** HMMs can model underlying, unobserved structures (hidden states) that influence the observed sequence. This is crucial for tasks where there's a latent process at play, such as:
    - **Part-of-Speech (POS) Tagging:** Inferring the grammatical category of each word, which is not directly observed.
    - **Speech Recognition:** Modeling the underlying phonemes that generate the acoustic signal.
    - **Named Entity Recognition (NER):** Identifying and categorizing named entities based on the surrounding words and context.

Code:
https://github.com/amanjeetsahu/Natural-Language-Processing-Specialization/blob/master/Natural%20Language%20Processing%20with%20Probabilistic%20Models/Week%202/C2_W2_Assignment_Solution.ipynb