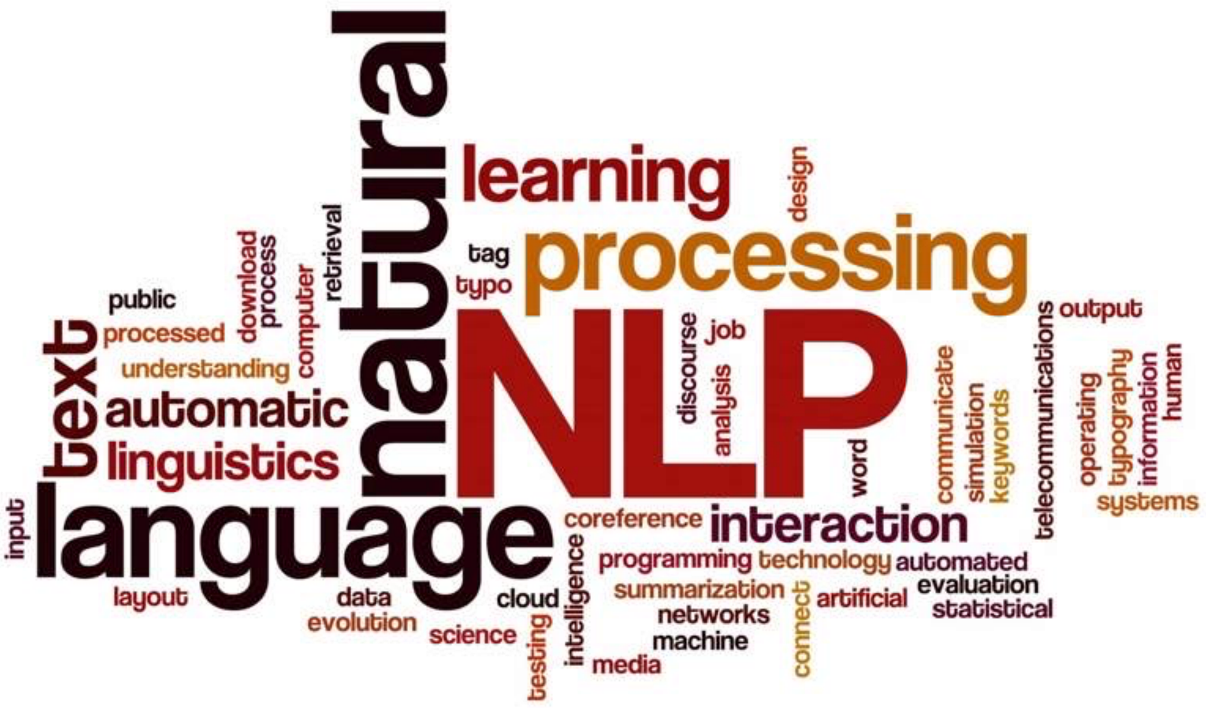

NLP problems like Part of Speech Tagging using Hidden Markov Models

Markov Chains Assignment | Prepared By: Asma Gite (D021), Moksha Gandhi (D019)

Natural Language Processing - Overview

What is Natural Language Processing (NLP)?

Natural Language Processing (NLP) is a branch of **Artificial Intelligence (AI)** that enables computers to understand, interpret, and generate human language. It is used in applications like **chatbots, voice assistants, machine translation, and text analysis** to help machines process and respond to text or speech naturally. **NLP combines linguistics and machine learning to analyze text, recognize speech, and extract meaningful information.**

What is Part of Speech (POS) Tagging?

Part of speech tagging (POS tagging) is a natural language processing (NLP) technique that involves assigning grammatical categories or “parts of speech” to each word in a given text or sentence.

This process of of labeling words in a sentence with their corresponding parts of speech, such as **noun, verb, adjective, adverb, preposition etc,** helps machines understand the structure and meaning of a sentence.

Example:

👉 The cat sleeps on the mat.

A POS tagger assigns:

- **The** → Determiner (DT)
- **cat** → Noun (NN)
- **sleeps** → Verb (VBZ)
- **on** → Preposition (IN)
- **the** → Determiner (DT)
- **mat** → Noun (NN)

Why do we need POS Tagging?

POS Tagging has several applications and immense importance in today's world.

1. **Improves Machine Understanding of Language** – Helps AI and NLP models grasp sentence structure, making text processing more accurate.
2. **Enhances Sentence Parsing** – Aids in syntactic and grammatical analysis, allowing better interpretation of word relationships.
3. **Enables Contextual Meaning Extraction** – Differentiates between words with multiple meanings (e.g., "book" as a noun vs. "book" as a verb).
4. **Boosts NLP Applications** – Essential for machine translation, speech recognition, chatbots, and search engines.
5. **Supports Feature Engineering for ML Models** – Provides crucial linguistic features for text classification, sentiment analysis, and entity recognition.
6. **Improves Information Retrieval & Search Engines** – Helps in understanding query intent by recognizing key terms and their roles.
7. **Facilitates Grammar & Writing Tools** – Powers grammar checkers and text correction systems to improve writing accuracy.

Hidden Markov Model - Overview

What is a Hidden Markov Model?

A **Hidden Markov Model (HMM)** is a **probabilistic model** used to predict a sequence of hidden states based on observed data.

HMM predicts the most likely sequence of hidden states (POS tags, spoken words, etc.) based on the given observations using algorithms like the **Viterbi Algorithm**.

Key Concepts of Hidden Markov Model (HMM) with a Simple Example

Let's imagine a simple **weather prediction** example where you can't see the actual weather but can only observe a friend's daily activity.

1. States (Hidden Layer)

The **hidden states** are what we want to predict but cannot directly observe.

🔷 In this case, the possible states are:

- **Sunny**
- **Rainy**

---

2. Observations (What We See)

The **observations** are the clues that help us guess the hidden states.

🔷 Your friend's activities depend on the weather:

- **Walking** → More likely on a **sunny** day
- **Watching TV** → More likely on a **rainy** day

---

3. Transition Probability (Weather Change Likelihood)

This tells us how likely the weather will change from one day to another.

Example probabilities:

- If today is **Sunny**, there's a **70% chance** tomorrow will also be **Sunny**, but a **30% chance** it will be **Rainy**.
- If today is **Rainy**, there's a **60% chance** tomorrow will still be **Rainy**, but a **40% chance** it will be **Sunny**.

---

4. Emission Probability (Activity Given the Weather)

This tells us how likely an observation (activity) happens given a particular state (weather).

Example probabilities:

- On a **Sunny** day:
  - 80% chance the friend will **walk**
  - 20% chance they will **watch TV**
- On a **Rainy** day:
  - 90% chance they will **watch TV**
  - 10% chance they will **walk**

---

5. Initial Probability (Starting Weather)

If we don’t know the weather, we assume a probability for the first day:

- **50% chance** it’s **Sunny**
- **50% chance** it’s **Rainy**

The **Hidden Markov Model (HMM)** helps determine hidden states (e.g., weather) based on observed data (e.g., activities) using the **Viterbi Algorithm**..

Given the sequence **[Walking, Watching TV, Walking]**, HMM uses **transition and emission probabilities** to find the most likely weather sequence:

✅ **Day 1 (Walking)** → **Sunny** (Since walking is more likely on a sunny day)

✅ **Day 2 (Watching TV)** → **Rainy** (Since watching TV is more common on a rainy day)

✅ **Day 3 (Walking)** → **Sunny** (Since walking is again more likely on a sunny day)

Thus, HMM predicts the **hidden weather states** as:

👉 **Sunny** → **Rainy** → **Sunny**

This shows how **observations (activities) help determine hidden states (weather)** using probability-based calculations.

Application of HMM in POS Tagging

In **POS tagging**, the **words in a sentence are observations**, and the **hidden states are their respective parts of speech (noun, verb, etc.)**. HMM helps predict the most likely sequence of POS tags for a given sentence using **transition** and **emission probabilities**.

The central idea behind HMMs is to model a sequence of observations as a sequence of hidden states, where the transitions between states and the emissions of observations are governed by probabilistic models.

Problem Satement

Understanding the grammatical structure of a sentence is fundamental in **Natural Language Processing (NLP)** for tasks such as speech recognition, machine translation, and information retrieval. One of the key challenges in NLP is accurately **tagging words with their respective parts of speech (POS)**—such as nouns, verbs, and adjectives—especially when dealing with ambiguous words or context-dependent meanings.

For example, consider the sentence:

👉 **Time flies like an arrow.**

- The word **"flies"** can be either a **noun** (insects) or a **verb** (third-person singular of "fly").
- The word **"like"** can function as a **verb** ("to like something") or a **preposition** ("similar to").

Without accurate POS tagging, a system may misinterpret the meaning of the sentence, leading to incorrect language understanding. To address this challenge, our project focuses on developing a **POS tagging system using Hidden Markov Models (HMMs)**. Our model learns from a labeled dataset, utilizing **transition and emission probabilities** to determine the most likely sequence of POS tags for a given sentence. To decode the best possible tag sequence, we employ the **Viterbi Algorithm**, ensuring accurate word categorization even in ambiguous cases.

How to tackle this?

📌 Step 1: Define the Components of HMM

A **Hidden Markov Model (HMM)** consists of the following components:

- **States (S):** The possible POS tags (e.g., Noun, Verb, Preposition, etc.).
- **Observations (O):** The words in the sentence **["Time", "flies", "like", "an", "arrow"]**.
- **Transition Probabilities (A):** The probability of a POS tag following another POS tag.
- **Emission Probabilities (B):** The probability of a word being associated with a certain POS tag.
- **Initial Probabilities (π):** The probability of a POS tag being the first in a sentence.

📌 Step 2: Assign Possible POS Tags

Each word in the sentence can have multiple possible POS tags:

Word	**Possible POS Tags**
Time	Noun, Verb
flies	Noun, Verb
like	Verb, Preposition
an	Determiner
arrow	Noun

📌 Step 3: Compute Transition & Emission Probabilities

To determine the most likely POS sequence, we use probabilities derived from a labeled POS dataset, such as the **Penn Treebank**.

Transition Probabilities (A):

- P(Noun → Verb) = 0.3
- P(Verb → Preposition) = 0.5
- P(Determiner → Noun) = 0.9
- P(Noun → Noun) = 0.1

Emission Probabilities (B):

- P("Time" | Noun) = 0.6
- P("flies" | Verb) = 0.8
- P("like" | Preposition) = 0.7
- P("an" | Determiner) = 1.0
- P("arrow" | Noun) = 0.9

📌 Step 4: Apply the Viterbi Algorithm

The **Viterbi Algorithm** is used to find the most probable sequence of POS tags for the given sentence. The algorithm follows these steps:

1. **Initialization:** Assign probabilities to the first word, "Time."
2. **Recursion:** For each word, compute the highest probability sequence based on transition & emission probabilities.
3. **Backtrace:** Retrieve the best sequence of tags.

📌 Step 5: Determine the Best POS Sequence

After applying the Viterbi Algorithm, we obtain the most probable POS tagging:

- **Time/Noun**
- **flies/Verb**
- **like/Preposition**
- **an/Determiner**
- **arrow/Noun**

This correctly interprets the sentence as:

"Time" is a **Noun** (not a verb).

"flies" is a **Verb** (not a noun referring to insects).

"like" is a **Preposition** (not a verb like "I like apples").

> To measure accuracy, we can compare predicted tags with true tags from a labeled dataset using a **Confusion Matrix**.

Python Implementation

📎 Steps in the Implementation

1. Define the **states (POS tags)** and **observations (words in a sentence).**
2. Define the **transition probabilities, emission probabilities,** and **initial probabilities.**

3. Implement the **Viterbi Algorithm** to find the **most probable sequence** of POS tags for the given sentence.

🖇 Codes

```
import numpy as np

# Define the states (POS tags)
states = ["Noun", "Verb", "Preposition", "Determiner"]

# Define the words (observations)
observations = ["Time", "flies", "like", "an", "arrow"]

# Transition probabilities (probability of POS tag following another)
transition_prob = {
    "Noun": {"Noun": 0.1, "Verb": 0.3, "Preposition": 0.1, "Determiner": 0.0},
    "Verb": {"Noun": 0.4, "Verb": 0.2, "Preposition": 0.5, "Determiner": 0.0},
    "Preposition": {"Noun": 0.7, "Verb": 0.0, "Preposition": 0.1, "Determiner": 0.2},
    "Determiner": {"Noun": 0.9, "Verb": 0.0, "Preposition": 0.0, "Determiner": 0.0},
}

# Emission probabilities (probability of word being a certain POS tag)
emission_prob = {
    "Noun": {"Time": 0.6, "flies": 0.2, "like": 0.1, "an": 0.0, "arrow": 0.9},
    "Verb": {"Time": 0.1, "flies": 0.8, "like": 0.2, "an": 0.0, "arrow": 0.1},
    "Preposition": {"Time": 0.0, "flies": 0.0, "like": 0.7, "an": 0.0, "arrow": 0.0},
    "Determiner": {"Time": 0.0, "flies": 0.0, "like": 0.0, "an": 1.0, "arrow": 0.0},
}

# Initial probabilities (probability of a POS tag being the first word)
initial_prob = {"Noun": 0.5, "Verb": 0.2, "Preposition": 0.1, "Determiner": 0.2}

# Viterbi Algorithm Implementation
def viterbi_algorithm(observations, states, initial_prob, transition_prob,
emission_prob):
    n = len(observations)
    m = len(states)

    # Initialize the DP table and backpointer
    viterbi = np.zeros((m, n))
    backpointer = np.zeros((m, n), dtype=int)

    # Initialize the first column (starting probabilities)
    for i, state in enumerate(states):
        viterbi[i, 0] = initial_prob[state] * emission_prob[state].get(observations[0],
0)

    # Fill the DP table
    for t in range(1, n):
        for i, state in enumerate(states):
            max_prob, max_state = max(
                (viterbi[j, t - 1] * transition_prob[prev_state].get(state, 0) *
emission_prob[state].get(observations[t], 0), j)
                for j, prev_state in enumerate(states)
            )
            viterbi[i, t] = max_prob
            backpointer[i, t] = max_state

    # Backtrace to find the best path
    best_last_state = np.argmax(viterbi[:, n - 1])
    best_path = [best_last_state]

    for t in range(n - 1, 0, -1):
        best_last_state = backpointer[best_last_state, t]
        best_path.insert(0, best_last_state)

    # Convert indices to state names
    best_path_states = [states[i] for i in best_path]

    return best_path_states

# Run the Viterbi Algorithm
best_tags = viterbi_algorithm(observations, states, initial_prob, transition_prob,
emission_prob)

# Print the results
print("Sentence:", " ".join(observations))
print("POS Tags:", " ".join(best_tags))
```

Output:

```
Sentence: Time flies like an arrow POS
Tags: Noun Verb Preposition Determiner Noun
```

Conclusion

In this project, we successfully implemented **Part-of-Speech (POS) Tagging** using **Hidden Markov Models (HMMs)**. We explored the fundamental components of HMMs, including **states (POS tags), observations (words), transition probabilities, emission probabilities, and initial probabilities**.

Through the example sentence **"Time flies like an arrow."**, we demonstrated how **POS tagging can resolve ambiguities** in language. By applying the **Viterbi Algorithm**, our model effectively predicted the correct sequence of POS tags, ensuring **contextually accurate** interpretations.

This project provides a **strong foundation** for further advancements, such as integrating **Deep Learning-based POS tagging models (e.g., LSTMs, CRFs)** to improve accuracy. The HMM-based approach remains a **robust and interpretable** technique, making it valuable for various **NLP applications** like **chatbots, machine translation, and search engines**.

References

Part of Speech (POS) tagging with Hidden Markov Model

POS or Part of Speech tagging is a task of labeling each word in a sentence with an appropriate part of...

Author **Great Learning Editorial Team**

Part Of Speech tagging and Hidden Markov Models

Introduction

Publisher **Medium** Author **Arnav Goel** Published **9/6/2023**

Google Colab

https://colab.research.google.com/drive/15HUeqMUEbKBDv4RhMJS67H0H1iaXWG_S?usp=sharing#scrollTo=Xd1fxoU-wT0v