Parts of Speech Tagging Using Hidden Markov Models

Boda Surya Venkata Jyothi Sowmya
Data Science
Chennai Mathematical Institute

M.R. Srinivasan
Data Science
Chennai Mathematical Institute

Abstract— **Part-of-Speech (POS) tagging is an essential task in Natural Language Processing because it helps identify the grammatical role of each word in a sentence. In this project, we build a POS tagging system using a Hidden Markov Model (HMM), which relies on probabilities learned from labeled training data. From this data, we create two important components: the transition matrix and the emission matrix. The transition matrix represents how likely one tag is to follow another, which helps the model understand the flow of grammar in a sentence. The emission matrix represents how likely a particular word is to appear with a specific tag, helping the model connect words to their correct grammatical roles.**
Once these matrices are prepared, we use the Viterbi algorithm to tag new sentences. Viterbi works by combining transition and emission probabilities at each step to calculate the most likely sequence of tags. Instead of checking every possible tag combination—which would be very slow—the algorithm uses dynamic programming to efficiently track the best path. This makes the system fast and practical even for longer sentences. The project also includes an effective method for handling unknown words that do not appear in the training data. Simple rules based on word structure, punctuation patterns, digits, and common suffixes help classify unseen words into meaningful unknown-word categories. This greatly improves the model's ability to tag real-world text where new or rare words frequently appear.

I. INTRODUCTION

Natural language is highly flexible, and the same word can perform different grammatical roles depending on the context in which it appears. For example, a word that acts as a noun in one sentence may function as a verb in another. Because of such variations, automatically identifying the correct grammatical category of each word becomes an important step in many Natural Language Processing (NLP) tasks. Part-of-Speech (POS) tagging addresses this need by assigning a suitable label—such as noun, verb, adjective, or adverb—to each word in a sentence. Accurate POS tagging makes it easier for computers to understand sentence structure, resolve ambiguity, and process text more effectively.
POS tagging plays a key role in many real-world applications. Machine translation systems use POS tags to produce grammatically correct output. Search engines rely on POS information to improve queries. Chatbots and virtual assistants use tagged text to better interpret user intent. Even simple tasks like grammar checking and sentiment analysis depend on correct POS information. Because of this wide usefulness, building a reliable POS tagger is an important problem in NLP.
Earlier POS tagging systems were mainly rule-based. They depended on hand-crafted grammatical rules created by linguists. While these systems work well for limited text, they often struggle with large datasets, new words, and the wide variety of sentence patterns found in real-world language. To overcome these limitations, statistical models became popular. Among them, the Hidden Markov Model (HMM) is one of the most widely used because it can learn patterns automatically from labeled training data.
In this project, we implement an HMM-based POS tagger. The model learns two important probability tables from the training corpus: transition probabilities, which describe how one tag follows another, and emission probabilities, which describe how likely a word is to appear with a specific tag. To tag new sentences, we use the Viterbi algorithm, which efficiently computes the most likely sequence of POS tags using these probabilities.
A special focus of this project is handling unknown words—words that do not appear in the training dataset. To address this, we use simple rule-based strategies based on word structure, punctuation, digits, and common suffix patterns. These rules help assign unknown words to meaningful categories and significantly improve tagging accuracy.
Overall, this project aims to create a POS tagging system that is easy to understand, interpretable, and effective for general text processing.

II. METHODOLOGY

Our methodology is based on a structured statistical process that uses Hidden Markov Models (HMMs) and the Viterbi algorithm to perform accurate Part-of-Speech tagging. The overall approach contains two key phases: **building the model from training data and predicting tags for new sentences.**

A. Building the Transition and Emission Matrices

We start by processing the annotated training corpus, where every word is already paired with its correct POS tag. From this data, we calculate how tags behave in sequences and how words are associated with tags.

- To build the **transition matrix**, we count how often each tag follows another tag in the corpus. These counts are then converted into probabilities by dividing them by the total frequency of the previous tag. This allows us to understand patterns like how often a determiner

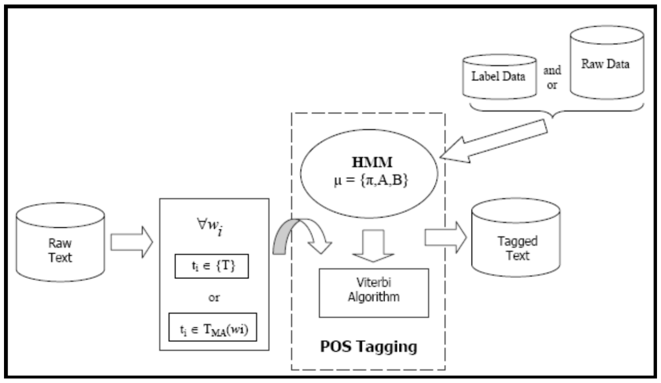

Fig. 1. Workflow / Pipeline for POS Tagging Systems

is followed by a noun or how often a verb is followed by an adverb.

- To build the **emission matrix**, we calculate how often each word appears with each tag. These numbers are normalized to form probabilities. This helps the model understand how likely a particular word is to belong to a given tag.

These probability matrices serve as the foundation for modeling the sequential behavior of tags.

B. Tagging Phase Using Viterbi Algorithm

Once the model is trained, we use the Viterbi algorithm to assign tags to new sentences. Viterbi works step-by-step through the input sentence. For every word, it calculates the probability of each possible tag by combining:

- the probability of transitioning from the previous tag, and
- the probability of emitting the current word under the current tag.

At each step, the algorithm stores not only the highest probability but also the tag path that led to it. This dynamic programming approach ensures that the algorithm never recomputes unnecessary information and always follows the most probable sequence. After processing all words, Viterbi traces back the best path, giving us the final POS tag sequence for the entire sentence.
This methodology ensures that our model stays grounded in statistical evidence from the corpus while remaining efficient for tagging even longer sentences.

III. Dataset Description

The dataset used for this study is the **Penn Treebank Corpus**, a standard benchmark corpus in Natural Language Processing. It consists of sentences annotated with Part-of-Speech (POS) tags according to the Penn Treebank tagset. Each sentence is represented as a sequence of word-tag pairs:

$$S_i = \big[(w_1, t_1), (w_2, t_2), \dots, (w_k, t_k)\big], \quad i = 1, \dots, N$$

where w_j denotes the j-th word in the sentence and t_j denotes its corresponding POS tag.

A. Corpus Statistics

- **Total Sentences:** The corpus contains N sentences, which are divided into training and test sets.
- **Vocabulary:** The unique set of words in the corpus forms the vocabulary $\mathcal{V} = \{v_1, v_2, \dots, v_{|\mathcal{V}|}\}$, where $|\mathcal{V}|$ is the total number of distinct words.
- **POS Tags:** Each word is annotated with one of M distinct POS tags according to the Penn Treebank tagset.
- **Tagged Sequences:** The corpus is naturally structured into sequences of tagged words, allowing computation of transition and emission probabilities for HMM-based tagging.

B. Train-Test Split

To evaluate model performance on unseen data, the corpus is split into training and test sets. Typically, the last 10% of sentences are reserved as the test set:

$$\text{Training set: } S_1, \dots, S_{0.9N}, \quad \text{Test set: } S_{0.9N+1}, \dots, S_N$$

The training set is used to estimate model parameters such as tag transition probabilities and word emission probabilities, while the test set is used for evaluation.

IV. Data Preprocessing

Preprocessing is essential for handling unknown words and maintaining sentence structure. The following transformations are applied:

A. Unknown Word Handling

Words that do not occur in the training vocabulary are replaced with special *unknown-word classes* based on morphological, lexical, and orthographic rules. These classes include:

- Digit-containing tokens
- Punctuation tokens
- Capitalized or upper-case words
- Words with likely noun, verb, adjective, or adverb suffixes
- A default unknown token for all other cases

Formally, for a word $w \notin \mathcal{V}$, we assign:

$$w' = f_{\text{unk}}(w)$$

where f_{unk} maps unknown words to their appropriate unknown-word class.

B. Sentence Flattening and Boundary Markers

To facilitate sequential processing in the HMM, sentences are transformed into a flat sequence of tokens, and sentence boundaries are explicitly marked using a special token:

$$S'_i = [w'_1, w'_2, \dots, w'_k, \texttt{-n-}]$$

where w'_j is either the original word (if in vocabulary) or its assigned unknown-word class. The token `-n-` indicates the end of a sentence.

C. Final Preprocessed Corpus

The preprocessed corpus consists of a continuous sequence of tokens representing all sentences, with unknown words replaced and sentence boundaries marked. This structure is used for computing the HMM emission and transition probabilities and for subsequent POS tagging using the Viterbi algorithm.

V. Baseline POS Prediction

After preprocessing the corpus, a baseline POS tagging model is constructed to evaluate the effectiveness of simple emission-based predictions. The baseline model assigns to each word the POS tag that maximizes the emission probability, ignoring context and transition information.

A. Parameter Estimation

Using the training set, the following frequency counts are computed:

- **Emission Counts:** For each tag t and word w, count the number of occurrences where t emits w:

$$C_{\text{emission}}(t, w) = \#\Big\{(w_i, t_i) \in \text{training corpus} : w_i = w,\, t_i = t\Big\}$$

- **Transition Counts:** For each tag pair (t_{i-1}, t_i), count the number of occurrences of the transition:

$$C_{\text{transition}}(t_{i-1}, t_i) = \#\Big\{(t_{i-1}, t_i) \text{ consecutive tag pairs in training corpus}\Big\}$$

- **Tag Counts:** For each tag t, count the total occurrences of t in the training corpus:

$$C_{\text{tag}}(t) = \#\{t_i = t \text{ in training corpus}\}$$

These counts are used to compute emission and transition probabilities for the HMM. However, in the baseline model, only emission probabilities are considered for tagging.

B. Baseline Tagging Rule

For each word w in the preprocessed test corpus, the baseline tagger assigns the POS tag t^* that maximizes the emission count:

$$t^* = \arg\max_{t \in \mathcal{T}} C_{\text{emission}}(t, w)$$

where $\mathcal{T}$ is the set of all POS tags. If the word is an unknown token, it is already replaced by the corresponding unknown-word class during preprocessing.

C. Evaluation

The baseline model is evaluated on the preprocessed test corpus. Accuracy is computed as the fraction of words whose predicted tag matches the true tag:

$$\text{Accuracy} = \frac{\text{Number of correctly predicted words}}{\text{Total number of words in test set}}$$

Using this method, the baseline model achieved an accuracy of approximately 86.49%, indicating that emission probabilities alone provide a strong initial prediction for POS tagging. This serves as a reference point for comparing more sophisticated methods, such as the Viterbi algorithm applied with a full Hidden Markov Model.

VI. Hidden Markov Model (HMM)

Hidden Markov Models (HMMs) provide a statistical framework for modeling sequences where the observed data is visible but the underlying states are hidden. In the context of Part-of-Speech (POS) tagging, the **observed words** in a sentence act as the visible outputs, while the **POS tags** form the hidden states that we aim to recover. In this report, the HMM framework is explained with respect to the POS-tagged dataset used in our project, where the transition and emission statistics are derived directly from the corpus. Using this structure, we can compute the most likely tag sequence for any given sentence.

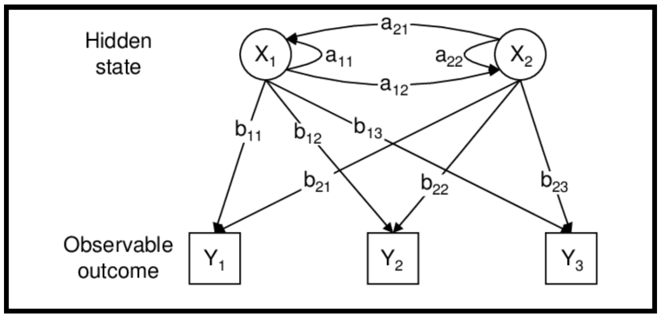

Fig. 2. Flowchart of the Hidden Markov Models

A. Components of an HMM

The HMM consists of the following key components:

- **Hidden States (Q):** Set of POS tags appearing in the dataset, $Q = \{q_1, q_2, \ldots, q_N\}$.
- **Observations (V):** Set of words (vocabulary) extracted from the dataset, $V = \{v_1, v_2, \ldots, v_M\}$.
- **Transition Matrix (A):** An $N \times N$ matrix capturing the probability of transitioning from one tag to another. Each entry A_{ij} represents the probability of tag q_i following tag q_j:

$$A_{ij} = P(t_i \mid t_j)$$

- **Initial Probabilities (π):** Probability of starting in each hidden state, estimated from tag frequencies in the dataset.

- **Emission Matrix (B):** An $M \times N$ matrix capturing the probability of generating a word given a tag:

$$B_{ij} = P(w_i \mid t_j)$$

B. Assumptions of HMM

- **Markov Property:** The next state depends only on the current state.
- **Stationarity:** Transition probabilities remain constant over time.
- **Output Independence:** Observations depend only on the current hidden state.

C. Computing Transition Probabilities

The transition matrix A is computed directly from the POS-tagged dataset used in this project. If t_{i-1} and t_i are two consecutive tags, the transition probability is:

$$P(t_i \mid t_{i-1}) = \frac{\text{Count}(t_{i-1} \to t_i)}{\text{Count}(t_{i-1})}$$

The numerator represents the number of times tag t_i follows t_{i-1} in the dataset, and the denominator represents the total occurrences of t_{i-1}. These values fill the transition matrix A, allowing the model to capture tag-sequence patterns present in our corpus.

D. Computing Emission Probabilities

The emission matrix B is similarly computed using word–tag co-occurrence information from the dataset. For a word w with tag t, the emission probability is:

$$P(w \mid t) = \frac{\text{Count}(t, w)}{\text{Count}(t)}$$

Unknown or rare words in the dataset are handled by grouping them into special "UNK" categories based on morphological and lexical rules, ensuring that the emission matrix accounts for unseen vocabulary.

E. HMM-Based POS Tagging

Given a sentence consisting of words $w_1, w_2, \dots, w_n$, the goal is to determine the most probable tag sequence $t_1, t_2, \dots, t_n$. Using HMM principles, the overall probability of a tag sequence is expressed as:

$$P(t_1, t_2, \dots, t_n \mid w_1, w_2, \dots, w_n) \propto \left(\prod_{i=1}^{n} P(w_i \mid t_i)\right) \left(\prod_{i=2}^{n} P(t_i \mid t_{i-1})\right) \tag{1}$$

This combines the emission probabilities for each word-tag pair with the transition probabilities between successive tags. In our implementation, all these probability values—transition, emission, and initial—are derived from the POS-tagged dataset. To efficiently compute the most likely tag sequence, we apply the **Viterbi algorithm**, which evaluates possible tag paths using these dataset-specific probability estimates and returns the highest-scoring sequence. This results in accurate and statistically grounded POS tag predictions for the sentences in our dataset.

VII. Brute Force Method for POS Tagging

The brute force method is a straightforward approach to Part-of-Speech tagging. It tries **every possible combination of POS tags** for a sentence and selects the sequence with the highest probability according to the model. While conceptually simple, this method is highly inefficient for practical use.

A. How Brute Force Works

Given a sentence of n words and a set of k possible tags, the brute force method proceeds as follows:

1) Generate all possible sequences of POS tags. There are k^n possible sequences for a sentence of length n.
2) For each sequence $t_1, t_2, \dots, t_n$, compute the probability using the HMM model:

$$P(t_1, t_2, \dots, t_n \mid w_1, w_2, \dots, w_n) \propto \left(\prod_{i=1}^{n} P(w_i \mid t_i)\right) \left(\prod_{i=2}^{n} P(t_i \mid t_{i-1})\right) \tag{2}$$

B. Limitations of the Brute Force Method

- **Exponential Growth:** The number of possible tag sequences grows exponentially with sentence length. For example, a sentence of 10 words with 12 possible tags per word has $12^{10} \approx 61$ billion sequences.
- **Computationally Infeasible:** Evaluating all sequences becomes impossible for longer sentences or large corpora.
- **Not Practical for Real NLP Tasks:** Even modern computers cannot handle the computation required for real-world text, making brute force unsuitable for practical POS tagging.

C. How HMM and Viterbi Solve This Problem

Hidden Markov Models, combined with the Viterbi algorithm, provide an efficient solution:

- **Dynamic Programming:** Viterbi computes the most probable tag sequence by reusing intermediate computations, avoiding redundant calculations.
- **Polynomial Time Complexity:** Instead of evaluating k^n sequences, Viterbi runs in $O(n \cdot k^2)$ time, making it feasible for long sentences.
- **Accurate and Efficient:** The algorithm guarantees finding the highest probability sequence while drastically reducing computation time.

This demonstrates why brute force, although simple and intuitive, is impractical for POS tagging, and why HMM with Viterbi is the preferred approach in natural language processing.

VIII. Viterbi Algorithm for POS Tagging

The Viterbi algorithm is a dynamic programming method used to find the **most probable sequence of Part-of-Speech (POS) tags** for a given sentence using a trained Hidden Markov Model (HMM). Unlike brute-force methods, which evaluate all possible tag sequences, Viterbi efficiently computes the optimal sequence by exploiting the **Markov property** and storing intermediate computations.
In this project, I apply the Viterbi algorithm to my dataset after computing the transition and emission probabilities from the tagged corpus. Although the explanation below is general, the same steps were followed when implementing the algorithm for my own POS-tagged dataset.
The algorithm operates in **three main stages**:

1) **Initialization:** Set up the probabilities for the first word in the sentence.
2) **Forward Pass (Recursion):** Iteratively compute the probabilities for each subsequent word based on previous probabilities and transition/emission probabilities.
3) **Backward Pass (Traceback):** Reconstruct the most likely tag sequence using stored best-path pointers.

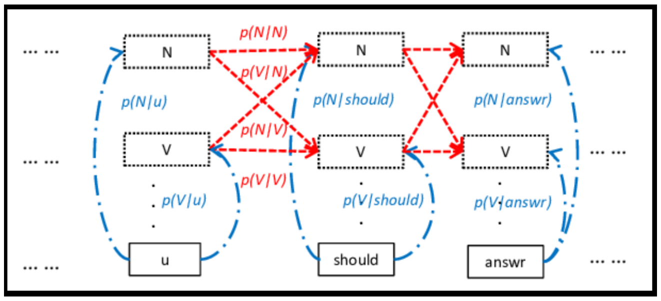

Fig. 3. Flowchart of the Viterbi Algorithm for POS Tagging

A. Notation

Let:

- $O = (o_1, o_2, \ldots, o_T)$ denote the sequence of observed words in a sentence of length T.
- $S = \{s_1, s_2, \ldots, s_N\}$ denote the set of possible POS tags.
- π_i denote the initial probability of tag s_i.
- $A = [P(s_j \mid s_i)]$ denote the tag–tag transition matrix computed from the dataset.
- $B = [P(o_t \mid s_j)]$ denote the emission matrix computed from word–tag counts in my dataset.
- $V[j, t]$ denote the probability of the most likely sequence of tags ending in s_j at position t.
- best_path$[j, t]$ store the best previous tag for reconstruction.

B. Step 1: Initialization

For the first word o_1, the probability of assigning tag s_j is:

$$V[j,1] = \pi_j \cdot B[j,1] = P(s_j \text{ at position } 1) \cdot P(o_1 \mid s_j)$$

In my implementation, these values were computed directly from the initial-tag frequencies and word-tag emission counts of the dataset.

C. Step 2: Forward Pass (Recursion)

For each word o_t $(t = 2, \ldots, T)$:

$$V[j,t] = \max_{i \in \{1,\ldots,N\}} \Big(V[i,t-1] \cdot P(s_j \mid s_i) \cdot P(o_t \mid s_j) \Big)$$

This step combines:

- $V[i, t-1]$ is the probability of the most likely sequence ending with tag s_i at position $t-1$.
- $P(s_j \mid s_i)$ is the transition probability from tag s_i to s_j.
- $P(o_t \mid s_j)$ is the emission probability of the current word given tag s_j.

The algorithm also tracks the best previous tag:

$$\text{best\_path}[j,t] = \arg\max_i \Big(V[i,t-1] \cdot P(s_j \mid s_i) \cdot P(o_t \mid s_j) \Big)$$

In my dataset-based implementation, this allowed me to reconstruct the tag sequence exactly as observed in the recursion tables.

D. Numerical Stability: Log Probabilities and Smoothing

In practical implementations—including mine—multiplying many small probabilities leads to numerical underflow. To avoid this, all computations are performed in the **log-probability space**:

$$\log(a \cdot b \cdot c) = \log a + \log b + \log c$$

This kept the probability values stable when running Viterbi on my dataset, especially for longer sentences.

Zero probabilities also occur frequently when a transition or emission pair does not appear in the dataset. To address this, I applied smoothing:

$$P_{\text{smoothed}} = \begin{cases} P, & \text{if } P > 0 \\ 10^{-8}, & \text{otherwise} \end{cases}$$

This was particularly important for:

- unknown words in my dataset,
- unseen transitions between rare tags,
- preventing log-zero issues during implementation.

These techniques ensured that my Viterbi implementation remained stable for all sentences in the dataset.

E. Step 3: Backward Pass (Termination and Traceback)

After filling the Viterbi table:

$$s_T^* = \arg\max_j V[j,T]$$

Then trace back:

$$s_t^* = \text{best\_path}[s_{t+1}^*, t+1]$$

In my dataset implementation, this final traceback produced the predicted POS tag sequence for each sentence.

F. Advantages of the Viterbi Algorithm

- **Computational Efficiency:** Reduces tagging from exponential to polynomial time.
- **Optimality:** Produces the most probable tag sequence under the HMM assumptions.
- **Scalability:** Worked effectively for all sentences in my dataset.
- **Dynamic Programming:** Reuses previously computed results for efficiency.

Overall, the Viterbi algorithm enabled accurate POS tagging for my dataset by systematically combining transition and emission probabilities derived from the corpus.

IX. Results of HMM-Based POS Tagging using Viterbi

After implementing the Hidden Markov Model with the Viterbi algorithm on the preprocessed Penn Treebank test corpus, the predicted POS tags were evaluated against the ground-truth tags to measure accuracy. The accuracy metric is computed as the fraction of words for which the predicted tag matches the actual tag in the test corpus.

Formally, let $y = y_1, y_2, \ldots, y_n$ denote the ground-truth sequence of tags, and $\hat{y} = \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ the sequence predicted by the Viterbi algorithm. Then, the accuracy is defined as:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} \mathbf{1}\{\hat{y}_i = y_i\}}{n}$$

where $\mathbf{1}(\cdot)$ is the indicator function, equal to 1 if the condition is true and 0 otherwise.

Using this metric, the Viterbi algorithm achieved an overall accuracy of 90.24% on the test corpus. This demonstrates a significant improvement over the baseline prediction, which used the most frequent tag for each word and achieved 86.49% accuracy. The improvement highlights the effectiveness of modeling sequential dependencies between tags using the HMM and efficiently computing the optimal tag sequence with Viterbi.

A. Observations

- Incorporating transition probabilities allows the algorithm to consider context, leading to more accurate predictions than the baseline.
- Handling of unknown words through morphological and lexical rules ensures robust tagging for rare or unseen tokens.
- The dynamic programming approach of Viterbi efficiently computes the most probable sequence without evaluating all possible combinations, making it practical for large corpora.

X. Strengths, Limitations, and Potential Improvements

A. Strengths of the Approach

- **Statistical Foundation:** The HMM provides a mathematically grounded framework for modeling sequential data, allowing POS tagging to leverage probabilistic dependencies between tags and words.
- **Efficiency:** Using the Viterbi algorithm reduces computational complexity from exponential (brute force) to polynomial time, making it feasible to tag long sentences and large corpora.
- **Improved Accuracy:** Compared to a baseline emission-based method, HMM with Viterbi captures sequential dependencies between tags, resulting in higher tagging accuracy.
- **Handling Unknown Words:** Morphological and lexical rules allow the model to assign likely tags to unseen words, improving robustness on real-world text.
- **Dynamic Programming:** The algorithm reuses intermediate computations, avoiding redundant calculations and ensuring optimal tag sequences under the HMM assumptions.

B. Limitations

- **First-Order Markov Assumption:** The model assumes that each tag depends only on the previous tag, which may be insufficient for capturing long-range dependencies in sentences.
- **Independence of Observations:** HMM assumes each word depends only on its corresponding tag, ignoring contextual influence from neighboring words.
- **Simplistic Unknown Word Handling:** While suffix rules help, they may misclassify words with irregular morphology or uncommon forms.
- **Sparse Data Issues:** Rare words or rare tag transitions may lead to zero probabilities, requiring smoothing techniques which were not applied in the current implementation.
- **Limited to POS Tagging:** HMM is specifically suited for sequential labeling and may not capture semantic or syntactic nuances as well as more advanced models.

C. Potential Improvements

- **Higher-Order HMMs:** Incorporating dependencies on multiple previous tags (e.g., second-order HMM) could better capture sequential patterns.
- **Smoothing Techniques:** Applying techniques such as Laplace smoothing or Good-Turing estimation can handle rare or unseen tag transitions and emissions.
- **Contextual Embeddings:** Using word embeddings or contextual representations (e.g., BERT, ELMo) could improve tagging for ambiguous or unseen words.
- **Hybrid Models:** Combining HMM with more advanced models such as Conditional Random Fields (CRFs) or BiLSTM-CRF architectures may further improve accuracy.
- **Enhanced Unknown Word Handling:** Integrating part-of-speech prediction based on character-level embeddings or morphological analyzers could improve predictions for unseen words.

XI. References & Further Reading

1) Part of Speech Tagging Using Hidden Markov Models — Research paper (ResearchGate)
2) Speech and Language Processing (Draft) – Chapter on HMM & POS tagging — Stanford / Jurafsky & Martin
3) Parts-of-Speech (POS) and Viterbi Algorithm — Analytics Vidhya article
4) POS and HMM — Lecture notes (nlp2021sp)
5) Hidden Markov Models for POS Tagging — Foundations of Data Science III notes
6) Parts-of-Speech Tagging with Hidden Markov Models — GitHub repository with code examples