AID-829 Assignment 1 Report

Part-of-Speech Tagging using Hidden Markov Models

Team Members:

Vamsi Nalla	(MT2025719)
Uday Kiran Sure	(MT2025727)
Sathvik Teja Moturi	(MT2025717)
Jishnu Satwik Kacherlapalli	(MT2025707)

1 Introduction

This report documents the implementation of a Part-of-Speech (POS) tagging system using a Hidden Markov Model (HMM) and the Viterbi decoding algorithm. The implementation was built from scratch in Python, without using pre-built POS taggers or HMM libraries, as required by the assignment specifications.

The Universal Dependencies English Web Treebank (UD-EWT) was used as the dataset, with training data used to estimate HMM parameters and test data used for evaluation. The goal was to predict the most probable POS tag sequence for each sentence using the Viterbi algorithm with explicitly constructed probability and backpointer matrices.

2 Methodology

2.1 Data Parsing and Preprocessing

The CoNLL-U format files were parsed using a custom parser that extracts words and their corresponding Universal POS (UPOS) tags. The parser handles:

- Sentence boundary detection (blank lines)
- Comment filtering (lines starting with #)
- Multi-word token exclusion (e.g., 1-2 ranges)
- Extraction of word forms and UPOS tags from each token

The training dataset contained **12,544 sentences**, while the test set comprised **2,077 sentences**.

2.2 Unknown Word Handling

To handle unknown words (words appearing in test data but not in training data), a vocabulary smoothing strategy was implemented:

- Word frequencies were computed from the training data

- Words appearing only once (threshold = 1) were replaced with `<UNK>`
- Final vocabulary size: 9,874 unique words (including `<UNK>`)

This approach ensures that the model learns emission probabilities for the `<UNK>` token, which represents all rare or unseen words during testing.

2.3 HMM Parameter Estimation

Three types of probabilities were estimated from the training data using Maximum Likelihood Estimation (MLE):

2.3.1 Initial Probabilities

Initial probabilities represent the likelihood of a sentence starting with each POS tag:

$$P(\text{tag}) = \frac{\text{Count(tag at sentence start)}}{\text{Total sentences}} \tag{1}$$

2.3.2 Transition Probabilities

Transition probabilities capture the likelihood of transitioning from one POS tag to another:

$$P(\text{tag}_j \mid \text{tag}_i) = \frac{\text{Count}(\text{tag}_i \rightarrow \text{tag}_j)}{\text{Count}(\text{tag}_i)} \tag{2}$$

2.3.3 Emission Probabilities

Emission probabilities represent the likelihood of a tag generating a particular word:

$$P(\text{word} \mid \text{tag}) = \frac{\text{Count(tag emits word)}}{\text{Count(tag)}} \tag{3}$$

2.4 Viterbi Algorithm Implementation

The Viterbi algorithm was implemented following the standard dynamic programming approach with explicit matrix construction:

2.4.1 Viterbi Matrix (V)

A matrix V of size (sentence_length $\times$ number_of_tags) stores the maximum probability of reaching each tag at each position in the sentence. The recursion formula:

$$V_t(\text{tag}_j) = \max_i \left(V_{t-1}(\text{tag}_i) \times P(\text{tag}_j \mid \text{tag}_i) \times P(\text{word}_t \mid \text{tag}_j) \right) \tag{4}$$

2.4.2 Backpointer Matrix (B)

A matrix B of the same size stores the previous tag that led to the maximum probability at each position, enabling path reconstruction:

$$B[\text{tag}_j][t] = \arg\max_i \left(V_{t-1}(\text{tag}_i) \times P(\text{tag}_j \mid \text{tag}_i) \times P(\text{word}_t \mid \text{tag}_j) \right) \tag{5}$$

2.4.3 Algorithm Steps

The implementation follows these steps:

1. **Initialization:** Set $V_0[\text{tag}] = P(\text{tag}) \times P(\text{word}_0 \mid \text{tag})$ for all tags
2. **Recursion:** For each timestep t and tag, compute maximum probability from all previous tags
3. **Termination:** Identify the tag with maximum probability at the final timestep
4. **Backtracking:** Follow backpointers from the final tag to reconstruct the optimal sequence

3 Results and Evaluation

3.1 Overall Accuracy

The implemented HMM-based POS tagger achieved an accuracy of **89.42%** on the test set, correctly predicting 22,437 out of 25,094 tags.

This performance is competitive for a basic HMM model without advanced smoothing techniques or additional features. The result demonstrates that the fundamental HMM approach with Viterbi decoding is effective for POS tagging tasks.

3.2 Sample Prediction

First test sentence:

Field	Value
Words	What if Google Morphed Into GoogleOS ?
Gold Tags	PRON SCONJ PROPN VERB ADP PROPN PUNCT
Predicted	PRON SCONJ PROPN PROPN PROPN PROPN PUNCT

The model correctly identified most tags but confused some proper nouns (PROPN) with verbs (VERB) and prepositions (ADP), particularly for capitalized words in sequence.

3.3 Error Analysis

A confusion matrix was computed to identify common misclassification patterns. For the NOUN tag (one of the most frequent), the confusion breakdown was:

Predicted Tag	Count
NOUN (correct)	3,502
PROPN	362
VERB	110
ADJ	95
Others	54

Table 1: Confusion breakdown for gold NOUN tags

The most common confusion for NOUNs was with proper nouns (PROPN), which is linguistically reasonable as both categories share similar syntactic distributions. Confusion with VERBs and ADJectives was also observed, reflecting the ambiguity inherent in natural language.

4 Implementation Details

4.1 Libraries Used

As per assignment requirements, only basic Python libraries were used:

- `collections.Counter`: For word frequency counting
- `collections.defaultdict`: For probability storage and confusion matrix

No external NLP libraries, HMM packages, or sequence modeling tools were used. All components were implemented from scratch.

4.2 Code Structure

The implementation was organized into modular functions:

- `parse_conllu()`: Parses CoNLL-U format files
- `get_word_frequencies()`: Computes word frequency distribution
- `replace_rare_words()`: Handles unknown word smoothing
- `compute_*_probs()`: Estimates HMM parameters
- `viterbi_decode()`: Implements Viterbi algorithm with explicit matrices
- `decode_test_set()`: Processes entire test set
- `compute_accuracy()`: Evaluates performance

5 Conclusion

This assignment successfully implemented a complete POS tagging pipeline using Hidden Markov Models and the Viterbi algorithm. The implementation demonstrates a solid understanding of:

- HMM parameter estimation from annotated corpora
- Dynamic programming for sequence labeling
- Unknown word handling through vocabulary smoothing
- Model evaluation using standard metrics

The achieved accuracy of 89.42% demonstrates that the basic HMM approach is effective for POS tagging, though there is room for improvement through advanced techniques such as:

- Add-k smoothing for unseen tag-word combinations
- Suffix-based features for unknown word classification
- Trigram or higher-order HMMs for better context modeling

- Integration of lexical features (capitalization, word length, etc.)

The explicit construction of Viterbi and backpointer matrices provided clear insight into the dynamic programming process and facilitated debugging. The modular code structure ensures maintainability and allows for easy experimentation with different smoothing strategies and model variants.

All deliverables have been completed as specified: source code for HMM training, Viterbi decoding implementation, accuracy evaluation