Report: HMM-based POS Tagger Implementation and Optimizations

Ethan Reinhart

May 5, 2025

Abstract

This report describes the implementation of a Hidden Markov Model (HMM) part-of-speech tagger, the use of the Viterbi decoding algorithm to infer tag sequences, and a series of optimizations—ranging from Laplace smoothing through feature-based backoff and higher-order transitions—that improved model accuracy from 22.73% to nearly 94.6% on development data.

1 Introduction

Part-of-speech (POS) tagging assigns syntactic categories (e.g. `NN`, `VBZ`, `DT`) to each word in a sentence. A classic approach is to model the joint distribution of tags and words using a Hidden Markov Model (HMM) and then find the single most likely tag sequence via the Viterbi algorithm.

2 Data Preprocessing

2.1 Corpus Format

Our training and development corpora use one token per line in `.pos` files:

$$\begin{array}{rl} Ms. & NNP \\ Haag & NNP \\ plays & VBZ \\ Elianti & NNP \\ . & . \end{array}$$

Sentences are separated by blank lines. Similarly, `.words` files list words only, one per line, with blank lines for sentence breaks.

3 HMM Parameter Estimation

An HMM for POS tagging includes:

- Initial distribution: $\pi_t = P(y_1 = t)$.
- Transition matrix: $a_{u \to v} = P(y_i = v \mid y_{i-1} = u)$.
- Emission matrix: $b_t(w) = P(x_i = w \mid y_i = t)$.

We collect raw counts:

$$\text{start\_counts}[t],\ \text{trans\_counts}[u, v],\ \text{emit\_counts}[t, w]$$

and then normalize.

3.1 Laplace (add-α) Smoothing

To avoid zero probabilities, we add $\alpha > 0$ to every count:

$$\pi_t = \frac{\text{start\_counts}[t] + \alpha}{N + \alpha T}, \quad a_{u \to v} = \frac{\text{trans\_counts}[u, v] + \alpha}{\sum_w (\text{trans\_counts}[u, w] + \alpha)}, \quad b_t(w) = \frac{\text{emit\_counts}[t, w] + \alpha}{\sum_{w'} (\text{emit\_counts}[t, w'] + \alpha}$$

We reserve one extra emission slot "<UNK>" for unseen words:

$$b_t(\texttt{<UNK>}) = \frac{\alpha}{\sum_{w'} (\text{emit\_counts}[t, w'] + \alpha)}.$$

4 Viterbi Decoding

Given a new sentence $x_1, \ldots, x_n$, Viterbi finds $\hat{y}_{1:n} = \arg\max_{y_{1:n}} P(y_{1:n}, x_{1:n})$ in $O(n\,T^2)$. In log-space:

$$V_i(v) = \max_u \big[V_{i-1}(u) + \log a_{u \to v}\big] + \log b_v(x_i),$$

with backpointers $\text{bp}_i(v) = \arg\max_u \ldots$. After filling $V_i(\cdot)$, we backtrace from $\arg\max_t V_n(t)$.

5 Optimizations

1. **Laplace smoothing and <UNK>.** Gives all word probabilities a nonzero value so that unseen words don't default to the same value.
2. **Log-space arithmetic.** Stores $\log \pi, \log A, \log B$ to avoid underflow.
3. **Restricted tag-space.** For each seen word, only consider the tags that emitted it during training (via a `word_tag` dictionary).

6 Experimental Results

6.1 Optimization Implementations

Model variant	Dev accuracy
Baseline (no smoothing)	22.73%
+ Restricted tag-space (faster, same acc)	22.73%
+ Laplace smoothing ($\alpha = 0.1$) and <UNK>	94.6%
+ Log Space	94.6%

Table 1: Accuracy improvements from successive optimizations.

6.2 Hyperparameter Tuning

α	Dev accuracy
0.01	94.289715%
0.1	94.578882%
0.5	94.578882%
1.0	94.499741%
10.0	94.088820%

Table 2: Accuracy improvements from different α values.

7 Conclusion

We implemented a standard HMM POS tagger with Viterbi decoding, and applied a suite of optimizations—Laplace smoothing, log-space, restricted tag lists boosted dev-set accuracy from 22.73% to nearly 94.6%. Future work includes migrating to beam search, integer indexing for quicker lookups, and a suffix model to classify unknown words using their suffix.