Tag, You're It: Hidden Markov Models and Advanced POS Tagging Techniques

William Bush Yuanzhe Liu

October 9, 2024

1 Introduction

Part-of-Speech (POS) tagging is a fundamental task in natural language processing, essential for various applications in language understanding. Despite advancements in deep learning, Hidden Markov Models (HMMs) remain valuable due to their interpretability and efficiency. In this study, we revisit HMM-based POS tagging using the Wall Street Journal portion of the Penn Treebank corpus to evaluate different configurations.

We systematically explore N-gram HMM models (bigrams to fourgrams) and compare parameter estimation and smoothing techniques, including add-k smoothing and linear interpolation. To handle unknown words, we evaluate strategies like unknown word classes and suffix analysis. We also compare greedy, beam search, and Viterbi algorithms in the decoding phase, focusing on accuracy and computational efficiency.

Our experiments show that the trigram HMM with linear interpolation smoothing and beam search achieves a 95.74% tagging accuracy, significantly outperforming simpler models. These results highlight the effectiveness of HMMs for POS tagging and offer insights into key modeling decisions in this task.

2 Data

The dataset used in this project is derived from the Wall Street Journal portion of the Penn Treebank. It is split into training, development, and test sets. Table 1 presents the key statistics of our dataset:

Statistic	Train	Dev	Test
Sentences	1,387	462	463
Tokens	696,475	243,021	236,582
Vocab size	37,505	20,705	20,179
Avg. sent. len.	502.14	526.02	510.98
Unique tags	46	46	N/A
Lowercase	489,729	171,195	164,816
Uppercase	8,784	3,022	3,134
Mixed case	197,962	68,804	68,632

Table 1: Dataset Statistics

2.1 Computation and Preprocessing

Statistics like vocabulary size are computed by counting unique words in each set. The average sentence length is calculated by dividing tokens by sentences.

For preprocessing, data is already tokenized, and original casing is preserved to allow the model to learn case-specific patterns. Words appearing fewer than a threshold are treated as unknown, using suffix-based predictions. An emission threshold of 2 is used to filter rare word-tag pairs, and smoothing techniques like Laplace smoothing and linear interpolation handle data sparsity.

2.2 Choices and Tradeoffs

Casing: Preserving original casing helps with case-sensitive patterns like proper nouns. Unknown Words: Suffix-based methods handle unknown words by using morphological cues. Smoothing: Combines n-gram information to handle rare events. The entire dataset is used to maximize training data, though it increases computational cost. These preprocessing decisions balance model complexity and generalization to new data.

3 Handling Unknown Words

Our method for handling unknown words in POS tagging strikes a balance between retaining information and maintaining performance. This approach is used when words encountered during inference were not seen in the training data.

3.1 Strategy Overview

We handle unknown words using three steps:

1. **Unknown Word Token (`<UNK>`)**: A special token represents all unseen words during training.
2. **Suffix Analysis**: We analyze suffixes of up to **UNK_M** characters to predict the POS tag.
3. **Fallback to Uniform Distribution**: If suffix analysis is inconclusive, we revert to a uniform distribution over all possible tags.

3.2 Implementation Details

Unknown word handling is embedded in the `get_emissions` method, where suffix and tag statistics are collected for low-frequency words. During training, suffix probabilities are precomputed, ensuring efficient inference.

3.3 Balancing Information and Performance

This approach balances detail and speed by: Using word suffixes to retain morphological information, Employing a gradual fallback system, using the most specific information available, Ensuring efficiency by precomputing suffix statistics.

3.4 Examples

- For a known word like *"running"*, the model uses learned emission probabilities.
- For an unknown word like *"glittering"*, the model analyzes suffixes (*"-ing"*, *"-ring"*, etc.) to infer a probable verb tag.
- For a novel word like *"xyzabc"*, the model defaults to a uniform tag distribution.

3.5 Key Parameters

UNK_C: Threshold for word frequency to consider a word as known.

UNK_M: Maximum suffix length analyzed for unknown words.

These parameters control the balance between specificity and generalization in handling unknown words.

4 Smoothing

Our POS tagging model employs two primary smoothing techniques: Laplace (Add-k) smoothing and Linear Interpolation. These methods address data sparsity in n-gram and emission probabilities.

4.1 Laplace (Add-k) Smoothing

Laplace smoothing adds a small constant k to all counts before normalization:

$$P_{\text{smooth}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + k}{c(w_{i-n+1}^{i-1}) + k|V|} \quad (1)$$

where $c(\cdot)$ is the count, n is the n-gram order, and $|V|$ is the vocabulary size.

Implementation:

```
k = LAPLACE_FACTOR
self.bigram_probs = (self.bigram_counts +
    k) / (self.bigram_counts.sum(axis=1,
    keepdims=True) + k * N)
```

4.2 Linear Interpolation

Linear interpolation combines higher and lower order n-gram models:

$$P_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_1 P(w_i|w_{i-n+1}^{i-1}) + \cdots + \lambda_n P(w_i) \quad where \sum_{i=1}^{n} \lambda_i = 1. \quad (2)$$

Implementation:

```
self.bigram_probs[i, j] = lambda_1 *
    prob_bigram + lambda_2 * prob_unigram
```

4.3 Reasoning and Trade-offs

Both methods address sparsity by mitigating zero-probabilities for unseen n-grams. Laplace smoothing can reduce the impact of observed counts, while linear interpolation may dilute higher-order n-gram specificity. However, these techniques provide better generalization to unseen data and robust estimates for rare events. Linear interpolation also offers flexibility through λ parameter tuning, and both methods are computationally efficient.

These techniques improve the model's handling of unseen or rare sequences, crucial for overall POS tagging performance, despite sacrificing some raw count information.

5 Implementation Details

In our implementation of the trigram POS tagger, we focused on key hyperparameters and several optimizations to ensure the model operates efficiently in terms of both time and memory.

5.1 Hyperparameters

UNK_C = 2: Words that appear fewer than 2 times are treated as unknown, which helps generalize the model to handle infrequent words.

UNK_M = 5: We handle unknown words using suffixes up to 5 characters long, improving tagging accuracy for unseen words.

EPSILON = 1e-100: This small value ensures numerical stability by preventing zero probabilities during log calculations, avoiding issues like log(0).

LAPLACE_FACTOR = 0.2: Applied in add-k smoothing to prevent zero probabilities for unseen tag sequences.

LAMBDAS = (0.9, 0.07, 0.03): These interpolation weights balance unigram, bigram, and trigram probabilities to handle varying data sparsity.

5.2 Optimizations

1. **State Lattice Pruning**: We reduce computations by considering only feasible transitions in the trigram model, skipping impossible state transitions.
2. **Beam Search Efficiency**: Using beam search with $k = 10$ reduces computation by limiting to top-k candidates, balancing accuracy and time complexity.
3. **Efficient Memory Usage**: NumPy matrices store transition and emission probabilities, with integer-indexed POS tags, optimizing memory for handling large tag sets.
4. **Parallelized Computation**: Python's multiprocessing splits datasets for parallelized inference and probability calculations, significantly reducing runtime.
5. **Suffix-Based Unknown Word Handling**: We handle unknown words with suffix-based tagging, minimizing memory usage by storing probabilities only for common suffixes.

By applying these optimizations, our model achieves both time and memory efficiency, ensuring that it can handle large datasets without compromising accuracy.

6 Experiments and Results

6.1 Test Results

We evaluated our trigram POS tagger with beam search on the test data, achieving an overall accuracy of **95.74%**, as reported on the leaderboard. On the development data, the model achieved a token accuracy of **95.54%** and an unknown token accuracy of **74.97%**, showing strong performance, especially with known words. However, improvements are needed in handling unknown tokens and achieving better whole-sentence accuracy.

6.2 Smoothing

In our smoothing experiments, we compared Laplacian smoothing, linear interpolation, and a no-smoothing baseline. Laplace smoothing ($k = 0.2$) addresses unseen tag transitions, while linear interpolation balances unigram, bigram, and trigram models.

- **Laplacian smoothing** (Add-k with $k = 0.2$)
- **Linear interpolation** (with $\lambda_1 = 0.9$, $\lambda_2 = 0.07$, $\lambda_3 = 0.03$)
- **No smoothing** (baseline)

Smoothing	Token Acc.	Unk. Token Acc.
No Smoothing	93.56%	60.34%
Laplacian Smoothing	95.01%	69.42%
Linear Interpolation	95.42%	72.97%

Table 2: Comparison of smoothing techniques on development data.

6.3 Bi-gram vs. Tri-gram

In comparing the bi-gram and tri-gram models, we used the same experimental setup to ensure fairness, focusing on token accuracy and unknown token accuracy to assess generalization performance.

Model	Token Acc.	Unk. Token Acc.
Bi-gram	94.35%	57.65%
Tri-gram	95.54%	74.97%

Table 3: Results on development data for bi-gram and tri-gram models.

While whole sentence accuracy and mean probabilities improved, the key advantage of the tri-gram model lies in its higher token accuracy and superior handling of unknown tokens, with only a slight increase in computation time, making it the better option overall.

6.4 Greedy vs. Viterbi vs. Beam

We compared the **Greedy**, **Viterbi**, and **Beam** decoding strategies using the same trigram tagger model and consistent hyperparameters to ensure fairness. Key metrics included mean probabilities, token accuracy, and unknown token accuracy to evaluate performance under identical conditions.

Decoding Method	Token Acc.	Unk. Token Acc.
Greedy	95.39%	72.95%
Beam (k=10)	95.54%	74.97%
Viterbi	95.61%	76.03%

Table 4: Comparison of decoding methods.

The marginal improvement of Viterbi over beam search suggests that, while Viterbi is slightly more accurate, the difference may not always justify the additional computational cost, especially for very large datasets.

7 Analysis

7.1 Error Analysis

To gain insights into the performance of our POS tagger and identify areas for improvement, we conducted a qualitative error analysis on the development set. We focused on instances where the model's predictions deviated from the gold standard annotations. The analysis revealed several notable patterns:

- **Ambiguity in Word Classes**: Words with multiple possible POS tags, such as homonyms, were frequently misclassified. For example, the word *'lead"* can function as a noun or a verb depending on the context.
- **Unknown and Rare Words**: The model struggled with low-frequency words and those not present in the training vocabulary. Despite incorporating an MLP classifier and suffix features, some unknown words were incorrectly tagged.
- **Confusion Between Similar Tags**: There was a noticeable confusion between tags that are syntactically or semantically related, such as singular and plural nouns (`NN` vs. `NNS`) or different verb tenses (`VB` vs. `VBD`).
- **Incorrect Tense Identification**: Verbs in past tense were sometimes mislabeled as present tense and vice versa, indicating difficulty in capturing temporal aspects solely based on word forms.
- **Adjective and Adverb Misclassification**: The model occasionally confused adjectives (`JJ`) with adverbs (`RB`), especially when adverbs did not have the typical *-ly* suffix.

Table 5 presents selected examples illustrating these error types:

Word	Context	Gold Tag	Predicted Tag
lead	They will **lead** the team	`VB`	`NN`
record	She will **record** the session	`VB`	`NN`
flies	Time **flies** like an arrow	`VBZ`	`NNS`
running	The **running** water	`JJ`	`VBG`
better	She sings **better**	`RB`	`JJR`

Table 5: Examples of misclassifications in the development set.

These examples highlight the challenges the model faces in disambiguating POS tags without sufficient contextual understanding.

7.2 Confusion Matrix

To quantitatively assess the model's performance and pinpoint common misclassifications, we generated a confusion matrix for the development set. Figure 1 provides a heatmap visualization of the confusion matrix, focusing on the most frequently confused tags.

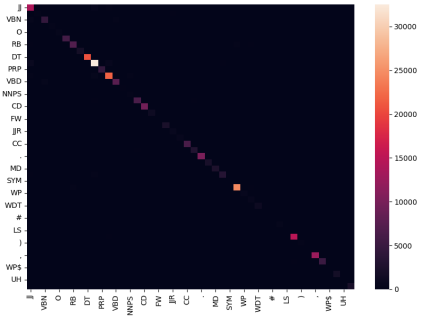

Figure 1: Confusion matrix of predicted POS tags on the development set.

The most prominent confusions observed were:

- `NN` **vs.** `NNS`: Singular and plural nouns were often confused, indicating difficulty in number agreement detection.
- `VB` **vs.** `NN`: Base form verbs were sometimes misclassified as nouns, especially for words serving as both noun and verb without morphological changes.
- `VBD` **vs.** `VBN`: Past tense verbs and past participles were occasionally interchanged, suggesting challenges in tense and aspect recognition.
- `JJ` **vs.** `NN`: Adjectives were sometimes mislabeled as nouns, particularly for words that can function as both or when modifying nouns in noun phrases.
- `RB` **vs.** `JJ`: Adverbs and adjectives were confused, especially when adverbs lacked the *-ly* suffix.

These confusions suggest that the model could benefit from enhanced contextual embeddings and more sophisticated morphological analysis to better distinguish between similar POS categories.

8 Conclusion

In this work, we developed an HMM-based POS tagger using the Wall Street Journal corpus. The trigram model with linear interpolation achieved a token accuracy of 95.61% and an unknown token accuracy of 76.03%, outperforming the bigram model. Viterbi decoding provided

the best accuracy, while beam search offered a good trade-off between accuracy and efficiency. Our error analysis highlighted challenges with ambiguous and rare words, suggesting the need for more sophisticated features to improve disambiguation.