CIS5300: Assignment 2 Report

Edward Ho Benjamin Yoon

October 9, 2024

1 Introduction

The task at hand is to implement a part-of-speech (POS) tagger using various methods, including MLE estimation, greedy decoding, Viterbi algorithm, and beam search, with smoothing techniques applied. The goal is to predict the POS tags for a sequence of words, with a focus on unknown words, using data from the Penn Treebank. The dataset contains sentences from the Wall Street Journal (WSJ) and includes training, development, and test sets, with missing POS tags in the test set to be predicted.

The key experiments conducted involved developing a POS tagger with bigram and trigram models, handling unknown words through an MLP or Gradient Boosting classifier, and testing with Laplace smoothing and Linear Interpolation smoothing. All techniques were evaluated using both development and test data, and their performance was compared based on metrics such as token accuracy, unknown word accuracy, and whole sentence accuracy. In addition to Laplace smoothing, Linear Interpolation smoothing was implemented to balance better lower- and higher-order n-gram contexts. This method helped improve the overall performance by effectively combining unigram, bigram, and trigram probabilities, which smoothed out the sparsity issue in higher-order n-grams. Furthermore, we experimented with beam search alongside greedy decoding and Viterbi. Beam search, by exploring multiple paths rather than committing to a single one, notably improved sentence-level accuracy, especially in handling ambiguous sequences. Ultimately, our best results were generated from the trigram model using Beam-3 search with Linear Interpolation smoothing. Although Beam Search and Viterbi outperformed Greedy by about 1%, the difference between Viterbi and Beam Search was only a fraction of a percent.

2 Data

The dataset used consists of sentences from the Wall Street Journal, split into training, development, and test sets. The dataset statistics are summarized in Table 1. Each sentence in the dataset is tokenized into words, and the corresponding POS tags are provided in the training and development sets. The test set contains words without tags. Some of the summary statistics of the data can be found in Table 1 (see Appendix A).

Before computing features, data preprocessing included tokenization, lowercasing, and handling punctuation. Unknown words were identified as those not present in the training data. The vocabulary size was computed after excluding words occurring less than 10 times, and suffixes/prefixes were extracted for unknown word handling.

To address the issue of data sparsity—where certain n-gram combinations (particularly trigrams) may not appear frequently enough in the training data—we implemented Linear Interpolation smoothing. This approach helped mitigate the lack of coverage for unseen or rare n-grams by combining unigram, bigram, and trigram probabilities, ensuring more robust predictions. By leveraging information from lower-order n-grams, we were able to reduce the impact of data sparsity and improve the overall performance of the POS tagger.

3 Handling Unknown Words

Handling unknown words effectively is a critical aspect of POS tagging. We implemented an MLP-based tagger (POSTagger_MLP) that uses a combination of word embeddings and additional features, such as suffixes, prefixes, capitalization, word length, and the presence of numbers or punctuation. Unknown words were identified as those not found in the training vocabulary.

The feature extraction process involved using a TF-IDF vectorizer to convert words into embeddings, followed by dimensionality reduction using PCA. Additionally, suffix and prefix indices were extracted for each word to provide more contextual information. For example, the suffixes 'ing' or 'ed' can give strong indications about the part of speech (e.g., verbs). These features were concatenated with the reduced word embeddings to create input vectors for the MLP model.

To balance the complexity of the model and prevent overfitting, words occurring more than 10 times in the training data were excluded from the training of the unknown word classifier. This approach was effective in predicting unknown words without overly relying on high-frequency words in the training set.

We implemented a Gradient Boosting classifier as well but found more success using the MLP classifier so that was our primary classifier of interest for unknown words moving forward.

4 Smoothing

In POS tagging, smoothing is essential to handle cases where certain tag transitions or emissions are not observed in the training data. Without smoothing, unseen transitions or emissions would result in zero probabilities, causing inference failures. To address this, two smoothing techniques were implemented: Laplace Smoothing and Linear Interpolation.

Laplace Smoothing adds a small constant ($LAPLACE\_FACTOR$) to all counts, ensuring non-zero probabilities for unseen events. This prevents inference failures by accounting for rare transitions. The smoothed probability is given by:

$$P(x_i|x_{i-1}) = \frac{count(prev\_tag, tag) + m \cdot \frac{1}{|V^*|}}{count(prev\_tag) + m}$$

While this method can overestimate rare sequences, it improves robustness, particularly for trigram models.

Linear Interpolation combines unigram, bigram, and trigram models, weighting them to balance generalization across different n-grams. The combined probability is calculated as:

$$\begin{aligned}\hat{p}(x_i|x_{i-1}, x_{i-2}) = {} & \lambda_3 \cdot p(x_i|x_{i-1}, x_{i-2}) \\ & + \lambda_2 \cdot p(x_i|x_{i-1}) \\ & + \lambda_1 \cdot p(x_i) \end{aligned} \tag{1}$$

This technique helps balance specificity and generalization by combining different n-gram contexts, which proved particularly effective when used in conjunction with beam search. Beam search, in exploring multiple paths, benefits from Linear Interpolation's ability to account for a wider range of tag transitions. Although smoothing introduces some loss of specificity by assigning non-zero probabilities to unlikely sequences, the overall performance improved, particularly in unknown word scenarios where tag transitions may not have been seen in training.

5 Implementation Details

The implementation involved developing a POS tagger that supports unigram, bigram, and trigram models, with the ability to switch between different inference methods: greedy decoding, beam search at different k values, and the Viterbi algorithm. The model was designed to handle unknown words using an MLP-based classifier.

Special handling was applied for sentence boundaries using `-DOCSTART-` markers to represent the beginning of sentences, treating them as `<START>` tags. Emission probabilities, $P(word|tag)$, were computed by counting occurrences of word-tag pairs and normalizing across all words for each tag.

Key hyperparameters included the beam width for beam search, which was set to $k = 3$ after experimentation, balancing performance, and computation time. Beam search was particularly useful in improving sentence-level accuracy by exploring multiple candidate paths rather than committing to a single path as greedy decoding does. This exploration of alternate tag sequences allowed the model to recover from early prediction errors, leading to better overall sentence accuracy.

To support beam search efficiently, we optimized our implementation by leveraging NumPy for fast array operations, and multiprocessing for parallel evaluation, significantly reducing runtime. Probabilities were stored in log space to prevent underflow during calculations, particularly when dealing with very small transition and emission probabilities in the trigram model.

For handling unknown words, the MLP classifier used a three-layer architecture with hidden layers of 100, 100, and 50 units, respectively. The PCA component reduced the dimensionality of the TF-IDF embeddings to 100 features, preventing overfitting while maintaining sufficient information. This dimensionality reduction improved computation time and model generalization.

In addition, Linear Interpolation smoothing was employed to combine unigram, bigram, and trigram probabilities, providing a balance between generalization and specificity in n-gram modeling. When combined with beam search, this smoothing method helped the model explore a wider range of tag sequences while mitigating data sparsity issues in the training set.

6 Experiments and Results

Test Results Our final model, submitted to the leaderboard, achieved an overall accuracy of 96.11164%. This result was obtained using a trigram model with Linear Interpolation smoothing and beam search ($k = 3$). These metrics were computed on the test data and reflect the model's ability to generalize beyond the development set.

Leaderboard Results:

- Overall Accuracy: 96.11164%

The accuracy metrics on the development set are shown in Table 2 (see Appendix A). We report token accuracy, unknown word accuracy, and whole sentence accuracy, providing insights into the model's performance on both individual words and entire sentences.

Smoothing We experimented with Laplace smoothing and Linear Interpolation smoothing to improve model robustness in the presence of unseen transitions and emissions. Originally, Laplace smoothing outperformed Linear Interpolation smoothing. However, through experimentation with different parameters for the Linear Interpolation, namely trying different lambda values, we were able to find those that were most optimal $(0.1, 0.3, 0.6)$. Using these lambda values with Linear Interpolation smoothing consistently outperformed Laplace smoothing across most metrics. By combining unigram, bigram, and trigram probabilities, Linear Interpolation enabled better handling of sparse data, resulting in higher token and unknown word accuracy. The overall results of the smoothing experiments are presented in Table 3 in Appendix A.

We conclude that Linear Interpolation provides a more nuanced way to address the data sparsity issue than Laplace smoothing, particularly when combined with beam search. This smoothing method better captures lower-order n-gram context while retaining the specificity of higher-order n-grams.

Bi-gram vs. Tri-gram To compare the overall performance of bi-gram and tri-gram models, we averaged the accuracies across all decoding methods (greedy, beam-3, and Viterbi) for both models. The results show that tri-gram models consistently outperformed bi-grams in both sentence and token accuracy, while bi-grams achieved slightly higher unknown word accuracy.

On average, the tri-gram model produced a 38.73% sentence accuracy, 95.48% token accuracy, and 77.12% unknown word accuracy. The additional context provided by tri-grams helps the model better capture the relationships between tags, especially at the sentence level, leading to higher overall accuracy.

The bi-gram model averaged 36.07% sentence accuracy, 95.46% token accuracy, and 77.75% unknown word accuracy. Although the bi-gram model underperformed on sentence and token accuracy, it slightly outperformed the tri-gram model in handling unknown words, suggesting that bi-grams can generalize better for rare or unseen words.

The results are summarized in Table 4 (see Appendix A).

The tri-gram model outperforms the bi-gram model by a notable margin, averaging 38.73% compared to 36.07%. This suggests that the additional context from the tri-gram model helps capture more complex tag sequences, improving sentence-level coherence. The difference in token accuracy, however, is small (95.48% vs 95.46%), indicating that both models are effective at the token level. Interestingly, the bi-gram model averaged a slightly higher unknown word accuracy (77.75% vs 77.12%). This could be because bi-grams generalize better when dealing with rare or unseen words, as they rely on simpler tag transitions that are less specific.

Greedy vs. Viterbi vs. Beam We compared the performance of greedy decoding, Viterbi, and beam search (with $k = 3$). Beam search generally outperformed both greedy and Viterbi decoding, particularly in sentence-level accuracy. As shown in Table 5, beam search achieved the highest accuracy across all metrics when used in combination with Linear Interpolation smoothing.

- Greedy decoding: While fast and efficient, greedy decoding often fails to find the optimal sequence, particularly for longer sentences or those with more complex dependencies. It underperforms relative to the other methods.

- Viterbi decoding: Viterbi guarantees an optimal solution for the given probabilities but was only slightly better than greedy in our experiments. However, Viterbi's computational cost was higher than Greedy.

- Beam search ($k = 3$): Beam search offered a practical trade-off between computational cost and accuracy. Exploring multiple candidate paths improved both sentence and token accuracy without the overhead of Viterbi.

The results are summarized in Table 5 (see Appendix A).

We found that beam search (k=3) provided the best balance of performance and computational cost, outperforming Viterbi by a fraction of a percent. Greedy decoding, though efficient, consistently fell short of the optimal solution, with lower sentence accuracy.

7 Analysis

Error Analysis Error analysis reveals that certain classes of mistakes are common across different types of sentences. Table 6 provides examples of incorrect predictions made by the model on the development set, focusing on specific failure types (see Appendix A). Common errors include:

- **Adjectives and Verbs**: Confusion between past participles (**VBN**) and adjectives (**JJ**) is frequent, especially in sentences where past participles function as adjectives (e.g., "proposed").

- **Noun Forms**: Singular and plural nouns (**NN** vs. **NNS**) are often confused, particularly in cases where number marking is ambiguous.

- **Proper Nouns**: Multi-word proper nouns (e.g., company names like "Comair Holdings Inc.") are sometimes misclassified as regular nouns (**NN**), rather than proper nouns (**NNP**).

Confusion Matrix Figure 1 presents the confusion matrix for the POS tagger, where the most common confusion can be visualized (see Appendix A). The matrix highlights key areas where the model struggled:

- **Noun Confusion**: Frequent confusion between **NN** (singular noun) and **NNS** (plural noun), likely due to ambiguous contexts in number marking.

- **Verb Confusion**: Verbs in different forms, such as **VB** (base form) and **VBG** (gerund), are often misclassified, particularly in sentences involving auxiliary verbs.

- **Adjectives and Verbs**: There is notable confusion between **JJ** (adjective) and **VBN** (past participle), especially when past participles function as adjectives in noun phrases.

8 Conclusion

In conclusion, the POS tagger achieved a strong performance with a token accuracy of 95.86%, an unknown word accuracy of 76.19%, and a sentence accuracy of 41.04%. The use of Linear Interpolation for smoothing and beam search ($k = 3$) proved effective in handling sparse data and improving unknown word predictions. While Viterbi decoding guarantees optimality, beam search offers a practical balance between accuracy and computational efficiency. Future work could involve experimenting with additional features for unknown word handling and exploring higher-order N-grams; this is because 3-grams consistently outperformed 2-grams across all metrics.

A Appendix A

Dataset	**Sentences**	**Tokens**	**Unique Words**
Train Set	1,387	696,475	37,505
Dev Set	462	243,021	20,705
Test Set	463	236,582	20,179

Table 1: Data Statistics

Metrics	**Accuracy**
Token	0.95859
Unknown Token	0.76195
Whole Sentence	0.41041

Table 2: Accuracy Metrics for Final Model on Development Set

Smoothing Method	**Avg. Sentence Acc**	**Avg. Token Acc.**	**Avg. Unknown Acc.**
No Smoothing	30.32%	93.27%	78.04%
Laplace	38.52%	95.37%	77.28%
Linear Interpolation	38.93%	95.58%	76.96%

Table 3: Smoothing Techniques Average Accuracy Across Metrics

Model	**Avg. Sentence Acc.**	**Avg. Token Acc.**	**Avg. Unknown Acc.**
Tri-gram (3-gram)	38.73%	95.48%	77.12%
Bi-gram (2-gram)	36.07%	95.46%	77.75%

Table 4: Average Accuracies for Bi-gram and Tri-gram Models

Decoder/Smoother	**Sentence Acc.**	**Token Acc.**	**Unknown Word Acc.**
Greedy/Laplace	33.57%	94.77%	77.83%
Beam-3/Laplace	41.01%	95.56%	77.07%
Viterbi/Laplace	40.97%	95.78%	76.95%
Greedy/Linear Interpolation	34.72%	95.02%	78.18%
Beam-3/Linear Interpolation	41.04%	95.86%	76.19%
Viterbi/Linear Interpolation	41.03%	95.85%	76.52%

Table 5: Accuracies for Tri-gram Models Across Different Decoding Methods and Smoothing Techniques

Sentence (Truncated)	Gold Tags	Predicted Tags
”A shareholder filed suit, seeking to block Unitel Video Inc.’s proposed plan...”	...VBN NN...	...JJ NN...
”The fund invests mainly in gold and silver bullion.”	...VBZ NN...	...VBZ NN...
”Comair Holdings Inc. said in Cincinnati that it bought...”	...NNP NNP...	...NN NNP...

Table 6: Error examples from the development set

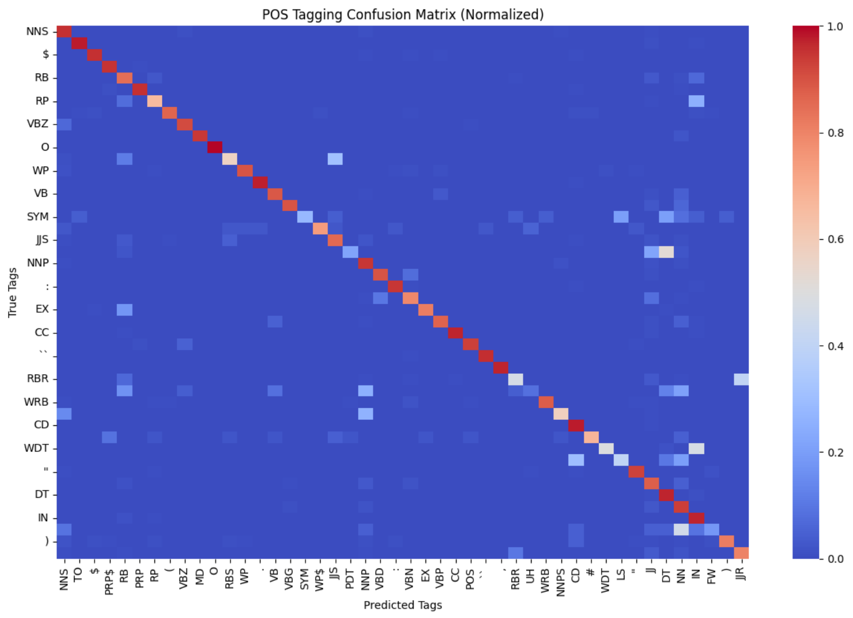

Figure 1: POS Tagging Confusion Matrix (Normalized)