CIS530 Assignment 2: Comparative Analysis of Approaches in Part of Speech (POS) Tagging

Jingyi Guo Jiacheng Zhu

1 Introduction

Part-of-speech (POS) tagging assigns each word in a text its grammatical label, such as noun or verb. It's fundamental for tasks like machine translation and sentiment analysis. For our model, we used a subset of the Penn Treebank project, which contains POS-tagged passages data from the Wall Street Journal established by Marcus et al. (1993). It stands as one of the most influential resources in Natural Language Processing (NLP), serving as a benchmark in many syntactic analysis tasks (Vadas, 2007).

We developed a model that merges the feed forward neural network (FFNN) classifier, for handling unseen words, with the Hidden Markov Model, which predicts tag sequences for word series. Our investigation spanned the accuracy and efficiency of different state sequences (unigram to tetragram), smoothing methods (interpolation and add-k smoothing), and inference algorithms (greedy decoding, Beam Search, Viterbi) within the Hidden Markov Model framework. Our standout result was the combination of bigram Viterbi with Add-One smoothing, which achieved an F1 score of 96.06% on the test subset.

2 Data

The training dataset contained 1,387 sentences; The development dataset had 462 sentences, and the test dataset comprised 463 sentences. Each word in training and development sets was annotated with a POS tag. The meaning of each tag can be found on the official website of the dataset. [3] Given that symbols, including periods and quotation marks, were assigned POS tags in our analysis, we treated them as words. We converted all words to lowercase before computing word statistics. Table 1 presents the vocabulary size, detailing the count of unique words and the total number of words in each dataset subset as well as across the entire combined dataset. In Table 2, we depict the document length using sentence length, which is defined by the number of words in each sentence. For statistics related to POS tags, Figure 1 provides their distributions in training and development sets combined.

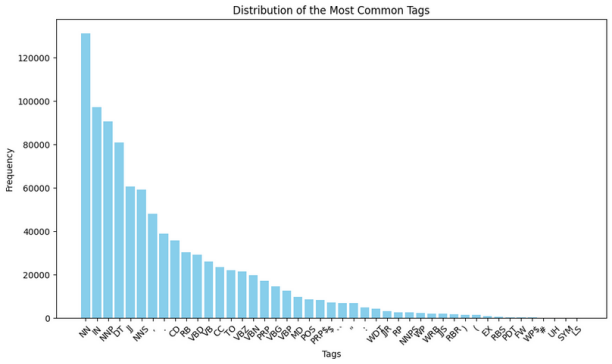

Figure 1: Enter Caption

	Unique Words	Total Words
Training Set	34121	695088
Development Set	18934	242559
Test Set	18447	236119
Whole Dataset	43766	1173766

Table 1: Vocabulary Size of Training, Development, Test and whole Dataset

	Average SL	Median SL	Maximum SL	Minimum SL
Training Set	501.14	276	4534	22
Development Set	525.02	296	3385	24
Test set	509.98	262	3949	27
Whole Dataset	507.68	276	4534	22

Table 2: Sentence Length (SL) of Training, Development, Test, and Whole Dataset

3 Handling Unknown Words

Handling unknown words is vital for precise part-of-speech (POS) tagging. To address this, a training dataset is created by tokenizing words from the original training sets designed for the Hidden Markov Model (HMM) POS tagger, pairing each token with its corresponding tag. Features are then

extracted from these tokens, including 1-, 2-, and 3-character prefixes and suffixes, capitalization, hyphen segments, numeric attributes, word length, vowel counts, and special characters. These features are essential as they hint at the token's tag, for instance, words ending in "ing" are typically verbs, and those starting with a capital letter could be nouns or proper nouns.

A feed-forward neural network (FFNN) is then trained using PyTorch. The network consists of five fully connected layers with sizes 256, 128, 64, 32, and a final layer corresponding to the number of tags. Each layer includes batch normalization and a Leaky ReLU activation function, with a 0.5 dropout applied after each activation to prevent overfitting.

The trained FFNN is integrated into the HMM inference function, which can use greedy, beam, or Viterbi algorithms. When an unknown word appears during inference, the word is processed by the FFNN to predict a tag. This prediction is then combined with the HMM model by setting a deterministic emission probability, ensuring that the probability of the unknown word given the classified tag equals to one. This approach keeps the inference time efficient while ensuring accuracy in POS tagging decisions. The overall accuracy rate of this FFNN model is around 83.89%.

4 Smoothing

In our part-of-speech tagging system, we implemented the Add-One smoothing technique within the Hidden Markov Model framework. This approach addresses the zero-frequency problem, ensuring that events not observed during training are not entirely discounted during testing. Specifically, by adding a fixed constant equals to every count of n-grams, Add-One smoothing avoids having any probability value as strictly zero. This is crucial because, without this adjustment, the model might be excessively inflexible, failing to accommodate any unseen events or transitions in the test data. However, Add-One smoothing could lead to a loss of information. If the model encounters a specific transition in the test data that was absent in the training set, Add-One smoothing would still give it a non-zero probability, potentially misrepresenting its actual likelihood. Moreover, choosing an inappropriately large constant can lead to over-smoothing, which can dilute the significance of the training data, leading to suboptimal predictions. However, despite these challenges, it ensures that the model remains adaptable, promoting more accurate and generalized performance on new, unseen data by preventing over-reliance on the training set.

5 Implementation Details

While the Viterbi algorithm tends to be more time-consuming compared to other inference methods, it has been optimized for enhanced computational efficiency. By employing backpointers, the algorithm quickly identifies the most probable tag sequence through optimal tracking of previous states. Additionally, it utilizes NumPy to perform vectorized operations, which are markedly faster than traditional Python loops. The pre-computation of word indices prior to the main algorithmic execution further refines the process by removing redundant look-ups.

6 Experiments and Results

Test Results Within our Hidden Markov Model framework, we integrated the Viterbi algorithm for sequence decoding and added a feed-forward neural network classifier with 83.89% accuracy to manage unknown words. By employing bigram statistics and applying add-one smoothing, we enhanced prediction accuracy for new word sequences. As a result, our model achieved an approximate F1 score of 96.06% on the test dataset. The accuracy on the development set and time efficiency statistics can be found in Table 6.

Smoothing The results of different smoothing techniques applied to a trigram context combined with the Viterbi algorithm are presented in Table 3. The token accuracy for the Add-One smoothing surpasses that of both the models employing linear interpolation smoothing and the no-smoothing methods. The accuracy rates for unknown tokens remain consistent across all conditions which can be attributed to our approach in the unknown word handling section, where we assigned a predetermined emission probability to each unknown word. While the no-smoothing method might excel at handling known words when compared to linear interpolation, it could falter with unseen words. However, this discrepancy isn't necessarily highlighted by our method of handling unknown words. As for the underperformance of linear interpolation relative to Add-One smoothing, We believe linear interpolation's relative underperformance may be due to its suboptimal tuning for this dataset, causing imbalanced n-gram probability weights.

Unigram vs. Bigram vs. Tri-gram vs. Tetragram In our experiment presented in Table 4, we evaluated token accuracy across n-grams using the Viterbi algorithm with Add-One smoothing. The ranking in accuracy was: bigram, tetragram, trigram,

	Token Accuracy	Unknown Token Accuracy	Whole Sentence Accuracy
Add-One	95.46%	83.89%	37.75%
Linear Interpolation	92.75%	83.89%	21.48%
None	95.10%	83.89%	35.03%

Table 3: Accuracy under Different Smoothing Using Trigram Context and Viterbi

and unigram. Bigrams might offer an ideal trade-off between context and flexibility, while tetragrams and trigrams could risk overfitting with too specific contexts, especially on limited datasets. Unigrams, lacking context, naturally exhibited the lowest accuracy. This hierarchy underscores the significance of choosing the appropriate n-gram length based on the dataset and task. Unigrams, on the other hand, lack contextual information altogether, relying solely on individual token probabilities, which could explain their lower accuracy. This hierarchy in performance underscores the significance of choosing the appropriate n-gram length based on the dataset and task.

	Unigram	Bigram	Trigram	Tetragram
Performance of Time Efficiency				
Inference Runtime (Minutes)	0.1537	1.082	0.4169	0.1592
Probability Estimation Runtime (Minutes)	0.1434	0.1402	0.1401	0.1389
Performance of Prediction Accuracy				
Whole Sentence Accuracy	15.52%	39.87%	37.75%	33.39%
Mean Probability (Log Probability)	-2207.15	-3440.79	-4630.11	-6407.83
Token Accuracy	90.14%	95.82%	95.46%	94.65%
Unknown Token Accuracy	83.89%	83.89%	83.89%	83.89%

Table 4: Time Efficiency and Accuracy across Different Number of Grams using Viterbi Inference Algorithm and Add-One Smoothing

Greedy vs. Viterbi vs. Beam When comparing the performance of our model using Add-One smoothing and bigram context, we evaluated various decoding methods: greedy decoding, Beam Search (with k = 2, 3, 20), and the Viterbi algorithm, as shown in Table 5. The greedy decoding method, despite being the fastest, demonstrated the lowest whole sentence accuracy. For instance, it failed to predict correct tags for all words in the sentence: "The British Petroleum Co. PLC said its BP Exploration unit has produced the first oil from its Don oilfield in the North Sea. In an official release, BP said initial production from the field was 11,000 barrels a day, and that it expects peak output from the field of 15,000 barrels a day to be reached in 1990." Moreover, this method also registered the lowest token accuracy.

The whole sentence and token accuracy increased while increasing the beam width. Using Beam Search with a width of k=2 led to a 2.18% improvement in whole sentence accuracy and a 0.30% increase in token accuracy over the greedy method. Broadening the beam width to k=3 added another 1.76% to the whole sentence accuracy and 0.27% to token accuracy. However, as the beam width increased, the performance gains started to diminish, underscoring the importance of balancing beam size with computational efficiency and accuracy.

Compared to the bigram greedy decoder, which had 5.41% sub-optimal results, the Viterbi algorithm recorded none. Even with trigrams, the greedy decoder yielded more sub-optimal results relative to the trigram Viterbi algorithm. Overall, the Viterbi algorithm boasted the highest accuracy for both token and whole sentence, requiring about one minute for inference.

	Greedy	Beam Search (k = 2)	Beam Search (k = 3)	Beam Search (k = 20)	Viterbi
Inference Time (min)	0.1626	0.2595	0.3244	2.4998	1.0820
Probability Estimation Time (min)	0.1422	0.1359	0.1478	0.1388	0.1402
Whole Sentence Accuracy (%)	34.67	36.85	38.61	39.70	39.87
Token Accuracy(%)	95.06	95.36	95.63	95.79	95.82
Unknown Token Accuracy (%)	83.89	83.89	83.89	83.89	83.89

Table 5: Performance Comparison: Greedy, Beam Search, and Viterbi

7 Analysis

Error Analysis Table 5 lists the top five tag categories that most frequently suffer from misclassifications. These misclassifications underline the intricate nature of the English language and the inherent challenges in accurately modeling it. One of the primary challenges is the high degree of polysemy in English, where a word can adopt multiple parts-of-speech based on its context. This makes definitive tagging a formidable task. For instance, the word "silver" can function as both a noun and an adjective, while "record" can be a noun or a verb, depending on its usage.

Morphological similarities further complicate classification. Words such as "traded" and "listed"

can easily be misclassified due to shared suffixes with other parts-of-speech tags. Compound words and phrases introduce another layer of complexity. For example, “labor-management” typically functions as a noun but can also be used as an adjective. The term ”Airline” can represent either a proper noun or a common noun.

Inflectional variations present their own set of challenges. Words like “buying” can be classified as either nouns or verbs, while “valued” can be interpreted as a verb or an adjective. Furthermore, the misclassification of proper nouns frequently occurs. Words that are capitalized, such as “Flexible,” can be mistakenly tagged as common nouns or adjectives. Conversely, articles like “a” might be misconstrued as proper nouns in certain contexts.

Positional context also plays a significant role in errors. Depending on its position in a sentence, the word “worth” might be classified as an adjective or a preposition, and “close” can be tagged as either an adverb or a verb.

To mitigate these challenges, one potential solution is to refine the model. By employing a larger context window and integrating more syntactic and semantic information, the model’s capacity to disambiguate words based on context can be significantly enhanced.

True Tag	Sample Tokens with Predicted Tags	Misclassifications Rates
NN	silver(JJ), record(JJ), labor-management(JJ) Airline(NNP), buying(VBG), original(JJ), Investment(NNP), public(JJ), total(JJ), Business(NNP)	18.08%
JJ	traded(VBN), listed(VBN), held(VBN), such(PDT), sure(RB), such(PDT), dissatisfied(VBN), worth(IN), equal(VBP)	12.49%
RB	as(IN), together(RP), further(RBR), close(VB), back(RP), First(NNP), First(NNP), because(IN), earlier(RBR)	8.93%
VBN	reported(VBD), proposed(JJ), reduced(JJ), estimated(JJ), patched(JJ), ended(VBD), valued(VBD), estimated(JJ), called(VBD), lowered(VBD)	8.26%
NNP	Closed(JJ), End(NN), Bond(NN), Flexible(JJ), Portfolio(NN), Convertible(JJ), A(DT), Institutes(NNPS), O’Dwyer’s(NNS), The(DT)	7.04%

Table 6: Top 5 Part-of-Speech Tags with the Highest Misclassification Rates

Confusion Matrix Tags that are closely related are picked here to form the confusion matrix (Figure 2). The misclassification of part-of-speech tags in English, such as confusing past tense verbs for past participles, nouns for adjectives, or proper nouns for common nouns, can arise due to the inherent ambiguity and flexibility of the English language. Many

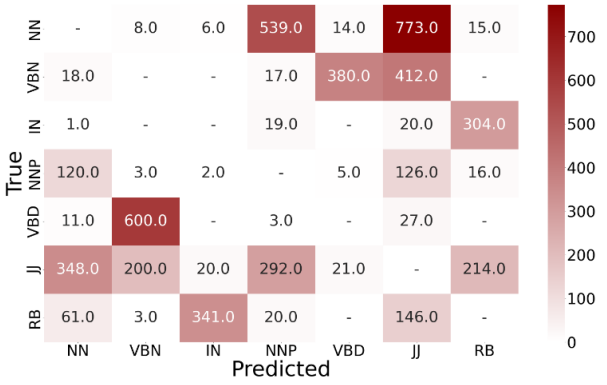

Figure 2: Confusion Matrix of Related Tags

words can function in multiple grammatical roles depending on context. Without accurate contextual understanding or sufficiently diverse training data, POS taggers might struggle to discern the subtle distinctions.

8 Conclusion

In conclusion, our study provided a thorough examination of methods to enhance POS tagging accuracy, showcasing the efficacy of merging contemporary deep learning techniques with conventional models like the Hidden Markov Model. By experimenting with various smoothing techniques, n-gram contexts, and inference methods, we brought the theoretical models discussed in class to practical application. Our results underscored the significance of contextual understanding and highlighted the innate challenges posed by the English language. Moving forward, there’s potential for further advancements by refining these techniques, expanding the context window, and integrating deeper syntactic and semantic information

9 Reference

1. Vadas, D., & Curran, J. R. (2007, June). Adding noun phrase structure to the Penn Treebank. In Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (pp. 240-247).
2. Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank.
3. Alphabetical list of part-of-speech tags used in the Penn Treebank Project. Penn Treebank P.O.S. tags. (n.d.). https://www.ling.upenn.edu/courses/Fall2003/ling001/penntreebankpos.html