Exploring Part of Speech Tagging: Evaluating Hidden Markov Models with Viterbi Decoding

Chelsey Rush
University of California, Santa Cruz
NLP 201 HW3
chrush@ucsc.com

1 Part 1

1.1 Why does $argmax_t P(t|w) = argmax_t log(P(t,w))$?

Rewriting the probability of a tag t given word w in terms of Bayes' theorem would results in the following expression.

$$P(t|w) = \frac{P(t,w)}{P(w)}$$

When maximizing this expression with respect to t, the denominator $P(w)$ is a constant and can be excluded, which gives:

$$argmax_t P(t|w) = argmax_t P(t,w)$$

Taking the logarithm of probabilities is a common mathematic trick which provides numerical stability, simplifies multiplication to addition, and provides easier integration for machine learning algorithms. Taking the log of the previous equation then gives:

$$\therefore \arg\max_t P(t|w) = \arg\max_t \log P(t,w)$$

1.2 For every value of t_j, let $\pi_j(t_j)$ be the log probability of the highest scoring tag sequence of length j ending in tag t_j. Show that $\pi_j(t_j)$ can again be computed using $\pi_j - 1$.

The log probability of the highest scoring tag sequence of length j and ending in tag t_j can be computed by finding the log probability of the highest scoring sequence up to position $j-1$. The log probability of the highest scoring sequence up to that point can be represented with the expression $\pi_{j-1}(t_j - 1)$.

Then, we can find the contribution of the score that results from transitioning to t_{j-1} to t_j. Separating the scores of the previous tags and positions from the contribution of the score to the final tag in terms of the previous tag can be represented then as:

$$\pi_j(tj) =$$
$$\max_{t_{j-1}}(\pi_{j-1}(t_{j-1}) + score(w, j, t_j, t_{j-1}))$$

1.3 Give an algorithm for computing $\hat{t}$ defined in the last line of Eq.(1). What is the time complexity of the algorithm. Be sure to carefully argue your answer.

In order to compute a predicted tag, $\hat{t}$, we need to maximize the probability of the tag given the previous tags, given the previous words. Incorporating the chain rule of probability this means we can try to maximize: $\prod_{t=1}^{T} P(w_t|t_t)P(t_t|t_{t-1})$. Where T is the length of the sequence.

- 1. Begin with the first word in the sentence, where $\pi(t,i)$ is the maximum probability of the tag sequence ending with tag t at position i in the sequence. The probability of tag t at position 1 is then: $\pi(t,1) = P(w_1|t)P(t| < START >)$.
- At each step through the sequence, keep a record of the tag t' which maximizes the probability of the tag sequence.
- At the final step, compute the tag which maximizes the probability of the $< STOP >$token: $\pi(< STOP >, T) = P(t,T)P(< STOP > |t)$.
- Then, trace backwards from the $< STOP >$ to $< START >$ tokens, using the log of best tags at each time step to construct the tag sequence $\hat{t}$.
- The time complexity for this algorithm would be $O(T \cdot K^2)$ where K is the number of different tags we could assign to each word. The

	Micro-Avg F1	Macro-Avg F1
Dev Set	0.90	0.57
Test Set	0.91	0.65

Table 1: Micro and macro average F1 scores of the POS tagger on the development and test sets.

most time consuming part of this algorithm is step two, which needs to compute the probability of each of the tags, and find the one which maximizes our equation.

1.4 Which semiring properties did you use in your proof? Describe the changes needed to your algorithm in question 2 to solve Eq. (2).

After factoring out the final term in the product shown in Eq. (2), we are given:

$$\pi_j(t_j) = \oplus_{t_{j-1}}(\pi_{j-1}(t_{j-1}) \otimes score(w, j, t_j, t_{j-1})).$$

This means that we can find the tag which maximizes the probability of the tag sequence relying only on $\pi_{j-1}(t_{j-1})$ and the score from the transition from the previous tag to the current tag.

The $\oplus$ and $\otimes$ semiring's **associativity property** asserts that when summing or multiplying (separately) over possible tag sequences, those sums or products can be broken down into parts without affecting the final calculation.

And finally, the **distributive property** of $\otimes$ over $\oplus$ means that we can distribute the summation over the product such that $a(b * c) = ab + ac$.

2 Hidden Markov Model with Viterbi Decoding

The following section reports and evaluates the performance of a Part of Speech (POS) Tagging Hidden Markov Model (HMM) with Viterbi decoding.

The HMM uses add-α smoothing and estimates parameters from the training set using maximum a posteriori (MAP) estimation. The Viterbi decoder implemented add-α smoothing with $\alpha = 1$ and assigned the most frequent tag for each word.

The accuracy of the POS tagger on the development and test sets are shown below in Table 1.

Furthermore, the hyper-parameter α was fine-tuned using the development set. The performance

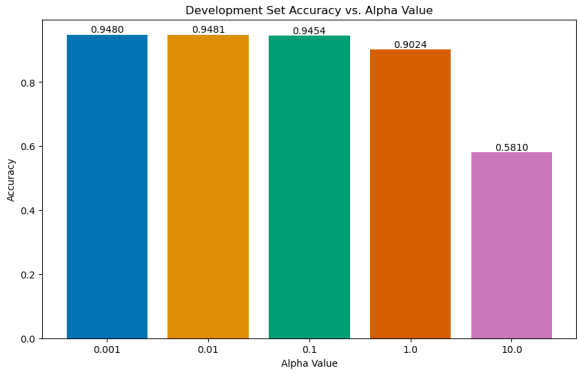

Figure 1: Micro-average F1 scores of the POS Tagging models on the development set with hyperparameter alpha set to 0.001, 0.01, 0.1, and 10.

of the POS HMM when α, the smoothing parameter, was set to 0.001, 0.01, 0.1, 1.0, and 10 was evaluated. The performance of the model for each value of alpha is shown in Figure 1. The models with $\alpha < 1$ performed better than those where $\alpha > 1$. **The best according to the fine-tuning experiments is $\alpha = 0.01$ which achieved an accuracy of 0.948** on the development set, narrowly edging out $\alpha = 0.001$ which achieved 0.948 and $\alpha = 0.1$ which achieved 0.945.

The model's performance against the test set with tuned hyperparameter $\alpha = 0.01$ is summarized in Table 2. The confusion matrix for this model is shown in Figure 2. The hyperparameter-tuned model outperformed the baseline model, increasing the accuracy of predicted tags on the test set by 4%. However, the improvement in overall accuracy did not hold for class accuracy, the macro-average F1 score of the tuned model was lower than the baseline model by 7%. Although this model may improve over all accuracy, it is less able to accurately predict tags for less common POS classes.

	Micro-Avg F1	Macro-Avg F1
Baseline	0.91	0.65
Alpha-Tuned	0.95	0.58

Table 2: Micro and macro average F1 scores of the baseline POS tagger and the model with tuned hyperparameter $\alpha = 0.01$ on the test set.

The most common misclassification for the 'NN' or *noun singular* class was 'NNP' (*proper noun singular*) with 291 incorrect predictions and 'NNS' (*noun, plural*) with 203 incorrect predictions.

One example sentence which resulted in an im-

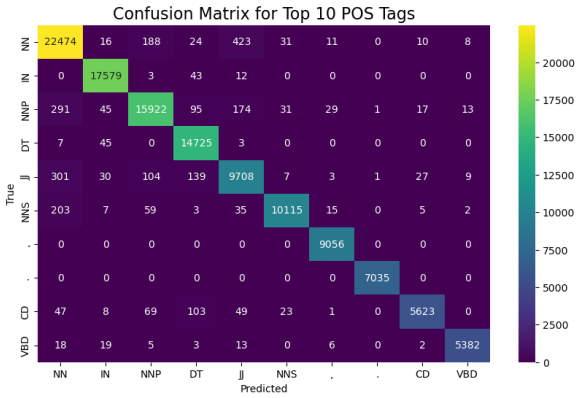

Figure 2: Confusion matrix for the top 10 POS tags of the alpha-tuned model when evaluated on the test set.

proper classification of an 'NN' tag as 'NNP' was is shown below. The word '*mortgage*' is improperly tagged as a proper noun instead of a singular noun, most likely because it occurs in the sentence-initial position, and our HMM model has learned that capitalization is a key indicator of being a proper noun 'NNP'.

```
Mortagage[True:NN, Predicted:NNP]
securities gave up most of Friday's
gains as active issues ended 24/32
to 30/32 point lower.
```

Another example of the same misclassification is shown below, where the HMM POS tagger predicted the tag of '*S&P*' as 'NNP', but the true label was 'NN'. This case is more interesting, because the *S&P 500* is actually a proper noun which refers to the name of a specific stock market index.

```
When the Merc imposed its first halt
in the S&P[True:NN, Pred:NNP] 500
futures contract after the 12-point
drop , the stock index on which [sic]
```

This could be a case in which the data is improperly annotated, or that myself and the annotator disagree on the proper POS tag for the utterance. Examples such as these demonstrate that in addition to our model's capacity for performance, **we must also understand the ambiguities that are inherent in POS tagging based on decisions and assumptions already inherent in the data** as a result of being annotated by humans with differing viewpoints or biases.

3 Improvements

3.1 What can you do to improve the performance of your tagger?

There are multiple ways to begin approaching improving the performance of the tagger. In my opinion, the most pressing challenge is the model's lack of semantic insight into the word sequences. To improve the model's ability to predict POS tags, we might first incorporate pre-processing techniques such as lowercase normalization additional features such as word embeddings. Combining these techniques would likely mitigate the model's assumption that capitalized nouns should always be tagged as 'NNP'.

In addition, improving the model's ability to predict less frequent tags would likely improve the model's macro-average F1 score. To do this, one would need to increase the number of training samples available for less frequent classes. This could be done using oversampling of the rare classes or the undersampling of the more common classes.

3.2 What improvements could make the tagger run faster?

- **Pre-compute log probabilities**. This is a technique I already implemented in my code in order to speed up the decoding process. By precomputing each log probability once and storing it in a dictionary, it is then only necessary to calculate the log probabilities once, as opposed to computing and summing $math.log()$ for the emission and transition probabilities for each word at run time.
- **Using NumPy arrays or vectors.** My tagger currently uses nested dictionaries to store the values for the dynamic programming aspect of the Viterbi algorithm. Changing these to NumPy arrays or vectors would likely speed up the training and decoding steps.
- **Pruning.** Implementing some kind of pruning to reduce the search space during decoding would likely drastically reduce the time needed for the algorithm to complete. A beam search would be an excellent method for this, I believe.