NLP 201 Assignment 3

Steven Au

December 2023

1: Viterbi Algorithm

Question 1

Because the logarithm is a monotonically increasing function, it preserves the order of the probabilities. Thus,

$$\text{argmax}_t P(t|w) = \text{argmax}_t \log P(t,w)$$

for any probability distribution P(t,w).

The `argmax` operation is looking for the value of t that maximizes the given expression. Taking the log of a probability does not change the location of the maximum because the logarithm is a monotonically increasing function. This means that if $P(t|w)$ is maximized at some point, $\log P(t,w)$ is also maximized at the same point. Therefore, the two expressions are equivalent in terms of finding the argument that maximizes them.

Question 2

For every value of j, let $\pi(j)$ be the log probability of the highest scoring tag sequence of length j ending in tag i:

$$\pi(j) = \max_{t_{j-1}} [\pi(j-1) + \text{score}(w, i, t_{j-1})]$$

This recursive formula allows us to compute $\pi(j)$ efficiently by storing the values of $\pi(j-1)$ as we iterate through j. This prevents redundant calculations, which is the essence of dynamic programming.

Question 3

Once the base case $\pi(0)$ is established, we compute the values for subsequent positions. For $\pi(1)$, the computation considers all possible transitions from the start state to each possible tag at position 1:

$$\pi(1) = \max_{t_0} [\pi(0) + \text{score}(w, 1, t_0)]$$

Here, $\text{score}(w, 1, t_0)$ represents the score of transitioning from the start state t_0 to the tag at position 1 given the observation w at that position. Similarly, for $\pi(M)$, which is the last position in the sequence, we consider all possible transitions from the second-to-last position to the end:

$$\pi(M) = \max_{t_{M-1}} [\pi(M-1) + \text{score}(w, M, t_{M-1})]$$

In this case, $\text{score}(w, M, t_{M-1})$ represents the score of transitioning to the end of the sequence from the tag at position $M - 1$.
Each $\pi(j)$ is computed using the maximum score from the previous position, which ensures that we are building the most probable sequence of tags given the sequence of observations w.

The time complexity of the Viterbi algorithm can be analyzed by considering the computations required for each step of the sequence. For each position j ranging from 1 to M, where M is the length of the sequence, the algorithm performs the following computation for each of the N possible tags:

$$\pi(j) = \max_{t_{j-1}} [\pi(j-1) + \text{score}(w, j, t_{j-1})]$$

Here, the maximization operation is carried out N times for each j, once for each possible tag at position $j - 1$. Therefore, for each j, there are $N \times N$ operations, considering all pairs of previous and current tags. Summing over the entire sequence, the total number of operations becomes N^2 for each j, multiplied by the sequence length M, resulting in a total time complexity of $O(N^2 \cdot M)$.

Question 4

For the Viterbi semiring $S = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, where R is the set of real numbers, $\oplus$ is the max operation, $\otimes$ is the addition operation, $\mathbf{0}$ is $-\infty$ (representing the identity element for $\oplus$), and $\mathbf{1}$ is 0 (representing the identity element for $\otimes$).

The Forward algorithm sums over all possible paths, using addition for probabilities and multiplication for transitions and emissions. With the Viterbi semiring, you change the summation to maximization to find the most likely path and the multiplication to addition to use log probabilities).

The equivalence for the Viterbi semiring to the Forward algorithm is because a sum-product algorithm can be turned into a max-sum algorithm by changing the operations. Both algorithms would then compute the same thing: the score of the best path.

Let's show this with the recursive formula. In the Forward algorithm, the recursion is:

$$F(j) = \sum_{t_{j-1}} F(j-1) \cdot \text{score}(w, i, t_{j-1})$$

In the Viterbi algorithm, using the Viterbi semiring, the recursion becomes:

$$V(j) = \max_{t_{j-1}} [V(j-1) + \text{score}(w, i, t_{j-1})]$$

Here, $F(j)$ represents the sum of probabilities of all paths up to position j, and $V(j)$ represents the maximum score (log probability) of the best path up to position j.

Because of the properties of logarithms (logarithm of a product is the sum of logarithms, and logarithm turns ratios into differences), the use of log probabilities turns products into sums, which aligns with the semiring operations. Therefore, using the Viterbi semiring effectively transforms the Forward algorithm into the Viterbi algorithm.

The Forward algorithm using the Viterbi semiring becomes:

$$F(j) = \bigoplus_{t_{j-1}} F(j-1) \otimes \text{score}(w, i, t_{j-1})$$

Where $\bigoplus$ is the max operation and $\otimes$ is addition. The equivalence is shown by:

$$V(j) = \max_{t_{j-1}} [V(j-1) + \text{score}(w, i, t_{j-1})]$$

Both $F(j)$ and $V(j)$ compute the score of the best path, making the algorithms equivalent under the Viterbi semiring.

2: Programming: Hidden Markov Model

Since we are provided starter code, I figured out the functions that were given. The starter code gave preprocessing functions, data split and the evaluate function. I needed to call my functions to main that accept the format from 'load_treebank_splits.' The evaluate functions is used towards the end, so I needed to focus on my 'process_training_data' and 'add_alpha_smoothing' functions. My processing functions takes in the tokenized training data appending START token to the front and the STOP token to the end of sentence. The tokens are added to the transition table for their respective probability of each tag and position.The emission table is created through the same function as we iterate through the same token to create the likelihood of lexical tokens being generated from POS tags.

The alpha smoothing functions is applied to both the transition and emission table. We calculate the total sum if their respective tables to get the alpha smoothing and then apply the alpha smoothing for all tags. Both the table run through the same loop.

transition_table[prev_tag][tag] 0

$$P(\text{tag}|\text{prev\_tag}) = \frac{\text{Count}(\text{prev\_tag}, \text{tag}) + \alpha}{\text{Total Transitions from prev\_tag} + \alpha \times \text{Total Tags}}$$

emission_table[tag][word]

$$P(\text{word}|\text{tag}) = \frac{\text{Count}(\text{tag}, \text{word}) + \alpha}{\text{Total Emissions from tag} + \alpha \times \text{Total Tags}}$$

3: Programming: Viterbi Algorithm

The core of the POS tagger is the Viterbi algorithm function to find the most probable sequence of Part-of-Speech (POS) tags for a given sentence, using a Hidden Markov Model (HMM). It initializes a Viterbi matrix to store probabilities and a backpointer matrix to keep track of the most probable paths. For each word in a sentence, the function calculates the most probable tag sequence using dynamic programming.

The code uses two different alpha scores. Alpha defined in the beginning of the code controls the viterbi function, while alpha in main refers to the creation of emission and transition tables.

0.1 Initialization

The first step initializes the probabilities for the first observation in the sequence. For each state s in the set of states:

$$V_0[s] = \log(\text{start\_prob}[s]) + \log(\text{emission\_prob}[s][\text{observations}[0]]) \tag{1}$$

Where:

- $V_0[s]$ is the log probability of state s being the start state.
- start_prob[s] is the probability of the sequence starting with state s.
- emission_prob[s][observations[0]] is the probability of state s emitting the first observation.

0.2 Recursion

For each subsequent observation t (from 1 to the length of observations), and for each state s in the set of states:

$$V_t[s] = \max_{s' \in \text{states}} (V_{t-1}[s'] + \log(\text{transition\_prob}[s'][s]) + \log(\text{emission\_prob}[s][\text{observations}[t]])) \tag{2}$$

Where:

- $V_t[s]$ is the log probability of the most probable state sequence ending in state s at time t.

- transition_prob$[s'][s]$ is the probability of transitioning from state s' to state s.
- emission_prob$[s]$[observations$[t]$] is the probability of state s emitting the observation at time t.

0.3 Termination

The final step involves finding the state with the highest probability at the last time step:

$$\text{final\_state} = \arg\max_{s \in \text{states}} V_{\text{len(observations)}-1}[s] \tag{3}$$

$$\text{final\_probability} = \max_{s \in \text{states}} V_{\text{len(observations)}-1}[s] \tag{4}$$

Where:

- final_state is the state with the highest probability at the last observation.
- final_probability is the probability of this state sequence.

0.4 Path Backtracking

The algorithm also keeps track of the path of states that leads to each $V_t[s]$ for backtracking the most probable state sequence at the end.

4: Evaluation

POS Tagger Results

Train set size: 51681
Dev set size: 7863
Test set size: 9046

Alpha = 1
Viterbi Tags: ['“', 'EX', 'MD', 'RB']
Baseline Tags: [('Your', 'PRP$'), ('test', 'NN'), ('sentence', 'NN'), ('here', 'RB')]
POS Tagger Accuracy on Dev Set: 72.83%
POS Tagger Accuracy on Test Set: 73.91%
Alpha = 0.1
Viterbi Tags: ['PRP$', 'NN', 'MD', 'RB']
Baseline Tags: [('Your', 'PRP$'), ('test', 'NN'), ('sentence', 'NN'), ('here', 'RB')]
POS Tagger Accuracy on Dev Set: 78.20%
POS Tagger Accuracy on Test Set: 79.21%

Alpha = 1e-6
Baseline Tagger Accuracy on Dev Set: 91.50%
Baseline Tagger Accuracy on Test Set: 91.70%
POS Tagger Accuracy on Dev Set: 91.56%
POS Tagger Accuracy on Test Set: 92.04%

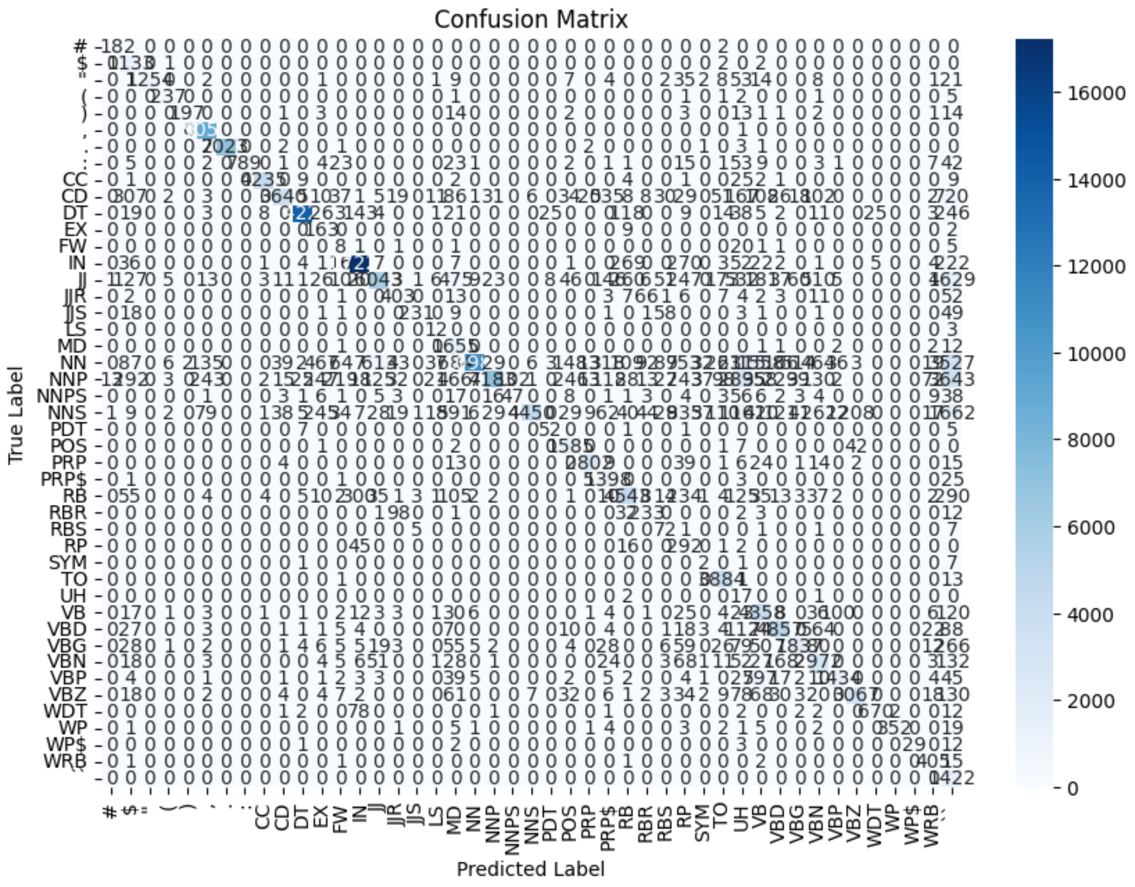

Figure 1: Confusion Matrix of Viterbi Tagger.

Inference

The dual tags are omitted for simplicity of the classification report and the confusion matrix. All the dual tags were never predicted due to being very rare in the dataset and ambiguous.

Having a lower alpha score gives less probability for words that are not in the test set. When reducing alpha the accuracy increases because when assigning values to new words they have a higher probability due to a high alpha smoothing as opposed to a lower one. This causes a increase in accuracy. Normally 1e-6 is standard for alpha and gave a 92% test accuracy. The code takes 30 minutes to run and alhpa test was not done extensively due to computations times. A high alpha scores reduces the more probable tags if new words are ignored. Smoothing reduces all the probabilities to consider OOV and increase the chances to mislabel tags. When alpha = 1, the test sentence is mislabeled for "the" as double quotes as opposed to PRP$.

The confusion matrix shows a huge density around NN variations tags and CD and JJ. The words that are ambiguous to tag are more prone to be mislabeled as we look at states. CRF allows us to look at tag dependency and reduce this classification. This explains why our HMM model does not do well with ambiguous tags for adjectives and cardinal directions, as they can be nouns and pronouns respectively. For example yellow is both an adjective and a noun. One can be used as an adjective, or a proper nouns for "one's"

The HMM Tagger performs pretty well with alpha smoothing as opposed to the baseline tagger. The baseline tagger is a simple heuristic that capture the most common usage of a word as opposed to its context. The baseline performs well on simple sentences due to the large training set. Surprisingly, the baseline performs almost as well as the HMM tagger, which is more complex and slower.

To improve the model's accuracy, we can increase the dateset, include more context, or implement conditional random field (CRF) for POS logic.CRF allows us to capture more of the POS structure that HMM did not observe.

To improve the performance, we can add parallel processing and batching to make the code run faster for larger datasets. Viterbi is n cubed time.

Label	Precision	Recall	F1-Score	Support
#	0.76	1.00	0.86	22
$	0.59	1.00	0.74	1138
”	1.00	0.95	0.97	1423
(	0.92	1.00	0.96	249
)	0.99	0.94	0.97	252
,	0.98	1.00	0.99	9056
.	1.00	1.00	1.00	7035
:	1.00	0.94	0.97	983
CC	0.99	0.99	0.99	4289
CD	0.99	0.70	0.82	6023
DT	1.00	0.96	0.98	14946
EX	0.18	0.99	0.30	174
FW	0.01	0.34	0.02	38
IN	0.97	0.94	0.96	18147
JJ	0.86	0.66	0.75	10704
JJR	0.65	0.77	0.71	581
JJS	0.95	0.81	0.87	374
LS	0.14	0.80	0.24	15
MD	0.56	0.99	0.71	1674
NN	1.00	0.56	0.71	23468
NNP	0.99	0.50	0.67	17236
NNPS	0.25	0.47	0.32	239
NNS	1.00	0.56	0.72	10697
PDT	0.48	0.86	0.62	66
POS	0.79	0.99	0.87	1638
PRP	0.98	0.98	0.98	2930
PRP$	0.65	0.99	0.78	1433
RB	0.84	0.82	0.83	5853
RBR	0.37	0.72	0.49	382
RBS	0.12	0.90	0.22	87
RP	0.10	0.91	0.19	356
SYM	0.21	0.82	0.33	11
TO	0.93	1.00	0.96	3899
UH	0.00	0.85	0.00	20
VB	0.71	0.92	0.80	4766
VBD	0.91	0.86	0.89	5869
VBG	0.73	0.84	0.78	2592
VBN	0.69	0.86	0.77	3580
VBP	0.85	0.77	0.81	2209
VBZ	0.89	0.92	0.91	3608
WDT	0.89	0.95	0.92	773
WP	0.99	0.96	0.98	397
WP$	1.00	0.91	0.96	47
WRB	0.73	0.99	0.84	425
“	0.14	1.00	0.24	1422
accuracy			0.79	171138
macro avg	0.48	0.59	0.49	171138
weighted avg	0.92	0.79	0.83	171138

Table 1: Classification Report