Interpolation	Add-K (prior, emission)	OOV cutoff	Accuracy
0.90 0.10	0.1 0.1	10	90.19%
0.90 0.10	0.05 0.5	5	91.59%
0.90 0.10	0.1 0.1	5	91.54%
0.85 0.15	0.05 0.05	2	92.89%
0.90 0.10	0.05 0.05	2	92.88%
0.95 0.05	0.05 0.05	2	92.81%
0.99 0.01	0.05 0.05	2	92.78%

Table 1: Bigram HMM Tagger Accuracy

Question 1.1 Bigram HMM

In the bigram HMM model, we handled OOV symbols in the emission model by classifying it based on its word form into *MENTION*, *HASHTAG*, *URL*, *ALLCAP*, *CAPFIRST*, *NUMBER*, *EMOJI* and if not any, as *UNK*. The cut-off for what counts as OOV was configurable, although 5 seems to have been a reasonable number.

We used linear interpolation on the transition model for smoothing and add-K smoothing for the emission model and prior models.

From looking at the confusion matrix generated, it seems the greatest source of mistakes were classifications between proper nouns and common nouns, verbs mistaken as proper nouns and between common nouns and verbs. This is most likely due to the treatment of capital letters as part of the word form. This could be explored more, but I am lazy. But the hunch is supported by the fact that if we moved more of the emission symbols into OOV by raising the cut-off, we ended up with more errors in the confusion matrix for those tags. This makes sense as more of the emission symbols were replaced with our word forms.

We tested on different hyper-parameters for the smoothing models and cut-off for OOV (See Table 1. The best results were with *./hmm2 -oov 2 -a1 0.85 -a2 0.15 -ge 0.05 -gp 0.05 -test* for an accuracy of 92.94%.

Question 1.2 Trigram HMM

The trigram joint probability model is:

$$p(x, y) = q(y_0)e(x_0|y_0)q(y_1|y_0)e(x_1|y_1)\prod_{i=2}^{n} q(y_i|y_{i-1}, y_{i-2})e(x_i|y_i) \tag{1}$$

The trigram Viterbi decoding recursion is (here we move to use log probability):

$$\pi(i, y_i, y_{i-1}) = \arg\max_{y_{i-2} \in Y}(\pi(i-1, y_{i-1}, y_{i-2}) + q(y_i|y_{i-1}, y_{i-2})) + e(x_i|y_i) \tag{2}$$

We handled OOV vocabulary in the same way as in the bigram model. The only addition is another interpolation parameter for the trigram and far slower training as the Viterbi algorithm is now $O(N^4 * M)$.

We tried a bunch of hyperparameter choices, of which the best results were with *./hmm3 -oov 2 -a1 0.60 -a2 0.30 -a3 0.10 -ge 0.05 -gp 0.05 -test* for an accuracy of 93.01%. The full table of results can be found in Table 2.

Interpolation	Add-K (prior, emission)	OOV cutoff	Accuracy
0.60 0.30 0.10	0.05 0.05	2	92.96%
0.75 0.20 0.05	0.05 0.05	2	92.91%
0.85 0.012 0.03	0.05 0.05	2	92.85%
0.90 0.09 0.01	0.05 0.05	2	92.83%
0.95 0.049 0.001	0.05 0.05	10	90.23%
0.95 0.049 0.001	0.05 0.05	5	91.56%
0.95 0.049 0.001	0.05 0.05	2	92.78%
0.95 0.049 0.001	0.1 0.1	10	90.18%
0.95 0.049 0.001	0.2 0.2	10	90.07%
0.97 0.029 0.001	0.05 0.05	2	92.77%
0.99 0.009 0.001	0.05 0.05	2	92.77%

Table 2: Bigram HMM Tagger Accuracy

Question 2 Probabilistic Context Free Grammars

This is answers to both part (1) and part (2). SInce the probability distribution is the same and the only way to create a non-terminal X is via the rule $X \to Y$. We have 2^k non-terminals with probability $\frac{1}{2|\Sigma|}$. The probability of a sentence $p(w) = \frac{1}{|\Sigma|^M}$. The probability of $p(t|w)$ is the probability of max probability parse tree given a sentence of terminals with uniform distribution. No matter how this tree is parsed, as long as M is 2^k, then the production rule $X \to XX$ is used $2^k - 1$ times and the nonterminals with probability $\frac{1}{2|\Sigma|}$ are used $2^k = M$ times. Therefore

$$\begin{aligned} p(w) &= \frac{1}{|\Sigma|^M} \\ p(t|w) &= \frac{1}{2}^{M-1} \frac{1}{2|\Sigma|}^{M} \\ &= \frac{1}{2}^{2M-1} \frac{1}{|\Sigma|}^{M} \\ p(t,w) &= p(t|w)p(w) \\ &= \frac{1}{2}^{2M-1} \frac{1}{|\Sigma|}^{M+1} \end{aligned} \tag{3}$$