Homework 2
Abhijit Prakash Bhatnagar

Any thoughts as to why we are reducing the tagset over time? - I realized that the Viterbi matrix requires probabilities for all possible tag permutations. Therefore the larger the tagset, the more time this algorithm takes.

Part 1

Q1. Describe your design decisions on smoothing and OOV handling. Compare the model performance of different design choices on the train and dev sets. Pick your best model and report its performance on the test set.

Design choices:

1. I add both start and stop symbols.
2. I count symbols as words
3. I did not convert cases of words
4. Unking is done at the time of reading the raw data, and not as part of the Ngram code.
5. I unked those words in train data set that appeared once upto a max of 10% of all words
6. I use linear interpolation with add-k smoothing

Bigram model accuracies:

Lambda 1	Lambda 2	K	Train - %	Dev - %
0.9	0.1	0.01	96.4541	90.4734
0.8	0.2	0.01	96.5272	90.4121
0.75	0.25	0.01	96.5393	90.3346
0.7	0.3	0.01	96.5348	90.2257
0.65	0.35	0.01	96.48357	90.1604

Test accuracy of best model: Lambda 1: 0.9, Lambda 2: 0.1, K = 0.01 : 90.5186%

Q2. Error analysis: provide error analysis on the dev set by discussing prominent error cases you may have noticed. Discuss with concrete examples. You can also compute the confusion matrix.

Prominent error cases I observed were:

1. "~ @" being predicted in place of "~ ^"
2. "V G" predicted in place of "V ,"
3. "<S> @" predicted in place of "<S> ^"

Note, I was confused how to generate a proper confusion matrix, and these were observations that I eyeballed on the dev dataset. I understand the right way to identify the top-n prominent error cases would be to look at accuracy and precision of individual tags using a confusion matrix.

Part 2

Q1. Provide the formal equation for the joint probability model.

$$
\begin{aligned}
& P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_{n+1} = y_{n+1}) \\
= & \prod_{i=1}^{n+1} P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \prod_{i=1}^{n} P(X_i = x_i | Y_i = y_i)
\end{aligned}
$$

Q2. Provide the viterbi decoding algorithm by providing the recursion with $\pi(i, y_i, y_{i-1})$. Make sure to handle the base case.

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
Initialization: Set $\pi(0, *, *) = 1$, and $\pi(0, u, v) = 0$ for all (u, v) such that $u \neq *$ or $v \neq *$.
Algorithm:

- For $k = 1 \dots n$,
  - For $u \in \mathcal{K}, v \in \mathcal{K}$,

$$
\begin{aligned}
\pi(k, u, v) &= \max_{w \in \mathcal{K}} (\pi(k-1, w, u) \times q(v|w,u) \times e(x_k|v)) \\
bp(k, u, v) &= \arg\max_{w \in \mathcal{K}} (\pi(k-1, w, u) \times q(v|w,u) \times e(x_k|v))
\end{aligned}
$$

- Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n, u, v) \times q(\text{STOP}|u,v))$
- For $k = (n-2) \dots 1$,

$$y_k = bp(k+2, y_{k+1}, y_{k+2})$$

- **Return** the tag sequence $y_1 \dots y_n$

Note, the above algorithm is derived from the given lecture notes on HMM.

Q3. Describe your design decisions on smoothing and OOV handling. Compare the model performance of different design choices on the train and dev sets. Pick your best model and report its performance on the test set.

Design choices:

1. I add both start and stop symbols.
2. I count symbols as words
3. I did not convert cases of words
4. Unking is done at the time of reading the raw data, and not as part of the Ngram code.
5. I unked those words in train data set that appeared once upto a max of 10% of all words
6. I use linear interpolation with add-k smoothing

Lambda 1	Lambda 2	Lambda 3	K	Train - %	Dev - %
0.5	0.4	0.1	0.01	95.0973%	88.5248%
0.6	0.3	0.1	0.01	94.9411%	88.4812%
0.5	0.35	0.15	0.01	95.3821%	88.5547%
0.5	0.3	0.2	0.01	94.6424%	88.6010%
0.7	0.15	0.15	0.01	94.9765%	88.6010%

Test set accuracy of best model: Lambda 1: 0.5, Lambda 2: 0.3, Lambda 3: 0.2, K = 0.01 : 88.6010 %

Q4. Compare the results with the bigram HMMs. Is the performance, better, worse, or about the same? Provide some commentary (a hypothesis and some evidence; this need not be conclusive) as to why the results are what you found.

Overall, the accuracy on bigram model is higher than that of a trigram model. My hypothesis is that this is so because there is no uniform writing style in the twitter dataset as it is generated by a variety of authors, all using different terminologies. Because tweets are primarily written in shorthand there is less language continuity for trigrams to be more meaningful than bigrams. Even with a very high unk percentage in these test runs, the number of new trigrams encountered in dev and test data set are too high compared to number of new bigram. Therefore, probability of trigrams is lowered.