CS5740: Assignment 3
https://github.com/cornell-cs5740-20sp/assignment-3-as-mk-a3

Mandy Kong
mk2568

Ananya Shivaditya
aks298

1 Introduction (5pt)

In this assignment, we implemented different techniques for part-of-speech (POS) tagging, including inference methods of beam search, greedy decoding, and Viterbi, as well as deleted linear interpolation (DLI) and add-k smoothing. We also experimented with ngram models (n = 2,3). Our POS tagger generates sequences of predicted POS tags corresponding to each word and is evaluated based on the accuracy of the predicted tags. The data being used to build our POS tagger comes from the Penn Treebank.

First, we tested combinations of smoothing methods and inference methods on a trigram model. Next, we compared bigram and trigram models with other parameters kept constant. Based on our experiments, we observed that DLI smoothing produced better F1 scores in comparison to no smoothing and add-k smoothing. Also, increasing the beam size k in beam search did not increase the F1 score significantly. Lastly, we noticed only a slight increase in F1 score on the development set with no smoothing and using Viterbi, when compared to beam search. This was surprising, since we expected the best test score to be from Viterbi, which scored 0.93073 on the leaderboard. Our best accuracy on the development set was 0.93226 using beam search, DLI smoothing, and k = 3.

2 Data (5pt)

The data from the Penn Treebank was separated into training, development, and testing. After loading the data, the data is further separated into individual documents based on the $-DOCSTART-$ tokens. As seen in Table 1, the training data consisted of 696,475 total words and 1,387 individual documents, the development data consisted of 243,021 words and 462 documents, and the test data consisted of 236,582 words and 463 documents. Also listed in Table 1 is change of vocabulary size once we handled rare words. The "Vocab" column lists the original number of unique words found in each data set while the "Tokens" column shows the number of unique words found after tokenizing rare words in the training data and unknown words in the development and test data. Rare words are words in the training data that occurred with frequency < 5, and are mapped to 1 of 7 tokens: {_PUNCS_, _CAPITAL_, _NUM_, _NOUNLIKE_, _VERBLIKE_, _ADJLIKE_, _RARE_}. This led to a drastic decrease in vocabulary size for each data set. By tokenizing words that occur with a frequency < 5, we are able to handle unknown words found in the development and testing data sets. However, as seen from Table 1, we also end up generalizing many words and therefore, lose information relating to those tokenized words.

3 Handling Unknown Words (15pt)

We handled unknown words by tokenizing words that occur with a frequency less than our cutoff (default = 5) in our training data set and training our POS tagger using the mapped tokens instead. The probabilities of these mapped tokens are calculated the same way as the unmapped words. As stated in the previous section, these unknown/rare words are mapped to 1 of 7 tokens: {_PUNCS_, _CAPITAL_, _NUM_, _NOUNLIKE_, _VERBLIKE_, _ADJLIKE_, _RARE_} based on Regex expressions with the word being mapped to $\_RARE\_$ if the previous six expressions did not match the word. As seen in Table 2, we look at various parts of the word to determine which token to map to, such as the first letter, the prefix, and the suffix. The words we mapped do not occur very frequently in the training set and not much information about the relationship of the word to

Data	Words	Docs	Vocab	Tokens
train	696,475	1,387	37,505	9,949
dev	243,021	462	20,705	4,889
test	236,582	463	20,179	4,685

Table 1: Data Statistics

Regex Expr.	Token	train	dev	test
no word character	_PUNCS_	1	3	2
capital letter	_CAPITAL_	14,794	7,729	7,182
digit	_NUM_	5,017	2,188	2,047
-ion, -ty, -ics, -ment, -ence, -ance, -ness, -ist, -ism	_NOUNLIKE_	2,032	861	814
-ate, -fy, -ize, en-, em-	_VERBLIKE_	701	277	263
un-, in-, -ble, -ry, -ish, -ious, -ical, -non	_ADJLIKE_	2,054	807	766
other	_RARE_	19,235	8,112	7,517

Table 2: Unknown/Rare Words Mapping

its matching POS tag would have been obtained. By mapping these rare words to subcategorized tokens, we have increased the number of occurrences in the training data for each mapped token and can learn more about the probability of each token being mapped to a POS tag. Also seen in Table 2 are the counts of mapped tokens for each of the datasets. We included in our POS tagger a *cutoff* parameter that defines the max frequency for which a word is mapped in the training data.

4 Smoothing (10pt)

For our POS tagger, we implemented two types of smoothing techniques. The first smoothing technique implemented was add-k smoothing. Add-k smoothing moves probability mass from seen to unseen events by adding a fractional count $k = (0, 1)$ to each observation and then normalizing. For our trigram model, the original transition probability and emission probability would be calculated using the respective equations:

$$P(y_i|y_{i-1}, y_{i-2}) = \frac{C(y_{i-2}, y_{i-1}, y_i)}{C(y_{i-2}, y_{i-1})} \quad (1)$$

$$P(x|y) = \frac{C(x, y)}{C(y)} \quad (2)$$

When applying add-k smoothing, they would be calculated as follows:

$$P(y_i|y_{i-1}, y_{i-2}) = \frac{C(y_{i-2}, y_{i-1}, y_i) + k}{C(y_{i-2}, y_{i-1}} \quad (3)$$

$$P(x|y) = \frac{C(x, y) + k}{C(y) + k|y|} \quad (4)$$

with k being a parameter that can be set and $|y|$ being the number of possible POS tags. Although add-k smoothing introduces some problems with generating counts of poor variance and incorrect discounts, it also ensures we never have a case of unseen events.

The second smoothing technique we implemented was deleted linear interpolation (DLI) over unigrams, bigrams, and trigrams. The transition probability for a trigram with DLI smoothing is calculated as follows:

$$P(y_i|y_{i-1}, y_{i-2}) = \lambda_1\hat{P}(y_i) + \lambda_2\hat{P}(y_i|y_{i-1}) + \lambda_3\hat{P}(y_i|y_{i-1}, y_{i-2}) \quad (5)$$

where $\hat{P}$ is the MLE of the probabilities. The lambdas were calculated using deleted interpolation where each trigram from the training data is successively removed in order to estimate the best values for the lambdas. The algorithm to calculate the lambdas was based on the pseudocode described in (Brants, 2000). Although we recognize that linear interpolation does not produce very accurate results if the data itself is non-linear, we believe using DLI smoothing with unigrams and bigrams provides a mixture of related, denser contexts that support our probability calculations.

5 Implementation Details (5pt)

Our POS tagger model can take in the following parameters: inference_method, smoothing_method, handle_unknowns, *addk*, *cutoff*, *n*, and *k*. The inference methods are listed in Section 1 while the smoothing methods are listed in Section 4. The parameter, handle_unknowns, denotes whether the model will map words with frequency $<$ cutoff to their subcategorized tokens, and defaults to True with cutoff=5. The parameter *addk* is used with our add-k smoothing method, and defaults to 1.0. The parameter n denotes which ngram model we are using and defaults to 3 for trigrams. Our last parameter, k, denotes the number of beams to search for when using our beam search inference method.

For our inference methods, we used a trellis to keep track of the current π score and backpointers. Our trellis is implemented as two matrices and can be indexed for a pair of tags i and j by doing the following: M[i * num_tags + j, t] where t is the current time step (word) in the sequence. Our π score is implemented in log space to prevent underflow and increase speed. It can be calculated by taking the sum of the logs of equations 1, 2, and the previous π value, if no smoothing is applied. If equation 1 is represented by the variable q and equation 2 is represented by the variable e, the following equation is used to calculate the π score for the current word in the sequence:

$$\pi(i, y_i) = \pi(i-1, y_{i-1}) + math.log(q) + math.log(e) \quad (6)$$

Inference	Smoothing	F1 Score
Viterbi	None	0.93073

Table 3: Test Results

where $\pi(i, y_i) = 0$ if y_i is the start tag. If smoothing is applied, the previous equation is modified by replacing q and e with equations 3 and 4, for add-k smoothing, and replacing q with equation 5 for deleted linear interpolation.

6 Experiments and Results

Test Results (3pt) As seen in Table 3, our best F1 score on the test dataset was 0.93073, which was achieved using Viterbi and no smoothing.

Smoothing (5pt) The results on the development dataset of our POS tagger model with and without smoothing are listed in Table 4. We experimented with each smoothing on all inference methods and observed that no matter the inference method used, using DLI smoothing on our data produced better results overall. For example, with greedy decoding, the F1 scores for no smoothing, add-k of 0.05, and DLI were 0.92865, 0.92960, and 0.93226, respectively. For our add-k smoothing method, by experimenting with the value k being added to each observation, we noticed that a smaller value of k produced better F1 scores. This can also be seen in Table 4 where we tested k = {0.05, 0.1, 0.5} and observed a decrease in F1 scores {0.92960, 0.92788, 0.90868} as we increased k.

Bi-gram vs. Tri-gram (5pt) Using beam search, we experimented with bigrams and trigrams, keeping no smoothing, cutoff=5 and k=3. We found that the bigram model achieves a smaller F1 score (0.88728) compared to the trigram model (0.92639) on the development data. This is expected, as the capacity of the model to hold context increases with increase in n of n-grams. We also notice a dramatic drop of accuracy in tagging unknown words while using a bigram model as seen in Table 5.

Greedy vs. Viterbi vs. Beam (10pt) As seen in Table 4, the results achieved by greedy decoding and beam search are relatively similar with a slightly higher F1 score achieved by beam search when k=2 and no smoothing is done. This leads to the belief that beam size does not affect the accuracy by much. We've also noticed that for greedy decoding and beam search, the overall F1 score may reach 0.90+ but sub-optimal solutions are often found since we rarely get an entire, correctly mapped sequence. In comparison to beam search (k=3), the Viterbi inference method gains a marginal increase in F1 score of magnitude 10^{-4}.

7 Analysis

The following analyses were done on our results with beam search (k=3) and DLI smoothing. This setup performed best on the development set.

Error Analysis (7pt) Table 6 shows the classes of most frequent errors. It appears that adjectives and nouns are most frequently misclassified. Plural nouns have frequently been mistaken as singular or mass nouns. The Wh-determiner is next, often tagged as preposition/conjunction. Also, within the verb class, it is seemingly very difficult to distinguish between past participle and past tense. An example along with the counts of each class is given in table 6, with its index from development set.

Confusion Matrix (5pt) Figure 1 displays erroneous tags from the development set. Some classes of verbs are most severely mistagged, as we see in the central region of the matrix. Nouns and adjectives are often mixed up, as we see closer to the top left. Within nouns, plural nouns have frequently been mistaken as singular or mass nouns, and there is also confusion between past participle and past tense verbs.

8 Conclusion (3pt)

After experimenting with inference methods, smoothing methods, and ngram models, we determined that the two most important factors for improving accuracy of our POS tagger are the smoothing method and the ngram model used. Trigrams work better than bigrams due to the additional context used while smoothing methods help with lack of data for our probability calculations. This can be seen in Tables 5 and 6 where the trigram model achieves its highest F1 score of 0.93226 with DLI smoothing and a higher F1 score than a bigram model (0.92639 vs. 0.88728). From our results we can also determine that our tagger has trouble with nouns and differentiating between past participle and past tense verbs.

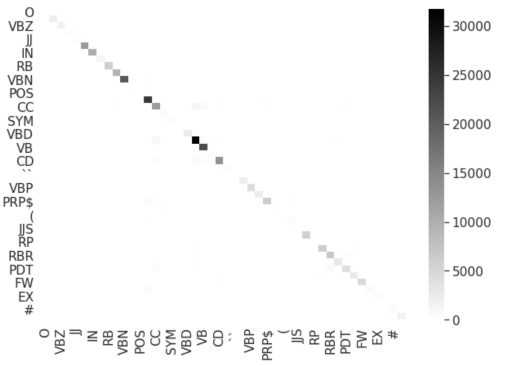

Figure 1: Confusion Matrix

Inference	Smoothing	F1 Score	Unk.
Greedy	None	0.92865	0.63323
Greedy	Add-K (0.05)	0.92960	0.63257
Greedy	Add-K (0.1)	0.92788	0.63082
Greedy	Add-K (0.5)	0.90868	0.61335
Greedy	DLI	0.93226	0.63718
Beam (2)	None	0.92638	0.63322
Beam (2)	Add-K (0.05)	0.92960	0.63257
Beam (2)	DLI	0.93226	0.63718
Beam (3)	None	0.92630	0.63323
Beam (3)	Add-K (0.05)	0.92960	0.63257
Beam (3)	DLI	0.93226	0.63713
Viterbi	None	0.92643	0.63288

Table 4: Development Results (DLI = Deleted Linear Interpolation)

Inference	n-gram	F1 Score	Unk.
Beam (3)	trigram	0.92639	0.63308
Beam (3)	bigram	0.88728	0.17134

Table 5: Development Results comparing bi-grams vs tri-grams

Word	idx	Actual	Predicted	Count
dealer-to-dealer	38	JJ	NN	1178
lockup	177	NN	JJ	1013
Ex-dividend	67	NN	NNP	665
Troubled	2474	JJ	NNP	638
arbs	574	NNS	NN	621
that	1188	WDT	IN	507
proposed	113	VBN	JJ	503
as	285	RB	IN	502
reported	18	VBN	VBD	498
kickbacks	3373	NNS	JJ	436

Table 6: Incorrect POS tags