LING571 – Hw4_ec (Extra Credit)
Handling OOV in PCFG

1. Overview

In this assignment, an attempt was made to make a principled approach to handling Out-of-Vocabulary (OOV) tokens during parsing.

2. Challenges

For the original Hw4, I tried to implement a degree of OOV support. At that time, I attempted to do this in the parser and only with very limited success. I was able to get a parse for all but 2 sentences, however this improvement had a negative effect on accuracy as these parses were all perceived to be erroneous.

This time round, I was really hopefully that a more principled approach to handling OOV would result in an improvement in accuracy from the baseline of 99.04%.

3. Approaches

First Approach

My first approach attempted to substitute **tokens** with the least instances in the corpus. I noticed this caused some surprising changes in the data. Most noticeable was the reduction of entire word classes to, for example, VBD -> '<UNK>' [1.0]. Notice the probability of 100%. This means that, due to a low token count, the entire word class was reduced to a meaningless representation.

My error was I overlooking the fact that an infrequently referenced word might, nevertheless, be very highly conserved in a particular context.

Take the word "better" for instance. As a word, it is not frequently referenced in the corpus of training data; however, it is the only terminal with a LHS of ADJP_JJR. To then assert that all words are equally likely to be ADJP_JJR seemed unreasonable, as it is the only word in the corpus for that non-terminal.

Second Approach

My second approach attempted to substitute the least probable (or prolific) **terminal productions** (rather than the terminals themselves) with UNK productions. If we reconsider the production discussed above where ADJP_JJR -> 'better' became ADJP_JJR -> '<UNK>' when using the first approach. Using the second approach, however, the situation would be handled much better because ADJP_JJR -> 'better' would have a weight of 100%, since all 2/2 references to "better" are associated with this production. So, in this formulation of the problem, the least common productions are removed.

4. Open- vs Closed-Class Words

I chose to follow the above approach because it implicitly handles both open-class and closed-class words well. Due to the prolific nature of open-class words, there's a relatively high probability of encountering an open-class word that is OOV. One would expect that, in the corpus, there would be a similar number of low-frequency open-class words; whereas closed-class words tend to be better represented by virtue of their finiteness.

To illustrate and quantify the point, let's consider the weight (percentage) assigned to UNK word tokens in some well-known word classes according to our model:

Open-Class Word Categories:

Common Nouns:	NN -> '<UNK>'	[0.2673611111111111]
Proper Nouns:	NNP -> '<UNK>'	[0.28728070175438597]

Verbs:	VB -> '<UNK>'	[0.11764705882352941]

Closed-Class Word Categories:

Determiners:	DT -> '<UNK>'	[0.03686635944700461]
Prepositions:	IN -> '<UNK>'	[0.10433070866141732]
Pronouns:	NP_PRP -> '<UNK>'	[0.022222222222222223]

Here you can see that the open-class word categories anticipate a higher proportion of OOV than the closed-class words.

This analysis also reveals what I believe to be biases in the corpus that skew the results. The verb class seems surprisingly conserved with a weight of only 11%. In contrast, the proposition class allocates 10% weight to OOV, which seems high for such a conservative word class. These anomalies are attributed to characteristics of the corpus , which is relatively small; having less than 5000 tokens. (In fact, the entire corpus does not include the words, 'he', 'she', nor 'we'.) Also, the domain is fairly specific, limiting the range of verbs, for example, to only those related to flight and travel.

I am hopeful that this algorithm would work even better with a more eclectic corpus from which to learn.

5. Results

A statistical analysis was run using Evalb, using a modified parser based on the parser used in Hw4.

Setting the hide_proportion to 0.0001 produced identical results to my baseline implementation in Hw4. In the baseline, the number of skip sentences was also 6. I assume that the number of tokens substituted with <UNK> becomes statistically insignificant when selecting such a small percentage of the corpus as UNKs.

The best results were achieved when using a hide_proportion of 1%. Under these conditions, an overall accuracy of 98.65% is reported (slightly less than the 99.04% in the baseline implementation); however, this includes 4 additional parses (measurable as a drop in the number of skip sentences from 6 to 2).

hide_proportion	0.0001	0.001	0.01	0.1
Number of sentence	55	55	55	55
Number of Error sentence	0	0	0	0
Number of Skip sentence	6	4	**2**	4
Number of Valid sentence	49	51	53	51
Bracketing Recall	88.36	88.38	88.13	78.62
Bracketing Precision	88.36	88.38	88.13	78.62
Bracketing FMeasure	88.36	88.38	88.13	78.62
Complete match	65.31	64.71	62.26	47.06
Average crossing	0.49	0.47	0.42	0.82
No crossing	79.59	80.39	81.13	66.67
2 or less crossing	91.84	92.16	94.34	84.31
Tagging accuracy	99.04	98.83	**98.65**	91.67

6. Closing Comments

I'm really grateful for this opportunity to tackle OOV again. It was an interesting and thought-provoking exercise. I was not able to exceed the accuracy of the baseline implementation (99.04%). But I understand now that this figure hides my failure to measure the 6 skipped sentences. And the slight drop in accuracy is worth being able to parse 4 additional sentences; even though they contained OOV.