6.863J/9.611J: Natural Language Processing Prof. Robert C. Berwick

Assignment 2: Context-free grammar writing

My posse consists of: nobody *Will Whitney*

1. **Hand in the output of a typical sample run of 10 sentences from your random sentence generator. Be sure to attach the code so we can test it.**

2. (a) **Why does your program generate so many long sentences?** Specifically, what grammar rule is responsible and why? What is special about this rule?

   The rule `NP -> NP PP` causes long sentences because it has a recursive component. The NP in the predicate can then be expanded into further `NP PP` instances ad infinitum. Furthermore, there is a rule `PP -> Prep NP`, leading to a high probability of any `NP` generating another `NP`.

   (b) The grammar allows multiple adjectives, as in, `the fine perplexed pickle`. **Why do your generated program's sentences exhibit this so rarely?**

   Few of the generated sentences contain adjectives because `Noun -> Adj Noun` is only one rule, and `Noun -> terminal` has five options, any of which ends the cycle. The odds of any one noun having an adjective is 1/6, and the odds of having multiple adjectives decreases as $(\frac{1}{6})^n$.

   (c) **Which numbers must you modify to fix the problems in 2(a) and 2(b), making the sentences shorter and the adjectives more frequent? (Check your answer by running your new generator and show that they work.)**

   (d) **What other numeric adjustments can you make to the grammar in order to favor more natural sets of sentences? Experiment. Hand in your grammar file as a file named `grammar2`, with comments that motivate your changes, together with 10 sentences generated by the grammar.**

3. **Modify the grammar into a new single grammar that can also generate the types of phenomena illustrated in the following sentences.**

   (a) `Sally ate a sandwich .`

   (b) `Sally and the president wanted and ate a sandwich .`

   (c) `the president sighed .`

   (d) `the president thought that a sandwich sighed .`

   (e) `that a sandwich ate Sally perplexed the president .`

   (f) `the very very very perplexed president ate a sandwich .`

   (g) `the president worked on every proposal on the desk .`

   Briefly discuss your modifications to the grammar. Hand in the new grammar (commented) as a file named `grammar3` and about 10 random sentences that illustrate your modifications.

4. **Give your program an option "`-t`" that makes it produce trees instead of strings. Generate about 5 more random sentences, in tree format. Submit them as well as the commented code for your program.**

5. When I ran my sentence generator on `grammar`, it produced the sentence:

   `every sandwich with a pickle on the floor wanted a president .`

   This sentence is ambiguous according to the grammar, because it could have been derived in either of two ways.

(a) One derivation is as follows; **what is the other one?**

```
(START (ROOT (S (NP (NP (NP (Det every)
                            (Noun sandwich))
                                 (PP (Prep with)
                                     (NP (Det a)
                                         (Noun pickle))))
                            (PP (Prep on)
                                (NP (Det the)
                                    (Noun floor))))
                        (VP (Verb wanted)
                            (NP (Det a)
                                (Noun president))))
               .))
```

(b) **Is there any reason to care which derivation was used?**

6. (a) **Does the parser always recover the original derivation that was "intended" by `randsent`? Or does it ever "misunderstand" by finding an alternative derivation instead? Discuss. (This is the only part of question 6a that you need to answer.)**

   (b) **How many ways are there to analyze the following Noun Phrase (NP) under the original grammar? Explain your answer.**

   (c) By mixing and matching the commands above, generate a bunch of sentences from `grammar`, and find out how many different parses they have. Some sentences will have more parses than others. **Do you notice any patterns? Try the same exercise with `grammar3`.**

   i. **Probability analysis of first sentence: Why is `p(best_parse)` so small? What probabilities were multiplied together to get its value of 5.144032922e-05?**
   `p(sentence)` is the probability that `randsent` would generate this sentence. **Why is it equal to `p(best_parse)`?**
   Why is `p(best_parse|sentence)`=1?

   ii. **Probability analysis of the second sentence:**
   What does it mean that `p(best_parse|sentence)` is 0.5 in this case?
   Why would it be *exactly* 0.5?

   iii. **Cross-entropy of the two sentence corpus. Explain exactly how the following numbers below were calculated from the two sets of numbers above, that is, from the parse of each of the two sentences.**

   iv. **Based on the above numbers, what *perplexity* per word did the grammar achieve on this corpus?**

   v. **The compression program might not be able to compress the following corpus that consists of just two sentences very well. Why not? What cross-entropy does the grammar achieve this time? Try it and explain.** (The new 2 sentence corpus is given below.)

   vi. **How well does `grammar2` do on average at predicting word sequences that it generated itself? Please provide an answer in bits per word. State the command (a Unix pipe) that you used to compute your answer.**

   vii. If you generate a corpus from `grammar2`, then `grammar2` should on average predict this corpus better than `grammar` or `grammar3` would. In other words, the entropy will be *lower* than the cross-entropies. **Check whether this is true: compute the numbers and discuss.**

7. **Think about all of the following phenomena, and extend your grammar from question 3 to handle them. (Be sure to handle the particular examples suggested.) Call your resulting grammar `grammar4` and be sure to include it in your write-up along with examples of it in action on new sentences like the ones illustrated below.**

   (a) *Yes-no questions.*

   (b) *WH-word questions.*