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.)**

   I changed two lines of the grammar like this: `5 NP Det Noun` and `2 Noun Adj Noun`, thus decreasing the relative incidence of `NP -> NP PP` and increasing that of `Noun -> Adj Noun`.

   (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))))
                .))
```

The other derivation is this, with the second prepositional phrase referring to the pickle:

```
(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?**

Absolutely; in the first case, the sandwich is on the floor. In the second case, these sandwiches have pickles which are on the floor. It's a different semantic result.

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.)**

It sometimes "misunderstands" sentences generated by grammar3, because grammar3 is more flexible and has more options for interpreting a sentence, and even using the same grammar there are many interpretations for a given sentence. Furthermore, some things from my grammar3 are always misinterpreted by the parser running `grammar`, because I have reclassified them into more specific categories.

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

Five ways. Each of the prepositional phrases can be descended from any of the preceding noun phrases, so the options are to have all three directly descended from "every sandwich", two descended from "every sandwich" and the last descended from either the first or second prepositional phrase, and having only the first coming directly from "every sandwich" and then either the last two both referring to "a pickle" or "on the floor" referring to "a pickle" and "under the chief of staff" referring to "the floor".

Running `echo 'every sandwich with a pickle on the floor under the chief of staff' ./parse -c -s 'NP' -g grammar`— yields as the count `no._parses= 5`.

(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`.**

Using `grammar`, the obvious pattern is that longer sentences have more parses (and take longer to run. One I left running for fifteen minutes or so before killing. Running this on Athena is... painful.). This is especially true because most of the constituents of long sentences under this grammar are prepositional phrases, which as we saw in the last question are highly ambiguous.

Running on `grammar3`, my changes that reduced the incidence of `NP PP` chaining result in far fewer parses for the average sentence, and correspondingly faster runtime.

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?
The probability is so very small because every branching is, by itself, unlikely, and even a small sentence has many branchings.
The probabilities multiplied were, for each nonterminal, $\frac{1}{\#children}$, where $\#children$ is the number of rules with that element as the left side.
`p(best_parse|sentence)` is equal to `p(best_parse)` because there is only one parsing of this sentence, so the probability of the sentence (which is the sum of the probabilities of each parse) is equal to the probability of the only parse. `p(best_parse|sentence)`=1 because of this exact thing; for the given sentence, this parse is guaranteed; there is no alternative.

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?
This means that for this sentence, it was 50/50 whether you would get this parsing.
`p(best_parse|sentence) = 0.5` because there are exactly two parsings, of equal probability. These involve putting the second prepositional phrase either on "every sandwich" or on "a pickle".

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.**
These numbers are calculated by taking log_2 of the probability of each sentence, then adding them, then dividing by the number of words in each sentence. This yields the entropy per word.

iv. **Based on the above numbers, what *perplexity* per word did the grammar achieve on this corpus?**
Perplexity is 5.4.

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.)
This parse achieves infinity cross-entropy, because one of the sentences could not be parsed at all. The only rule that takes a `VP` also generates a predicate, so "the president ate ." is impossible.

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.**
`grammar2` has an entropy of 2.06 using the command
`~/6.863/assign2/6.863-pset2/randsent grammar2 10 | ./parse -g grammar2 -C`
The entropy of my `grammar3` is 2.22 bits. This is only slightly higher than my `grammar2`, I think due to the even lower incidence of `NP PP` chains that results from it. Those result in extremely low-probability sentences, which bias the results. By generating more varieties of sentences, but shorter, the results are only slightly lower probability per word.

When I try to calculate the entropy of `grammar`, the computer hangs, because there are so absurdly many parses for some of the sentences that have many chained prepositional phrases. When I do it with a very small sample size (2) then I get results around 2.23. This entropy is higher due to the aforementioned long phrase chains.

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.**
The cross-entropy of interpreting `grammar2` with `grammar` is 2.17 bits.
The cross-entropy of interpreting `grammar2` with `grammar3` is 3.075 bits.
`grammar3` can produce many things that `grammar2` can't, and those unreachable outputs bias the cross-entropy down. `grammar` is more similar to `grammar2`, and thus has a lower cross-entropy.

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.*

   My grammar can now generate sentences such as "will the president want and eat the president ?" and "did the president sigh ?"

   (b) *WH-word questions.*

   Now I can get sentences like "when will the proposal want a perplexed delicious chief of staff ?", "when did the proposal sigh under a desk in the floor ?", "who worked with a pickle on the desk ?", and "where did every proposal want and eat Sally ?"

   More samples are included in plaintext.