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

Assignment 2: Context-free grammar writing

My posse consists of: Chong-U Lim *Sang Woo Jun*

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.**

   The output is included as `q1_sample.txt`

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 responsible is `NP` → `NP PP`. This is because this rule has a self-reference in it, and has the possibility to infinitely recurse. In some of the sample runs, 1/5 of the dynamic rule execution was this rule. `PP` → `Prep NP` and `NP` → `Det Noun` also took up 1/5 of the dynamic execution each, but these rules executing is dependent on `NP` appearing repeatedly, which `NP` → `NP PP` is responsible for.

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

   In order to have multiple adjectives, the rule `Noun` → `Adj Noun` has to be executed multiple times, with the resulting noun expanding into an adjective and a noun again. However, there are 5 other rules a `Noun` can expand into. Since all rules are given equal weights, the possibility of a `Noun` expanding into an adjective and a noun again is only 1/6.

   (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 can modify the weight for `NP` → `NP PP` and make it smaller, in order to make the resulting sentences shorter, and I can increase the possibility of consecutive adjectives by increasing the weight of the rule `Noun` → `Adj Noun`. I tried setting the two weights to 0.1 and 50 respectively, and the program generated the following sentence:

   every fine fine pickled fine fine perplexed pickled fine fine delicious floor ate a delicious pickled fine perplexed fine perplexed delicious fine fine perplexed delicious pickled fine perplexed delicious fine fine pickled pickled delicious perplexed delicious perplexed perplexed pickled perplexed floor !

   (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.**

   There are no more meaningful ways to numerically tweak grammar2. Except for the two rules (`NP` and `Noun`) that were edited in the above question, all nonterminal have only one expansion that expands into a nonterminal. Since the terminal words themselves are not any more natural than any other, numerical tweaking by itself cannot make the grammar more natural. The weight of the rule that generates chained adjectives was lowered compared to the above question because chained adjectives did not seem natural.

   Semantic naturalness can somewhat be enhanced by reducing the possibility of `picked` or `in`, but this did not seem like what the question was asking.

   The output is included as `q2_sample.txt`

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.

   (a) I have added proper noun (NNP) Sally, and made NP expandable to it.

   (b) The coordinating conjunction 'and' has been added, and both noun phrase and verb phrase can be expanded to have it between two of themselves.

   (c) Verb phrase was made possible to expand to just a verb, and the verb sighed was added.

   (d) The subordinating conjunction 'that' was added, and verb phrase can be expanded into a new clause followed by a verb.

   (e) since the new clause from above item is lead by a subordinating conjunction, it was made to be availabel at root.

   (f) The adjective phrase and the adverb was added, and was made possible to loop into itself.

   (g) The verb phrase was made possible to be expanded into a verb followed by a prepositional phrase.

   The output is included as `q3_sample.txt`

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.**

   This option is included in the submitted code. However, the order of the arguments must not change. The first argument must either be `-t` or grammar name

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

```
(START (ROOT (S (NP (NP (Det every)
                        (Noun sandwich))
                    (PP (Prep with)
                        (NP (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?**

Yes, because the meanings are different. The given sentence describes a sandwich that has a pickle, and is on the floor, whereas the second sentence describes a sandwich, and a pickle that is on the floor.

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.)**

Yes it does misunderstand sometimes. For example, the phrase "sally and sally and pickle" was generated as (sally and sally) and pickle, but was parsed as "sally and (sally and pickle)". This is because while the generation of sentences are random and chooses whatever rule at random, the parser only chooses the most likely derivation deterministically.

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

There are 5 different ways. These are depicted in a relatively higher abstraction level below (The totally deterministic parsing of nouns and noun phrases have been ommitted).

i. NP(NP PP(Prep NP(NP(NP PP(Prep NP)) PP(Prep NP))))
ii. NP(NP(NP PP(Prep NP(NP PP(Prep NP)))) PP(Prep NP))
iii. NP(NP PP(Prep NP(NP PP(Prep NP(NP PP(Prep NP))))))
iv. NP(NP(NP(NP PP(Prep NP)) PP(Prep NP)) PP(Prep NP))
v. NP(NP(NP PP(Prep NP)) PP(Prep NP(NP PP(Prep NP))))

This can be verified using the following command, with two new arguments `-s NP` for starting parsing from NP, and `-c` for pringting the possible number of parses.

```
echo "every sandwich with a pickle on the floor under the chief of staff" | \
            ./assignment2code/parse -s NP -g ./grammar\_orig -c
```

(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`.**

Usually, the sentences that are longer has more parses. The cases when this is untrue is when rules that are very deterministic, such as those regarding multiple "very"s.

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 of `best_parse` is low because during the calculation of the value using the probalistic CYK algorithm, the probability of each rule that is used to generate this parse is multiplied together, and the probabilities are all sub-one values.
The reason it is equal to `p(best_parse)` is because this sentence has only one possible parse tree.

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?
There are two possible parses for this tree, and these two parses use the exact same rules, just in different order. Therefore the possibilies of the two resulting parses are identical. And because the possibility of the best sentence is the sum of these two possibility, `p(best_parse|sentence)` is 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.**
The two $p(sentence)$s were multiplied, and taken a base 2 log of, then devided by the numberof words, then multiplied by -1. This is exactly the formula of cross-entropy.
$-(log2(5.144032922e-05*1.240362877e-09)/18) = 2.435185067$

iv. **Based on the above numbers, what *perplexity* per word did the grammar achieve on this corpus?**
The perplexity is approximately 5.40833 per word.

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.)
If using the original grammar before adding any new rules, it failes to parse the second sentence **the president ate .** This means the `p(best_parse)` is zero, and the cross-entropy will go to infinity.
Unless new rules are added to parse the second sentence correctly, as grammar3 does, the compression will not be very good.

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.**

```
./randsent grammar2 30 | ./assignment2code/parse -g grammar2 -C
```

The cross-entropy of `grammar2` is **1.8454 bits**. This is lower than the examples we've been above, and much lower than the cross-entropy of `grammar3`, which is **2.5407 bits**. This is probably because the degree of freedom in grammar2 is much lower than in grammar3. More rules are deterministic because they have fewer or single possible expansions, compared to those in grammar3.
Calculating cross-entropy of the original grammar didn't work out because it would run into a recursion depth problem when trying to run any meaningful number of generations. I could modify the script to cut off the recursion after a certain depth, but that would not reflect the actual nature of the grammar. When run with a small number of runs (< 10), the cross-entropy is around 2 bits, putting it between grammar2 and grammar3. This is probably because of the various possible recursion depths giving it more freedom.

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.**
This is true, as can be demonstrated below. This is probably due to the same reason as discussed above. The probability of a particular parse or sentence is calculated to be lower

when there is a larger freedom of interpretation, which is the case for the original grammar and grammr3.

A. Original grammar: 2.02248 bits
B. Grammar 2 : 1.92526 bits
C. Grammar 3 : 2.7468 bits

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.*

   A new rule, SQ, was added for sentences starting with 'did' or 'will'. This rule expands into a noun phrase and a present-tense verb phrase VPP.

   The output is included as `q4_sample_1.txt`

   (b) *WH-word questions.*

   A new rule, SBARQ, was added for sentences starting with 'where' or 'what', and 'who'. And a new SQ derivative SQI was added for verb+NP that comes after the wh words.

   The output is included as `q4_sample_2.txt`