Assignment 3 Proabilistic Context Free Grammar

In this assignment, you will apply what you learnt about probabilistic context free grammar (PCFG) to answer questions about different PCFG implementation and design your own grammar for better PCFG parsing.

Get Started

Download the **tarball** file.
What's included are two scripts and three data files.The script `cfggen.pl` randomly samples sentences from a PCFG. The script `cfgparse.pl` parses sentences using a PCFG by finding the most probable parse.The output of `cfgparse.pl` will report to you three values, which are the probability of the best parse and the sentence $P(T, S)$, the total probability of the sentence $P(S)$, and the probability of the parse it gives you, given the sentence $P(T|S)$. Meanwhile, it will also demonstrate the best parse of the sentence to you in WSJ format. An example of output for the sentence *"Arthur is the king."* is attached below:

```
1.238e-13 2.476e-13 0.500 (ROOT (S2 (+Misc (Misc do) (+Misc (
Misc coconuts) (+Misc (Misc speak) (+Misc (Misc ?)))))))
```

The three data files are `grammar1`, `grammar2`, and `lexicon`, which give you the starting point for building a PCFG for a subset of English (namely , all English sentence using the vocabulary in the file called wordlist). The

file `examples.sen` gives you some sentences in that subset of English.

You are encouraged to read the documentation at the top of the two scripts to understand how to use them. The **–pretty** option (on both scripts) and the **–text** option (on `cfggen.pl`) is going to be helpful.

Task 1

Look at the file `grammar2`. Try parsing the example sentences using `grammar2`, `lexicon`, and the script we provide by running the following command:

```
./cfgparse.pl grammar2 lexicon < examples.sen
```

Explain what kind of model does this PCFG implement and why?

Hint: Think about the language model that we have learned before.

Task 2

Run `cfgparse.pl` with the `grammar1` and `lexicon` on the example sentences with the following command:

```
./cfgparse.pl grammar1 lexicon < examples.sen
```

The perl code with parse the sentences in the examples.sen with respect to the PCFG defined in `grammar 1` and `lexicon`.

Note that `cfgparse.pl` can take more than one grammar file on the command-line; it just merges the rules together and sums up each rule's weights if they occur more than once. Also, the weights don't have to be probabilities; the script will automatically renormalize them into probabilities.
Compare the outcome when you run:

```
./cfgparse.pl grammar1 lexicon < examples.sen
```

versus

```
./cfgparse.pl grammar1 grammar2 lexicon < examples.sen
```

Explain what are the difference between the outcomes and why.

Task 3

Use the generating script, `cfggen.pl`, to generate sentences in three ways:

```
./cfggen.pl --text <N> grammar1 lexicon
./cfggen.pl --text <N> grammar2 lexicon
./cfggen.pl --text <N> grammar1 grammar2 lexicon
```

N should be an integer which indicates number of sentences to be generated.

Explain the difference between the three outputs and why.

Hint: refer to your answer from Task 1: what kind of model is grammar2? Consider the weights on the `ROOT -> ...` rules in trying to understand the output of the third command.

Task 4

This is the most fun and challenging part of this assignment.

Build your own PCFG grammar!

Feel free to use rules from grammar1, grammar2, and/or lexicon. You can also borrow rules that you have seen in the lectures and textbook. The requirements and goals of your designed grammar is listed as following:

1. You will need to create two files named as `mygrammar` and `mylexicon`. `mygrammar` defines a set of PCFG rules to derive any non-terminal symbols to other non_terminal symbols with the format:
   `Nonterminal_A-> Nonterminal_B weight`
   or
   `Nonterminal_A -> Nonterminal_B Nonterminal_C weight`.
   In `mylexicon`, you will define a set of rules to derive any non-terminal symbols to terminal symbols with the format:
   'non_terminal -> terminal weight'
   The `weight` after each rule represents the probability of deriving the righthand side from its lefthand side. Notice that you don't have to scale the weight down to 0 and 1.
2. When you define your `mylexicon` rules, you can only use the

words from the file `wordlist` for terminals and never adding new words in your mylexicon.

3. Ideally, Your grammar should assign high probability to any grammatical sentences. For ungrammatical sentences, you want your grammar to give a very low probability. Please also design your grammar in such a way that, it will never fail when parse any string of words.
4. When sampling sentences from your grammar with `cfggen.pl`, you want the sampled sentences to be grammatical.
5. You cannot include new words as the terminals in your 'mylexicon'. All the words have to come from the file `wordlist`
6. Explain the stratgies you used to design your grammar.
7. This part of the assignment is competitive. We will compare your grammar to everyone else's grammar using cross-entropy as the evaluation metric. The minimum requirement is to beat the performance of a **merged grammar** from `grammar1` and `grammar2`. Meeting the minimum requirement will assure you 60% of the points for this task. To earn the rest of the points, your grammar's performance will need to be in the top 70% of the whole class. The top 10% will get bonus points for this problem. Details of the competition will be introduced later.

Task 5

Use `cfggen.pl` to generate 20 sentences from your grammar, turn them in as one file called `generated.txt`. Then turn in a version of the file with all of the ungrammatical sentences removed, called 'generated-grammatical.txt'. Tell us what fraction of your 20 sentences you decided

were grammatical.The submitted grammatical sentences can only consist of the words from the file ‘wordlist’. **You will not be graded on the fraction of the grammatical sentences in your generated sentences.**

Intorduction to Competition.

We will evaluate your grammar using cross entropy on a test dataset, which captures how well can you predict the output of the other team’s grammars. The test dataset is constructed from the genreated grammatical sentences from each of your grammar in task 5. The cross-entropy is a measure of whether the grammar gives high or low probability to the test dataset sentences. It is defined as:

$$2^{\frac{-log_2 p(s_1)-log_2 p(s_2)-log_2 p(s_3)...-log_2 p(s_n)}{n}}$$

where $p(s_1)$ is the maximum probability that your grammar assigns to sentence 1. We will get these probabilities values by running `cfgparse.pl` with your grammar on the test dataset. A smaller number of cross-entropy is better.

To evaluate your grammar’s performance by yourself, you can:

(1) Compute the cross entropy scores of your grammar by using it to parse the sentences in `examples.sen` and try to lower that score by modifying your grammar.

(2) Expand your evaluation sentences by including some human generated grammatical sentences using the words from the `wordlist` file or get some generated grammatical sentences from your classmates’ grammars. Then reduce the cross entropy scores on the expanded evaluation

sentences.

(3) Check the fraction of the grammatical sentences generated with your grammar using `cfggen.pl`.

Submission

Please submit a zip archive containing the following items.

(1) `answers.pdf`: A pdf file with answers to all the questions.

(2) `mygrammar`: A set of probablistic context free grammars that you designed to expand non-terminal symbols to non-terminal symbols.

(3) `mylexicon`: A set of probablistic context free grammars that you designed to expand non-terminal symbols to teminal symbols.

(4) `generated.txt`: Your generated text from Task 5.

(5) `generated-grammatical.txt`: Your grammatical-only text from Task 5.

Before you submit, please double-check that you have followed all instructions for the submission files!