Context Free Grammars

Francisco Iacobelli

January 16, 2018

1 Introduction

This assignment is to give you a better understanding of context free grammars and techniques to generate and understand language using them. It is heavily based (a subset actually) of an assignment by professor J. Eisner at John Hopkins University.

There will be a quiz on this assignment shortly after the due date.

Lastly, the programming questions must be done in python and you may not use any external libraries except the standard python libraries (such as `re, collections, math`, etc.)

1.1 Grammar File Format

The grammar file has two kinds of lines. Comment lines and grammar lines. Comment lines are preceeded with a # and must be ignored. They are there to make the file more human readable and they contain specifications about the file. You must read them.

Grammar lines look like this:

```
1       ROOT    S .
1       S       NP VP
1       NP      DT Noun #multiple rules for NP
1       NP      NP PP
1       Noun    president
1       Noun    chief of staff
```

and they correspond to rules like these:

ROOT	→	S .
S	→	NP VP
NP	→	DT Noun
		NP PP
Noun	→	president
		chief of staff

The grammars lines consist of three parts: a number, the left-hand-side (LHS) non terminal and a sequence of zero or more terminal and nonterminal symbols, which is called the "right-hand side" (RHS) of the rule. The number will be used later.

Please download the grammar file from this link: http://fid.cl/courses/nlp/hw/grammar.

2 Programming Questions

2.1 Basic Random Sentence Generator

Write a random sentence generator. Each time you run the generator, it should read the (context- free) grammar from a file and print one or more random sentences that conform to that grammar. It should take as its first argument a path to the grammar file. If its second argument is present and is numeric, then it should generate that many random sentences, otherwise defaulting to one sentence. Name the program randsent so it can be run as:

```
python randsent grammar 5
```

where `grammar` is the name of the grammar file.

The grammar's start symbol will be `ROOT`, because it is the root of the tree. Depth-first expansion is probably easiest, but that's up to you. Each time your generator needs to expand (for example) `NP`, it should randomly choose one of the `NP` rules to use. If there are no `NP` rules, it should conclude that `NP` is a terminal symbol that needs no further expansion. Thus, the terminal symbols of the grammar are assumed to be the symbols that appear in RHSes but not in LHSes.

Remember, your program should read the grammar from a file. It must work not only with the sample grammar, but with any grammar file that follows the correct format, no matter how many rules or symbols it contains. So your program cannot hard-code anything about the grammar, except for the start symbol, which is always`ROOT`.

2.2 Smarter Sentence Generator

Modify your generator so that it can pick rules with unequal probabilities. The number before a rule now denotes the relative odds of picking that rule. For example, in the grammar:

```
1 NP DT Noun
1 NP NP PP
```

both `NP` rules have equal chance of being picked. However:

```
2 NP DT Noun
1 NP NP PP
```

the first `NP` rule has $\frac{2}{3}$ chance of being picked, while the second has only $\frac{1}{3}$ chance of being picked.

Name your program `randpsent` and have it be run as the previous program you wrote.

3 More Complex Sentences

Modify the grammar so it can also generate the types of phenomena illustrated in the following sentences. You want to end up with a single grammar that can generate all of the following sentences as well as grammatically similar sentences.

1. Sally ate a sandwich .

2. Sally and the president wanted and ate a sandwich .

3. the president sighed .

4. the president thought that a sandwich sighed .

5. it perplexed the president that a sandwich ate Sally .

6. that a sandwich ate Sally perplexed the president .

7. the very very very perplexed president ate a sandwich .

8. the president worked on every proposal on the desk .

Note: Yes, (6) is grammatical in standard written English. It means the same thing as (5).

While your new grammar may generate some very silly sentences, it should not generate any that are obviously ungrammatical. For example, your grammar must be able to generate (4) but not

*the president thought that a sandwich sighed a pickle .

since that is not okay English. The symbol * is traditionally used to mark "not okay" language. Again, while the sentences should be okay structurally, they don't need to really make sense. You don't need to distinguish between classes of nouns that can eat, want, or think and those that can't.

Have your grammar be called `grammar2` and it should work on both the first and second programs.

4 Conceptual Questions

1. For the first program: Why does it generate so many long sentences? What grammar rule is responsible?

2. How many ways are there to analyze the following noun phrase under the original grammar? (That is, how many ways are there to derive this string if you start from the NP symbol of grammar?) `every sandwich with a pickle on the floor under the chief of staff` Explain your answer.

5 Programming Recommendations

- Make sure your code is clear and simple. If it is not, revisit it.

- The data structure you use to represent the grammar and rules should probably be quick to lookup and should be able to be indexed by strings. The dictionary in python provides such structure.

6 Submitting your homework

You must submit ONE zip file with the two programs, the two grammar files (the one given and the one you create) and a TEXT (not word) file with the answer to the questions. It must all be in the same directory.

7 Python File Formatting

The file must comply with the following convention (THIS IS VERY IMPORTANT)

- The first line of your file should indicate the python version as follows:
  - If you are using a flavor of python 3.x, your first line should be: `#!/usr/bin/env python3`
  - If you are using a flavor of python 2.7.x your first line should be: `#!/usr/bin/env python2`
- The second line should have the name of the homework and optionally a couple of words about it. These should be enclosed in three quotation marks. For example: `""" Eliza homework. Relationship advisor """`
- The third line should have your name assigned to the `__author__` variable. For example if your name is "John Doe" your next line should be:

  `__author__="John Doe"`
- Optionally, you can specify a file encoding on your second line and then follow the convention. You do this by adding the following line as a second line: `# # -*- coding: utf-8 -*-`

A sample hello world file created with python 2.7.x for John Doe looks like this:

```
#!/usr/bin/env python2
# # -*- coding: utf-8 -*-
""" Hello World program """
__author__="John Doe"

print "Hello World"
```

The same file created with python 3.x looks like:

```
#!/usr/bin/env python3
# # -*- coding: utf-8 -*-
""" Hello World program """
__author__="John Doe"

print ("Hello World")
```