NLP Homework 3
Due Tuesday, November 7 by midnight

This homework requires Java. If you need to install Java, we will give directions. If you have any difficulties, contact me.

Writing a PCFG, a Probabilistic Context-Free Grammar

This homework is based on a similar homework from the Stanford NLP course, which was based on idea originally from Jason Eisner and Noah Smith at Johns Hopkins University.

For this homework assignment, you will be developing a PCFG for a small subset of English. There will be just one part, writing grammar rules, with no option for programming. We will focus on the part that writes CFG rules and ignore the probabilities of the rules, except for the Start symbol.

The grammar rules will be written in two files, one for a grammar called S1 (and whose start symbol is S1) and one to assign non-terminal symbols for the words in the vocabulary of this limited language, where these symbols can be thought of as POS tags, but are not required to be.

There is a second grammar, S2, which you do not have to write and which will generate all word strings as a linear sequence with no structure if there is no rule in S1 that will apply. This grammar is a convenience for the scoring function. Essentially S2 is a backoff model.

Your rules will be written in simple text files, ending in the file extension ".gr". You will have a set of sentences to try to parse, given in a file called dev.sen. There is a java program provided that tries to parse these sentences with the PCFG in S1.gr, S2.gr and Vocab.gr. It scores the resulting parse trees with a perplexity measure, but in our assignment we are going to ignore these scores. (The perplexity measure is lower if your grammar models the sentences well; that is, it is not surprised/perplexed by the sentences.)

Instead, your goal is to produce as many parse trees as possible that use your rules from S1, instead of the backoff rules from S2.

For the homework, download the PCFG.zip file and extract the directory called PCFG that has the following structure:

```
PCFG/
        allowed_words
        challenge.sentences.txt
        dev.sen
        pcfg.jar
        S1.gr
        S2.gr
        S2generator.py
        Vocab.gr
```

The allowed_words list is just for your own information in which words can appear in sentences for this grammar.

The sentences that you are required to work on are in dev.sen. In addition, there are a number of harder sentences in the file challenge.sentences.txt. You will pick two of these sentences to also write grammar rules for and copy them into your dev.sen file.

The file pcfg.jar is the java program that runs the pcfg parser for you.

You will add grammar rules to the files S1.gr and Vocab.gr. Every time that you add a new symbol to the grammar rules in either file, you should run the python program S2generator.py to create an updated version of the S2 grammar in the file S2.gr.

If you wish, you can also choose different weights for the rules in the grammar, expressing your knowledge of which English expressions are common and which ones aren't. But this part is not required for our small grammar to work to find a parse, as long as you don't change the relative weights of S1 and S2.

More details on grammars

More formally, a probabilistic context free grammar consists of:

- A set of non-terminal symbols
- A set of terminal symbols (which we can call the vocabulary)
- A set of rewrite or derivation rules, each with an associated probability
- A start symbol

For natural language PCFGs, we think of the start symbol as indicating "sentence" (in this case it will be START), and the terminal symbols as the words.

A derivation rule gives one way to rewrite a non-terminal symbol into a sequence of non-terminal symbols and terminal symbols. For example, S -> NP VP says that an S (perhaps indicating a declarative sentence) can be rewritten as an NP (noun phrase) followed by a VP (verb phrase). For the system we've provided you with, the rules need to be written one per line in a plain text file with the following syntax:

weight <tab> non-terminal symbol <tab> child1 <space> child2 <space> . . .

Each of the children can be a non-terminal symbol or a word/terminal symbol. So, for example, you could have a rule in the vocabulary file that always requires the words Holy and Grail to occur in sequence as a proper noun:

1 <tab> NNP <tab> Holy <space> Grail

Words/terminal symbols can occur both on the right hand sides of rules in S1 and in the vocabulary file, e.g. the word "was" can occur as a past tense verb in the vocabulary file and can also be explicitly allowed as an auxiliary verb in the grammar rules in S1, in a rule like

1 <tab> Vbar <tab> was <space> VBpast

In the file Vocab.gr, we are already giving you the complete set of terminal symbols embedded in rules of the form “Tag -> word”. But most of the Tags are given as the tag “Misc” and one of your jobs is to replace the Misc. tags with a more meaningful tag whenever you need to parse the word.

In the file S1 is a simple grammar to start with. To extend this grammar, you can add new rewrite rules, you can add new phrase type symbols with their own rules. All of these rules can use the new tag symbols that you write in the Vocab.gr file.

Note that the grammar in the file S1.gr includes the special rules START -> S1 and START -> S2. These are the rules that allow the parser to use either rules from S1 or rules from S2.

Running the parser

When you want to see what your grammar does, you can parse sentences with it. Although you can run the parser just to get the perplexity scores, we are going to use the part that shows the highest scoring parse tree that it found.

Open a terminal or command prompt window and navigate to the PCFG directory. Note that you shouldn’t change the names of the files dev.sen, S1.gr, S2.gr and Vocab.gr. First generate the S2.gr that goes with your S1 grammar and Vocab:

```
[Mac and Windows – or use python or whatever python you use to start the interpreter]
$  python S2generator.py > S2.gr
```

(Note that the “>” puts the output of the S2generator program into the file S2.gr.)
Now run the parser:

```
[Mac and Windows]
$  java -jar pcfg.jar parse -t dev.sen *.gr
```

On Windows, if your system complains that it doesn’t know the command java and if java is installed, give the complete path to the java (but do not copy/paste the quotes from this document). Substitute the path on your computer:

```
$ "C:\Program Files (x86)\Java\jre1.8.0_51\bin\java" -jar pcfg.jar parse -t dev.sen *.gr
```

Recommended development procedure

1. Run the parser and observe from the output which sentences have parse trees from S2. Pick one of these sentences to work on.

2. If there are any words labeled Misc (other than .), try to label them with something else that is either a POS tag or similar. For example, we may use “Adj” for words that are being used as adjectives. Go to the Vocab.gr file and change that tag for that word, and any other words in the same class.
3. Add one or more rewrite rules to the grammar S1. Create new non-terminal symbols as needed.
4. Generate the backoff grammar with your new symbols by running the S2generator.py program.
5. Run the parser and check the parse trees!
6. Write down the sentence and the new rules that you created in Vocab.gr and S1.gr for your report.
7. Repeat as needed.

If you need ideas for grammar rules, put your sentence into the Stanford demo parser and figure out what kind of rules could generate that tree. Your rules cannot and should not try to get exactly the parses from the Stanford parser (see the Notes below), but do try to make sensible phrases, e.g. verb phrases should start with a verb.

Part 1: Writing grammar rules
You should write grammar rules that parse all of the sentences in dev.sen. In addition, you should choose two additional sentences from the challenge sentences file.

Notes:
In a PCFG, the prepositional attachments may be wrong. That’s o.k. You can’t really fix these with probabilities and no lexical information used from the grammar.

The parsing program puts the PCFG into Chomsky Normal Form before parsing. So when you write a rule with a RHS that has 3 children, you’ll get a parse tree with an extra non-terminal that derives the first two children and then it is the first child of the main rule. Don’t try to fix this, either.

Restrictions:
Do not treat all the verb modifiers as if they were just verbs.
Prepositional phrases should start with a preposition and noun phrases should not start with a preposition.
Noun phrases should also not start with most types of verbs.

Part 2: Exemplar Sentences
Given the grammar that you wrote for part 1,

- make up a sentence that uses some of the same words in the sentences that you already parsed and is an actual English sentence, but cannot be parsed by those rules. [To check if it’s an English sentence, you could try it in the Stanford parser.] The point of this sentence is to demonstrate some type of phrase structure that this grammar does not parse.
- make up a string of words that should not be an actual English sentence, but your grammar will parse it. [If the grammar parses obvious non-English sentences, it is “over-

generalization", i.e. it is not restrictive enough in the forms that are allowed. Note that we wouldn't want a grammar that overgeneralized too much (hence the restrictions above), but for the sake of simplicity in writing the grammar rules, we do allow some.]

What to submit for Homework:

Part 1:
Each student should submit a report that includes the final grammar that you got for S1 with the original sentences in dev.sen and your two challenge sentences. Then for each sentence,

- write down the sentence,
- show the parse tree and
- write down the rules that you added to Vocab.gr and S1.gr.
- Add a sentence or two that describes how the rules are applied to parse the sentence, including the significant rules from the grammar; one way is to discuss the parse and describe which part(s) are new for that sentence. You must also explain in your own words how those rules are generating the subphrases.

If you had any difficulties, you can describe them.

Part 2:
Give your two exemplar "sentences". For the first sentence, explain what part of the phrase structure is not represented by the grammar in part 1 and show the PCFG output with no parse (i.e. only a parse from S2). For the second "sentence", explain how overgeneralization allows this so-called sentence to be parsed and give the parse tree output from the PCFG.

How to Submit Homework:

Go to the Blackboard system and the Assignment for Homework 3 and attach your report and your grammar files. You may submit a .zip file, but please do not submit .rar files.