423/723 NLP Homework #2 - 2017

Due: 11:59 pm October 23 to the D2L Dropbox

(Note: the following questions are based on readings from Chapter 8 of the NLTK book)

Place your answers to questions 1-3 and question 4 parts a-c, and the discussion part of part d, in a file called hw2_writeup (use pdf, or doc, not rtf). The code, input and output files for question 4 should be separate, with appropriate file extensions for each. The files should be combined into a single file archive using tar or zip before you upload them to D2L.

1. Suppose a grammar allows conjoined constructions such as

S -> S and S S-> S while S
S-> S or S S -> S when S
S -> S but S

Provide two correct parse trees for the following sentence, using nested, bracketed notation. (Note that this is another one that the MS SPLAT demo parser gets totally wrong.)

John slept while Mary ate and the dog barked

2. Consider the following long literary sentences and for each part provide the following:

i) Give a set of all the additional S-level rules that you would need including any additional conjunctions or other function words.

ii) Show each passage with brackets around all the complete sentences and noun phrases (possibly nested) based on the combined S grammar (from part 2i and question 1) and your understanding of noun phrases (e.g. from homework 1).

a)E.B. White, "Stuart Little." 107 words.

"In the loveliest town of all, where the houses were white and high and the elms trees were green and higher than the houses, where the front yards were wide and pleasant and the back yards were bushy and worth finding out about, where the streets sloped down to the stream and the stream flowed quietly under the bridge, where the lawns ended in orchards and the orchards ended in fields and the fields ended in pastures and the pastures climbed the hill and disappeared over the top toward the wonderful wide sky, in this loveliest of all towns Stuart stopped to get a drink of sarsaparilla."

b) (GRAD ONLY) W.G. Sebald, "The Rings of Saturn." 107 words.

"All I know is that I stood spellbound in his high-ceilinged studio room, with its north-facing windows in front of the heavy mahogany bureau at which Michael said he no longer worked because the room was so cold, even in midsummer; and that, while we talked of the difficulty of heating old houses, a strange feeling came upon me, as if it were not he who had abandoned that place of work but I, as if the spectacles cases, letters and writing materials that had evidently lain untouched for months in the soft north light had once been my spectacle cases, my letters and my writing materials."

3. Consider the following examples of "garden path" (GP) sentences and some similar non-garden path sentences. (They are from the Wikipedia entry on Garden Path Sentences). Using the MSR SPLAT demo (http://msrsplatdemo.cloudapp.net/), compute the precision and recall of the GP versions based on the number of correct phrases when compared to the non-GP versions. (Show the unsimplified expressions that you used to compute the precision and recall.).

a) The horse raced past the barn fell. vs The horse that was raced past the barn fell.
b) The old man the boats vs The old sank the boats.

c) The complex houses married and single soldiers and their families. vs The complex was sold to married and single soldiers and their families

4. Programming Question (probabilistic phrase structure grammars and chart parsers)

Introduction:
This part of the assignment is based on exercises and code from Henry Thompson, Bharat Ram Ambati, and Sharon Goldwater which has been licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Code files:

fixesNLTK3.py	fix buggy NLTK3 with different fixes for different versions
BetterICP.py	This code fixes bugs in nltk.parse.pchart related to how it adds edges to the chart and keeps track of them. BetterICP class has a method parse which parses a sentence and prints all possible parses. This method takes three arguments. First argument is tokens, which is a list of words in the sentence. Second argument is notify. notify=True will print each parse as it is found without waiting until the end of the process. Third argument max defines the number of possible parses to be printed. The parsing process will stop once max number of parses have been found. It uses NLTK's InsideChartParser module which is a probabilistic chart parser (help(nltk.InsideChartParser) for more details).
runHW2.py	This code contains the main body of code that imports the other two files. It also loads the Penn Phrase Structure Treebank dataset, which consists of around 3900 sentences, where each sentence is annotated with its phrase structure tree. Start Python and type execfile('runHW2.py') and rerun it after any edits you make to the file.

Probabilistic Phrase Structure Grammars (PCFGs) are Phrase Structure Grammars, where each production has a probability assigned to it. In NLTK, the data type of this grammar is ProbabilisticGrammar.

The given file runHW2.py begins by importing the functions it needs, extracting tagged sentences using NLTK libraries, and defining a toy PCFG grammar. The relevant lines (after the import statements) are shown below:

```
## Main body of code ##
# Extracting tagged sentences using NLTK libraries
psents = treebank.parsed_sents()
# Comment out the following 3 lines if you get tired of seeing them
print "\n 1st parsed sentence: \n", psents[0]
print "\n Productions in the 1st parsed sentence: \n"
pprint(psents[0].productions())

grammar = parse_pgrammar("""

# Grammatical productions.
S -> NP VP [1.0]
NP -> Pro [0.1] | Det N [0.3] | N [0.5] | NP PP [0.1]
VP -> Vi [0.05] | Vt NP [0.9] | VP PP [0.05]
Det -> Art [1.0]
PP -> Prep NP [1.0]
# Lexical productions.
Pro -> "i" [0.3] | "we" [0.1] | "you" [0.1] | "he" [0.3] | "she" [0.2]
Art -> "a" [0.4] | "an" [0.1] | "the" [0.5]
Prep -> "with" [0.7] | "in" [0.3]
N -> "salad" [0.4] | "fork" [0.3] | "mushrooms" [0.3]
Vi -> "sneezed" [0.6] | "ran" [0.4]
```

```
Vt -> "eat" [0.2] | "eats" [0.2] | "ate" [0.2] | "see" [0.2] | "saw" [0.2]
""")

sentence1 = "he ate salad"
sentence2 = "he ate salad with mushrooms"
sentence3 = "he ate salad with a fork"
```

Once you load the data from the treebank corpus, as in the code provided, the variable psents will contain all the parsed sentences in the treebank. The statement prods = get_costed_productions(psents) gives all the productions in the treebank with their costs. Unlike in the toy grammar however, the cost is computed as the -base-2-log of the maximum-likelihood estimate of the probability of each production, based on its frequency in the treebank.

Note that punctuation symbols appear both in the trees and the grammar. Many common punctuation marks occur, but in order to avoid confusion with the tree displays, brackets are replaced with three letter acronyms (e.g. LRB = left round bracket, RRB= right round bracket, LSB= left square bracket, LCB = left curly bracket etc).
The digits on some tags are coreference markers which pair up e.g. a relative pronoun and a subsequent gap, as in:

```
(NP
(NP (DT the) (NN executive) (NNS functions) )
(SBAR
        (WHNP-2 (WDT that) )
                (S
                        (NP-SBJ (DT the) (NNP Confederation) (NNP Congress) )
                        (VP (VBD had)
                                (VP (VBN performed)
                                        (NP (-NONE- *T*-2) ))))))
```

the executive functions that the Confederation Congress had performed

You can also use NLTK to help you draw a parse tree from a string as follows, using your own tree represented in bracketed notation.

```
treeString = "(S (NP (Pro I))) (VP (Vt ate) (NP (N salad) ) ) ) "
tree= nltk.tree.Tree(treeString)
tree.draw()
```

Warm-up exercises (ungraded):

Here are a few things to try to before working on the programming parts:

1. Start python and type execfile('runHW2.py') to run the file and see the productions for the first sentence.
2. Comment out the three lines that print out the first sentence and its productions and try some of the other examples:
    - Run the parser on sentences "he sneezed" and "he sneezed the fork" and look at the output for each of these sentences. You can set a variable, or just type: `sppc.parse(str.split("he sneezed"))`
    - Uncomment the next two parses, so the PCFG parser runs on the sentences "he ate salad with mushrooms", and "he ate salad with a fork". ; rerun the file using execfile. Note which reading is preferred. *(The parses are listed based on their probabilities (high to low)/costs(low to high).)*
    - Edit the grammar and switch the probabilities of the two rules involved. Re-load, with appropriate commenting so that only sentence2 and 3 are parsed. Notice the changes.
    - Turn tracing on, by adding the following before the sppc.parse lines in runHW2.py: sppc.trace(1) and rerun the three parses. Note how the order in which edges are added is determining what analysis gets found first.
    - Restore the probabilities of the VP rules and rerun.

Graded Programming exercises: For subparts a-c below, follow the instructions and briefly answer questions (marked in bold). Part d contains a programming question as well as asking for a written summary of your solution.

a) Create a PCFG by uncommenting the prods=... and ppg=PCFG(Nonterminal('NP'), prods) lines in runHW2.py and re-loading. (The first argument to PCFG is the start symbol and the second argument is the productions with

their costs. To see the parts within the returned list, you can look at `prods[i].rhs(), prods[i].lhs()` and `prods[i].cost()`. To use a field as a key value as a parameter, you can use lambda, e.g. you can write: key= lambda x: x.cost()) Now try sprods = ppg.productions(Nonterminal('S')), which gives a list of all the productions that start with 'S'. (Note that unlike lisp, the expression will not automatically print the result; you need to evaluate it separately.) **How many productions are there in the treebank that start with 'S'**? **How many start with NP (or a variant)**? (Write short functions to do these calculations.)

b) In BetterICP, the second parameter in the constructor specifies the width of the beam (which is the maximum length of the queue used to represent the chart.) Un-comment the following lines and run the code:

```
ppc=BetterICP(ppg,1000)
print "beam = 1000"
ppc.parse("the men".split(),True,3)
```

Now comment out the last two of the above three lines, and remove the comments from the next three lines, to widen the beam to 1050, and run again and see what happens. Now, uncomment three more lines to increase the width 1200 (but do not uncomment the trace lines). **Describe what happens after changing the beam width and what that signifies in terms of the grammar.**

c) Finally, comment out the last three lines and try editing runHW3.py to include three new lines of the form below, where BIGGERNUM is the smallest number big enough to get the simple DT NNS structure one might expect?

```
ppc.beam(BIGGERNUM)
print "beam = BIGGERNUM"
ppc.parse("the men".split(),True,3)
```

What was the smallest value you found for BIGGERNUM that finds the correct parse? Provide a table that shows different beam widths and the number of parses returned, where you include at least one beam value for each possible number.

Width	Parses returned
	0
	1
	2
	3

d. Modify the grammar: Create a new file, **`runGrammar2.py`** that defines a new grammar (a modification of the the one provided), defines a new function that reads the sentences from an input file, and then runs the parser on the sentences from the file and prints the output to another file called hw2_output.txt. Your updated toy grammar, called grammar2, should add grammatical and lexical productions (or add new items on the RHS of existing rules) as needed to handle:

- imperative sentences such as "eat the salad",
- ditransitive sentences, such as "I read her the book" (but not *I read she the book")
- passive sentences, such as "The book was read to her" (but not *I read the book to she)

You should add a new category of pronoun, Pro_obj, for pronouns that can be indirect objects or objects of prepositions (such as him, her, them) but not subjects. Create a text file, hw2_input.txt with enough test sentences, each on a separate line, to demonstrate that your grammar works correctly, and then run your runGrammar2.py code.