hw4

December 15, 2017

1 Homework 4: CKY Algorithm and Dependency Parsing

1.0.1 1: CKY Algorithm (30 points)

In this section, you will implement the CKY algorithm for an unweighted CFG. See the starter code cky.py.

Question 1.1 (8 points) Implement the acceptance version of CKY as `cky_acceptance()`, which returns True if there is a S covering the entire sentence. Does it return True or False for the following sentences? Please `pprint()` the chart cells for each case as well. * the the * the table attacked a dog * the cat

Hint: A simple way to implement the chart cells is by maintaining a list of nonterminals at the span. This list represents all possible nonterminals over that span.

Hint: `pprint()`ing the CKY chart cells may be useful for debugging.

Hint: Python dictionaries allow tuples as keys. For example, `d={}; d[(3,4)] = []`. A slight shortcut is that `d[3,4]` means the same thing as `d[(3,4)]`.

```
In [1]: import cky
        from cky import *
        from pprint import pprint
        print "Grammar rules in tuple form:"
        pprint(cky.grammar_rules)
        print "Rule parents indexed by children:"
        pprint(cky.possible_parents_for_children)

Grammar rules in tuple form:
[('S', ('NP', 'VP')),
 ('NP', ('Det', 'Noun')),
 ('VP', ('Verb', 'NP')),
 ('PP', ('Prep', 'NP')),
 ('NP', ('NP', 'PP')),
 ('VP', ('VP', 'PP')),
 ('S', ('NPS', 'VPS')),
 ('NPS', ('Det', 'Nouns')),
 ('VPS', ('Verbs', 'NPS')),
 ('VPS', ('Verbs', 'NP')),
 ('PP', ('Prep', 'NPS')),
 ('NPS', ('NPS', 'PP')),
```

```
  ('VPS', ('VPS', 'PP'))]
Rule parents indexed by children:
{('Det', 'Noun'): ['NP'],
 ('Det', 'Nouns'): ['NPS'],
 ('NP', 'PP'): ['NP'],
 ('NP', 'VP'): ['S'],
 ('NPS', 'PP'): ['NPS'],
 ('NPS', 'VPS'): ['S'],
 ('Prep', 'NP'): ['PP'],
 ('Prep', 'NPS'): ['PP'],
 ('VP', 'PP'): ['VP'],
 ('VPS', 'PP'): ['VPS'],
 ('Verb', 'NP'): ['VP'],
 ('Verbs', 'NP'): ['VPS'],
 ('Verbs', 'NPS'): ['VPS']}
```

```python
In [2]: #Print the result here
        import cky;reload(cky)
        print "the the"
        print cky.cky_acceptance(["the", "the"])
        print "the table attacked a dog"
        print cky.cky_acceptance(["the", "table", "attacked", "a", "dog"])
        print "the cat"
        print cky.cky_acceptance(["the", "cat"])
```

```
the the
{(0, 1): 'Det', (0, 2): [], (1, 2): 'Det'}
False
the table attacked a dog
{(0, 1): 'Det',
 (0, 2): 'NP',
 (0, 3): [],
 (0, 4): [],
 (0, 5): 'S',
 (1, 2): 'Noun',
 (1, 3): [],
 (1, 4): [],
 (1, 5): [],
 (2, 3): 'Verb',
 (2, 4): [],
 (2, 5): 'VP',
 (3, 4): 'Det',
 (3, 5): 'NP',
 (4, 5): 'Noun'}
True
the cat
{(0, 1): 'Det', (0, 2): 'NP', (1, 2): 'Noun'}
```

```
False
```

Question 1.2 (15 points) Implement the parsing version of CKY, which returns one of the legal parses for the sentence (and returns None if there are none). If there are multiple real parses, we don't care which one you print. Implement this as `cky_parse()`. You probably want to start by copying your `cky_acceptance()` answer and modifying it. Have it return the parse in the following format, using nested lists to represent the tree (this is a simple Python variant of the Lisp-style S-expression that's usually used for this.)

```
['S',          [['NP', [['Det', 'the'], ['Noun', 'cat']]],          ['VP',
[['Verb', 'attacked'],                      ['NP', [['Det', 'the'], ['Noun',
'food']]]]]]]
```

Print out the parses for the following sentences.
* the cat saw a dog * the cat saw a dog in a table * the cat with a table attacked the food

Hint: In the chart cells, you will now have to store backpointers as well. One way to do it is to store a list of tuples, each of which is (`nonterminal, splitpoint, leftchild nonterm, rightchild nonterm`). For example, if the state (`'NP', 3, 'Det', 'Noun'`) is in the cell for span (2,4), that means this is a chart state of symbol NP, which came from a Det at position (2,3) and a Noun at position (3,4).

Hint: It may be useful to use a recursive function for the backtrace.

```
In [3]: # Output the results for each sentence.
        #TODO: Print out the parse tree for each of the three sentence
        pprint( cky.cky_parse(['the','cat','with','a','table','attacked','the','food']) )
        print "*" * 50
        pprint( cky.cky_parse(["the", "cats", "saw", "a", "dog"]))
        print "*" * 50
        pprint( cky.cky_parse(["the", "cats", "with", "a", "table","attacked","the","food"]))
        #pprint( cky.cky_parse(['the','the']) )
```

```
{(0, 1): ['Det', 'the'],
 (0, 2): ['NP', [1, 'Det', 'Noun'], [4, 'Det', 'Noun'], [7, 'Det', 'Noun']],
 (0, 3): [],
 (0, 4): [],
 (0, 5): ['NP', [2, 'NP', 'PP']],
 (0, 6): [],
 (0, 7): [],
 (0, 8): ['S', [5, 'NP', 'VP'], [5, 'NP', 'VP']],
 (1, 2): ['Noun', 'cat'],
 (1, 3): [],
 (1, 4): [],
 (1, 5): [],
 (1, 6): [],
 (1, 7): [],
 (1, 8): [],
 (2, 3): ['Prep', 'with'],
 (2, 4): [],
 (2, 5): ['PP', [3, 'Prep', 'NP']],
```

```
 (2, 6): [],
 (2, 7): [],
 (2, 8): [],
 (3, 4): ['Det', 'a'],
 (3, 5): ['NP', [1, 'Det', 'Noun'], [4, 'Det', 'Noun'], [7, 'Det', 'Noun']],
 (3, 6): [],
 (3, 7): [],
 (3, 8): ['S', [5, 'NP', 'VP'], [5, 'NP', 'VP']],
 (4, 5): ['Noun', 'table'],
 (4, 6): [],
 (4, 7): [],
 (4, 8): [],
 (5, 6): ['Verb', 'attacked'],
 (5, 7): [],
 (5, 8): ['VP', [6, 'Verb', 'NP']],
 (6, 7): ['Det', 'the'],
 (6, 8): ['NP', [1, 'Det', 'Noun'], [4, 'Det', 'Noun'], [7, 'Det', 'Noun']],
 (7, 8): ['Noun', 'food']}
[S [NP [NP [Det the] ] [Noun cat] ] ] [PP [Prep with] ] [NP [Det a] ] [Noun table] ] ] ] ] [VP

True
**************************************************
{(0, 1): ['Det', 'the'],
 (0, 2): ['NPS', [1, 'Det', 'Nouns']],
 (0, 3): [],
 (0, 4): [],
 (0, 5): [],
 (1, 2): ['Nouns', 'cats'],
 (1, 3): [],
 (1, 4): [],
 (1, 5): [],
 (2, 3): ['Verb', 'saw'],
 (2, 4): [],
 (2, 5): ['VP', [6, 'Verb', 'NP'], [3, 'Verb', 'NP']],
 (3, 4): ['Det', 'a'],
 (3, 5): ['NP',
          [1, 'Det', 'Noun'],
          [4, 'Det', 'Noun'],
          [7, 'Det', 'Noun'],
          [4, 'Det', 'Noun']],
 (4, 5): ['Noun', 'dog']}
None
**************************************************
{(0, 1): ['Det', 'the'],
 (0, 2): ['NPS', [1, 'Det', 'Nouns'], [1, 'Det', 'Nouns']],
 (0, 3): [],
 (0, 4): [],
 (0, 5): ['NPS', [2, 'NPS', 'PP']],
```

```
(0, 6): [],
(0, 7): [],
(0, 8): [],
(1, 2): ['Nouns', 'cats'],
(1, 3): [],
(1, 4): [],
(1, 5): [],
(1, 6): [],
(1, 7): [],
(1, 8): [],
(2, 3): ['Prep', 'with'],
(2, 4): [],
(2, 5): ['PP', [3, 'Prep', 'NP'], [3, 'Prep', 'NP']],
(2, 6): [],
(2, 7): [],
(2, 8): [],
(3, 4): ['Det', 'a'],
(3, 5): ['NP',
        [1, 'Det', 'Noun'],
        [4, 'Det', 'Noun'],
        [7, 'Det', 'Noun'],
        [4, 'Det', 'Noun'],
        [4, 'Det', 'Noun'],
        [7, 'Det', 'Noun']],
(3, 6): [],
(3, 7): [],
(3, 8): ['S', [5, 'NP', 'VP'], [5, 'NP', 'VP'], [5, 'NP', 'VP']],
(4, 5): ['Noun', 'table'],
(4, 6): [],
(4, 7): [],
(4, 8): [],
(5, 6): ['Verb', 'attacked'],
(5, 7): [],
(5, 8): ['VP', [6, 'Verb', 'NP'], [3, 'Verb', 'NP'], [6, 'Verb', 'NP']],
(6, 7): ['Det', 'the'],
(6, 8): ['NP',
        [1, 'Det', 'Noun'],
        [4, 'Det', 'Noun'],
        [7, 'Det', 'Noun'],
        [4, 'Det', 'Noun'],
        [4, 'Det', 'Noun'],
        [7, 'Det', 'Noun']],
(7, 8): ['Noun', 'food']}
None
```

Question 1.3 (7 points)   Revise the grammar as follows.

- Add four words to the lexicon: two verbs "attack" and "attacks", and the nouns "cats" and "dogs".
- Revise the rules to enforce subject-verb agreement on number.

The new grammar should accept and reject the following sentences. Please run your parser on these sentences and report the parse trees for the accepted ones. Also, describe how you changed the grammar, and why.

ACCEPT: `the cat attacks the dog`
REJECT: `the cat attack the dog`
ACCEPT: `the cats attack the dog`
REJECT: `the cat with the food on a dog attack the dog`

Hint: you will need to introduce new nonterminal symbols, and modify the currently existing ones.

```
In [4]:  # Output the results for each sentence.
         #TODO: Print out the parse tree for each of the four sentence
         #print "as"
         #pprint( cky.cky_acceptance(['the','cats','attack','the','dog']) )
         pprint( cky.cky_parse(['the','cat','attacks','the','dog']) )
         print "*" * 50
         pprint( cky.cky_parse(['the','cat','attack','the','dog']) )
         print "*" * 50
```

```
{(0, 1): ['Det', 'the'],
 (0, 2): ['NP',
          [1, 'Det', 'Noun'],
          [4, 'Det', 'Noun'],
          [7, 'Det', 'Noun'],
          [4, 'Det', 'Noun'],
          [4, 'Det', 'Noun'],
          [7, 'Det', 'Noun'],
          [1, 'Det', 'Noun'],
          [4, 'Det', 'Noun']],
 (0, 3): [],
 (0, 4): [],
 (0, 5): ['S',
          [5, 'NP', 'VP'],
          [5, 'NP', 'VP'],
          [5, 'NP', 'VP'],
          [2, 'NP', 'VP']],
 (1, 2): ['Noun', 'cat'],
 (1, 3): [],
 (1, 4): [],
 (1, 5): [],
 (2, 3): ['Verb', 'attacks'],
 (2, 4): [],
 (2, 5): ['VP',
          [6, 'Verb', 'NP'],
```

```
                [3, 'Verb', 'NP'],
                [6, 'Verb', 'NP'],
                [3, 'Verb', 'NP']]),
 (3, 4): ['Det', 'the'],
 (3, 5): ['NP',
                [1, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [7, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [7, 'Det', 'Noun'],
                [1, 'Det', 'Noun'],
                [4, 'Det', 'Noun']],
 (4, 5): ['Noun', 'dog']}
[S [NP [Det the] ] [Noun cat] ] ] [VP [Verb attacks] ] [NP [Det the] ] [Noun dog] ] ] ]

True
***************************************************
{(0, 1): ['Det', 'the'],
 (0, 2): ['NP',
                [1, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [7, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [7, 'Det', 'Noun'],
                [1, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [1, 'Det', 'Noun'],
                [4, 'Det', 'Noun']],
 (0, 3): [],
 (0, 4): [],
 (0, 5): [],
 (1, 2): ['Noun', 'cat'],
 (1, 3): [],
 (1, 4): [],
 (1, 5): [],
 (2, 3): ['Verbs', 'attack'],
 (2, 4): [],
 (2, 5): ['VPS', [3, 'Verbs', 'NP']],
 (3, 4): ['Det', 'the'],
 (3, 5): ['NP',
                [1, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [7, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [4, 'Det', 'Noun'],
                [7, 'Det', 'Noun'],
```

```
          [1, 'Det', 'Noun'],
          [4, 'Det', 'Noun'],
          [1, 'Det', 'Noun'],
          [4, 'Det', 'Noun']],
 (4, 5): ['Noun', 'dog']}
None
*************************************************
```

1.0.2 Answer:

I created two new word categories to incorporate the plural nouns and the corresponding verbs that would go with it. 'Noun': set(['cat', 'dog', 'table', 'food']),

```
'Nouns':set(['cats','dogs']),
```

```
'Verbs':set(['attack']),
```

```
'Verb': set(['attacked', 'saw', 'loved', 'hated','attacks']),
```

I also included two new Non Terminals which to include the new word categories.I made changes to the grammar rules to include the two new NT NPS and VPS.

S -> NPS VPS
NPS ->Det Nouns
VPS ->Verbs NPS
VPS ->Verbs NP
PP ->Prep NPS
NPS ->NPS PP
VPS -> VPS PP

1.0.3 2: Weighted CKY Algorithm (40 points)

In this section you will implement the weighted CKY Algorithm for a Probabilistic CFG. You will have to make modifications to the existing algorithm to account for the probabilities and your parse function should output the most probable parse tree. Please write all your code in `weighted_cky.py` file for this section.

Question 2.1 (7 points) The CKY Algorithm requires the CFG to be in Chomsky Normal Form. Convert the following CFG into Chomsky Normal Form. (For the sake of uniformity, replace the leftmost pairs of non-terminals with new non-terminal)

S -> Aux NP VP
S -> VP
VP -> Verb NP
VP -> VP PP
Verb -> book
Aux -> does

1.0.4 Answer here

S -> Aux_NP VP
    Aux_NP ->Aux NP
    VP -> Verb NP
    VP -> VP PP
    S-> Verb NP
    S -> VP PP
    Verb -> book
    Aux -> does

Question 2.2 (8 points) We will now implement a weighted CYK algorithm to parse a sentence and return the most probable parse tree. The grammar is defined in `pcfg_grammar_original.txt`. As you can notice, some of the rules are not in CNF. Modify the `pcfg_grammar_modified.txt` file such that all the rules are in Chomsky Normal Form. (For the sake of uniformity, replace the leftmost pairs of non-terminals with new non-terminal)

Note: When transforming the grammar to CNF, must set production probabilities to preserve the probability of derivations.
    S -> NP VP 0.8
    S -> Aux_NP VP 0.1
    Aux_NP -> Aux NP 1
    S -> book 0.01
    S -> include 0.004
    S -> prefer 0.006
    S -> Verb NP 0.05
    S -> VP PP 0.03
    NP -> I 0.1
    NP -> he 0.02
    NP -> she 0.02
    NP -> me 0.06
    NP -> Houston 0.16
    NP -> NWA 0.04
    NP -> Det Nominal 0.6
    Nominal -> book 0.03
    Nominal -> flight 0.15
    Nominal -> meal 0.06
    Nominal -> money 0.06
    Nominal -> Nominal Noun 0.2
    Nominal -> Nominal PP 0.5
    VP -> book 0.1
    VP -> include 0.04
    VP -> prefer 0.06
    VP -> Verb NP 0.5
    VP -> VP PP 0.3
    PP -> Prep NP 1.0
    Det -> the 0.6
    Det -> a 0.2
    Det -> that 0.1

Det -> this 0.1
Noun -> book 0.1
Noun -> flight 0.5
Noun -> meal 0.2
Noun -> money 0.2
Verb -> book 0.5
Verb -> include 0.2
Verb -> prefer 0.3
Pronoun -> I 0.5
Pronoun -> he 0.1
Pronoun -> she 0.1
Pronoun -> me 0.3
Proper-Noun -> Houston 0.8
Proper-Noun -> NWA 0.2
Aux -> does 1.0
Prep -> from 0.25
Prep -> to 0.25
Prep -> on 0.1
Prep -> near 0.2
Prep -> through 0.2
Question 2.3 (5 points)

Explain briefly what are the changes you made to convert the grammar into CNF Form. Why did you make these changes?

1.0.5 Answer here

I made sure there are exactly two Non-Terminals in each grammar rule as CNF should contain only 2 NT in a rule. Also each grammar rule should not contain either two NT or a Terminal. So I replaced rules which did not adhere to the above conditions by creating new Non-Terminals.

As for the probability, in the case where I add a new NT to remove 3 NTS from a rule, I kept the probabilty of such change as 1. The probabilities in the new grammar have to be such that you'll get equivalent parse probabilities as for parsing in the original grammar.

Question 2.4 (8 points) Complete the `populate_grammar_rules()` function in the `weighted_cky.py` script. This function will have to read in the grammar rules from `pcfg_grammar_modified.txt` file and populate the `grammar_rules` and `lexicon` data structure. Additionally you would need to store the probability mapping in a suitable data structure.

Hint: You can modify the starter code provided in cky.py for this task.

```
In [2]: from weighted_cky import populate_grammar_rules
        populate_grammar_rules()

Grammar rules in tuple form:
[('S', ('NP', 'VP')),
 ('S', ('Aux_NP', 'VP')),
 ('Aux_NP', ('Aux', 'NP')),
 ('S', ('Verb', 'NP')),
 ('S', ('VP', 'PP')),
```

```
  ('NP', ('Det', 'Nominal')),
  ('Nominal', ('Nominal', 'Noun')),
  ('Nominal', ('Nominal', 'PP')),
  ('VP', ('Verb', 'NP')),
  ('VP', ('VP', 'PP')),
  ('PP', ('Prep', 'NP'))]
Rule parents indexed by children:
{('Aux', 'NP'): ['Aux_NP'],
 ('Aux_NP', 'VP'): ['S'],
 ('Det', 'Nominal'): ['NP'],
 ('NP', 'VP'): ['S'],
 ('Nominal', 'Noun'): ['Nominal'],
 ('Nominal', 'PP'): ['Nominal'],
 ('Prep', 'NP'): ['PP'],
 ('VP', 'PP'): ['S', 'VP'],
 ('Verb', 'NP'): ['S', 'VP']}
probabilities
{('Aux', 'does'): ['1.0'],
 ('Aux_NP', ('Aux', 'NP')): ['1'],
 ('Det', 'a'): ['0.2'],
 ('Det', 'that'): ['0.1'],
 ('Det', 'the'): ['0.6'],
 ('Det', 'this'): ['0.1'],
 ('NP', 'Houston'): ['0.16'],
 ('NP', 'I'): ['0.1'],
 ('NP', 'NWA'): ['0.04'],
 ('NP', 'he'): ['0.02'],
 ('NP', 'me'): ['0.06'],
 ('NP', 'she'): ['0.02'],
 ('NP', ('Det', 'Nominal')): ['0.6'],
 ('Nominal', 'book'): ['0.03'],
 ('Nominal', 'flight'): ['0.15'],
 ('Nominal', 'meal'): ['0.06'],
 ('Nominal', 'money'): ['0.06'],
 ('Nominal', ('Nominal', 'Noun')): ['0.2'],
 ('Nominal', ('Nominal', 'PP')): ['0.5'],
 ('Noun', 'book'): ['0.1'],
 ('Noun', 'flight'): ['0.5'],
 ('Noun', 'meal'): ['0.2'],
 ('Noun', 'money'): ['0.2'],
 ('PP', ('Prep', 'NP')): ['1.0'],
 ('Prep', 'from'): ['0.25'],
 ('Prep', 'near'): ['0.2'],
 ('Prep', 'on'): ['0.1'],
 ('Prep', 'through'): ['0.2'],
 ('Prep', 'to'): ['0.25'],
 ('Pronoun', 'I'): ['0.5'],
 ('Pronoun', 'he'): ['0.1'],
```

```
('Pronoun', 'me'): ['0.3'],
('Pronoun', 'she'): ['0.1'],
('Proper-Noun', 'Houston'): ['0.8'],
('Proper-Noun', 'NWA'): ['0.2'],
('S', 'book'): ['0.01'],
('S', 'include'): ['0.004'],
('S', 'prefer'): ['0.006'],
('S', ('Aux_NP', 'VP')): ['0.1'],
('S', ('NP', 'VP')): ['0.8'],
('S', ('VP', 'PP')): ['0.03'],
('S', ('Verb', 'NP')): ['0.05'],
('VP', 'book'): ['0.1'],
('VP', 'include'): ['0.04'],
('VP', 'prefer'): ['0.06'],
('VP', ('VP', 'PP')): ['0.3'],
('VP', ('Verb', 'NP')): ['0.5'],
('Verb', 'book'): ['0.5'],
('Verb', 'include'): ['0.2'],
('Verb', 'prefer'): ['0.3']}
Lexicon
{'Aux': ['does'],
 'Det': ['the', 'a', 'that', 'this'],
 'NP': ['I', 'he', 'she', 'me', 'Houston', 'NWA'],
 'Nominal': ['book', 'flight', 'meal', 'money'],
 'Noun': ['book', 'flight', 'meal', 'money'],
 'Prep': ['from', 'to', 'on', 'near', 'through'],
 'Pronoun': ['I', 'he', 'she', 'me'],
 'Proper-Noun': ['Houston', 'NWA'],
 'S': ['book', 'include', 'prefer'],
 'VP': ['book', 'include', 'prefer'],
 'Verb': ['book', 'include', 'prefer']}
```

Question 2.5 (12 points) Implement the weighted parsing version of CKY, which returns the most probable legal parse for the sentence (and returns None if there are none). If there are multiple real parses, this function will always return the most probable parse i.e the one with maximum probability. Complete the `pcky_parse()`. Print the parse tree and the probabilities for the following sentences: * book the flight through Houston * include this book * the the

Hint: You can use the code in `cky_parse()` and modify it to accomodate probabilities and compute the most probable parse.

Note: The topmost cell should contain rules associated with the S non terminal, if any.

```
In [5]: from weighted_cky import pcky_parse
        # Output the results for each sentence.
        #TODO: Print out the parse tree for each of the three sentence

        pprint(pcky_parse(['include','this','book']) )
        print "*" * 50
```

```
          pprint( pcky_parse(['book','this','flight','through','Houston']) )
          print "*" * 50
          pprint(pcky_parse(['the','the']) )
          print "*" * 50

("[ 'S' , [ 'Verb' , 'include', 0.2] , [ 'NP' , [ 'Det' , 'this', 0.1] , [ 'Nominal' , 'book',
 1.800000000000004e-05)
**************************************************
("[ 'S' , [ 'Verb' , 'book', 0.5] , [ 'NP' , [ 'Det' , 'this', 0.1] , [ 'Nominal' , [ 'Nominal
 3.5999999999999994e-06)
**************************************************
S not found
None
**************************************************
```

1.0.6   3: Dependency parser output (30 points)

You will conduct manual error analysis of CoreNLP's dependency parser.

Create an English sentence where the parser makes an error, and you know what the correct analysis ought to be, according to the Universal Dependencies grammatical standard. You will want to play around with different sentences, look at their output, and check against the Universal Dependencies annotation standard. The current version of CoreNLP outputs according to the "UD version 1" standard, so please use this page: * UD v1 homepage * and in particular, the UD v1 dependency relations list

For quickly looking at things, their online demo may be useful.

However, for this assignment, you need to run the parser to output in "conllu" format, which is human readable. You need to download and run the parser for this. (It requires Java.) Use version 3.8.0 (it should be the current version). You can it working in interactive mode so you can just type sentences into it on the terminal like this:

```
./corenlp.sh -annotators tokenize,ssplit,pos,lemma,depparse -outputFormat conllu
[...]
Entering interactive shell. Type q RETURN or EOF to quit.
NLP>
```

For example then you can type

```
NLP> I saw a cat.
1       I       I       _       PRP     _       2       nsubj   _       _
2       saw     see     _       VBD     _       0       root    _       _
3       a       a       _       DT      _       4       det     _       _
4       cat     cat     _       NN      _       2       dobj    _       _
5       .       .       _       .       _       2       punct   _       _
```

You can also use the -inputFile flag if you'd rather give it a whole file at once.

As you can see in the parser documentation, the 7th and 8th columns describe the dependency edge for the word's parent (a.k.a governor): it has the index of its parent, and the edge label (a.k.a.

the relation). For example, this parse contains the dependency edge *nsubj(saw:2, I:1)* meaning that "I" is the subject of "saw".

Question 3.1: Once you've decided your sentence, please put the conllu-formatted parser output below in the markdown triple-quoted area. Please be very careful where it goes since we will use a script to pull it out.

PARSE GOES BELOW HERE

```
NLP> The President and the Speaker, Donald  Trump and Paul Ryan respectively, find themselves
1    The the _   DT  _   2   det _   _
2    President   President    _   NNP _   14  nsubj   _   _
3    and and _   CC  _   2   cc  _   _
4    the the _   DT  _   5   det _   _
5    Speaker Speaker _   NNP _   2   conj    _   _
6    ,   ,   _   ,   _   2   punct   _   _
7    Donald  Donald  _   NNP _   8   compound    _   _
8    Trump   Trump   _   NNP _   2   conj    _   _
9    and and _   CC  _   2   cc  _   _
10   Paul    Paul    _   NNP _   11  compound    _   _
11   Ryan    Ryan    _   NNP _   2   conj    _   _
12   respectively    respectively    _   RB  _   2   advmod  _   _
13   ,   ,   _   ,   _   2   punct   _   _
14   find    find    _   VBP _   0   root    _   _
15   themselves  themselves  _   PRP _   14  dobj    _   _
16   on  on  _   IN  _   19  case    _   _
17   the the _   DT  _   19  det _   _
18   opposite    opposite    _   JJ  _   19  amod    _   _
19   ends    end _   NNS _   14  nmod    _   _
20   with    with    _   IN  _   21  case    _   _
21   regards regard  _   VBZ _   14  nmod    _   _
22   to  to  _   TO  _   24  case    _   _
23   Roy Roy _   NNP _   24  compound    _   _
24   Moore   Moore   _   NNP _   21  nmod    _   _
25   .   .   _   .   _   14  punct   _   _
```

PARSE GOES ABOVE HERE

Question 3.2: Please describe the error you found. Also give a citation and link to the relevant part of the UD documentation describing one of the relations that the parser predicted in error, or did something wrong with.

The parser does not seem to match the two persons with their positions. It uses a conjuction to connect Donald Trump and Paul Ryan and the words President and Speaker. It also correctly guesses Donald Trump as the president. The only mistake is it does not recognise Paul Ryan as the Speaker or the word respectively.Since they seem to be opposing each other in this context,I do feel this is a The internal structure over the noun phrase is a little fishy.It should contain an apposition of Paul Ryan and Donald Trump.

http://universaldependencies.org/docsv1/u/dep/appos.html http://universaldependencies.org/docsv1/u

Question 3.3: Please give correct that error in the parse . Put your corrected parse, again in that conllu textual format, below. You should take a copy of the output and manually change some of the 7th/8th dependency edge columns.

PARSE GOES BELOW HERE

```
NLP> The President Donald Trump and the Speaker Paul Ryan , find themselves on the opposite sid
1    The the _    DT  _   4   det _   _
2    President   President    _   NNP _   4   compound    _   _
3    Donald  Donald  _   NNP _   4   compound    _   _
4    Trump   Trump   _   NNP _   11  nsubj   _   _
5    and and _   CC  _   4   cc  _   _
6    the the _   DT  _   9   det _   _
7    Speaker Speaker _   NNP _   9   compound    _   _
8    Paul    Paul    _   NNP _   9   compound    _   _
9    Ryan    Ryan    _   NNP _   4   conj    _   _
10   ,   ,   _   ,   _   4   punct   _   _
11   find    find    _   VBP _   0   root    _   _
12   themselves  themselves  _   PRP _   11  dobj    _   _
13   on  on  _   IN  _   16  case    _   _
14   the the _   DT  _   16  det _   _
15   opposite    opposite    _   JJ  _   16  amod    _   _
16   sides   side    _   NNS _   11  nmod    _   _
17   with    with    _   IN  _   18  case    _   _
18   regards regard  _   VBZ _   11  nmod    _   _
19   to  to  _   TO  _   21  case    _   _
20   Roy Roy _   NNP _   21  compound    _   _
21   Moore   Moore   _   NNP _   18  nmod    _   _
```

PARSE GOES ABOVE HERE

Question 3.4: Please describe your correction and why it solves the error.

I modified the internal structure of the noun phrase and seperated their names and their positions to remove ambigiuity for the parser. The change is from "The President and the Speaker, Donald Trump and Paul Ryan respectively" to "The President Donald Trump and the Speaker Paul Ryan" Since there is a clear distinction between the two names and their respective positions now, the parser is able to detect the difference.