Implementing the CKY algorithm

LIN2571 Computational Morphology and Syntax
Project Task 3 — part 2

Claudia Borg

March 19, 2017

1 Background

The aim of this task is to implement the CKY algorithm and understand the process of how constituent parse trees are produced in a bottom-up fashion. Our focus will be on a non-probabilistic implementation of the CKY parser. Make sure to read all the material covered in class to understand the CKY algorithm before attempting this task.

2 Task

A common grammar in Chomsky Normal Form is being provided - `grammar.cnf`. However you are free to use the (CNF) grammar that you developed as well (see note further on).

Your main task is to create a CKYParser that will take as input (i) the CNF grammar and (ii) a file containing sentences (the one we have used already). It will then return all possible parses for each sentence as output. `NLTK` provides a number of functions that you will find useful:

Accessing the Grammar In order to access the grammar you can use the following NLTK functions:

- load the grammar using `g = nltk.data.load(filename)` When you load the grammar you can check for a number of things, including:

```
>>> g
<Grammar with 351 productions>
>>> g.is_binarised()
True
>>> g.is_chomsky_normal_form()
True
```

- Access the production rules: `g.productions()`
- Print the information about the grammar: `print(g)`
- Check which is the start symbol: `g._start`
- Check for a word: `g.productions(rhs='sat')`
- Retrieve a list of non terminals in the grammar: `g._categories`
- When you want to check if a rule of the shape `A -> B C` exists, you must query the grammar in terms of `B` (the first Nonterminal on the rhs). For example, you want to check for the rules `A -> VBZ VP`. To do so, first you retrieve the productions containing `VBZ`, using the following:

  ```
  >>> from nltk.grammar import Nonterminal
  >>> curr_rules = g.productions(rhs=Nonterminal('VBZ'))
  [TOP -> VBZ NP, TOP -> VBZ ADJP, TOP -> VBZ VP, TOP -> VBZ NP,
  TOP -> VBZ ADJP, TOP -> VBZ VP, S -> VBZ NP, S -> VBZ ADJP,
  S -> VBZ VP, X9 -> VBZ NP, X13 -> VBZ VBG, VP -> VBZ NP,
  VP -> VBZ ADJP, VP -> VBZ VP]
  >>>
  ```

  This returns a list of production rules containing `VBZ` as the first item on the `rhs`. Next you iterate through every production `p` and look for the second element of the tuple, in this example `VP`. This query would look like:
  `p._rhs[1] == Nonterminal('VP')`
  Where there is a match, you will need to retrieve the lhs of the rule: `p.lhs()`, and thus retrieving `A`.

CKY Implementation Understanding how to handle the grammar through the NLTK functions will make things easier. Now we turn our attention to the actual implementation of the algorithm. Remember that we need to use a table where we will store in the individual cells the things that we retrieve from our grammar. These are returned as either Nonterminal objects or strings in the case of the terminals.

The best approach to implement the algorithm is to go through the notes, write down on paper what the cells need to store, and only then start implementing.

Suggestion: The first version of the CKY can simply return whether it found a valid parse tree for a sentence or not. Then implement the 'backtracking' option to return the number of suggested trees. Although NLTK provides a tree drawing function, you do not need to use this. Outputting a tree in a textual format is good enough.

If at any point in time you have questions, (i) there is the forum, and (ii) send me an email with your code. Also, feel free to answer questions posed by your colleagues in the forum.

3 Deliverables and practicalities

As a deliverable, you should submit:

1. The python script
2. The output of your script
3. The documentation – this should describe what the important functions/parts of your code do, what problems you encountered, how you tried to solve them, any known bugs, suggested improvements to known bugs that you didn't have the chance to implement.

Due date: 19th May - submission via `VLE`. Via email is optional if you want to confirm 100% that you submitted your task on time, especially in the case of technical problems with the `VLE`
Late submission: A 10% deduction from the grade of this task per every late day.
Submission type: ZIP file containing all the above, please name the zip file as: NameSurname.zip
Marking Scheme:
General code clarity: 10 marks
CKY Implementation: 50 marks
Parsing output: 20 marks
Documentation: 20 marks