Computational Linguistics

Assignment 3 (2021-11-19)

Winter Semester 2021/22 – Prof. Dr. Alexander Koller

Context-free grammars and CKY parsing

In this assignment, you get to experiment with large-scale context-free grammar (CFG) parsing using Python and NLTK. More specifically, we are asking you to implement the Cocke-Kasami-Younger (CKY) algorithm for bottom-up CFG parsing, and apply it to the word and the parsing problem of English.

The ingredients are: the grammar, the test sentences, and the parser. We provide the first two ingredients. Please implement the parser from scratch, using NLTK only to represent grammars and trees.

The grammar stems from a project dealing with implementing spoken language processing systems in the airline industry – the Airline Travel Information System (ATIS). The ATIS CFG is available in the NLTK `data` package, together with 98 test sentences. You can initialize the resources this way:

```
grammar = nltk.data.load("grammars/large_grammars/atis.cfg")     # load the grammar
s = nltk.data.load("grammars/large_grammars/atis_sentences.txt") # load raw sentences
t = nltk.parse.util.extract_test_sentences(s)                    # extract test sentences
```

At this point, `t` is a list of 2-tuples, one per sentence. The first element of each tuple is the tokenized sentence, and the second element is the number of parses for that sentence according to the `grammar`. Note that this number can be zero.

NLTK already implements a number of parsing algorithms (see the documentation of `nltk.parse` for the list). You can try one to see if you loaded the grammar correctly:

```
# initialize the parser
parser = nltk.parse.BottomUpChartParser(grammar)
# parse all test sentences
for sentence in t:
    parser.chart_parse(sentence[0])
```

However, the NLTK version of the ATIS grammar is not in Chomsky normal form (CNF), which you will need for your CKY parser. Feel free to implement a conversion module for extra credit, but for your convenience, we have already converted the ATIS CFG into CNF; you can download it from `http://www.coli.uni-saarland.de/~koller/materials/anlp/atis.zip`. You can then read the grammar from the file using `CFG.fromstring()` and utilize the features of the `nltk.grammar` module on the resulting object.

Recognizer. Implement the CKY algorithm and use it as a recognizer. That is, given an input sentence, the procedure should decide whether the sentence is in the language of the CFG or not. Test the recognizer on the ATIS test sentences (not all of which are grammatical), but also by feeding it other sentences to see whether it properly rejects ungrammatical sentences as well. Submit some of the ungrammatical sentences you tried.

Parser. Now extend your CKY recognizer into a parser by adding backpointers. Also implement a function that extracts the set of all parse trees from the backpointers in the chart. Feel free to use the NLTK module `nltk.tree` for this purpose; notice that only `ImmutableTree`s can be used as elements of Python sets, whereas raw `Tree`s cannot.

Submit your code, outputs, and a `README` file. The outputs should consist of at least: (1) your parsing results (number of parses per sentence), as a text file with one line of the form `sentence\t#parses` for each test sentence, where `\t` is a tab character; (2) pictures of the parse trees for an ATIS test sentence with two to five parses. You can visualize an NLTK tree using its `draw` method, and check your parse tree counts against the counts in NLTK.

Extra credit. If you still have time left, here's a project for extra credit. Perhaps it has occurred to you that it is quite wasteful to compute all parse trees just to find out how many parse trees there are. Figure out how to compute the number of parse trees for an entry $A \in \mathrm{Ch}(i, k)$ from your chart with backpointers, without actually computing these parse trees. Verify that you get the correct results, and compare the efficiency of your new procedure to your earlier solution.

Turn in before class on 2021-12-03 on Google Classroom.