Advanced Natural Language Processing

Assignment 4: CKY Parsing

Credits:

1 Introduction

In this assignment, you will implement the CKY algorithm for English and apply it to the word recognition and parsing problem. You can use the NLTK modules for representing context-free grammars and parse trees, but you should implement the parser from scratch. (But you don't have to use `nltk.grammar` if you find it easier without. Some NLTK modules are overly complicated, and in this assignment you mainly need a lookup function from right-hand-sides of grammar rules to left-hand-sides.)

2 Getting started

You should start by unpacking the archive `assignment4.zip`. This results in a directory `assignment4` with the following structure:

```
- data/
--- atis-grammar-cnf.cfg
- model/
--- __init__.py
--- parser.py
--- recognizer.py
- assignment4.py
```

3 ATIS Data

We provide the grammar and the test sentences. The grammar stems from the Airline Travel Information System (ATIS), a project working on spoken dialog systems for air travel. (What do you think, is this a grammar motivated by syntactic theory, or perhaps more motivated by other concerns? Does it look like the sample grammars we've seen in class, only larger, or does it fundamentally look different?) The ATIS CFG is available in the NLTK data package, together with 98 test sentences. The resources can be initialized this way:

```
import nltk
# load the grammar
grammar = nltk.data.load("grammars/large_grammars/atis.cfg")
# load the raw sentences
s = nltk.data.load("grammars/large_grammars/atis_sentences.txt", "auto")
# extract the test sentences
t = nltk.parse.util.extract_test_sentences(s)
# initialize the parser
parser = nltk.parse.BottomUpChartParser(grammar)
# parse all test sentences
for sentence in t[:10]:
    parser.chart_parse(sentence[0])
```

NLTK already implements a number of parsing algorithms (see `nltk.parse` for the list). You can try one to see if you loaded the grammar correctly. Note that NLTK's `chart_parse()` throws an error when it encounters an unknown word, which is undesirable behavior for any parser. If you want to test the grammars with the pre-implemented parser (for example, to check whether the CNF version is in fact weakly equivalent), you may need to catch this error.

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, provided with this assignment. You can then read the grammar from the file using `nltk.data.load()` and utilize the features of the `nltk.grammar` module on the resulting object.

4 Starter Code

Let's look at the main program in `assignment4.py`. It is supposed to ...

1. reads in the ATIS dataset using `nltk.data.load()`,
2. Use a CKY algorithm as a recognizer to recognize whether if a sentence is in the language of CFG.
3. Extend the CKY recognizer into a parser and evaluate it on the test data.

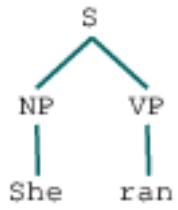
However, there are several missing features that need to be implemented. Let's implement them one by one!

4.1 Structural Ambiguity [20 pts]

First of all, devise at least two sentences that exhibit structural ambiguity (i.e., ambiguity that comes from different structural analyses, not from ambiguity in the meaning of an individual lexical item) to test your understanding. Indicate the different analyses (at least two per sentence) with a syntactic tree. Use the flag `--structural` when running `assignment4.py`.

You can draw trees by using the NLTK package:

```
from nltk.tree import Tree
Tree.fromstring('(S (NP (She)) (VP (ran)))')
```



4.2 Word Recognition [50 pts]

Next, implement the CKY algorithm in `model/recognizer.py` 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. Provide a list of grammatical and ungrammatical test

sentences (at least 10 each) and test your recognizer on these sentences. When you are done, run the main script `assignment4.py` with the flag `--recognizer` to check if your implementation is working properly.

4.3 Parsing [30 pts]

Now extend your CKY recognizer into a parser by adding backpointers in `model/parser.py`. You should implement functionality 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. Then, test the parser by providing the list of ATIS test sentences with tab-separated numbers of parse trees. At last, choose an ATIS test sentence with a number of parses p such that $1 < p < 5$. Provide pictures of its parses. You can visualize an NLTK tree using its `draw` method. **Discuss the structural differences**. Don't forget to use the flag `--parser` when executing this part of the assignment.

4.4 Bonus [10 pts]

If you still have time left, you can attempt the following 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 $A \in Ch(i, k)$ from your chart with backpointers, without actually computing these parse trees in `model/parser.py`. Verify that you get the correct results, and compare the efficiency of your new procedure to your earlier solution. (Don't forget to use the flag `--count` when running `assignment4.py`)

5 Submission

Upload your `assignment4_LASTNAME.zip` file on Moodle.