CMPUT 461/501 Intro to NLP
Assignment 2: Grammar Checker

Introduction

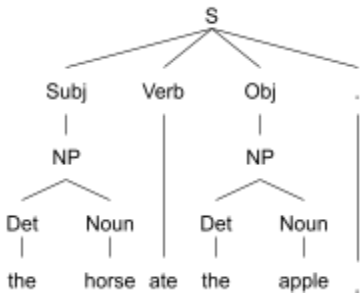
Grammar checkers like Grammarly or the one built in Microsoft Word might have helped you a lot when writing essays. In this assignment, you will use what you have learned about context-free grammars (CFG) and constituency parsing to build a very simple grammar checker of your own. We provide a dataset of sentences written by English learners with grammatical mistakes so you can evaluate the accuracy of your grammar checker. After that, you are required to analyse errors made by the grammar checker and report your findings.

So how exactly does our simple grammar checker work? First of all, we have to define the set of English grammar rules and make them machine-understandable. To be specific, we will try to come up with a very crude context-free grammar of English. After we have the CFG for English, we will run a constituency parser to parse the input sentence with that CFG. If the sentence can be successfully parsed with the CFG, we mark the input sentence as grammatically correct. And, if the sentence can not be parsed, it is considered ungrammatical.

Here is an example to illustrate our grammar checker. We start with a simplified CFG that only considers the basic structure in English.

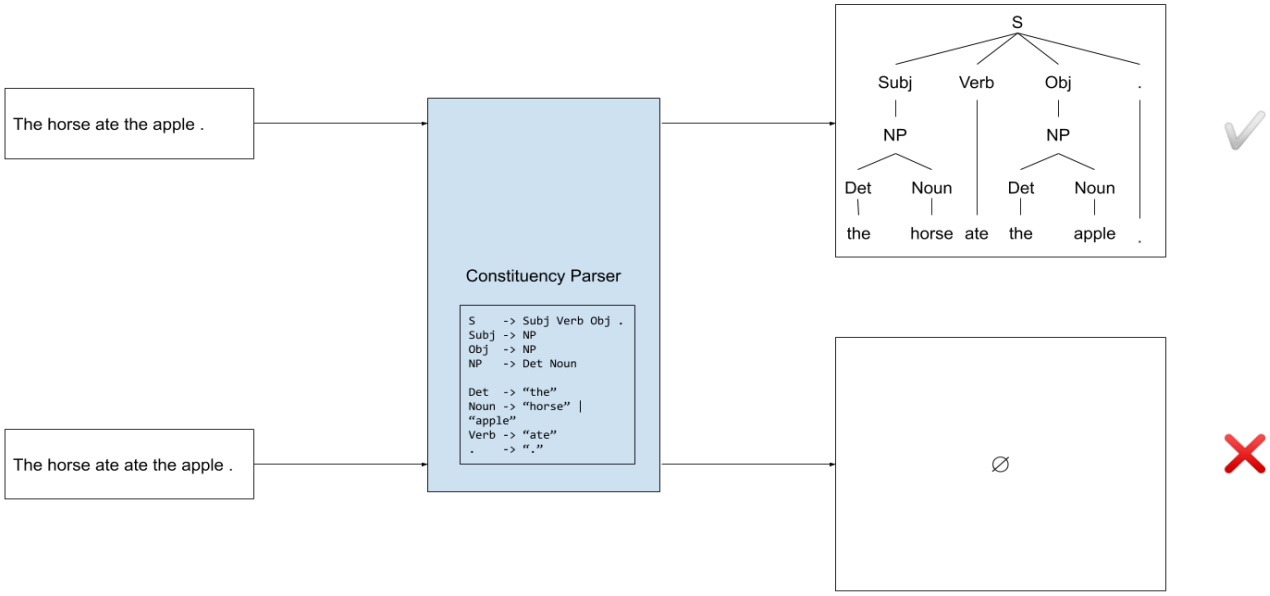
`S    -> Subj Verb Obj .`	`A sentence = subject + verb + object + period.`
`Subj -> NP`	`The subject is a noun phrase.`
`Obj  -> NP`	`The object is also a noun phrase.`
`NP   -> Det Noun`	`A noun phrase is a determiner + a noun.`
`Det  -> “the”`	`The terminal symbols (lexicon) are simplified`
`Noun -> “horse” | “apple”`	`for illustration purposes.`
`Verb -> “ate”`	
`.    -> “.”`	

If we use a constituency parser (for example, the CYK algorithm) with the above CFG to parse the sentence “*the horse ate the apple.*”, the parsing will succeed with an output like:



But if you try the same thing with an ungrammatical sentence like "*the horse ate ate the apple.*", the parser will fail. In this way, we will know that this sentence does not conform to the English grammar we defined and can call it ungrammatical.

This figure shows the workflow of our grammar checker.



Usually when writing grammars, one important component is the lexicon or the terminal symbols. For example, a grammar usually has a list of nouns like `Noun->"horse"|"apple"`. We simplify this process by providing the part of speech tags for all input sentences. Part of speech indicates the grammatical function of a word in a sentence. For example, whether a word is a noun, adjective, verb, or a preposition.

In this assignment, **instead of parsing the actual sentence, you will write a grammar and a parser to parse the POS tag sequence.** This will simplify the task and relieve you from building lexicons. For our above example "`The horse ate the apple.`", you will be given the POS tag sequence "`DT NN VBD DT NN .`". To modify the parser so it can parse the POS sequence, we replace the English words in our CFG with tag names.

`S -> Subj Verb Obj .`
`Subj -> NP`
`Obj -> NP`
`NP -> Det Noun`
 	
`Det -> "DT"`
`Noun -> "NN"`
`Verb -> "VBD"`
`. -> "."`
	`A sentence = subject + verb + object + period.`
	`The subject is a noun phrase.`
	`The object is also a noun phrase.`
	`A noun phrase is a determiner + a noun.`
	
	`No need to put an entire English dictionary here!`

Tasks

Input: sentences with POS tags

The input is a tsv (tab-separated values) file like the sample:

id	label	sentence	pos
73	0	Many thanks in advance for your cooperation .	JJ NNS IN NN IN PRP$ NN .
74	1	At that moment we saw the bus to come .	IN DT NN PRP VBD DT NN TO VB .

The `id` column is the unique id for each sentence. The `label` column indicates whether a sentence contains grammar errors (`1` means having errors and `0` means error-free). In the `sentence` column, the original sentence is already tokenized and separated by a single space, so you can use the `str.split()` function to get the tokens. The `pos` column contains the POS tags for each token in the sentence, also separated by a single space. The POS tags follow the Penn Treebank (PTB) tagging scheme, as described here.

Part 1: Building a toy grammar

The first step is to write a toy CFG for English in NLTK's `.cfg` format. You can all start with this grammar below (included in the repository) and try to modify the rules or add new production rules.

```
S  -> NP VP Punct          Sentence = noun phrase + verb phrase + .
PP -> Adp NP               Prepositional phrase = preposition + noun phrase
NP -> Det Noun | NP PP     Noun phrase
VP -> Verb NP | VP PP      Verb phrase
Verb -> VB | VBD | VBG     Include all the inflections of a verb
| VBN | VBP | VBZ
......                     More production rules go here

Det   -> 'DT'
Noun  -> 'NN'
VB    -> 'VB'              This part will be included in the template. It may
Adp   -> 'ADP'             need minor tweaks to work with NLTK. See the
Punct -> 'PUNCT'           definitions here.
......
```

Part 2: Constituency parsing

Use the chart parser from NLTK to parse each of the POS sequences in the dataset with the toy grammar you wrote in Part 1. The **results should be stored in a `TSV` file (NOT CSV)** with three columns:

Column name	Description
`id`	The id of the input sentence.
`ground_truth`	The ground truth label of the input sentence, copied from the dataset.
`prediction`	`1` if the sentence has grammar errors, `0` if not. In other words, whether the POS sequence can be parsed successfully with your grammar and parser.

Part 3: Evaluation and error analysis

In this part, you will evaluate the performance of your grammar checker by calculating its precision and recall on the data available to you. To do that, you will compare the prediction of your system on a given sentence and its corresponding label in the dataset. Each sentence will fall into one of the following four cases:

		`ground_truth`	
		1	0
`prediction`	1	true positive	false positive
	0	false negative	true negative

Define *TP*, *FP*, *FN* and *TN* as the **number of sentences in the data** falling in each of the four cases: True Positives, False Positives, False Negatives and True Negatives, respectively. The precision and recall are defined respectively as:

$$Precision = TP / (TP + FP)$$
$$Recall = TP / (TP + FN)$$

After getting these two numbers, please look at how and why your grammar checker did not perform well (if so). Identify **at most** 3 reasons for the false positives and **at most** 3 reasons for the false negatives.

Graduate Student Extension

Consider how you would approach this task differently if you wanted to also identify colloquial expressions based on the author's location or background. What would you need to do differently? What could be kept the same? What resources might you need to consult?

Please provide 2-3 paragraphs answering these questions.

Repository

Save the toy grammar you created in the `grammars/` folder. The grammars should have NLTK's `.cfg` format.

Write a program `src/main.py` that takes three positional command-line arguments in this order: the first is a path to the input data file, the second is a path to the grammar file, and the third is a path to the output TSV file. For example, the program **should be executed in this way:**

```
python3 src/main.py data/train.tsv grammars/toy.cfg output/train.tsv
```

Report

In your report, include the precision, recall and how and why the grammar checker produced false positives and negatives, as instructed in Part 2.

Answer the questions in the report:

- With our current design, is it possible to build a perfect grammar checker?
- If so, what resources or improvements are needed?
- If not, briefly justify your answer.

Documentation

Complete the TODO list in the README file.

You are encouraged to talk to others about the assignment, consult websites (i.e. Stack Overflow), use external libraries. If you use these resources **you must acknowledge them** in your README (with links if you consulted a website).

Suggestions

General suggestions

- If or when in doubt, ask for help right away.
- Don't push yourself too hard, as English is probably **NOT** a context-free language.
- Start early, familiarise yourself with the task and start thinking about it sooner rather than later.

- When building the toy grammar, look at English sentences from a corpus and try to sum up the rules. You are also encouraged to find resources about English grammar or CFGs for English.

Useful links

- Chapter 8 of the NLTK book introduces grammar, CFG, and constituency parsing in the context of NLTK.
- Appendix D of J+M introduces CFG and English grammar rules. Chapter 17 covers algorithms for constituency parsing.
- The PTB POS tag set is described here.
- The NLTK documentation gives more details about its APIs.

More information

Deadline	Tuesday, 24 October 2023, 11:55 PM
Topic	Grammars & Parsing
Book Chapter	Appendix D and Chapter 17
Learning Objectives	Learn how to use context-free grammar to formally model constituent structures in English, and how to use a parser for constituency parsing.

Acknowledgement

The sample toy grammar was adapted from the NLTK dataset `grammars/sample/toy.cfg`.