SI 630: Homework 3 – Parsing

Due: Sunday, February 25, 11:59pm

1 Introduction

Homework 3 is a two-part assignment. Part 1 will ask you to implement the Cocke-Younger-Kasami (CYK) algorithm for parsing probabilistic context-free grammars. The second part is a competitive grammar writing exercise. Part 1 should be done independently. Part 2 should be done in groups of three.

2 Task 1: Implementing a probabilistic CYK Parser

In Task 1, you'll implement a version of the CYK Parser that supports a grammar with probabilities assigned to each non-terminal's production rules. The probabilities will be used to break ties in ambiguous parses and to assign an overall probability for the whole sentence. For this task, you can assume a grammar and lexicon will be supplied for you; i.e., you do not need to worry about where you learn a grammar or the weight. However, we do note that learning the grammar and weights from data is a very important problem in NLP for developing parsers!

For this task, we've given you skeleton code `parser.py` that has the basic input and output functions specified. Your job will be to finish this code for two functions:

1. `load_grammar` This function will read in a file containing a grammar, where each non-terminal has weights (or probabilities) assigned to it. You'll store these grammar in the internal representation of your choosing. As a part of this function, you'll need to normalize the weights of each non-terminal's production rules to be probabilities (note: if the grammar you're give is already in terms of probabilities, this will have no effect!). All grammars that your code will read will always be in CNF.[1]

2. `parse` This function will parse a sentence and return (1) invalidParse, $True$ if the tree cannot be parsed and $False$ otherwise (2) max log probability of all possible parse tree for the input sentence and (3) backpointer table of this parse tree, as computed using the production rules' probabilities. Backpointer table is a upper triangular matrix with size of $(n-1) \times n$. Each cell in backpointer is a dictionary, with key being the left-hand-side non-terminal rule and value being a triple $(split, B, C)$, where B is the first right-hand-side non-terminal rule and C is the second rule. We define that a tree cannot be parsed as the last cell is empty. If the input sentence cannot be parsed, the log probability should be zero. This function should implement the probabilistic CYK parsing algorithm.

[1] `https://en.wikipedia.org/wiki/Chomsky_normal_form`

3. `pretty_print` This function will trace back parsing path based on backpointer and return a parse tree. If there is no root non-terminal found, nothing will be returned.

The skeleton code will handle a few input and output operations for you, most importantly writing parse trees to a string. Your program will eventually be evaluated by running it on the hidden test grammar and sentences and verifying that the parse strings look identical. Your job is just to make sure the `parse` function generates a correct parse tree, which gets printed.

To get you started with these, we've provided a very simple grammar, its lexicon, a set of example inputs that both do and do not parse, and the expected output for each. We strongly encourage you to try creating your own grammars

■ **Problem 1.** (50%) Finish the `parser.py` code by implementing the `parse` function. At this point the program should be able to read in files for (1) a grammar (2) a lexicon, and (3) sentences to parse, specified as one per line and then print out the parse tree and probability for each input sentence.

3 Task 2: Competitive Grammar Writing

The second task will be a **group task** focused on writing a grammar that captures as much of English as possible. This is a group task and all submissions must be for groups of three; if you do not have a team, please post on the Discussion forum on Canvas. No single-person teams will be allowed without prior approval.

You have been supplied with materials for writing a probabilistic grammar that should parse as much of English as possible, assigning it a high probability, while rejecting ungrammatical English by assigning it a low probability or 0.

- `S1.gr` – This is the main grammar file with the `Start` symbol used to initiate parsing. You will likely modify this and the `Vocab.gr` file the most.
- `S2.gr` – This is a back-off grammar file. Since not all sentences can be parsed using S1, S2 provides a way of estimating the probability according to how likely are non-terminal sequences.[2]
- `Vocab.gr` – This file contains all the mappings from Non-terminals to terminals. Right now most terminals are generated by the `Misc` category, which allow you to generate anything.
- `allowed_words.txt` – This file specifies the whole vocabulary. You should not add any words to this. Note that you are required to be able to parse any sentence with these words.
- `example_sentences.txt` – Example sentences to help get you thinking about how to parse. As distributed, only the first two sentences can be parsed using S1.
- `pcfg_parse_gen.py` – A probabilistic CFG parser that you can use to test parsability of sentences with your grammars and to sample new sentences from S1.

[2]For example, if you defined Noun, Adjective, and Verb non-terminals, you could modify S2 to give different weights to a sequence of "Verb Noun Adjective" versus "Adjective Noun Verb"

3.1 How the competition works

Teams have the whole two weeks to develop their grammars, at which point they will submit all their files with the homework. Teams will then be evaluated on two aspects: (1) how grammatical are sentences sampled from S1? and (2) does the grammar assign high probability to grammatical sentences. To test the first aspect, we will sample 20 sentences from each team's grammar and three of your peers will rate them for grammaticality (if 2 of the 3 say yes, the sentence is grammatical). For the second aspect, we will collect the grammatical sentences and then compute the cross-entropy (log perplexity) of parsing the grammatical sentences using each team's grammar.

3.2 Strategies and Suggestions

The dual objectives of the competition mean that teams should generate valid English while also recognizing as much English as possible. As such, we have a few suggestions on how to get started, but many others are possible such as coordinating conjunctions, modal verbs, number words, adverbs and so on. How would you handle addressing singular/plural or different tenses in your grammar?

- The current grammar lumps most words into the `Misc` non-terminal. How would you update the current structure to create specific rules for different parts of speech? We conveniently left some suggestions on how you might *start* distinguishing these.
- The example sentences provide a great initial collection of sentences that the grammar *should* parse. A good first step is to see what you would need to change to get each to parse. We also recommend adding your own examples to this list! Take a look at the lexicon and see what kind of sentences you can generate.
- As you start making changes so you can parse more, it helps to also check that what you're generating is still grammatical. Be sure to keep sampling periodically. This sampling can also help identify places where you might want to tweak the grammar's weights.
- As you progress try considering modeling more advanced grammatical phenomena like questions, negation, transitive and intransitive verbs.
- In past versions of this, reasonably successful teams added 100-300 new rules across all of the grammar files.

4 Submission

Please upload the following to Canvas by the deadline:

1. Your completed `parser.py` file
2. All `.gr` files produced by your team
3. A pdf document listing the members of your team and (at a high level) describing how you approached building your grammar.

5 Post-Submission Tasks

After submission, we will be assigning each student 60 sentences sampled from competing teams to judge as grammatical or ungrammatical. You'll have one week to do this, though we anticipate you being able to get it done in just a few minutes. We will use your judgments to complete the grading.

6 Academic Honesty

Unless otherwise specified in an assignment all submitted work must be your own, original work. Any excerpts, statements, or phrases from the work of others must be clearly identified as a quotation, and a proper citation provided. Any violation of the University's policies on Academic and Professional Integrity may result in serious penalties, which might range from failing an assignment, to failing a course, to being expelled from the program. Violations of academic and professional integrity will be reported to Student Affairs. Consequences impacting assignment or course grades are determined by the faculty instructor; additional sanctions may be imposed.