Assignment 4 – Parsing with PCFGs

CS499 - Intro to NLP
Antonis Anastasopoulos

February 2021

Due Date: 4/2/2021 (eod)
Submission instructions: see at the end of the assignment.

Make sure to download the assignment materials from the class page.

Intro

In this assignment you will build and improve a simple parser trained from the ATIS portion of the Penn Treebank. ATIS (Air Traffic Information System) portion consists of short queries and commands spoken by users of a fake robot travel agent.

Setup Download the assignment materials from the class page. It contains the following files:

- `train.trees`: training data
- `dev.trees`: development data (trees)
- `dev.strings`: development data (strings)
- `test.trees`: test data (trees): don't peak!
- `test.strings`: test data (strings): don't peak!
- `preprocess.py`: preprocessor
- `unknown.py`: replaces one-count words with ⟨`unk`⟩
- `postprocess.py`: postprocessor
- `evalb.py`: compute label precision/recall

Try the following:

1. Run `train.trees` through `preprocess.py` and save the output to `train.trees.pre`. This script makes the trees strictly binary branching. When it binarizes, it inserts nodes with labels of the form `X*`, and when it removes unary nodes, it fuses labels so they look like `X_Y`.

2. Run `train.trees.pre` through `postprocess.py` and verify that the output is identical to the original `train.trees`. This script reverses all the modifications made by `preprocess.py`.

3. Run `train.trees.pre` through `unknown.py` and save the output to `train.trees.pre.unk`. This script replaces all words that occurred only once with the special symbol ⟨`unk`⟩.

You may write code in any language you choose. You are free to use any of the Python code provided when you program your own solutions to this assignment. (In particular, the module `tree.py` has useful code for handling trees.)

1 Training (3 credits total)

First, we will learn a probabilistic CFG from trees, and store it in the following format:

```
NP -> DT NN # 0.5
NP -> DT NNS # 0.5
DT -> the # 1.0
NN -> boy # 0.5
NN -> girl # 0.5
NNS -> boys # 0.5
NNS -> girls # 0.5
```

[**1 credit**] **(a)** Write code to read in trees and to count all the rules used in each tree. Run your code on `train.trees.pre.unk`. How many unique rules are there? What are the top five most frequent rules, and how many times did each occur?

[**2 credits**] **(b)** Write code to compute the conditional probability of each rule and print the grammar out in the above format. What are the top five highest-probability rules, and what are their probabilities?

To submit: Code that implements the above two items under a directory named "part1". Make sure to answer all questions in your PDF report.

2 Parsing (7 credits total)

[**3 credits**] **(a)** Now write a CKY parser that takes your grammar and a sentence as input, and outputs the highest-probability parse. If you can't find any parse, output a blank line. Don't forget to replace unknown words with ⟨`unk`⟩. Don't forget to use log-probabilities to avoid underflow.

[**1 credit**] **(b)** Run your parser on all sentences in `dev.strings` and save the output to `dev.parses`. In your report, show the output of your parser on the first five lines of `dev.strings`, along with their log-probabilities (base 10).

[**1 credit**] **(c)** Show a plot of parsing time (y axis) versus sentence length (x axis). Use a log-log scale. Estimate the value of k for which $y \approx cx^k$ (you can do a least-squares fit or just eyeball it). Is it close to 3, and why or why not?

[1 credit] (d) Run your parser output through `postprocess.py` and save the output to `dev.parses.post`. Evaluate your parser output against the correct trees in `dev.trees` using the command:

```
python evalb.py dev.parses.post dev.trees
```

In your report, show the output of this script, including your F1 score, which should be at least 88%.

[1 credit] (e) Run your parser on `test.strings` and save the output to `text.parses`. In your report, show the output of your parser on the first 3 lines of `test.strings`, along with their log-probabilities (base 10). Then, run the output through `postprocess.py` and save the output to `test.parses.post`. Include this file (with the correct name), as part of your submission. We will evaluate the output of your model on the test file. The best-performing models will receive one (1) bonus credit.

To submit: Code that implements the above items under a directory named "part2". Make sure to answer all questions in your PDF report, and include the `dev.parses`, `dev.parses.post`, `test.parses`, and `test.parses.post` output files.

Submission Instructions

This assignment must be done individually. Discussing the assignment on the class forum is absolutely ok – posting the solutions is not. If you use someone else's code (e.g. taken from a book or a website), make sure to properly cite it in your report.

Please submit all of the following in a gzipped tar archive (.tgz or .tar.gz; not .zip or .rar) via Blackboard. If you're making a full submission, please name your file `gmuid-hw4.tgz` (for example, the instructor's submission would be `antonis-hw4.tgz`). If you're making a partial submission, please name your file `gmuid-hw4-part.tgz` where part is the part (1, 2, 3, or 4) that you're submitting. **Make sure that your .tgz file expands into a self-contained folder with your gmuid in it, e.g. to `antonis-hw4` or something like that.**

Your submission should contain:

- A PDF file `(not .doc or .docx)` with your responses to the instructions/questions above. The report should include your name and gmuid, and it should be self-sufficient (as in, we shouldn't have to look elsewhere to grade you). If your name or an answer does not appear in the report, you will receive 0 credits for that part.
- All of the code that you wrote.

- A brief README file with instructions on how to build/run your code. We **will** run the code and check whether it produces all required outputs and if these outputs match the results in your report.