CSCI 544: Applied Natural Language Processing

Assignment 4: Parsing

Out: **19. September 2018**
Due: **10. October 2018**

Edit of 2018-09-19 03:16:01Z

In this assignment you will build a simple constituency parser trained from the ATIS portion of the Penn Treebank. Quite a bit of starter code and data is available on Vocareum. You are free to use any of the Python code in this archive when you program your own solutions to this assignment, however as long as you produce proper answers to the 5 written questions and a runnable shell script to Vocareum in Q6 you may use other languages or your own code. As usual, all work must be your own and may not be copied from or modified from others, including those you know or don't know.

Preparing the data

The file `train.trees` contains a sequence of trees, one per line, each in the following format:

```
(TOP (S (VP (VB Book) (NP (DT that) (NN flight)))) (PUNC .))
```

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. Run `train.post` 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`.

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>`.

Learning a grammar

Write code to 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
```

Run your code on `train.trees.pre.unk`.

Q1. (15 pts) How many rules are there in your grammar? What is the most frequent rule, and how many times did it occur?

Parsing sentences

Now write a parser that takes your grammar and a sentence as input, and outputs the highest-probability parse. Don't forget to replace unknown words with `<unk>`. Don't forget to use log-probabilities to avoid underflow (or use `bigfloat.py`). Run your parser on `dev.strings` and save the output to `dev.parses`.

Structuring your code

Please take a close look at the `runme` script provided. This script must take in *any* training trees file and test strings file and should create the specified test trees file. Once you have written preprocessing, grammar creation, and parsing code, you should modify `runme`, altering the names and invocation of the scripts as necessary to match what you have written. You should ***not***, however, replace the variables `$TRAINING`, `$INPUT`, or `$OUTPUT` with hard-coded paths.

You should invoke the command `./runme train.trees dev.strings dev.parses` to generate `dev.parses`. If you instead were to invoke, say, `./runme train.trees dev2.strings dev2.parses`, that would generate the file `dev2.parses` from `dev2.strings`. If you wanted to use less training data (maybe your parser is running slowly), you could create a file called `minitrain.trees` and then you should be able to invoke the command `./runme minitrain.trees dev.strings dev.minitrain.parses` to generate parses using less training data.

When we run your code for grading (see Q6 below) we will invoke `./runme train.trees test.strings test.parses` using our own files that you do not have access to. If you use hard-coded paths instead of following the instructions above, your code will not execute correctly and you will lose points on the assignment. Please ask questions on Piazza if this is not clear.

To make sure your code is doing the right thing, first make sure that `dev2.parses` does not exist, then run `./runme train.trees dev2.strings dev2.parses` and score this against `dev2.trees`. The set `dev2` is simply the first 20 sentences from `train.trees` so you should see a higher score than you do for `dev.parses`. If you don't get the answer you expect you probably have not designed the code as suggested.

Q2. (10 pts) Show the output of your parser on the first line of `dev.strings`, along with its log-probability (base 10).

Q3. (10 pts) 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 $y \approx x^k$ (you can do a least-squares fit or just guess). Is it close to 3, and why or why not?

Q4. (10 pts) 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
```

Show the output of this script, including your F1 score.

Improving your parser

Make at least three modifications to try to improve the accuracy of your parser. You can try the techniques described in class, or any other techniques you can think of or find in the literature. Remember that if you modify `preprocess.py`, you should also modify `postprocess.py` accordingly.

Q5. (45 pts) Describe your modifications and report your new F1 score on `dev.strings`. What helped, or what didn't? Why?

Q6. (10 pts) Modify the `runme` batch script in vocareum such that all the pre- and post-processing needed to train your grammar on an arbitrary training corpus and evaluate it on an arbitrary test corpus are done (the included batch script will run `preprocess.py` and `postprocess.py`, will generate an artificial

grammar, and will run a trivial parser; replace the dummy code with your own code!). When you submit your batch script will report a run that trains on `train.trees` and decodes `dev.strings`, evaluating against `dev.trees`, so you can verify proper behavior. After the submission window closes, your parser will be trained on a corpus similar to `train.trees` and then evaluated on a hidden `test.strings`. Your score on this question (maximum 10 points out of 100) will depend on your F1 score relative to the rest of the class.

What to turn in

- **On CrowdMark:** Your answers to questions Q1–Q5.
- **On Vocareum:** Code, including the `runme` batch script, in order to evaluate Q6 on a hidden test corpus.