Homework 5: Parsing

CSCI 562

due: 11am, 29 Nov 2012

In this assignment you will build a simple parser trained from the ATIS portion of the Penn Treebank. Before you begin, please download `hw5.tgz` from Blackboard. You are free to use any of the Python code in this archive when you program your own solutions to this assignment. (In particular, the module `tree.py` has useful code for handling trees.)

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 .))
```

Q1. What is the maximum branching factor (maximum number of children of any node) of the trees in this file?

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`.

Q2. How many word types (distinct words) appear in `dev.strings` that do not appear in `train.trees`?

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`.

Q3. How many rules are there in your grammar?

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`.

Q4. 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).

Q5. Run your parser output through `postprocess.py` and save the output to `dev.parses.post`. Calculate your labeled precision/recall 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 some 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.

Q6. Describe your modifications and report your new F1 score on `dev.strings`. What helped, or what didn't? Why?

Q7. Finally, run your best parser on `test.strings`. What F1 score did you get? Your credit on this question (maximum 10 points out of 120) will depend on your F1 score.

What to turn in

- **On paper:** Your answers to questions Q1–Q7.
- **On Blackboard:** Any code that you wrote for this assignment, and your final parser outputs on `dev.strings` and `test.strings`.