Language Technology, Spring 2013 HW3: Parsing

due Tuesday May 7 at midnight on blackboard

In this assignment you will

1. train a probabilistic context-free grammar (PCFG) from a treebank (need binarization);
2. build a simple CKY parser (which handles unary rules), and report its parsing accuracy on test data.

You will need to download from the `hw3/hw3-data.tgz`:

corpora:

`train.trees`	training data, one tree per line
`test.txt`	test data, one sentence per line
`test.trees`	gold-standard trees for the test data (for evaluation)

scripts:

`tree.py`	`Tree` class and reading tool
`evalb.py`	evaluation script

You need to submit:

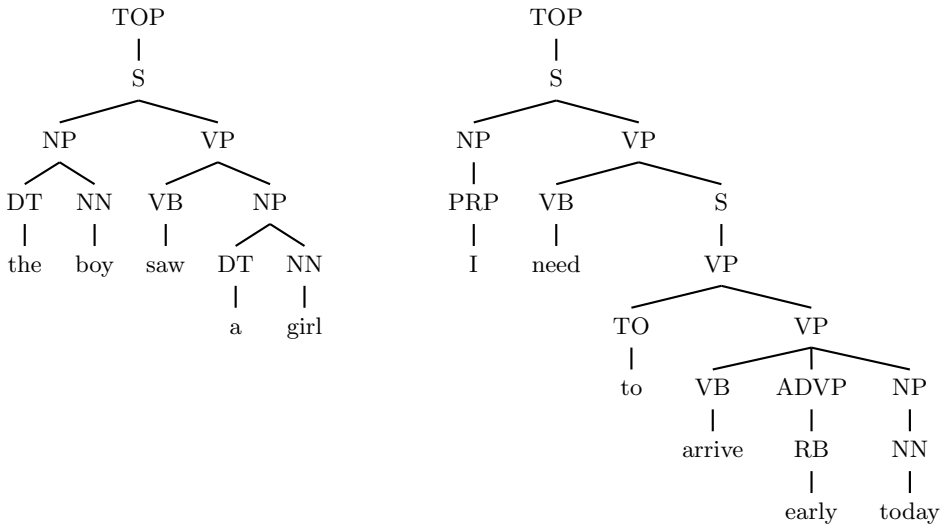
1. a PDF report (include the answers, e.g., accuracies or statistics, to each question);
2. all your python programs;
3. result files mentioned in each question.

1 Learning a grammar (40 pts)

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

```
(TOP (S (NP (DT the) (NN boy)) (VP (VB saw) (NP (DT a) (NN girl)))))
(TOP (S (NP (PRP I)) (VP (VB need) (S (VP (TO to) (VP (VB arrive) (ADVP (RB early)) (NP (NN today))))))))
```

N.B. `tree.py` provides tools for you to read these trees (building a `Tree` object from a string).



Note that the second example is particularly interesting because it covers

- unary rules on preterminals like `NP -> PRP` and `NP -> NN`;
- unary rules on nonterminals like `S -> VP` and `TOP -> S`;
- ternary rules like `VP -> VB ADVP NP` (so that you need binarization).

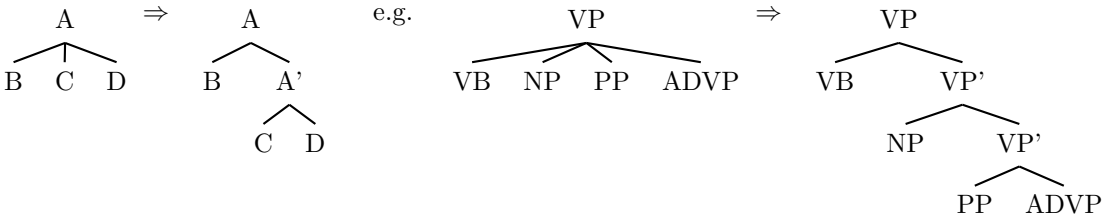
Your job is to learn a probabilistic context-free grammar (PCFG) from these trees.

Q1.a Preprocessing: replace all one-count words (those occurring only once, *case-sensitive*) with `<unk>`:

```
cat train.trees | python replace_onecounts.py > train.trees.unk 2> train.dict
```

where `train.trees.unk` is the resulting trees with `<unk>` symbols, and `train.dict` is a file of words which appear more than once (one word per line). **Explain why we do this.**

Q1.b Binarization: we binarize a non-branching tree (recursively) in the following way:




```
cat train.trees.unk | python binarize.py > train.trees.unk.bin
```

What are the properties of this binarization scheme? Why we would use such a scheme? (What are the alternatives, and why we don't use them?)

Q1.c Learn a PCFG from trees:

```
cat train.trees.unk.bin | python learn_pcfg.py > grammar.pcfg.bin
```

The output `grammar.pcfg.bin` should have the following format:

```
TOP
TOP -> S # 1.0000
S -> NP VP # 0.8123
S -> VP # 0.1404
VP -> VB NP # 0.3769
VB -> saw # 0.2517
...
```

where the first line (`TOP`) denotes the start symbol of the grammar, and the number after `#` in each rule is its probability, with four digits accuracy.

We group rules into three categories: binary rules (`A -> B C`), unary rules (`A -> B`), and lexical rules (`A -> w`). How many rules are there in each group?

2 CKY Parser (60 pts)

Now write a CKY parser that takes a PCFG and a test file as input, and outputs, for each sentence in the test file, the best parse tree in the grammar. For example, if the input file is

```
the boy saw a girl
I need to arrive early today
```

the output (according to some PCFG) could be:

```
(TOP (S (NP (DT the) (NN boy)) (VP (VB saw) (NP (DT a) (NN girl)))))
(TOP (S (NP (PRP I)) (VP (VB need) (S (VP (TO to) (VP (VB arrive) (ADVP (RB early)) (NP (NN today))))))))
```

Q2.a Design a toy grammar `toy.pcfg` and its binarized version `toy.pcfg.bin` such that the above two trees are indeed the best parses for the two input sentences, respectively. **How many strings do these two grammars generate?**

Q2.b Write a CKY parser. Your code should have the following input-output protocol:

```
cat toy.txt | python cky.py toy.pcfg.bin > toy.parsed
```

Verify that you get the desired output in `toy.parsed`. **Note that your output trees should be debinarized (see examples above).**

Q2.c Now that you passed the toy testcase, apply your CKY parser to the real test set:

```
cat test.txt | python cky.py grammar.pcfg.bin  > test.parsed
```

Your program should handle (any levels of) **unary rules** correctly, even if there are unary cycles.

How many sentences failed to parse? Your CKY code should simply output a line

```
NONE
```

for these cases (so that the number of lines in `test.parsed` should be equal to that of `test.txt`). What are main reasons of parsing failures?

Q2.d Now modify your parser so that it first replaces words that appear once or less in the training data:

```
cat test.txt | python cky.py grammar.pcfg.bin train.dict > test.parsed.new
```

Note that:

- `train.dict` should be treated as an **optional** argument: your CKY code should work in either case! (and please submit only **one version** of `cky.py`).
- your output tree, however, should use the original words rather than the `<unk>` symbol. Like binarization, it should be treated as internal representation only.

Now how many sentences fail to parse?

Q2.e Evaluate your parsing accuracy by

```
python evalb.py test.trees test.parsed
python evalb.py test.trees test.parsed.new
```

Report the labeled precisions, recalls and F-1 scores of the two parsing results. Do their differences make sense?

Debriefing

Please answer these questions in **debrief.txt** and submit it along with your work. **Note:** You get 5 points off for not responding to this part.

1. How many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Are the lectures too fast, too slow, or just in the right pace?
4. Any other comments (to the course or to the instructors)?