

Assignment 5: Grammars, Parsing, Forest, and Generation

CS 562, Fall 2009

due: **electronic version only** at oana@isi.edu, **Tuesday Nov 17, 11am**

In this assignment you will build a simple parser trained from the ATIS portion of the Penn Treebank. You will need:

- Tiburon version 0.5.1 <http://www.isi.edu/licensed-sw/tiburon/>
- <http://www.isi.edu/natural-language/teaching/cs562/2009/hw5/hw5-data.tgz>, which contains:
 - data:*
 - `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:*
 - `evalb.py` evaluation script
 - `tree.py` used by `evalb.py`
 - `cfg2rtg` converting context-free grammar (CFG) to regular tree grammar (RTG) (for Tiburon)

The Python scripts require Python 2.5 or later.

1 Learning a grammar (25 pts)

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(.))
```

The trees have been *binarized* for you so that the context-free grammar is in Chomsky Normal Form (CNF): every node has either two nonterminal children or one terminal child. The new nodes (i.e., “virtual non-terminals”) that are inserted have labels like NP’; also, nodes with only one nonterminal child (i.e., unary productions like $S \rightarrow VP$) had to be fused with their child, creating node labels like `S_VP`. Finally, words occurring only once have been replaced with the token `<unk>`. You do *not* need to treat any of these labels specially.

Write code to learn a probabilistic context-free grammar (PCFG) from these trees, and output it in the following format:

```
S
S -> NP VP # 1
NP -> DT NN # 0.6
NP -> DT NNS # 0.4
DT -> the # 0.7
DT -> a # 0.3
NN -> boy # 0.5
NN -> girl # 0.5
NNS -> boys # 1
VP -> VB NP # 1
VB -> saw # 1
```

where the first line (S) is the start nonterminal of the grammar, similar to the final-state in carmel. Name it `mygrammar.pcfg`.

Q1.a Your code to learn the grammar should have the following input-output protocol:

```
cat train.trees | your-code > mygrammar.pcfg
```

Include both your code and the resulting grammar in your submission.

Q1.b How many rules are there in your grammar? How many rules are there to rewrite each nonterminal?

Q1.c Use Tiburon to generate the top 10 trees from your grammar:

```
cat mygrammar.pcfg | ./cfg2rtg | tiburon -k10 -
```

The `cfg2rtg` script converts your PCFG into a regular tree grammar (to be covered in class) that Tiburon understands. You don't need to worry about this part. Show the output.

Q1.d Use Tiburon to generate 10 random sentences from your grammar:

```
cat mygrammar.pcfg | ./cfg2rtg | tiburon -yg10 -
```

Show the output.

2 CKY Parsing (35 pts)

Now write a simple 1-best CKY parser that takes your grammar and a sentence as input, and outputs the best tree in the grammar for this sentence. The output trees should have the same format as those in the training file, along with its probability. For example, if the grammar is the above example PCFG, and the input sentence is `the boy saw a girl`, the output should be:

```
S(NP(DT(the) NN(boy)) VP(VB(saw) NP(DT(a) NN(girl)))) # 0.0189
```

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

```
cat test.txt | your-cky mygrammar.pcfg > test.parsed
```

Include your CKY code and the resulting `test.parsed` file in your submission.

Q2.b How many sentences failed to parse? Your CKY code should output a 0 for these cases (so that the number of lines in `test.parsed` should be equal to that of `test.txt`).

Q2.c How accurate is your parsing result compared to the gold-standard? Calculate your labeled precision/recall against the correct trees in `test.trees`, using the command:

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

This script tells you how many brackets are in the two files, and how many brackets matched between the two files. Show the output of this script, and calculate the precision, recall, and F1 score (it should be around 84%).

3 Forest and Generation (40 pts)

We assume this is your first time to write a CKY parser. It's quite an accomplishment, isn't it! But how could you verify the correctness of your implementation?

Well, apart from evaluation, the simplest way is to use Tiburon again. We have already seen in Questions Q1.c and Q1.d that Tiburon can be used to *generate* from the grammar (using the Knuth 77 algorithm taught in class). So we can dump the *intersected* PCFG (i.e., the **forest**) and feed it into Tiburon, which will generate 1-best and k-best trees from the grammar for us. You just need to compare your parsing result with Tiburon's output. Modify your CKY program to output a forest for each input sentence. The forest itself is another PCFG, like the following for the example sentence with example grammar above:

```
S,0,5
S,0,5 -> NP,0,2 VP,2,5 # 1
NP,0,2 -> DT,0,1 NN,1,2 # 0.6
NP,3,5 -> DT,3,4 NN,4,5 # 0.6
DT,0,1 -> the # 0.7
DT,3,4 -> a # 0.3
NN,1,2 -> boy # 0.5
NN,4,5 -> girl # 0.5
VP,2,5 -> VB,2,3 NP,3,5 # 1
VB,2,3 -> saw # 1
```

Now you can use Tiburon to do 1-best parsing for you:

```
cat forest | ./cfg2rtg | tiburon -k1 - | perl -ple 's/,\d+,\d+//g'
```

which should give you the same result as your CKY parsing. The perl one-liner at the end removes the indices (i.e., converting S,0,5 back into S) so that the result is a normal tree in the old CFG which is compatible with CKY's output.

Q3.a Your code for forest generation should have a similar input-output protocol to the CKY case:

```
cat test.trees | your-forest-generator mygrammar.pcfg > test.forests
```

These forests should be stored in one file, with a **blank line** separating consecutive forests. Include your forest generator code and the forest for the first sentence in the test data (name it **1.forest**) in your submission (N.B.: do **not** submit all forests; that will be too large).

Q3.b Use Tiburon to generate a 10-best list of trees from **1.forest** (with option **-k10**). Show the output. Verify that the 1-best in this list is exactly the same as your CKY output.

Q3.c Show a plot of number of hyperedges (i.e., intersected rules) in the forest (y axis) versus sentence length (x axis) for all sentences in **test.txt**. It may help to use a log-log scale, in which $y = x^k$ appears as a line with slope k . What is k ? (You can do curve-fitting or just eyeball it.) Then include on the same plot another curve: the number of nodes (i.e., intersected nonterminals) in the forest (y axis) versus sentence length, and similarly figure out the slope. Do these slopes make sense with what you have learned in class? Explain.

Q3.d How many parses are there in **1.forest**? You can use Tiburon (with option **-c**) for that. Then show a plot of number of parses per forest versus sentence length for all sentences in **test.txt**, and figure out their (rough) relationship. How did you do it, and does your result make sense? Explain.