CS 294-5: Statistical Natural Language Processing

Assignment 0
A Classical Chart Parser

This assignment is designed to get you up and running with the eecs servers and the code this course will be using. It's also designed to let you play a little with a simple chart parser. There's nothing to hand in, but I'm expecting that you'll complete it, and in the process, hammer out any new account, login, and start up problems before the first graded assignment is handed out next Friday. I also strongly recommend you work through these exercises and get your feet wet now, for their instructional content. If you have general questions about an assignment, especially access issues, bugs, and so on, please post to the class newsgroup: ucb.class.cs294-5.

Getting an account to the instructional servers. If you do not have an eecs instructional account, you'll need to get one for this course. There's new account information at

http://www-inst.eecs.berkeley.edu/~iesg/new-users.html

Machines and labs available for class use are outlined at

http://www-inst.eecs.berkeley.edu/~iesg/iesglabs.html

For the majority of assignments, including this one, you can use any machines you like. In the event that an assignment requires unusual computing resources (usually lots of RAM), we'll have a few machines reserved for our class's use, and I'll announce the details.

Getting the course code to run. I'm going to give directions for running code from a unix shell, and I'm going to assume you know how to make directories, edit files, change your path, etc. under unix. If you want to run code locally, for example on a windows machine, or in an IDE, it's up to you to get that to work, including installing Java 1.5. So, pick your favorite server from the machine list and log in via ssh (info on how to do this can be found at http://inst.eecs.berkeley.edu/connecting.html). Make a directory for class code, let's say it's

~/cs294-5/java/

and change to that directory. The class code I'm providing lives in

/home/ff/cs294-5/java/src/

and a compiled set of class files lives in

/home/ff/cs294-5/java/classes

You can run a simple test file by typing

/home/ff/cs294-5/java/bin/java –cp /home/ff/cs294-5/java/classes edu.berkeley.nlp.Test

You should get a confirmation message back. I recommend adding "/home/ff/cs294-5/java/bin/" to your path and /home/ff/cs294-5/java/classes to your classpath ; I'll assume it's in your path from here out. If so, the following should run the test class:

java edu.berkeley.nlp.Test

If you get a Java version error, your path probably has another java executable which is pre-1.5 listed before /home/ff/cs294-5/java/bin.

Now the real test. Invoke the chart parser with:

java edu.berkeley.nlp.classical.ChartParser

If also goes well, you'll get a usage message. It wants a grammar and lexicon file, as well as a sentence to parse. Copy into your local directory the files

/home/ff/cs294-5/java/other/slides.lexicon
/home/ff/cs294-5/java/other/slides.grammar

These files have the miniature grammar we used to parse the sentence "new art critics write reviews with computers." Try running

java edu.berkeley.nlp.classical.ChartParser slides.lexicon slides/grammar "new art critics write reviews with computers"

There should be a few parses printed out in Penn Treebank format. Try to sort out which parse corresponds to which interpretation. If you want to see the operations of the parser (edge pops, etc.) run it with "–verbose" as the first argument. You're more than welcome to poke around in the code, of course (you'll get a chance to write parts of a parser in a later assignment).

Modifying the grammar and lexicon. This sample grammar doesn't parse much. Try to write a grammar that parses the following sentences in a sensible fashion. There's a guide on basic syntax in M+S, Ch. 3 (or J+M Ch. 9). You can use both n-ary rules and empty rewrites (if you want). The special symbol for empty is *e*, so if you want a rule that says NPs can be empty, you can add NP $\rightarrow$ *e*. Be careful with empties; they can easily cause sentences to have infinite parses.

The man in the moon is a crater pattern that can be seen from Earth.
New York lawyers often charge by the hour.
The kitten that I bought from the store turned out to be a little crazy.
Young cats and dogs eat more food than older ones with slower metabolisms.

Count the ambiguities as your grammar grows – how bad is it? Which rules are contributing to the ambiguity? Try to keep the grammar as general as possible, while controlling the level of ambiguity.

Note: Unless you've got a lot of linguistics classes under your belt, you may not feel confident that the parses you have in mind for the sentences above are right. There's no single right answer (and none of the future assignments will involve deciding annotation standards). Also, it's important to remember that even linguists argue about how best to analyze various constructions – it took months of debate for the Penn Treebank annotation practices to be established (and they made some weird choices, see below). What's important in practice is to be consistent, so the tree configurations you assign reliably correspond to readings of the sentence. For example, the Penn Treebank, which has an important place in modern NLP, analyzes

the cat in the sink

as

```
(NP  (NP (DT the) (NN cat))
     (PP (IN in)
         (NP (DT the) (NN sink))))
```

contrary to what most linguists would choose. Draw this bracketed structure out as a tree if it helps to read it. Note that the PP is a sibling to "the cat", not "cat". If we do semantic interpretation on this structure the way we suggested in class, it won't work out to the "correct" meaning of

$$\textit{unique-member}(\lambda x.\text{cat}(x) \wedge \text{in}(x, \textit{unique-member}(\lambda y.\text{sink}(y))))$$

That is, the unique entity that is both a cat and in the sink. Why not? What kind of structure, lexical meanings, and rule translations would we need to get that interpretation?