CS457 - Parsing Lab

For this lab, we're going to be analyzing two publicly available, state-of-the-art parsers along a number of metrics. Below, I have given you three areas ways in which you can analyze the parsers:

- Speed - How long does it take to parse a sentence?
- Robustness - How well does it handle long and/or irregular output?
- Parsing performance - How different are the parses between the two systems?

You should investigate as many of these topics as you can, but feel free to spend as much time as you want diving into one. Don't feel obliged to investigate all of them. I'd rather a more detailed analysis on one than a quick pass over all three.

During the last 15 minutes of class, you will have ~2 minutes to "present" your results and we will discuss all the results as a class.

The Parsers

I have downloaded two parses into:

`/home/dkauchak/PUBLIC/cs457/parsing_lab`

- Stanford's parser: http://nlp.stanford.edu/software/lex-parser.shtml
- Berkeley's parser: http://code.google.com/p/berkeleyparser/

Both are Java-based and you can either run them from within these directories or copy them into your own home directories.

To run them:

Stanford

```
java -Xmx1g -cp stanford-parser.jar edu.stanford.nlp.parser.lexparser.LexicalizedParser -outputFormat
"oneline" grammar/englishFactored.ser.gz <input_file>
```

(all on one line)

Berkeley

```
java -Xmx1g -jar berkeleyParser.jar -gr eng_sm6.gr
```

The input is read from STDIN and output sent to STDOUT, so if you want to run data from a file use '<' to pipe it in and '>' to pipe the output out.

Data

For this assignment I've provided you with 100K sentences from three sources that we've previously analyzed:

- Twitter posts
- Simple English Wikipedia articles
- English Wikipedia articles

They can all be found in the data directory for the lab.

Experiments

Pick one or more of these experimental areas and compare the performance of the two parsers.

Speed

Compare the speed of the two parsers. The `date` command may be useful here or you can also time things within your favorite programming language.

Some things to think about:

- Try and separate the load time vs. the parse time. One easy way to do this is to just run a single short sentence through the parser and time how long that takes.
- Try to evaluate using a metric like parses/second that would allow you to extrapolate to other data sets.
- Anytime you do timing experiments, it's a good idea to do multiple repetitions of the same experiment and aggregate the data, since timing can be affected by a lot of different things.
- How does the performance vary with the length of the sentence?
- How does the performance vary with the source of data?
- How do the two compare?

Robustness

Sometimes parsers don't always work on all sentences. Sometimes they fail gracefully (like outputting an empty parse) and sometimes less gracefully (like throwing an exception). Investigate the robustness of the two parsers. Some things to think about:

- Parsers can have problems with a variety of input. Try some of the more common problem cases:
  - Long sentences
  - Sentences with less traditional vocabulary
  - Sentences with less traditional punctuation (e.g. trying to parse two sentences)
- Run a number of sentences through from the different data sets. Do they parse all of the sentences? Any errors or unparsed sentences?
- Parsing is computationally expensive, so you may have to pick and choose interesting examples from the data rather than trying to run all of it

Parsing Performance

Parsing is most frequently evaluated by comparing the output of the system to a "correct" human translation. Unfortunately, there is not much free, publicly available parse data in English. However, instead of comparing against a human gold standard, we can still compare the two systems with respect to eachother to understand how much variety there is in the system's output and where they differ.

In the `evaluator` directory, I've provided you with a script that calculates precision, recall and F1:

```
java -cp /home/dkauchak/PUBLIC/cs457/parsing_lab/stanford-parser-2011-09-14/stanford-parser.jar:.
nlp.parsing.lab.Evaluator
```

(all one line)
To run it from any directory, change the "." to "/home/dkauchak/PUBLIC/cs457/parsing_lab/evaluator"

If you run it, you'll see that it takes 3 parameters:

- <`eval_method`>: one of `-basic` or `-categories`. The categories flag gives a breakdown for the different constituents and is more detailed.
- <`system_trees`>: the system trees, one per line
- <`system_trees`>: the corresponding "correct" trees, one per line

Note that in our case you'll just be picking one of the system outputs for correctness.

A few things to think about:

- The two parsers don't output in exactly the same format. The Berkeley parser always starts the tree with a `ROOT` node at the top, whereas the Stanford parser always surrounds the tree with an extra set of parenthesis. You should be able to coerce one into the other format fairly easily (for example, with `sed`).
- How do the results differ as the length differs?
- How do the results differ over different data sets?
- Are there particular constituents that seem to differ more?
- Are there other systematic differences?