CS159 - 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 ~1-2 minutes to "present" your results and we will discuss all the results as a class.

All of the resources for this lab can be found at:

```
/common/cs/cs159/assignments/parsing_lab/
```

You may either copy the contents of this directory over or just create a symlink using `ln -s`.

The Parsers

I downloaded two parses to play with:

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

Both are Java-based. I have created wrapper scripts for both of them that each take two files as parameter, the first an input text file to be parsed and the second an output filename.

- `stanford/parse_stanford.sh`
- `berkeley/parse_berkeley.sh`

For example, to run the Stanford parser with input file "`input.txt`" and output to the file "`input.parsed.txt`", I would type:

```
stanford/parse_stanford.sh intput.txt input.parsed.txt
```

Data

In the `data` director I've provided you with two types of data. First, I've provided you with some raw text from sources we've previously analyzed. Specifically, 100K sentences from:

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

Second, I've provided you with a portion of the PennTreebank corpus, both in raw text form and with the human parses.

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 `time` 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. I have, however, obtained a preprocessed a small portion of the PennTreebank corpus[1] 3914 sentences.

To evaluated how well the parses are doing you can have them parse a portion of the raw text (the `penntreebank.partial.txt` file) and then compare the results to the corresponding portion in the human parsed file (the `penntreebank.partial.parsed` file).

Besides the parser, the Stanford parser package also comes with a number of useful tools, including some evaluation scripts. The one we'll use takes two parameters:

[1] From: `http://www.nltk.org/nltk_data/`

- <system_trees>: the system trees, one per line
- <gold_trees>: the corresponding "correct" trees, one per line

I've created a wrapper around this and it can be run as:

```
stanford/stanford-eval.sh <gold_trees> <system_trees>
```

It will output some statistics about each of the sets of trees and then the last line will be of the form:

```
Evalb LP/LR summary evalb: LP: XX LR: XX F1: XX Exact: XX N: XX
```

where

LP	is precision
LR	is recall
F1	is F1
Exact	is the percentage of trees that matched exactly between the system and the gold
N	the number of sentences

A few things to think about for parsing quality:

- 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?
- You might also try comparing one system's output to the other, i.e. choose one of the system's as the "gold" and see how much they differ and how.