Computational Linguistics

Assignment 4 (2019-12-06)

Winter Semester 2019/20 – Prof. Dr. Alexander Koller

PCFG Parser Evaluation

In this assignment, you get to evaluate and compare two state-of-the-art syntactic parsers on real-life English text from various domains. For this, you will need the following:

- the parsers,
- the test sets, and
- the evaluation scripts.

Parsers

Our evaluation campaign focuses on PCFG parsing using two *de facto* standard systems: the Berkeley parser and the Stanford parser. You can get them through the official repositories:

- `https://github.com/slavpetrov/berkeleyparser`
- `http://nlp.stanford.edu/software/lex-parser.shtml`

Both parsers are written in Java and distributed as archives bundled with pre-trained models. Use the latest versions if possible.

The Berkeley parser is simple and easy to use. It is provided together with a PCFG grammar for English trained on the Penn Treebank training set. Run the parser with this grammar as input. To try it out, you can feed it a sentence from STDIN:

```
$ java -jar BerkeleyParser.jar -gr <grammar_file>
Test me !
( (SQ (VP (VB Test) (NP (PRP me))) (. !)) )
```

Run the parser without parameters to see all the options. It reads multiple file formats, including CoNLL and one-sentence-per-line, and can also perform preprocessing if needed.

The Stanford parser is a mature system with many options and features, including a graphical user interface for exploring the outputs. There is an extensive manual including a FAQ available through the official page, and the parser is highly customizable. It also comes with predefined shell scripts illustrating its usage. Give them a try:

```
$ ./lexparser.sh data/testsent.txt
(ROOT (S (VP (VB Test) (NP (PRP me))) (. !)))
```

Through the GUI, you can loading a text, select a parsing model, parse and visualize the output (see `lexparser-gui.sh`).

However, for performing the actual evaluation, you'll need to explore the parameters of the parser. For example, our test files are already tokenized, and you will want to indicate this to the parser to prevent that it tries to tokenize the test sentences again.

Also, the Stanford parser by default adds a `ROOT` node on top of every parse tree it outputs, and the trees it prints may not be formatted in the way that EVALB expects. Address these issues, either by passing the appropriate options to the Stanford parser or by writing a post-processing script. Feel free to use NLTK in your script.

As the parser comes bundled with several models for English, feel free to give them all a try, provided that they output a phrase structure tree. Note that only some of them are trained on the Penn Treebank training set (WSJ), and that only these are directly comparable to the Berkeley parser or the gold test trees.

Test sets

We provide three test sets for you, and you can download them from here: `http://www.coli.uni-saarland.de/~koller/materials/anlp/a4_test_sets.zip`. The test sets come from different domains, and your evaluation effort thus includes cross-domain testing. Feel free to speculate about where the test sentences come from.

The three test sets come in pairs of files: `{1,2,3}.gold` and `{1,2,3}.plain` contain the gold-standard annotations and the plain text sentences for the three test sets, respectively. Run the parsers on the plain text and evaluate against the gold-standard annotations.

We use the one-sentence-per-line format in all files.

Evaluation

Evaluate the parsers on the test sets for labeled and unlabeled precision, recall, and F_1 scores. You can implement your own evaluators, but we strongly recommend that you use the standard scoring tool for constituency parsing, EVALB (`http://nlp.cs.nyu.edu/evalb/`). Get to know EVALB, especially its parameter file syntax; take a look at the examples they provided (`COLLINS.prm` and `new.prm`).

A quick technical sidenote: EVALB was written in C, over twenty years ago, and you will have to compile it yourself on your own system, which requires a C compiler. You may have to remove the line `#include <malloc.h>` from `evalb.c` to make it work with a modern compiler.

Implement at least one statistical significance test to measure the significance of differences between the outputs of two parsers. Two widely used significance tests are approximate randomization and bootstrap testing; see the links on the course website.

Your submissions

Test at least two parser models: one from the Berkeley parser, and the other from the Stanford parser. For these two, provide the parser outputs for each parser and for each of the test sets separately. In a `README` file, document the scores and write your observations. You should document the scores – unlabeled and labeled precision, recall, and F_1 – for each of the test sets, and the overall results, i.e., the scores for all test sets combined into a single test set. Indicate the statistical significance of differences between the two parsers for each of the test sets at $p < 0.05$ and $p < 0.01$.

Extra credit: Test additional Stanford parser models and include the results in your report. Implement more than one statistical significance test and compare the results of significance testing across your tests. Provide more evaluation details, such as per-label accuracy, or sentence length vs. accuracy curves. Or anything else that is interesting and that you came up with yourself!

Turn in before class on 2019-12-20 on Piazza.