BM1 Advanced Natural Language Processing
University of Potsdam, Winter 2016/17

Assignment 4: Evaluating Statistical Parsers

Tatjana Scheffler, Ph.D.

Due: Friday, December 16, 10:00 a.m.

Objective In this assignment we are exploring the evaluation of statistical phrase structure parsers.

Data We will use the development section of the Penn Treebank. We provide gold parses (`.gold`), automatic parses (`.berkeley`; see below) as well as the pre-tokenized raw data (`.text`). Each file contains one section, as one sentence per line. The text files are pre-tokenized using white space.

Problem 1: Reimplement EVALB

Standardly, phrase structure parsers are evaluated with the Parseval metric EVALB, which computes labelled and unlabelled precision, recall, and F_1-scores over the brackets in a phrase structure parse. You can find the standard tool at (`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, almost 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.

Then reimplement labelled and unlabelled precision, recall and F_1 in Python. Consult the lecture slides for the definitions. In order to represent the trees, you may use NLTK's Tree class, but note that it doesn't keep track of the leaves' *string indices*, so it may not be easy to distinguish where you are in a sentence like "The man with the hat hit the man with the shovel".

Alternatively, you can implement your own recursive tree data structures and make sure to keep track of the *string indices* of the leaf nodes directly. Your data structure should be able to read in bracketed trees.

Notes on EVALB : Some pre- or postprocessing may be necessary on the trees to compare them. Note in particular, that some trees contain an additional ROOT or TOP node which should not be part of the evaluation.

Please make sure that your evaluation script only compares the first part of the label (e.g., NP for NP-SBJ, etc.). Any remainder should be stripped before evaluating (they have been stripped from the gold data). By convention, POS tags are not compared (not even for labelled prec and recall).

The gold parse trees contain empty categories (traces), which have the label (POS tag) "-NONE-". You should remove all such traces and any nodes that contain nothing but traces before evaluating. For example,
`... (VP (VBD reported) (SBAR (-NONE- 0) (S (-NONE- *T*-1)))) (. .)))`
would be processed to give `... (VP (VBD reported)) (. .)))`

Run your evalb implementation on the provided test set. It was produced using the Berkeley parser: `https://github.com/slavpetrov/berkeleyparser`
Note any observations.

Problem 2: Error analysis

We're trying to evaluate an existing parser in more detail to see where its problems lie. We're going to try the Stanford parser, a mature system with many options and features, including a graphical user interface for exploring the outputs: `http://nlp.stanford.edu/software/lex-parser.html`.

There is an extensive manual including a FAQ available through the official page, and the parser is highly customizable. Fortunately, 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 might want to indicate this for the parser. Also, the parser output might slightly differ from the expected output, e.g., in having the ROOT label before S, in tree formatting, etc. Investigate if the issues are addressable through the options, or should some of them be addressed through post-processing.

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. Note that only some of them are trained on the Penn Treebank training set (WSJ).

Perform an evaluation of the parser on the provided dataset and compute EVALB scores with your evaluation script.

In order to learn more about the parses, also report plots for accuracy-by-constituent-size and accuracy-by-label for both the Berkeley parser output and a Stanford parser model.

Accuracy by constituent size It is easier to predict a correct labelled bracketing for a smaller consituent (e.g., two words) than for a bigger one. Compute accuracy scores by size of constituent and plot them.

Accuracy by label Which kinds of constituents are hard to predict? For each label type (S, NP, VP, PP, etc.), report the number of such constituents and parsing accuracy.

Discuss. Compare the results of the Berkeley and the Stanford parser. What did you learn by playing with the Stanford parser's models and from the detailed accuracy plots? Which other tests may help you understand the strong and weak points of the different parsers?

Problem 3: Dependency Grammar

Pick 3 test sentences and provide reasonable dependency structures for them.

Briefly speculate: What are the advantages/disadvantages of dependency grammars over phrase structure grammars? What kinds of syntactic phenomena can be captured more or less easily?

Submission Submit your EVALB script, parsing outputs, and a pdf write-up documenting evaluation scores and all observations.

Extra credit. Ideas for further work: testing additional parsers, testing the parsers on out-of-domain data.

Submit your solutions and code via email to `johannsmeier@uni-potsdam.de`