NLMI 2016: Project Exercises
Part B

The goal of this project is to consider how grammar transformations affect the accuracy of a treebank PCFG parser. For this, you will be provided with an implementation of the CKY algorithm (a java implementation, described below). The project has two steps: in the first one you will deal with basic binarization of a grammar, in the second one you will implement horizontal and vertical Markovization plus (optionally) consider a new transformation of your choice.

Parser and Data

You are provided with a basic PCFG parser which supports only binarized treebanks. It implements a fairly generic form of the CKY algorithm (incl. unary rules).

Training The PCFG model is estimated on the binarized treebank provided as input on the command-line. The PCFG model performs smoothing for generation of words given POS tags, but no smoothing of inner rules.

Testing Assuming that you follow the conventions described below, the parser will perform de-binarization (and grammar annotation removal) automatically during testing.

Evaluation The parser reports evaluation results (bracketing precision, recall and F1 + exact match on sentences).

Usage The command line looks like:

```
java -ea -Xmx1G -cp path-to-bin nlp.assignments.parsing.SimpleParser
-train binarized-train-file -test not-binarized-test-file
```

where

- `path-to-bin` is the path to parser binaries (e.g., `bin/`),
- `binarized-train-file` is the binarized treebank file you will be producing in the assignment,
- `not-binarized-test-file` is one of the test sets (IMPORTANT: it should not be binarized).

The package includes example binarized files, see README for details.

STEP 1: Binarization

Due Thurs 10th March 2016

Perform a transformation **binarization** as described in class (Lecture 7).

A fragment in the original treebank (i.e. a sub-tree) looks like:

```
(NP (NNP Rolls-Royce) (NNP Motor) (NNPS Cars) (NNP Inc.))
```

After binarization, it should look like

```
(NP (NNP Rolls-Royce) (@NP->_NNP (NNP Motor) (@NP->_NNP_NNP (NNPS
Cars) (@NP->_NNP_NNP_NNPS (NNP Inc.)))))
```

Perform the above transformation on the training data `train20.text`, and save the binarized trees in a file `train20.text.binarized` . (Your code should take as input non-binarized files and save the results of the binarization procedure as output in a separate file.)
Command line format:

```
./b-step1 -input [non-binarized] -output [binarized]
```

Use `train20.text.binarized` for training the parser, and one of the evaluation sets for evaluation. Specifically, the validation (development) set should be used during the debugging stage `validate20.txt`, and final testing should be performed on the test set (`test20.txt`). The results you will obtain in Step 1 should be around 76.5% F1.

If you follow the convention above, after parsing a sentence, the parser will remove nodes with labels beginning with "@" before evaluation, and attach *their children so that the transformation is reversed. The resulting recovered tree will be evaluated against the evaluation set. If you do not follow the convention, the reverse transformation will not succeed and the results will be nonsensical. (You can see additional examples of binarization in* `train-tiny-lossless.txt`*.)*

Submit your code for binarization, along with a very short report, giving your results on the validation and the test sets.

STEP 2: Markovization

Due Friday 18th March 2016

In this step, you have to extend / transform the treebank grammar, by implementing horizontal and vertical Markovization as discussed in class. Try at least 4 combinations of the horizontal order $h = \{1, 2\}$ with the vertical order $v = \{1, 2\}$ (the binarization implemented in step 1 can be regarded as $h = \infty$ and $v = 1$). The results with $h = 2$ and $v = 2$ should be around 81% F1 and 29% exact match.

To encode Markovization use the same notation as used in class; this is important for the parser to be able to perform the reverse transformation. In addition to removing nodes with labels beginning with "@" (as described in Step 1), it will also remove all material on node labels which follow their base symbol (cuts at the leftmost _, -, ^, or : character). *Examples: a node with label* `@NP->_DT_JJ` *will be spliced out, and a node with label NP^S will be reduced to NP.* See `train-tiny-h2v2.txt` for some more examples of $h = 2$, $v = 2$ Markovization.

Submit your code and final report
Command line format for Markovization:

```
./b-step2 -h [number] -v [number] -input [non-binarized] -output
[binarized]
```

In the report, state your results for various values of horizontal and vertical markovization, including no markovization.

The final report, summarizing the two stages, submitted along with the last version of the code, should discuss your experiments (e.g., annotation strategies you tried) and findings (at most 4 A4 pages total). Please follow exactly the same procedure to submit your assignments (code and papers) and use the same format as was requested for the Part A of the class.

Extra credit : Option 1

Implement one or more extra transformation(s), including transformations you may invent yourself. You can consider, for example:

- Klein and Manning [1] for different types of transformations additionally to vertical and horizontal Markovization (partially discussed in class);
- Schlund et al. [2] for a new transformation which has not even been tried on English.

Or, you can come up with your own (even radically new!) ideas. IMPORTANT: (1) do not add any annotation to the ROOT symbol; (2) make sure that your annotation is included after '_' or '^' as described above; (3) the annotation should be done before binarization (unless you have good reasons to change the order).

As before, use the validation set for initial experiments and the final test for results you include in the report. If you implemented an additional transformation method, please describe how to use it.

Extra credit : Option 2

Implement the parser (i.e. the CKY algorithm) itself. For pre-terminal rules you should rely on exactly the same smoothing methods you used for emission probabilities in POS tagging ("task model"). The hardest part will be implementation of the CKY algorithm but, given that you have implemented Viterbi in Part A, it should be manageable. Moreover, you can rely on the pseudocode presented in the lecture. In this case, you can either implement scoring yourself or rely on an external script to compute R, P and F1 (e.g., *evalb*: http://nlp.cs.nyu.edu/evalb/).

References

[1] D. Klein and C. D. Manning, *Accurate Unlexicalized Parsing.* Proceedings of the Association for Computational Linguistics (ACL), 2003. http://acl.ldc.upenn.edu/P/P03/P03-1054.pdf

[2] M. Schlund, M. Luttenberger and J. Esparza. *Fast and Accurate Unlexicalized Parsing via Structural Annotations.* Proceedings of European Chapter of the Association for Computational Linguistics (EACL), 2014. http://aclweb.org/anthology//E/E14/E14-4032.pdf