Stanford Parser

How-to by Artur Schmidt modified by Franziska Horn

1 What is the Stanford Parser?

The Stanford Parser is a statistical natural language parser from the Stanford Natural Language Processing Group. Used to parse input data written in several languages such as English, German, Arabic and Chinese it has been developed and maintained since 2002, mainly by Dan Klein and Christopher Manning. The application is licensed under the GNU GPL, but commercial licensing is also available.

2 Installation and Requirements

The parser runs under Windows and Unix/Linux/MacOSX and requires a Java Runtime Environment (JRE) (Java 1.5 or higher). You can download [1] the recommended version 1.6.5 from the website of the Stanford NLP Group.
In a next step, extract the Stanford Parser into a directory of your choice (using Windows, you need to extract it twice). The package contains also a basic graphical user interface (GUI) for visualization of structure trees.

3 Running the Parser

3.1 Parser Models

Parsers for different languages such as Chinese, Arabic, English and German are provided. In most cases, the probabilistic context-free grammar (PCFG) parser will be sufficient, since it processes fast, shows good accuracy values and moderate memory usage. A PCDG model is not provided for parsing German texts. The parser model called *FACTORED* is more complex and requires more memory because it contains two grammars and leads the system to run three parsers. Furthermore, there are two parser files *wsjPCFG.ser.gz* and *wsjFACTORED.ser.gz,* which refer to a parser trained on the Wall Street Journal section of the Penn Treebank project.

[1] For compatibility with Stanford NER (version 1.1.1) download Stanford Parser version 1.6.1.

3.2 Using the GUI

Using the GUI is recommended when you use the Stanford parser for the first time. In order to start the application just launch the file *lexparser-gui.bat* (on Windows systems). Single sentences which can be entered or received by opening a text file can be tagged after selecting a parser file. Furthermore, a visualization of the structure tree can be seen in Figure 1.

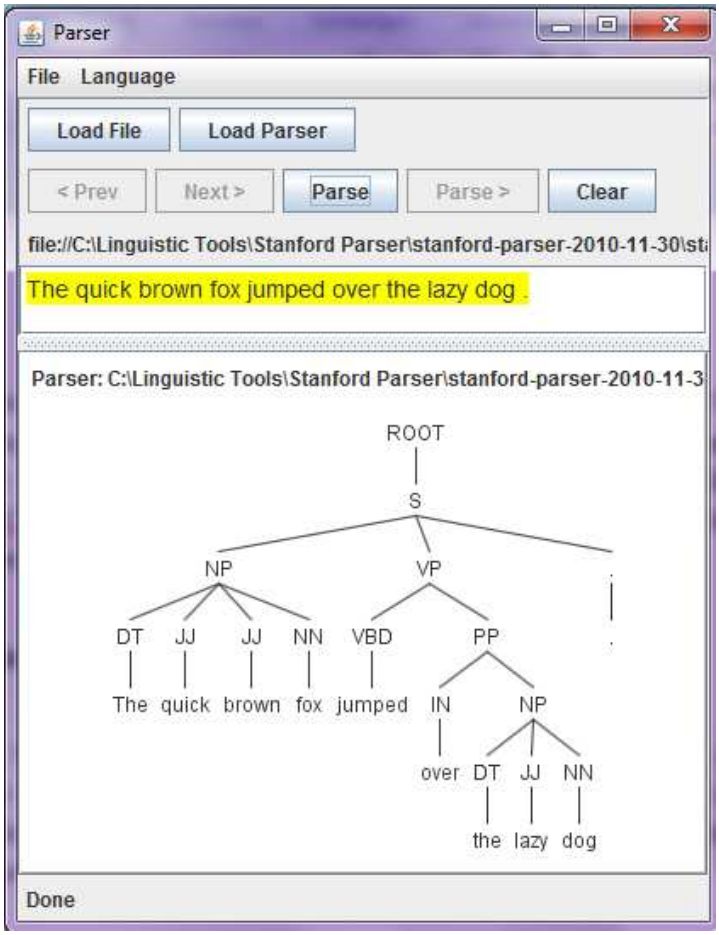

Figure 1 Using the GUI for tagging

There is no possibility to save tagged sentences or tag a whole document at once. Compared with command-line, there are no further output or input options provided.

3.3 Using the command-line

Using the application by command line is recommended because more fine grained control over the parsing process is provided. To apply the Stanford Parser, go into the directory where you have extracted the parser and type the following commands on a command line (no line breaks!):

```
java -mx150m -cp stanford-parser.jar
edu.stanford.nlp.parser.lexparser.LexicalizedParser OPTIONS
parserFile input1 input2 ...
```

ParserFile is the parser model (grammars, lexicon, etc.). For English, for instance, *englishPCFG.ser.gz* and *wsjPCFG.ser.gz* are provided. input1 input2 ... is a list of input files. Further OPTIONS include specifications for input and output format, which are described in the following sections.

3.4 Input Options

The Stanford Parser provides a wide range of functionalities to parse "raw", not preprocessed (e.g. not tokenized) texts. In case you want to use other tools for tokenization, sentence splitting, or tagging, you can add the following command(s) to the OPTIONS parameter:

-maxLength length Limit sentence length to prevent the system running out of memory (useful for unknown texts) by just replacing length by a number, e.g., 50

-tokenized Assumes the input is tokenized

-sentences delimitingToken Assumes that sentences are already split by delimiting Token (e.g. newline to use line breaks)

-tagSeparator separator Assumes the input is already tagged with separator, separating token and tag according to the standards in the Treebank (e.g. the/DT quick/JJ brown/JJ fox/NN … saved in a *.parse file), input data which is only partially tagged can also be used.

```
...-tokenized -sentences newline -tagSeparator / ...
```

3.5 Output Options

Add -outputFormat format to the OPTIONS placeholder to change the output format. Possible options for format written in brackets are presented as follows, exemplified on the tagging process: "The quick brown fox jumped over the lazy dog."

penn Penn Treebank format
This output format is not usable for German because the German parser files are trained on the Negra Corpus using other tags. Using "penn" as output format allows further analysis and visualization of the generated syntactic structures by the tool TreGex.

Example for output:

```
(ROOT
 (S
   (NP (DT The) (JJ quick) (JJ brown) (NN fox))
   (VP (VBD jumped)
     (PP (IN over)
       (NP (DT the) (JJ lazy) (NN dog))))
   (. .)))
```

oneline Penn Treebank format on a single line

Example for output:

```
(ROOT (S (NP (DT The) (JJ quick) (JJ brown) (NN fox)) (VP (VBD jumped) (PP (IN over) (NP (DT the) (JJ lazy) (NN dog)))) (. .)))
```

wordsAndTags Use the parser as a POS tagger

Example for output:

```
The/DT quick/JJ brown/JJ fox/NN jumped/VBD over/IN the/DT lazy/JJ dog/NN ./.
```

typedDependenciesCollapsed Typed dependency format, i.e. grammatical relations between words

Example for output:

```
det(fox-4, The-1)
amod(fox-4, quick-2)
amod(fox-4, brown-3)
nsubj(jumped-5, fox-4)
det(dog-9, the-7)
amod(dog-9, lazy-8)
prep_over(jumped-5, dog-9)
```

-printPCFGkBest n Obtaining multiple (n) parse trees for a single input sentence with their log probabilities to compare. Only applicable using the PCFG parser.

Example for output using option -printPCFGkBest 2

```
# Parse 1 with score -78.23169921338558
(ROOT
  (S
```

```
    (NP (DT The) (JJ quick) (JJ brown) (NN fox))
    (VP (VBD jumped)
      (PP (IN over)
        (NP (DT the) (JJ lazy) (NN dog))))
    (. .)))

# Parse 2 with score -79.60303545906208
(ROOT
  (S
    (NP (DT The) (JJ quick) (JJ brown) (NN fox))
    (VP (VBD jumped)
      (PRT (RP over))
      (NP (DT the) (JJ lazy) (NN dog)))
    (. .)))
```

latexTree LaTeX format to be used with Avery Andrews' trees.sty package[2].

You can also combine several options using a comma-separated list, e.g.

```
...-outputFormat  "penn,typedDependenciesCollapsed"  ...
```

The default output is displayed in the command line. To save the output to file, e.g. output.txt, append the following to your command:

```
> output.txt
```

More command line options are available in the Javadoc of the LexicalizedParser class, especially in the main method.

4 Summary

The Stanford Parser has a good accuracy but further training is possible, e.g. for applying it on domain specific texts. Information on how to train a tagger can be found online.
An online version of the parser is also presented for testing the application.
There is also a plug-in which allows using the parser within GATE which is described here.
Besides its accuracy and various options concerning input as well as output data, its compatibility with the tool TregEx is another advantage of the Stanford Parser. TregEx allows visualization of structure trees generated by the Stanford Parser and

[2] Further pieces of information are available online.

querying them for certain syntactic patterns. Thus, the applications can be used for analyzing the tree data structures statistically.

5 References

Andrews, A. (2006). *Tree typesetting package for LaTex.* (last access 02.02.2011 23:38) retrieved from http://arts.anu.edu.au/linguistics/People/AveryAndrews/Software/latex/

Manning, C. & Klein, D. (2008). *The stanford parser.* (last access 20.01.11 14:54) retrieved from http://nlp.stanford.edu/software/lex-parser.shtml