Patrick Jeuniaux

Java

Download and Install Java

http://java.sun.com/javase/downloads/index.jsp

Example 1: Syntactic Parsing

The Problem

Imagine that you want to retrieve parts-of-speech and a syntactical structure for each sentence in a large corpus. For the sake of the demonstration, let’s focus on a small set of sentences (see Table 1).

Table 1. A five-sentence text in multiple languages.

The fact that he smiled at me gives me hope
The event that he smiled at me gives me hope
Le fait qu'il me sourit, me donne de l'espoir
Het feit dat hij glimlachte naar me geeft me hoop
Die Tatsache, dass er lächelte mich an, gibt mir Hoffnung

A Solution

Klein and Manning (2003) have proposed a statistical technique for inferring the syntactical structure of sentences. Their parser, called the “Stanford Parser”, is freely available at http://nlp.stanford.edu/software. It is written in Java but is also accessible through Ruby or a Web Service within GATE. Since we are most familiar with Java, we decide to demonstrate the use of the parser in Java.

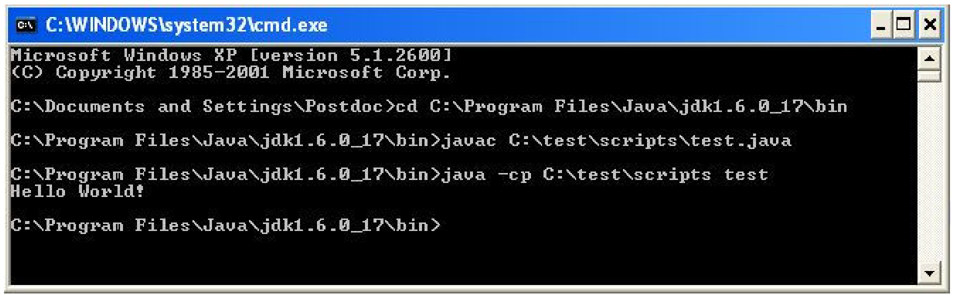
If we only wanted to run a precompiled Java program, Java Runtime Environment (JRE) would suffice, but in this exercise we will compile the parser ourselves in order to increase the educational value of this exercise, and therefore we need a Java Development Kit (JDK). We use Sun's freely available Java SE (Standard Edition) Development Kit to compile Java programs. The kit can be downloaded at http://java.sun.com/javase/downloads/ The file you need to download is an executable (on Windows they usually end by `.exe`). Most browsers will ask your permission for downloading such files. When you have downloaded the file, execute it, and choose the default options.

Testing the compiler: Hello World !

We now want to test `javac` – the Java compiler shipped with Java SE – on a simple "Hello World!" program (download test.java). However, note that the program looks a little bit more complicated than many examples shown in the chapter because Java has an object-oriented focus (i.e., programs are composed of objects, which are defined by classes). To facilitate reading, comments have been added in the code. They start with two slashes (`//`). We are now going to compile test.java. Use the terminal and type the commands provided in Table 2, one by one. Figure 1 shows the oucome of these commands.

Table 2. Commands for testing that the Java compiler is running properly

	Command to enter	Effect of the command
1	`cd C:\Program Files\Java\jdk1.6.0_17\bin`	You move to the folder where the programs javac and java are located.
2	`javac C:\test\scripts\test.java`	The test.java program is compiled. This creates a test.class file in the scripts folder.
3	`java -cp C:\test\scripts test`	The test.class file is executed. You should see `Hello World!` in the terminal.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Postdoc>cd C:\Program Files\Java\jdk1.6.0_17\bin

C:\Program Files\Java\jdk1.6.0_17\bin>javac C:\test\scripts\test.java

C:\Program Files\Java\jdk1.6.0_17\bin>java -cp C:\test\scripts test
Hello World!

C:\Program Files\Java\jdk1.6.0_17\bin>
```

Figure 1: Hello World! in Java

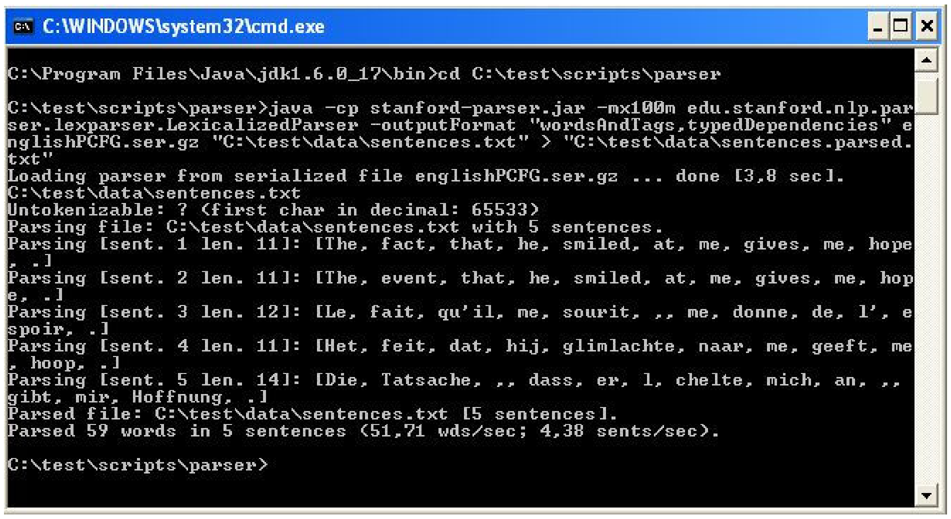
Use the Standford Parser on the command line

Now that we know we can compile and run a Java program, let's download the Stanford Parser at http://nlp.stanford.edu/software/lex-parser.shtml. The downloaded file is a compressed .tar file (e.g., stanford-parser-2008-10-26.tgz.tar). You can uncompress it with the freeware 7-zip (http://www.7-zip.org). This will result in a .tgz file, which also needs to be uncompressed. The final result is a folder (let's rename it to *parser*, for short) containing two folders (*javadoc* and *src*) and a large number of files (some of which are compressed; do not uncompress them). Move the *parser* folder inside the scripts folder. Look into the parser folder. Documentation about the parser can be found in the javadoc subfolder, but a good place to start is the frequently asked question (FAQ) page at http://nlp.stanford.edu/software/parser-faq.shtml. The folder structure that you should have is apparent in Figure 1.

Let's first check that the parser is working by running it at the command line. For that, use the terminal and type the instructions given in Table 2. The outcome of these commands is displayed in Figure 2.

Table 2 Commands for directly using the parser at command line

	Command to enter	Effect of the command
1	`cd C:\test\scripts\parser`	You move to the folder where the parser is.
2	`java -cp stanford-parser.jar -mx100m edu.stanford.nlp.parser.lexparser.LexicalizedParser -outputFormat "wordsAndTags,typedDependencies" englishPCFG.ser.gz "C:\test\data\sentences.txt" > "C:\test\data\sentences.parsed.txt"`	Java is informed where the parser is (.jar file) through the classpath (cp) argument. The class to be used in the jar file is specified (mx100m). The outputFormat argument specifies the nature of the output. The last arguments specify the paths for the input and output. The output file sentences.parsed.txt is generated with syntactical information for each of the sentences of Table 5.



```
C:\Program Files\Java\jdk1.6.0_17\bin>cd C:\test\scripts\parser

C:\test\scripts\parser>java -cp stanford-parser.jar -mx100m edu.stanford.nlp.par
ser.lexparser.LexicalizedParser -outputFormat "wordsAndTags,typedDependencies" e
nglishPCFG.ser.gz "C:\test\data\sentences.txt" > "C:\test\data\sentences.parsed.
txt"
Loading parser from serialized file englishPCFG.ser.gz ... done [3,8 sec].
C:\test\data\sentences.txt
Untokenizable: ? (first char in decimal: 65533)
Parsing file: C:\test\data\sentences.txt with 5 sentences.
Parsing [sent. 1 len. 11]: [The, fact, that, he, smiled, at, me, gives, me, hope
, .]
Parsing [sent. 2 len. 11]: [The, event, that, he, smiled, at, me, gives, me, hop
e, .]
Parsing [sent. 3 len. 12]: [Le, fait, qu'il, me, sourit, ,, me, donne, de, l', e
spoir, .]
Parsing [sent. 4 len. 11]: [Het, feit, dat, hij, glimlachte, naar, me, geeft, me
, hoop, .]
Parsing [sent. 5 len. 14]: [Die, Tatsache, ,, dass, er, l, chelte, mich, an, ,,
gibt, mir, Hoffnung, .]
Parsed file: C:\test\data\sentences.txt [5 sentences].
Parsed 59 words in 5 sentences (51,71 wds/sec; 4,38 sents/sec).

C:\test\scripts\parser>
```

Figure 2 Using the NLP parser at command line.

It might be intimidating at first, and it surely is not a piece of artworks, but if you take the time to inspect it, it gives you valuable information about what happened during the process. For instance, the output informs us of the number of sentences being parsed, the time it took to parse them, the number of words, and the fact that there was an accident during tokenization. That error occurred because of the diacritic letter ä in the German sentence, which is not supported by ASCII (the "American Standard Code for Information Interchange"), a character-encoding scheme which is strictly based on the English alphabet. Luckily for us the last command creates a new file `sentences.parsed.txt` with a reduced amount of information. The content of that file is displayed for the first sentence in Table 3. First, each word in the sentence is associated to its part-of-speech. For instance, the part-of-speech *DT* (i.e., determiner) is associated to the word *The*. Second, the syntactical structure of the sentence follows in a typed dependencies format. For instance, a determiner relation exists between the noun phrase *fact* and the determiner *the*. It is interesting to see that the same syntactical tree is proposed for the second English sentence, which is not grammatically correct, in conventional English. This illustrates the fact that this parser is not about sanctioning the appropriateness of language fragments but attempting to see how its elements can fit together.

Table 3 Words and part-of-speech tags and typed dependencies returned by the Stanford parser for one sentence.

```
The/DT fact/NN that/IN he/PRP smiled/VBD at/IN me/PRP gives/VBZ me/PRP hope/VBP ./.

det(fact-2, The-1)

nsubj(gives-8, fact-2)

complm(smiled-5, that-3)

nsubj(smiled-5, he-4)

dep(fact-2, smiled-5)

prep_at(smiled-5, me-7)

nsubj(hope-10, me-9)

ccomp(gives-8, hope-10)
```

Use the Standford Parser programmatically

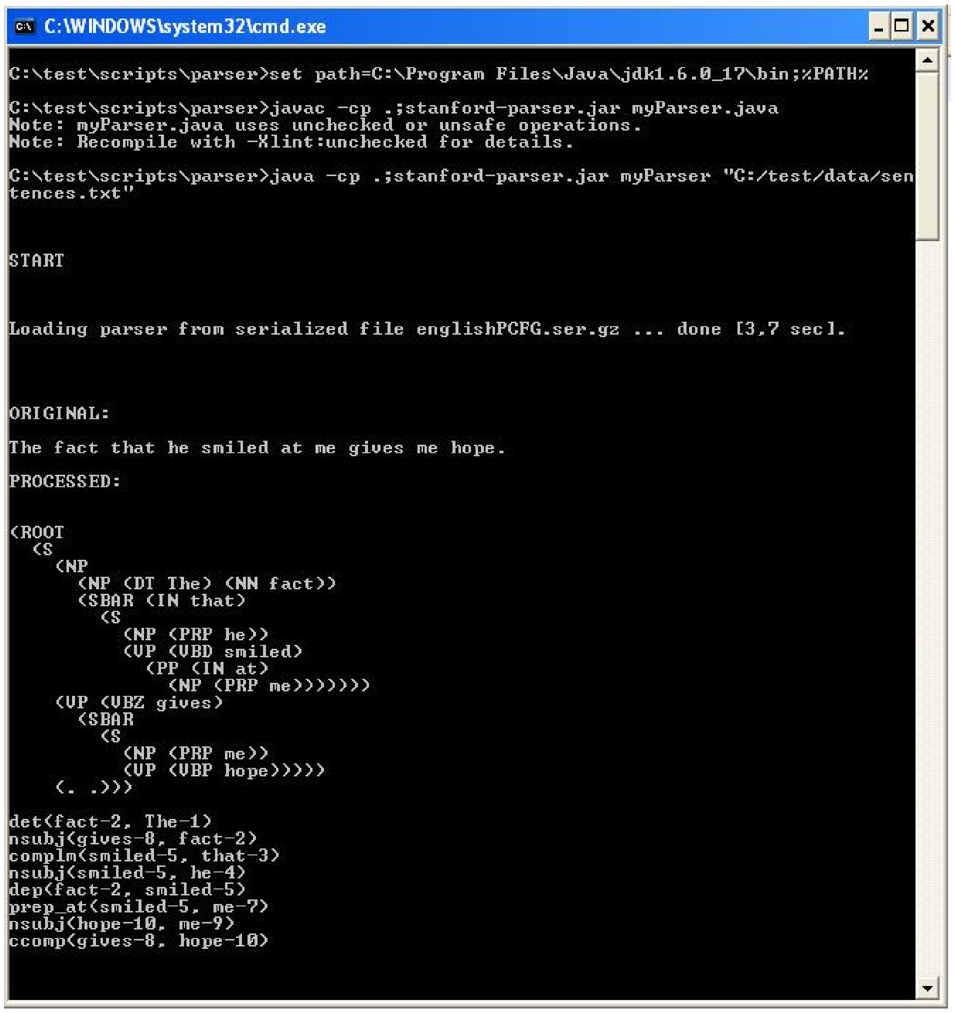
Now that we know that the parser works properly, we are going to call it programmatically (i.e., from within a Java program). Why would we want to do that? Because the point with programming is that you can flexibly reuse parts of code to achieve your objectives. Hence, if the nature of the commands given in Table 2 does not fit your needs, a bit of programming can help you meet them. Download the program `myParser.java` and save it in the parser folder (i.e., `C:\test\scripts\parser`). This program is a variant of some examples described in the documentation of the Stanford Parser. It involves sophisticated actions: sentences are read one by one from a file, they are tokenized, parsed, and the best parse is printed in two formats. This is a good example of the power of efficient code reuse, in which objects are called to perform various operations.

Now invoke the terminal and follow the instructions listed in Table 4 to compile and run the program. When the program is running, you should see on the screen syntactical information like the one presented in 3, plus another syntactical representation. Execute it to see the difference (or see Figure 3)!

Table 4 Commands for compiling a class using the parser.

	Command to enter	Effect of the command
1	`javac`	If you are not in the bin's SDK folder, this command will not be recognized.
2	`set path=C:\Program Files\Java\jdk1.6.0_17 \bin;%PATH%`	Temporarily add the bin's SDK folder to the environment variable called *path* (until you close the terminal).
3	`javac`	Now, it should be recognized. If not, take a deep breath and troubleshoot!
4	`cd C:\test\parser`	Go to where the parser files are.

5	`javac -cp .;stanford-parser.jar myParser.java`	Compile the file.
6	`java -cp .;stanford-parser.jar myParser "C:/test/data/sentences.txt"`	Run the Java program, passing the path to the input file, as an argument.



```
C:\test\scripts\parser>set path=C:\Program Files\Java\jdk1.6.0_17\bin;%PATH%

C:\test\scripts\parser>javac -cp .;stanford-parser.jar myParser.java
Note: myParser.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\test\scripts\parser>java -cp .;stanford-parser.jar myParser "C:/test/data/sen
tences.txt"


START


Loading parser from serialized file englishPCFG.ser.gz ... done [3.7 sec].



ORIGINAL:

The fact that he smiled at me gives me hope.

PROCESSED:

(ROOT
  (S
    (NP
      (NP (DT The) (NN fact))
      (SBAR (IN that)
        (S
          (NP (PRP he))
          (VP (VBD smiled)
            (PP (IN at)
              (NP (PRP me)))))))
    (VP (VBZ gives)
      (SBAR
        (S
          (NP (PRP me))
          (VP (VBP hope)))))
    (. .)))

det(fact-2, The-1)
nsubj(gives-8, fact-2)
complm(smiled-5, that-3)
nsubj(smiled-5, he-4)
dep(fact-2, smiled-5)
prep_at(smiled-5, me-7)
nsubj(hope-10, me-9)
ccomp(gives-8, hope-10)
```

Figure 3 Using the NLP parser programmatically.