Web-based Interfaces for Natural Language Processing Tools

Marc Light[†] and **Robert Arens**[*] **and Xin Lu**[*]
[†]Linguistics Department
[†]School of Library and Information Science
[*†]Computer Science Department
University of Iowa
Iowa, USA 52242
{marc-light,robert-arens,xin-lu}@uiowa.edu

Abstract

We have built web interfaces to a number of Natural Language Processing technologies. These interfaces allow students to experiment with different inputs and view corresponding output and inner workings of the systems. When possible, the interfaces also enable the student to modify the knowledge bases of the systems and view the resulting change in behavior. Such interfaces are important because they allow students **without** computer science background to learn by doing. Web interfaces also sidestep issues of platform dependency in software packages, available computer lab times, etc. We discuss our basic approach and lessons learned.

1 Introduction

The Problem: Natural language processing (NLP) technology is relevant to non-computer scientists: our classes are populated by students from neuroscience, speech pathology, linguistics, teaching of foreign languages, health informatics, etc. To effectively use NLP technology, it is helpful understand, at some level, how it works. Hands-on experimentation is an effective method for gaining such understanding. **Unfortunately**, to be able to experiment, non-computer scientists often need to acquire some programming skills and knowledge of the Unix operating system. This can be time consuming and tedious and can distract students from their central goal of understanding how a technology works and how best to employ it for their interests.

In addition, getting a technology to run on a set lab machines can be problematic: the programs may be developed for a different platform, e.g., a program was developed for Linux but the lab machines run MSWindows. Another hurdle is that machine administrators are often loath to install applications that they perceive as non-standard. Finally, lab times can be restrictive and thus it is preferable to enable students to use computers to which they have easy access.

Our Solution: We built web interfaces to many core NLP modules. These interfaces not only allow students to use a technology but also allow students to modify and extend the technology. This enables experimentation. We used server-side scripting languages to build such web interfaces. These programs take input from a web browser, feed it to the technology in question, gather the output from the technology and send it back to the browser for display to the student. Access to web browsers is nearly ubiquitous and thus the issue of lab access is side-stepped. Finally, the core technology need only run on the web server platform. Many instructors have access to web servers running on different platforms and, in general, administering a web server is easier than maintaining lab machines.

An Example: Finite state transduction is a core NLP technology and one that students need to understand. The Cass partial parsing system (Abney, 1997) makes use of a cascade of FSTs. To use this system, a student creates a grammar. This grammar is compiled and then applied to sentences provided

Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL, pages 28–31,
Ann Arbor, June 2005.

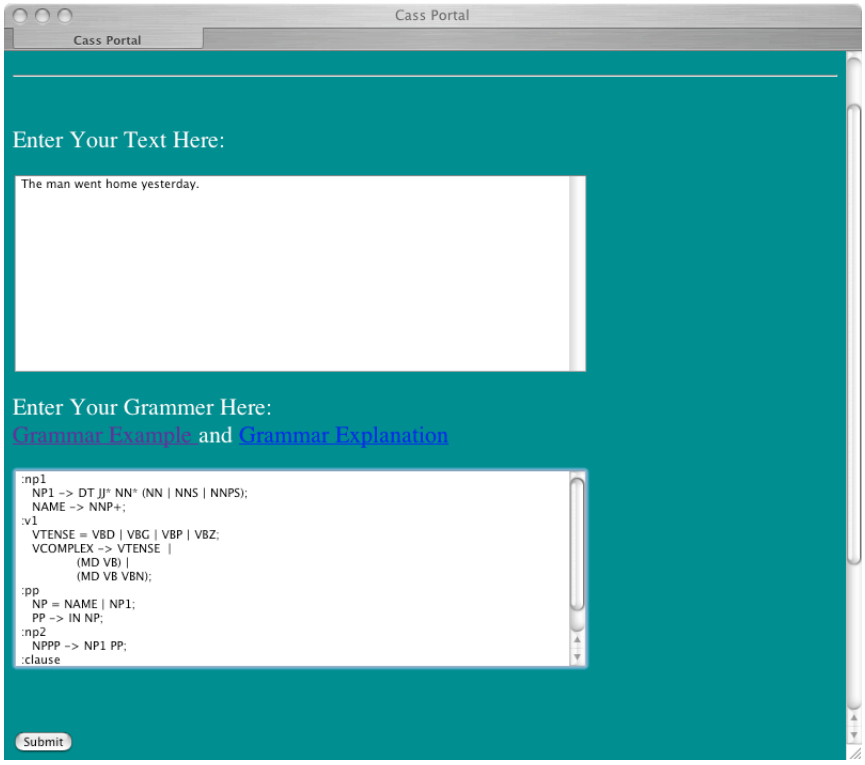

Figure 1: Web interface to Cass

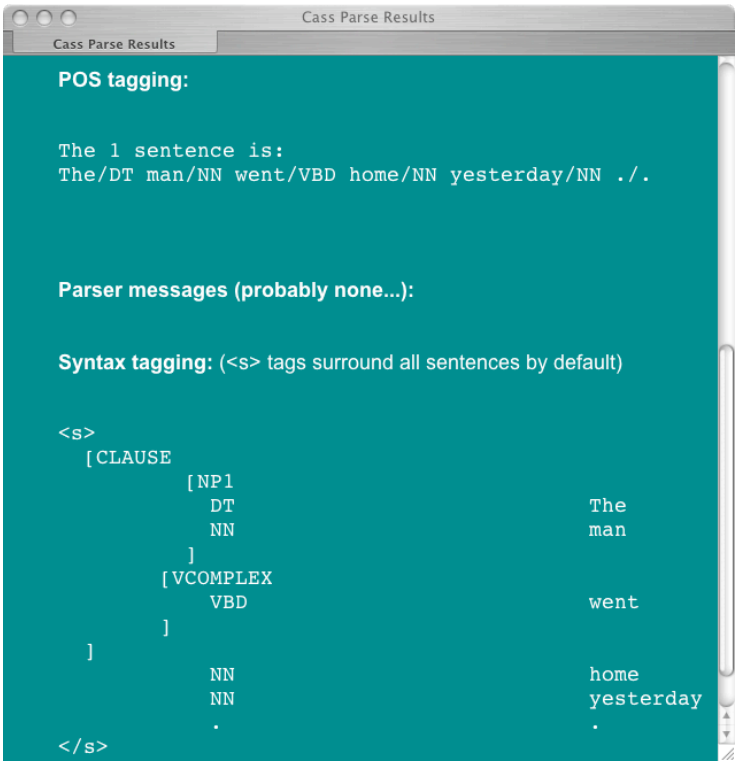

Figure 2: Cass Output

by the student. Prior to our work, the only interface to Cass involved the Unix command line shell. Figure 3 shows an example session with the command line interface. It exemplifies the sort of interface that users must master in order to work with current human language technology.

```
1 emacs input.txt &
2 emacs grammar.txt &
3 source /usr/local/bin/setupEnv
3 reg gram.txt
4 Montytagger.py inTagged input.txt
5 cat inTagged |
6 wordSlashTagInput.pl |
7 cass -v -g gram.txt.fsc > cassOut
8 less cassOut
```

Figure 3: Cass Command Line Interface

A web-based interface hides many of the details, see Figure 1 and Figure 2. For example, the use of an ASCII-based text editor such as `emacs` become unnecessary. In addition, the student does not need to remembering flags such as `-v -g` and does not need to know how to use Unix pipes, |, and output redirection, >. None of this knowledge is terribly difficult but the amount accumulates quickly and such information does *not* help the student understand how Cass works.

2 What we have built

To date, we have built web interfaces to nine NLP-related technologies:

- the Cass parser (Abney, 1997),
- the MontyTagger Brill-style part-of-speech tagger (Liu, 2004),
- the NLTK statistical part-of-speech tagger,
- a NLTK context-free grammar parser (Loper and Bird, 2002),
- the Gsearch context-free grammar parser (Corley et al., 2001),
- the SenseRelate word sense disambiguation system (Pedersen et al., 2005),
- a Perl Regular expression evaluator,
- a linguistic feature annotator,
- and a decision tree classifier (Witten and Frank, 1999).

These interfaces have been used in an introduction to computational linguistics course and an introduction to creating and using corpora course. Prior to the interface construction, no hands-on lab assignments were given; instead all assignments were pencil and paper. The NLP technologies listed above were chosen because they fit into the material of the course and because of their availability.

2.1 Allowing the student to process input

The simplest type of interface allows students to provide input and displays corresponding output. All the interfaces above provide this ability. They all start with HTML forms to collect input. In the simplest case, PHP scripts process the forms, placing input into files and then system calls are made to run the NLP technology. Finally, output files are wrapped in HTML and displayed to the user. The basic PHP program remains largely unchanged from one NLP technology to the next. In most cases, it suffices to use the server file system to pass data back and forth to the NLP program — PHP provides primitives for creating and removing unique temporary files. In only one case was it necessary to use a semaphore on a hard-coded filename. We also experimented with Java server pages and Perl CGI scripts instead of PHP.

2.2 Allowing the student to modify knowledge resources

The web interfaces to the Cass parser, Gsearch, and MontyTagger allow the student to provide their corresponding knowledge base. For Cass and Gsearch, an additional text box is provided for the grammars they require. The rule sequence and lexicon that the MontyTagger uses can be large and thus unwieldy for a `textarea` form input element. We solved the problem by preloading the `textarea`s with a "standard" rule sequence and lexicon which the student can then modify. We also provided the ability to upload the rule sequences and lexicon as files. One problem with the file upload method is that it assume that the students can generate ASCII-only files with

the appropriate line break character. This assumption is often false.

An additional problem with allowing students to modify knowledge resources is providing useful feedback when these student-provided resources contain syntax or other types of errors. At this point we simply capture the `stderr` output of the program and display it.

Finally, with some systems such as Spew (Schwartz, 1999), and The Dada Engine (Bulhak, 1996), allowing web-based specification of knowledge bases amounts to allowing the student to execute arbitrary code on the server machine, an obvious security problem.

2.3 Allowing the student to examine internal system processing

Displaying system output with a web interface is relatively easy; however, showing the internal workings of a system is more challenging with a web interface. At this point, we have only displayed traces of steps of an algorithm. For example, the NLTK context-free grammar parser interface provides a trace of the steps of the parsing algorithm. One possible solution would be to generate Flash code to animate a system's processing.

2.4 Availability

The web pages are currently available at que.infoscience.uiowa.edu/~light/classes/compLing/ However, it is not our intent to provide server cycles for the community but rather to provide the PHP scripts open source so that others can run the interfaces on their own servers. An instructor at another university has already made use of our code.

3 Lessons learned

- PHP is easier to work with than Java Server Pages and CGI scripts;
- requiring users to paste input into text boxes is superior to allowing user to upload files (for security reasons and because it is easier to control the character encoding used);
- getting debugging information back to the student is very important;
- security is an issue since one is allowing users to initiate computationally intensive processes;
- it is still possible for students to claim the interface does not work for them (even though we used no client-side scripting).
- Peer learning is less likely than in a lab setting; however, we provided a web forum and this seems to alleviated the problem somewhat.

4 Summary

At the University of Iowa, many students, who want to learn about natural language processing, do not have the requisite Unix and programming skills to do labs using command line interfaces. In addition, our lab machines run MSWindows, the instructors do not administer the machines, and there are restrictive lab hours. Thus, until recently assignments consisted of pencil-and-paper problems. We have built web-based interfaces to a number of NLP modules that allow students to use, modify, and learn.

References

Steven Abney. 1997. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4).

Andrew Bulhak. 1996. The dada engine. http://dev.null.org/dadaengine/.

S. Corley, M. Corley, F. Keller, M. Crocker, and S. Trewin. 2001. Finding Syntactic Structure in Unparsed Corpora: The Gsearch Corpus Query System. *Computers and the Humanities*, 35:81–94.

Hugo Liu. 2004. Montylingua: An end-to-end natural language processor with common sense. homepage.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proc. of the ACL-02 Workshop on Effective Tools and Methods for Teaching Natural Language Processing and Computational Linguistics*.

Ted Pedersen, Satanjeev Banerjee, and Siddharth Patwardhan. 2005. Maximizing Semantic Relatedness to Perform Word Sense Disambiguation. Supercomputing institute research report umsi 2005/25, University of Minnesota.

Randal Schwartz. 1999. Random sentence generator. *Linux Magazine*, September.

Ian H. Witten and Eibe Frank. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.