Multidisciplinary Instruction with the Natural Language Toolkit

Steven Bird
Department of Computer Science
University of Melbourne
sb@csse.unimelb.edu.au

Ewan Klein
School of Informatics
University of Edinburgh
ewan@inf.ed.ac.uk

Edward Loper
Computer and Information Science
University of Pennsylvania
edloper@gradient.cis.upenn.edu

Jason Baldridge
Department of Linguistics
University of Texas at Austin
jbaldrid@mail.utexas.edu

Abstract

The Natural Language Toolkit (NLTK) is widely used for teaching natural language processing to students majoring in linguistics or computer science. This paper describes the design of NLTK, and reports on how it has been used effectively in classes that involve different mixes of linguistics and computer science students. We focus on three key issues: getting started with a course, delivering interactive demonstrations in the classroom, and organizing assignments and projects. In each case, we report on practical experience and make recommendations on how to use NLTK to maximum effect.

1 Introduction

It is relatively easy to teach natural language processing (NLP) in a single-disciplinary mode to a uniform cohort of students. Linguists can be taught to program, leading to projects where students manipulate their own linguistic data. Computer scientists can be taught methods for automatic text processing, leading to projects on text mining and chatbots. Yet these approaches have almost nothing in common, and it is a stretch to call either of these NLP: more apt titles for such courses might be "linguistic data management" and "text technologies."

The Natural Language Toolkit, or NLTK, was developed to give a broad range of students access to the core knowledge and skills of NLP (Loper and Bird, 2002). In particular, NLTK makes it feasible to run a course that covers a substantial amount of theory and practice with an audience consisting of both linguists and computer scientists. NLTK is a suite of Python modules distributed under the GPL open source license via `nltk.org`. NLTK comes with a large collection of corpora, extensive documentation, and hundreds of exercises, making NLTK unique in providing a comprehensive framework for students to develop a computational understanding of language. NLTK's code base of 100,000 lines of Python code includes support for corpus access, tokenizing, stemming, tagging, chunking, parsing, clustering, classification, language modeling, semantic interpretation, unification, and much else besides. As a measure of its impact, NLTK has been used in over 60 university courses in 20 countries, listed on the NLTK website.

Since its inception in 2001, NLTK has undergone considerable evolution, based on the experience gained by teaching courses at several universities, and based on feedback from many teachers and students.[1] Over this period, a series of practical online tutorials about NLTK has grown up into a comprehensive online book (Bird et al., 2008). The book has been designed to stay in lock-step with the NLTK library, and is intended to facilitate "active learning" (Bonwell and Eison, 1991).

This paper describes the main features of NLTK, and reports on how it has been used effectively in classes that involve a combination of linguists and computer scientists. First we discuss aspects of the design of the toolkit that

[1](Bird and Loper, 2004; Loper, 2004; Bird, 2005; Hearst, 2005; Bird, 2006; Klein, 2006; Liddy and McCracken, 2005; Madnani, 2007; Madnani and Dorr, 2008; Baldridge and Erk, 2008)

Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08), pages 62–70,
Columbus, Ohio, USA, June 2008. ©2008 Association for Computational Linguistics

arose from our need to teach computational linguistics to a multidisciplinary audience (§2). The following sections cover three distinct challenges: getting started with a course (§3); interactive demonstrations (§4); and organizing assignments and projects (§5).

2 Design Decisions Affecting Teaching

2.1 Python

We chose Python[2] as the implementation language for NLTK because it has a shallow learning curve, its syntax and semantics are transparent, and it has good string-handling functionality. As an interpreted language, Python facilitates interactive exploration. As an object-oriented language, Python permits data and methods to be encapsulated and re-used easily. Python comes with an extensive standard library, including tools for graphical programming and numerical processing, which means it can be used for a wide range of non-trivial applications. Python is ideal in a context serving newcomers and experienced programmers (Shannon, 2003).

We have taken the step of incorporating a detailed introduction to Python programming in the NLTK book, taking care to motivate programming constructs with linguistic examples. Extensive feedback from students has been humbling, and revealed that for students with no prior programming experience, it is almost impossible to over-explain. Despite the difficulty of providing a self-contained introduction to Python for linguists, we nevertheless have also had very positive feedback, and in combination with the teaching techniques described below, have managed to bring a large group of non-programmer students rapidly to a point where they could carry out interesting and useful exercises in text processing.

In addition to the NLTK book, the code in the NLTK core is richly documented, using Python docstrings and Epydoc[3] support for API documentation.[4] Access to the code documentation is available using the Python `help()` command at the interactive prompt, and this can be especially useful for checking the parameters and return type of functions.

[2]`http://www.python.org/`
[3]`http://epydoc.sourceforge.net/`
[4]`http://nltk.org/doc/api/`

Other Python libraries are useful in the NLP context: NumPy provides optimized support for linear algebra and sparse arrays (NumPy, 2008) and PyLab provides sophisticated facilities for scientific visualization (Matplotlib, 2008).

2.2 Coding Requirements

As discussed in Loper & Bird (2002), the priorities for NLTK code focus on its teaching role. When code is readable, a student who doesn't understand the maths of HMMs, smoothing, and so on may benefit from looking at how an algorithm is implemented. Thus consistency, simplicity, modularity are all vital features of NLTK code. A similar importance is placed on extensibility, since this helps to ensure that the code grows as a coherent whole, rather than by unpredictable and haphazard additions.

By contrast, although efficiency cannot be ignored, it has always taken second place to simplicity and clarity of coding. In a similar vein, we have tried to avoid clever programming tricks, since these typically hinder intelligibility of the code. Finally, comprehensiveness of coverage has never been an overriding concern of NLTK; this leaves open many possibilities for student projects and community involvement.

2.3 Naming

One issue which has absorbed a considerable amount of attention is the naming of user-oriented functions in NLTK. To a large extent, the system of naming *is* the user interface to the toolkit, and it is important that users should be able to guess what action might be performed by a given function. Consequently, naming conventions need to be consistent and semantically transparent. At the same time, there is a countervailing pressure for relatively succinct names, since excessive verbosity can also hinder comprehension and usability. An additional complication is that adopting an object-oriented style of programming may be well-motivated for a number of reasons but nevertheless baffling to the linguist student. For example, although it is perfectly respectable to invoke an instance method `WordPunctTokenizer().tokenize(text)` (for some input string `text`), a simpler version is also provided: `wordpunct_tokenize(text)`.

2.4 Corpus Access

The scope of exercises and projects that students can perform is greatly increased by the inclusion of a large collection of corpora, along with easy-to-use corpus readers. This collection, which currently stands at 45 corpora, includes parsed, POS-tagged, plain text, categorized text, and lexicons.[5]

In designing the corpus readers, we emphasized simplicity, consistency, and efficiency. *Corpus objects*, such as `nltk.corpus.brown` and `nltk.corpus.treebank`, define common methods for reading the corpus contents, abstracting away from idiosyncratic file formats to provide a uniform interface. See Figure 1 for an example of accessing POS-tagged data from different tagged and parsed corpora.

The corpus objects provide methods for loading corpus contents in various ways. Common methods include: `raw()`, for the raw contents of the corpus; `words()`, for a list of tokenized words; `sents()`, for the same list grouped into sentences; `tagged_words()`, for a list of (*word*, *tag*) pairs; `tagged_sents()`, for the same list grouped into sentences; and `parsed_sents()`, for a list of parse trees. Optional parameters can be used to restrict what portion of the corpus is returned, e.g., a particular section, or an individual corpus file.

Most corpus reader methods return a *corpus view* which acts as a list of text objects, but maintains responsiveness and memory efficiency by only loading items from the file on an as-needed basis. Thus, when we print a corpus view we only load the first block of the corpus into memory, but when we process this object we load the whole corpus:

```
>>> nltk.corpus.alpino.words()
['De', 'verzekeringsmaatschappijen',
'verhelen', ...]
>>> len(nltk.corpus.alpino.words())
139820
```

2.5 Accessing Shoebox Files

NLTK provides functionality for working with "Shoebox" (or "Toolbox") data (Robinson et al., 2007). Shoebox is a system used by many documentary linguists to produce lexicons and interlinear glossed text. The ability to work straightforwardly with Shoebox data has created a new incentive for linguists to learn how to program.

As an example, in the Linguistics Department at the University of Texas at Austin, a course has been offered on Python programming and working with corpora,[6] but so far uptake from the target audience of core linguistics students has been low. They usually have practical computational needs and many of them are intimidated by the very idea of programming. We believe that the appeal of this course can be enhanced by designing a significant component with the goal of helping documentary linguistics students take control of their *own* Shoebox data. This will give them skills that are useful for their research and also transferable to other activities. Although the NLTK Shoebox functionality was not originally designed with instruction in mind, its relevance to students of documentary linguistics is highly fortuitous and may prove appealing for similar linguistics departments.

3 Getting Started

NLP is usually only available as an elective course, and students will vote with their feet after attending one or two classes. This initial period is important for attracting and retaining students. In particular, students need to get a sense of the richness of language in general, and NLP in particular, while gaining a realistic impression of what will be accomplished during the course and what skills they will have by the end. During this time when rapport needs to be rapidly established, it is easy for instructors to alienate students through the use of linguistic or computational concepts and terminology that are foreign to students, or to bore students by getting bogged down in defining terms like "noun phrase" or "function" which are basic to one audience and new for the other. Thus, we believe it is crucial for instructors to understand and shape the student's expectations, and to get off to a good start. The best overall strategy that we have found is to use succinct nuggets of NLTK code to stimulate students' interest in both data and processing techniques.

[5] `http://nltk.org/corpora.html`

[6] `http://comp.ling.utexas.edu/courses/2007/corpora07/`

```
>>> nltk.corpus.treebank.tagged_words()
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ...]
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
>>> nltk.corpus.floresta.tagged_words()
[('Um', '>N+art'), ('revivalismo', 'H+n'), ...]
>>> nltk.corpus.cess_esp.tagged_words()
[('El', 'da0ms0'), ('grupo', 'ncms000'), ...]
>>> nltk.corpus.alpino.tagged_words()
[('De', 'det'), ('verzekeringsmaatschappijen', 'noun'), ...]
```

Figure 1: Accessing Different Corpora via a Uniform Interface

3.1 Student Expectations

Computer science students come to NLP expecting to learn about NLP algorithms and data structures. They typically have enough mathematical preparation to be confident in playing with abstract formal systems (including systems of linguistic rules). Moreover, they are already proficient in multiple programming languages, and have little difficulty in learning NLP algorithms by reading and manipulating the implementations provided with NLTK. At the same time, they tend to be unfamiliar with the terminology and concepts that linguists take for granted, and may struggle to come up with reasonable linguistic analyses of data.

Linguistics students, on the other hand, are interested in understanding NLP algorithms and data structures only insofar as it helps them to use computational tools to perform analytic tasks from "core linguistics," e.g. writing a set of CFG productions to parse some sentences, or plugging together NLP components in order to derive the subcategorization requirements of verbs in a corpus. They are usually not interested in reading significant chunks of code; it isn't what they care about and they probably lack the confidence to poke around in source files.

In a nutshell, the computer science students typically want to analyze the tools and synthesize new implementations, while the linguists typically want to use the tools to analyze language and synthesize new theories. There is a risk that the former group never really gets to grips with natural language, while the latter group never really gets to grips with processing. Instead, computer science students need to learn that NLP is not just an application of techniques from formal language theory and compiler construction, and linguistics students need to understand that NLP is not just computer-based housekeeping and a solution to the shortcomings of office productivity software for managing their data.

In many courses, linguistics students or computer science students will dominate the class numerically, simply because the course is only listed in one department. In such cases it is usually enough to provide additional support in the form of some extra readings, tutorials, and exercises in the opening stages of the course. In other cases, e.g. courses we have taught at the universities of Edinburgh, Melbourne, Pennsylvania, and Texas-Austin or in summer intensive programs in several countries, there is more of an even split, and the challenge of serving both cohorts of students becomes acute. It helps to address this issue head-on, with an early discussion of the goals of the course.

3.2 Articulating the Goals

Despite an instructor's efforts to add a cross-disciplinary angle, students easily "revert to type." The pressure of assessment encourages students to emphasize what they do well. Students' desire to understand what is expected of them encourages instructors to stick to familiar assessment instruments. As a consequence, the path of least resistance is for students to remain firmly monolingual in their own discipline, while acquiring a smattering of words from a foreign language, at a level we might call "survival linguistics" or "survival computer science." If they ever get to work in a multidisciplinary team they are likely only to play a type-cast role.

Asking computer science students to write their first essay in years, or asking linguistics students to write their first ever program, leads to stressed students who complain that they don't know what is expected of them. Nevertheless, students need to confront the challenge of becoming bilingual, of working hard to learn the basics of another discipline. In parallel, instructors need to confront the challenge of synthesizing material from linguistics and computer science into a coherent whole, and devising effective methods for teaching, learning, and assessment.

3.3 Entry Points

It is possible to identify several distinct pathways into the field of Computational Linguistics. Bird (2008) identifies four; each of these are supported by NLTK, as detailed below:

Text Processing First: NLTK supports variety of approaches to tokenization, tagging, evaluation, and language engineering more generally.

Programming First: NLTK is based on Python and the documentation teaches the language and provides many examples and exercises to test and reinforce student learning.

Linguistics First: Here, students come with a grounding in one or more areas of linguistics, and focus on computational approaches to that area by working with the relevant chapter of the NLTK book in conjunction with learning how to program.

Algorithms First: Here, students come with a grounding in one or more areas of computer science, and can use, test and extend NLTK'S reference implementations of standard NLP algorithms.

3.4 The First Lecture

It is important that the first lecture is effective at motivating and exemplifying NLP to an audience of computer science and linguistics students. They need to get an accurate sense of the interesting conceptual and technical challenges awaiting them. Fortunately, the task is made easier by the simple fact that language technologies, and language itself, are intrinsically interesting and appealing to a wide audience. Several opening topics appear to work particularly well:

The holy grail: A long term challenge, mythologized in science fiction movies, is to build machines that understand human language. Current technologies that exhibit some basic level of natural language understanding include spoken dialogue systems, question answering systems, summarization systems, and machine translation systems. These can be demonstrated in class without too much difficulty. The Turing test is a linguistic test, easily understood by all students, and which helps the computer science students to see NLP in relation to the field of Artificial Intelligence. The evolution of programming languages has brought them closer to natural language, helping students see the essentially linguistic purpose of this central development in computer science. The corresponding holy grail in linguistics is full understanding of the human language faculty; writing programs and building machines surely informs this quest too.

The riches of language: It is easy to find examples of the creative richness of language in its myriad uses. However, linguists will understand that language contains hidden riches that can only be uncovered by careful analysis of large quantities of linguistically annotated data, work that benefits from suitable computational tools. Moreover, the computational needs for exploratory linguistic research often go beyond the capabilities of the current tools. Computer scientists will appreciate the cognate problem of extracting information from the web, and the economic riches associated with state-of-the-art text mining technologies.

Formal approaches to language: Computer science and linguistics have a shared history in the area of philosophical logic and formal language theory. Whether the language is natural or artificial, computer scientists and linguists use similar logical formalisms for investigating the formal semantics of languages, similar grammar formalisms for modeling the syntax of languages, and similar finite-state methods for manipulating text. Both rely on the recursive, compositional nature of natural and artificial languages.

3.5 First Assignment

The first coursework assignment can be a significant step forwards in helping students get to grips with the material, and is best given out early, perhaps even in week 1. We have found it advisable for this assignment to include both programming and linguistics content. One example is to ask students to carry out NP chunking of some data (e.g. a section of the Brown Corpus). The `nltk.RegexpParser` class is initialized with a set of chunking rules expressed in a simple, regular expression-oriented syntax, and the resulting chunk parser can be run over POS-tagged input text. Given a Gold Standard test set like the CoNLL-2000 data,[7] precision and recall of the chunk grammar can be easily determined. Thus, if students are given an existing, incomplete set of rules as their starting point, they just have to modify and test their rules.

There are distinctive outcomes for each set of students: linguistics students learn to write grammar fragments that respect the literal-minded needs of the computer, and also come to appreciate the noisiness of typical NLP corpora (including automatically annotated corpora like CoNLL-2000). Computer science students become more familiar with parts of speech and with typical syntactic structures in English. Both groups learn the importance of formal evaluation using precision and recall.

4 Interactive Demonstrations

4.1 Python Demonstrations

Python fosters a highly interactive style of teaching. It is quite natural to build up moderately complex programs in front of a class, with the less confident students transcribing it into a Python session on their laptop to satisfy themselves it works (but not necessarily understanding everything they enter first time), while the stronger students quickly grasp the theoretical concepts and algorithms. While both groups can be served by the same presentation, they tend to ask quite different questions. However, this is addressed by dividing them into smaller clusters and having teaching assistants visit them separately to discuss issues arising from the content.

The NLTK book contains many examples, and the instructor can present an interactive lecture that includes running these examples and experimenting with them in response to student questions. In early classes, the focus will probably be on learning Python. In later classes, the driver for such interactive lessons can be an externally-motivated empirical or theoretical question.

As a practical matter, it is important to consider low-level issues that may get in the way of students' ability to capture the material covered in interactive Python sessions. These include choice of appropriate font size for screen display, avoiding the problem of output scrolling the command out of view, and distributing a log of the instructor's interactive session for students to study in their own time.

4.2 NLTK Demonstrations

A significant fraction of any NLP syllabus covers fundamental data structures and algorithms. These are usually taught with the help of formal notations and complex diagrams. Large trees and charts are copied onto the board and edited in tedious slow motion, or laboriously prepared for presentation slides. It is more effective to use live demonstrations in which those diagrams are generated and updated automatically. NLTK provides interactive graphical user interfaces, making it possible to view program state and to study program execution step-by-step. Most NLTK components have a demonstration mode, and will perform an interesting task without requiring any special input from the user. It is even possible to make minor modifications to programs in response to "what if" questions. In this way, students learn the mechanics of NLP quickly, gain deeper insights into the data structures and algorithms, and acquire new problem-solving skills.

An example of a particularly effective set of demonstrations are those for shift-reduce and recursive descent parsing. These make the difference between the algorithms glaringly obvious. More importantly, students get a concrete sense of many issues that affect the design of algorithms for tasks like parsing. The partial analysis constructed by the recursive descent parser bobs up and down as it steps forward and backtracks, and students often go wide-eyed as the parser retraces its steps and does "dumb" things like expanding N to *man* when it has already tried the rule unsuccessfully (but is now trying to match a bare NP rather than an NP with a PP modifier). Linguistics students who are extremely

[7] `http://www.cnts.ua.ac.be/conll2000/chunking/`

knowledgeable about context-free grammars and thus understand the representations gain a new appreciation for just how naive an algorithm can be. This helps students grasp the need for techniques like dynamic programming and motivates them to learn how they can be used to solve such problems much more efficiently.

Another highly useful aspect of NLTK is the ability to define a context-free grammar using a simple format and to display tree structures graphically. This can be used to teach context-free grammars interactively, where the instructor and the students develop a grammar from scratch and check its coverage against a testbed of grammatical and ungrammatical sentences. Because it is so easy to modify the grammar and check its behavior, students readily participate and suggest various solutions. When the grammar produces an analysis for an ungrammatical sentence in the testbed, the tree structure can be displayed graphically and inspected to see what went wrong. Conversely, the parse chart can be inspected to see where the grammar failed on grammatical sentences.

NLTK's easy access to many corpora greatly facilitates classroom instruction. It is straightforward to pull in different sections of corpora and build programs in class for many different tasks. This not only makes it easier to experiment with ideas on the fly, but also allows students to replicate the exercises outside of class. Graphical displays that show the dispersion of terms throughout a text also give students excellent examples of how a few simple statistics collected from a corpus can provide useful and interesting views on a text—including seeing the frequency with which various characters appear in a novel. This can in turn be related to other resources like Google Trends, which shows the frequency with which a term has been referenced in news reports or been used in search terms over several years.

5 Exercises, Assignments and Projects

5.1 Exercises

Copious exercises are provided with the NLTK book; these have been graded for difficulty relative to the concepts covered in the preceding sections of the book. Exercises have the tremendous advantage of building on the NLTK infrastructure, both code and documentation. The exercises are intended to be suitable both for self-paced learning and in formally assigned coursework.

A mixed class of linguistics and computer science students will have a diverse range of programming experience, and students with no programming experience will typically have different aptitudes for programming (Barker and Unger, 1983; Caspersen et al., 2007). A course which forces all students to progress at the same rate will be too difficult for some, and too dull for others, and will risk alienating many students. Thus, course materials need to accommodate self-paced learning. An effective way to do this is to provide students with contexts in which they can test and extend their knowledge at their own rate.

One such context is provided by lecture or laboratory sessions in which students have a machine in front of them (or one between two), and where there is time to work through a series of exercises to consolidate what has just been taught from the front, or read from a chapter of the book. When this can be done at regular intervals, it is easier for students to know which part of the materials to re-read. It also encourages them to get into the habit of checking their understanding of a concept by writing code.

When exercises are graded for difficulty, it is easier for students to understand how much effort is expected, and whether they even have time to attempt an exercise. Graded exercises are also good for supporting self-evaluation. If a student takes 20 minutes to write a solution, they also need to have some idea of whether this was an appropriate amount of time.

The exercises are also highly adaptable. It is common for instructors to take them as a starting point in building homework assignments that are tailored to their own students. Some instructors prefer to include exercises that do not allow students to take advantage of built-in NLTK functionality, e.g. using a Python dictionary to count word frequencies in the Brown corpus rather than NLTK's `FreqDist` (see Figure 2). This is an important part of building facility with general text processing in Python, since eventually students will have to work outside of the NLTK sandbox. Nonetheless, students often use NLTK functionality as part of their solutions, e.g., for managing frequencies and distributions. Again,

```
nltk.FreqDist(nltk.corpus.brown.words())

fd = nltk.FreqDist()
for filename in corpus_files:
    text = open(filename).read()
    for w in nltk.wordpunct_tokenize(text):
        fd.inc(w)

counts = {}
for w in nltk.corpus.brown.words():
    if w not in counts:
        counts[w] = 0
    counts[w] += 1
```

Figure 2: Three Ways to Build up a Frequency Distribution of Words in the Brown Corpus

this flexibility is a good thing: students learn to work with resources they know how to use, and can branch out to new exercises from that basis. When course content includes discussion of Unix command line utilities for text processing, students can furthermore gain a better appreciation of the pros and cons of writing their own scripts versus using an appropriate Unix pipeline.

5.2 Assignments

NLTK supports assignments of varying difficulty and scope: experimenting with existing components to see what happens for different inputs or parameter settings; modifying existing components and creating systems using existing components; leveraging NLTK's extensible architecture by developing entirely new components; or employing NLTK's interfaces to other toolkits such as Weka (Witten and Frank, 2005) and Prover9 (McCune, 2008).

5.3 Projects

Group projects involving a mixture of linguists and computer science students have an initial appeal, assuming that each kind of student can learn from the other. However, there's a complex social dynamic in such groups, one effect of which is that the linguistics students may opt out of the programming aspects of the task, perhaps with view that their contribution would only hurt the chances of achieving a good overall project mark. It is difficult to mandate significant collaboration across disciplinary boundaries, with the more likely outcome being, for example, that a parser is developed by a computer science team member, then thrown over the wall to a linguist who will develop an appropriate grammar.

Instead, we believe that it is generally more productive in the context of a single-semester introductory course to have students work individually on their own projects. Distinct projects can be devised for students depending on their background, or students can be given a list of project topics,[8] and offered option of self-proposing other projects.

6 Conclusion

We have argued that the distinctive features of NLTK make it an apt vehicle for teaching NLP to mixed audiences of linguistic and computer science students. On the one hand, complete novices can quickly gain confidence in their ability to do interesting and useful things with language processing, while the transparency and consistency of the implementation also makes it easy for experienced programmers to learn about natural language and to explore more challenging tasks. The success of this recipe is borne out by the wide uptake of the toolkit, not only within tertiary education but more broadly by users who just want try their hand at NLP. We also have encouraging results in presenting NLTK in classrooms at the secondary level, thereby trying to inspire the computational linguists of the future!

Finally, we believe that NLTK has gained much by participating in the Open Source software movement, specifically from the infrastructure provided by `SourceForge.net` and from the invaluable contributions of a wide range of people, including many students.

7 Acknowledgments

We are grateful to the members of the NLTK community for their helpful feedback on the toolkit and their many contributions. We thank the anonymous reviewers for their feedback on an earlier version of this paper.

[8] `http://nltk.org/projects.html`

References

Jason Baldridge and Katrin Erk. 2008. Teaching computational linguistics to a large, diverse student body: courses, tools, and interdepartmental interaction. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*. Association for Computational Linguistics.

Ricky Barker and E. A. Unger. 1983. A predictor for success in an introductory programming class based upon abstract reasoning development. *ACM SIGCSE Bulletin*, 15:154–158.

Steven Bird and Edward Loper. 2004. NLTK: The Natural Language Toolkit. In *Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, pages 214–217. Association for Computational Linguistics.

Steven Bird, Ewan Klein, and Edward Loper. 2008. Natural Language Processing in Python. `http://nltk.org/book.html`.

Steven Bird. 2005. NLTK-Lite: Efficient scripting for natural language processing. In *4th International Conference on Natural Language Processing, Kanpur, India*, pages 1–8.

Steven Bird. 2006. NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, Sydney, Australia, July. Association for Computational Linguistics.

Steven Bird. 2008. Defining a core body of knowledge for the introductory computational linguistics curriculum. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*. Association for Computational Linguistics.

Charles C. Bonwell and James A. Eison. 1991. *Active Learning: Creating Excitement in the Classroom*. Washington, D.C.: Jossey-Bass.

Michael Caspersen, Kasper Larsen, and Jens Bennedsen. 2007. Mental models and programming aptitude. *SIGCSE Bulletin*, 39:206–210.

Marti Hearst. 2005. Teaching applied natural language processing: Triumphs and tribulations. In *Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 1–8, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Ewan Klein. 2006. Computational semantics in the Natural Language Toolkit. In *Proceedings of the Australasian Language Technology Workshop*, pages 26–33.

Elizabeth Liddy and Nancy McCracken. 2005. Hands-on NLP for an interdisciplinary audience. In *Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 62–68, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 62–69. Association for Computational Linguistics.

Edward Loper. 2004. NLTK: Building a pedagogical toolkit in Python. In *PyCon DC 2004*. Python Software Foundation.

Nitin Madnani and Bonnie Dorr. 2008. Combining open-source with research to re-engineer a hands-on introductory NLP course. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*. Association for Computational Linguistics.

Nitin Madnani. 2007. Getting started on natural language processing with Python. *ACM Crossroads*, 13(4).

Matplotlib. 2008. Matplotlib: Python 2D plotting library. `http://matplotlib.sourceforge.net/`.

William McCune. 2008. Prover9: Automated theorem prover for first-order and equational logic. `http://www.cs.unm.edu/~mccune/mace4/manual-examples.html`.

NumPy. 2008. NumPy: Scientific computing with Python. `http://numpy.scipy.org/`.

Stuart Robinson, Greg Aumann, and Steven Bird. 2007. Managing fieldwork data with Toolbox and the Natural Language Toolkit. *Language Documentation and Conservation*, 1:44–57.

Christine Shannon. 2003. Another breadth-first approach to CS I using Python. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 248–251. ACM.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.