NLTK: The Natural Language Toolkit

Steven Bird
Department of Computer Science
and Software Engineering
University of Melbourne
Victoria 3010, Australia
sb@csse.unimelb.edu.au

Edward Loper
Department of Computer
and Information Science
University of Pennsylvania
Philadelphia PA 19104-6389, USA
edloper@gradient.cis.upenn.edu

Abstract

The Natural Language Toolkit is a suite of program modules, data sets, tutorials and exercises, covering symbolic and statistical natural language processing. NLTK is written in Python and distributed under the GPL open source license. Over the past three years, NLTK has become popular in teaching and research. We describe the toolkit and report on its current state of development.

1 Introduction

The Natural Language Toolkit (NLTK) was developed in conjunction with a computational linguistics course at the University of Pennsylvania in 2001 (Loper and Bird, 2002). It was designed with three pedagogical applications in mind: assignments, demonstrations, and projects.

Assignments. NLTK supports assignments of varying difficulty and scope. In the simplest assignments, students experiment with existing components to perform a wide variety of NLP tasks. As students become more familiar with the toolkit, they can be asked to modify existing components, or to create complete systems out of existing components.

Demonstrations. NLTK's interactive graphical demonstrations have proven to be very useful for students learning NLP concepts. The demonstrations give a step-by-step execution of important algorithms, displaying the current state of key data structures. A screenshot of the chart parsing demonstration is shown in Figure 1.

Projects. NLTK provides students with a flexible framework for advanced projects. Typical projects might involve implementing a new algorithm, developing a new component, or implementing a new task.

We chose Python 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. The recently added generator syntax makes it easy to create interactive implementations of algorithms (Loper, 2004; Rossum, 2003a; Rossum, 2003b).

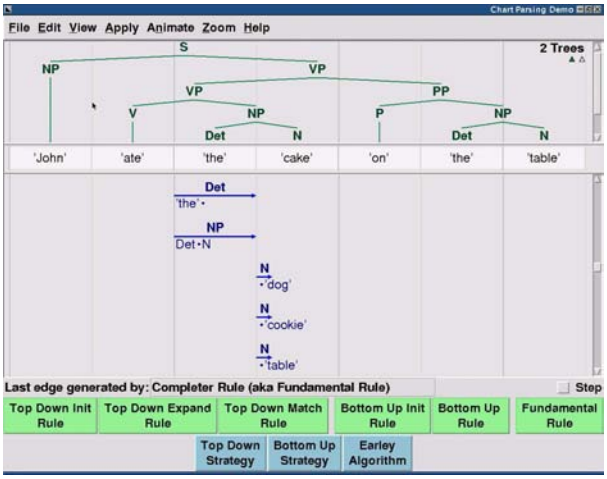

Figure 1: Interactive Chart Parsing Demonstration

2 Design

NLTK is implemented as a large collection of minimally interdependent modules, organized into a shallow hierarchy. A set of core modules defines basic data types that are used throughout the toolkit. The remaining modules are *task modules*, each devoted to an individual natural language processing task. For example, the `nltk.parser` module encompasses to the task of *parsing*, or deriving the syntactic structure of a sentence; and the `nltk.tokenizer` module is devoted to the task of *tokenizing*, or dividing a text into its constituent parts.

2.1 Tokens and other core data types

To maximize interoperability between modules, we use a single class to encode information about natural language texts – the `Token` class. Each `Token` instance represents a unit of text such as a word, sentence, or document, and is defined by a (partial) mapping from property names to values. For example, the `TEXT` property is used to encode a token's text content:[1]

```
>>> from nltk.token import *
>>> Token(TEXT="Hello World!")
<Hello World!>
```

The `TAG` property is used to encode a token's part-of-speech tag:

```
>>> Token(TEXT="python", TAG="NN")
<python/NN>
```

The `SUBTOKENS` property is used to store a tokenized text:

```
>>> from nltk.tokenizer import *
>>> tok = Token(TEXT="Hello World!")
>>> WhitespaceTokenizer().tokenize(tok)
>>> print tok['SUBTOKENS'])
[<Hello>, <World!>]
```

In a similar fashion, other language processing tasks such as word-sense disambiguation, chunking and parsing all add properties to the `Token` data structure.

In general, language processing tasks are formulated as annotations and transformations involving `Tokens`. In particular, each processing task takes a token and extends it to include new information. These modifications are typically *monotonic*; new information is added but existing information is not deleted or modified. Thus, tokens serve as a *blackboard*, where information about a piece of text is collated. This architecture contrasts with the more typical *pipeline* architecture where each processing task's output discards its input information. We chose the blackboard approach over the pipeline approach because it allows more flexibility when combining tasks into a single system.

In addition to the `Token` class and its derivatives, NLTK defines a variety of other data types. For instance, the `probability` module defines classes for probability distributions and statistical smoothing techniques; and the `cfg` module defines classes for encoding context free grammars and probabilistic context free grammars.

[1] Some code samples are specifi c to NLTK version 1.4.

2.2 The corpus module

Many language processing tasks must be developed and tested using annotated data sets or corpora. Several such corpora are distributed with NLTK, as listed in Table 1. The `corpus` module defines classes for reading and processing many of these corpora. The following code fragment illustrates how the Brown Corpus is accessed.

```
>>> from nltk.corpus import brown
>>> brown.groups()
['skill and hobbies', 'popular lore',
'humor', 'fiction: mystery', ...]
>>> brown.items('humor')
('cr01', 'cr02', 'cr03', 'cr04', 'cr05',
'cr06', 'cr07', 'cr08', 'cr09')
>>> brown.tokenize('cr01')
<[<It/pps>, <was/bedz>, <among/in>,
<these/dts>, <that/cs>, <Hinkle/np>,
<identified/vbd>, <a/at>, ...]>
```

A selection of 5% of the Penn Treebank corpus is included with NLTK, and it is accessed as follows:

```
>>> from nltk.corpus import treebank
>>> treebank.groups()
('raw', 'tagged', 'parsed', 'merged')
>>> treebank.items('parsed')
['wsj_0001.prd', 'wsj_0002.prd', ...]
>>> item = 'parsed/wsj_0001.prd'
>>> sentences = treebank.tokenize(item)
>>> for sent in sentences['SUBTOKENS']:
...     print sent.pp() # pretty-print
(S:
  (NP-SBJ:
    (NP: <Pierre> <Vinken>)
    (ADJP:
      (NP: <61> <years>)
      <old>
    )
    ...
```

2.3 Processing modules

Each language processing algorithm is implemented as a class. For example, the `ChartParser` and `RecursiveDescentParser` classes each define a single algorithm for parsing a text. We implement language processing algorithms using classes instead of functions for three reasons. First, all algorithm-specific options can be passed to the constructor, allowing a consistent interface for applying the algorithms. Second, a number of algorithms need to have their state initialized before they can be used. For example, the `NthOrderTagger` class

Corpus	Contents and Wordcount	Example Application
20 Newsgroups (selection)	3 newsgroups, 4000 posts, 780kw	text classifi cation
Brown Corpus	15 genres, 1.15Mw, tagged	training & testing taggers, text classifi cation
CoNLL 2000 Chunking Data	270kw, tagged and chunked	training & testing chunk parsers
Project Gutenberg (selection)	14 texts, 1.7Mw	text classifi cation, language modelling
NIST 1999 IEER (selection)	63kw, named-entity markup	training & testing named-entity recognizers
Levin Verb Index	3k verbs with Levin classes	parser development
Names Corpus	8k male & female names	text classifi cation
PP Attachment Corpus	28k prepositional phrases, tagged	parser development
Roget's Thesaurus	200kw, formatted text	word-sense disambiguation
SEMCOR	880kw, POS & sense tagged	word-sense disambiguation
SENSEVAL 2 Corpus	600kw, POS & sense tagged	word-sense disambiguation
Stopwords Corpus	2,400 stopwords for 11 lgs	text retrieval
Penn Treebank (sample)	40kw, tagged & parsed	parser development
Wordnet 1.7	180kw in a semantic network	WSD, NL understanding
Wordlist Corpus	960kw and 20k affi xes for 8 lgs	spell checking

Table 1: Corpora and Corpus Samples Distributed with NLTK

must be initialized by training on a tagged corpus before it can be used. Third, subclassing can be used to create specialized versions of a given algorithm.

Each processing module defines an *interface* for its task. Interface classes are distinguished by naming them with a trailing capital "`I`," such as `ParserI`. Each interface defines a single *action method* which performs the task defined by the interface. For example, the `ParserI` interface defines the `parse` method and the `Tokenizer` interface defines the `tokenize` method. When appropriate, an interface defines *extended action methods*, which provide variations on the basic action method. For example, the `ParserI` interface defines the `parse_n` method which finds at most n parses for a given sentence; and the `TokenizerI` interface defines the `xtokenize` method, which outputs an iterator over subtokens instead of a list of subtokens.

NLTK includes the following modules: `cfg`, `corpus`, `draw` (`cfg`, `chart`, `corpus`, `featurestruct`, `fsa`, `graph`, `plot`, `rdparser`, `srparser`, `tree`), `eval`, `featurestruct`, `parser` (`chart`, `chunk`, `probabilistic`), `probability`, `sense`, `set`, `stemmer` (`porter`), `tagger`, `test`, `token`, `tokenizer`, `tree`, and `util`. Please see the online documentation for details.

2.4 Documentation

Three different types of documentation are available. Tutorials explain how to use the toolkit, with detailed worked examples. The API documentation describes every module, interface, class, method, function, and variable in the toolkit. Technical reports explain and justify the toolkit's design and implementation. All are available from `http://nltk.sf.net/docs.html`.

3 Installing NLTK

NLTK is available from `nltk.sf.net`, and is packaged for easy installation under Unix, Mac OS X and Windows. The full distribution consists of four packages: the Python source code (`nltk`); the corpora (`nltk-data`); the documentation (`nltk-docs`); and third-party contributions (`nltk-contrib`). Before installing NLTK, it is necessary to install Python version 2.3 or later, available from `www.python.org`. Full installation instructions and a quick start guide are available from the NLTK homepage.

As soon as NLTK is installed, users can run the demonstrations. On Windows, the demonstrations can be run by double-clicking on their Python source files. Alternatively, from the Python interpreter, this can be done as follows:

```
>>> import nltk.draw.rdparser
>>> nltk.draw.rdparser.demo()
>>> nltk.draw.srparser.demo()
>>> nltk.draw.chart.demo()
```

4 Using and contributing to NLTK

NLTK has been used at the University of Pennsylvania since 2001, and has subsequently been adopted by several NLP courses at other universities, including those listed in Table 2.

Third party contributions to NLTK include: Brill tagger (Chris Maloof), hidden Markov model tagger (Trevor Cohn, Phil Blunsom), GPSG-style feature-based grammar and parser (Rob Speer, Bob Berwick), finite-state morphological analyzer (Carl de Marcken, Beracah Yankama, Bob Berwick), decision list and decision tree classifiers (Trevor Cohn), and Discourse Representation Theory implementation (Edward Ivanovic).

NLTK is an open source project, and we welcome any contributions. There are several ways to contribute: users can report bugs, suggest features, or contribute patches on Sourceforge; users can participate in discussions on the *NLTK-Devel* mailing list[2] or in the NLTK public forums; and users can submit their own NLTK-based projects for inclusion in the nltk_contrib directory. New code modules that are relevant, substantial, original and well-documented will be considered for inclusion in NLTK proper. All source code is distributed under the GNU General Public License, and all documentation is distributed under a Creative Commons non-commercial license. Thus, potential contributors can be confident that their work will remain freely available to all. Further information about contributing to NLTK is available at `http://nltk.sf.net/contrib.html`.

5 Conclusion

NLTK is a broad-coverage natural language toolkit that provides a simple, extensible, uniform framework for assignments, demonstrations and projects. It is thoroughly documented, easy to learn, and simple to use. NLTK is now widely used in research and teaching. Readers who would like to receive occasional announcements about NLTK are encouraged to sign up for the low-volume, moderated mailing list *NLTK-Announce*.[3]

6 Acknowledgements

We are indebted to our students and colleagues for feedback on the toolkit, and to many contributors listed on the NLTK website.

[2] `http://lists.sourceforge.net/lists/listinfo/nltk-devel`

[3] `http://lists.sourceforge.net/lists/listinfo/nltk-announce`

Graz University of Technology, Austria *Information Search and Retrieval*
Macquarie University, Australia *Intelligent Text Processing*
Massachusetts Institute of Technology, USA *Natural Language Processing*
National Autonomous University of Mexico, Mexico *Introduction to Natural Language Processing in Python*
Ohio State University, USA *Statistical Natural Language Processing*
University of Amsterdam, Netherlands *Language Processing and Information Access*
University of Colorado, USA *Natural Language Processing*
University of Edinburgh, UK *Introduction to Computational Linguistics*
University of Magdeburg, Germany *Natural Language Systems*
University of Malta, Malta *Natural Language Algorithms*
University of Melbourne, Australia *Human Language Technology*
University of Pennsylvania, USA *Introduction to Computational Linguistics*
University of Pittsburgh, USA *Artificial Intelligence Application Development*
Simon Fraser University, Canada *Computational Linguistics*

Table 2: University Courses using NLTK

References

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. Somerset, NJ: Association for Computational Linguistics. `http://arXiv.org/abs/cs/0205028`.

Edward Loper. 2004. NLTK: Building a pedagogical toolkit in Python. In *PyCon DC 2004*. Python Software Foundation. `http://www.python.org/pycon/dc2004/papers/`.

Guido Van Rossum. 2003a. *An Introduction to Python*. Network Theory Ltd.

Guido Van Rossum. 2003b. *The Python Language Reference*. Network Theory Ltd.