Natural Language Toolkit as a Tool for Education and Information Extraction

Tyler Davis

The Natural Language Toolkit is an open source suite of program modules, tutorials and problem sets implemented in Python that can be used for many natural language processing tasks in the domain of research and education (Loper and Bird, 2002). The toolkit has its roots in computational linguistics education and was designed with that purpose in mind alongside providing tools for simplifying natural language processing research tasks. NLTK has been foundational to the domain of natural language processing in the Python ecosystem and has inspired libraries such as NLTK-Lite, which seeks to resolve some of NLTK's shortcomings. As NLTK grew and developers to the project sought to expand and refine the toolkit's abilities, it introduced some significant overhead for programmers that the developers of NLTK-Lite decided to address (Bird, 2005). Essentially, it presents a simplified version of NLTK that addresses these issues. The Natural Language Toolkit has a wide-ranging application domain from instruction to research with some significant overlap between the two, lowering the barrier to entry for anyone interested in pursuing a natural language processing project.

The Natural Language Toolkit was designed primarily to alleviate many of the complications that arise out of teaching introductory computational linguistics courses. Traditionally, an instructor must rely on a multitude of programming languages for different aspects of computational linguistics, such as Prolog for text parsing, Perl for corpus processing, and a finite-state toolkit for morphological analysis (Loper and Bird, 2002). NLTK helps an instructor to avoid having to develop a lot of software infrastructure in order to introduce students to a practical programming component for assignments. It bridges these various components into one package for "researchers, developers, and educators" (Loper and Bird, 2002). It was developed in conjunction with a course taught at the University of Pennsylvania. Certainly it has since found its place in the natural language processing problem domain: in the U.S., it is used for coursework at 32 universities, and 25 countries use NLTK in their courses (Yao, 2019).

NLTK was implemented in Python for a few reasons: Python is an object-oriented scripting language, offers a shallow learning curve, is an interpreted language "suitable for rapid prototyping," is self-documenting code, and writing GUIs for it is fairly straightforward because it has an interface to the TK graphics toolkit (Loper and Bird, 2002). The toolkit was designed with some requirements in mind, the first of which being ease of use. Because its primary purpose is to allow students to concentrate on building NLP systems, "the more time students must spend learning to use the toolkit, the less useful it is" (Loper and Bird, 2002). Secondly, the designers wanted the toolkit to be consistent in its use of data structures and interfaces, which is arguably a requirement that the toolkit failed in later extensions and led to the creation of NLTK-Lite (Bird, 2005). The designers also wanted the toolkit to be easily extendable, accommodating new components whether those components replace or extend the existing functionality. The toolkit, its data structures, and its implementation should be carefully and thoroughly documented — nomenclature in the source code should be carefully chosen and consistently used. NLTK should structure the complexities of building natural language processing projects rather than hiding them: "Each class should be simple enough that a student could implement it by the time they finish an intro course in computational linguistics" (Loper and Bird, 2002). Finally, the designers wanted the toolkit to be modular: Interaction between components should be kept to a minimum using "simple, well-defined interfaces." A user should be able to use one module without worrying about any others, which enables both incremental learning and easy extension and modification of the toolkit. This is an element also addressed in the NLTK-Lite implementation of the toolkit. The toolkit is not intended to be comprehensive by any means nor was it intense to be highly optimized, although it should be usable to perform real tasks.

Each module in the Natural Language Toolkit defines a specific data structure or task. The core modules define basic data types and processing used throughout the toolkit. The `token` module provides basic classes for processing individual elements of text, which include words and sentences. The `tree` module defines data structures for representing tree structures over text, including syntax and morphological trees. The `probability` module implements classes that encode frequency and probability distributions with a variety of statistical smoothing techniques. The remaining core modules define data structures and interfaces for specific NLP tasks, including modules for parsing, tagging, finite state automata, type checking, visualization, and text classification. Parsing modules include a high-level `parser` module that generates trees representing text structure. A `chunkparser` module defines a sub-interface for parsers that identify "non-overlapping lingustic groups" such as base noun phrases (Loper and Bird, 2002). Abstract interfaces to these parsers are implemented across four modules, including `srparser`, which implements a shift-reduce parser; `chartparser`, which defines a flexible parser using a `chart` to "record hypotheses about syntactic constituents"; `pcfgparser`, which provides a number of different parsers for probabilistic grammars; and `rechunkparser`, which defines a regular-expression based implementation of the `chunkparser` interface.

The `tagger` module defines a standard interface with several implementations for giving each token of text some supplementary information about it, including but not limited to its part of speech and its WordNet synset tag. The module for finite state automata, `fsa`, defines data type for encoding finite state automata and an interface for creating such automata using regular expressions. A type checking module can be used to ensure that functions are given valid arguments, which reduces the amount of time that students must spend debugging at the cost of lower performance. Type checking must be invoked explicitly and has no performance penalty when disabled. Visualization modules define the graphical interfaces for seeing and manipulating data structures as well as the graphical tools needed to experiment with NLP tasks. Text classification provides interfaces for classifying text into categories, implemented by two modules. The first of which defines text based on a Naive Bayes assumption. The second defines the maximum entropy model for text classification and implements two algorithms for training the model: Generalized Iterative Scaling and Improved Iterative Scaling. `classifier.feature` provides encoding for information used to make decisions for a classifier task, while `classifier.featureselection` defines an interface for choosing which features are relevant to a given classification task. The toolkit also includes extensive documentation for explanation of use and how to extend the toolkit.

NLTK-Lite is a simplified version of NLTK used to support efficient scripting for working on natural language processing problems. It is quite similar to NLTK but it was developed to address some issues that re-engineering in NLTK 1.4 had brought to the forefront (Bird, 2005) and to streamline other processes. In the initial implementation of the original NLTK, it was unclear how to generalize tasks so that they could be applied independently of each other. Due to the nature of tasks such as stemming and tagging, tagging loses information when stemming is performed first and stemming must be able to skip over the tags when tagging is done first. If both are done independently the results must then be able to be aligned. Furthermore, “as task combinations multiply managing the data becomes extremely difficult” (Bird, 2005). NLTK 1.4 introduced a new architecture for tokens based on the native Python dictionary data type. This unified many different NLTK data types but came with significant overhead for programmers, with Bird noting: “It was clear that the re-engineering done in NLTK 1.4 mainly got in the way of efficient authoring of NLP scripts.” NLTK-Lite keeps the fundamental representations of objects as simple as possible (strings, tuples, trees). Streaming tasks are implemented as iterators rather than lists to limit memory usage and increase performance. The default pipeline processing paradigm in NLTK-Lite also leads to more transparent code. Taggers incorporate backoff by default, which enables more comprehensive POS tagging and fallback mechanisms. Additionally, method names are shorter and it is easier to contribute to the package because there is no requirement to support any special token architecture, as there is in NLTK.

Because NLTK has its roots in education, it performs very well in a pedagogical environment. NLTK provides extensive documentation for explanation of use and how to extend the toolkit, including tutorials designed to teach students how to use the toolkit. Each tutorial included in the toolkit focuses on a specific problem domain such as tagging, probabilistic systems, and text classification. It also includes an extensive visualization module, which defines graphical interfaces for seeing and manipulating data structures as well as graphical tools that can be used to experiment with natural language processing tasks. Loper and Bird note that this has huge implications for computational linguistics instruction. The graphical tools available in NLTK can be used in class demonstrations to help explain basic NLP concepts and algorithms. It can be used to show relevant data structures and step-by-step execution of algorithms. Both the structures and control flow can be modified during the demonstration in response to queries. Educators at University of Pennsylvania found that the experience of using NLTK for classwork was very positive for students and instructors. Students appreciated being able to take on interesting projects and being able to run NLP code on their computers at home. One problem instructors found was finding large clean corpora students could use for their assignments.

Beyond educational purposes, NLTK is also robust enough to have research applications. Since the original NLTK paper was written, NLTK has added some semantic analysis methods that can be used to predict emotional scores for different sentences. Those predictions can be compared with ground truth scores given by human evaluators. Semantic analysis aims to determine the attitude or emotional reaction of a speaker with respect to some topic, document, interaction, or event. Researcher Jiawei Yao of the Beijing University of Posts and Telecommunication used NLTK for this purpose using the following methodology: After importing modules, an input CSV file is loaded and its contents — text and ratings — are read into the program. Each text review is then processed through NLTK’s sentiment analysis function (`SentimentIntensityAnalyzer().polarity_scores(text)`) to retrieve a predicted score. All predicted scores from the entire dataset are then collected. The correlation between user-generated ratings and

NLTK-generated scores are then correlated. Finally these findings are visualized in R. Yao found that the number of predicted ratings for 4 and 5 (out of 5) were numerous — far more than any other predicted rating. This implies that this method of sentiment analysis is limited to a small amount of text due to the nature of NLTK's rather simplistic sentiment analysis. If a large number of sentences are handled under this method, it will reduce the accuracy of score prediction. NLTK's sentiment prediction system looks at words in isolation, giving points to positive terms and negative points to negative terms. These points are then summed. The order of words is ignored and other important information is lost, which further implies the limits of a unigram text model in sentiment analysis given that a unigram text model assumes words are generated independently. The author proposes a deep learning model for better understanding and prediction built using TensorFlow and TFlearn, which is a deep learning library built on TensorFlow.

As noted by Loper and Bird, NLTK is unique due to a combination of three factors. First, it is deliberately designed as courseware, giving educational goals priority. Second, the target audience is both linguistics and computer science, which makes it accessible and challenging at many levels of skill. Certainly it has been proven to work in a research context, although it does have its limits as noted in the semantic analysis study discussed above. Third, because the toolkit is implemented in an object-oriented language, it supports rapid prototyping and literate programming. The Natural Language Toolkit has carved out its place as a premier technology for learning programmatic natural language processing techniques and for many NLP research tasks and will likely hold that place as long as it is continuously extended and improved to keep pace with modern NLP techniques. It does have its limits at the forefront of research, but every technology has its place, and NLTK serves as a strong foundational tool for both researchers and students to build on as they pursue more advanced techniques and concepts.

References

Association for Computational Linguistics (2002) Loper, E & Bird, S. G. Philadelphia, PA, July 2002. Available at: https://arxiv.org/abs/cs/0205028 (Accessed: 3 Nov. 2021).

Bird, S. G. (2005). NLTK-Lite: Efficient Scripting for Natural Language Processing. Proceedings of the 4th International Conference on Natural Language Processing, pp.11-18. Allied Publishers. Available at: https://minerva-access.unimelb.edu.au/bitstream/handle/11343/34057/66445_00001453_01_icon-05.pdf?sequence=1 (Accessed: 11/3/2021).

Yao, J. (2019). "Automated Sentiment Analysis of Text Data with NLTK", Journal of Physics: Conference Series, Volume 1187, Issue 5. doi: https://doi.org/10.1088/1742-6596/1187/5/052020.