Brian Reinbold
brianjr3

NLTK Library in Python

Information systems and natural language processing (NLP) are broad disciplines that require numerous techniques for processing and analytics. Text processing can include word tokenization and lemmatization; however, one can go further by performing part-of-speech tagging, named entity recognition for semantic understanding, and syntactic tree parsing for extracting grammatical structure from a sentence. Analytical techniques can include classification for sentiment analysis. With all these techniques for text processing and analytics, one may think they would need many tools from different libraries to accomplish a simple task, but the *Natural Language Toolkit* (NLTK) in python provides a one-stop shop for all of your NLP needs.

NLTK provides a cohesive interface for text processing and analytical tools used throughout the entire NLP pipeline, including over 50 corpora and lexical resources. NLTK is not exclusively for the English language but can process other languages too. Although a short review cannot cover everything that NLTK is capable of, this review will try to cover the most relevant tools that students may find useful for finishing their projects. These tools include word tokenization and lemmatization, frequency distributions, and classification tasks like sentiment analysis.

Word Tokenization – Text Processing

The first step of any text processing pipeline is word tokenization, i.e., segmenting a sentence into words. On the surface, tokenization seems straightforward because words are often separated by spaces, however, there are some nuances that NLTK covers. For example, punctuation is not separated by spaces, and should a contraction like, “don’t” be one word or two words? Also, languages like Chinese and Japanese do not use spaces to separate words, making word tokenization a challenge. Thankfully, NLTK makes tokenization easy, and one simply needs to call the function, *word_tokenize,* on the target sentence to return a list of tokens[1].

Oftentimes the next step in text processing is to remove stop words. Stop words are frequently occurring words such as articles (e.g., *the, a, an*). These word generally do not add much value to understanding what a document is about. Since they likely appear in all documents, they are not useful features for categorizing documents. NLTK provides a list of commonly used English stop words. One can simply import them and remove them from their list of tokens. Also, the stop words list from NLTK can be further edited by the user to include additional stop words, or a user can remove some.

Finally, it may be worth considering an additional text processing step – stemming and lemmatization. A lemma is the root of a word with its inflective forms removed. Words often have stems and affixes attached to add additional meaning or to change the word’s function, but this can lead to many additional, infrequently occurring words to the vocabulary. For

[1] See https://www.nltk.org/howto/tokenize.html.

example, it may not be necessary to distinguish singular and plural forms of words, e.g., both "dog" and "dogs" being represented by the lemma, "dog," may be sufficient for most analytical tasks. Stemming is a crude yet simple way to convert a word into its lemma by simply removing affixes[2]. One can use the *PorterStemmer()* class in NLTK, which implements the Porter stemming algorithm.

If stemming is not sufficient, then one can use more complicated lemmatization algorithms that use context to disambiguate the most appropriate lemma. NLTK provides a lemmatizer that uses WordNet, an opensource English lexicon. Unfortunately, it requires the user to provide the part-of-speech for each word[3,4]. See table 1 and table 2 in the appendix for examples of the differences between the Porter stemmer and the WordNet lemmatizer.

Frequency Distributions

After processing text documents, the next step often is to calculate term frequencies. A few highlights in NLTK are the *FreqDist* object, *TextCollection* object, and the *ngram* function. The *FreqDist* object is very similar to the *Counter* dictionary from the *collections* library with some additional functionalities. It is useful for processing a single text document[5].

If there are multiple documents in the corpora, the *TextCollection* object makes it easy to calculate term frequencies and TF-IDFs. One simply passes an iteratable of documents[6].

Sometimes a bag-of-words representation is not enough, and it is useful to look at various n-grams in a text. The *ngram* function provides a convenient implementation. One simply passes a sequence of text and the n-gram degree (value of n), and the function outputs a list of tuples of all the n-grams in the sequence. There are also additional parameters to control padding[7].

Sentiment Analysis

Sentiment analysis analyzes how the author of a text *feels* about something (whether positive or negative). NLTK comes with a pre-trained sentiment classifier called VADER (**V**alence **A**ware **D**ictionary and s**E**ntiment **R**easoner)[8]. To use, just instantiate a *SentimentIntensityAnalyzer* class and pass a sentence to the method *polarity_score.* It returns a value between zero and one for a positive, neutral, and negative valence. These three values sum to one. Also, the method returns a metric ranging from -1 to 1, compound, that captures overall sentiment. Since VADER was trained on social media posts, it may not be ideal for sentences with longer, more formal structures. Still, it is easy to set up and receive immediate results, which can serve as a useful baseline. Thankfully, NLTK contains additional tools for training one's own classifier.

The *classify* module in NLTK comes with several implemented models, which include *NaiveBayesClassifier*, *ConditionalExponentialClassifier, DecisionTreeClassifier, MaxentClassifier, and*

---

[2] See https://www.nltk.org/howto/stem.html.
[3] See https://www.nltk.org/_modules/nltk/stem/wordnet.html.
[4] For information on WordNet, see https://wordnet.princeton.edu/.
[5] See https://tedboy.github.io/nlps/generated/generated/nltk.FreqDist.html.
[6] See https://tedboy.github.io/nlps/generated/generated/nltk.TextCollection.html.
[7] See https://tedboy.github.io/nlps/generated/generated/nltk.ngrams.html.
[8] See https://www.nltk.org/_modules/nltk/sentiment/vader.html.

WekaClassifier[9]. A user will have to label their text data, split the data into training and test sets, and train the model themselves[10]. A useful feature of NLTK is that it can easily interface with models from scikit-learn, one of the most popular machine learning libraries in python. Passing a scikit-learn model into *nltk.classify.SklearnClassifier()* will allow one to easily train additional classifiers within the NLTK framework[11].

Conclusion

The *Natural Language Toolkit* (NLTK) provides a one-stop shop for analyzing natural language data. It has a robust library for text processing, such as word tokenization, stemming, and lemmatization. It also provides several convenient interfaces for calculating term frequencies within corpora. Finally, NLTK makes it simple to implement a text sentiment classifier. NLTK comes with a pretrained classier, VADER, or one can train their own using on of several available models. A user can easily train a classifier from scikit-learn and interface it with the tools in NLTK. This review only scratches the surface of what can be done in NLTK, but there are many additional features to discover. If one is interested in processing text or working with NLP, learning the NLTK library is essential.

[9] See https://www.nltk.org/howto/classify.html.
[10] See https://www.nltk.org/howto/sentiment.html.
[11] See https://realpython.com/python-nltk-sentiment-analysis/ for a useful tutorial on how to implement a sentiment classifier in NLTK.

References

Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

Hutto, C.J. & Gilbert, E.E. (2014). *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

Jurafsky, Daniel, and Martin, James H. *Speech and Language Processing:* An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Third Edition Draft, 2022.

Appendix

Table 1: Examples of Stemming and Lemmatization

Word	Porter Stemmer	WordNet Lemmatizer
cat	cat	cat
cats	cat	cat
fox	fox	fox
foxes	fox	fox
agrees	agre	agree
agreements	agreement	agreement
is	is	be
are	are	be
am	am	be
was	wa	be
were	were	be
to be	to b	to be

Table 2: Examples of Stemming and Lemmatization for forms of “compute”

Word	Porter Stemmer	WordNet Lemmatizer
computed	comput	compute
computers	comput	computer
computations	comput	computation
computational	comput	computational
uncomputational	uncomput	uncomputational
computationally	comput	computationally
uncomputationally	uncomput	uncomputationally
computerizing	computer	computerize
computerization	computer	computerization