Crossroads
The ACM Student Magazine

Getting Started on Natural Language Processing with Python

by ***Nitin Madnani***

Motivation

The intent of this article is to introduce readers to the area of natural language processing, commonly referred to as NLP. However, rather than just describing the salient concepts of NLP, this article uses the Python programming language to illustrate them as well. For readers unfamiliar with Python, the article provides a number of references to learn how to program in Python.

Introduction

Natural Language Processing

The term *natural language processing* encompasses a broad set of techniques for automated generation, manipulation, and analysis of natural or human languages. Although most NLP techniques inherit largely from linguistics and artificial intelligence, they are also influenced by relatively new areas such as machine learning, computational statistics, and cognitive science.

Before we see some examples of NLP techniques, it will be useful to introduce some very basic terminology. Please note that as a side effect of keeping things simple, these definitions may not stand up to strict linguistic scrutiny.

- **Token**: Before any real processing can be done on the input text, it needs to be segmented into linguistic units such as words, punctuation, numbers, or alphanumerics. These units are known as tokens.
- **Sentence**: An ordered sequence of tokens.
- **Tokenization**: The process of splitting a sentence into its constituent tokens. For segmented languages such as English, the existence of whitespace makes tokenization relatively easy and uninteresting. However, for languages such as Chinese and Arabic, the task is more difficult since there are no explicit boundaries. Furthermore, almost all characters in such non-segmented languages can exist as one-character words by themselves, and can also join together to form multi-character words.
- **Corpus**: A body of text, usually containing a large number of sentences.
- **Part-of-speech (POS) tag**: A word can be classified into one or more lexical or part-of-speech categories such as nouns, verbs, adjectives, and articles, to name a few. A POS tag is a symbol representing such a lexical category, e.g., NN (noun), VB (verb), JJ (adjective), AT (article). One of the oldest and most commonly used tag sets is the Brown corpus tag set. We will discuss the Brown corpus in more detail below.
- **Parse tree**: A tree defined over a given sentence that represents the syntactic structure of the sentence as defined by formal grammar.

Now that we have introduced the basic terminology, let us look at some common NLP tasks:

- **POS tagging**: Given a sentence and a set of POS tags, a common language processing task is to automatically assign POS tags to each word in the sentence. For example, given the sentence, "The ball is red," the output of a POS tagger would be, "The/AT ball/NN is/VB red/JJ." State-of-the-art POS taggers [**9**] can achieve accuracy as high as 96%. Tagging text with parts-of-speech turns out to be extremely useful for more complicated NLP tasks such as parsing and machine translation, which are discussed below.
- **Computational morphology**: Natural languages consist of a very large number of words that are built upon basic building blocks known as morphemes (or stems), the smallest linguistic units possessing meaning. Computational morphology is concerned with the discovery and analysis of the internal structure of words using computers.
- **Parsing**: In the parsing task, a parser constructs the parse tree given a sentence. Some parsers assume the existence of a set of grammar rules in order to parse, but recent parsers are smart enough to deduce the parse trees directly from the given data using complex statistical models [**1**]. Most parsers also operate in a supervised setting and require the sentence to be POS-tagged before it can be parsed. Statistical parsing is an area of active research in NLP.
- **Machine translation (MT)**: In machine translation, the goal is to have the computer translate the given text in one natural language to fluent text in another language, without human interference. This is one of the most difficult tasks in NLP and has been tackled in a lot of different ways over the years. Almost all MT approaches use POS tagging and parsing as preliminary steps.

Python

The Python programming language is a dynamically-typed, object-oriented, interpreted language. Although its primary strength lies in the ease with which it allows a programmer to rapidly prototype a project, its powerful and mature set of standard libraries make it a great fit for large-scale production-level software engineering projects as well. Python has a very shallow learning curve and an excellent online tutorial [**11**].

Natural Language ToolKit (NLTK)

Although Python already has most of the functionality needed to perform simple NLP tasks, it is still not powerful enough for most standard NLP tasks. This is where the natural language toolkit (NLTK) comes in [**12**]. NLTK is a collection of modules and corpora, released under an open-source license, that allows students to learn and conduct research in NLP.

The most important advantage of using NLTK is that it is entirely self-contained. Not only does it provide convenient functions and wrappers that can be used as building blocks for common NLP tasks, it also provides raw and preprocessed versions of standard corpora used in NLP literature and courses.

Using NLTK

The NLTK Web site contains excellent documentation and tutorials for learning to use the toolkit [**13**]. It would be unfair to the authors, as well as to this publication, to simply reproduce their content. Instead, I introduce NLTK by showing how to perform three NLP tasks, in increasing order of difficulty. Each task is either an unsolved exercise from the NLTK tutorial or a variant thereof. Therefore, the solution

and analysis of each task represents original content written solely for this article.

NLTK Corpora

As mentioned earlier, NLTK ships with several useful text corpora that are used widely in the NLP research community. In this section, we look at three of these corpora that we will be using in our tasks below.

- **Brown corpus**: The Brown corpus of standard American English is considered to be the first general English corpus that could be used in computational linguistic processing tasks [**6**]. The corpus consists of one million words of American English texts printed in 1961. For the corpus to represent as general a sample of the English language as possible, 15 different genres were sampled, including fiction, news, and religious text. Subsequently, a POS-tagged version of the corpus was also created, with substantial manual effort.
- **Gutenberg corpus**: The Gutenberg corpus is a selection of 14 texts chosen from *Project Gutenberg*, the largest online collection of free e-books [**5**]. The corpus contains a total of 1.7 million words.
- **Stopwords corpus**: Besides regular content words, there is another class of words called stop words that perform important grammatical functions, but are unlikely to be interesting by themselves. These include prepositions, complementizers, and determiners. NLTK comes bundled with the Stopwords corpus, a list of 2400 stop words across 11 different languages (including English).

NLTK Naming Conventions

Before we begin using NLTK for our tasks, it is important to familiarize ourselves with the naming conventions used in the toolkit. The top-level package is called `nltk_lite` and we can refer to the included modules by using their fully qualified dotted names, e.g., `nltk_lite.corpora` and `nltk_lite.utilities`. The contents of any such module can then be imported into the top-level namespace by using the standard `from-import` Python construct.

Task 1: Exploring Corpora

NLTK is distributed with several NLP corpora, as mentioned previously. In this task we explore one such corpus.

Task: Use the NLTK `corpora` module to read the corpus `austen-persuasion.txt` , included in the Gutenberg corpus collection, and answer the following questions:

- How many total words does this corpus contain?
- How many unique words does this corpus contain?
- What are the counts for the ten most frequent words?

Besides the `corpora` module that allows us to access and explore the bundled corpora with ease, NLTK also provides the `probability` module that contains several useful classes and functions for the task of computing probability distributions. One such class is called `FreqDist`, and it keeps track of the sample frequencies in a distribution. Figure 1 shows how to use these two modules to perform the first task.

```
>>>  from nltk_lite.corpora import gutenberg          # import the gutenberg collection
>>>  print gutenberg.items                            # what corpora are in the collection ?
['austen-emma', 'austen-persuasion', 'austen-sense', 'bible-kjv', 'blake-poems', 'blake-songs',
'chesterton-ball', 'chesterton-brown', 'chesterton-thursday', 'milton-paradise',
'shakespeare-caesar', 'shakespeare-hamlet', 'shakespeare-macbeth', 'whitman-leaves']
>>>  from nltk_lite.probability import FreqDist       # import FreqDist class
>>>  fd = FreqDist()                                  # create frequency distribution object
>>>  for word in gutenberg.raw('austen-persuasion'):  # for each token in the relevant text
...   fd.inc(word)                                    # increment its counter
...                                                   # Done counting.
>>>  print fd.N()                      # total samples(words)
98171
>>>  print fd.B()                      # number of bins or unique samples(words)
6141
>>>  ss = fd.sorted_samples()       # Use sorted_samples() method to get frequency-sorted list of words
>>>  for word in ss[:10]:           # for the first 10 words in this sorted list ...
...    print word, fd.count(word)   # ... print word and its count. Task finished !
...
, 6739
the 3121
to 2775
. 2741
and 2739
of 2567
a 1529
in 1346
was 1330
; 1286
```

Figure 1. Exploring NLTK's bundled corpora.

Solution: Jane Austen's book *Persuasion* contains 98,171 total tokens and 6141 unique tokens. Out of these, the most common token is a comma, followed by the word *the*.

The last part of this task is the perfect segue for one of the most interesting empirical observations about word occurrences: If we were to take a large corpus, count up the number of times each word occurs in that corpus, and then list the words according to the number of occurrences (starting with the most frequent), we would be able to observe a direct relationship between the frequency of a word and its position in the list.

George Kingsley Zipf claimed this relationship could be expressed mathematically, i.e., for any given word, *f * r* is the same constant, where *f* is the frequency of that word and *r* is the rank, or the position of the word in the sorted list. So, for example, the 5th most frequent word should occur exactly two times more frequently than the 10th most frequent word. In NLP literature, this relationship is referred to as *Zipf's law*.

Even though the mathematical relationship prescribed by Zipf's law does not hold exactly, it is useful to describe how words are distributed in human languages - there are a few words that are very common, a few that occur with medium frequency, and a very large number of words that occur very rarely. It is simple to extend the last part of Task 1 and graphically visualize this relationship using NLTK, as shown in Figure 2.

```
>>> from nltk_lite.corpora
>>> from nltk_lite.probability import FreqDist
>>> from nltk_lite.draw.plot import Plot  # import the Plot function that we will use to plot the relationship

>>> fd = FreqDist()
>>> for text in gutenberg.items:          # for each text in the gutenberg collection ...
...   for word in gutenberg.raw(text):    # ... and for each token in the the current text ...
...       fd.inc(word)                    # ... increment its counter



>>> ss = fd.sorted_samples()              # Use sorted_samples() method to get frequency-sorted list of words
>>> points = []                           # Initialize an empty list that will hold our plotting points

>>> for index,word in enumerate(ss):             # for each word and its index in the sorted list ...
...   points.append((index+1, fd.count(word)))   # ... append the (rank, frequency) tuple to the list. We have to
...                                              # add 1 since python lists are zero-indexed.

>>> Plot(points, scale='log').mainloop()   # plot rank vs frequency on a log-log plot and show the plot
```

Figure 2. Using NLTK to plot Zipf's law.

The corresponding log-log plot, shown in Figure 3, clearly illustrates that the relationship does hold, to a large extent, for our corpus.

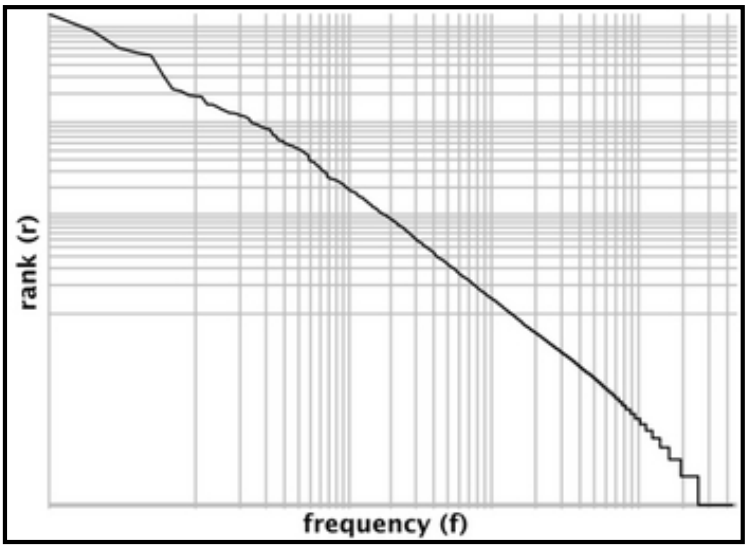

Figure 3. A plot showing that Zipf's law holds for the Gutenberg corpus.

Task 2: Predicting Words

Now that we have learned how to explore a corpus, let us define a task that can put such explorations to use.

Task: Train and build a word predictor, i.e., given a training corpus, write a program that can predict the word that follows a given word. Use this predictor to generate a random sentence of 20 words.

To build a word predictor, we first need to compute a distribution of two-word sequences over a training corpus, i.e., we need to keep count of the occurrences of a word given the previous word as a context for that word. Once we have computed such a distribution, we can use the input word to find a list of all possible words that followed it in the training corpus and then output a word at random from this list.

To generate a random sentence of 20 words, all we have to do is to start at the given word, predict the next word using this predictor, then the next, and so on until we get a total of 20 words. Figure 4 illustrates how to accomplish this easily using the modules provided by NLTK. Again we use Jane

Austen's *Persuasion* as the training corpus. To make things even easier, Python provides a `random` module that contains a function `choice` to pick an item at random from a list.

```
>>> from nltk_lite.corpora import gutenberg                    # import gutenberg collection
>>> from nltk_lite.probability import ConditionalFreqDist      # import ConditionalFreqDist class
>>> from random import choice                                  # import choice()
>>> cfd = ConditionalFreqDist()                    # create distribution object
>>> prev_word = None                               # initialize previous word to nothing
>>> for word in gutenberg.raw('austen-persuasion'):            # for each word in training corpus
...  cfd[prev_word].inc(word)                                   # count current word given previous
...  prev_word = word                          # set previous word to current word
...
>>> word = 'therefore'             # say that the given word is "therefore"
>>> i = 1                          # we are on the first word
>>> while i < 20:
...  print word,                     # print current word
...  lwords = cfd[word].samples()    # get all words that can possibly follow current word
...  follower = choice(lwords)       # pick one at random
...  word = follower                 # move on to the newly generated word
...  i += 1                          # increment word counter
...
therefore given , too wonderful and bruises she
supposed ; then appeared to want every anxiety , shewed himself between
```

Figure 4. Predicting words using NLTK.

Solution: The 20 word output sentence is, of course, not grammatical; but every two-word sequence is, because the training corpus that we used for estimating our conditional frequency distribution is grammatical, and because of the way that we estimated the conditional frequency distribution.

Note that for our task we used only the previous word as the context for our predictions. It is certainly possible to use the previous two, or even three words as the prediction context. A longer context would improve the performance of our word predictor and lead to more grammatical sentences.

Task 3: Discovering Part-Of-Speech Tags

NLTK comes with an excellent set of modules to allow us to train and build relatively sophisticated POS taggers. However, for this task, we will restrict ourselves to a simple analysis on an already tagged corpus included with NLTK.

Task: Tokenize the included Brown corpus and build one or more suitable data structures so that you can answer the following questions:

- What is the most frequent tag?
- Which word has the greatest number of distinct tags?
- What is the ratio of masculine to feminine pronouns?
- How many words are ambiguous, in the sense that they appear with at least two tags?

For this task, it is important to note that there are two versions of the Brown corpus that come bundled with NLTK: the first is the raw corpus that we used in the last two tasks, and the second is a tagged version wherein each token of each sentence of the corpus has been annotated with the correct POS tag. Each sentence in this version of a corpus is represented as a list of 2-tuples, each of the form (token, tag). For example, a sentence like, "The ball is green," from a tagged corpus, would be represented inside NLTK as the list [(*'The'*, *'AT'*), (*'ball'*, *'NN'*), (*'is'*, *'VB'*), (*'green'*, *'JJ'*)].

The Brown corpus comprises 15 different sections represented by the letters '*a*' through '*r*.' Each of the sections represents a different genre of text, and for certain NLP tasks not discussed in this article, this division proves very useful.

Given this information, all we should have to do is build the data structures to analyze this tagged corpus. Looking at the kinds of questions that we need to answer, it will be sufficient to build a frequency distribution over the POS tags and a conditional frequency distribution over the tags using the tokens as the context. Figure 5 shows the code.

```
>>> from nltk_lite.corpora import brown                  # import brown collection
>>> from nltk_lite.probability import FreqDist, ConditionalFreqDist  # import both classes
>>> fd = FreqDist()                  # create distribution object
>>> cfd = ConditionalFreqDist()      # create conditional distribution object
>>> for text in brown.items:                 # for each text in the brown collection ...
>>> for sentence in brown.tagged():        # for each tagged sentence in the brown corpus ...
...     for (token, tag) in sentence:       # ... and for each 2-tuple in tagged sentence ...
...         fd.inc(tag)                     # update count(tag)
...         cfd[token].inc(tag)             # update count(tag given token)
...
>>> fd.max() # most frequent tag is ...
'nn'
>>> wordbins = []  # a list to hold (numtags, word) tuples
>>> for token in cfd.conditions():              # for each token encountered ...
...   wordbins.append((cfd[token].B(), token)) # ... append (n(uniq tags for token),token) to list
...
>>> wordbins.sort(reverse=True) # sort tuples by first field but in descending order
>>> print wordbins[0]           # token with max no. of unique tags is ...
(12, 'that')

>>> male = ['he','his','him','himself']     # masculine pronouns
>>> female = ['she','hers','her','herself'] # feminine pronouns
>>> n_male = 0       # initialize counters
>>> n_female = 0
>>> for m in male:           # for each masculine pronoun ...
...   n_male += cfd[m].N()  # ... accumulate count
...
>>> for f in female:       # same for each feminine pronoun
...   n_female += cfd[f].N()
...
>>> print float(n_male)/n_female # calculate required ratio
3.25768844221

>>> n_ambiguous = 0                  # initialize counter
>>> for (ntags, token) in wordbins:  # for each (n(uniqtags), token) tuple
...   if ntags > 1:                   # if token appears with >1 tags
...       n_ambiguous += 1            # increment counter
...
>>> n_ambiguous     # number of ambiguous words is ...
8729
```

Figure 5. Analyzing tagged corpora using NLTK.

Solution: The most frequent POS tag in the Brown corpus is, unsurprisingly, the noun (NN). The word that has the greatest number of unique tags (12) is, in fact, the word *that*. There are more than three times as many masculine pronouns in the corpus as feminine pronouns and, finally, there are as many as 8700 words in the corpus that can be deemed ambiguous - a number that should indicate the difficulty of the POS-tagging task.

Task 4: Word Association

The task of free word association is a very common one when it comes to psycholinguistics, especially in the context of lexical retrieval, wherein human subjects respond more readily to a word if it follows another highly associated word, as opposed to a completely unrelated word. The instructions for

performing the association are fairly straightforward: the subject is asked for the word that immediately comes to mind upon hearing a particular word.

Task: Use a large POS-tagged text corpus to perform free word association. You may ignore function words and assume that the words to be associated are always nouns.

For this task, we will use the concept of word co-occurrences, i.e., counting the number of times words occur in close proximity with each other, and then using these counts to estimate the degree of association. For each token in each sentence, we will look at all following tokens that lie within a fixed window and count their occurrences in this context using a conditional frequency distribution. Figure 6 shows how we accomplish this using Python and NLTK with a window size of 5 and the POS-tagged version of the Brown corpus.

```
>>> from nltk_lite.corpora import brown, stopwords                  # import the brown and stopwords corpora
>>> from nltk_lite.probability import FreqDist, ConditionalFreqDist
>>> fd = FreqDist()
>>> cfd = ConditionalFreqDist()
>>> stopwords_list = list(stopwords.raw('english')) # get a list of all English stop words

>>> def is_noun(tag):  # define a function that return true if a given tag is a noun according to the Brown tagset
...   return tag in ['nn','nns','nn$','nn-tl','nn+bez','nn+hvz','nns$','np','np$','np+bez','nps','nps$','nr','nrs','nr$']
...

>>> for sentence in brown.tagged():                    # for each tagged sentence ...
...  for (index, tagtuple) in enumerate(sentence):    # ... get the 2-tuple and the index
...   (token, tag) = tagtuple                         # get the tag and the token from the 2-tuple
...   token = token.lower()                           # lowercase the token to decrease ambiguity
...   if token not in stopwords_list and is_noun(tag): # if the token is a content word and a noun ...
...       window = sentence[index+1:index+5]          # ... get the window of size 5 that follows this token
...       for (window_token, window_tag) in window:   # get each token and tag in the window
...           window_token = window_token.lower()     # lowercase the token from the window
...           if window_token not in stopwords_list and is_noun(window_tag):   # if we have a noun ...
...               cfd[token].inc(window_token)                                 # ... update contextual count
...

>>> # OK. We are done ! Let's start associating !
...
>>> print cfd['bread'].max() # what is the word that's most associated with bread ?
butter
>>> print cfd['life'].max()
death
>>> print cfd['man'].max()
woman
>>> print cfd['woman'].max()
world
>>> print cfd['boy'].max()
girl
>>> print cfd['girl'].max()
trouble
>>> print cfd['male'].max()
female
>>> print cfd['female'].max()
figure
>>> print cfd['doctor'].max()
bills
>>> print cfd['road'].max()
block
```

Figure 6. Performing free word association using NLTK.

Solution: The word associator that we have built seems to work surprisingly well, especially when compared to the minimal amount of effort that was required. (In fact, in the context of folk psychology, our associator would almost seem to have a personality, albeit a pessimistic and misogynistic one).

The results of this task should be a clear indication of the usefulness of corpus linguistics in general. As a further exercise, the association task can be easily extended in sophistication by utilizing parsed corpora and using information-theoretic measures of association [3].

Discussion

Although this article used Python and NLTK to provide an introduction to basic natural language processing, it is important to note that there are other NLP frameworks used by the NLP academic and industrial community. A popular example is GATE (general architecture for text engineering), developed by the NLP research group at the University of Sheffield [4]. GATE is built on Java and provides, besides the framework, a general architecture that describes how language processing components connect to each other, as well as a graphical environment. GATE is freely available and is primarily used for text mining and information extraction.

Every programming language and framework has its own strengths and weaknesses. For this article, we chose to use Python because it possesses a number of advantages over the other programming languages, such as high readability, an easy to use object-oriented paradigm, easy extensibility, strong Unicode support, and a powerful standard library. It is also extremely robust and efficient, and has been used in complex and large-scale NLP projects such as a state-of-the-art machine translation decoder [2].

Conclusions

Natural language processing is a very active field of research and attracts many graduate students every year. It allows for a coherent study of human language from the vantage points of several disciplines: linguistics, psychology, computer science, and mathematics.

Another, perhaps more important, reason for choosing NLP as an area of graduate study, is the sheer number of very interesting problems with well-established constraints, but no general solutions. For example, the original problem of machine translation that spurred the growth of the field remains, even after two decades of intriguing and active research, one of the hardest problems to solve.

There are several other cutting-edge areas in NLP that currently draw a large amount of research activity. It would be informative to discuss a few of them here:

- **Syntax-based machine translation**: For the past decade or so, most of the research in machine translation has focused on using statistical methods on very large corpora to learn translations of words and phrases. However, more and more researchers are starting to incorporate syntax into such methods [10].
- **Automatic multi-document text summarization**: There are a large number of efforts underway to use computers to automatically generate coherent and informative summaries for a cluster of related documents [8]. This task is considerably more difficult compared to generating a summary for a single document, because there may be redundant information present across multiple documents.
- **Computational parsing**: Although the problem of using probabilistic models to automatically generate syntactic structures for a given input text has been around for a long time, there are still significant improvements to be made. The most challenging task is to be able to parse, with reasonable accuracy, languages that exhibit very different linguistic properties when compared to English, such as Chinese and Arabic [7].

Python and the natural language toolkit (NLTK) allow any programmer to get acquainted with NLP tasks easily without having to spend too much time on gathering resources. This article is intended to make this task even easier by providing working examples and references for anyone interested in learning about NLP.

References

1

Bikel, Dan. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models.* PhD Thesis. 2004. <**http://www.cis.upenn.edu/~dbikel/papers/thesis.pdf**>

2

Chiang, David. *A hierarchical phrase-based model for statistical machine translation.* Proceedings of ACL. 2005.

3

Church, Kenneth W. and Hanks, Patrick. *Word association norms, mutual information, and lexicography.* Computational Linguistics, 16(1). 1990.

4

Cunningham, H., Maynard D., Bontcheva K. and Tablan V. *GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications*. Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). 2002.

5

Hart, Michael and Newby, Gregory. *Project Gutenberg.* <**http://www.gutenberg.org/wiki/Main_Page**>

6

Kucera, H. and Francis, W. N. *Computational Analysis of Present-Day American English.* Brown University Press, Providence, RI. 1967.

7

Levy, Roger and Manning, Christopher D. *Is it harder to parse Chinese, or the Chinese Treebank ?* Proceedings of ACL. 2003.

8

Radev, Dragomir R. and McKeown, Kathy. *Generating natural language summaries from multiple on-line sources.* Computational Linguistics. 24:469-500. 1999.

9

Ratnaparkhi, Adwait. *A Maximum Entropy Part-Of-Speech Tagger.* Proceedings of Empirical Methods on Natural Language Processing. 1996.

10

Wu, Dekai and Chiang, David. *Syntax and Structure in Statistical Translation.* Workshop at HTL-NAACL 2007.

11

The Official Python Tutorial. <**http://docs.python.org/tut/tut.html**>

12

Natural Language Toolkit. **http://nltk.sourceforge.net**>

13

NLTK Tutorial. <**http://nltk.sourceforge.net/lite/doc/en/**>

Biography

Nitin Madnani is a PhD student in the **Department of Computer Science** at the **University of Maryland, College Park**. He works as a graduate research assistant with the **Institute for Advanced Computer Studies** and works in the area of **statistical natural language processing**, specifically machine translation and text summarization. His programming language of choice for all tasks, big or small, is Python.