INTRODUCTION TO TEXT ANALYSIS

With Python and the Natural Language Toolkit

Matthew Menzenski

March 6, 2015

Digital Jumpstart Workshop
Institute for Digital Research in the Humanities
University of Kansas

Introduction

The Natural Language Toolkit

Tokenization and text preprocessing

Collocations

HTML and Concordances

Frequencies and Stop Words

Plots

Searches

Conclusions

CODE AND LICENSE

All Python code used in this workshop (as well as the LaTeX code for this presentation) is available at

`http://www.menzenski.com/nltk/`

INTRODUCTION

WHAT IS PYTHON?

Python is a programming language that is...

- high-level
- human-readable
- interpreted, not compiled
- object-oriented
- very well-suited to text analysis and natural language processing

WHAT IS THE NATURAL LANGUAGE TOOLKIT?

The Natural Language Toolkit (NLTK)

- contains many modules designed for natural language processing
- a powerful add-on to Python

WHAT ARE THE GOALS OF THIS WORKSHOP?

By end of today, you will be able to:

- Read, write, and understand basic Python syntax
- Run an interactive Python session from the command line
- Fetch text from the internet and manipulate it in Python
- Use many of the basic functions included in the NLTK
- Seek out, find, and utilize more complex Python syntax

PYTHON IS FAMOUSLY HUMAN-READABLE

Python

```
for line in open(”file.txt”):
    for word in line.split():
        if word.endswith(”ing”):
            print word
```

Perl

```
while (<>) {
    foreach my $word (split) {
        if ($word =~ /ing$/) {
            print ”\$word\n”;
} }
}
```

(Perl is another programming language used in text analysis.)

- Which of the two programs above is easier to read?
- These two programs do the same thing (what?)

WHAT DOES ‘HUMAN-READABLE’ MEAN?

Our Python program

```
for line in open(”file.txt”):
    for word in line.split():
        if word.endswith(”ing”):
            print word
```

How to read this Python program

- for each line in the text file `file.txt`
- for each word in the line (split into a list of words)
- if the word ends with `-ing`
- print the word

SOME IMPORTANT DEFINITIONS

Functions

A function is a way of packaging and reusing program code.

```
def repeat(message):
    return message + message

monty = ”Monty Python”

repeat(monty)

#.. ”Monty Python Monty Python”
```

In line 1 we define a function `repeat` that takes one argument, `message`.

SOME IMPORTANT DEFINITIONS

Functions

A function is a way of packaging and reusing program code.

```
def repeat(message):
    return message + message

monty = ”Monty Python”

repeat(monty)
#.. ”Monty Python Monty Python”
```

When called, this function returns that argument doubled.

SOME IMPORTANT DEFINITIONS

Functions

A function is a way of packaging and reusing program code.

```
def repeat(message):
    return message + message

monty = ”Monty Python”

repeat(monty)
#.. ”Monty Python Monty Python”
```

We define a variable `monty` and then call the function with `monty` as its argument.

SOME IMPORTANT DEFINITIONS

Functions

A function is a way of packaging and reusing program code.

```
def repeat(message):
    return message + message

monty = ”Monty Python”

repeat(monty)

#.. ”Monty Python Monty Python”
```

We could also skip that definition and simply call `repeat(”Monty Python”)`

DATA TYPES IN PYTHON

A number stores a numeric value.

```
variable_1 = 10
```

A string is a sequence of characters.

```
variable_2 = ”John Smith” # or ’John Smith’
```

A list is an ordered sequence of items.

```
variable_3 = [10, ”John Smith”, [’another’, ’list’]]
```

A tuple is like a list, but immutable.

```
variable_4 = (10, ”John Smith”)
```

A dictionary contains key-value pairs.

```
variable_5 = {”name”: ”John Smith”, ”age”: 45}
```

THE NATURAL LANGUAGE TOOLKIT

PRELIMINARIES

Opening a new Python console

```
import nltk
import re
from urllib import urlopen
```

- Call import statements at the beginning.
- `re` is the regular expressions module. We won’t need it until later.

GETTING DATA FROM A PLAIN-TEXT FILE ON THE INTERNET

```
url = ”http://menzenski.pythonanywhere.com/text/fathers_and_sons.txt”
# url = ”http://www.gutenberg.org/cache/epub/30723/pg30723.txt”

raw = urlopen(url).read()

type(raw)
#.. <type ”str”>

len(raw)
#.. 448367

raw[:60]
#.. ”The Project Gutenberg eBook, Fathers and Children, by Ivan S”
```

GETTING DATA FROM A PLAIN-TEXT FILE ON THE INTERNET

Create and define the variable `url`

```
url = ”http://menzenski.pythonanywhere.com/text/fathers_and_sons.txt”
# url = ”http://www.gutenberg.org/cache/epub/30723/pg30723.txt”
```

Our source text is the 1862 Russian novel Fathers and Sons (also translated as Fathers and Children) by Ivan Sergeevich Turgenev.

GETTING DATA FROM A PLAIN-TEXT FILE ON THE INTERNET

Open the url and read its contents into the variable `raw`

```
raw = urlopen(url).read()
## or as two separate steps:
# webpage = urlopen(url)
# raw = webpage.read()
```

Query the `type` of data stored in the variable `raw`

```
type(raw)
#.. <type ”str”>
```

GETTING DATA FROM A PLAIN-TEXT FILE ON THE INTERNET

Query the length of our text file

```
len(raw)
#.. 448367
```

The string `raw` contains 448,367 characters.

Display the first characters of `raw`

```
raw[:60] # from the beginning to the sixtieth character
#.. ”The Project Gutenberg eBook, Fathers and Children, by Ivan S”
# raw[10:60] ## from the tenth character to the sixtieth
# raw[40000:] ## from the 40,000th character to the end
```

TOKENIZATION AND TEXT PREPROCESSING

WHAT IS A ‘TOKEN’?

A token is a technical term for a sequence of characters which we want to treat as a group. ”Token” is largely synonymous with ”word”, but there are differences.

Some example tokens

- his
- sesquipedalian
- didn’t
- ’s
- ;
- state-of-the-art

BACK TO “FATHERS AND SONS”

We have the novel’s text (in `raw`), but it’s not very useful as a single long string. Let’s break it down into tokens.

```
tokens = nltk.word_tokenize(raw)

type(tokens)
#.. <type ”list”>

len(tokens)
#.. 91736

tokens[:10]
#.. [”The”, ”Project”, ”Gutenberg”, ”eBook”, ”,”, ”Fathers”,
#.. ”and”, ”Children”, ”,”, ”by”]
```

TOKENIZING "FATHERS AND SONS"

```
tokens = nltk.word_tokenize(raw)
```

`nltk.word_tokenize()` is the NLTK's default tokenizer method. There are others (or you can define your own!), but `nltk.word_tokenize()` is appropriate in most situations.

You can compare the NLTK's various tokenizers at `http://text-processing.com/demo/tokenize/`.

TOKENIZING “FATHERS AND SONS”

```
type(tokens)
#.. <type ”list”>
```

```
len(tokens)
#.. 91376
```

Our variable `tokens` is a list, which is comprised of 91,736 items.

Does the number of tokens equal the number of words? Why or why not?

TOKENIZING "FATHERS AND SONS"

The variable `tokens` is a list of strings.

```
tokens[:10]
#.. ["The", "Project", "Gutenberg", "eBook", ",", "Fathers",
#.. "and", "Children", ",", "by"]
```

We used `raw[:10]` to print the first ten characters of a string.

We can use the same syntax to print the first ten items of a list: `tokens[:10]`.

COLLOCATIONS

MORE SOPHISTICATED TEXT ANALYSIS WITH THE NLTK

```
text = nltk.Text(tokens)
```

Calling the `nltk.Text()` module on our list `tokens` defines an ‘NLTK text’, on which we can use more specialized methods.

```
type(text)
#.. <class ”nltk.text.Text”>
```

We’re not dealing in built-in data types now. The object `text` is a custom class of object in the NLTK.

COLLOCATIONS

A collocation is a sequence of words which co-occur unusually often.

```
text.collocations()
#.. Building collocations list
#.. Nikolai Petrovitch; Pavel Petrovitch; Anna Sergyevna;
#.. Vassily Ivanovitch; Madame Odintsov; Project Gutenberg-tm;
#.. Arina Vlasyevna; Project Gutenberg; Pavel Petrovitch.;
#.. Literary Archive; Gutenberg-tm electronic; Yevgeny
#.. Vassilyitch; Matvy Ilyitch; young men; Gutenberg
#.. Literary; every one; Archive Foundation;
#.. electronic works; old man; Father Alexey
```

What sorts of word combinations turned up in the list of collocations? Why?

CLEANING UP THE TEXT

Why did “Project Gutenberg” appear as a collocation?

Each Project Gutenberg text file contains a header and footer with information about that text and about Project Gutenberg. Those two words appear in the header/footer often enough that they’re considered a collocation.

CLEANING UP THE TEXT

```
raw.find(”CHAPTER I”)
#.. 1872

raw.rfind(”***END OF THE PROJECT GUTENBERG”)
#.. 429664

raw = raw[1872:429664]

raw.find(”CHAPTER I”)
#.. 0
```

CLEANING UP THE TEXT

```
raw.find(”CHAPTER I”)
#.. 1872
```

The string method `find()` starts from the beginning of a string and returns the position in the string at which the search term `”CHAPTER I”` begins.

```
raw.rfind(”***END OF THE PROJECT GUTENBERG”)
#.. 429664
```

The method `rfind()` does the same thing, but searches backward from the end of the string towards the beginning. (This saves time when searching long strings.)

CLEANING UP THE TEXT

```
raw = raw[1872:429664]
```

Now that we know where the text proper begins and ends, we can redefine `raw` to exclude the header and footer.

```
raw.find(”CHAPTER I”)
#.. 0
```

We verify that `”CHAPTER I”` is the first position in the string. (In Python, as in many programming languages, the first position in a sequence is 0, not 1.)

HTML AND CONCORDANCES

PULLING TEXT FROM AN HTML FILE

We got the text of Fathers and Sons from the internet, but it was already in plain text. What about reading a more typical web page into the NLTK?

Web pages are more than just text

- The source code of most pages contains markup in addition to actual content.
- We'll want to strip these formatting commands before working with the text.
- You might see click here!, but the HTML might actually contain `<a href=”https://www.google.com”>click here!</a>`
- Fortunately removing HTML markup is straightforward in the NLTK.

PULLING TEXT FROM AN HTML FILE

```
web_url="http://www.menzenski.pythonanywhere.com/text/blog_post.html"
web_html = urlopen(web_url).read()
web_html[:60]
#.. '<DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">\n<html><head>\n'
```

- Just like with “Fathers and Sons”, we defined a `url`, opened it, and read it into a variable (`web_html`).
- But now there’s all sorts of markup to deal with.

PULLING TEXT FROM AN HTML FILE

```
web_raw = nltk.clean_html(web_html)
web_tokens = nltk.word_tokenize(web_raw)
web_tokens[:10]
#.. [”This”, ”is”, ”a”, ”web”, ”page”, ”!”,
#.. ”This”, ”is”, ”a”, ”heading”]
```

With a little trial and error we can trim the list of tokens to just the content.

```
web_tokens = web_tokens[10:410]
```

FINDING CONCORDANCES IN “FATHERS AND SONS”

A concordance search returns the contexts in which a given word appears.

```
text.concordance(”boy”)
#.. Displaying 10 of 10 matches:
#.. ? Get things ready , my good boy : look sharp.’ Piotr , who a
#.. frogs , ’ observed Vaska , a boy of seven , with a head as wh
#..  no earthly use. He ’s not a boy , you know ; it ’s time to t
#.. drawing himself up. ’Unhappy boy ! ’ wailed Pavel Petrovitch
#.. ral. ’I advise you , my dear boy , to go and call on the Gove
#.. n a low voice. ’Because , my boy , as far as my observations
#.. seen ups and downs , my dear boy ; she ’s known what it is to
#.. : ’You ’re still a fool , my boy , I see. Sitnikovs are indis
#.. , indicating a short-cropped boy , who had come in with him i
#..  ’I am not now the conceited boy I was when I came here , ’ A
```

FREQUENCIES AND STOP WORDS

COUNTING FREQUENCIES

Let’s find the fifty most frequent tokens in “Fathers and Sons”:

```
fdist = nltk.FreqDist(text)
fdist
#.. <FreqDist with 10149 samples and 91736 outcomes>
vocab = fdist.keys()
vocab[:50]
#.. [”,”, ”the”, ”to”, ”and”, ”a”, ”of”, ”’”, ”in”, ”;”, ”he”,
#.. ”you”, ”his”, ”I”, ”was”, ”?”, ”with”, ”’s”, ”that”,
#.. ”not”, ”her”, ”it”, ”at”, ”...”, ”for”, ”on”, ”!”, ”is”,
#.. ”had”, ”him”, ”Bazarov”, ”but”, ”as”, ”she”, ”–”, ”be”,
#.. ”have”, ”n’t”, ”Arkady”, ”all”, ”Petrovitch”, ”are”, ”me”,
#.. ”do”, ”from”, ”up”, ”one”, ”I”, ”an”, ”my”, ”He”, ]
```

COUNTING FREQUENCIES

What’s wrong with our list?

```
vocab[:50]
#.. [”,”, ”the”, ”to”, ”and”, ”a”, ”of”, ”’”, ”in”, ”;”, ”he”,
#.. ”you”, ”his”, ”I”, ”was”, ”?”, ”with”, ”’s”, ”that”,
#.. ”not”, ”her”, ”it”, ”at”, ”...”, ”for”, ”on”, ”!”, ”is”,
#.. ”had”, ”him”, ”Bazarov”, ”but”, ”as”, ”she”, ”–”, ”be”,
#.. ”have”, ”n’t”, ”Arkady”, ”all”, ”Petrovitch”, ”are”, ”me”,
#.. ”do”, ”from”, ”up”, ”one”, ”I”, ”an”, ”my”, ”He”, ]
```

- With three exceptions, these tokens would be the most frequent in any text.
- How to find those tokens which are uniquely common in “Fathers and Sons”?

A BETTER LIST OF FREQUENT WORDS

One thing we can do is strip the punctuation:

```
# import re ## if you haven't already
clean_text = ["".join(
                 re.split("[.,;:!?‘’-], word")) for word in text]
fdist2 = nltk.FreqDist(clean_text)
vocab2 = fdist2.keys()
vocab2[:50]
#.. ["", "the", "to", "and", "a", "of", "in", "you", "I",
#.. "he", "his", "was", "with", "s", "that", "her", "not",
#.. "it", "at", "him", "Bazarov", "for", "on", "is", "had",
#.. "but", "as", "she", "Arkady", "Petrovitch", "be", "have",
#.. "me", "nt", "all", "are", "up", "do", "one", "from", "He",
#.. "my", "an", "The", "by", "You", "no", "your", "said",
#.. "what"]
```

A BETTER LIST OF FREQUENT WORDS

We could also convert all words to lowercase (‘he’ and ‘He’ should count as one word):

```
lower_text = [word.lower() for word in text]
fdist3 = nltk.FreqDist(lower_text)
vocab3 = fdist3.keys()
vocab3[:50]
#.. [”,”, ”the”, ”to”, ”and”, ”a”, ”of”, ”’”, ”he”, ”in”,
#.. ”you”, ”;”, ”his”, ”i”, ”was”, ”?”, ”with”, ”that”, ”’s”,
#.. ”not”, ”her”, ”it”, ”at”, ”but”, ”she”, ”...”, ”for”,
#.. ”on”, ”is”, ”!”, ”had”, ”him”, ”bazarov”, ”as”, ”–”,
#.. ”be”, ”have”, ”n’t”, ”arkady”, ”all”, ”petrovitch”, ”do”,
#.. ”are”, ”me”, ”one”, ”from”, ”what”, ”up”, ”my”, ”by”,
#.. ”an”]
```

A BETTER LIST OF FREQUENT WORDS

Finally, we could remove stop words.

- Stop words are words like ‘the’, ‘to’, ‘by’, and ‘also’ that have little semantic content.
- We usually want to remove these words from a text before further processing.
- Stop words are highly frequent in most texts, so their presence doesn’t tell us much about any specific text.

```
from nltk.corpus import stopwords
stopwords = stopwords.words(”english”)
```

A BETTER LIST OF FREQUENT WORDS

```
content = [word for word in text if word.lower() not in stopwords]
fdist4 = nltk.FreqDist(content)
content_vocab = fdist4.keys()
content_vocab[:50]
#.. [”,”, ”’”, ”;”, ”?”, ”’s”, ”...”, ”!”, ”Bazarov”, ”–”,
#.. ”n’t”, ”Arkady”, ”Petrovitch”, ”one”, ”I”, ”.”, ”said”,
#.. ”Pavel”, ”like”, ”Nikolai”, ”little”, ”even”, ”man”,
#.. ”though”, ”know”, ”time”, ”went”, ”could”, ”say”, ”Anna”,
#.. ”would”, ”Sergyevna”, ”Vassily”, ”old”, ”What”, ”began”,
#.. ”’You”, ”come”, ”see”, ”Madame”, ”go”, ”Ivanovitch”,
#.. ”must”, ”us”, ”””, ”eyes”, ”good”, ”young”, ”’m”,
#.. ”Odintsov”, ”without”]
```

COMBINING METHODS

Let’s combine all three methods: remove punctuation, ignore case, and remove stopwords:

```
text_nopunct = [”.join(re.split(
                        ”[.,;:!?‘’-]”, word)) for word in text]

text_content = [word for word in text_nopunct if word.lower(
                        ) not in stopwords]

fdist5 = nltk.FreqDist(text_content)

vocab5 = fdist5.keys()
```

COMBINING METHODS

```
vocab5[:50]
#.. [””, ”Bazarov”, ”Arkady”, ”Petrovitch”, ”nt”, ”one”, ”said”,
#.. ”Pavel”, ”like”, ”Nikolai”, ”little”, ”man”, ”even”, ”time”,
#.. ”though”, ”know”, ”went”, ”say”, ”could”, ”Sergyevna”, ”Anna”,
#.. ”would”, ”Vassily”, ”began”, ”old”, ”see”, ”away”, ”us”,
#.. ”come”, ”eyes”, ”Ivanovitch”, ”good”, ”day”, ”face”, ”go”,
#.. ”Fenitchka”, ”Madame”, ”Yes”, ”Odintsov”, ”Katya”, ”must”,
#.. ”Well”, ”head”, ”father”, ”young”, ”Yevgeny”, ”long”, ”m”,
#.. ”back”, ”first”]
```

FURTHER IMPROVEMENTS

We can add to the stopwords list:

```
more_stopwords = [””, ”nt”, ”us”, ”m”]

for word in stopwords:
    more_stopwords.append(word)
```

Which list are we adding to? `stopwords` or `more_stopwords`? Why?

FURTHER IMPROVEMENTS

```
text_content2 = [word for word in text_nopunct if word.lower(
                ) not in more_stopwords]

fdist6 = nltk.FreqDist(text_content2)

vocab6 = fdist6.keys()
```

FURTHER IMPROVEMENTS

```
vocab6[:50]
#.. [”Bazarov”, ”Arkady”, ”Petrovitch”, ”one”, ”said”, ”Pavel”,
#.. ”like”, ”Nikolai”, ”little”, ”man”, ”even”, ”time”, ”though”,
#.. ”know”, ”went”, ”say”, ”could”, ”Sergyevna”, ”Anna”, ”would”,
#.. ”Vassily”, ”began”, ”old”, ”see”, ”away”, ”come”, ”eyes”,
#.. ”Ivanovitch”, ”good”, ”day”, ”face”, ”go”, ”Fenitchka”,
#.. ”Madame”, ”Yes”, ”Odintsov”, ”Katya”, ”must”, ”Well”, ”head”,
#.. ”father”, ”young”, ”Yevgeny”, ”long”, ”back”, ”first”, ”think”,
#.. ”without”, ”made”, ”way”]
```

Now our list of most frequent words better represents the novel “Fathers and Sons”.

PLOTS

FREQUENCY DISTRIBUTIONS

Just what is that `FreqDist` thing we were using?

- `FreqDist` creates a dictionary in which the keys are tokens occurring in the text and the values are the corresponding frequencies.

```
type(fdist6)
#.. <class ”nltk.probability.FreqDist”>
fdist6
... <FreqDist with 8118 samples and 38207 outcomes>
print fdist6
#.. <FreqDist: ”Bazarov”: 520, ”Arkady”: 391,
#.. ”Petrovitch”: 358, ”one”: 272, ”said”: 213,
#.. ”Pavel”: 197, ”like”: 192, ”Nikolai”: 182,
#.. ”little”: 164, ”man”: 162, ...>
```

Thus ‘Bazarov’ occurs 520 times, ‘Arkady’ occurs 391 times, etc.

FREQUENCY DISTRIBUTIONS

Compare our first frequency distribution (prior to any cleaning-up)

```
print fdist
#.. <FreqDist: ”,”: 6721, ”the”: 2892, ”to”: 2145,
#.. ”and”: 2047, ”a”: 1839, ”of”: 1766, ”’”: 1589,
#.. ”in”: 1333, ”;”: 1230, ”he”: 1155, ...>
```

to our final one:

```
print fdist6
#.. <FreqDist: ”Bazarov”: 520, ”Arkady”: 391,
#.. ”Petrovitch”: 358, ”one”: 272, ”said”: 213,
#.. ”Pavel”: 197, ”like”: 192, ”Nikolai”: 182,
#.. ”little”: 164, ”man”: 162, ...>
```

FREQUENCY DISTRIBUTIONS

```
fdist.plot(50, cumulative=True)
```

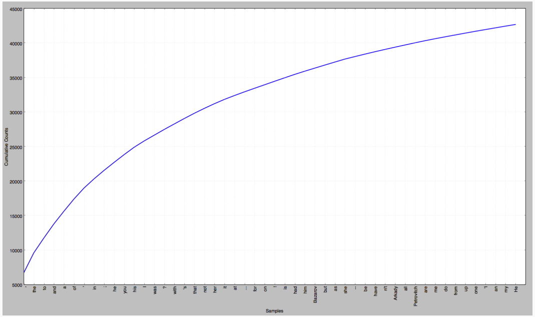

Figure: Cumulative frequency distribution prior to removal of punctuation and stop words.

FREQUENCY DISTRIBUTIONS

```
fdist6.plot(50, cumulative=True)
```

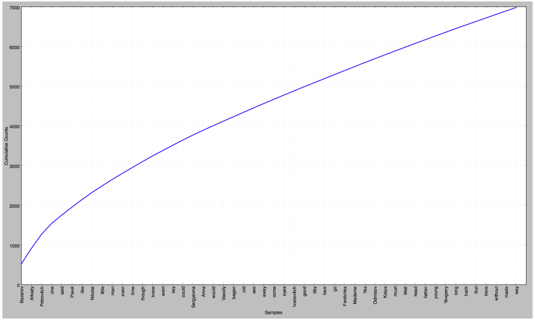

Figure: Cumulative frequency distribution after removal of punctuation and stop words.

```
fdist.plot(50, cumulative=False)
```

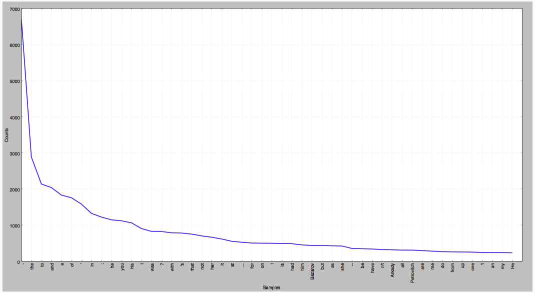

Figure: Non-cumulative frequency distribution prior to removal of punctuation and stop words.

```
fdist6.plot(50, cumulative=False)
```

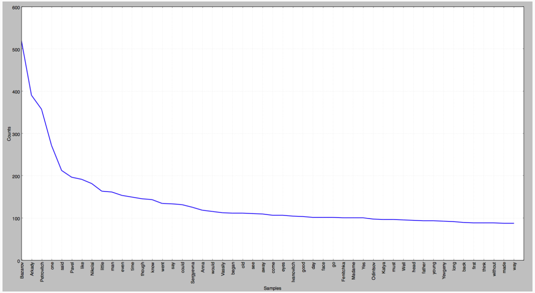

Figure: Non-cumulative frequency distribution after removal of punctuation and stop words.

SEARCHES

WHAT’S WRONG WITH THIS CONCORDANCE SEARCH?

```
text6 = nltk.Text(text_content2)
text6.concordance(”boy”)
#.. Building index...
#.. Displaying 11 of 11 matches:
#.. r hear Get things ready good boy look sharp Piotr modernised
#.. isite day today welcome dear boy Yes spring full loveliness T
#..  afraid frogs observed Vaska boy seven head white flax bare
#.. e Explain please earthly use boy know time throw rubbish idea
#..  said Arkady drawing Unhappy boy wailed Pavel Petrovitch posi
#.. reckoned liberal advise dear boy go call Governor said Arkady
#.. hinking women said low voice boy far observations go freethin
#.. d Arkady seen ups downs dear boy known hard way charming obse
#.. wing rejoinder re still fool boy see Sitnikovs indispensable
#.. dded indicating shortcropped boy come blue fullskirted coat r
#..  change said Katya conceited boy came Arkady went ve reached
```

DIGGING DEEPER INTO CONCORDANCES

Concordance searching should be done prior to cleaning up the text.

```
text.concordance(”boy”)
#.. Displaying 10 of 10 matches:
#.. ? Get things ready , my good boy : look sharp.’ Piotr , who
#.. frogs , ’ observed Vaska , a boy of seven , with a head as w
#..  no earthly use. He ’s not a boy , you know ; it ’s time to t
#.. drawing himself up. ’Unhappy boy ! ’ wailed Pavel Petrovitch
#.. ral. ’I advise you , my dear boy , to go and call on the Gove
#.. n a low voice. ’Because , my boy , as far as my observations
#.. seen ups and downs , my dear boy ; she ’s known what it is to
#.. : ’You ’re still a fool , my boy , I see. Sitnikovs are indis
#.. , indicating a short-cropped boy , who had come in with him i
#..  ’I am not now the conceited boy I was when I came here , ’ A
```

DIGGING DEEPER INTO CONCORDANCES

NLTK can use concordance data to look for similar words:

```
text.similar(”boy”)
#.. man child girl part rule sense sister woman advise and bird bit
#.. blade boast bookcase bottle box brain branch bucket
```

```
text.common_contexts([”boy”, ”girl”])
#.. a_of a_who
```

Both ‘boy’ and ‘girl’ occur in these two contexts (e.g., ‘a boy of seven’, ‘a girl of eighteen’).

DIGGING DEEPER INTO CONCORDANCES

```
text.concordance(”girl”)
#.. Displaying 4 of 15 matches:
#..  housewife , but as a young girl with a slim figure , innoce
#.. s two daughters – Anna , a girl of twenty , and Katya , a c
#.. s , and after him entered a girl of eighteen , black-haired
#.. ned to a bare-legged little girl of thirteen in a bright red
```

CONCLUSIONS

WHERE TO GO FROM HERE?

- Bird, Steven, Ewan Klein, and Edward Loper. 2009. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. Sebastopol, CA: O'Reilly. (Available online at http://www.nltk.org/book/)
- Perkins, Jacob. 2010. Python Text Processing with NLTK 2.0 Cookbook. Birmingham: Packt. (Available online through KU Library via ebrary)

Thank you