Introduction to Text Analysis with the Natural Language Toolkit

Matthew Menzenski
menzenski@ku.edu

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

March 7, 2014

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

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

Two ways to use Python

In the Python interpreter

- Type each line at the Python command prompt (> > >)
- Python interprets each line as it's entered

Running a standalone script

- Write up your program in a plain-text file
- Save it with the extension `.py`
- Execute it on the command line: `python myscript.py`

What are the advantages of each?

- The interpreter: great for experimenting with short pieces of code
- Standalone scripts: better for more complex programs

Python is famously human-readable

Python

```
1 for line in open("file.txt"):
2     for word in line.split():
3         if word.endswith("ing"):
4             print word
```

Perl

```
1 while (<>) {
2     foreach my $word (split) {
3         if ($word =~ /ing$/) {
4             print "\$word\n";
5 } }
6 }
```

(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.

```
1 >>> def repeat(message):
2 ...     return message + message
3 >>> monty = "Monty Python"
4 >>> repeat(monty)
5 "Monty Python Monty Python"
```

- In line 1 we define a function `repeat` that takes one argument, `message`.
- When called, this function returns that argument doubled.
- We define a variable `monty` and then call the function with `monty` as its argument.

Data types in Python

A number stores a numeric value.

```
>>> variable_1 = 10
```

A string is a continuous sequence of characters in quotation marks.

```
>>> variable_2 = "John Smith" # or 'John Smith'
```

A list contains items separated by commas and enclosed in square brackets. Items can be of different data types, and lists can be modified.

```
>>> variable_3 = [10, "John Smith", ["another", "list"]]
```

A tuple is like a list, but it is immutable (i.e., tuples are read-only).

```
>>> variable_4 = (10, "John Smith")
```

A dictionary contains key-value pairs, and is enclosed in curly braces.

```
>>> variable_5 = {"name": "John Smith", "department": "marketing"}
```

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

Preliminaries

Opening a new Python console

```
1 import nltk
2 import re
3 from urllib import urlopen
```

- Call import statements at the beginning
- We won't use `import re` right away, but we'll need it later

Getting data from a plain text file on the internet

Fetching the text file

```
>>> 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[:70]
"The Project Gutenberg eBook, Fathers and Children, by Ivan Sergeevich\n"
```

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 (cont'd)

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"
```

Getting data from a plain text file on the internet (cont'd)

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

```
>>> raw = urlopen(url).read()
```

We could also separate this into two steps, one to open the page and one to read its contents:

```
>>> webpage = urlopen(url)
>>> raw = webpage.read()
```

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

```
>>> type(raw)
<type "str">
```

The data in `raw` is of the `type` *string*.

Getting data from a plain text file on the internet (cont'd)

Query the length of `raw`

```
>>> len(raw)
448367
```

A string consists of characters, so our `raw` content is 448,367 characters long.

Display `raw` from its beginning to its 70th character

```
>>> raw[:70]
"The Project Gutenberg eBook, Fathers and Children, by Ivan Sergeevich\n"
```

`raw[10:100]` would display the tenth character to the 100th character, `raw[1000:]` would display from the 1000th character to the end, etc.

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

What is a "token"?

A token is a technical name for a sequence of characters that we want to treat as a group. "Token" is largely synonomous with "word", but there are differences.

Some sample tokens

- his
- antidisestablishmentarianism
- didn't
- ;
- state-of-the-art

Back to *Fathers and Sons*

Our data so far (in `raw`) is a single string that is 448,367 characters long. It's not very useful to us in that format. Let's split it into tokens.

Tokenizing our text

```
>>> 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*

The NLTK word tokenizer

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

`word_tokenize()` is the NLTK's default tokenizer function. It's possible to write your own, but `word_tokenize()` is usually appropriate in most situations.

- What does this tokenizer treat as the boundary between tokens?

Querying the `type` of `tokens`

```
>>> type(tokens)
<type "list">
```

`tokens` is of the `type` *list*.

Tokenizing *Fathers and Sons* (cont'd)

Querying the length of `tokens`

```
>>> len(tokens)
91736
```

- A `list` consists of items (which can include strings, numbers, lists, etc.).
- Our list `tokens` contains 91,736 items.
- What sort of items make up the list `tokens`?
- Does 91,736 tokens equal 91,736 words? Why or why not?

Tokenizing *Fathers and Sons* (cont'd)

tokens is a list of strings

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

When doing text analysis in Python, it is convienient to think of a text as a *list of words*.
The command `tokens[:10]` prints our list of tokens from the beginning to the tenth item.

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

More sophisticated text analysis with the NLTK

Defining an NLTK `text`

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

Calling the `nltk.Text()` module on `tokens` defines an NLTK Text, which allows us to call more sophisticated text analysis methods on it.

Querying the `type` of `text`

```
>>> type(text)
<class "nltk.text.Text">
```

The `type` of `text` is not a list, but a custom object defined in the NLTK.

Collocations

A collocation is a sequence of words that occur together unusually often.

Finding collocations with the NLTK is easy

```
>>> 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 as collocations? Why?

Collocations (cont'd)

Why did *Project Gutenberg* appear as a collocation?

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

Let's get rid of the header and footer

```
>>> raw.find("CHAPTER I")
1872
>>> raw.rfind("***END OF THE PROJECT GUTENBERG")
429664
>>> raw = raw[1872:429664]
>>> raw.find("CHAPTER I")
0
```

Collocations (cont'd)

Find the beginning of the text proper

```
>>> raw.find("CHAPTER I")
1872
```

The method `find()` starts at the beginning of a string and looks for the sequence `"CHAPTER I"`. This sequence begins with character 1872 in our raw text.

Find the end of the text proper

```
>>> raw.rfind("***END OF THE PROJECT GUTENBERG")
429664
```

Note that we're calling `rfind()` here, which searches the text beginning at the end. The sequence we're searching for begins at character 429664 in our raw text.

Collocations (cont'd)

Trimming the raw text

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

Now that we've found the beginning and the end of the actual content, we can redefine `raw` to include the content itself, minus the header and footer text.

Now the beginning is actually at the beginning

```
>>> raw.find("CHAPTER I")
0
```

In Python, the first position in a sequence is number 0. The next position is number 1. (Don't ask me why.)

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

Pulling text from an HTML file

- Our Project Gutenberg text was on the internet, but it was a plain text file.
- What if we want to read a more typical web page into the NLTK?

Web pages are more than just text

- The source code of most web pages contains markup in addition to the actual content
- There are formatting commands which we'll want to get rid of before working with the text proper
- For example: you might see **click here!**, but the HTML might actually contain `<a href="https://www.google.com">click here!</a>`
- Fortunately the NLTK makes it simple to remove such markup commands

Pulling text from an HTML file

We're going to use a dummy web page for this one

```
>>> web_url = "http://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`, and then opened and read it (this time into the variable `html`).
- But now there's all this markup to deal with!

Pulling text from an HTML file (cont'd)

Fortunately the NLTK makes cleaning up HTML easy

```
>>> 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 find the beginning and end

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

Finding concordances in a *Fathers and Sons*

A concordance view allows us to look at a given word in its contexts

```
>>> text.concordance("boy")
Displaying 10 of 10 matches:
ear ? Get things ready , my good boy ; look sharp.' Piotr , who as a
 of frogs , ' observed Vaska , a boy of seven , with a head as white
t 's no earthly use. He 's not a boy , you know ; it 's time to throw
y , drawing himself up. 'Unhappy boy ! ' wailed Pavel Petrovitch , he
liberal. 'I advise you , my dear boy , to go and call on the Governor
id in a low voice. 'Because , my boy , as far as my observations go ,
 's seen ups and downs , my dear boy ; she 's known what it is to be
der : 'You 're still a fool , my boy , I see. Sitnikovs are indispens
ded , indicating a short-cropped boy , who had come in with him in a
tya. 'I am not now the conceited boy I was when I came here , ' Arkad
```

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

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 (cont'd)

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"]
```

- These might indeed be the fifty most common tokens in *Fathers and Sons*, but they don't tell us very much.
- Except for three tokens, most of these seem like they'd be very frequent in *any* text.
- How can we 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 ## we already imported the regular expressions parser
>>> 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 (cont'd)

We could also convert all words to lowercase

(Let's do this independently of stripping the punctuation, at first.)

```
>>> 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"]
```

This way, *he* and *He* aren't counted as separate tokens.

Removing stop words

Finally, we could remove the 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 this text specifically

The NLTK includes lists of stop words for several languages

```
1 >>> from nltk.corpus import stopwords
2 >>> stopwords = stopwords.words("english")
```

Removing stop words (cont'd)

Removing stop 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"]
```

Removing stop words (cont'd)

Let's combine all three methods

```
>>> 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()
```

Removing stop words (cont'd)

Let's combine all three 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"]
```

How might we improve this list even further?

Removing stop words (cont'd)

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?

Removing stop words (cont'd)

Let's go back and remove our updated list of stopwords

```
>>> text_content2 = [word for word in text_nopunct if word.lower() not
in more_stopwords]
>>> fdist6 = nltk.FreqDist(text_content2)
>>> vocab6 = fdist6.keys()
>>> 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 does a much better job of telling us which tokens are uniquely common in *Fathers and Sons*.

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

Frequency Distributions

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

`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"> # another custom type
>>> 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, ...>
```

The key `"Bazarov"` is paired with the value 520, the number of times that it occurs in the text.

Frequency Distributions (cont'd)

The beginning of our very first frequency distribution

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

The beginning of our final frequency distribution

```
>>> 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 (cont'd)

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

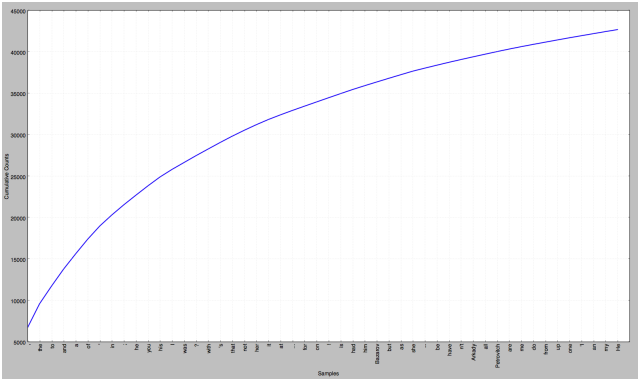

Figure: Cumulative frequency distribution prior to removal of punctuation and stop words.

Frequency Distributions (cont'd)

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

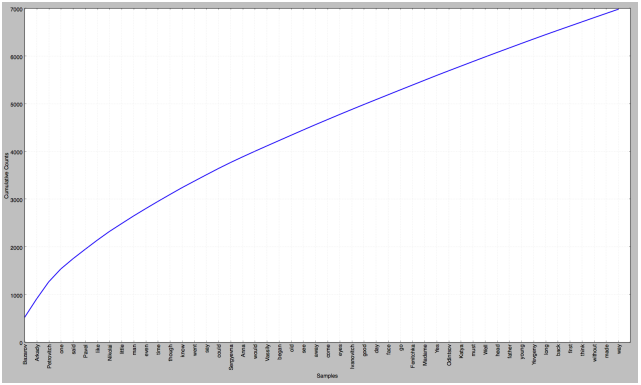

Figure: Cumulative frequency distribution after removal of punctuation and stop words.

Frequency Distributions (cont'd)

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

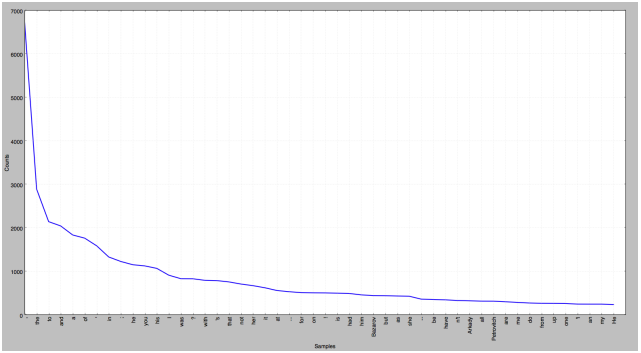

Figure: Non-cumulative frequency distribution prior to removal of punctuation and stop words.

Frequency Distributions (cont'd)

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

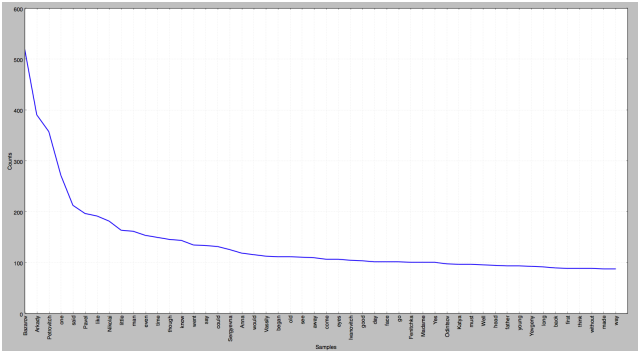

Figure: Non-cumulative frequency distribution after removal of punctuation and stop words.

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

Digging deeper into concordances

What's wrong with this data?

```
>>> text6 = nltk.Text(text_content2)
>>> text6.concordance("boy")
Building index...
Displaying 11 of 11 matches:
Piotr hear Get things ready good boy look sharp Piotr modernised serv
exquisite day today welcome dear boy Yes spring full loveliness Thoug
unny afraid frogs observed Vaska boy seven head white flax bare feet
while Explain please earthly use boy know time throw rubbish idea rom
ount said Arkady drawing Unhappy boy wailed Pavel Petrovitch positive
ugh reckoned liberal advise dear boy go call Governor said Arkady und
reethinking women said low voice boy far observations go freethinkers
arked Arkady seen ups downs dear boy known hard way charming observed
ollowing rejoinder re still fool boy see Sitnikovs indispensable unde
nt added indicating shortcropped boy come blue fullskirted coat ragge
 owe change said Katya conceited boy came Arkady went ve reached twen
```

Digging deeper into concordances (cont'd)

Concordance searching should be done *prior to* removing stopwords

```
>>> text.concordance("boy")
Building index...
Displaying 10 of 10 matches:
ear ? Get things ready , my good boy ; look sharp.' Piotr , who as a
 of frogs , ' observed Vaska , a boy of seven , with a head as white
t 's no earthly use. He 's not a boy , you know ; it 's time to throw
y , drawing himself up. 'Unhappy boy ! ' wailed Pavel Petrovitch , he
liberal. 'I advise you , my dear boy , to go and call on the Governor
id in a low voice. 'Because , my boy , as far as my observations go ,
 's seen ups and downs , my dear boy ; she 's known what it is to be
der : 'You 're still a fool , my boy , I see. Sitnikovs are indispens
ded , indicating a short-cropped boy , who had come in with him in a
tya. 'I am not now the conceited boy I was when I came here , ' Arkad
```

Digging deeper into concordances (cont'd)

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
```

Digging deeper into concordances (cont'd)

What makes these words similar to *boy*?

NLTK looks for words that occur in similar contexts.

We can search for those contexts too

```
>>> text.common_contexts(["boy", "girl"])
a_of a_who # both of these words occured between "a" and "of"
           # and between "a" and "who"
>>> text.concordance("girl")
Displaying 4 of 15 matches:
stic housewife , but as a young girl with a slim figure , innocently
o his two daughters -- Anna , a girl of twenty , and Katya , a child
 paws , and after him entered a girl of eighteen , black-haired and
 turned to a bare-legged little girl of thirteen in a bright red cot
```

Table of Contents

1 Introduction

2 The Natural Language Toolkit

3 Tokenization and text preprocessing

4 Collocations

5 HTML and Concordances

6 Frequencies and Stop Words

7 Plots

8 Searches

9 Conclusions

Where to go from here?

Books

- 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)