LG 467 Computers in Linguistics

[1-2021] Topic 4: Corpus Exploration

Sakol Suethanapornkul

Previously...

Corpus is a principled collection of naturally occurring texts

Corpus linguistics is a research approach (Biber & Reppen, 2015)

- ✔a methodological & analytical tool for linguistic analysis
- ✘ a hyphenated domain of enquiry (cf. sociolinguistics, psycholinguistics)

Biber & Reppen (2015) *Cambridge handbook of English corpus linguistics*

Previously...

Corpus contains nothing but frequency data (Gries, 2010)

Frequencies of occurrence

- how frequent are *morphemes*, *words*, *patterns*, *abstract constructions* in (part of) corpus
- Frequency list(s)

Types, tokens, Hapaxes

Frequencies of co-occurrence

- how often do elements co-occur with another element from this set or position in a text

Gries (2010) Useful statistics for corpus linguistics. In *A mosaic of corpus linguistics*

Previously...

List comprehension: Filtering elements from a list

Code 6.1

```
# [x for x in y if...]
doubt = "I doubt if this will work".split()

[w for w in doubt]
[w for w in doubt if 'w' in w]
[w for w in doubt if len(w) >= 4]

# [f(x) for x in y if...]
[w.upper() for w in doubt]
[len(w) for w in doubt]
[len(w) for w in doubt if len(w)>2 and w.startswith(w)]
```

Practice

Let's head over to Peter Norvig's website. Download a list of English words from `enable1.txt`.

Code 6.2

```
# Open a file with open()
# Call open() before reading in files
f = open('enable1.txt')          # or
f = open('Files/enable1.txt')

txt = f.read()
f.close()

words = txt.split()
print(words[0:51])
```

Aside: `read()` vs. `readlines()`

`read()` reads in an entire file at once & returns a string. Conversely, `readlines()` reads each line & returns a list of one-line strings.

Code 6.3

```
f = open('enable1.txt')
txt = f.read()
txt1 = f.readlines()

f.close()

print(txt[0:11])
print(txt1[0:11])
```

Practice

Use "words" to answer the following questions:

Code 6.4

```
# 1. What are the last ten words in the list?

# 2. What are the ten longest words?

# 3. Select words that start with 'a' and whose length > 15

# 4. Get word length of each word in the entire list

# 5. Get words that begin with and end with 'k'
```

Close files automatically

This may come in handy when you have a more complex code:

Code 6.5

```
with open('enable1.txt') as f:
    txt = f.read()
    words = txt.split()
```

For a more detailed explanation on `with`, check out Lubanovic (2020, p. 256)

Frequency counts

Now, let's get back to the Brown Corpus. NLTK offers an off-the-shelf frequency counts:

Code 6.6

```
from nltk.book import FreqDist
from nltk.corpus import brown

# If you get an error
import nltk
nltk.download(“book”)


words = brown.words(‘ca01')
wlist = FreqDist(words)
```

Frequency counts

You can obtain token frequencies in a text file with:

```
print(wlist)
# <FreqDist with 848 samples and 2242 outcomes>


wlist.most_common(10)
# [('the', 127), ('.', 88), (',', 87), ('of', 65),
('to', 55), ('a', 50), ('and', 40), ('in', 39), ('``',
34), ("''", 33)]

wlist.max()
wlist.N()
```

Code 6.6

[Continued]

Frequency counts

You can obtain token frequencies in a text file with:

```
# Reference a key-value pair with vars in a for-loop
print("words,", "counts")
for i, j in wlist.most_common():
    print(str(i) + ", " + str(j))

wlist.most_common(15)

wlist.plot()
wlist.plot(cumulative = True)
```

Code 6.6

[Continued]

Frequency counts

So, before we jump ahead and count tokens, it's important to:

Code 6.7

```
words  = [w.lower() for w in words]
wlist2 = FreqDist(words)

print(wlist2)
# <FreqDist with 800 samples and 2242 outcomes>


wlist2.most_common(10)
# [('the', 155), ('.', 88), (',', 87), ('of', 65),
('to', 55), ('a', 54), ('in', 40), ('and', 40), ('``',
34), ("''", 33)]
```

Frequency counts

The most frequent word in the list is "the."

- Does this word help us decipher a topic?
- Does this word provide useful information about the text?

Scroll further down through the list.

- What are some other "empty" words?
- What should we do with these words?

Frequency counts

We can remove **stop words** (commonly occurring words such as *a*, *an*, *the*, *of*, *in*, etc.) from the list before counting

Code 6.8

```
stop = ['a', 'an', 'the', 'in', 'on', 'at', 'to',
'for', 'of', 'and', '.']

words_sm = [w for w in words if w not in stop]

wlist3   = FreqDist(words_sm)
wlist3.most_common(10)
```

Frequency counts

NLTK provides a list of stop words that does a better job than ours. Import it from `nltk.corpus`:

Code 6.8

```
from nltk.corpus import stopwords
print(stopwords.words('english'))

stopwords = stopwords.words(‘english')
words_sm2 = [w for w in words if w not in stopwords]

wlist4    = FreqDist(words_sm2)
```

Frequency counts

We can calculate a type-token ratio (TTR) of the particular file we've been working on in the Brown Corpus

Code 6.9

```
len(words_sm2)
len(set(words_sm2))

# Type-token ratio
len(set(words_sm2))/len(words_sm2)
```

Question: Is `words_sm2` the right one for the calculation?

Frequency counts

What gets counted (and how) can yield different numbers:

- *walk–walked–walks*: one type or three different types?
  - if you're interested in word learning, one **lemma type** (walk)
  - if you're interested in learning of forms, three types
- Content vs. function words (and punctuation marks)

Questions you ask typically dictate analysis. So, think carefully

Frequency counts

NLTK comes bundled with a few novels; we'll use them to practice calculating TTR:

Code 6.10

```
print(text1)
print(text2)
print(text3)
print(text2[:31])
print(text3[:31])

len(set(text2))/len(text2)
len(set(text3))/len(text3)
```

Question: Calculate TTR of text1. Is Moby Dick most diverse?

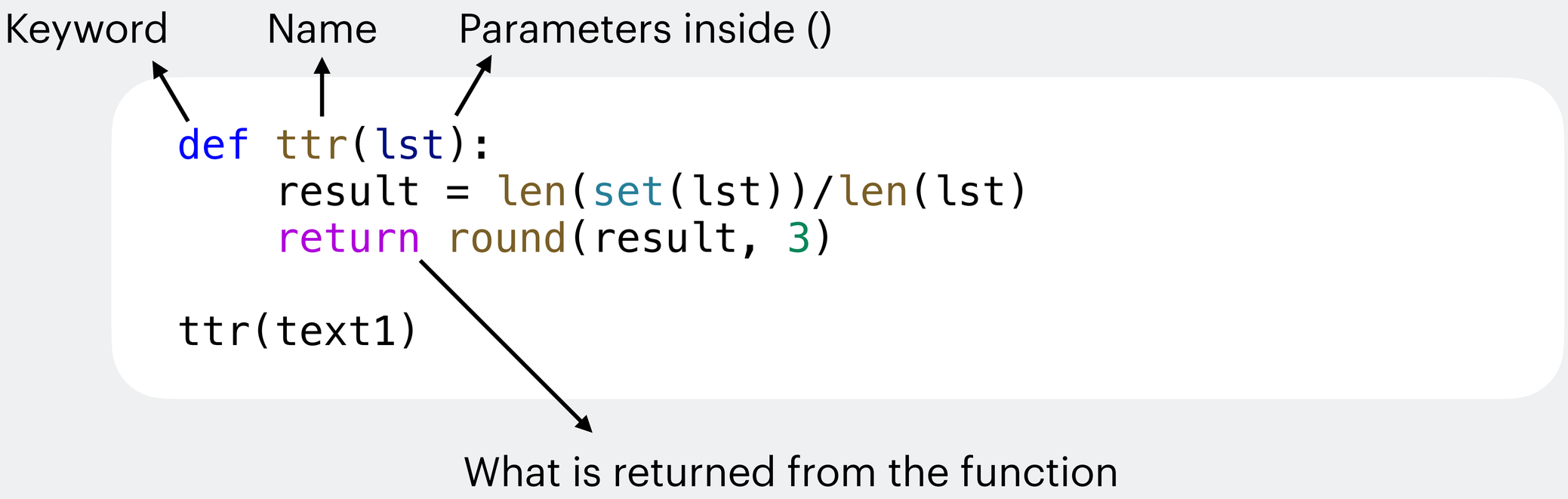
Frequency counts

TTR is affected by text size (i.e., how many tokens in a text?)

- Consider the following cases:
  - #1: Type = 12; Tokens = 24 TTR = 12/24 = 0.5
  - #2: Type = 12; Tokens = 200 TTR = 12/200 = 0.06
- Types don't grow linearly with text size
- TTR is meaningful when you have **comparatively sized texts**

Aside: defining functions

There was a lot of typing with our calculation of TTR. We can define a function to automate this calculation:



Code 6.11

Aside: defining functions

A function can take multiple arguments. Plus, you can add a **docstring** that tells others what this function does

Code 6.11

```
def ttr(lst, digits):
    """Compute type-token ratio on a list of strings,
    accepts one list and digits to round."""
    result = len(set(lst))/len(lst)
    return round(result, digits)

ttr(text1, 3)
ttr(text2, 4)
help(ttr)
```

N-grams

Language isn't just a bunch of words randomly strung together

Try this:

I don't know when I'll be back .....

- Most people might say: *again* or *here again* but not *yesterday*
- We can think of this as estimating the likelihood of word *w* given some history *h*

N-grams

We don't have a big enough corpus to estimate every possibility of every continuation. There's a clever way to go around this problem: **n-grams** (or **chunks**)

N-grams are units of *n* sequences

- unigrams, bigrams, trigrams, four-grams
  - character-level: fish (*f*, *i*, *s*, *h*) (*fi*, *is*, *sh*) (*fis*, *ish*)
  - word-level: Rose is red (*rose*, *is*, *red*) (*rose is*, *is red*)

N-grams and probabilities

Thinking of language as units of n sequences allows us to ask:

How likely is it for sequences of characters to appear?:

- *th ng kn kw qt tr pn np*

How likely is it for sequences of words to appear?:

- *boy the plan of go to you are that the*

We can use **conditional probability** to figure this out

N-grams and probabilities

Samples of n-grams from COCA

407044	more	than	
211840	may	be	
176666	most	of	
172563	me	to	
128502	must	be	
128363	make	a	
127343	might	be	
123719	many	of	
111762	make	it	
93378	may	have	
88996	much	of	
88282	make	the	
83028	me	a	
82497	my	life	
78572	make	sure	
77539	me	that	
74659	much	more	
72505	may	not	
72283	made	a	
72116	members	of	
71425	me	and	
71125	might	have	
69337	my	own	

80495	most	of	the	
52289	more	than	a	
45925	many	of	the	
40347	members	of	the	
32434	member	of	the	
31906	much	of	the	
29046	may	not	be	
26752	more	and	more	
26679	more	likely	to	
25762	middle	of	the	
23967	men	and	women	
20715	make	sure	that	
20432	may	have	been	
18151	might	have	been	
16574	many	of	them	
15758	more	than	one	
15739	must	have	been	
15249	more	of	a	
14593	may	be	a	
14309	more	than	the	
14093	most	of	them	
13432	me	in	the	
13276	much	of	a	
13095	my	name	is	
12977	more	or	less	
12852	may	be	the	

9434	most	of	the	time
6404	might	be	able	to
6347	middle	of	the	night
5935	may	or	may	not
5514	more	likely	to	be
4879	may	be	able	to
4434	more	than	half	of
4416	more	than	a	decade
4413	more	than	a	year
4378	my	husband	and	i
4120	more	than	a	few
3730	my	wife	and	i
3190	more	than	a	dozen
2937	make	it	to	the
2889	more	than	just	a
2888	more	than	any	other
2877	more	often	than	not
2857	make	sure	that	the
2836	me	tell	you	something
2836	much	for	joining	us
2804	may	not	be	the
2668	make	a	lot	of
2603	men	and	women	who
2571	most	of	the	people
2441	martin	luther	king	jr
2439	made	it	clear	that
2428	most	people	do	n't
2400	made	it	to	the
2378	may	not	be	able

Source: https://www.ngrams.info/samples_words.asp

N-grams and probabilities

Basic idea:

- CH: given *t*, what's the likelihood of the next character being *h*?
- WD: given *the*, what's the likelihood of the next word being *boy*?

Markov assumption

$$p(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(W_{n-1})}$$

N-grams are fundamental to many applications in NLP (spell checker, grammar checker, machine translation)

N-grams in NLTK

We can generate bigram (or any n-gram) counts with NLTK:

Code 6.12

```
import nltk

nltk.bigrams(text1)

list(nltk.bigrams(text1))

nltk.ngrams(text1, 2)
list(nltk.ngrams(text1, 2))

bigrams = list(nltk.ngrams(text1, 2))
bicount = FreqDist(bigrams)
```

Creating your own corpus with NLTK

You can "build" your own corpus with NLTK:

- http://www.nltk.org/book/ch02.html#loading-your-own-corpus

Make sure you have plain texts (.txt)

- `PlaintextCorpusReader` works with plain text files

`FreqDist` and its related functions will work with your corpus

Our plan next week...

- Part-of-speech (POS) tagging
  - POS tags
- Reading
  - NLTK Chapter 5