LING/C SC 581:

Advanced Computational Linguistics

Lecture 27

Today's Topics

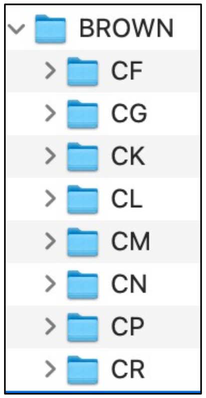
- Q&A for Optional Homeworks 11 and 12
- More on nltk and ptb
  - Zipf's Law
  - extracting the grammar rules (called **productions**)
  - looking for words with multiple POS tags

nltk.corpus: ptb

```
>>> import nltk
>>> from nltk.corpus import ptb
>>> ptb.fileids()
```

```
['BROWN/CF/CF01.MRG', 'BROWN/CF/CF02.MRG', 'BROWN/CF/CF03.MRG', 'BROWN/CF/CF04.MRG',
'BROWN/CF/CF05.MRG', 'BROWN/CF/CF06.MRG', 'BROWN/CF/CF07.MRG', 'BROWN/CF/CF08.MRG',
'BROWN/CF/CF09.MRG', 'BROWN/CF/CF10.MRG', 'BROWN/CF/CF11.MRG', 'BROWN/CF/CF12.MRG',
'BROWN/CF/CF13.MRG', 'BROWN/CF/CF14.MRG', 'BROWN/CF/CF15.MRG', 'BROWN/CF/CF16.MRG',
'BROWN/CF/CF17.MRG', 'BROWN/CF/CF18.MRG', 'BROWN/CF/CF19.MRG', 'BROWN/CF/CF20.MRG',
'BROWN/CF/CF21.MRG', 'BROWN/CF/CF22.MRG', 'BROWN/CF/CF23.MRG', 'BROWN/CF/CF24.MRG',
'BROWN/CF/CF25.MRG', 'BROWN/CF/CF26.MRG', 'BROWN/CF/CF27.MRG', 'BROWN/CF/CF28.MRG',
'BROWN/CF/CF29.MRG', 'BROWN/CF/CF30.MRG', 'BROWN/CF/CF31.MRG', 'BROWN/CF/CF32.MRG',
'BROWN/CG/CG01.MRG', 'BROWN/CG/CG02.MRG', 'BROWN/CG/CG03.MRG', 'BROWN/CG/CG04.MRG',
'BROWN/CG/CG05.MRG', 'BROWN/CG/CG06.MRG', 'BROWN/CG/CG07.MRG', 'BROWN/CG/CG08.MRG',
'BROWN/CG/CG09.MRG', 'BROWN/CG/CG10.MRG', 'BROWN/CG/CG11.MRG', 'BROWN/CG/CG12.MRG',
'BROWN/CG/CG13.MRG', 'BROWN/CG/CG14.MRG', 'BROWN/CG/CG15.MRG', 'BROWN/CG/CG16.MRG',
'BROWN/CG/CG17.MRG', 'BROWN/CG/CG18.MRG', 'BROWN/CG/CG19.MRG', 'BROWN/CG/CG20.MRG',
'BROWN/CG/CG21.MRG', 'BROWN/CG/CG22.MRG', 'BROWN/CG/CG23.MRG', 'BROWN/CG/CG24.MRG',
'BROWN/CG/CG25.MRG', 'BROWN/CG/CG26.MRG', 'BROWN/CG/CG27.MRG', 'BROWN/CG/CG28.MRG',
'BROWN/CG/CG29.MRG', 'BROWN/CG/CG30.MRG', 'BROWN/CG/CG31.MRG', 'BROWN/CG/CG32.MRG',
'BROWN/CG/CG33.MRG', 'BROWN/CG/CG34.MRG', 'BROWN/CG/CG35.MRG', 'BROWN/CG/CG36.MRG',
'BROWN/CK/CK01.MRG', 'BROWN/CK/CK02.MRG', 'BROWN/CK/CK03.MRG', 'BROWN/CK/CK04.MRG',
'BROWN/CK/CK05.MRG', 'BROWN/CK/CK06.MRG', 'BROWN/CK/CK07.MRG', 'BROWN/CK/CK08.MRG',
'BROWN/CK/CK09.MRG', 'BROWN/CK/CK10.MRG', 'BROWN/CK/CK11.MRG', 'BROWN/CK/CK12.MRG',
'BROWN/CK/CK13.MRG', 'BROWN/CK/CK14.MRG', 'BROWN/CK/CK15.MRG', 'BROWN/CK/CK16.MRG',
'BROWN/CK/CK17.MRG', 'BROWN/CK/CK18.MRG', 'BROWN/CK/CK19.MRG', 'BROWN/CK/CK20.MRG',
'BROWN/CK/CK21.MRG', 'BROWN/CK/CK22.MRG', 'BROWN/CK/CK23.MRG', 'BROWN/CK/CK24.MRG',
'BROWN/CK/CK25.MRG', 'BROWN/CK/CK27.MRG', 'BROWN/CK/CK28.MRG', 'BROWN/CK/CK29.MRG',
'BROWN/CL/CL01.MRG', 'BROWN/CL/CL02.MRG', 'BROWN/CL/CL03.MRG', 'BROWN/CL/CL04.MRG',
'BROWN/CL/CL05.MRG', 'BROWN/CL/CL06.MRG', 'BROWN/CL/CL07.MRG', 'BROWN/CL/CL08.MRG',
'BROWN/CL/CL09.MRG', 'BROWN/CL/CL10.MRG', 'BROWN/CL/CL11.MRG', 'BROWN/CL/CL12.MRG',
'BROWN/CL/CL13.MRG', 'BROWN/CL/CL14.MRG', 'BROWN/CL/CL15.MRG', 'BROWN/CL/CL16.MRG',
'BROWN/CL/CL17.MRG', 'BROWN/CL/CL18.MRG', 'BROWN/CL/CL19.MRG', 'BROWN/CL/CL20.MRG',
'BROWN/CL/CL21.MRG', 'BROWN/CL/CL22.MRG', 'BROWN/CL/CL23.MRG', 'BROWN/CL/CL24.MRG',
'BROWN/CM/CM01.MRG', 'BROWN/CM/CM02.MRG', 'BROWN/CM/CM03.MRG', 'BROWN/CM/CM04.MRG',
'BROWN/CM/CM05.MRG', 'BROWN/CM/CM06.MRG', 'BROWN/CN/CN01.MRG', 'BROWN/CN/CN02.MRG',
'BROWN/CN/CN03.MRG', 'BROWN/CN/CN04.MRG', 'BROWN/CN/CN05.MRG', 'BROWN/CN/CN06.MRG',
'BROWN/CN/CN07.MRG', 'BROWN/CN/CN08.MRG', 'BROWN/CN/CN09.MRG', 'BROWN/CN/CN10.MRG',
'BROWN/CN/CN11.MRG', 'BROWN/CN/CN12.MRG', 'BROWN/CN/CN13.MRG', 'BROWN/CN/CN14.MRG',
'BROWN/CN/CN15.MRG', 'BROWN/CN/CN16.MRG', 'BROWN/CN/CN17.MRG', 'BROWN/CN/CN18.MRG',
'BROWN/CN/CN19.MRG', 'BROWN/CN/CN20.MRG', 'BROWN/CN/CN21.MRG', 'BROWN/CN/CN22.MRG',
'BROWN/CN/CN23.MRG', 'BROWN/CN/CN24.MRG', 'BROWN/CN/CN25.MRG', 'BROWN/CN/CN26.MRG',
'BROWN/CN/CN27.MRG', 'BROWN/CN/CN28.MRG', 'BROWN/CN/CN29.MRG', 'BROWN/CP/CP01.MRG',
'BROWN/CP/CP02.MRG', 'BROWN/CP/CP03.MRG', 'BROWN/CP/CP04.MRG', 'BROWN/CP/CP05.MRG',
'BROWN/CP/CP06.MRG', 'BROWN/CP/CP07.MRG', 'BROWN/CP/CP08.MRG', 'BROWN/CP/CP09.MRG',
'BROWN/CP/CP10.MRG', 'BROWN/CP/CP11.MRG', 'BROWN/CP/CP12.MRG', 'BROWN/CP/CP13.MRG',
'BROWN/CP/CP14.MRG', 'BROWN/CP/CP15.MRG', 'BROWN/CP/CP16.MRG', 'BROWN/CP/CP17.MRG',
'BROWN/CP/CP18.MRG', 'BROWN/CP/CP19.MRG', 'BROWN/CP/CP20.MRG', 'BROWN/CP/CP21.MRG',
'BROWN/CP/CP22.MRG', 'BROWN/CP/CP23.MRG', 'BROWN/CP/CP25.MRG', 'BROWN/CP/CP26.MRG',
'BROWN/CP/CP27.MRG', 'BROWN/CP/CP28.MRG', 'BROWN/CP/CP29.MRG', 'BROWN/CR/CR01.MRG',
'BROWN/CR/CR02.MRG', 'BROWN/CR/CR03.MRG', 'BROWN/CR/CR04.MRG', 'BROWN/CR/CR05.MRG',
'BROWN/CR/CR06.MRG', 'BROWN/CR/CR07.MRG', 'BROWN/CR/CR08.MRG', 'BROWN/CR/CR09.MRG',

…
```

192



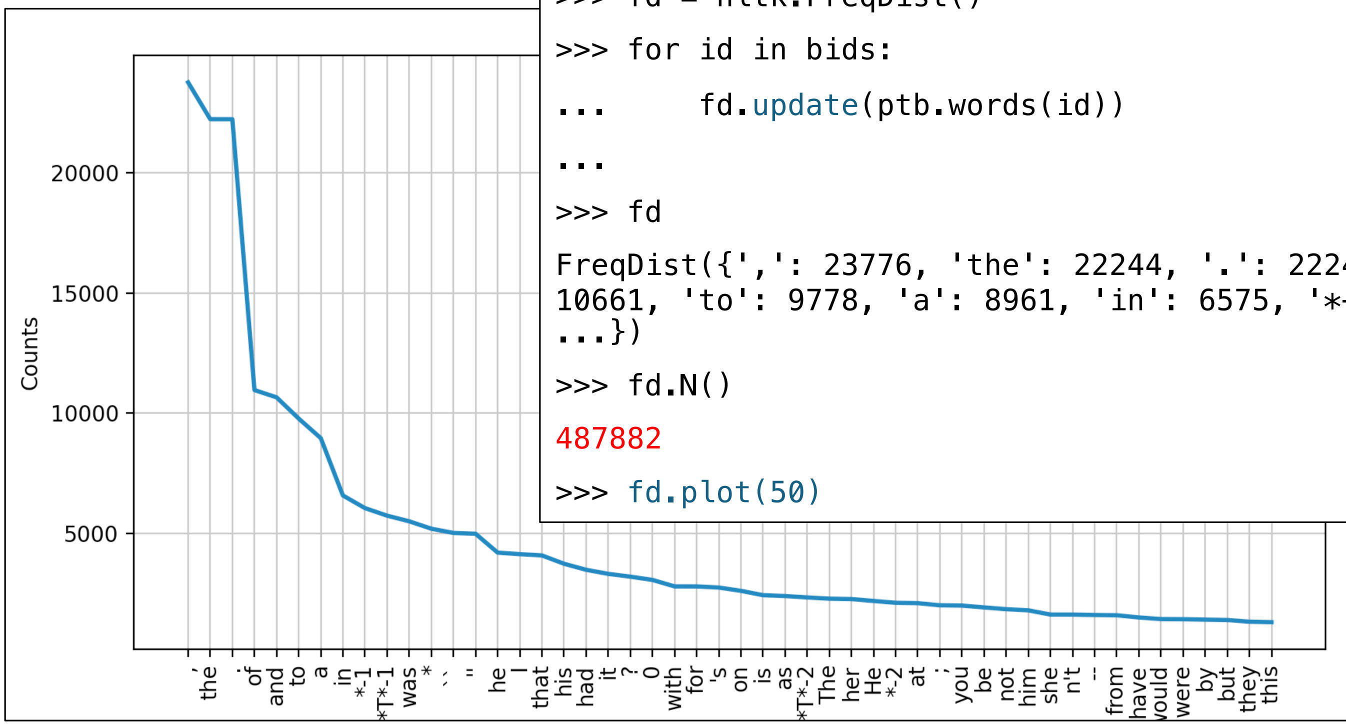

```
'WSJ/00/WSJ_0004.MRG', 'WSJ/00/WSJ_0005.MRG', 'WSJ/00/WSJ_0006.MRG', 'WSJ/00/WSJ_0007.MRG',
'WSJ/00/WSJ_0008.MRG', 'WSJ/00/WSJ_0009.MRG', 'WSJ/00/WSJ_0010.MRG', 'WSJ/00/WSJ_0011.MRG',
'WSJ/00/WSJ_0012.MRG', 'WSJ/00/WSJ_0013.MRG', 'WSJ/00/WSJ_0014.MRG', 'WSJ/00/WSJ_0015.MRG',
'WSJ/00/WSJ_0016.MRG', 'WSJ/00/WSJ_0017.MRG', 'WSJ/00/WSJ_0018.MRG', 'WSJ/00/WSJ_0019.MRG',
'WSJ/00/WSJ_0020.MRG', 'WSJ/00/WSJ_0021.MRG', 'WSJ/00/WSJ_0022.MRG', 'WSJ/00/WSJ_0023.MRG',
'WSJ/00/WSJ_0024.MRG', 'WSJ/00/WSJ_0025.MRG', 'WSJ/00/WSJ_0026.MRG', 'WSJ/00/WSJ_0027.MRG',
'WSJ/00/WSJ_0028.MRG', 'WSJ/00/WSJ_0029.MRG', 'WSJ/00/WSJ_0030.MRG', 'WSJ/00/WSJ_0031.MRG',
'WSJ/00/WSJ_0032.MRG', 'WSJ/00/WSJ_0033.MRG', 'WSJ/00/WSJ_0034.MRG', 'WSJ/00/WSJ_0035.MRG',
'WSJ/00/WSJ_0036.MRG', 'WSJ/00/WSJ_0037.MRG', 'WSJ/00/WSJ_0038.MRG',

…

'WSJ/23/WSJ_2398.MRG', 'WSJ/23/WSJ_2399.MRG', 'WSJ/24/WSJ_2400.MRG', 'WSJ/24/WSJ_2401.MRG',
'WSJ/24/WSJ_2402.MRG', 'WSJ/24/WSJ_2403.MRG', 'WSJ/24/WSJ_2404.MRG', 'WSJ/24/WSJ_2405.MRG',
'WSJ/24/WSJ_2406.MRG', 'WSJ/24/WSJ_2407.MRG', 'WSJ/24/WSJ_2408.MRG', 'WSJ/24/WSJ_2409.MRG',
'WSJ/24/WSJ_2410.MRG', 'WSJ/24/WSJ_2411.MRG', 'WSJ/24/WSJ_2412.MRG', 'WSJ/24/WSJ_2413.MRG',
'WSJ/24/WSJ_2414.MRG', 'WSJ/24/WSJ_2415.MRG', 'WSJ/24/WSJ_2416.MRG', 'WSJ/24/WSJ_2417.MRG',
'WSJ/24/WSJ_2418.MRG', 'WSJ/24/WSJ_2419.MRG', 'WSJ/24/WSJ_2420.MRG', 'WSJ/24/WSJ_2421.MRG',
'WSJ/24/WSJ_2422.MRG', 'WSJ/24/WSJ_2423.MRG', 'WSJ/24/WSJ_2424.MRG', 'WSJ/24/WSJ_2425.MRG',
'WSJ/24/WSJ_2426.MRG', 'WSJ/24/WSJ_2427.MRG', 'WSJ/24/WSJ_2428.MRG', 'WSJ/24/WSJ_2429.MRG',
'WSJ/24/WSJ_2430.MRG', 'WSJ/24/WSJ_2431.MRG', 'WSJ/24/WSJ_2432.MRG', 'WSJ/24/WSJ_2433.MRG',
'WSJ/24/WSJ_2434.MRG', 'WSJ/24/WSJ_2435.MRG', 'WSJ/24/WSJ_2436.MRG', 'WSJ/24/WSJ_2437.MRG',
'WSJ/24/WSJ_2438.MRG', 'WSJ/24/WSJ_2439.MRG', 'WSJ/24/WSJ_2440.MRG', 'WSJ/24/WSJ_2441.MRG',
'WSJ/24/WSJ_2442.MRG', 'WSJ/24/WSJ_2443.MRG', 'WSJ/24/WSJ_2444.MRG', 'WSJ/24/WSJ_2445.MRG',
'WSJ/24/WSJ_2446.MRG', 'WSJ/24/WSJ_2447.MRG', 'WSJ/24/WSJ_2448.MRG', 'WSJ/24/WSJ_2449.MRG',
'WSJ/24/WSJ_2450.MRG', 'WSJ/24/WSJ_2451.MRG', 'WSJ/24/WSJ_2452.MRG', 'WSJ/24/WSJ_2453.MRG',
'WSJ/24/WSJ_2454.MRG']
```

2312

Brown corpus: FreqDist

Brown corpus file ids

```
or x in ptb.fileids() if x.startswith('BROWN')]

>>> len(bids)

192

>>> fd = nltk.FreqDist()

>>> for id in bids:

...     fd.update(ptb.words(id))

...

>>> fd

FreqDist({',': 23776, 'the': 22244, '.': 22241, 'of': 10964, 'and': 10661, 'to': 9778, 'a': 8961, 'in': 6575, '*-1': 6048, '*T*-1': 5734, ...})

>>> fd.N()

487882

>>> fd.plot(50)
```



`nltk.FreqDist(`*`corpus`*`)`

nltk.probability.FreqDist

class nltk.probability.FreqDist **[source]**

Bases: `Counter`

A frequency distribution for the outcomes of an experiment. A frequency distribution records the number of times each outcome of an experiment has occurred. For example, a frequency distribution could be used to record the frequency of each word type in a document. Formally, a frequency distribution can be defined as a function mapping from each sample to the number of times that sample occurred as an outcome.

N() **[source]**

Return the total number of sample outcomes that have been recorded by this FreqDist. For the number of unique sample values (or bins) with counts greater than zero, use `FreqDist.B()`.

Return type

int

update(**args, **kwargs*) **[source]**

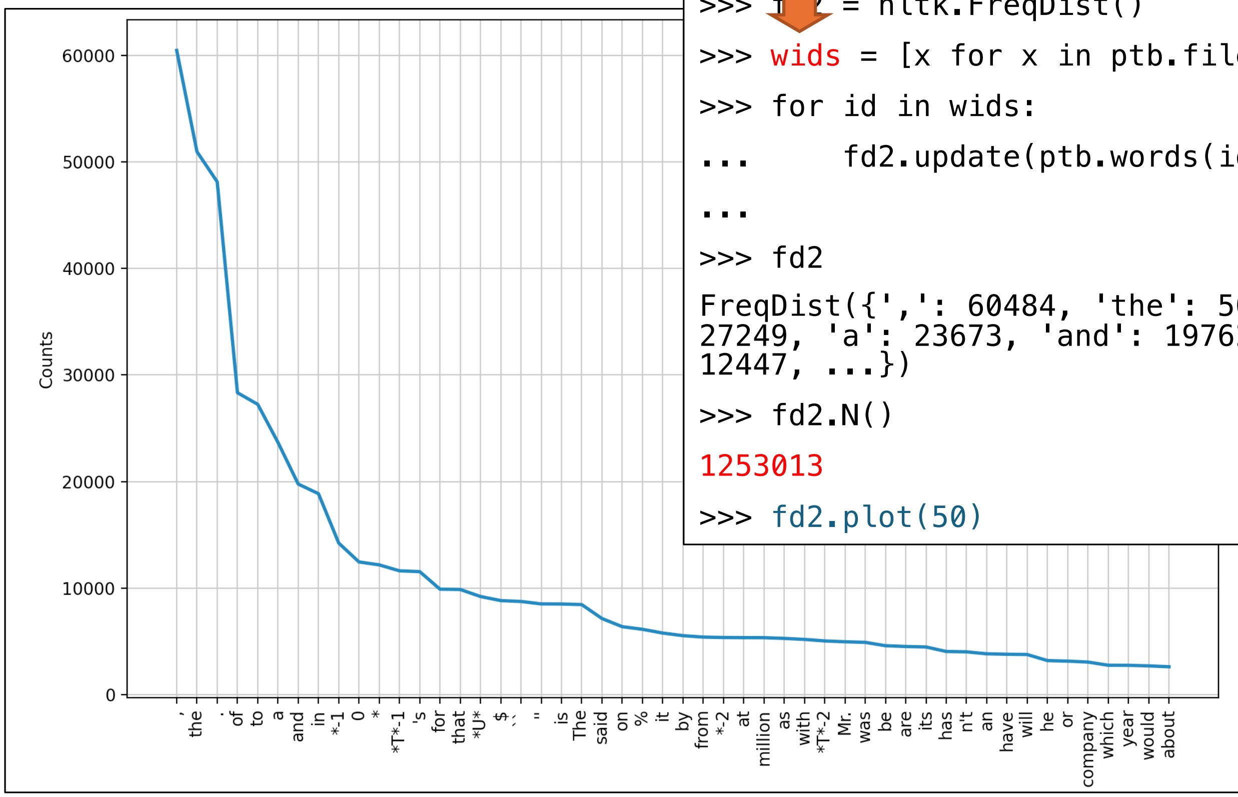
Override `Counter.update()` to invalidate the cached N

`FreqDist.update`(**args, **kwds*)

Like dict.update() but add counts instead of replacing them.

Source can be an iterable, a dictionary, or another Counter instance.

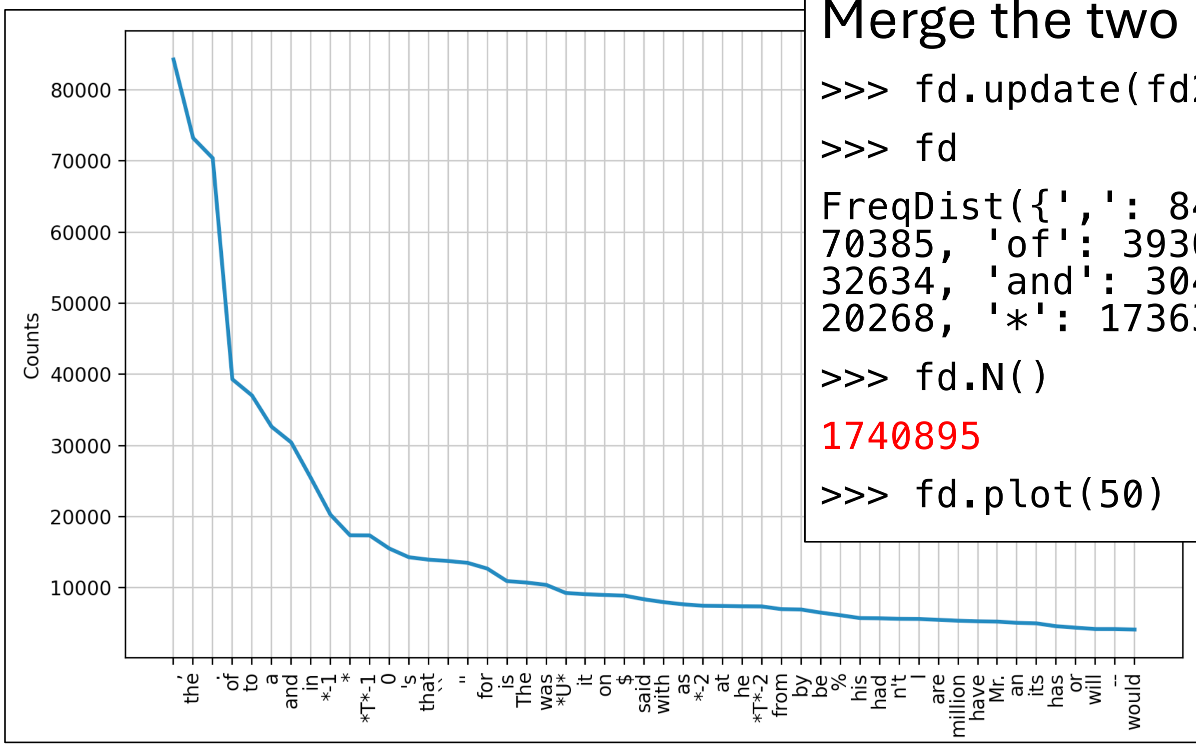
ptb



WSJ corpus file ids

```
>>> fd2 = nltk.FreqDist()
>>> wids = [x for x in ptb.fileids() if x.startswith('WSJ')]
>>> for id in wids:
...     fd2.update(ptb.words(id))
...
>>> fd2
FreqDist({',': 60484, 'the': 50975, '.': 48144, 'of': 28338, 'to': 27249, 'a': 23673, 'and': 19762, 'in': 18857, '*-1': 14220, '0': 12447, ...})
>>> fd2.N()
1253013
>>> fd2.plot(50)
```

ptb



Merge the two FreqDists:

```
>>> fd.update(fd2)
>>> fd
FreqDist({',': 84260, 'the': 73219, '.': 70385, 'of': 39302, 'to': 37027, 'a': 32634, 'and': 30423, 'in': 25432, '*-1': 20268, '*': 17363, ...})
>>> fd.N()
1740895
>>> fd.plot(50)
```

Zipf's Law

- Zipf's law was originally formulated in terms of quantitative linguistics, stating that given some corpus of natural language utterances, **the frequency of any word is inversely proportional to its rank** in the frequency table.
  - Thus, the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.
  - For example, in the Brown Corpus of American English text, the word "**the**" is the most frequently occurring word, and by itself accounts for nearly 7% of all word occurrences (69,971 out of slightly over 1 million).
  - True to Zipf's Law, the second-place word "of" accounts for slightly over 3.5% of words (36,411 occurrences), followed by "and" (28,852).
  - **Only 135 vocabulary items** are needed to account for half the Brown Corpus.

Zipf's Law

Wolfram

Probability and Statistics › Descriptive Statistics ›

Zipf's Law

In the English language, the probability of encountering the rth most common word is given roughly by $P(r) = 0.1 / r$ for r up to 1000 or so. The law breaks down for less frequent words, since the harmonic series diverges. Pierce's (1980, p. 87) statement that $\sum P(r) > 1$ for

Equation:

- freq = c . rank $^{-m}$
  - for positive constants m and c
- log(freq) = –m log(rank) + log(c)
- has the form of an equation of a straight line (i.e. y=mx+c)

Code:

zipf.py given on the course webpage

```
>>> import zipf
>>> zipf.plot(tokens)
```

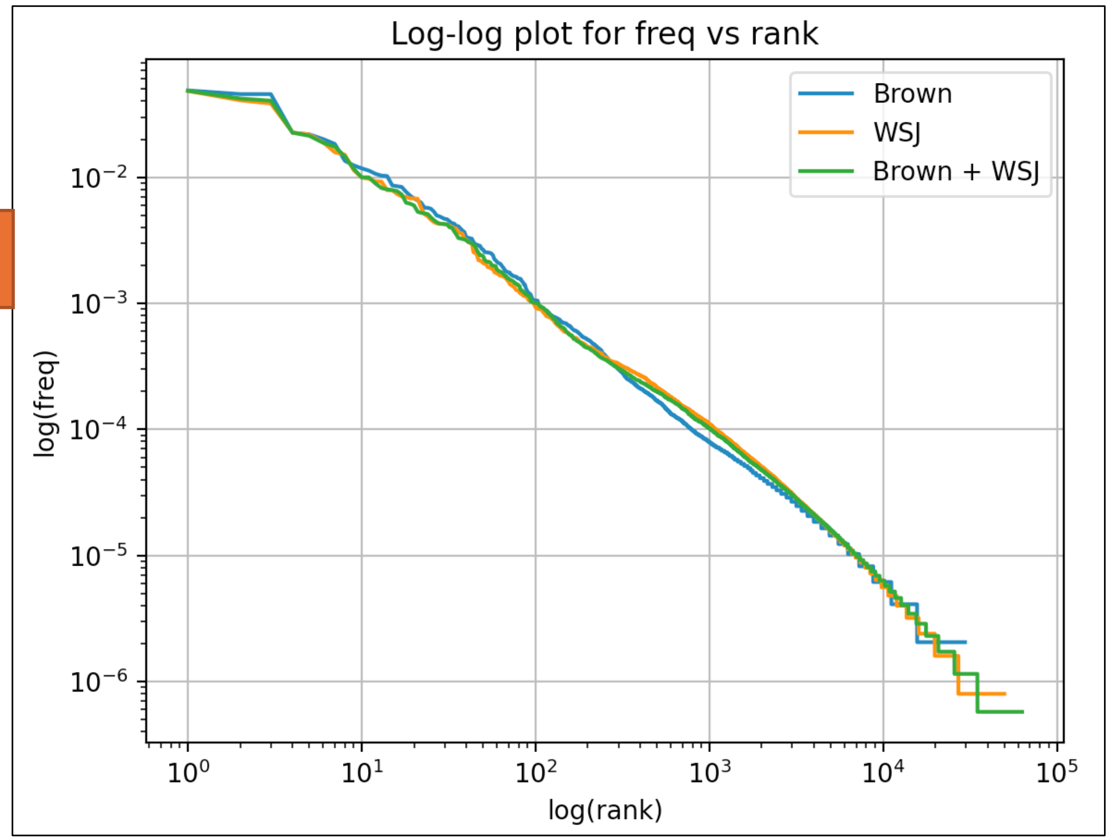
tokens = list of words (a corpus)

Zipf's Law: ptb

```
>>> wws = []
>>> for id in wids:
...     wws.extend(ptb.words(id))
...
>>> bws = []
>>> for id in bids:
...     bws.extend(ptb.words(id))
...
>>> len(bws)
487882
>>> len(wws)
1253013
>>> from zipf import *
>>> fig()
>>> plot(bws, "Brown")
>>> plot(wws, "WSJ")
>>> plot(bws+wws, "Brown + WSJ")
>>> plt.legend()
<matplotlib.legend.Legend object at 0x1278c4520>
>>> plt.show()
```

WSJ words

Brown words



Zipf's Law: ptb

On course website: `zipf.py`

```
# Sandiway Fong (c) University of Arizona 2019
# simple function to plot Zipf's Law
# assumes matplotlib
from collections import Counter
from math import log, log10
import matplotlib.pyplot as plt

def plot(tokens, text):
    size = len(tokens)
    c = Counter()
    for token in tokens:
        c[token] += 1
    mc = c.most_common()
    ranks = [x for x in range(1,len(mc)+1)]
    freq = [item[1]/size for item in mc]
    plt.plot(ranks,freq, label=text)
```

```

def fig():
    plt.figure(1)
    plt.xscale('log')
    plt.xlabel('log(rank)')
    plt.yscale('log')
    plt.ylabel('log(freq)')
    plt.grid(True)
    plt.title('Log-log plot for freq vs rank')
```

Trees and Productions

```
>>> len(list(ptb.parsed_sents()[0].subtrees()))
87
```

productions() [source]

Generate the productions that correspond to the non-terminal nodes of the tree. For each subtree of the form (P: C1 C2 ... Cn) this produces a production of the form P -> C1 C2 ... Cn.

```
>>> len(ptb.parsed_sents()[0].productions())
87
```

Trees and Productions

```
>>> ptb.parsed_sents()[0].productions()
[S -> S : S ., S -> PP , NP-SBJ-2 VP, PP -> IN NP, IN -> 'In', NP -> JJ
NN, JJ -> 'American', NN -> 'romance', , -> ',', NP-SBJ-2 -> RB NN, RB ->
'almost', NN -> 'nothing', VP -> VBZ S, VBZ -> 'rates', S -> NP-SBJ ADJP-
PRD, NP-SBJ -> -NONE-, -NONE- -> '*-2', ADJP-PRD -> ADJP PP, ADJP -> JJR,
JJR -> 'higher', PP -> IN SBAR-NOM, IN -> 'than', SBAR-NOM -> WHNP-1 S,
WHNP-1 -> WP, WP -> 'what', S -> NP-SBJ VP, NP-SBJ -> DT NN NNS, DT ->
'the', NN -> 'movie', NNS -> 'men', VP -> VB VP, VB -> 'have', VP -> VBN
S, VBN -> 'called', S -> NP-SBJ `` S-NOM-PRD '', NP-SBJ -> -NONE-, -NONE-
-> '*T*-1', `` -> '``', S-NOM-PRD -> NP-SBJ VP, NP-SBJ -> -NONE-, -NONE- -
> '*', VP -> NN NP, NN -> 'meeting', NP -> JJ, JJ -> 'cute', '' -> "''", :
-> '--', S -> S-ADV , NP-SBJ VP, S-ADV -> NP-SBJ VP, NP-SBJ -> DT, DT ->
'that', VP -> VBZ, VBZ -> 'is', , -> ',', NP-SBJ -> NN, NN -> 'boy-meets-
girl', VP -> VBZ ADJP-PRD SBAR-ADV, VBZ -> 'seems', ADJP-PRD -> RB JJ, RB
-> 'more', JJ -> 'adorable', SBAR-ADV -> IN S, IN -> 'if', S -> NP-SBJ VP,
NP-SBJ -> PRP, PRP -> 'it', VP -> VBZ RB VP, VBZ -> 'does', RB -> "n't",
VP -> VB NP PP, VB -> 'take', NP -> NN, NN -> 'place', PP -> IN NP, IN ->
'in', NP -> NP PP, NP -> DT NN, DT -> 'an', NN -> 'atmosphere', PP -> IN
NP, IN -> 'of', NP -> ADJP NN, ADJP -> JJ CC JJ, JJ -> 'correct', CC ->
'and', JJ -> 'acute', NN -> 'boredom', . -> '.']
```

Trees and Productions

- https://www.nltk.org/api/nltk.grammar.Production.html

```
>>> ptb.parsed_sents()[0].productions()[2]
PP -> IN NP
>>> type(ptb.parsed_sents()[0].productions()[2])
<class 'nltk.grammar.Production'>
>>> ptb.parsed_sents()[0].productions()[2].lhs()
PP
>>> ptb.parsed_sents()[0].productions()[2].rhs()
(IN, NP)
>>> type(ptb.parsed_sents()[0].productions()[2].rhs())
<class 'tuple'>
```

Trees and Productions

3rd rule is PP -> IN NP:

```
>>> ptb.parsed_sents()[0].productions()[2].rhs()[0]
IN
>>> ptb.parsed_sents()[0].productions()[2].rhs()[1]
NP
>>> len(ptb.parsed_sents()[0].productions()[2].rhs())
2
```

Trees and Productions

3rd rule is PP -> IN NP:

```
>>> ptb.parsed_sents()[0].productions()[2].is_nonlexical()
True
>>> ptb.parsed_sents()[0].productions()[2].is_lexical()
False
```

is_nonlexical()

Return True if the right-hand side only contains `Nonterminals`

is_lexical()

Return True if the right-hand contain at least one terminal token.

Words with multiple POS tags

- Let's write a program to find words with more than one part of speech tag.
- First, let's get all the word-tag items:

```
>>> wt = [item for tree in ptb.parsed_sents() for item in tree.pos()]
>>> len(wt)
1740895
```

- Next, let's get the set of word-tag items, no duplicates:

```
>>> wts = set(wt)
>>> len(wts)
74323
```

wts = word tag set

Words with multiple POS tags

- Let's create a dictionary with pos tags as values:

```
>>> d = {}
>>> for item in wts:
>>>     d.setdefault(item[0], []).append(item[1])
>>>
>>> len(d)
63073
```

Words with multiple POS tags

- *Let's look at a few examples …*

```
>>> d['any']
['DT', 'RB']
>>> d['Any']
['DT']
```

tagguid1.pdf

any can be a determiner (DT).

EXAMPLES: We don't have any/DT.
Don't you want any/DT more/JJR?

However, when it precedes a comparative adverb, it is an adverb (RB).

EXAMPLES: I can't run any/RB further/RBR.
I can't go on like this any/RB more/RBR.

Words with multiple POS tags

about when used to mean "approximately" should be tagged as an adverb (RB), rather than a preposition (IN).

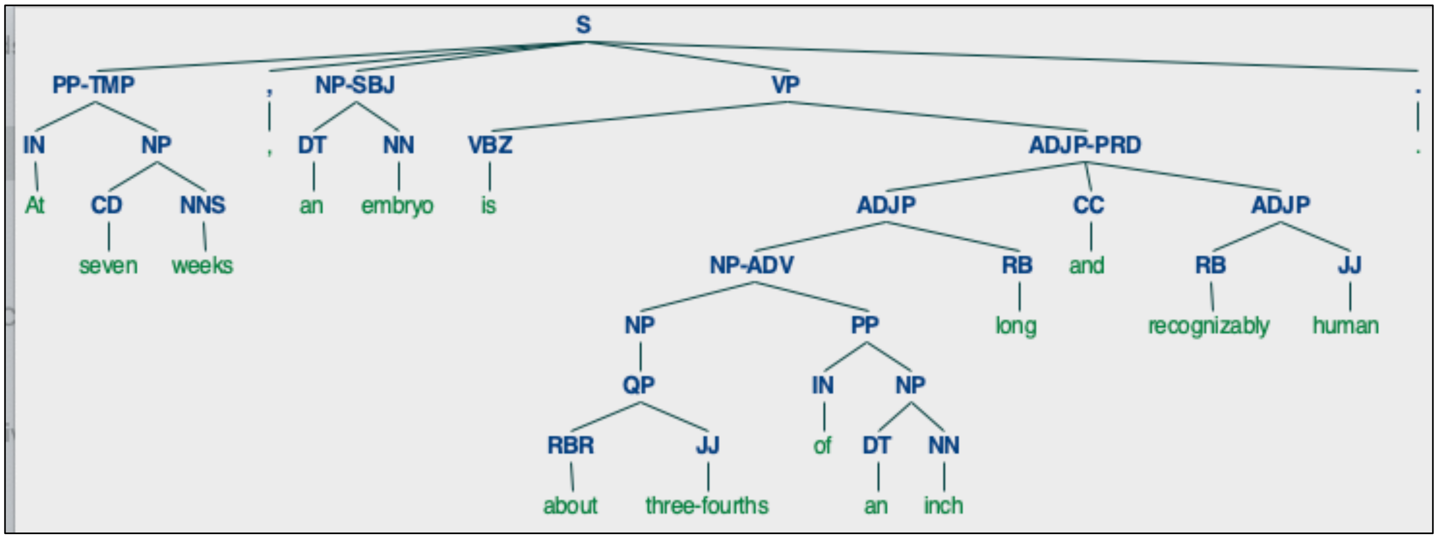
- Particle|RP
  - This category includes a numb er of mostly monosyllabic words that also double as prepositions.
- Adverb, comparative|RBR
  - *closer, later, less, more*, *further* – see previous slide

```
>>> d['about']
['IN', 'RB', 'RP', 'RBR', 'JJ']
>>> d['About']
['IN', 'RB']
```

Words with multiple POS tags

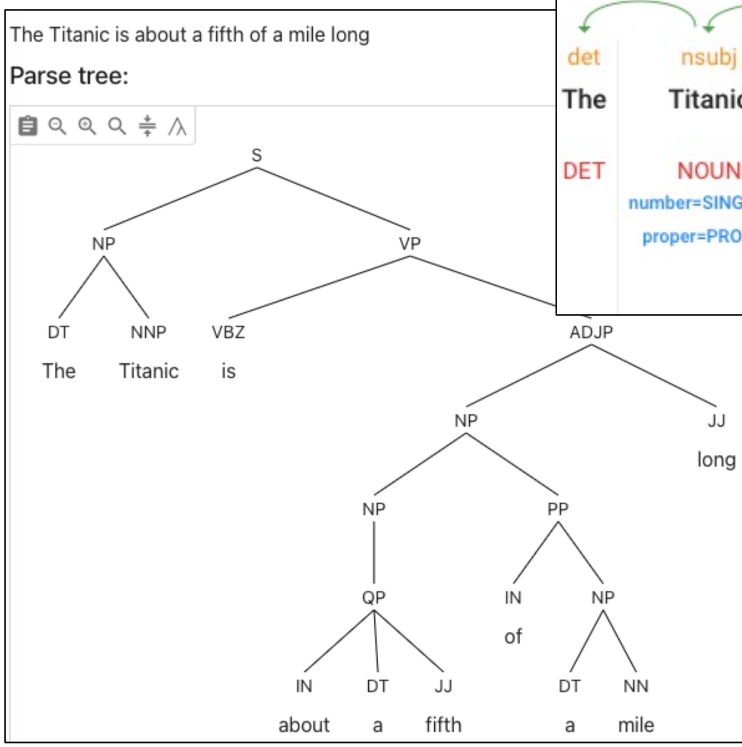
- *about*: https://www.merriam-webster.com/dictionary/about
  1. adverb: *about* a year ago
  2. adverb: looked about for a place to park
  3. adverb: They go about in circles.
  4. adverb: He spoke to the people standing about.
  5. adverb: the other way about
  6. preposition: People gathered about him
  7. preposition: He traveled about the country.
  8. preposition: Fish are abundant about the reefs.
  9. preposition: spoke *about* his past
  10. preposition: act as if they know what they're about
  11. adjective: is up and *about* by 7 a.m.
  12. adjective: There is a scarcity of jobs *about*.

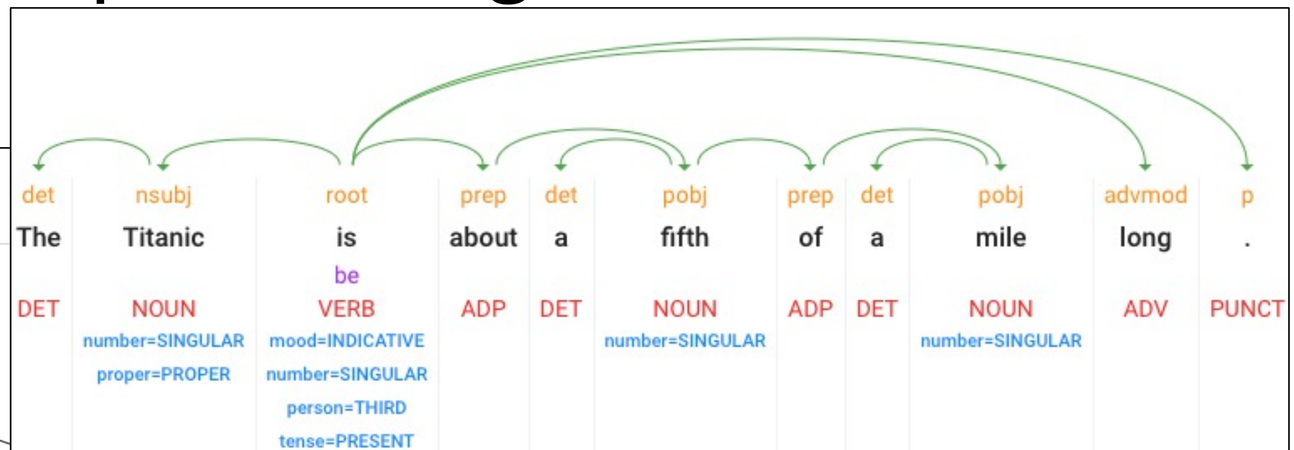
Words with multiple POS tags



- about = RBR?

Words with multiple POS tags





https://cloud.google.com/natural-language

https://parser.kitaev.io

Words with multiple POS tags

EXAMPLES: You should eat less/JJR (in terms of quantity).
(cf. You should eat less/JJR cheese.)

You should eat less/RBR (in terms of frequency).
(cf. You should eat rarely/RB.)

You should work less/RBR.
(cf. You should work harder/RBR.)

Less should be tagged as a comparative adjective (JJR) even when it occurs without a head noun, as in *less of a problem*.

Less in the sense of *minus* should be tagged as a coordinating conjunction (CC).

```
>>> d['less']
['RB', 'RBR', 'CC', 'NN', 'JJR', 'JJS']
>>> d['Less']
['RBR', 'NNP', 'JJR']
```

Words with multiple POS tags

JJ or NN

Nouns that are used as modifiers, whether in isolation or in sequences, should be tagged as nouns (NN, NNS) rather than as adjectives (JJ).

EXAMPLES:	wool/NN sweater (vs. woollen/JJ sweater)
	terminal/NN type (vs. terminal/JJ cancer)
	life/NN insurance/NN company

Hyphenated modifiers, on the other hand, should always be tagged as adjectives (JJ). Thus, we have different part-of-speech assignments in examples like the following—depending on the orthographic conventions used:

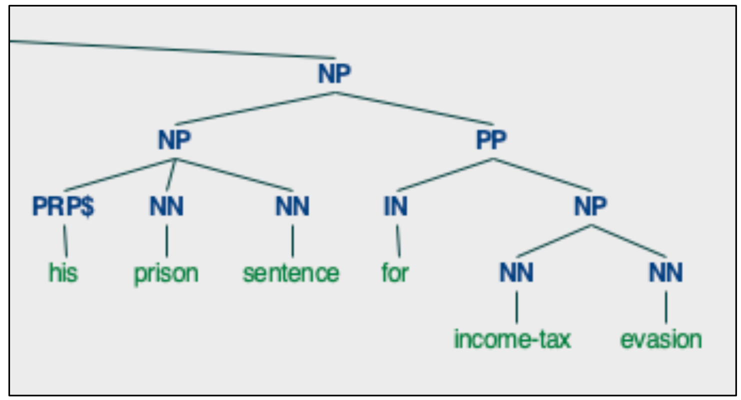
EXAMPLES:	income-tax/JJ return;	income/NN tax/NN return
	value-added/JJ tax;	value/NN added/VBN tax

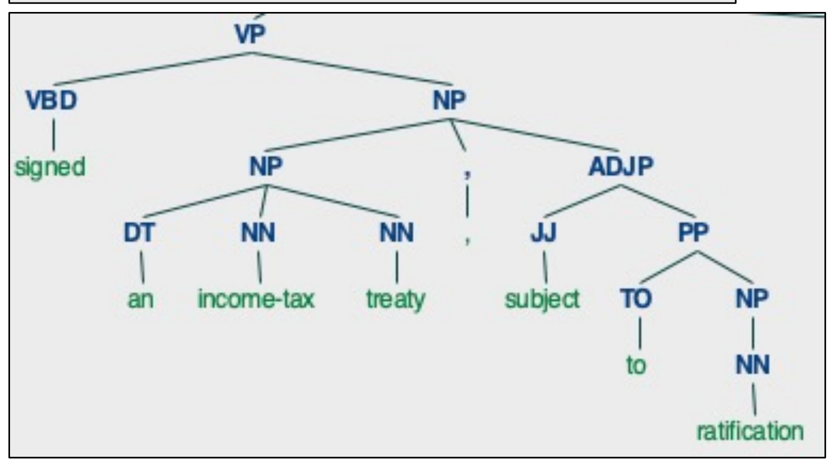
Words with multiple POS tags

```
>>> d['wool']
['NN']
>>> d['terminal']
['JJ', 'NN']
>>> d['woollen']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'woollen'
>>> d['Wool']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Wool'
```

Words with multiple POS tags

```
>>> d['income-tax']
['JJ', 'NN']
>>> d['income']
['NN']
>>> d['tax']
['NN', 'VB']
```





Words with multiple POS tags

- Let's look at the frequency distribution by # of pos tags:
  - *recall our dictionary* d *maps words to pos tags*

```
>>> fd = nltk.FreqDist([len(d[k]) for k in d])

>>> fd.most_common()

[(1, 54075), (2, 7137), (3, 1588), (4, 188), (5, 60), (6, 20), (8, 3), (7, 2)]

>>> fd.plot()
```