Lecture 6: N-grams and Conditional Probability

Ling 1330/2330 Computational Linguistics
Na-Rae Han, 9/14/2023

Objectives

- Exercise #4 review
  - Unigram frequency
  - Bigram frequency
  - So many data objects!

- Bigrams vs. conditional probability

- NLTK
  - `nltk.ConditionalFreqDist`:

    ← Conditional frequency distribution

Exercise #4 review

- https://sites.pitt.edu/~naraehan/ling1330/ex4.html

- Many different data object types!

  ← You must keep close tabs.

1. raw text (`str` type)
2. word tokens (`list` type)
3. word types (either `list` or `set`)
4. word frequency distribution (`nltk.FreqDist`)
   - key: word, value: frequency count
5. bigrams (*generator* type, you can cast it into a `list`)
6. bigram frequency distribution (`nltk.FreqDist`)
   - key: (w1, w2), value: frequency count

Exercise #4

- Pickling. What is the point?

- Shell crashing! Squeezing! Best practices?
  - Edit out big flashed chunks from your shell file before submission along with errors that aren't helpful. Your submission is also your notes for future reference!

- This way or tokenizing is not ideal. Why?
  - `etoks = nltk.word_tokenize(etxt.lower())`

- Working with complex data types (bigrams in particular)

- Membership test and data type:
  - `x in list` vs. `x in set`

  🡐 One of them is much more efficient. Which?

- Surprise! Looping through (=list-comprehending) NLTK's FreqDist already follows a default order: from most frequent to least

```
>>> efreq['so']
    968
>>> sograms = [gram for gram in e2gramfd if gram[0]=='so']
>>> sorted(sograms, key=e2gramfd.get, reverse=True)[:10]
    [('so', 'much'), ('so', 'very'), ('so', ','), ('so', 'well'),
    ('so', 'many'), ('so', 'long'), ('so', '.'), ('so', 'little'),
    ('so', 'far'), ('so', 'i')]
>>> for gram in sograms[:10]:
...     print(gram, e2gramfd[gram])
...
    ('so', 'much') 98
    ('so', 'very') 83
    ('so', ',') 34
    ('so', 'well') 31
    ('so', 'many') 29
    ('so', 'long') 27
    ('so', '.') 21
    ('so', 'little') 20
    ('so', 'far') 19
    ('so', 'i') 18
>>> e2gramfd.freq(('so', 'well'))
    0.00016164354990092815
```

In the latest NLTK version, sograms is already sorted by frequency!

So, sorting is not necessary.
We can just use sograms[:10]

```
>>> efreq['so']
    968
>>> sograms = [gram for gram in e2gramfd if gram[0]=='so']
>>> sorted(sograms, key=e2gramfd.get, reverse=True)[:10]
    [('so', 'much'), ('so', 'very'), ('so', ','), ('so', 'well'),
    ('so', 'many'), ('so', 'long'), ('so', '.'), ('so', 'little'),
    ('so', 'far'), ('so', 'i')]
>>> for gram in sograms[:10]:
...     print(gram, e2gramfd[gram])
...
    ('so', 'much') 98
    ('so', 'very') 83
    ('so', ',') 34
    ('so', 'well') 31
    ('so', 'many') 29
    ('so', 'long') 27
    ('so', '.') 21
    ('so', 'little') 20
    ('so', 'far') 19
    ('so', 'i') 18
>>> e2gramfd.freq(('so', 'well'))
    0.00016164354990092815
```

Jane Austen just typed in 'so'.
What is the **probability of 'well' being her next word**?

This is **conditional probability**:
Condition: 'so'
Outcome: 'well'

Nope, this is not it.
(Why?)

```
>>> efreq['so']
    968
>>> sograms = [gram for gram in e2gramfd if gram[0]=='so']
>>> sorted(sograms, key=e2gramfd.get, reverse=True)[:10]
    [('so', 'much'), ('so', 'very'), ('so', ','), ('so', 'well'),
    ('so', 'many'), ('so', 'long'), ('so', '.'), ('so', 'little'),
    ('so', 'far'), ('so', 'i')]
>>> for gram in sograms[:10]:
...     print(gram, e2gramfd[gram])
...
    ('so', 'much') 98
    ('so', 'very') 83
    ('so', ',') 34
    ('so', 'well') 31
    ('so', 'many') 29
    ('so', 'long') 27
    ('so', '.') 21
    ('so', 'little') 20
    ('so', 'far') 19
    ('so', 'i') 18
>>> e2gramfd.freq(('so', 'well'))
    0.00016164354990092815
```

Jane Austen just typed in 'so'.
What is the **probability of 'well' being her next word**?

This is **conditional probability**:
Condition: 'so'
Outcome: 'well'

Answer:
31 / 968 = 0.032

nltk.ConditionalFreqDist

- Builds on FreqDist as a conditional frequency distribution.

```
>>> e2grams[-10:]
    [('fully', 'answered'), ('answered', 'in'), ('in', 'the'),
    ('the', 'perfect'), ('perfect', 'happiness'), ('happiness',
    'of'), ('of', 'the'), ('the', 'union'), ('union', '.'), ('.',
    'finis')]

>>> e2gramcfd = nltk.ConditionalFreqDist(e2grams)

>>> e2gramcfd['so']
    FreqDist({'much': 98, 'very': 83, ',': 34, 'well': 31, 'many':
    29, 'long': 27, '.': 21, 'little': 20, 'far': 19, 'i': 18, ...})

>>> e2gramcfd['so']['well']
    31
>>> e2gramcfd['so'].freq('well')
    0.03202479338842975
```

Builds from bigrams

Key: w1, Value: FreqDist of w2

Lookup w1, then w2 on returned FreqDist

Conditional probability of 'well' following 'so'

Bad weather vs. Pitt

- `ConditionalFreqDist`: its keys are "conditions", and values are their respective frequency distribution `FreqDist`.
- Built from a list of `(condition, outcome)` tuples.

```
>>> school = [('rain', 'open'), ('rain', 'open'), ('rain', 'open'),
    ('rain', 'open'), ('rain', 'closed'), ('snow', 'closed'), ('snow',
    'closed'), ('snow', 'open'), ('snow', 'open'), ('snow', 'closed'),
    ('blizzard', 'closed'), ('blizzard', 'closed')]
>>> school_cfd = nltk.ConditionalFreqDist(school)
>>> school_cfd.keys()
    dict_keys(['snow', 'blizzard', 'rain'])
>>> school_cfd.values()
    dict_values([FreqDist({'closed': 3, 'open': 2}), FreqDist({'closed':
    2}), FreqDist({'open': 4, 'closed': 1})])
>>> school_cfd.conditions()
    ['snow', 'blizzard', 'rain']
```

Bad weather vs. Pitt

```
>>> school_cfd['snow']
    FreqDist({'closed': 3, 'open': 2})
>>> school_cfd['snow']['closed']
    3
>>> school_cfd['snow']['open']
    2
>>> school_cfd['snow'].freq('open')
    0.4
>>> school_cfd['blizzard']
    FreqDist({'closed': 2})
>>> school_cfd['blizzard']['closed']
    2
>>> school_cfd['blizzard']['open']
    0
>>> school_cfd.tabulate()
              closed open
    blizzard     2    0
        rain     1    4
        snow     3    2
>>>
```

Conditional probability of Pitt opening (outcome) when it snows (condition)

A bit of background

- P(A): the probability of A occurring
  - P(snow): the probability of having a snowy weather.
- P(A|B): **Conditional probability**

  the probability of A occurring, given that B has occurred
  - P(close|snow): given a snowy weather, the probability of Pitt closing.
  - P(snow|close): given Pitt's closure, the probability of the day being snowy.
- P(A, B): **Joint probability**

  the probability of A occurring *and* B occurring
  - Same as P(B, A).
  - If A and B are independent events, same as P(A)*P(B).

    If not, same as P(A|B)*P(B) and also P(B|A)*P(A).
  - P(close, snow): the probability of Pitt closing and the weather being snowy.

bigram FD vs. CFD: very different!

```
>>> e2grams[-10:]
    [('fully', 'answered'), ('answered', 'in'), ('in', 'the'), ('the',
    'perfect'), ('perfect', 'happiness'), ('happiness', 'of'), ('of', 'the'),
    ('the', 'union'), ('union', '.'), ('.', 'finis')]

>>> e2gramfd = nltk.FreqDist(e2grams)
>>> e2gramfd[('so', 'well')]
    31
>>> e2gramfd.freq(('so', 'well'))
    0.00016165113599032l8

>>> e2gramcfd = nltk.ConditionalFreqDist(e2grams)
>>> e2gramcfd['so']
    FreqDist({'much': 98, 'very': 83, ',': 34, 'well': 31, 'many': 29, 'long':
    27, '.': 21, 'little': 20, 'far': 19, 'i': 18, ...})
>>> e2gramcfd['so']['well']
    31
>>> e2gramcfd['so'].freq('well')
    0.03202479338842975
```

Made from the same bigrams as input, but returns different data objects

It's important you keep tabs on many data objects and their meaning!

bigram FD vs. CFD: Practice

10 minutes

```
>>> e2grams[-10:]
    [('fully', 'answered'), ('answered', 'in'), ('in', 'the'), ('the',
    'perfect'), ('perfect', 'happiness'), ('happiness', 'of'), ('of', 'the'),
    ('the', 'union'), ('union', '.'), ('.', 'finis')]

>>> e2gramfd = nltk.FreqDist(e2grams)
>>> e2gramfd[('so', 'well')]
    31
>>> e2gramfd.freq(('so', 'well'))
    0.0001616511359903218

>>> e2gramcfd = nltk.ConditionalFreqDist(e2grams)
>>> e2gramcfd['so']
    FreqDist({'much': 98, 'very': 83, ',': 34, 'well': 31, 'many': 29, 'long':
    27, '.': 21, 'little': 20, 'far': 19, 'i': 18, ...})
>>> e2gramcfd['so']['well']
    31
>>> e2gramcfd['so'].freq('well')
    0.03202479338842975
```

Poke your object in shell to understand its structure!

10 minutes

- FD vs. CFD practice
- CFD with **trigrams**!
  - How to build?
  - What are top words following 'so well'? How about 'of the'?
- Fun with ENABLE words
  - No vowels? Palindromes? Anagrams of 'stop'?
  - How many potential answers for Wordle?

← **Saved SHELL session posted** next to the lecture PDF!

Where are we on the NLTK Book?

- Ch.1 Language Processing and Python
  - https://www.nltk.org/book/ch01.html
  - NLTK built-in functions for exploring text, Python basics
- Ch.2 Accessing Corpora and Lexical Resources
  - https://www.nltk.org/book/ch02.html
  - A tour of various NLTK-loaded corpora and resources
- Ch.3 Processing Raw Text
  - https://www.nltk.org/book/ch03.html
  - Basic text processing pipeline – tokenization, etc.
  - Also: regular expressions

Wrap-up

- Homework #2 out
  - START EARLY! Get help earlier.
- Next class (Tue):
  - N-gram language models
- Review the NLTK Book, chapters 1 through 3.