Introduction to English Linguistics

13: Natural Language Processing

P. S. Langeslag

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN IN PUBLICA COMMODA
SEIT 1737

Concepts

String

A sequence of text as typically assigned to a (constant or) variable:

```
>>> phrase = 'the tortoise and the hare'
>>> print(phrase)
'the tortoise and the hare'
```

String

A sequence of text as typically assigned to a (constant or) variable:

```
>>> phrase = 'the tortoise and the hare'
>>> print(phrase)
'the tortoise and the hare'
```

In Python, any string is also a list of characters:

```
>>> if 'i' in phrase:
>>>     phrase.index('i')
9
```

String

A sequence of text as typically assigned to a (constant or) variable:

```
>>> phrase = 'the tortoise and the hare'
>>> print(phrase)
'the tortoise and the hare'
```

In Python, any string is also a list of characters:

```
>>> if 'i' in phrase:
>>>     phrase.index('i')
9
```

Python counts indices (but not tallies) from `0`:

```
>>> phrase[1]
'h'
```

Language Arithmetic

```
>>> from collections import Counter
>>> phrase = 'the tortoise and the hare'
>>> Counter(phrase)
Counter({'t': 4, 'e': 4, ' ': 4, 'h': 3, 'o': 2, 'r': 2, 'a': 2, 'i': 1,
's': 1, 'n': 1, 'd': 1})
```

Scrabble Design Made Easy

(See `frequency.py`)

Scrabble Design Made Easy

(See `frequency.py`)

Pop quiz: why is <p> such a rare letter in Old English?

What About Boggle?

(See `oedistribution.py`)

(Word) Token

When reading a text (corpus) sequentially, each instance of a word form you encounter is its own token. We may write or employ functions to **tokenize** a text:

```
>>> phrase = 'the tortoise and the hare'
>>> tokens = phrase.split()
>>> tokens
['the', 'tortoise', 'and', 'the', 'hare']
```

(Word) Token

When reading a text (corpus) sequentially, each instance of a word form you encounter is its own token. We may write or employ functions to **tokenize** a text:

```
>>> phrase = 'the tortoise and the hare'
>>> tokens = phrase.split()
>>> tokens
['the', 'tortoise', 'and', 'the', 'hare']

>>> len(tokens)
5
>>> tokens.sort()
>>> tokens
['and', 'hare', 'the', 'the', 'tortoise']
```

(Word) Token

When reading a text (corpus) sequentially, each instance of a word form you encounter is its own token. We may write or employ functions to **tokenize** a text:

```
>>> phrase = 'the tortoise and the hare'
>>> tokens = phrase.split()
>>> tokens
['the', 'tortoise', 'and', 'the', 'hare']

>>> len(tokens)
5
>>> tokens.sort()
>>> tokens
['and', 'hare', 'the', 'the', 'tortoise']
```

Bag of Words

A model storing information on each word type (i.e. form) and its frequency in a text (corpus), but discarding syntax and word order.

```
>>> Counter(tokens)
Counter({'the': 2, 'tortoise': 1, 'and': 1, 'hare': 1})
```

Term; or (Word) Type

Distinct orthographical form (i.e. spelling) in the corpus.

```
>>> tokens = ['the', 'tortoise', 'and', 'the', 'hare']
>>> terms = list(dict.fromkeys(tokens))
>>> len(terms)
4
>>> terms
['the', 'tortoise', 'and', 'hare']
```

Zipf's Law

A word token's frequency in a natural corpus *f(r)* is inversely proportional to its rank (*r*) in the word frequency table.

Zipf's Law

A word token's frequency in a natural corpus *f(r)* is inversely proportional to its rank (*r*) in the word frequency table.

$$f(r) \propto \frac{1}{(r+\beta)^{\alpha}}$$

where $\alpha \approx 1$ and $\beta \approx 2.7$

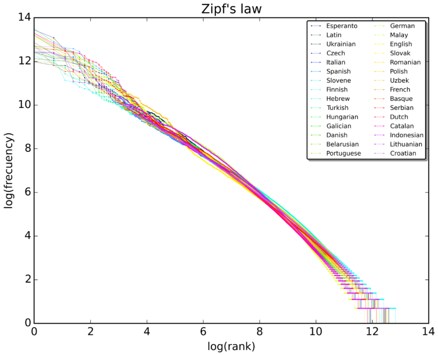


Figure 1: Frequency/rank log plot for the first 10 mln words in 30 Wikipedias (CC-BY-SA Sergio Jimenez)

Zipf's Law in Natural Languages

Brown Corpus tallies (from Lane, *Natural Language Processing in Action*, p. 87):

1. the: 69971
2. of: 36412
3. and: 28853
4. to: 26158
5. a: 23195
6. in: 21337
7. that: 10594
8. is: 10109
9. was: 9815
10. he: 9548

(etc.)

What's the Use of Zipf's Law in Natural Languages?

- **Topic modelling:** we know what a document is about not by finding the most frequent words, but by finding the words that transgress Zipf's Law the most (**TF-IDF**).
- This is how search engines work!

Stem

Linguistic Definition

The base of a given word form, to which inflectional information is added.

Stem

Linguistic Definition

The base of a given word form, to which inflectional information is added.

NLP Definition

The base to which a given type may be reduced ("stemming") by stripping away (known) inflectional (and sometimes derivational) information, whether or not the resulting form is linguistically recognized.

```
>>> import re
>>> sentence = 'Jael rushed hurtling down the stairs'
>>> tokens = sentence.split()
>>> pattern = '(s|ing|ed)$'
>>> stems = [re.sub(pattern, '', token) for token in tokens]
>>> stems
['Jael', 'rush', 'hurtl', 'down', 'the', 'stair']
```

Lemma

Linguistic Definition

Dictionary headword

Lemma

Linguistic Definition

Dictionary headword

NLP Definition

Unique identifier to which inflected forms of the same word may be assigned

n-Gram

A sequence consisting of n words as they occur in a string of text.

n-Gram

A sequence consisting of *n* words as they occur in a string of text.

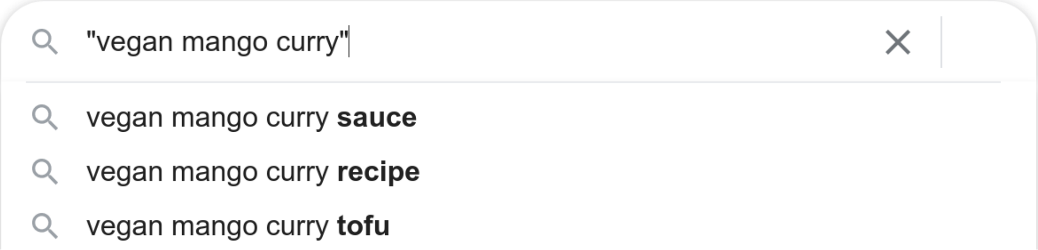


Figure 2: Double quotes yield *n*-grams on most search engines

n-Gram

A sequence consisting of n words as they occur in a string of text.

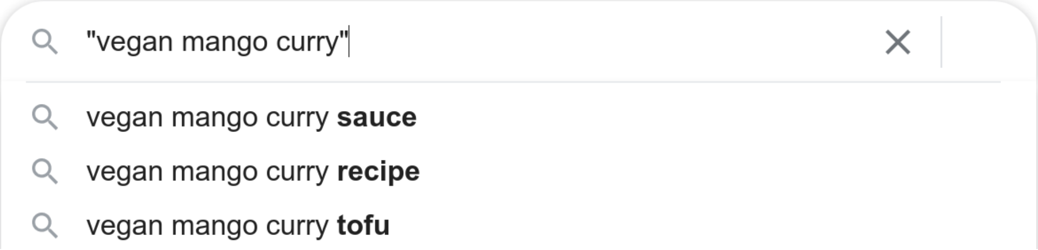
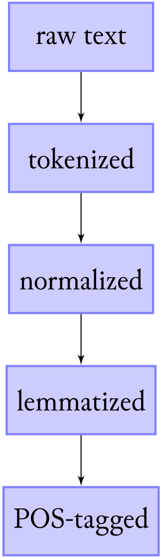


Figure 2: Double quotes yield n-grams on most search engines

- we speak of bigrams and trigrams but commonly write 2-gram, 3-gram
- n-grams offer the benefit of **dimension reduction** but also improve lexical precision.

Pipeline



The processing sequence from input to the desired structured data.

Google Books Ngram Viewer

What Is Google Books?

- Began in 2002
- Went live in 2004
- Aims to digitize large numbers of books
- Upwards of 25 million books scanned before they went quiet about progress
- Met with a great deal of litigation (notably Author's Guild and the American Association of Publishers)
- The project has slowed down since c. 2012 (but Ngram data set updated in 2024)
- Official (but dated) history page reads "we're not done—not until all of the books in the world can be found by everyone, everywhere, at any time they need them."

What Is the Value of Equipping Google Books with an *n*-gram Reader?

- The largest searchable corpus of print works and ebooks in the history of the world
- Historical value: quantify the historical use of concepts
- Linguistic value: quantify the historical use of words, phrases, spellings
  - Greatly facilitates *OED* attestation research!
- Methodology: sensible combination of word types and lemmatization

Demonstration

books.google.com/ngrams

Algorithm

Any unigram is scored against the full corpus of unigrams for the chosen language corpus;

Any bigram is scored against the full corpus of bigrams for the chosen language corpus.

Thus a graph plotting a unigram and a bigram is not, strictly speaking, a comparison.

Usage (1/2)

- Enter comma-separated queries to see them plotted against each other
- A wildcard (*) returns the top ten matches e.g. `the weather is *`
- `gram_INF` returns inflected forms of a lexical form `gram` e.g. `seek_INF` returns *sought*, *seek*, *seeking*, *seeks*
- `gram_NOUN`, `gram_VERB`, etc. tries to return only the matching part of speech e.g. `feast_VERB` should not find a hit in the sequence “a feast”
- `gram_*` plots all parts of speech for that form against each other e.g. `feast_*` returns the noun *feast*, the verb *feast*, the adjective *feast*, and some noise
- Parts of speech on their own return any match e.g. `kiss _PRON_ mother` should return “kiss your mother,” “kiss my mother,” etc., but plotted as a single function;
- Parts of speech preceded by a wildcard are separated out into different matches e.g. `kiss *_PRON mother` should return separate statistics on each of “kiss your mother,” “kiss my mother,” etc.

Usage (2/2)

- Sentence boundaries: `_START_` / `_END_`
- Dependency relations: `weather=>fair,weather=>beautiful,weather=>nice`
- Combined plots: `+`, e.g. `(ale + lager + beer)`
- Subtracted plots: `-`, e.g. `(ale + lager + beer) - (sparkly + sparkly wine + champagne)`
- Divided plots: `/`, e.g. `beer / wine`
- Multiplied plots: `*`, e.g. `fish,(wallaby * 1000)`
- Plots from multiple corpora: `:`, e.g. `wizard:eng,wizard:eng_fiction`
- Syntactic "root": `_ROOT_`, e.g. `_ROOT_=>eat` to return clauses with *eat* as the finite verb

Limitations

- Skewed corpus (synchronically)
  - Scientific literature overrepresented (e.g. “Figure” vs “figure”)
- Difference in skew over time
  - Early corpus skews towards religion, late corpus towards science
- Disregards print run/readership
- OCR errors
  - f vs ſ
- Not representative or reliable prior to c. 1800

Bibliography

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. Sebastopol, CA: O'Reilly, 2009. https://www.nltk.org/book/.

Younes, Nadja, and Ulf-Dietrich Reips. "Guideline for Improving the Reliability of Google Ngram Studies: Evidence from Religious Terms." *PLoS ONE* 14 (March 22, 2019). https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0213554.

Zhang, Sarah. "The Pitfalls of Using Google Ngram to Study Language." *Wired*, October 12, 2015. https://www.wired.com/2015/10/pitfalls-of-studying-language-with-google-ngram/.