# ECE 20875
# Python for Data Science

**Chris Brinton and David Inouye**

## n-grams and basic natural language processing

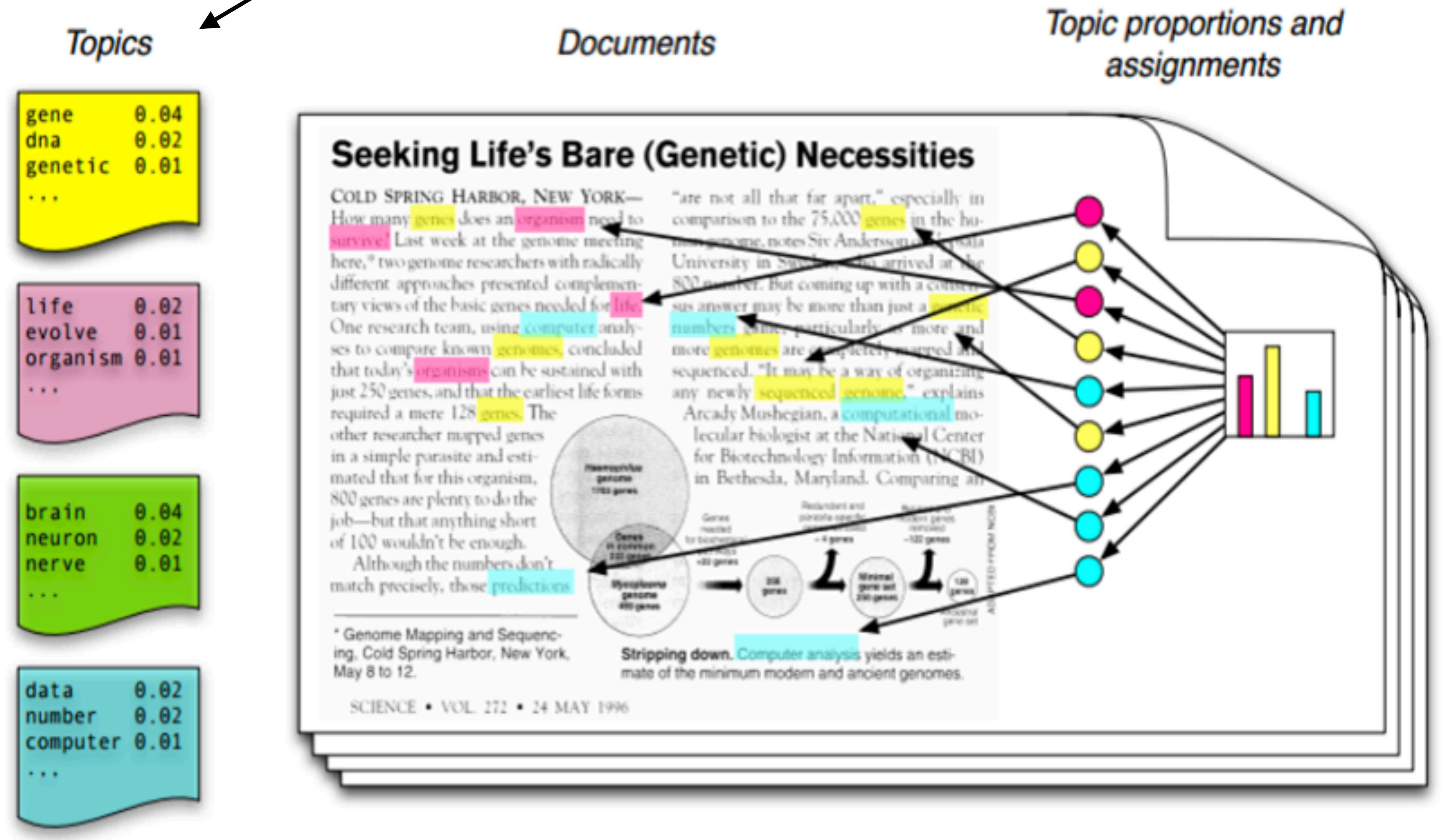
# text data analysis

- Written text is often treated as a form of data for analysis
- Some types of analyses:
    - Measuring similarity between **documents**
    - Extracting **topics** from documents
    - Finding the most frequently occurring words
    - Quantifying the importance of **phrases**
- Most of these involve breaking up documents into words or “*n*-grams”

*Popular example: Latent Dirichlet Allocation (LDA)*

*documents*: combinations of *topics*

*topics*: combinations of *words*



# n-grams

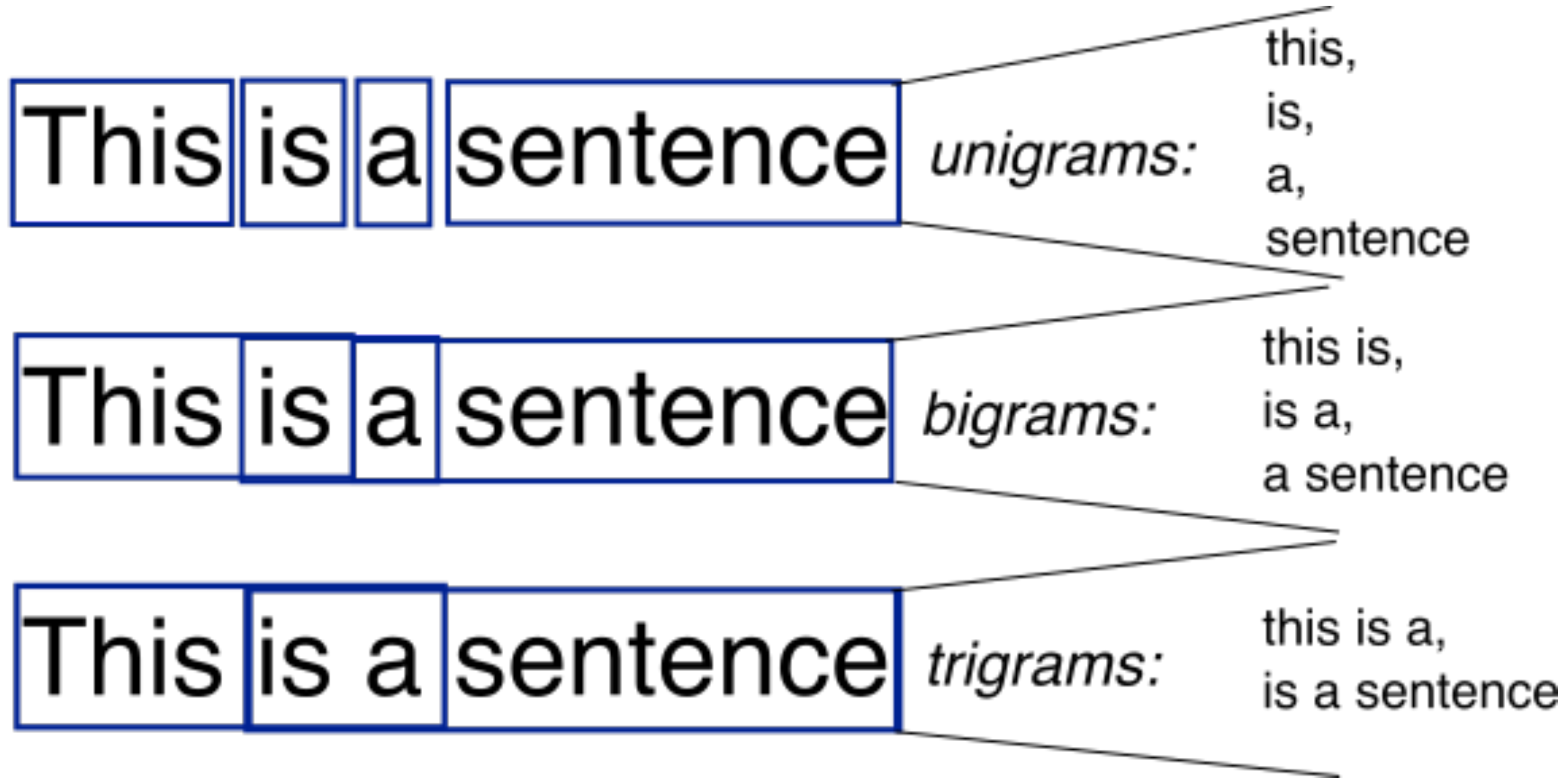
- ***n*-grams** break up a sentence into overlapping **subsequences** of length *n*
  - *n* typically refers to words or characters (though it could also be e.g., syllables)
  - Unigrams (*n*=1), bigrams (*n*=2), trigrams (*n*=3), …
- Consider the string: `“I saw a cat”`
  - Word-based 3-grams:

    `“I saw a”, “saw a cat”`

  - Character-based 3-grams:

    `“I␣s”, “␣sa”, “saw”, “aw␣”, “w␣a”, “␣a␣”, “a␣c”, “␣ca”, “cat”`

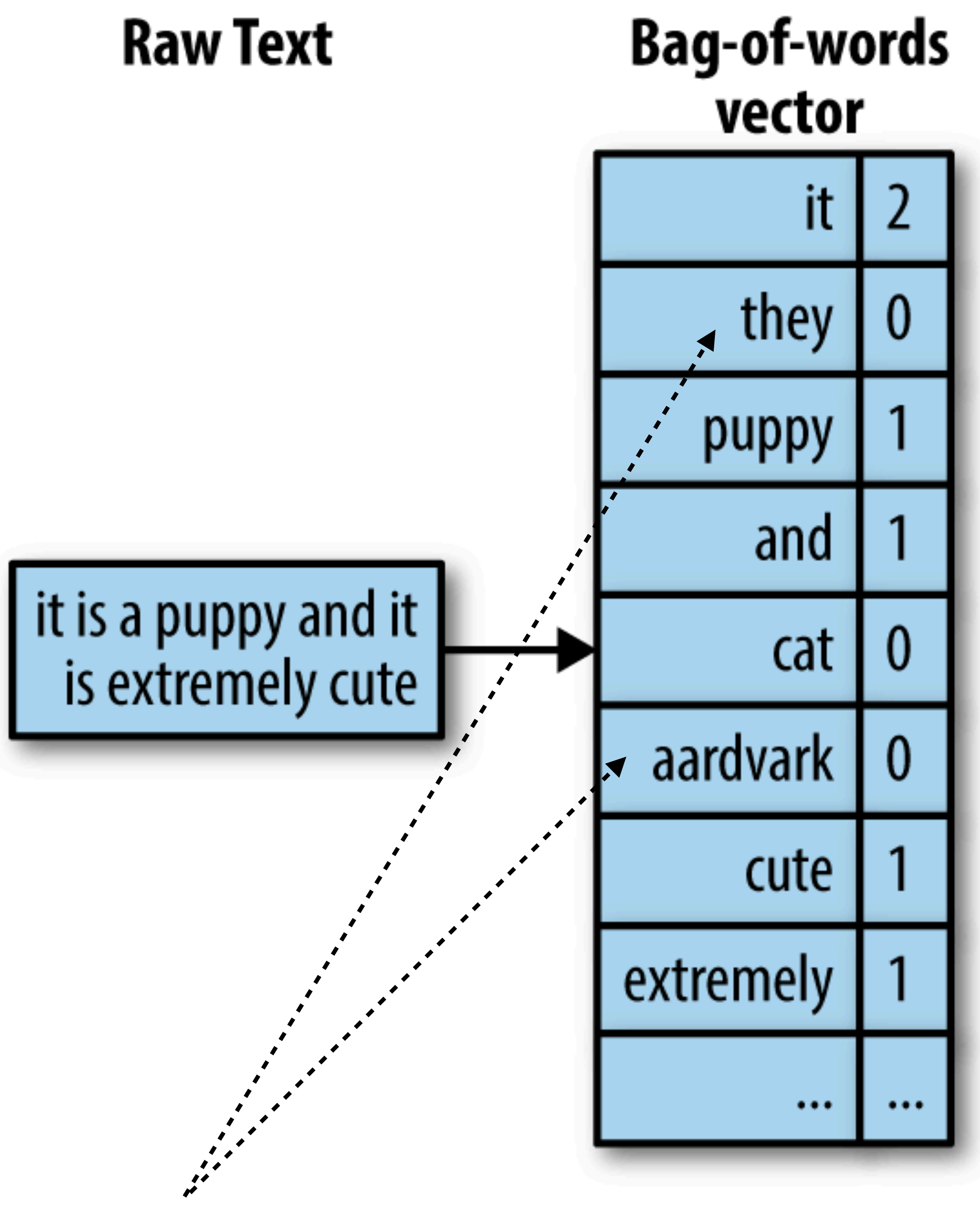
*word-based n-gram extraction*



# bag-of-words

- The same *n*-gram can appear multiple times in a string
    - This indicates a higher frequency
- Generally we only care about order *within* an n-gram, not *between* n-grams
- **Bag-of-words** model: Order between words (more generally, between *n*-grams) in a document is not considered
    - We call it “bag-of-words,” but it’s really “bag-of-*n*-grams”
- For example, consider this string: “`wan can cup`”
    - bag-of-words of character-based 3-grams:

```
wan : 1   an␣ : 2   n␣c : 2
␣ca : 1   can : 1   ␣cu : 1   cup : 1
```




- Where would the 0s come from?
- We often compare documents by their bag-of-words representations

# language classification

- Consider the commonly encountered **language classification** problem, i.e., identifying the language in which a document is written

- We could consider the *n*-grams of characters contained in the document

  - Documents written in a particular language will tend to have similar *n*-gram frequencies (e.g., “the” in English vs. “el” in Spanish)

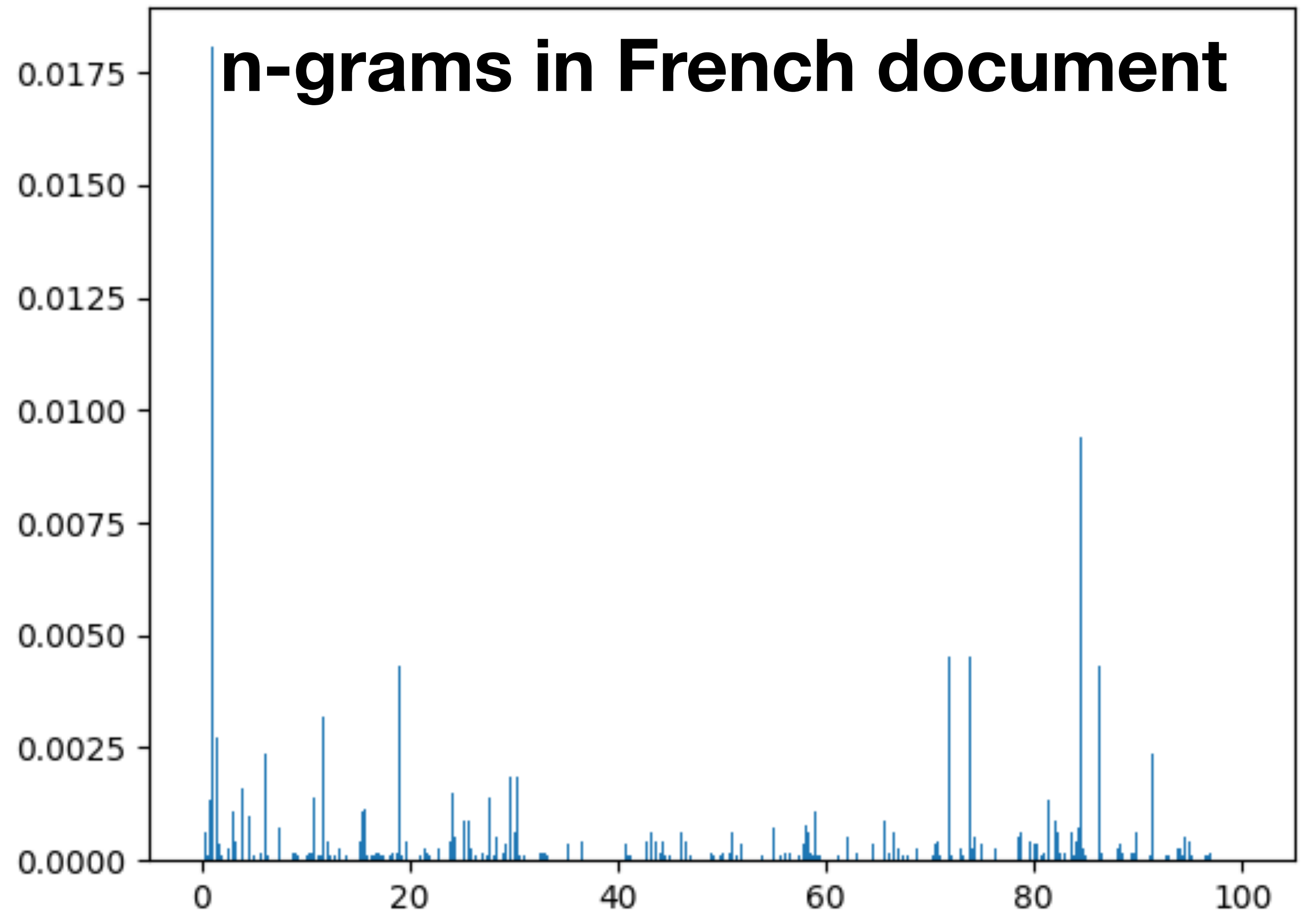
  - We can compare a document of interest to known *n*-gram language frequencies

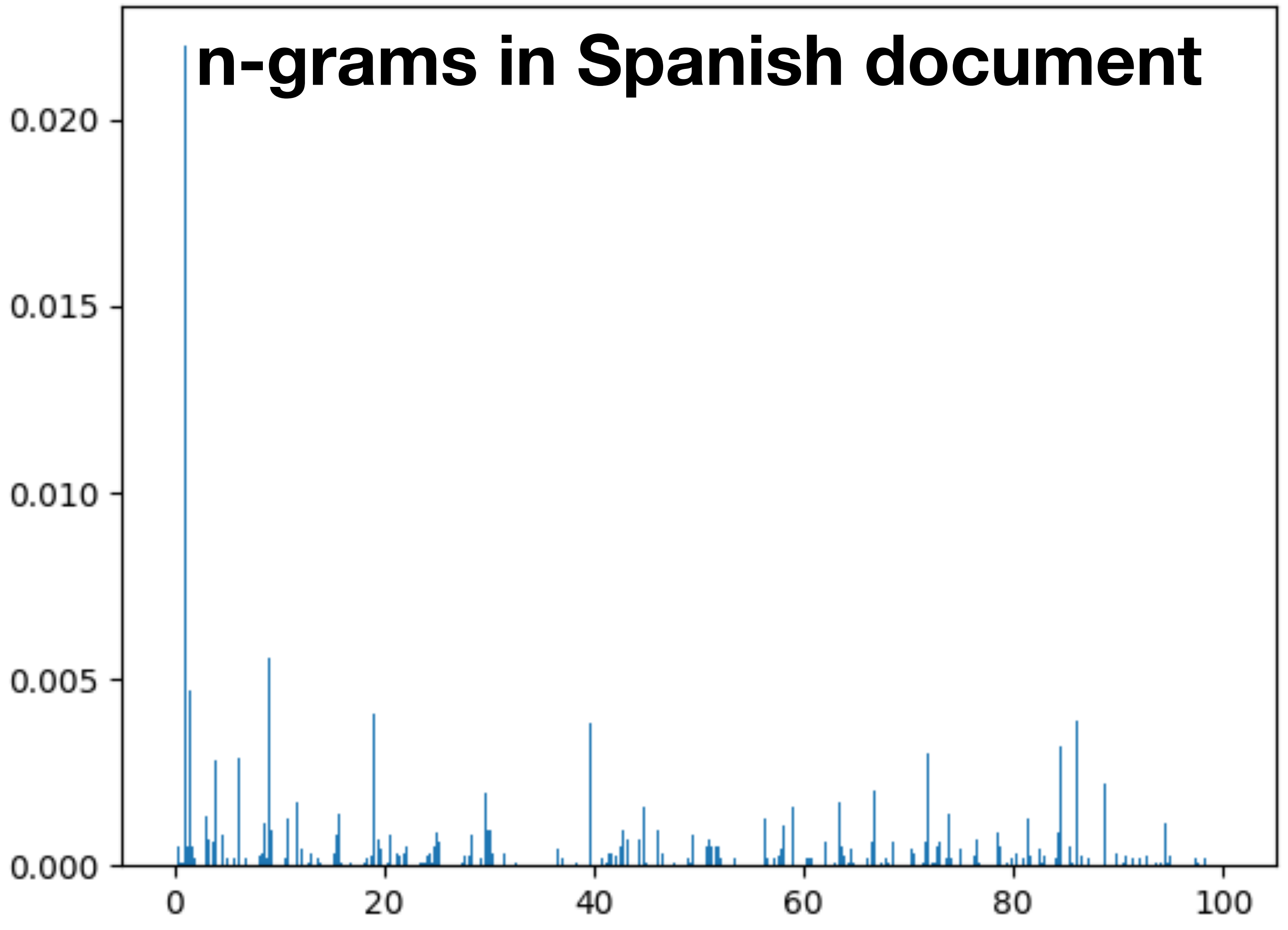
| unigram | | bigram | | trigram | |
|---|---|---|---|---|---|
| e | 12.6% | th | 3.9% | the | 3.5% |
| t | 9.1% | he | 3.7% | and | 1.6% |
| a | 8.0% | in | 2.3% | ing | 1.1% |
| o | 7.6% | er | 2.2% | her | 0.8% |
| i | 6.9% | an | 2.1% | hat | 0.7% |
| n | 6.9% | re | 1.7% | his | 0.6% |
| s | 6.3% | nd | 1.6% | tha | 0.6% |
| h | 6.2% | on | 1.4% | ere | 0.6% |
| ... | | ... | | ... | |

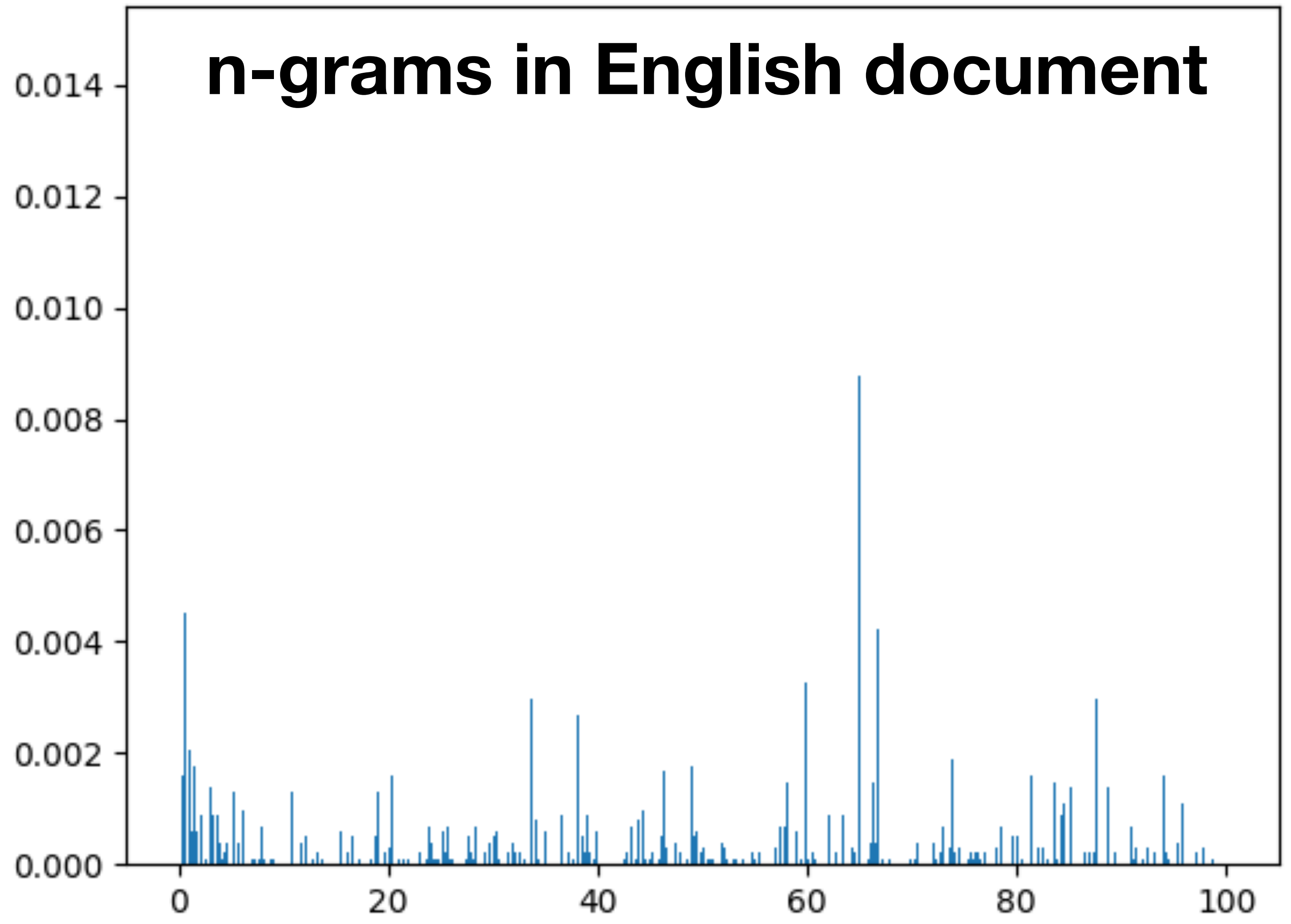
- Can visualize this by building a histogram of the *n*-grams

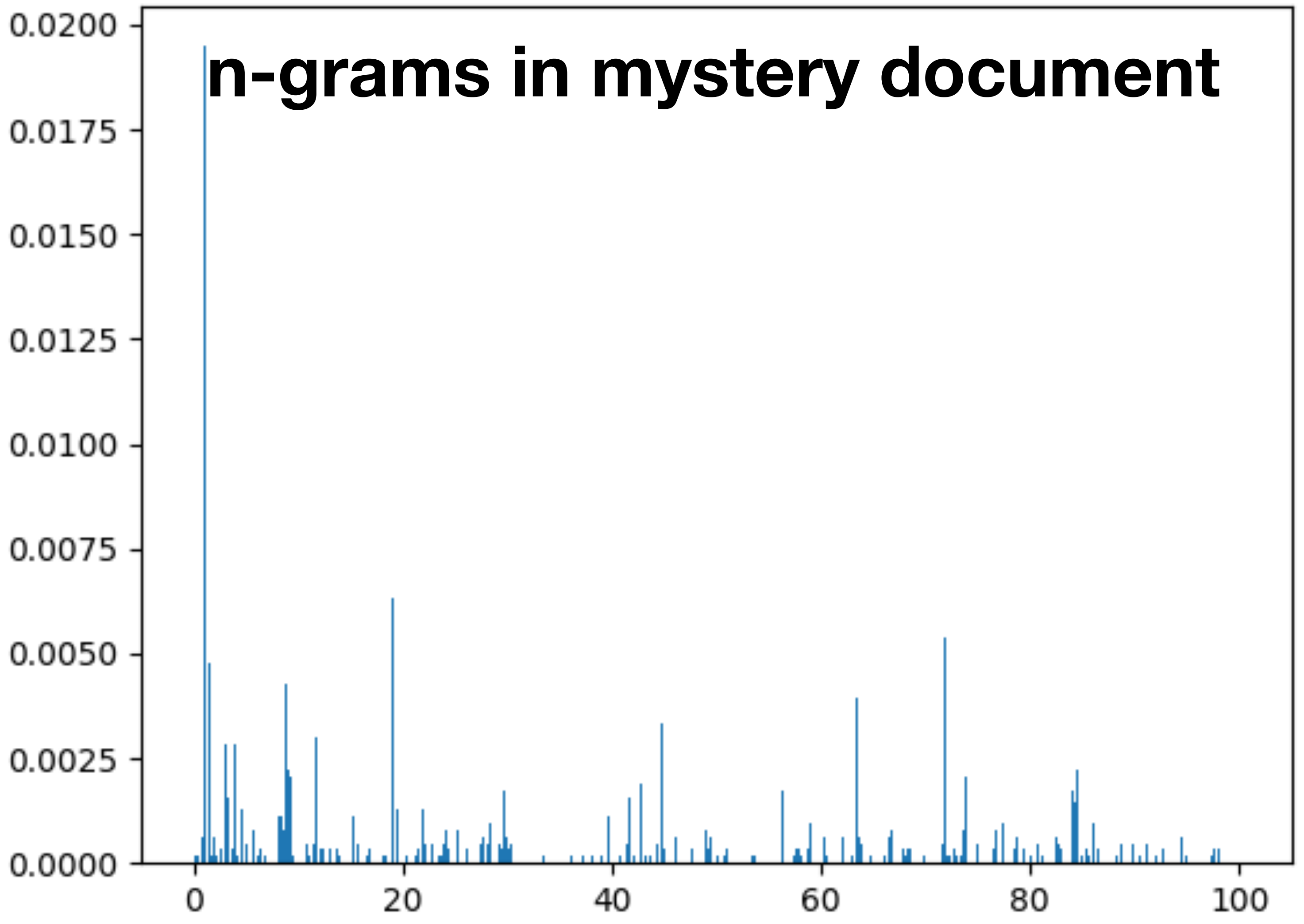
  - Treat each *n*-gram across the documents as a separate (categorical) bucket

# n-gram histogram examples



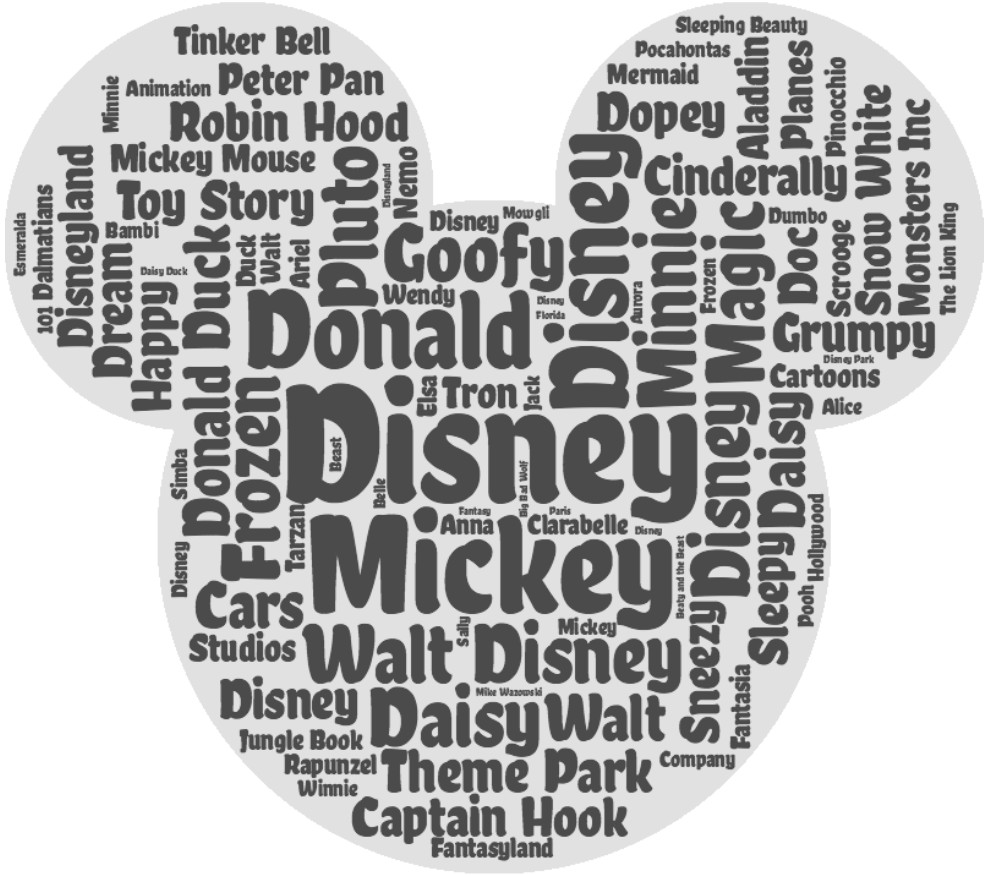




- How would we quantify which language is "closest" to the mystery document?
- We could use the MSE between the *n*-gram vectors

# n-gram importance

- How do we quantify the *importance* of an *n*-gram in a document?




- One possibility: Count the number of times it occurs, i.e., its **frequency**
    - More frequently occurring should be more important
- But what about common words like “a”, “as”, “is”, …?
    - These specific examples are **stopwords**, which we should probably remove from the analysis of “importance” anyway
    - But many high frequency non-stopwords will not provide much information in a given context (e.g., “`Disney`” in a collection of documents about “`Disney World`”)
- Need to somehow measure how “unique” the *n*-gram is across documents

# tf-idf score

- A statistic that quantifies this intuition is the **term frequency-inverse document frequency** or **tf-idf** score

  - One of the most popular schemes used today

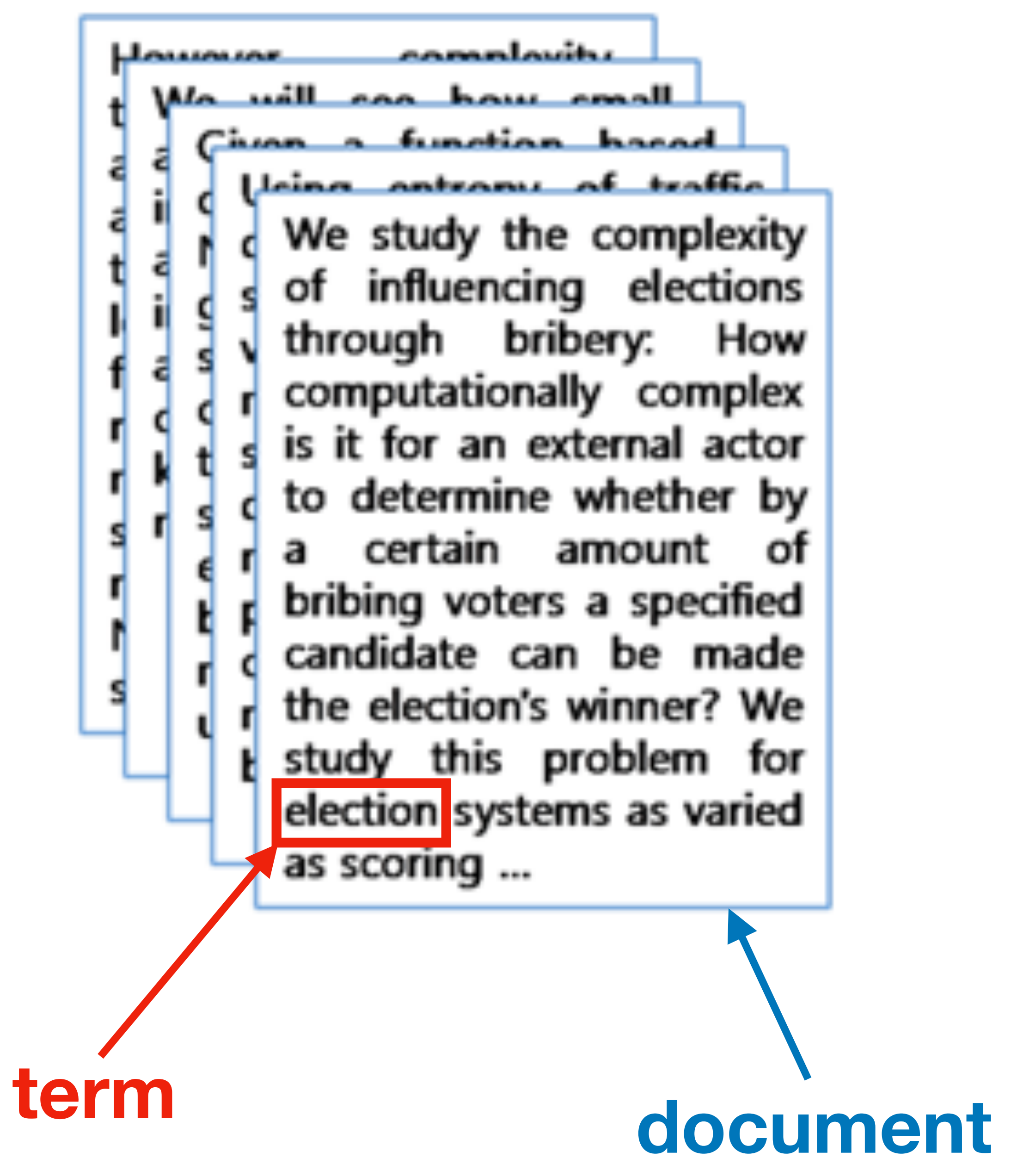
  - Let $t$ be a term ($n$-gram), $d$ be a document, and $D$ be a **corpus** (collection of documents) under consideration

  - The tf-idf score of term $t$ in document $d$ with respect to corpus $D$ is

$$\texttt{tfidf}(t, d, D) = \texttt{tf}(t, d) \cdot \texttt{idf}(t, D)$$

- Many different methods for quantifying `tf` and `idf`

**Corpus**




**Here we we will assume terms are words, but more generally they can be n-grams**

# tf-idf score

- Term frequency $\texttt{tf}(t, d)$: Typically the fraction of terms in document $d$ which are term $t$
    - Letting $f_{t,d}$ be the number of occurrences of $t$ in $d$,

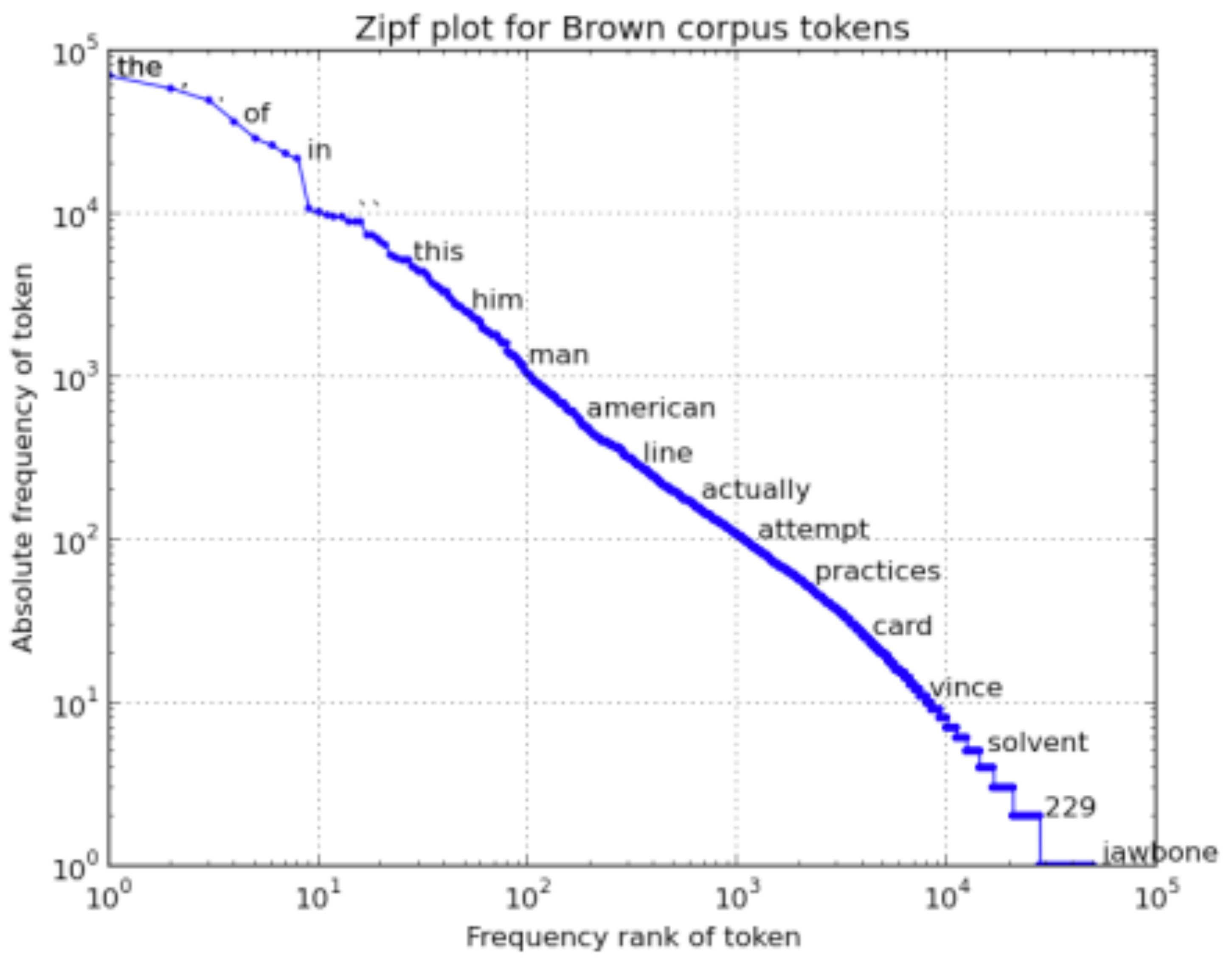
$$\texttt{tf}(t, d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}}$$

| Word | TF | | IDF | TF*IDF | |
|---|---|---|---|---|---|
| | A | B | | A | B |
| The | 1/7 | 1/7 | log(2/2) = 0 | 0 | 0 |
| Car | 1/7 | 0 | log(2/1) = 0.3 | 0.043 | 0 |
| Truck | 0 | 1/7 | log(2/1) = 0.3 | 0 | 0.043 |
| Is | 1/7 | 1/7 | log(2/2) = 0 | 0 | 0 |
| Driven | 1/7 | 1/7 | log(2/2) = 0 | 0 | 0 |
| On | 1/7 | 1/7 | log(2/2) = 0 | 0 | 0 |
| The | 1/7 | 1/7 | log(2/2) = 0 | 0 | 0 |
| Road | 1/7 | 0 | log(2/1) = 0.3 | 0.043 | 0 |
| Highway | 0 | 1/7 | log(2/1) = 0.3 | 0 | 0.043 |

- Inverse document frequency $\texttt{idf}(t, D)$: A measure of how rare term $t$ is across the corpus $D$ (i.e., how much information it provides about a document it appears in)
    - Letting $N = |D|$ be the number of documents in the corpus and $n_t$ be the number of documents where $t$ occurs, it is typically quantified as

$$\texttt{idf}(t, D) = \log_{10}\left(\frac{n_t}{N}\right)^{-1} = \log_{10}\frac{N}{n_t}$$

**Why log?**



# example

Dataset: Take the following four strings to be (very small) documents comprising a (very small) corpus:

1. “`The sky is blue.`”
2. “`The sun is bright today.`”
3. “`The sun in the sky is bright.`”
4. “`We can see the shining sun, the bright sun.`”

Task: Filter out obvious stopwords, and determine the tf-idf scores of each term in each document.

# solution

- After stopword filtering: (1) “`sky blue`”, (2) “`sun bright today`”, (3) “`sun sky bright`”, (4) “`can see shining sun bright sun`”

- TF: Find doc-word matrix, then normalize rows to sum to 1

$$f_{t,d}$$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **2** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| **3** | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| **4** | 0 | 1 | 1 | 1 | 1 | 0 | 2 | 0 |

$$\mathtt{tf}(t,d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}}$$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| **1** | 1/2 | 0 | 0 | 0 | 0 | 1/2 | 0 | 0 |
| **2** | 0 | 1/3 | 0 | 0 | 0 | 0 | 1/3 | 1/3 |
| **3** | 0 | 1/3 | 0 | 0 | 0 | 1/3 | 1/3 | 0 |
| **4** | 0 | 1/6 | 1/6 | 1/6 | 1/6 | 0 | 1/3 | 0 |

# solution

- IDF: Find number of documents each word occurs in, then compute formula

$$f_{t,d}$$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 1 | 1 | 1 | 0 | 2 | 0 |
| n_t | **1** | **3** | **1** | **1** | **1** | **2** | **3** | **1** |

$$N = 4$$

$$\texttt{idf}(t, D) = \log_{10} \frac{N}{n_t}$$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| | 0.602 | 0.125 | 0.602 | 0.602 | 0.602 | 0.301 | 0.125 | 0.602 |

$$\log_{10} \frac{4}{1} = 0.602 \qquad \log_{10} \frac{4}{3} = 0.125$$

# solution

$\mathtt{tf}(t, d)$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| 1 | 1/2 | 0 | 0 | 0 | 0 | 1/2 | 0 | 0 |
| 2 | 0 | 1/3 | 0 | 0 | 0 | 0 | 1/3 | 1/3 |
| 3 | 0 | 1/3 | 0 | 0 | 0 | 1/3 | 1/3 | 0 |
| 4 | 0 | 1/6 | 1/6 | 1/6 | 1/6 | 0 | 1/3 | 0 |

x

$\mathtt{idf}(t, D)$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| | 0.602 | 0.125 | 0.602 | 0.602 | 0.602 | 0.301 | 0.125 | 0.602 |

- TF-IDF: Multiply TF and IDF scores, use to rank importance of words within documents
  - Most important word for each document is highlighted

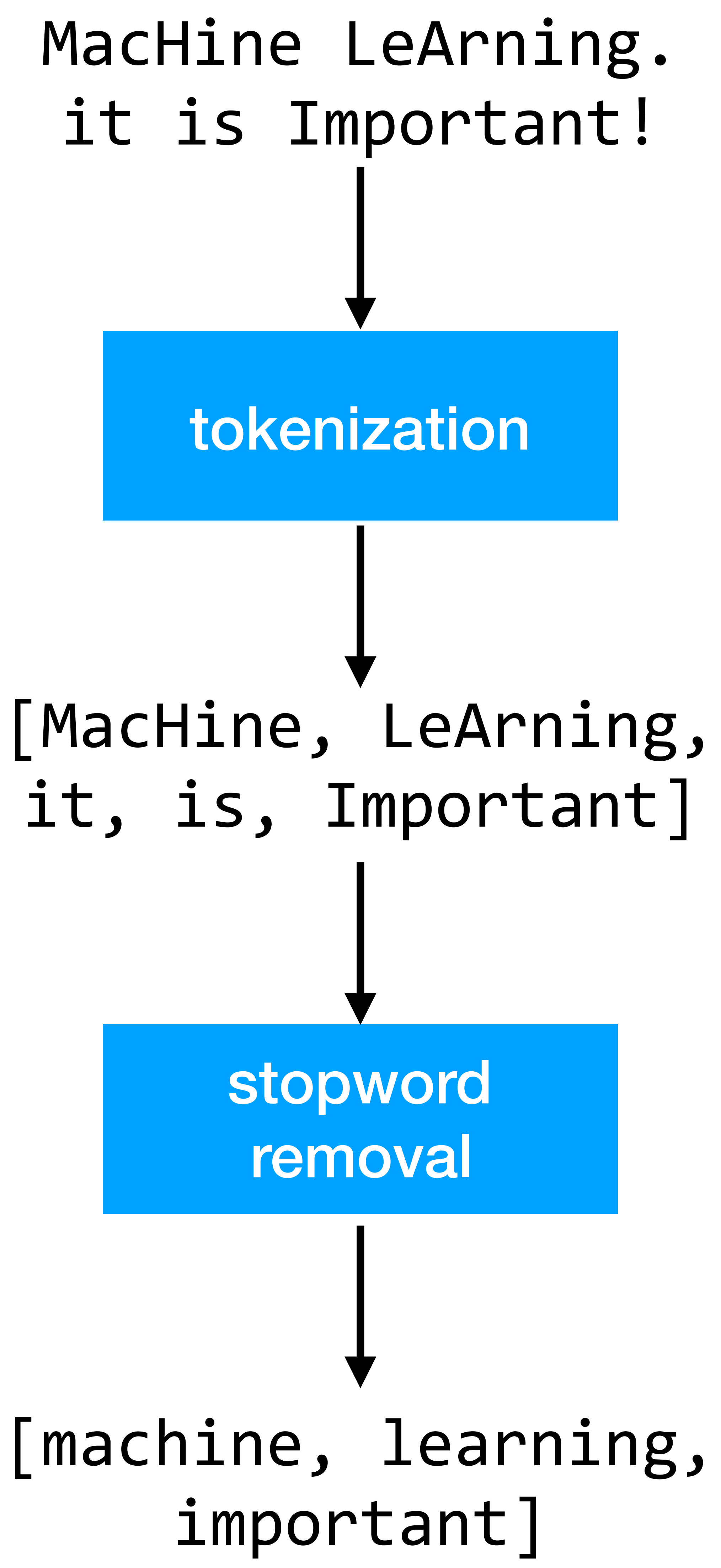
$$\mathtt{tfidf}(t, d, D) = \mathtt{tf}(t, d) \cdot \mathtt{idf}(t, D)$$

| | blue | bright | can | see | shining | sky | sun | today |
|---|---|---|---|---|---|---|---|---|
| 1 | **0.301** | 0 | 0 | 0 | 0 | 0.151 | 0 | 0 |
| 2 | 0 | 0.0417 | 0 | 0 | 0 | 0 | 0.0417 | **0.201** |
| 3 | 0 | 0.0417 | 0 | 0 | 0 | **0.100** | 0.0417 | 0 |
| 4 | 0 | 0.0209 | **0.100** | **0.100** | **0.100** | 0 | 0.0417 | 0 |

# text preprocessing

- Typically apply a series of preprocessing steps prior to analysis
  - Mostly using Python's `nltk` (natural language processing toolkit) library

1. **Tokenization**
   - Break text into tokens, e.g., n-grams of words (`nltk.word_tokenize(string)` or `string.split()`)
   - Remove non-word characters, e.g., punctuation
2. **Stopword removal**
   - Make words lowercase (`s.lower()`)
   - Remove common word tokens (`stopwords.words('english')`)



# text preprocessing

3. **Stemming / Lemmatizing**

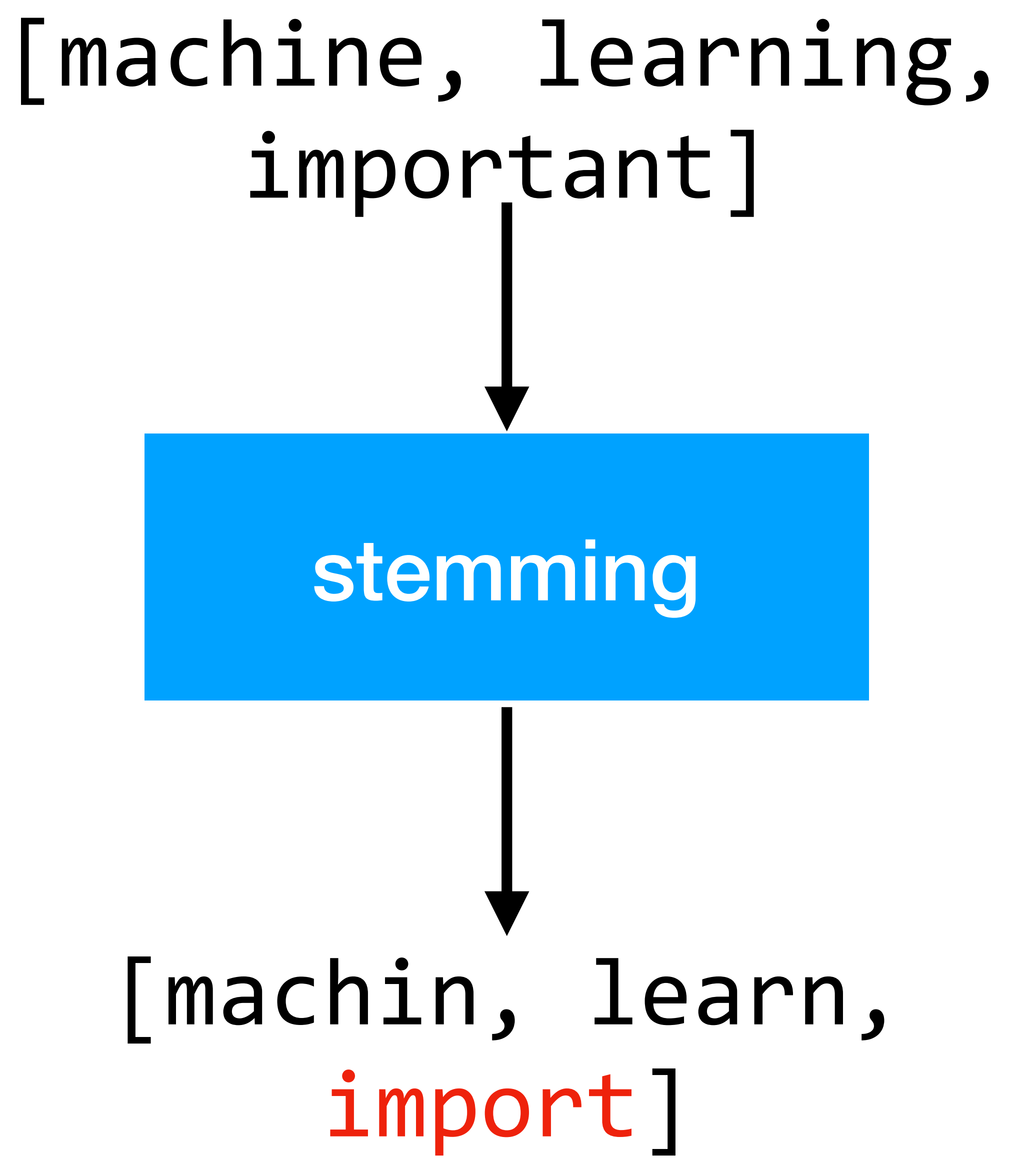
- **Stemming** reduces inflected words to their word stem (e.g., `studies, studying -> studi`)

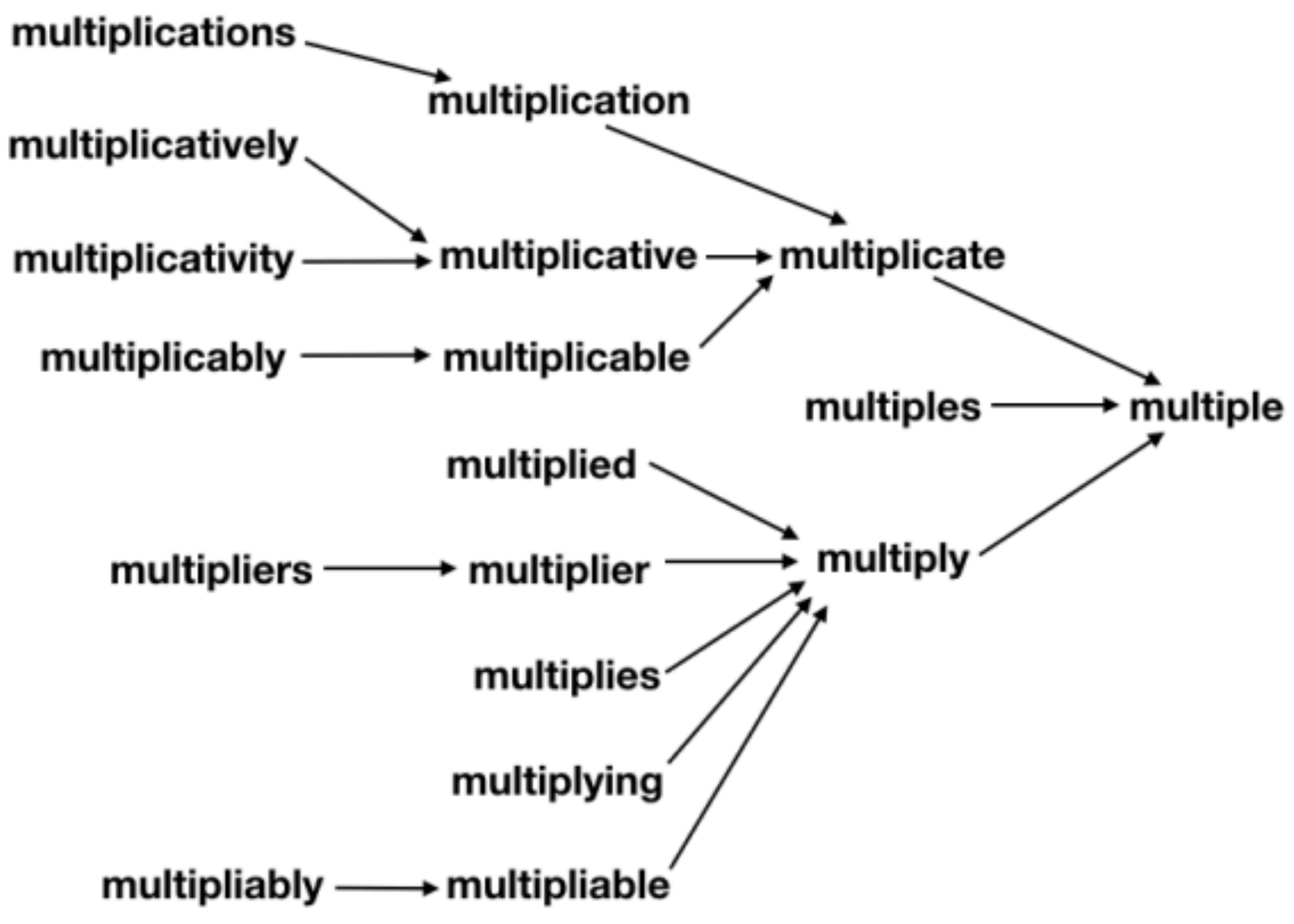
- **Lemmatization** maps words to their dictionary form, representing them as words (e.g., `studies, studying –> study`)

    - Requires part-of-speech (POS) specification

- Lemmatization is more complex (we need to tag a words POS to get the right result), but preferred when possible (e.g., on the right, the stemmed version of `important` is `import`)

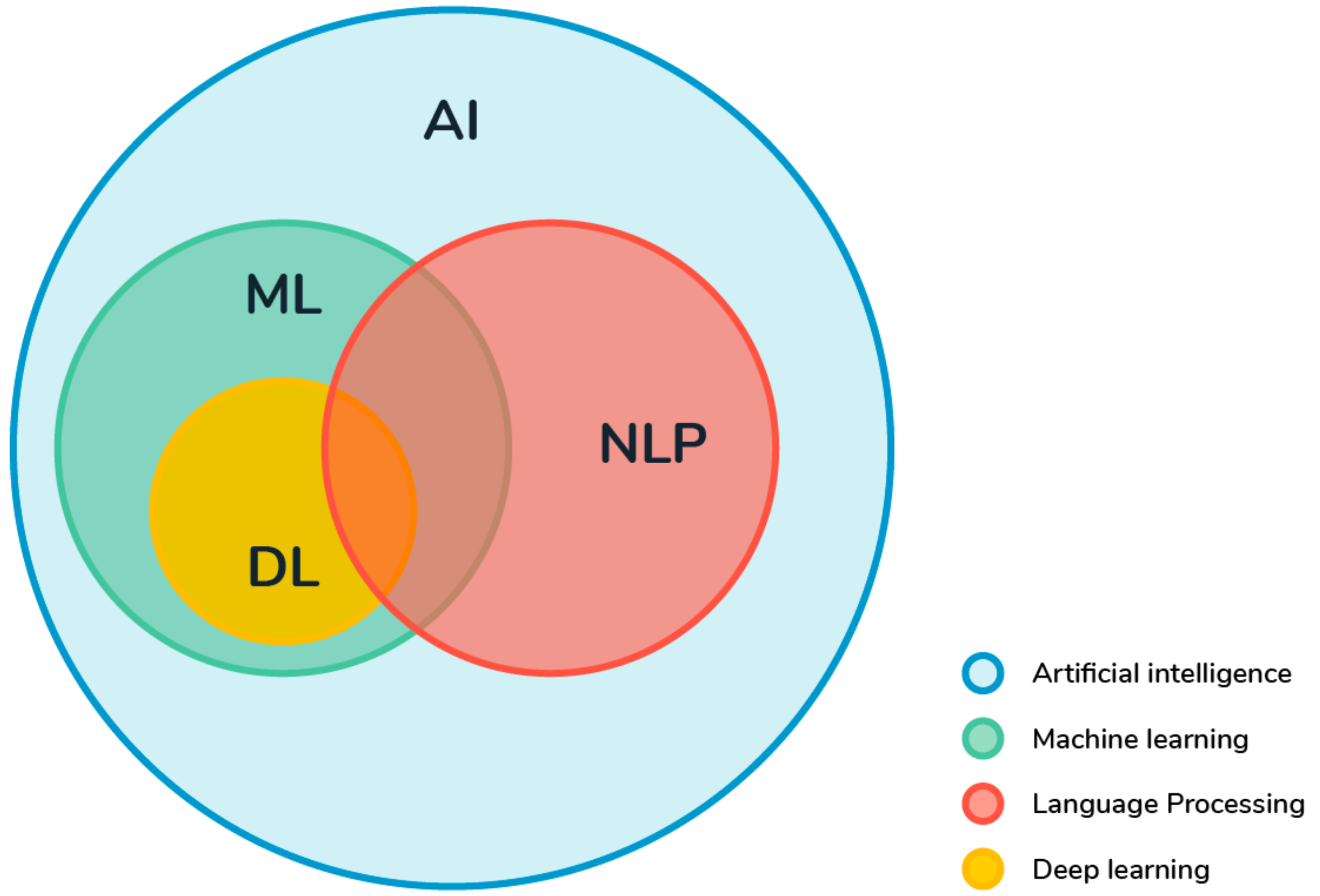
- `from nltk.stem import PorterStemmer, WordNetLemmatizer`



# natural language processing

- What we have been studying are specific methods in **natural language processing**, or **NLP**
  - NLP is concerned with how to automatically analyze large corpuses of text
- Two main classes of NLP: rules-based and statistical
  - tf-idf is a simple (yet widely used) statistical technique
  - Today's innovations are largely in the statistical category, leveraging *machine learning*
- Key is building **knowledge representations**



# natural language processing

- Some common functions of NLP
  - **Machine translation:** Translating between languages (e.g., Google translate)
  - **Speech recognition:** Determine the textual representation of an audio track (e.g., Siri)
  - **Document summarization:** Determine an effective summary of a document (e.g., Watson)
- All of these are constantly being innovated with new NLP algorithms