Homework 1 Reminders

- Due next week: Weds Sept 13 at 11:59pm
- Please ensure you have working python environment **this week**
- TA office hours posted on Blackboard
- Repeat submissions accepted on Blackboard; latest submission will be graded
- Blackboard Discussion groups available for questions

ILLINOIS INSTITUTE OF TECHNOLOGY

Lexical representations for NLP

CS-585

Natural Language Processing

Sonjia Waxmonsky

Based on slides from Derrick Higgins, Kai Shu

VECTOR SPACE MODEL OF TEXT

Vector Space Model for Text

A simple example:

- Sentence 0: "My dog sleeps"
- Sentence 1: "Your cat sleeps"
- Sentence 2: "That dog chases any cat"
- Vocab: `["cat","chases","dog","eats","sleeps"]`

Representing in vector space:

- Sentence 0: `[0,0,1,0,1]`
- Sentence 1: `[1,0,0,0,1]`
- Sentence 2: `[1,1,1,0,0]`

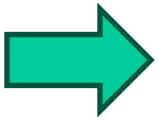
Vector Space Model for Text

A simple example:

- Sentence 0: "My dog sleeps"
- Sentence 1: "Your cat sleeps"
- Sentence 2: "That dog chases any cat"
- Vocabulary `["cat","chases","dog","eats","sleeps"]`

Representing in vector space:

- Sentence 0: `[0,0,1,0,1]`
- Sentence 1: `[1,0,0,0,1]`
- Sentence 2: `[1,1,1,0,0]`



cat	chases	dog	eats	sleeps
0	0	1	0	1
1	0	0	0	1
1	1	1	0	0

3 x 5 matrix
(Sentence x Vocab)

Scalars, Vectors, Matrices and Tensors

- Scalars are the numbers we know and love.

$$a = 1$$
$$b = e$$
$$c = -0.3$$

- Vectors are arrays of numbers – elements of $\mathbb{R}^n$
- They are typically written in a column (column vector)

$$\vec{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

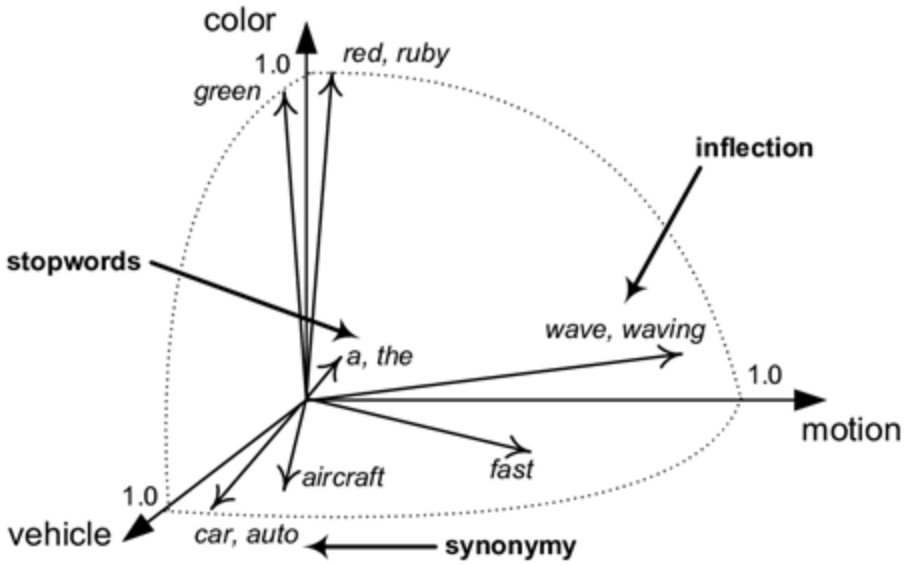
Slide from Lecture 2

Vector Space Model for Text

[Notebook]

Vector Space Model for Text

- Vector space model for **semantic** representations
  - Words and documents are represented as vectors in a **high-dimensional** space
  - Dimension may be as large as the vocabulary size (hundreds of thousands of dimensions or more)
  - Meaning corresponds to direction in the space



Polyvyanyy & Kuropka, *A Quantitative Evaluation of the Enhanced Topic-based Vector Space Model*

One-hot encoding of words

- A simple starting point for encoding words is to treat them as "one-hot" vectors in a space with one dimension for each word in the vocabulary (all zeros, except for a one in a specific location)
- This is a limited representation, because all words in the space are orthogonal to one another, but useful for building up larger units
- E.g.,

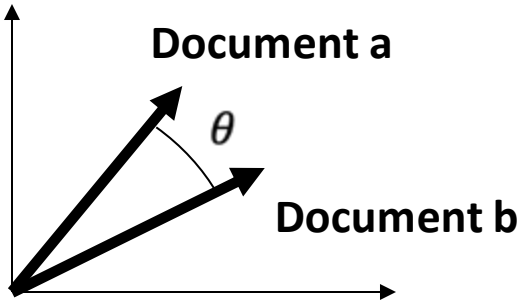
$$\text{cat} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \text{dog} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \text{mouse} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \text{bird} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Bag-of-words representation

- Given a vector representation of words, we can create a representation for ***documents*** in the same space (with one dimension per word, indicating the contribution of each word to the document's meaning)
- We call this a "bag of words" representation, because it treats document meaning solely as a function of the words used and their **counts**
- This is a **limited** representation of ***meaning***
  - Obviously, "Liverpool defeats Chelsea" means something different than "Chelsea defeats Liverpool"
- But it can be effective for many tasks

Cosine similarity

- The typical way of assessing the similarity between vector representations of documents is by looking at the angle between them
- The cosine of that angle is a convenient measure, since it is 0 when they are orthogonal (unrelated) and 1 when they point in the same direction
- Recall that for vectors *a* and *b*, the cosine of the angle **θ** between them is given by $\cos\theta = \frac{a \cdot b}{\|a\|_2 \|b\|_2} = \frac{a \cdot b}{\sqrt{a \cdot a \times b \cdot b}}$



Other similarity measures

- Euclidean:

$$\|a-b\|_2 = \sqrt{(a-b)\cdot(a-b)} = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

- Manhattan:

$$\|a-b\|_1 = \sqrt{\sum_{i=1}^{n}|a_i - b_i|}$$

VOCABULARY SELECTION

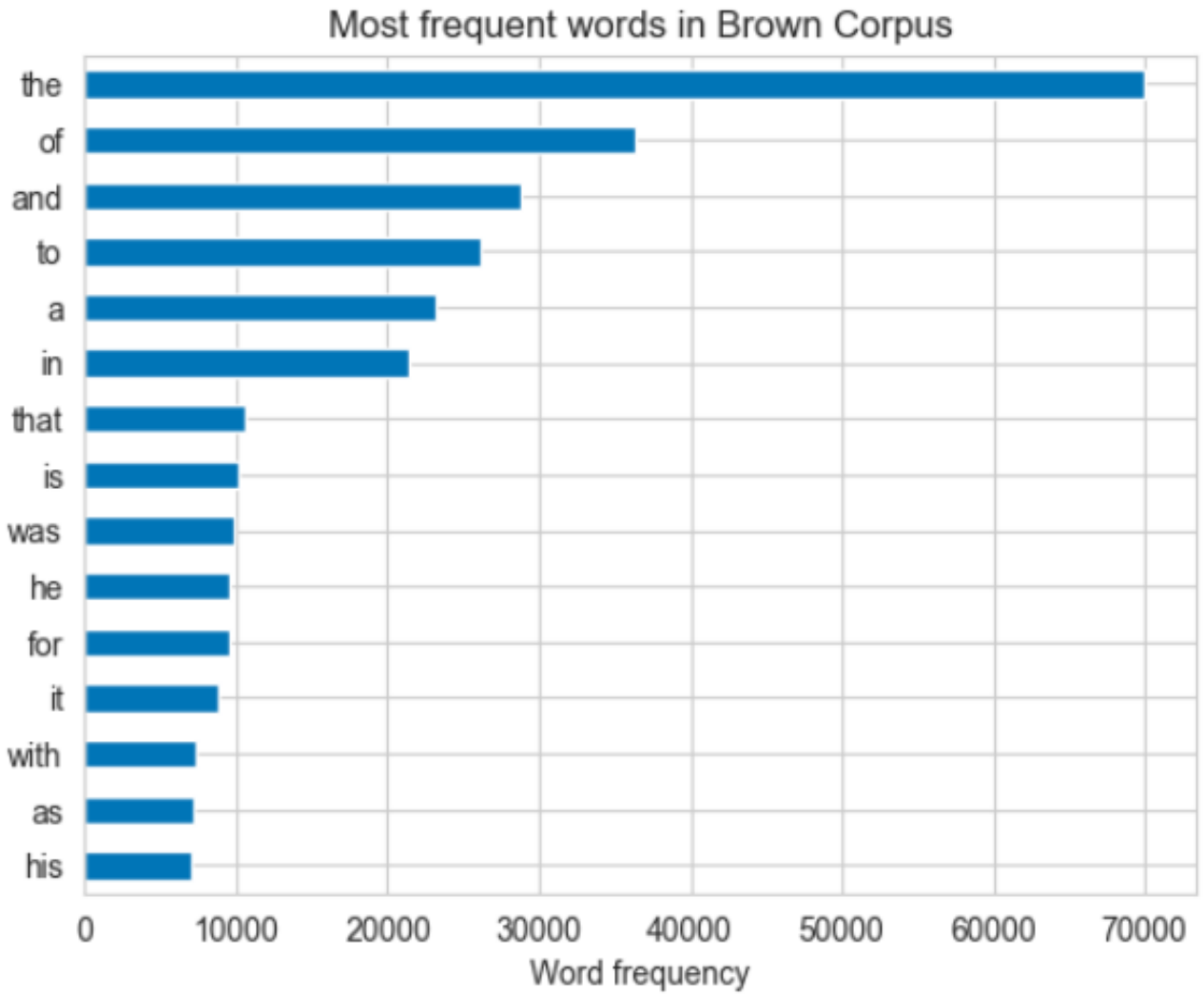
Vocabulary definition

- You've decided on which lexical units to use. Now which ones?
  - Keeping **ALL** tokens that occur in the data may be inefficient with respect to processing time and storage space
  - Very **infrequent** tokens may not provide enough evidence for models to learn much
  - Some tokens may not be **semantically important** or task-relevant

Stoplists

- A stop list is a list of words to be **excluded or ignored** for NLP tasks
- Typically, these include "closed-class" words such as determiners (*a, an, the*), prepositions (*of, with, by*) and conjunctions (*and, or*)
- More appropriate for tasks like text categorization or information retrieval, where we use bag-of-words representations that are ***insensitive*** to order.
- Less so for tasks like sequence tagging (e.g. POS tagging) where we need ***granular information*** about the context in which a word is used.

Stoplists



count_threshold	pct_of_wordtypes	coverage
0.0	0.00	1.000000
1.0	0.35	0.984347
2.0	0.50	0.972301
3.0	0.60	0.961901
4.0	0.65	0.952999
5.0	0.70	0.944722
7.0	0.75	0.930249
10.0	0.80	0.912732
15.0	0.85	0.888780
16.0	0.86	0.883777
17.0	0.87	0.879586
19.0	0.88	0.871968
22.0	0.89	0.860609
24.0	0.90	0.854308
28.0	0.91	0.842315
33.0	0.92	0.828618
38.0	0.93	0.815151
44.0	0.94	0.799795
53.0	0.95	0.780362
66.0	0.96	0.754687
85.0	0.97	0.725701
125.0	0.98	0.683092
226.0	0.99	0.616222
69971.0	1.00	0.000000

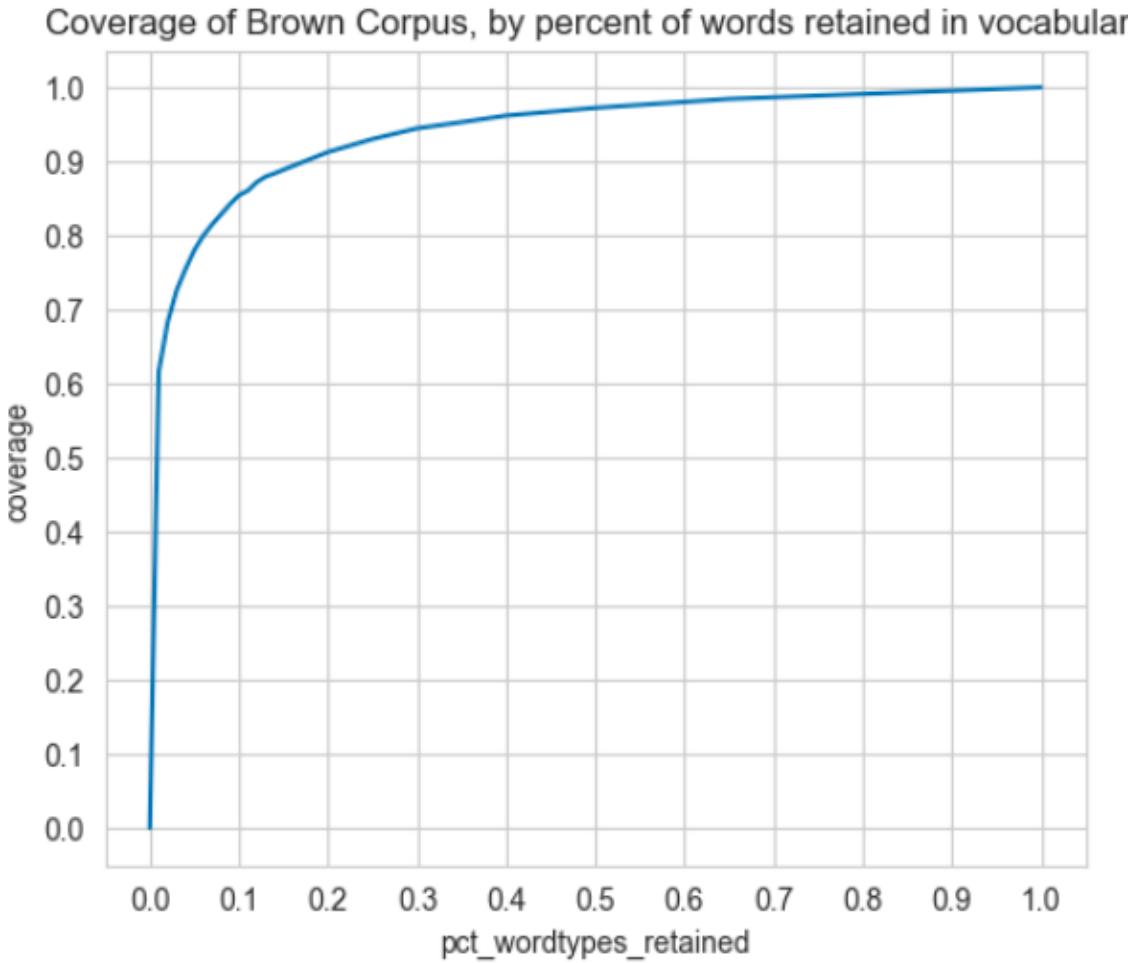
Low Count cutoffs

- We often use a count threshold to exclude very infrequent terms from the vocabulary.
- Depending on the corpus size, “infrequent” may mean 5, 10 or even 100 occurrences.

← Counts based on Brown Corpus

Low Count cutoffs

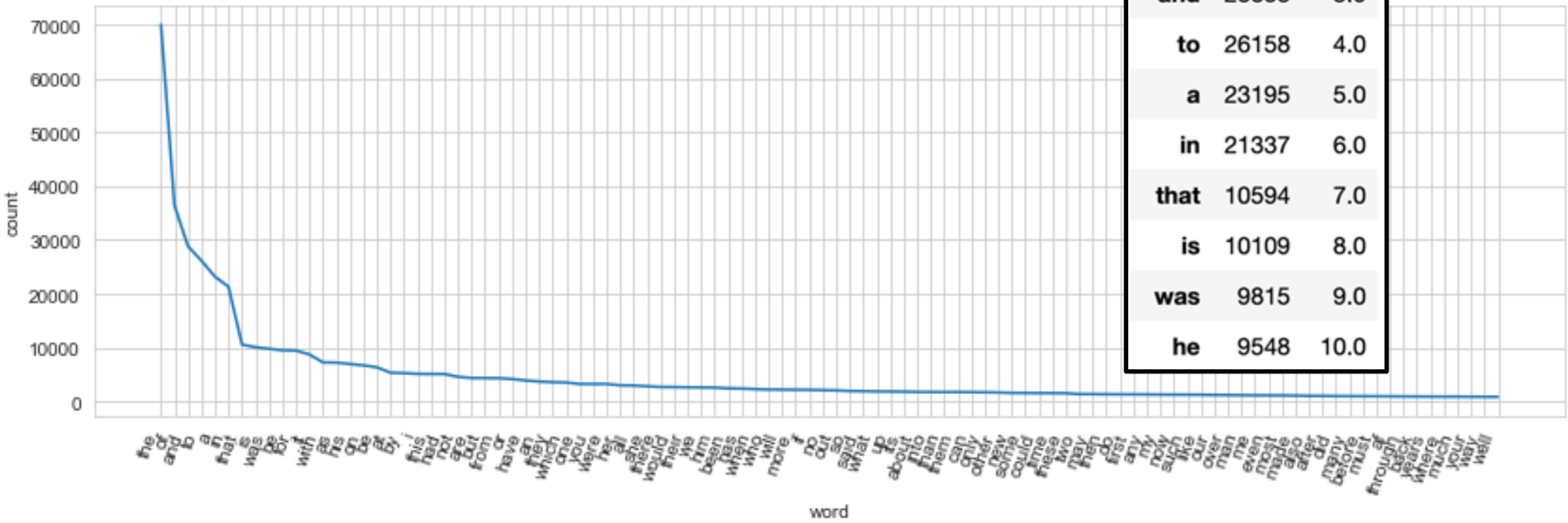
count_threshold	pct_of_wordtypes	coverage
0.0	0.00	1.000000
1.0	0.35	0.984347
2.0	0.50	0.972301
3.0	0.60	0.961901
4.0	0.65	0.952999
5.0	0.70	0.944722
7.0	0.75	0.930249
10.0	0.80	0.912732
15.0	0.85	0.888780
16.0	0.86	0.883777
17.0	0.87	0.879586
19.0	0.88	0.871968
22.0	0.89	0.860609
24.0	0.90	0.854308
28.0	0.91	0.842315
33.0	0.92	0.828618
38.0	0.93	0.815151
44.0	0.94	0.799795
53.0	0.95	0.780362
66.0	0.96	0.754687
85.0	0.97	0.725701
125.0	0.98	0.683092
226.0	0.99	0.616222
69971.0	1.00	0.000000



Zipf's law

- Word frequencies are highly skewed

word	count	rank
the	69971	1.0
of	36412	2.0
and	28853	3.0
to	26158	4.0
a	23195	5.0
in	21337	6.0
that	10594	7.0
is	10109	8.0
was	9815	9.0
he	9548	10.0

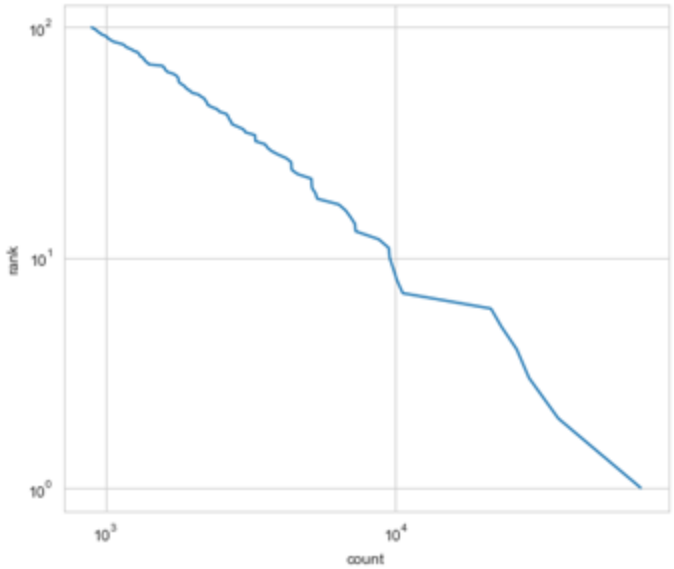


Zipf's law

word	count	rank	ratio
the	69971	1.0	69971.0
of	36412	2.0	72824.0
and	28853	3.0	86559.0
to	26158	4.0	104632.0
a	23195	5.0	115975.0
in	21337	6.0	128022.0
that	10594	7.0	74158.0
is	10109	8.0	80872.0
was	9815	9.0	88335.0
he	9548	10.0	95480.0
for	9489	11.0	104379.0
it	8760	12.0	105120.0
with	7289	13.0	94757.0
as	7253	14.0	101542.0
his	6996	15.0	104940.0
on	6741	16.0	107856.0
be	6377	17.0	108409.0
at	5372	18.0	96696.0
by	5306	19.0	100814.0
i	5164	20.0	103280.0

- Zipf's Law: a term's frequency in a corpus is approximately proportional to the inverse of its rank in a frequency-sorted list of terms within that corpus

$$Count(w_i) \propto \frac{1}{Rank(w_i)}$$



Out of vocabulary words

- How to deal with out-of-vocabulary (OOV) words?
  - Ignore them
  - Map them to placeholder tokens: `PERSON`, `PLACE`, `NUMBER`, ….
- It's a good idea to track OOV rate for your task, and to break down model performance by OOV rate

DOCUMENT REPRESENTATION

Define transformation function

- If we aim to represent **linguistic** units as vectors, how do we do that?
  - How do we represent a word?
  - How do we represent texts ("bags of words")?
- The function to produce a vector for a **word** is referred to as an *embedding*
- A function to produce a vector for a **text** is often referred to as a *vectorizer*

Vectorization: how to represent multiple occurrences

- If words are represented as one-hot vectors, how do we represent a document?

1. Binary vectorizer: record 0/1 value indicating presence/absence of word
2. Count vectorizer: record number of occurrences per word in the document

```
The first finisher gets the first prize
```

	Binary vector	Count vector
a	0	0
finisher	1	1
first	1	2
gets	1	1
pizza	0	0
prize	1	1
the	1	2

Problems with binary and count vectorizers

- Problem 1: Binary vectorization gives too little weight to words that occur multiple times. Count vectorization gives too much weight.
  - A single word that shows up many times can swamp the effect of the rest of the text in determining what other texts are most similar

- Problem 2: Not all words should be equal. Words with distinctive meaning *(turbine, diagonalize, cantilever)* should count more towards the document representation than very general terms (*people, things, stuff, day*)

Tf*idf Vectorization

- Solution: define a “score” for each component of the document vector associated with a given word *w*, broken down into two parts:
  - The ”**term frequency**” (*tf*) is a measure of the frequency of *w* within the document
  - The ”**inverse document frequency**” (*idf*) is a measure of the rarity of *w across documents*

Tf*idf Vectorization

- The *tf* value is typically a sublinear transformation of the raw document count—e.g., $\log(1 + Count(w))$
- The *idf* value is an inverse measure of the frequency with which *w* occurs across documents—e.g., $-\log \frac{DocumentCount(w)}{NumDocuments}$
- The total score for word *w* is $tf_w \times idf_w$; thus, the notation *tf*idf*

Tf*idf Vectorization

Intuition of Tf*Idf:

- ***term frequency (tf)*** can be chosen so that a lot of occurrences doesn't add up to too high a score (to address problem 1), and
- ***inverse document frequency (idf)*** will penalize the score for frequest words that appear in a high number of documents (to address problem 2)

Tf*idf Vectorization

[NOTEBOOK]

LEXICAL REPRESENTATION

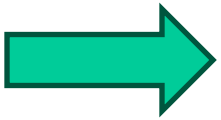
Vector Space Model for Text

A simple example:

- Sentence 0: "My dog sleeps"
- Sentence 1: "Your cat sleeps"
- Sentence 2: "That dog chases any cat"
- Vocab: `["cat","chases","dog","eats","sleeps"]`

Representing in vector space:

- Sentence 0: `[0,0,1,0,1]`
- Sentence 1: `[1,0,0,0,1]`
- Sentence 2: `[1,1,1,0,0]`



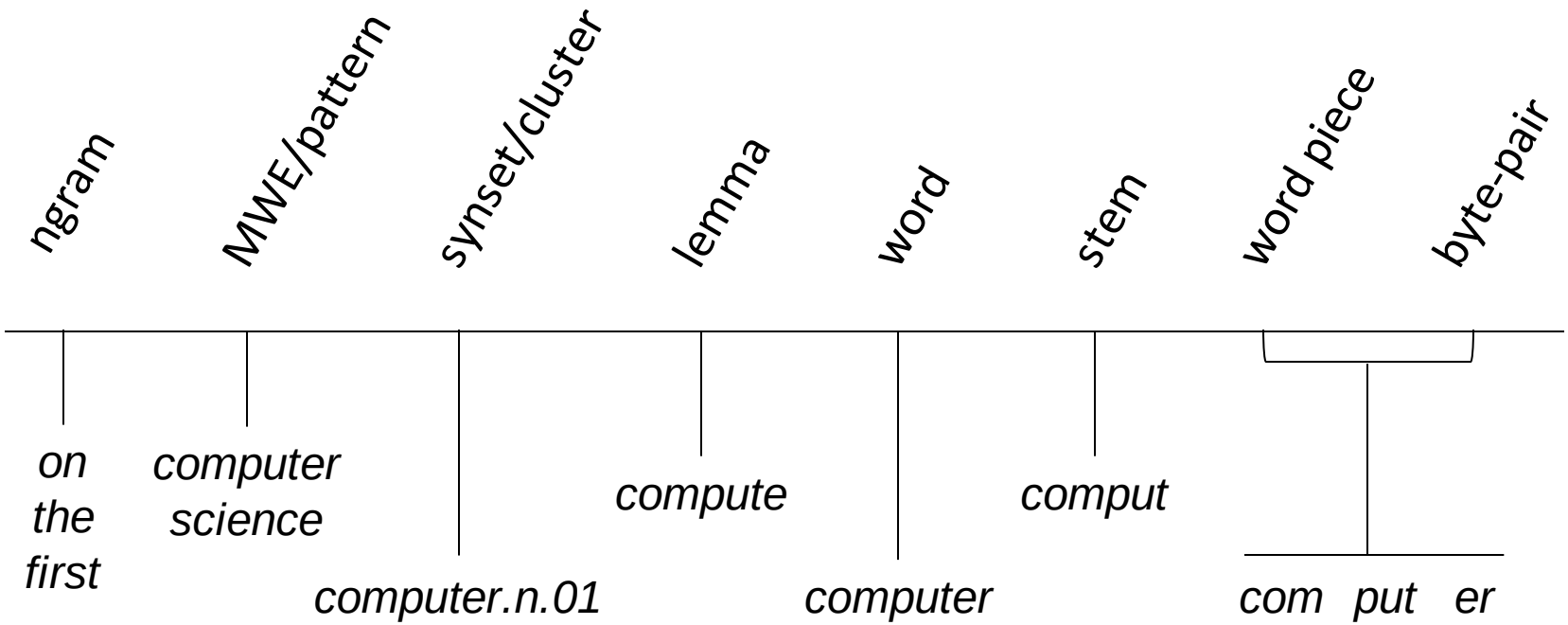
0	0	1	0	1
1	0	0	0	1
1	1	1	0	0

Does not represent characters and local context

Lexical Representations

- "Lexical"
  - Generally, relating to **words**, or to the vocabulary used for a task
    - *Lexical embeddings*: word representations for neural networks
    - *Lexical semantics*: meanings of words and word parts
    - *Lexical diversity*: the range of vocabulary in a text

Levels of Lexical Representation



- Different levels of **granularity** may be relevant for different tasks

Words

- For English and most alphabetic languages, **delimited** by whitespace
  - with some nuances related to tokenization
  - contractions, abbreviations, case, emojis…
- Un-normalized forms: fully inflected

```
you will rejoice to hear that no
disaster has accompanied the
commencement of an enterprise which you
have regarded with such evil forebodings
```

Frankenstein, Mary Shelley

Word stems

- As discussed previously, crude chopping of suffixes to get ***base form*** that is shared by multiple un-normalized words
- Loses information about inflection, but reduces vocabulary size and captures some ***meaning similarity***

```
you will rejoic to hear that no disast
ha accompani the commenc of an enterpris
which you have regard with such evil
forebod
```

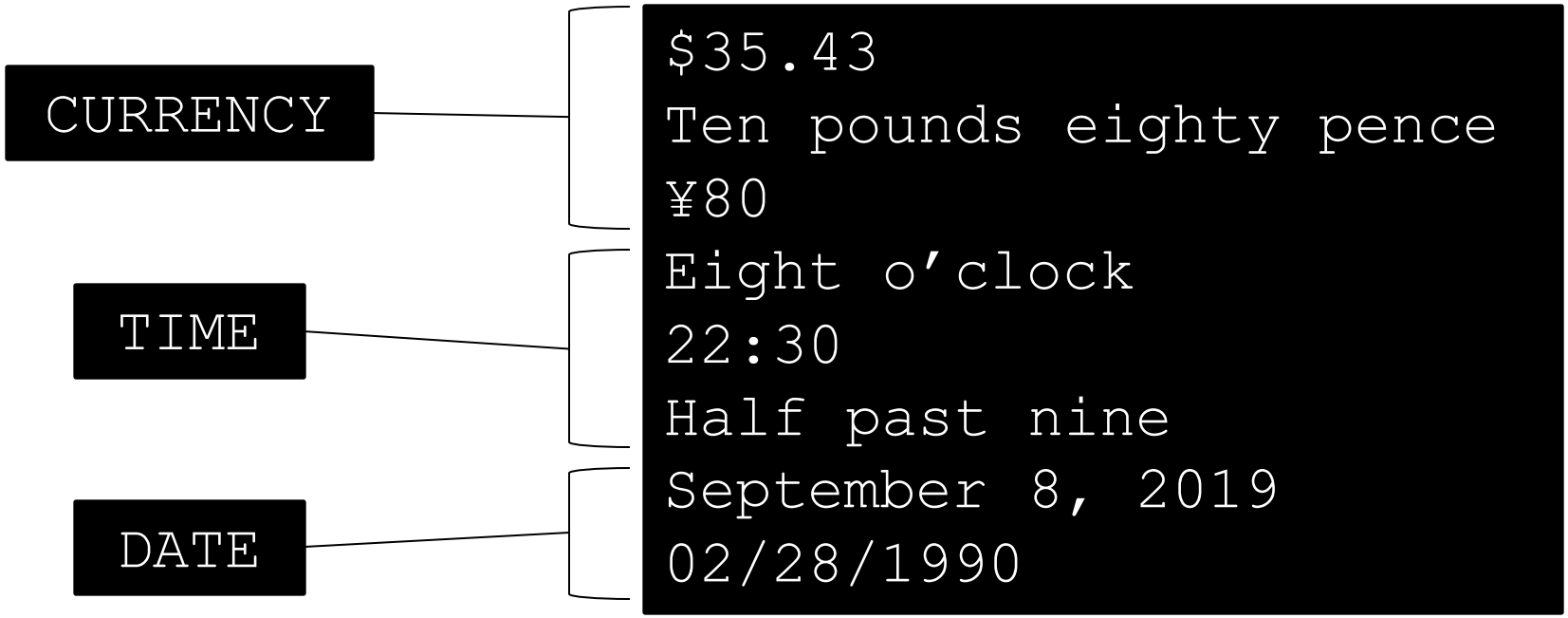
Lemmas (or lemmata)

- Select **canonical** form to represent all differently-inflected forms of a word
- Similar to stemming, but results in “real” words

```
you will rejoice to hear that no
disaster has accompany the commencement
of an enterprise which you have regard
with such evil foreboding
```

Multi-word expressions (MWEs)

- We may write **patterns** to capture and normalize specific types of token sequences



Multi-word expressions (MWEs)

- Other multi-word expressions may come from a dictionary or from statistical analyses of terms that occur together *more frequently than would be expected by chance*

```
New York
computer science
Supreme Court
life hack
Illinois Institute of Technology
```

N-grams

- Single words: unigrams:
- 2-word sequences: bigrams
- 3-word sequences: trigrams
- N-word sequences: ngrams

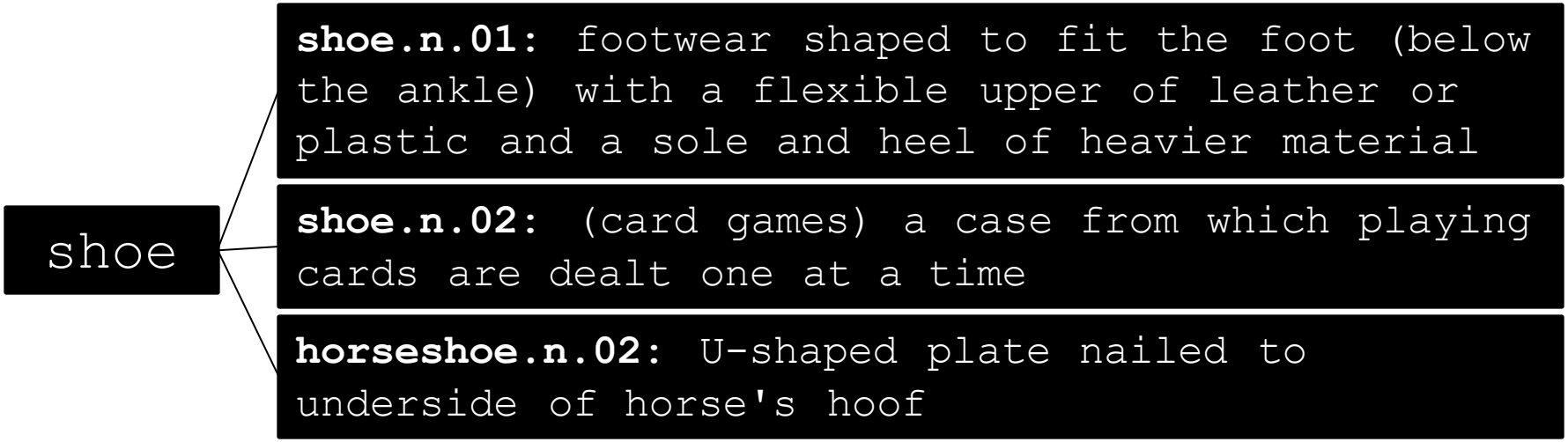
- Uses:
  - Similar to features based on MWEs or patterns, but less ***curated***
  - Local contextual features for language modeling / sequence labeling

Semantic word clusters

- Instead of (or in addition to) representing words directly, we could also represent the ***concept*** or ***word class*** that they represent
- The idea is to reduce the feature set (vocabulary) by using **higher-order** semantic units as the level of representation
- So, instead of learning something specific about the words *baseball*, *tennis*, and *golf* independently, the model could just learn about the supercategory *SPORTS*.
- Word clusters can come from a lexical resource like Wordnet, or from statistical clustering

Wordnet and Synsets

- **Wordnet** is a manually-compiled machine-readable dictionary for English (and a few other languages), maintained by Princeton University
  - https://wordnet.princeton.edu/
- It can be used programmatically to look up word "**synsets**" (senses related to a set of words)



Sub-word Representations

- **Sub-word**: Leverage character sequence representation of the word; "word-internal"
- Why needed?

  Neologism & OOV (out of vocabulary) words
- Generally developed in an *unsupervised* manner
  - Use statistical information about the data itself to identify the most useful units to include in the vocabulary
  - Not informed by external human labeling

Byte-pair encoding and word pieces

- *Byte pair encodings* and *word pieces* are two **unsupervised** methods for generating a **sub-word vocabulary** of a given size
- Based on Character *n-gram representation*
  - e.g. Bigram for *natural* : *#n, na, at, ur, ra, al, l#*
- More useful for **machine translation** and **natural language generation (**output individual tokens) than for text categorization
- Applied as tokenizers transformer models

Byte-pair encoding

- What makes a word part useful to include in our vocabulary?
  - It should show up in a lot of words
    - *having, liking, showing, making*
  - It should show up in frequent words
    - *is, not, the, whenever, whichever*
  - When we factor it out of a word, the remaining pieces should also be useful parts
    - *show+ing, when+ever, wait+er, blue+berry*
    - *easyc+hair, cran+berry, s+mid+gen*
- Byte-pair encoding uses these principles to implement an algorithm for sub-word vocabulary induction

Byte-pair encoding

1. Generate a word frequency list
2. Append a word-end pseudo-letter **</w>** to every word
3. Initialize vocabulary to set of distinct letters
4. Until the target vocabulary size is reached:
   - Select the pair of vocabulary items **most frequently** occurring together in sequence
   - Add it to the vocabulary as a new item
   - Regenerate the word frequency list segmentations using the new vocabulary

Byte-pair encoding

1. Generate a word frequency list
2. Append a word-end pseudo-letter **</w>** to every word

Word	Count
t h e </w>	69971
o f </w>	36412
a n d </w>	28853
t o </w>	26158
a </w>	23195
i n </w>	21337
t h a t </w>	10594
i s </w>	10109
w a s </w>	9815
h e </w>	9548
f o r </w>	9489
i t </w>	8760
w i t h </w>	7289
a s </w>	7253

Byte-pair encoding

3. Initialize vocabulary to set of distinct letters

V = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, </w>}

Byte-pair encoding

4. Until the target vocabulary size is reached:
   - Select the pair of vocabulary items most frequently occurring together in sequence
   - Add it to the vocabulary as a new item
   - Regenerate the word frequency list segmentations using the new vocabulary

Word	Count
t h e </w>	69971
o f </w>	36412
a n d </w>	28853
t o </w>	26158
a </w>	23195
i n </w>	21337
t h a t </w>	10594
i s </w>	10109
w a s </w>	9815
h e </w>	9548
f o r </w>	9489
i t </w>	8760
w i t h </w>	7289
a s </w>	7253

Bigram	Count
t h	69971+10594+7289 = **87854**
s </w>	10109+9815+7253 = **27177**
…	

Byte-pair encoding

4. Until the target vocabulary size is reached:
   - Select the pair of vocabulary items most frequently occurring together in sequence
   - Add it to the vocabulary as a new item
   - Regenerate the word frequency list segmentations using the new vocabulary

Word	Count
th e </w>	69971
o f </w>	36412
a n d </w>	28853
t o </w>	26158
a </w>	23195
i n </w>	21337
th a t </w>	10594
i s </w>	10109
w a s </w>	9815
h e </w>	9548
f o r </w>	9489
i t </w>	8760
w i **th** </w>	7289
a s </w>	7253

Bigram	Count
th e	**69971**
s </w>	10109+9815+7253 = **27177**
…	

Byte-pair encoding

[NOTEBOOK]

Word piece encoding

1. Generate a word frequency list
2. Append a word-end pseudo-letter **</w>** to every word
3. Initialize vocabulary to set of distinct letters
4. Until the target vocabulary size is reached:
   - Select the pair of vocabulary items *that increase the likelihood estimate most:*

Score = (freq. of pair) / (freq. of element-1 × freq. of element-2)

   - Add it to the vocabulary as a new item
   - Regenerate the word frequency list segmentations using the new vocabulary

REMOVE

SUMMARY - TEXT REPRESENATIONS

Representing Text

- Choose a document representation:
  - Bag-of-words: Binary, Count, "Tf-Idf"
  - Other methods leverage word-order
- Choose a word representation
  - Word, Lemma, MLE, n-gram
  - Atomic words vs sub-word
- Select a vocabulary
  - Dependent on corpus (text data)
  - Remove stop words and infrequent words?