Vector Semantics

SLP 6. Vector Semantics

https://web.stanford.edu/~jurafsky/slp3/6.pdf

tf-idf

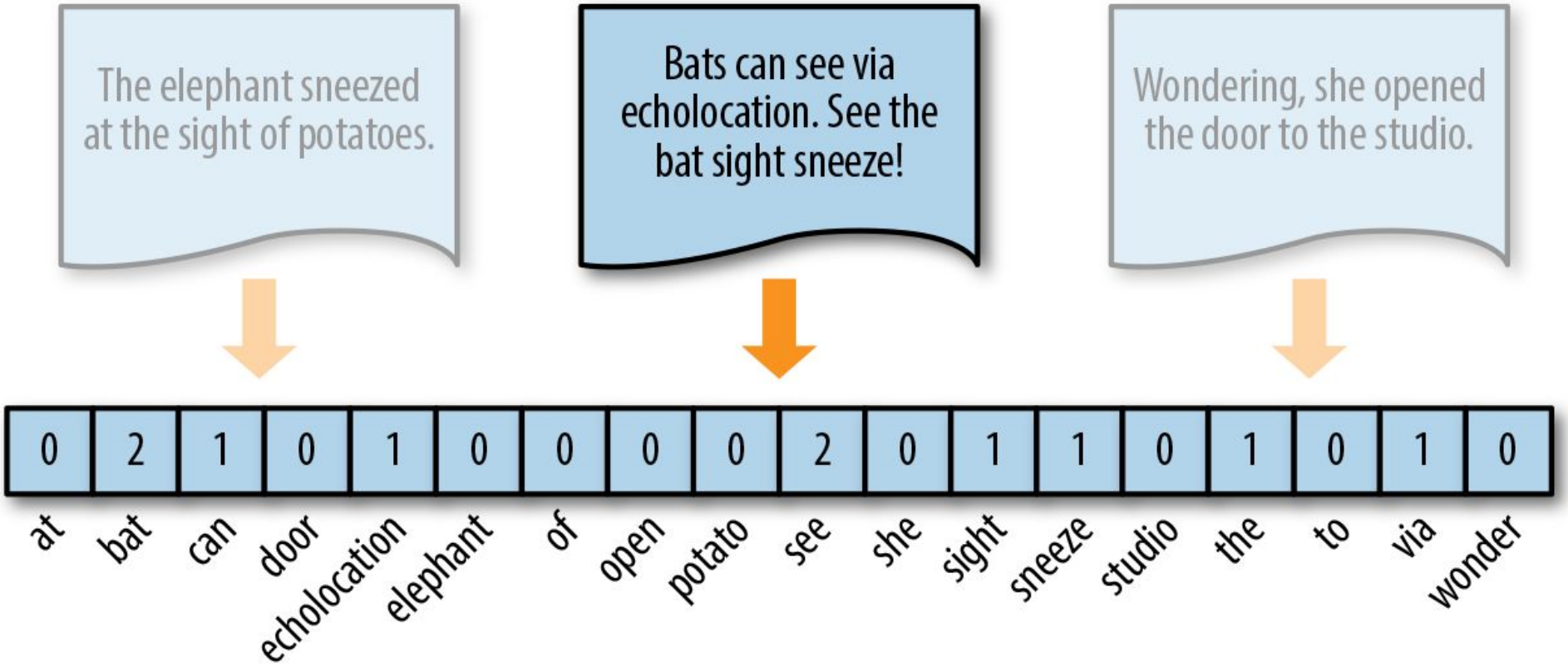
BOW (Bag-of-words) term frequency

- A sentence or a document is represented as the bag of its words, disregarding grammar and word order but keeping multiplicity
- Used for document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier

(1) John likes to watch movies. Mary likes movies too.
(2) John also likes to watch football games.

(1) [1, 2, 1, 1, 2, 1, 1, 0, 0, 0]
(2) [1, 1, 1, 1, 0, 0, 0, 1, 1, 1]

BoW1 = {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1};
BoW2 = {"John":1,"also":1,"likes":1,"to":1,"watch":1,"football":1,"games":1};



출처: Applied Text Analysis with Python - Bengfort, Bilbro, Ojeda (O'REilly)

Tf

- Convert a collection of text documents to a matrix of token counts
- fit_transform(): builds vocabulary dictionary and returns term-document matrix
- transform(): transform documents to document-term matrix

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.vocabulary_)
{'this': 8, 'is': 3, 'the': 6, 'first': 2, 'document': 1, 'second': 5, 'and': 0,
'third': 7, 'one': 4}
 >>> print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
 >>>print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
new = vectorizer.transform(['This This new document'])
>>> print(new.toarray())
[[0 1 0 0 0 0 0 0 2]]
```

Tf with options

- Parameters

analyzer: whether the feature should be made of word n-gram or character n-grams
binary: boolean, default=False. If True, all non zero counts are set to 1
ngram_range: tuple (min_n, max_n), default=(1, 1). lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted

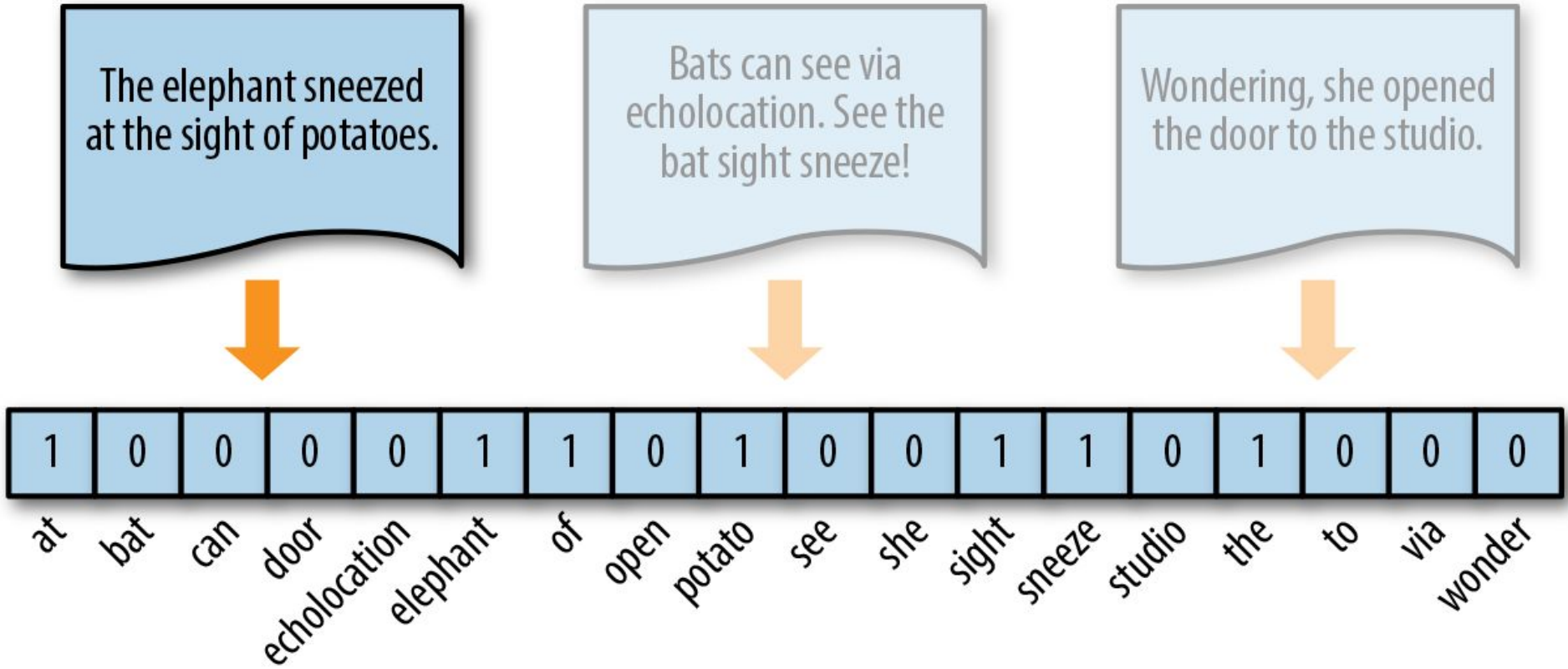
```
vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))
X2 = vectorizer2.fit_transform(corpus)
 >>> print(vectorizer2.get_feature_names())
['and this', 'document is', 'first document', 'is the', 'is this',
'second document', 'the first', 'the second', 'the third', 'third one',
 'this document', 'this is', 'this the']
 >>> print(X2.toarray())
 [[0 0 1 1 0 0 1 0 0 0 0 1 0]
 [0 1 0 1 0 1 0 1 0 0 1 0 0]
 [1 0 0 1 0 0 0 0 1 1 0 1 0]
 [0 0 1 0 1 0 1 0 0 0 0 0 1]]

one_hot = CountVectorizer(binary=True)
X = one_hot.fit_transform(corpus)
new = one_hot.transform(['This This new document'])
>>> print(new.toarray())
[[0 1 0 0 0 0 0 0 1]]
```

One-hot Encoding representation

- Represents categorical variables as **binary** vectors
- A word is denoted by a ***N*-long vector** where *N* represents the number of words in the vocabulary

I:	[**1**, 0, 0]
am:	[0, **1**, 0]
good:	[0, 0, **1**]

- Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers
- Creates a high dimensional yet sparse feature vector
- No ability to capture the word's meaning or to understand the relationships between the words



출처: Applied Text Analysis with Python - Bengfort, Bilbro, Ojeda (O'REilly)

tf-idf: combine two factors

- **tf**: **term frequency**. frequency count (usually log-transformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **idf** (**inverse document frequency)**: gives a higher weight to words that occur only in a few documents

Total # of docs in collection

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

# of docs that have word i

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

tf-idf: combine two factors

- **tf-idf** value for word *t* in document *d*:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Words like "the" or "good" have very low idf

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Normalizes word frequencies across documents

Document A: I love you so much

Document B: I love pizza so much

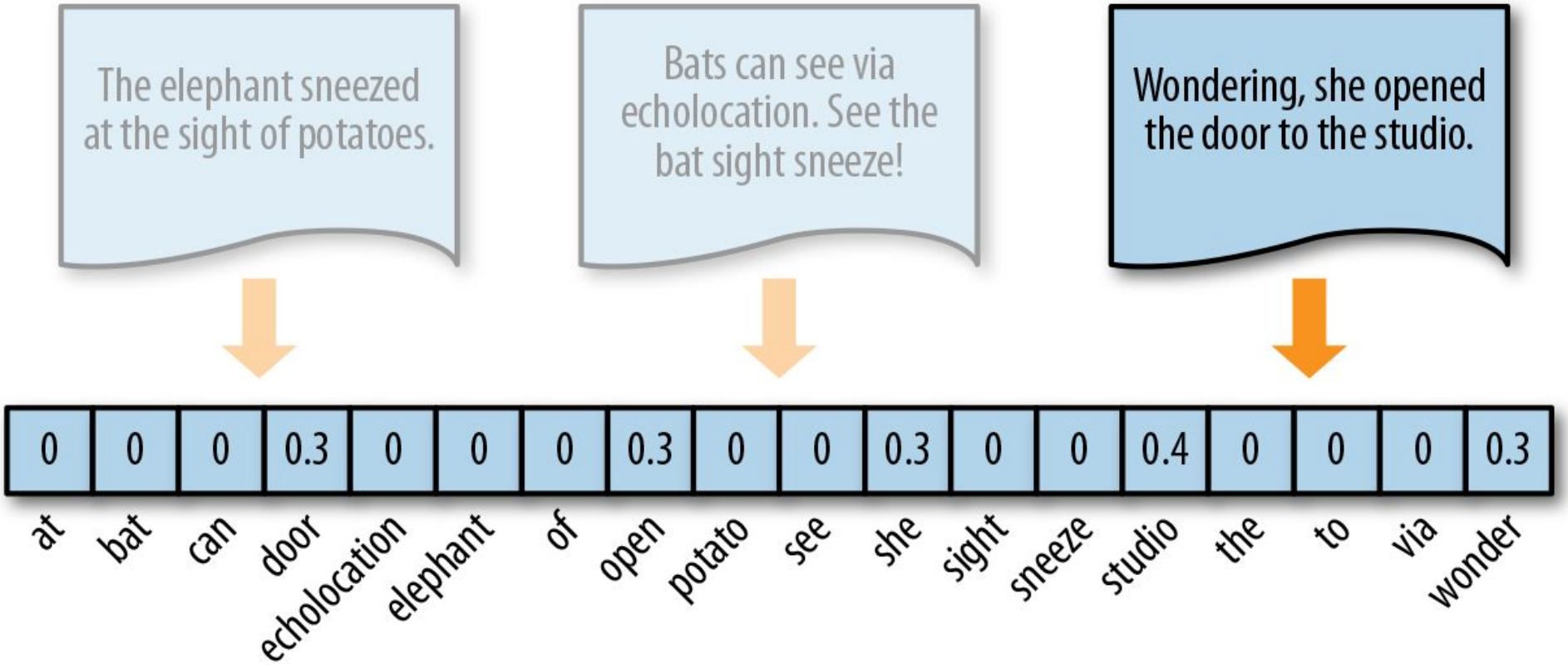
Document C: I miss you I want you

Document D: I want pizza

tf	i	love	you	so	much	pizza	miss	want
A	1	1	1	1	1	0	0	0
B	1	1	0	1	1	1	0	0
C	2	0	2	0	0	0	1	1
D	1	0	0	0	0	1	0	1

idf	i	love	you	so	much	pizza	miss	want
logN/n_t	log4/4	log4/2	log4/2	log4/2	log4/2	log4/2	log4/1	log4/2

Tf-idf	i	love	you	so	much	pizza	miss	want
A	1*log4/4	1*log4/2	1*log4/2	1*log4/2	1*log4/2	0*log4/2	0*log4/1	0*log4/2
B	1*log4/4	1*log4/2	0*log4/2	1*log4/2	1*log4/2	1*log4/2	0*log4/1	0*log4/2
C	2*log4/4	0*log4/2	2*log4/2	0*log4/2	0*log4/2	0*log4/2	1*log4/1	1*log4/2
D	1*log4/4	0*log4/2	0*log4/2	0*log4/2	0*log4/2	1*log4/2	0*log4/1	1*log4/2



출처: Applied Text Analysis with Python - Bengfort, Bilbro, Ojeda (O'REilly)

Example Code

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
from sklearn.feature_extraction.text import TfidfVectorizer
data=['I love you so much',
'I love pizza so much',
'I miss you I want you',
'I want pizza']

vectorizer = TfidfVectorizer((token_pattern=u"(?u)\\b\\w+\\b"))
vtr_data = vectorizer.fit_transform(data)

vtr_data = vtr_data.toarray()
featurenames = vectorizer.get_feature_names()
print(featurenames)
print(vtr_data)
```

```
['i',          'love',        'miss',        'much',        'pizza',       'so',          'want',        'you']
[[ 0.31418628  0.47468068  0.          0.47468068  0.          0.47468068  0.          0.47468068]
 [ 0.31418628  0.47468068  0.          0.47468068  0.47468068  0.47468068  0.          0.        ]
 [ 0.45780688  0.          0.43864554  0.          0.          0.          0.34583318  0.69166636]
 [ 0.42389674  0.          0.          0.          0.64043405  0.          0.64043405  0.        ]]
```

Limitations of Tf-idf

- Does not deal with the difference between terms such as “like” and “don’t like”
- We may introduce bigrams or trigrams into the tf-idf algorithm

An alternative to tf-idf

- **Positive Pointwise Mutual Information (PPMI)** to weigh the association between two words is to ask how much **more** the two words co-occur in our corpus **than we would have a priori expected them to appear by chance**

Pointwise Mutual Information (PMI)

Do events *x* and *y* co-occur more than if they were independent?

$$\text{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: Do words *x* and *y* co-occur **more than if they were independent?**

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

0: independent
High value: dependent

word 1	*word 2*	*count word 1*	*count word 2*	count of co-occurrences	PMI
puerto	rico	1938	1311	1159	10.0349081703
hong	kong	2438	2694	2205	9.72831972408
los	angeles	3501	2808	2791	9.56067615065
carbon	dioxide	4265	1353	1032	9.09852946116
prize	laureate	5131	1676	1210	8.85870710982
san	francisco	5237	2477	1779	8.83305176711
nobel	prize	4098	5131	2498	8.68948811416
ice	hockey	5607	3002	1933	8.6555759741
star	trek	8264	1594	1489	8.63974676575
car	driver	5578	2749	1384	8.41470768304
it	the	283891	3293296	3347	-1.72037278119
are	of	234458	1761436	1019	-2.09254205335
this	the	199882	3293296	1211	-2.38612756961
is	of	565679	1761436	1562	-2.54614706831
and	of	1375396	1761436	2949	-2.79911817902
a	and	984442	1375396	1457	-2.92239510038
in	and	1187652	1375396	1537	-3.05660070757
to	and	1025659	1375396	1286	-3.08825363041
to	in	1025659	1187652	1066	-3.12911348956
of	and	1761436	1375396	1190	-3.70663100173

pairs of words getting the most and the least PMI scores in the first 50 millions of words in Wikipedia (dump of October 2015) filtering by 1,000 or more co-occurrences. The frequency of each count can be obtained by dividing its value by 50,000,952.
(Note: natural log is used to calculate the PMI values in this example, instead of log base 2)

https://en.wikipedia.org/wiki/Pointwise_mutual_information

Positive Pointwise Mutual Information (PPMI)

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring *less often* than we expect by chance are unreliable without enormous corpora
  - Plus it's not clear people are good at "unrelatedness"
- So we just replace negative PMI values and log(0) by 0

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

Summary

- Survey of Lexical Semantics
- Idea of Embeddings: Represent a word as a function of its distribution with other words
- Tf-idf
- Cosines
- PMI, PPMI