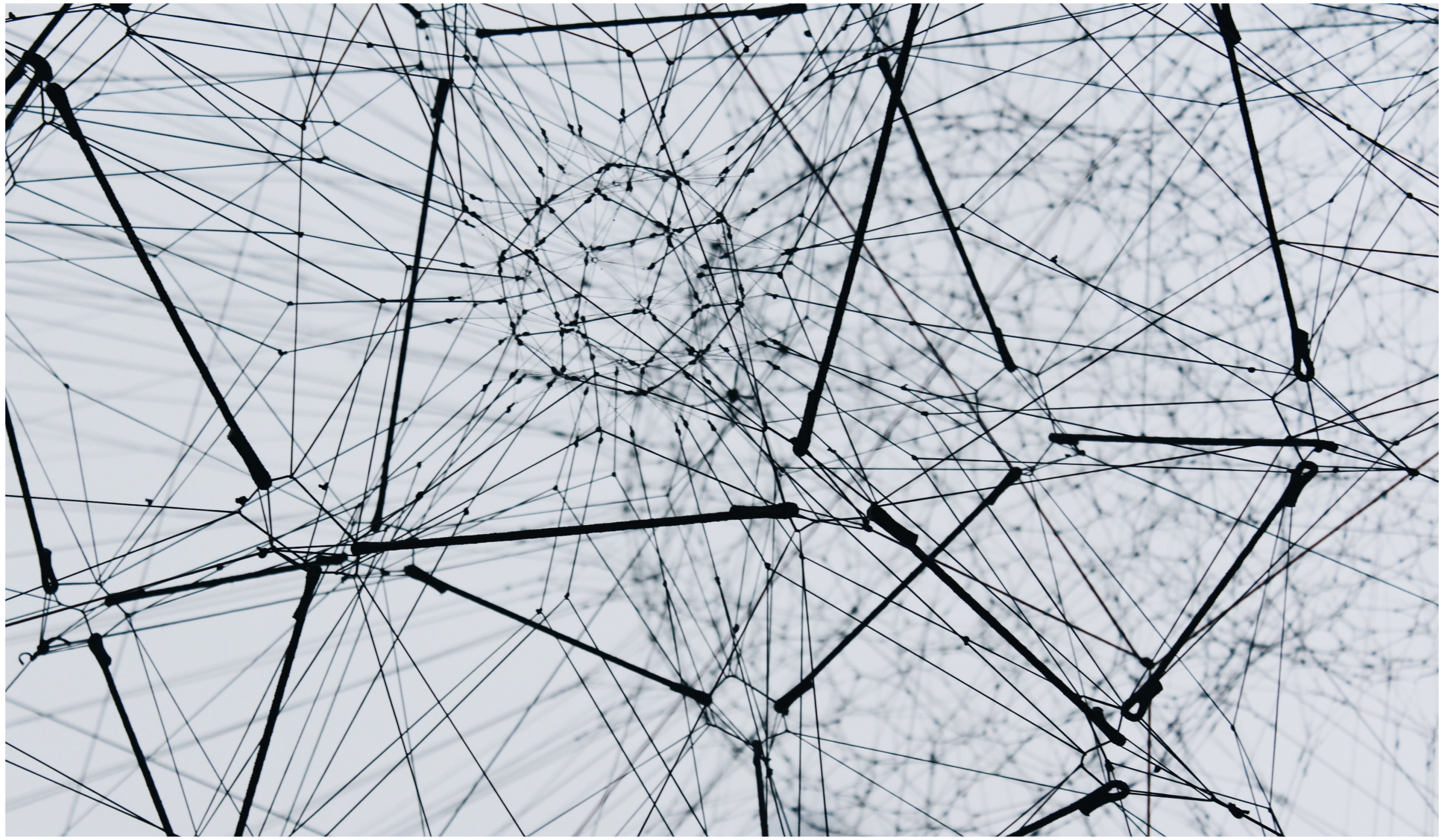


Word Embeddings

Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois at Chicago

CS 421: Natural Language Processing

Fall 2019

Many slides adapted from Jurafsky and Martin (https://web.stanford.edu/~jurafsky/slp3/).

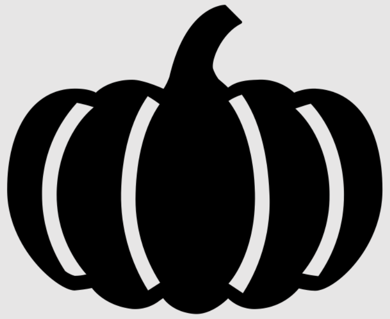
What are word embeddings?

- Vector representations of **word meaning**

pumpkin =	1	0	1	1	1	0	1	0	0	1

Distributional Hypothesis

- Words that occur in **similar contexts** tend to have **similar meanings**
  - It is time to harvest a **pumpkin**.
  - It is time to harvest a **squash**.
  - Add half a cup of canned **pumpkin** to the bread mix.
  - Add half a cup of canned **squash** to the bread mix.



Word embeddings make use of the distributional hypothesis to associate words with vectors.

Non-contextual Word Embeddings:
Every word has a single vector

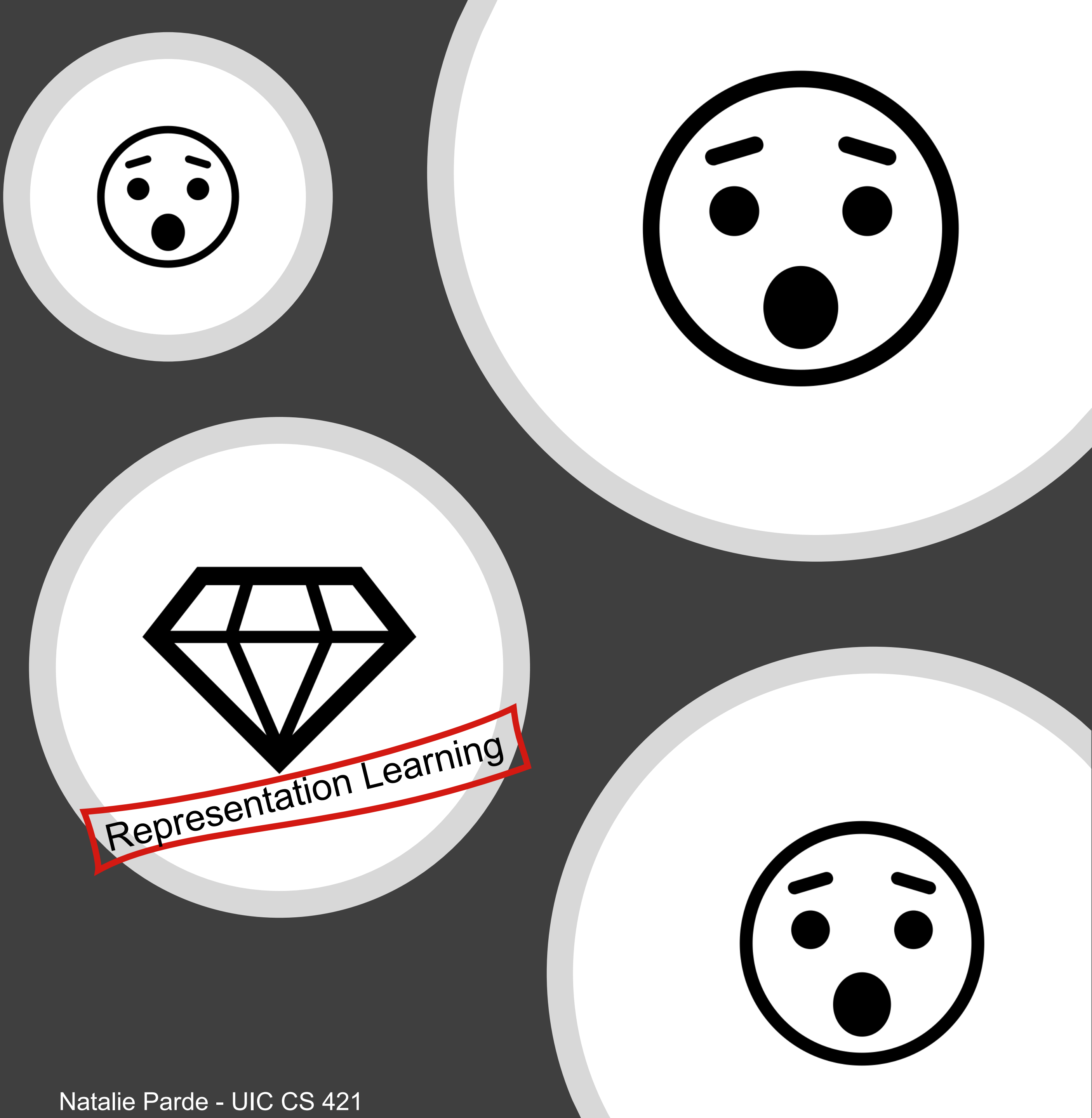
- Word2Vec
- GloVe

Contextual Word Embeddings:
A word can have multiple vectors

- ELMo
- BERT

Representation Learning

- Methods for learning word embeddings are examples of **representation learning**
  - **Representation Learning**: Automatically learning useful representations of input text
- Representation learning is **self-supervised**
- Recent trends in NLP have moved toward representation learning and away from creating representations by hand (i.e., **feature engineering**)



Lexical Semantics

- How should we represent the meaning of a word?
  - **N-gram models:** A word is a string of letters, or an index in a vocabulary list
  - **Logical representation:** A word defines its own meaning ("dog" = DOG)
- Problem?
  - These methods don't allow us to infer anything deeper about a word
    - Similar meanings
    - Antonyms
    - Positive/negative connotations
    - Related contexts

Some Useful Properties of Meaning

- Lemmas and senses
- Synonymy
- Word similarity
- Word relatedness
- Frames and roles
- Connotation

Lemmas and Senses

- **Lemma:** The base form of a word
  - Pumpkins → pumpkin
  - Mice → mouse
- **Word Sense:** Different aspects of meaning for a word
  - Mouse (1): A small rodent
  - Mouse (2): A device to control a computer cursor
- Words with the same lemma should (hopefully!) reside near one another in vector space
- Different senses of words should be represented as different vectors in **contextual word representations**, but not in **non-contextual word representations**

Synonymy

- How do word senses relate to one another?
  - When a word sense for one word is (nearly) identical to the word sense for another word, the two words are **synonyms**
- **Synonymy:** Two words are synonymous if they are substitutable for one another in any sentence without changing the situations in which the sentence would be true
- Synonymy = **propositional meaning**

Take a seat on my brand new leather **couch**!

Take a seat on my brand new leather **sofa**!

Word Similarity

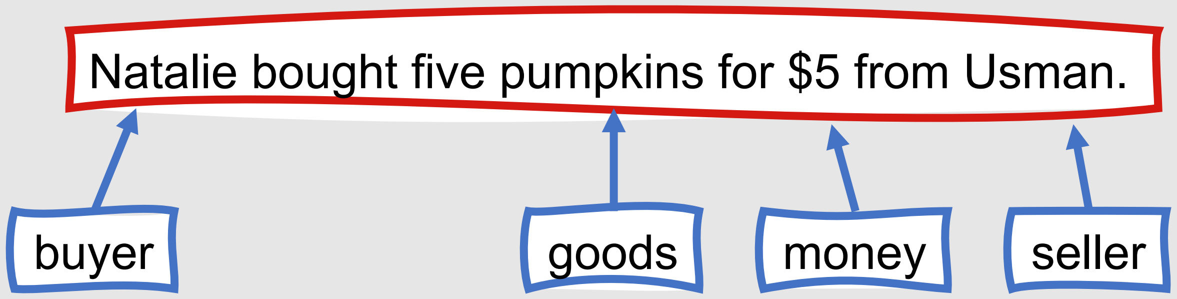
- Words don't often have that many synonyms, but they do have a lot of **similar** words
  - Pumpkin ≈ squash
- Similarity deals with **relations between words**, not word senses
  - Could word Y be commonly used in the same context as word X?
    - Half a cup of squash 🙂
    - Harvest a squash 🙂
    - Squash spice latte 🤨
- Estimates of word similarity can help with tasks like:
  - Question answering
  - Paraphrasing
  - Summarizing

Word Relatedness

- Words can also be **related**, but not similar, to one another
- **Word Relatedness:** An association between words based on their shared participation in an event or **semantic field**
  - **Semantic Field:** A set of words covering a particular semantic domain
    - Restaurant: {waiter, menu, plate, food, …, chef}
- Related words can often be determined using **topic modeling** approaches

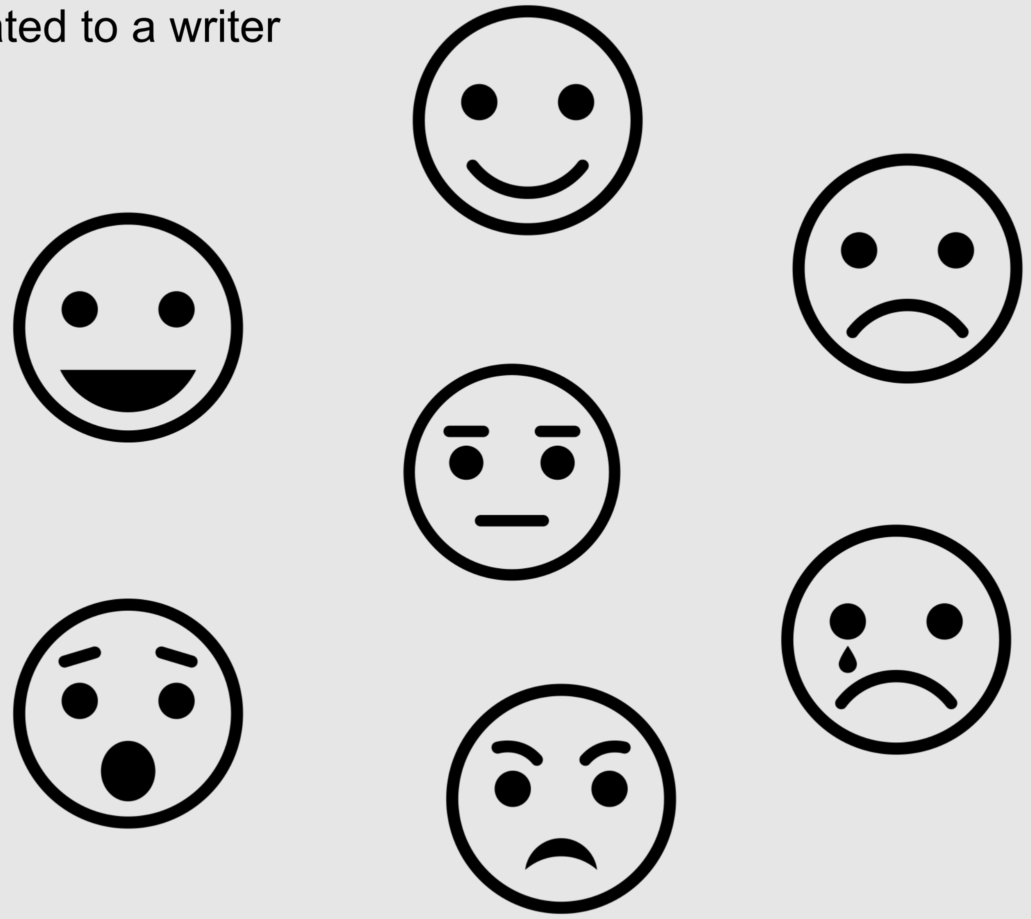
Semantic Frames and Roles

- **Semantic Frame:** A set of words that denote perspectives or participants in a particular type of event
  - Commercial Transaction = {buyer, seller, goods, money}
  - Closely related to semantic fields
- **Semantic Role:** A participant's underlying role with respect to the main verb in the sentence



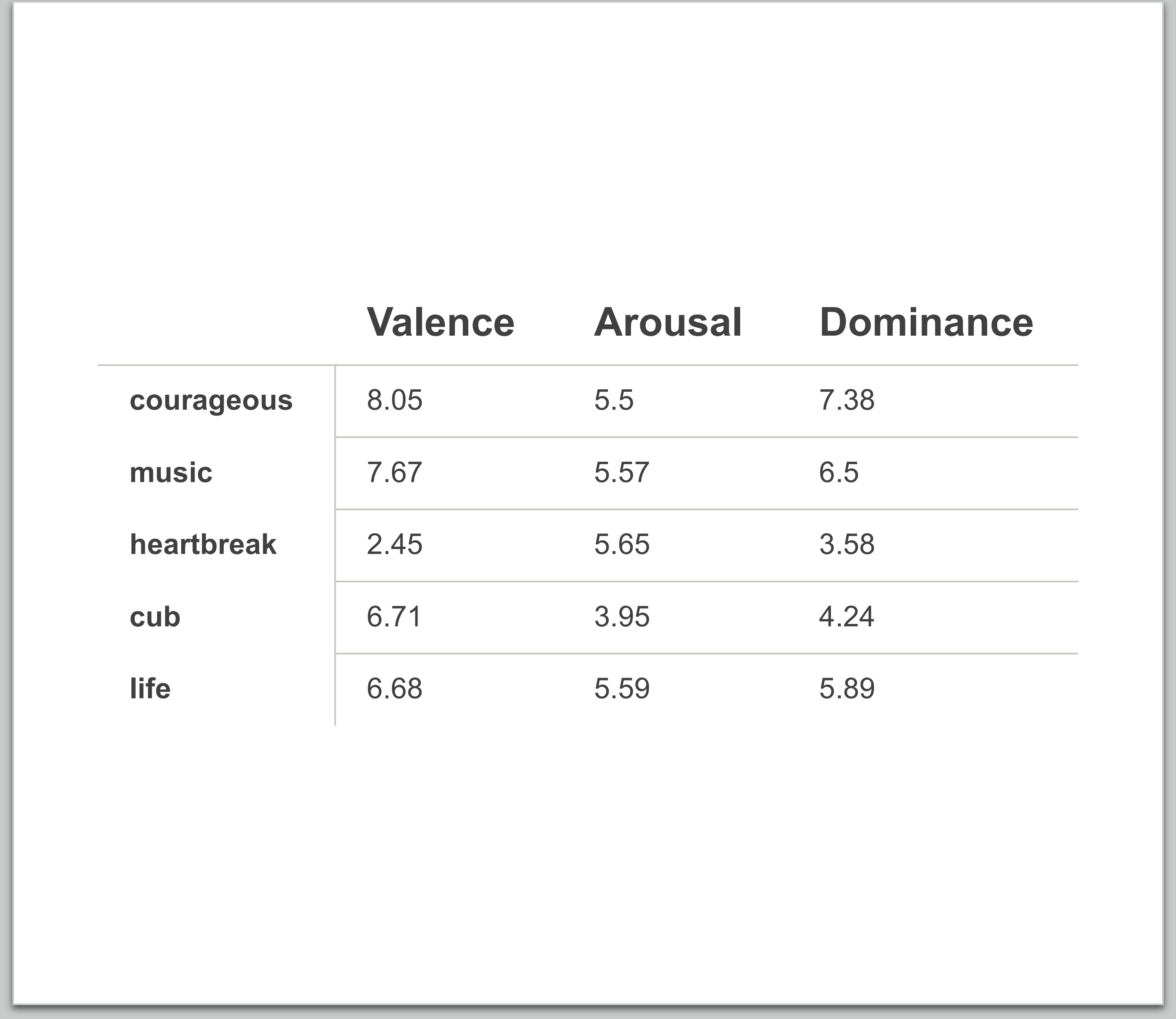
Connotation

- Also referred to as **affective meaning**
- **Connotation:** The aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations
- Generally three dimensions:
  - **Valence:** Positivity
    - High: Happy, satisfied
    - Low: Unhappy, annoyed
  - **Arousal:** Intensity of emotion
    - High: Excited, frenzied
    - Low: Relaxed, calm
  - **Dominance:** Degree of control
    - High: Important, controlling
    - Low: Awed, influenced



Connotation (Continued)

- Following this line of thought, each word can be represented by three numbers, corresponding to its value on each of the three affective dimensions
- When Osgood et al. (1957) did this, they ended up creating the first **word vectors**



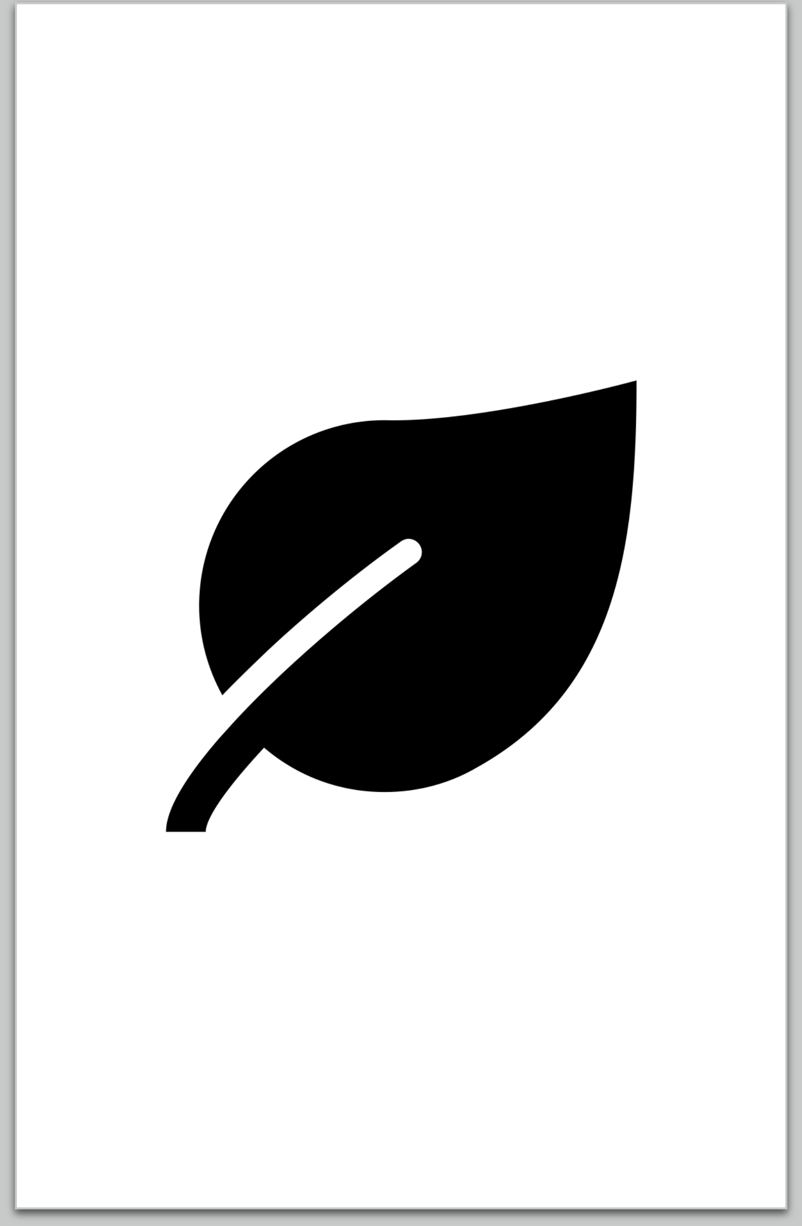
	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

Vector Semantics

- **Vector semantics** is a computational model that encodes different aspects of word meaning (e.g., word senses, word similarity and relatedness, lexical fields and frames, connotation, etc.) in vector form based on the word's usage in language
  - **A word is defined by its environment or distribution in language use**
- A word's distribution is the set of **contexts** in which it occurs
  - **Context:** Neighboring words or grammatical environments
- Two words that occur in very similar distributions are likely to have the same meaning

We do this all the time to infer meaning!

- Assume you don't know what the Cantonese word *ongchoi* means
- However, you read the following sentences:
  - Ongchoi is delicious sautéed with garlic.
  - Ongchoi is superb over rice.
  - …ongchoi leaves with salty sauces…
- You've seen many of the other context words in these sentences previously:
  - …spinach sautéed with garlic over rice…
  - …chard stems and leaves are delicious…
  - …collard greens and other salty leafy greens…
- Your (correct!) conclusion?
  - Ongchoi is probably a leafy green similar to spinach, chard, or collard greens

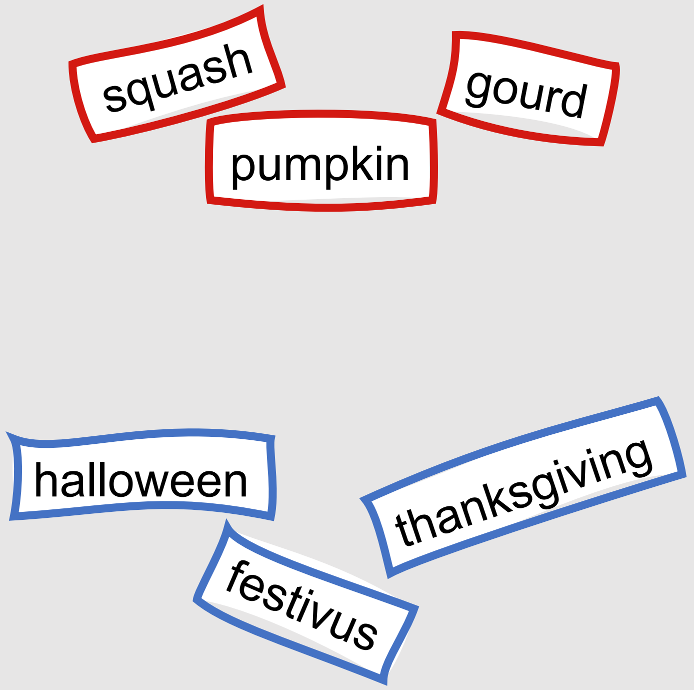


We can do the same thing computationally.

- Count the words in the context of *ongchoi*
- See what other words occur in those same contexts

Vector semantics combines two intuitions.

- **Distributionalist**
  - Defining a word by counting what other words occur in its environment
- **Vector**
  - Defining a word as a vector point in an n-dimensional space
- Different versions of vector semantics define the numbers in the vector differently; however, they're always based in some way on counts of neighboring words



Advantages of Vector Semantics

- Easily compute word and phrase **similarity**
  - How similar are the vectors representing a set of words?
- Allows models learned for downstream tasks to **generalize** to new words
  - If a new word is close in vector space to one with a known sentiment value, we might be able to assume that the sentiment for the new word is similar as well
- Can be learned automatically from text **without labels or supervision**

How do we learn word embeddings?

Two commonly used models:

- TF*IDF
  - Long, sparse word vectors
- Word2Vec
  - Short, dense word vectors

Many, many other models!

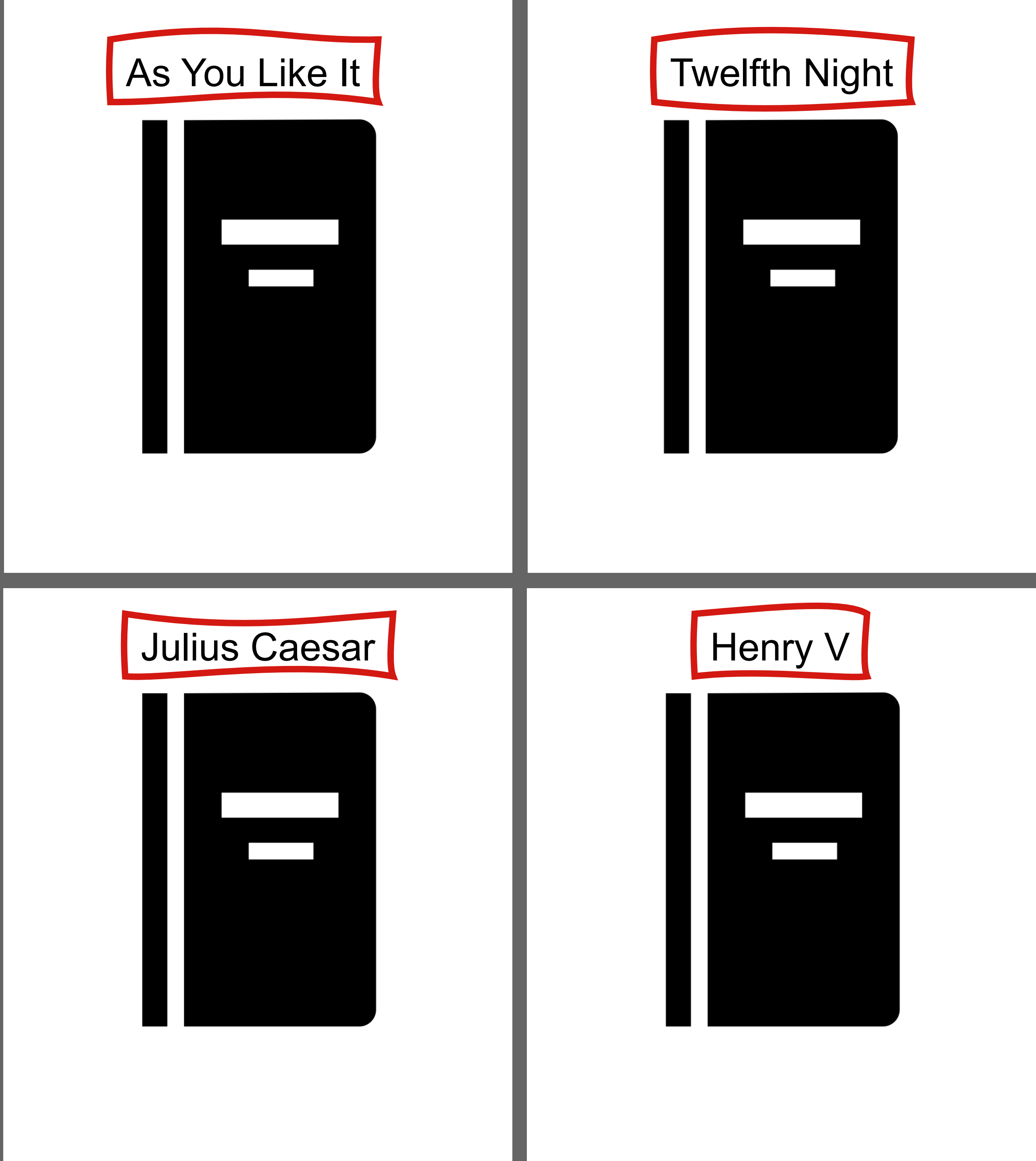
- GloVe
- ELMo
- BERT
- Etc….

TF*IDF

- Term Frequency * Inverse Document Frequency
- Meaning of a word is defined by the counts of nearby words
- To do this, a specific type of **co-occurrence matrix** is needed
  - **Term-document matrix**

Term-Document Matrix

- Rows: Words in a vocabulary
- Columns: Documents in a selection

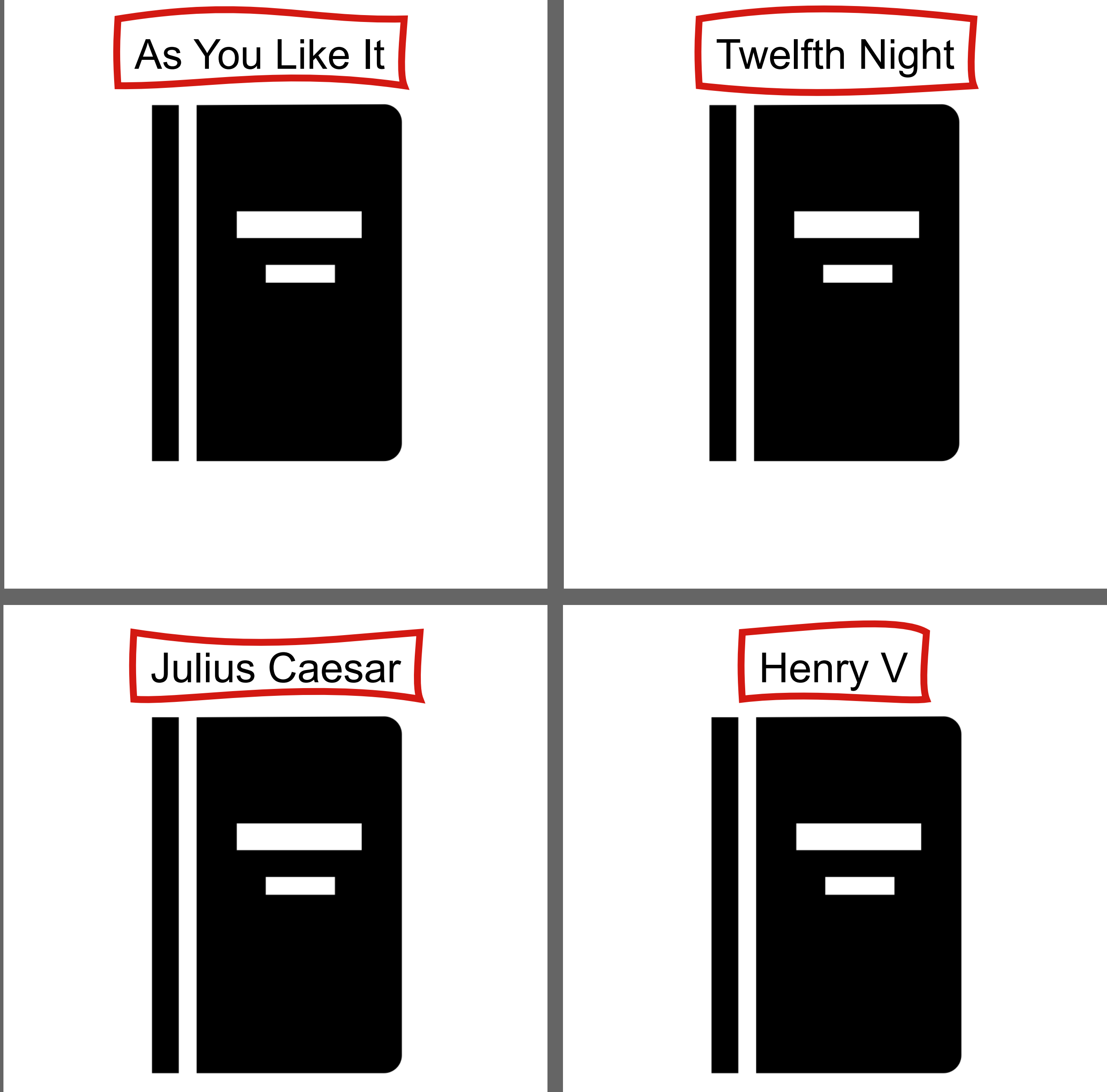


Term-Document Matrix

- Rows: Words in a vocabulary
- Columns: Documents in a selection

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

“wit” appears 3 times in Henry V

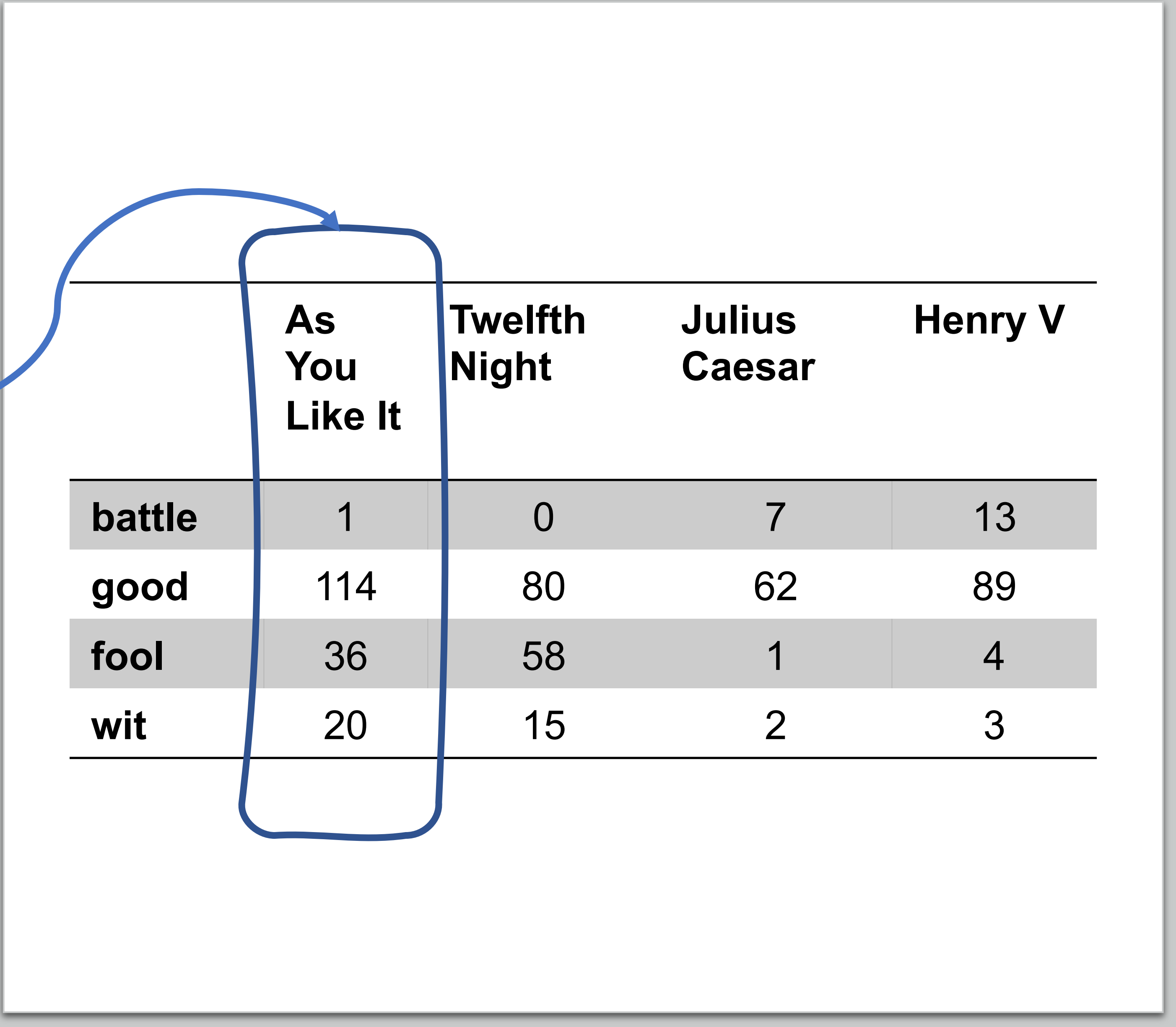


Ties to Information Retrieval

- Term-document matrices were first defined as part of the **vector space model** of **information retrieval**
  - Document = Count vector (column in the previous slide)
- Useful for searching through collections of documents (books, webpages, etc.) and returning those most relevant to a given search query

Linear Algebra Refresher

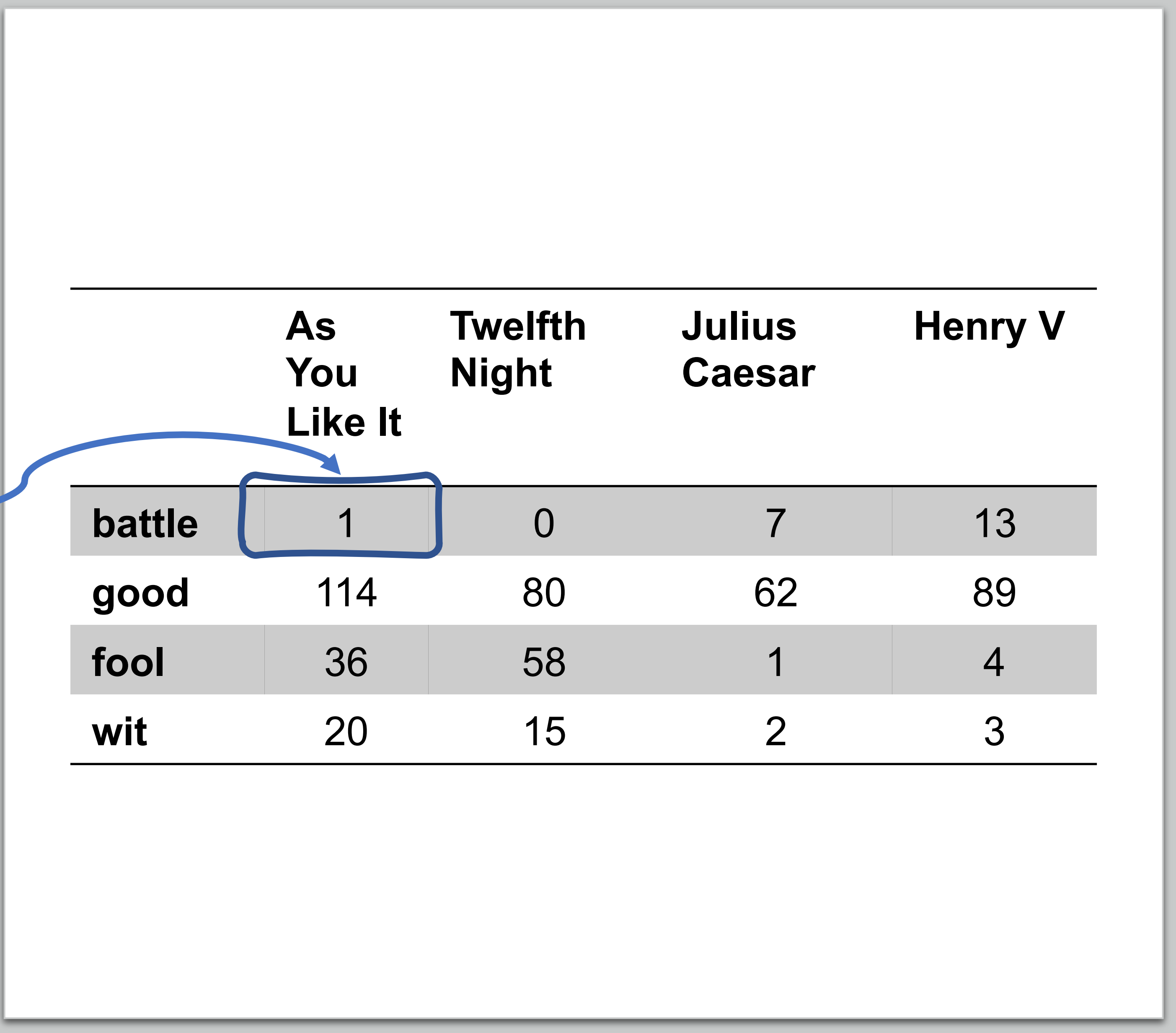
- **Vector:** A list or array of numbers
  - As You Like It = [1, 114, 36, 20]
  - Julius Caesar = [7, 62, 1, 2]
- **Vector Space:** Collection of vectors
- **Dimension:** Vector length
  - For TF*IDF, document vectors have a **dimensionality** of |V|, the vocabulary size



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Linear Algebra Refresher

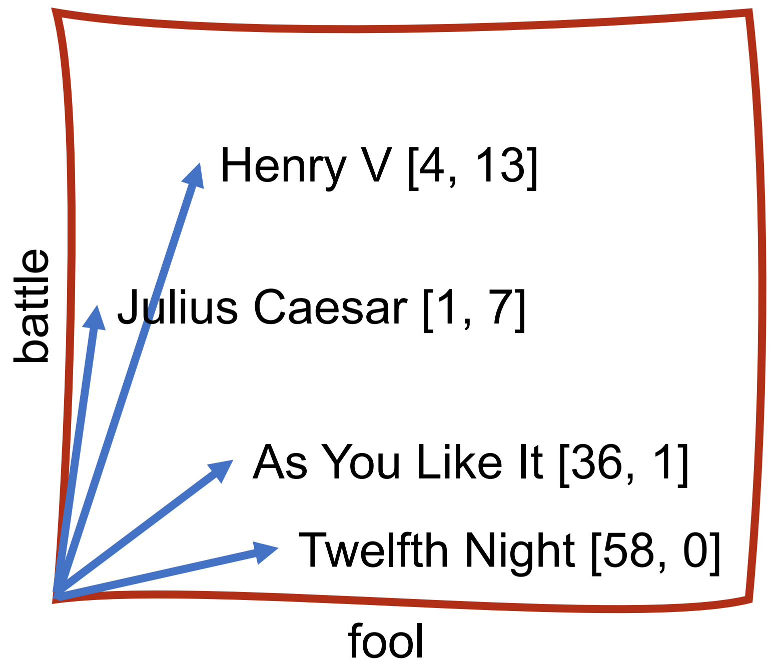
- Order is not arbitrary!
- Each position in a vector indicates a meaningful dimension in which a given unit (in this case, a document) can vary
  - AsYouLikeIt[0] = 1



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Linear Algebra Refresher

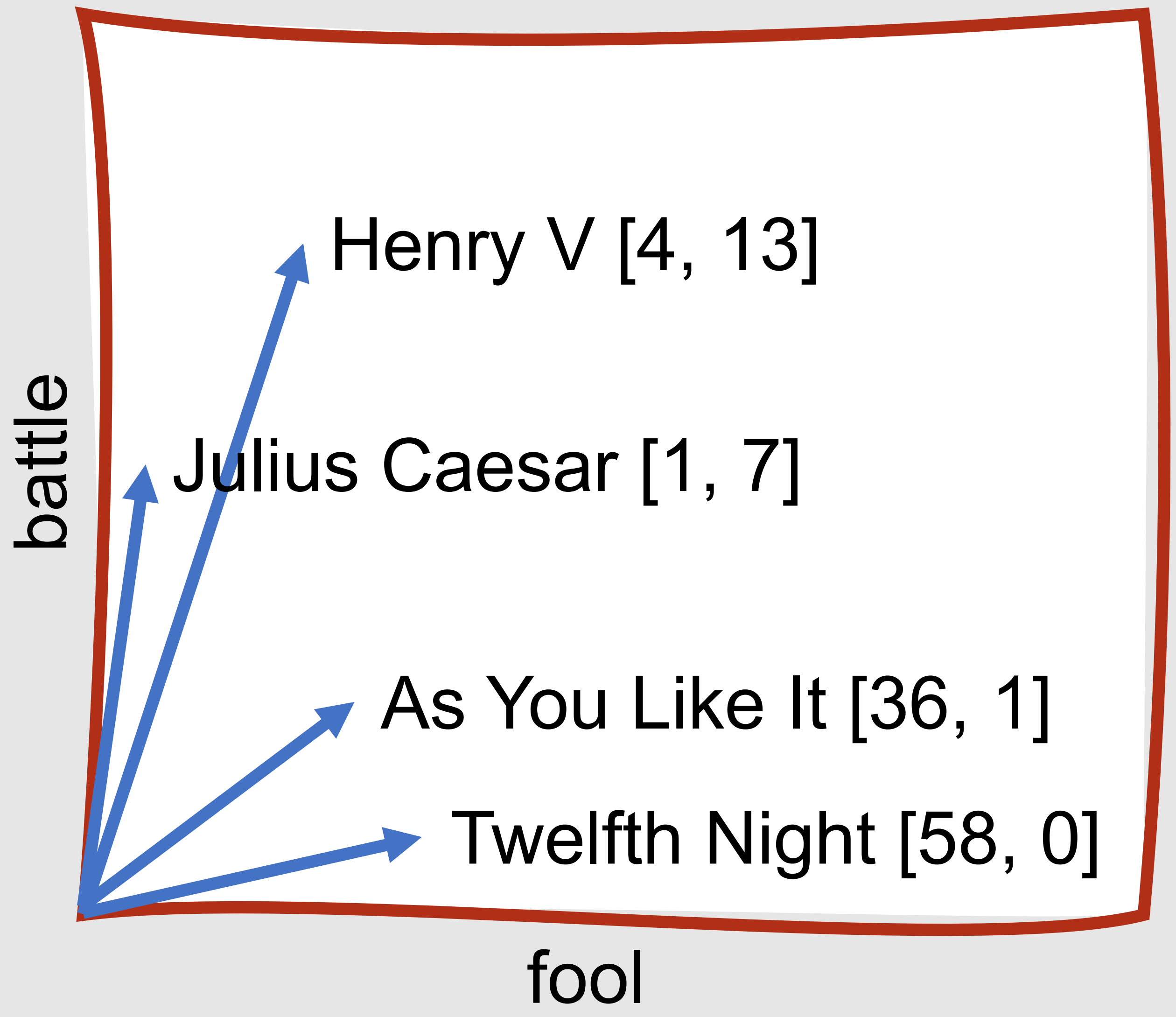
- Vectors indicate a unit's point in n-dimensional space



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

What can we infer from this visualization?

- As You Like It and Twelfth Night are closer (and therefore more similar!) to each other than to Julius Caesar or Henry V

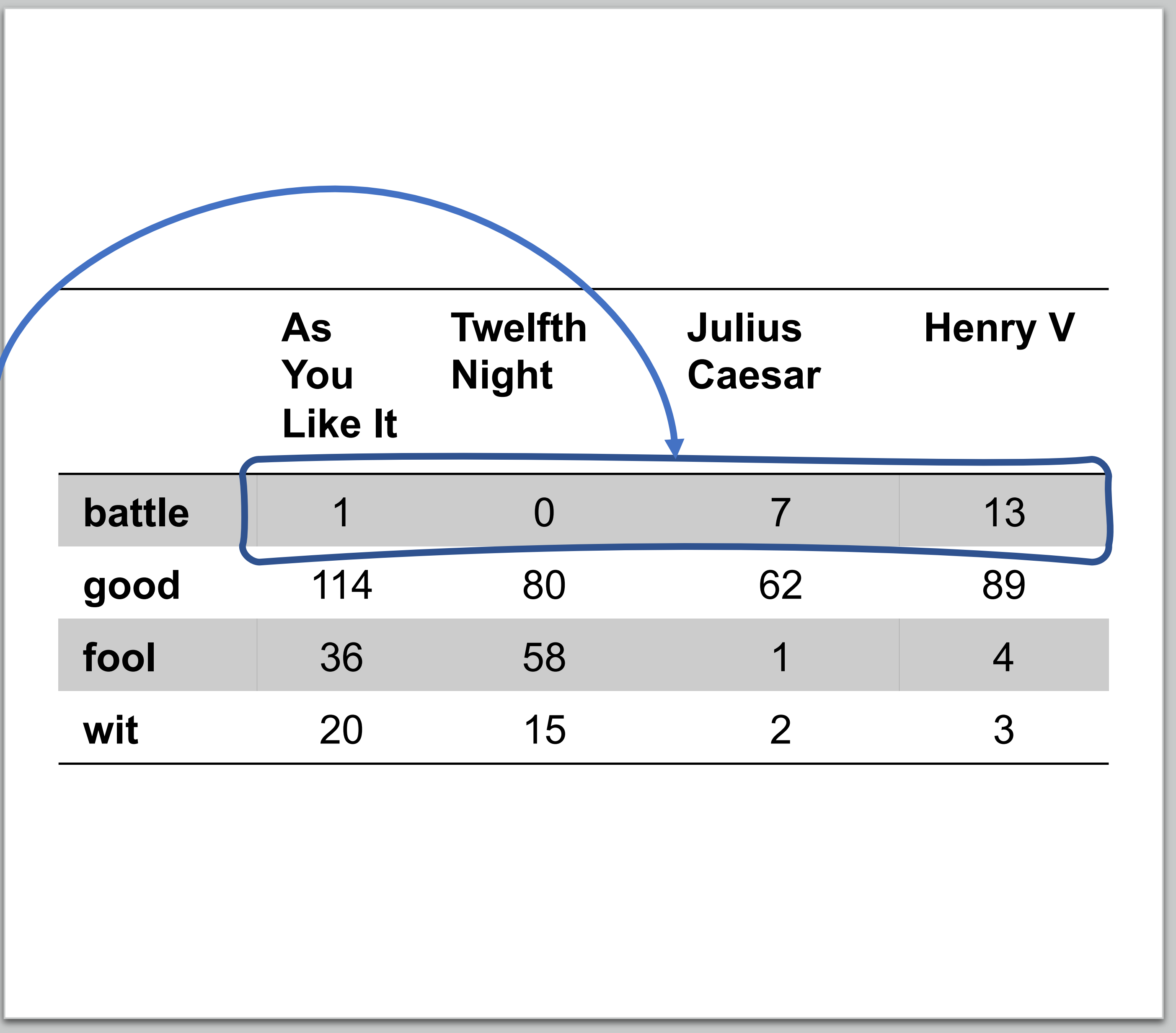


Real-world term-document matrices are much bigger!

- More generally, term-document matrices have |V| rows (one for each word type in the vocabulary) and D columns (one for each document in the collection)
- Vocabulary sizes are generally 10k+
- Number of documents can be enormous (e.g., all pages indexed by a web crawler)

Words as Vectors

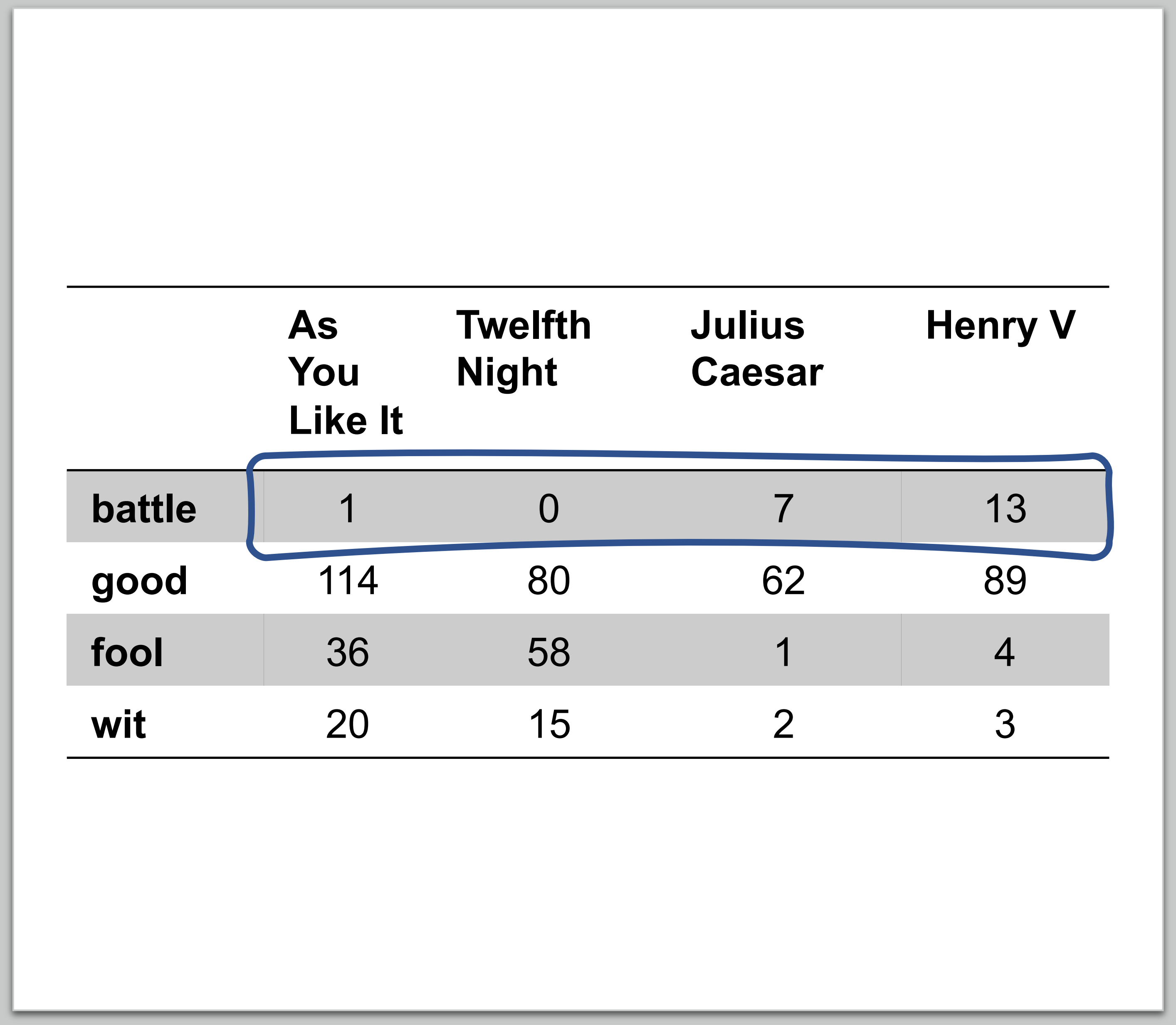
- In the previous example, we represented documents using vectors
- However, word embeddings represent **words** as vectors
  - battle = [1, 0, 7, 13]



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Words as Vectors

- In this scenario, similar words have similar vectors because they tend to occur in similar documents
- Thus, we're representing the meaning of a word by the documents in which it tends to occur



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Different Types of Context

- Documents aren't the most common type of context used to represent meaning in word vectors
- More common: **word context**
    - Referred to as a term-term matrix, word-word matrix, or term-context matrix
- In a **word-word matrix**, the columns are also labeled by words
    - Thus, dimensionality is |V| x |V|
    - Each cell records the number of times the row (target) word and the column (context) word co-occur in some context in a training corpus

How can you decide if two words occur in the same context?

- Common **context windows**:
  - Entire document
    - Cell value = number of times the words co-occur in the same document
  - Predetermined number of words surrounding the target
    - Cell value = number of times the words co-occur in this set of words

Example Context Window (Size = 4)

is	traditionally	followed	by	cherry	pie,	a	traditional	dessert
often	mixed,	such	as	strawberry	rhubarb	pie.	Apple	pie
computer	peripherals	and	personal	digital	assistants.	These	devices	usually
a	computer.	This	includes	information	available	on	the	internet

- We can take each occurrence of a word (e.g., strawberry) and count the context words around it to get a word-word co-occurrence matrix

Example Context Window (Size = 4)

is	traditionally	followed	by	cherry	pie,	a	traditional	dessert
often	mixed,	such	as	strawberry	rhubarb	pie.	Apple	pie
computer	peripherals	and	personal	digital	assistants.	These	devices	usually
a	computer.	This	includes	information	available	on	the	internet

	aardvark	…	computer	data	result	pie	sugar	…
cherry	0	…	2	8	9	442	25	…
strawberry	0	…	0	0	1	60	19	…
digital	0	…	1670	1683	85	5	4	…
information	0	…	3325	3982	378	5	13	…

vector for "digital"

- A simplified subset of a word-word co-occurrence matrix for the example words computed from Wikipedia could appear as shown

How can we measure the similarity between word vectors?

- **Cosine similarity**
  - Based on the **dot product** (also called **inner product**) from linear algebra
    - $\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_N w_N$
- Similar vectors (those with large values in the same dimensions) will have high values; dissimilar vectors (those with zeros in different dimensions) will have low values

Why don't we just use the dot product?

- More frequent words tend to co-occur with more words and have higher co-occurrence values with each of them
- Thus, the **raw dot product will be higher for frequent words**
- This is problematic!
  - We want our similarity metric to tell us how similar two words are regardless of frequency
- The simplest way to fix this problem is to **normalize for the vector length** (divide the dot product by the lengths of the two vectors)

Normalized Dot Product = Cosine of the angle between two vectors

- The cosine similarity metrics between two vectors v and w can thus be computed as:
  - $\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$
- This value ranges between 0 (dissimilar) and 1 (similar)

Example: Computing Cosine Similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

cos(cherry, information) = ?

Example: Computing Cosine Similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

cos(cherry, information) = $\frac{[442,8,2]\cdot[5,3982,3325]}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}}$

Example: Computing Cosine Similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\boxed{\text{cos(cherry, information)} = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}}}$$

Example: Computing Cosine Similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\boxed{\text{cos(cherry, information)} = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = 0.017}$$

Example: Computing Cosine Similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry, information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = 0.017$$

$$\cos(\text{digital, information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = 0.996$$

Example: Computing Cosine Similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\boxed{\cos(\text{cherry, information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = 0.017}$$

$$\boxed{\cos(\text{digital, information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = 0.996}$$

Result: *information* is way closer to *digital* than it is to *cherry*!

In-Class Exercise

- Compute the cosine similarity to determine whether **pumpkin** or **squash** is closer to **halloween**.
- https://www.google.com/search?q=timer

	pie	jack-o'-lantern	zucchini
pumpkin	3	3	2
squash	2	1	3
halloween	2	3	1

In-Class Exercise

	pie	jack-o'-lantern	zucchini
pumpkin	3	3	2
squash	2	1	3
halloween	2	3	1

$$\boxed{\text{cos(pumpkin, halloween)} = \frac{3*2+3*3+2*1}{\sqrt{3^2+3^2+2^2}\sqrt{2^2+3^2+1^2}} = \frac{17}{\sqrt{22}\sqrt{14}} = 0.969}$$

$$\boxed{\text{cos(squash, halloween)} = \frac{2*2+1*3+3*1}{\sqrt{2^2+1^2+3^2}\sqrt{2^2+3^2+1^2}} = \frac{10}{\sqrt{14}\sqrt{14}} = 0.714}$$

So far, the co-occurrence matrices have contained raw frequency counts of word co-occurrences.

- However, this isn't the best measure of association between words
  - Some words co-occur frequently with many words, so won't be very informative
    - the, it, they
- We want to know about **words that co-occur frequently with one another, but less frequently across all texts**

How do we compute this?

- **TF*IDF**
  - Term Frequency * Inverse Document Frequency
- **Term Frequency:** The frequency of the word *t* in the document *d*
  - $tf_{t,d} = \text{count}(t, d)$
- **Document Frequency:** The number of documents in which the word *t* occurs
  - Different from collection frequency (the number of times the word occurs in the entire collection of documents)

Computing TF*IDF

- **Inverse Document Frequency:** The inverse of document frequency, where *N* is the total number of documents in the collection
  - $idf_t = \frac{N}{df_t}$
- IDF is higher when the term occurs in fewer documents
- What is a document?
  - Depends on your corpus!
    - Website
    - Book
    - Play
    - Article
    - Etc.
- If desired (e.g., if the document collection is large), we can squash both TF and IDF by using the $\log_{10}(tf_{t,d}+1)$ and $\log_{10} idf_t$

Computing TF*IDF

- TF*IDF is then simply the combination of TF and IDF
  - $tfidf_{t,d} = tf_{t,d} \times idf_t$

Example: Computing TF*IDF

- TF*IDF(battle, As You Like It) = ?

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF*IDF

- TF*IDF(battle, As You Like It) = ?
- TF(battle, As You Like It) = 1

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF*IDF

- TF*IDF(battle, As You Like It) = ?
- TF(battle, As You Like It) = 1
- IDF(battle) = N/DF(battle) = 4/3 = 1.33

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF*IDF

- TF*IDF(battle, As You Like It) = ?
- TF(battle, As You Like It) = 1
- IDF(battle) = N/DF(battle) = 4/3 = 1.33
- TF*IDF(battle, As You Like It) = 1 * 1.33 = 1.33

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF*IDF

- TF*IDF(battle, As You Like It) = ?
- TF(battle, As You Like It) = 1
- IDF(battle) = N/DF(battle) = 4/3 = 1.33
- TF*IDF(battle, As You Like It) = 1 * 1.33 = 1.33
- Alternately, TF*IDF(battle, As You Like It) = $\log_{10}(1+1) * \log_{10} 1.33 = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Extending TF*IDF Computation

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	df	idf
battle	21	$\log_{10}\frac{37}{21} = 0.246$
good	37	$\log_{10}\frac{37}{37} = 0.000$
fool	36	$\log_{10}\frac{37}{36} = 0.012$
wit	34	$\log_{10}\frac{37}{34} = 0.037$

- TF*IDF(battle, As You Like It) = ?
- TF(battle, As You Like It) = 1
- IDF(battle) = N/DF(battle) = 0.246
- TF*IDF(battle, As You Like It) = 1 * 0.246 = 0.246
- Alternately, TF*IDF(battle, As You Like It) = $\log_{10}(1+1) * 0.246 = 0.074$

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	**idf**
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074			
good				
fool				
wit				

- TF*IDF(battle, Twelfth Night) = ?

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074			
good				
fool				
wit				

- TF*IDF(battle, Twelfth Night) = ?
- TF(battle, Twelfth Night) = 0

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074			
good				
fool				
wit				

- TF*IDF(battle, Twelfth Night) = ?
- TF(battle, Twelfth Night) = 0
- IDF(battle) = N/DF(battle) = 0.246

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074			
good				
fool				
wit				

- TF*IDF(battle, Twelfth Night) = ?
- TF(battle, Twelfth Night) = 0
- IDF(battle) = N/DF(battle) = 0.246
- TF*IDF(battle, Twelfth Night) = $\log_{10}(0+1) * 0.246 = 0.000$

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000		
good				
fool				
wit				

- TF*IDF(battle, Twelfth Night) = ?
- TF(battle, Twelfth Night) = 0
- IDF(battle) = N/DF(battle) = 0.246
- TF*IDF(battle, Twelfth Night) = $\log_{10}(0 + 1) * 0.246 = 0.000$

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000		
good				
fool				
wit				

- TF*IDF(good, As You Like It) = ?

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000		
good				
fool				
wit				

- TF*IDF(good, As You Like It) = ?
- TF(good, As You Like It) = 114

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000		
good				
fool				
wit				

- TF*IDF(good, As You Like It) = ?
- TF(good, As You Like It) = 114
- IDF(good) = N/DF(good) = 0.000

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000		
good				
fool				
wit				

- TF*IDF(good, As You Like It) = ?
- TF(good, As You Like It) = 114
- IDF(good) = N/DF(good) = 0.000
- TF*IDF(good, As You Like It) = $\log_{10}(114+1) * 0.000 = 0.000$

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000		
good	0.000			
fool				
wit				

- TF*IDF(good, As You Like It) = ?
- TF(good, As You Like It) = 114
- IDF(good) = N/DF(good) = 0.000
- TF*IDF(good, As You Like It) = $\log_{10}(114 + 1) * 0.000 = 0.000$

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

Example: Converting TF matrix to TF*IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word	idf
battle	$\log_{10}\frac{37}{21} = 0.246$
good	$\log_{10}\frac{37}{37} = 0.000$
fool	$\log_{10}\frac{37}{36} = 0.012$
wit	$\log_{10}\frac{37}{34} = 0.037$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

- As shown, using TF*IDF eliminates the importance of the ubiquitous “good”
- It also reduces the impact of the almost-ubiquitous “fool”
- It increases the importance of the rarer word “battle”

Applications of TF*IDF Models

- The TF*IDF model produces a sparse (most cells have values of 0) vector with TF*IDF values in each cell
- Common uses of this model:
  - Computing word similarity
  - Computing document similarity
    - Find the vectors of all words in a document, compute the centroid of those vectors, and then compute the similarity between two document centroids

Summary: Word Embeddings (Part 1)

- **Word embeddings** are vector representations of meaning
- A vector for a word is computed based on the **contexts** in which the word occurs
  - Context = Documents or windows of words
- Word embeddings can be **sparse** or **dense**
  - **Sparse:** Bag of words, TF*IDF
  - **Dense:** Word2Vec, GloVe, ELMo, BERT
- **TF*IDF** vectors represent a word's meaning based on a combination of term frequency and inverse document frequency
- **Cosine similarity** can be used to determine the similarity between two word vectors