# CS 481

## *Artificial Intelligence Language Understanding*

**March 28, 2023**

# Announcements / Reminders

- **Please follow the Week 20 To Do List instructions**
- **Written Assignment #03 is due on ~~Sunday 03/26/23~~ TONIGHT at 11:59 PM CST**
- **Programming Assignment #02 is due on Sunday 04/02/23 at 11:59 PM CST**

- **Final Exam date:**

  **Thursday 04/27/2023 (last week of classes!)**
  - **Ignore the date provided by the Registrar**
  - **Section 02 [Online]: contact Mr. Charles Scott (scott@iit.edu) to arrange your exam**

# Plan for Today

- **Sentiment Analysis**
- **Words and their meaning**

# Challenge

- We know word relationships exist
- How can we quantify them in a automated fashion?
- How do we represent them in numerical way?
- How can we use them in computational models and processes?

# Computational Models of Meaning

"*a word is characterized by the company it keeps*"

**- John Rupert Firth (English linguist)**

"*In most cases, the meaning of a word is its use*"

**- Ludwig Wittgenstein (Austrian philosopher)**

"*If A and B have almost identical environments we say that they are synonyms.*"

**- Zellig Harris (American linguist)**

# Words + Their Environment: Example

- **Suppose you see these sentences:**
  - *Ong choi is delicious sautéed with garlic.*
  - *Ong choi is superb over rice*
  - *Ong choi leaves with salty sauces*
- **And you've also seen these:**
  - *…spinach sautéed with garlic over rice*
  - *Chard stems and leaves are delicious*
  - *Collard greens and other salty leafy greens*
- **Conclusion:**
  - **Ong choi is a leafy green like spinach, chard, or collard greens**
  - **We could conclude this based on words like "*leaves*" and "*delicious*" and "*sautéed*"**

# Computational Models of Meaning

- So:
  - words are defined by their environments (the words around them)
- How can we represent word meaning with word environment?
  - Vector semantics

# Vector Semantics: Two Ideas

- **Idea 1:**
  - **Let's define the meaning of a word by its distribution in language use (neighboring words or grammatical environments)**
- **Idea 2:**
  - **Let's define the meaning of a word as a point in space**

# Bag of Words: Strings Representation

| Some document: |
|---|
| I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet! |

| Word: | Frequency: |
|---|---|
| it | 6 |
| I | 5 |
| the | 4 |
| to | 3 |
| and | 3 |
| seen | 2 |
| yet | 1 |
| whimsical | 1 |
| times | 1 |
| .... | ... |

**Bag of words assumption:** word/token position does not matter.

# Bag of Words: Meaning Ignored!

| Some document: |
|---|
| I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet! |

| Word: | Frequency: |
|---|---|
| it | 6 |
| I | 5 |
| the | 4 |
| to | 3 |
| and | 3 |
| seen | 2 |
| yet | 1 |
| whimsical | 1 |
| times | 1 |
| .... | ... |

**Bag of words assumption:** word/token position does not matter.

# Connotation as a Point in Space

- Words seem to vary along <u>**three affective DIMENSIONS**</u>:
  - **valence**: the pleasantness of the stimulus
  - **arousal**: the intensity of emotion provoked by the stimulus
  - **dominance**: the degree of control exerted by the stimulus

| | Word | Score | | Word | Score |
|---|---|---|---|---|---|
| valence | love | 1.000 | | toxic | 0.008 |
| | happy | 1.000 | | nightmare | 0.005 |
| arousal | elated | 0.960 | | mellow | 0.069 |
| | frenzy | 0.965 | | napping | 0.046 |
| dominance | powerful | 0.991 | | weak | 0.045 |
| | leadership | 0.983 | | empty | 0.081 |

*Source: NRC VAD Lexicon (https://saifmohammad.com/WebPages/nrc-vad.html)*

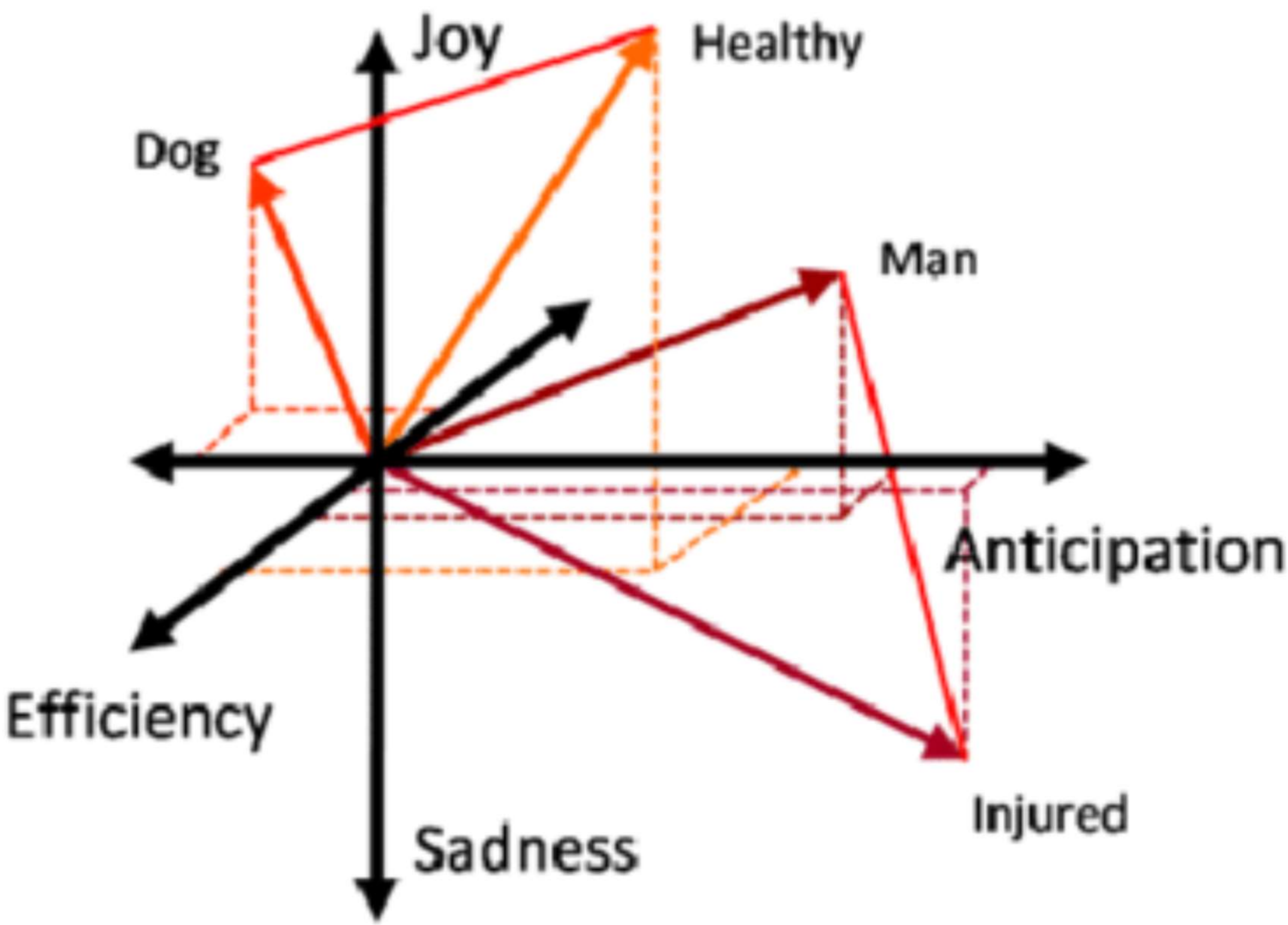
# Vector Semantics

- **The idea:**
  - **represent a word as a point in a multidimensional semantic space that is derived from the distributions of word neighbors**

# Point in Space Based on Distribution

- Each **word = a vector**
  - not just "*good*" or "$word_{45}$"
- **Similar words: "nearby in semantic space"**
- **We build this space automatically by seeing which words are nearby in text**

# Vector Semantics: Words as Vectors




*Source: Signorelli, Camilo & Arsiwalla, Xerxes. (2019). Moral Dilemmas for Artificial Intelligence: a position paper on an application of Compositional Quantum Cognition*

# Word Embedding: Definition

## Word Embedding:

*a term used for the **representation of words** for text analysis, **typically in the form of a real-valued vector** that **encodes the meaning of the word such** that the **words that are closer in the vector space are expected to be similar** in meaning*

**from Wikipedia**

# Word Embedding

- Embedding:
  - "embedded into a space"
  - mapping from one space or structure to another
- The standard way to represent meaning in NLP
- Fine-grained model of meaning for similarity

# The Why: Sentiment Analysis

- Using **words only**:
  - a feature is a word identity
  - for example
    - feature $x_5 = \begin{cases} 1 \; \textit{if the previous word was 'terrible'} \\ 0 \; \textit{otherwise} \end{cases}$
  - requires exact same word to be in training and test

# The Why: Sentiment Analysis

- Using embeddings:
  - a feature is a word vector
  - the previous word was vector [35, 22, 17]
  - now in the test set we might see a similar vector [34, 21, 14]
  - we can generalize to similar but unseen words

# Term-Document Matrix

- **Each document is represented by a vector of words**

| | 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 |

# Term-Document Matrix

- **Vectors are similar for the two comedies**
  - **“As you like it” and “Twelfth Night”**

| | 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 |

- **But comedies are different than the other two**
  - **more *fools* and *wit* and fewer *battles***

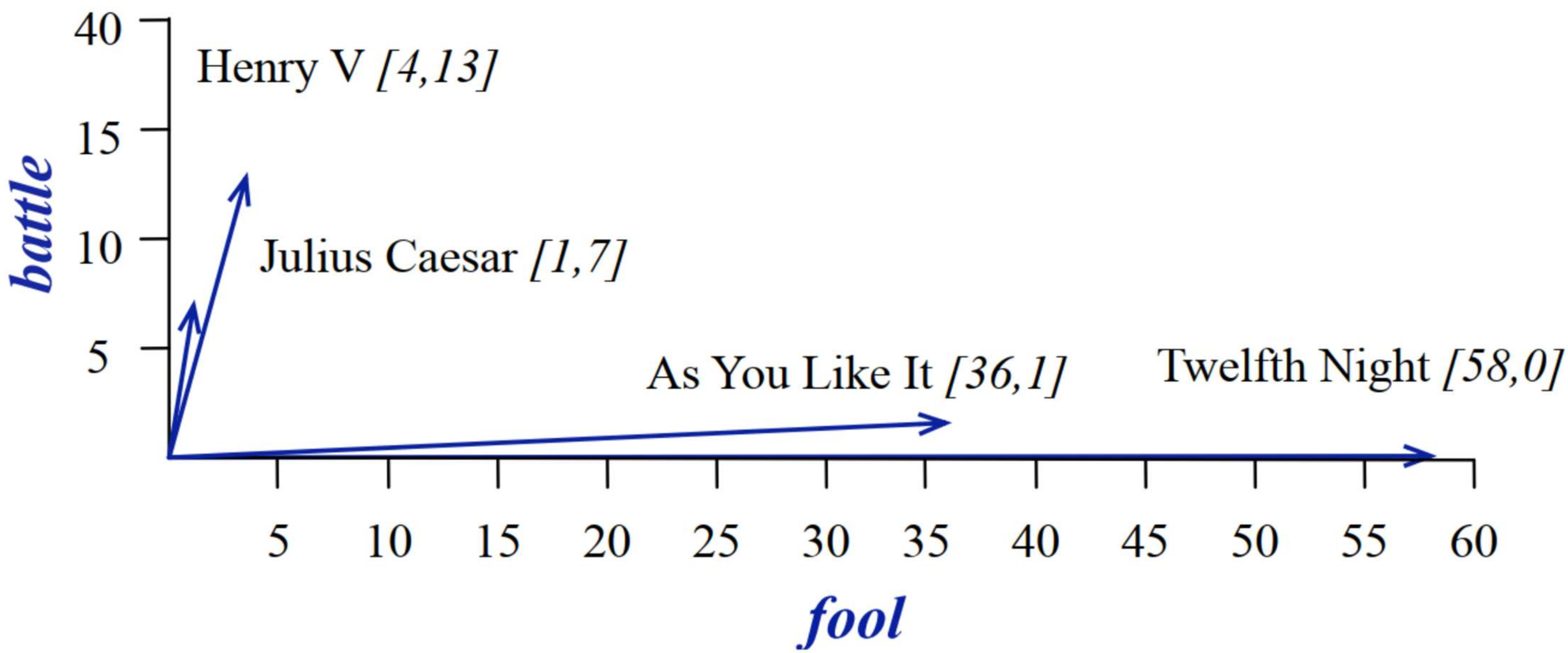
# Term-Document Matrix

- **Vectors are similar for the two comedies**
  - **“As you like it” and “Twelfth Night”**

| | 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 |

- **But comedies are different than the other two**
  - **more *fools* and *wit* and fewer *battles***

# Document Vector Visualization



# Words as Vectors

- *battle* is "the kind of word that occurs in Julius Caesar and Henry V"

| | 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 |

- *fool* is "the kind of word that occurs in comedies, especially Twelfth Night"

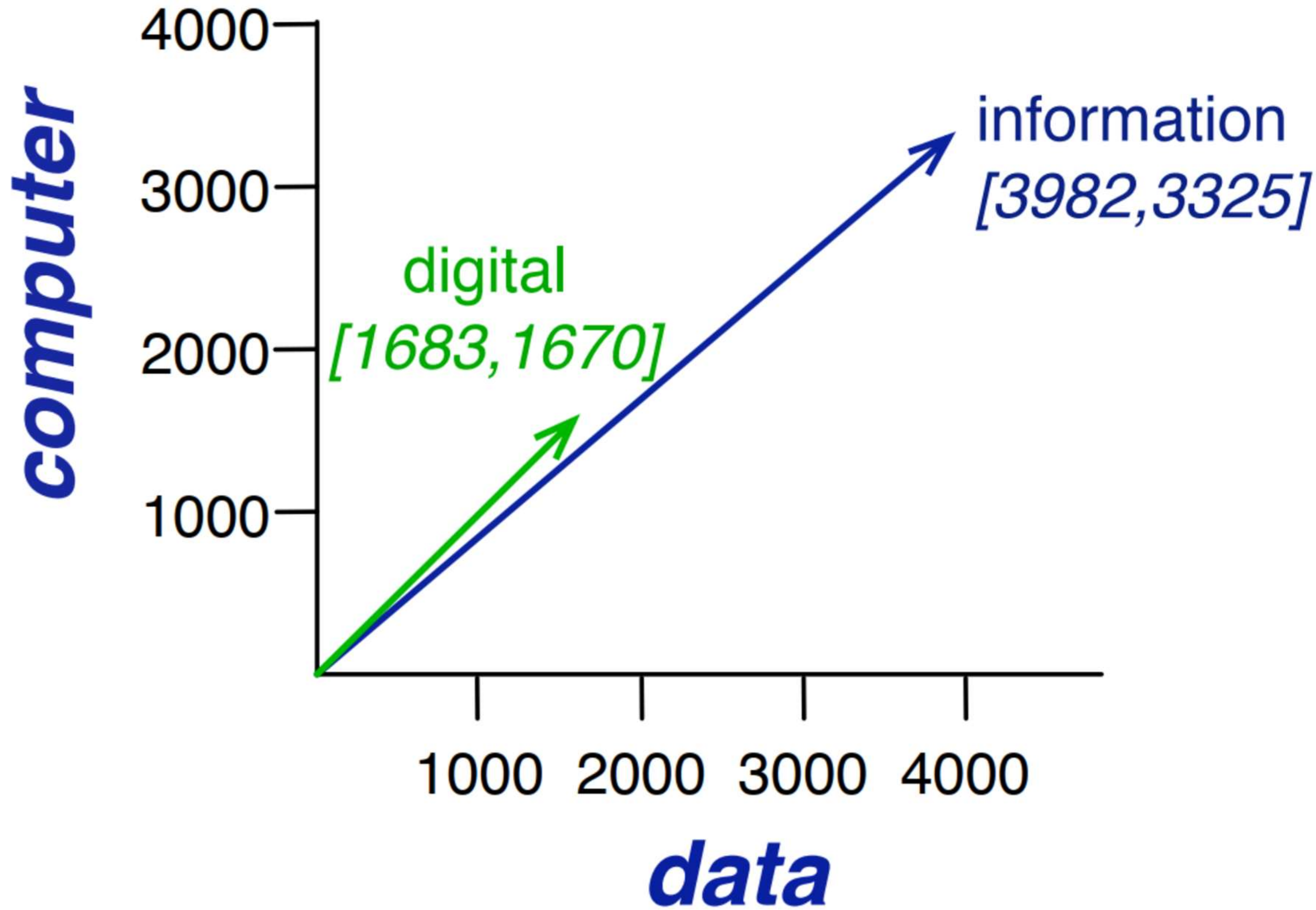
# Word-Word (Term-Context) Matrix

- **Two words are similar in meaning if their context vectors are similar**

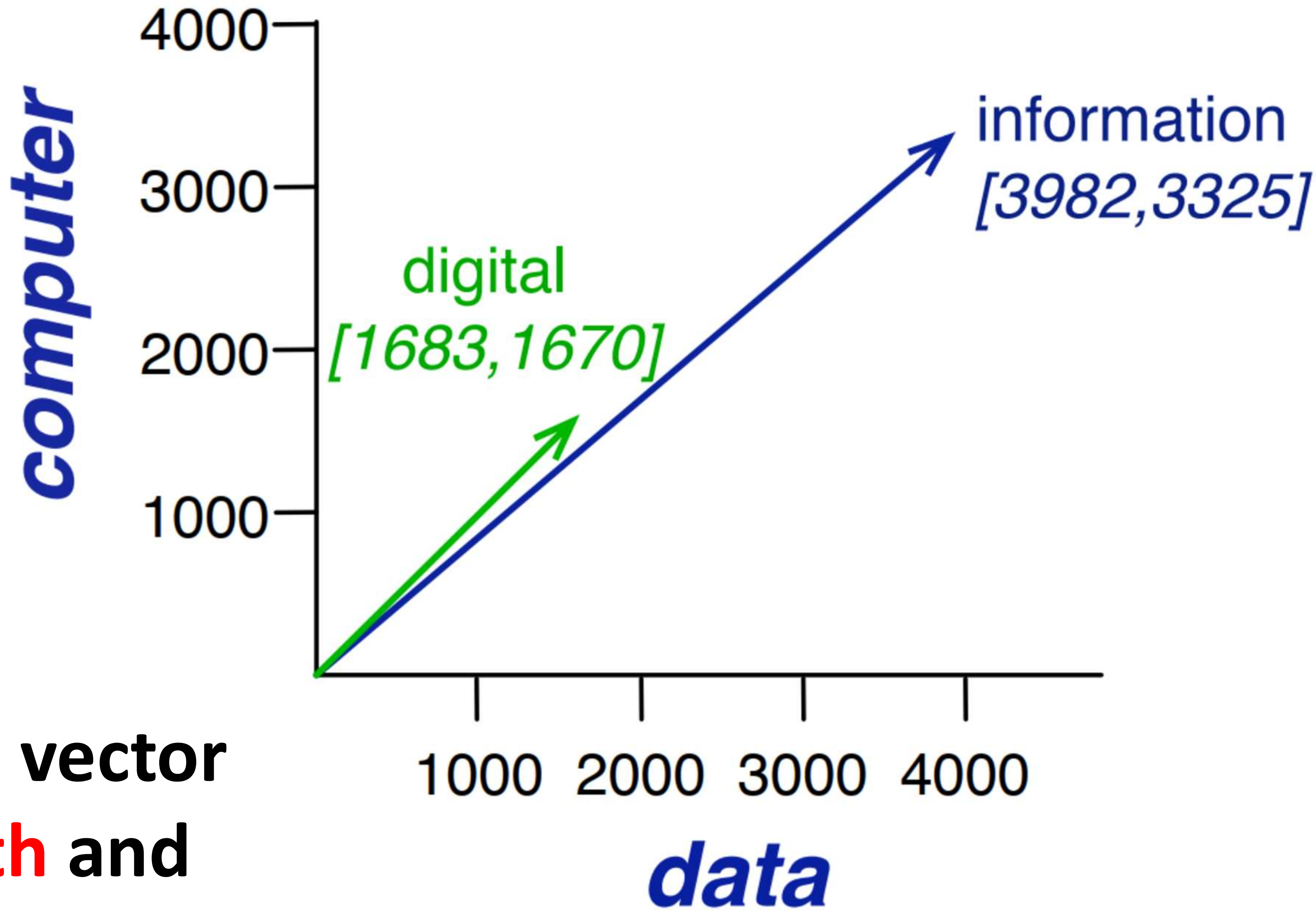
| | | |
|---|---|---|
| is traditionally followed by | **cherry** | pie, a traditional dessert |
| often mixed, such as | **strawberry** | rhubarb pie. Apple pie |
| computer peripherals and personal | **digital** | assistants. These devices usually |
| a computer. This includes | **information** | available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

# Document Vector Visualization



# Document Vector Visualization




**Note vector length and direction**

# Vector Dot / Scalar Product

**Given two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ ($N$ - vector space dimension):**

$$\boldsymbol{a} = [a_1, a_2, \dots, a_N] \text{ and } \boldsymbol{b} = [b_1, b_2, \dots, b_N]$$

**their vector dot/scalar product is:**

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{N} a_i * b_i = a_1 * b_1 + a_2 * b_2 + \dots + a_N * b_N$$

**Using matrix representation:**

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^{\boldsymbol{T}} = [a_1 \quad a_2 \quad \cdots \quad a_N] \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

# Vector Dot / Scalar Product

- **Vector dot/scalar product is a scalar:**

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{N} a_i * b_i = a_1 * b_1 + a_2 * b_2 + \ldots + a_N * b_N$$

- **Vector dot/scalar:**
  - **high values when the two vectors have large values in the same dimensions**
  - **useful similarity measure**

# Vector Dot / Scalar Product: Problem

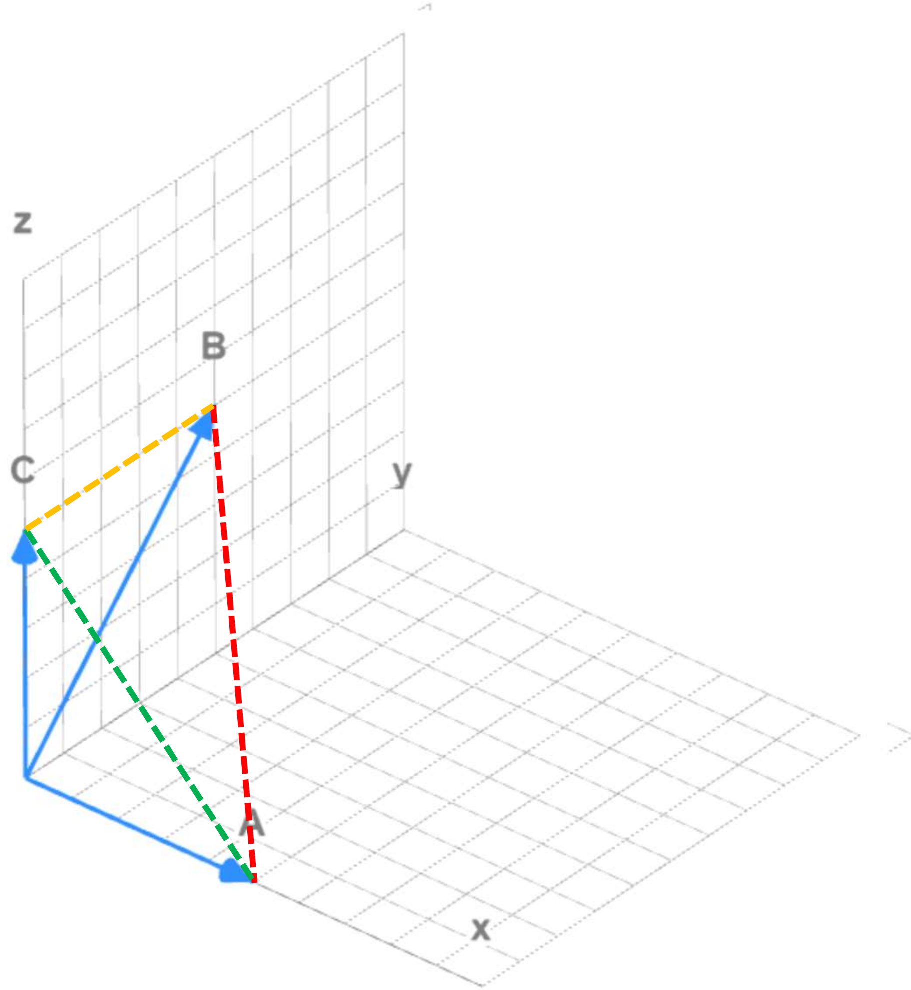
- Dot product **favors long vectors:** higher if a vector is longer (has higher values in many dimension)
- Vector length:

$$|\boldsymbol{a}| = \sqrt{\sum_{i=1}^{N} a_i^2}$$

- Frequent words (of, the, you) have long vectors (since they occur many times with other words).
- dot product overly **favors frequent words**

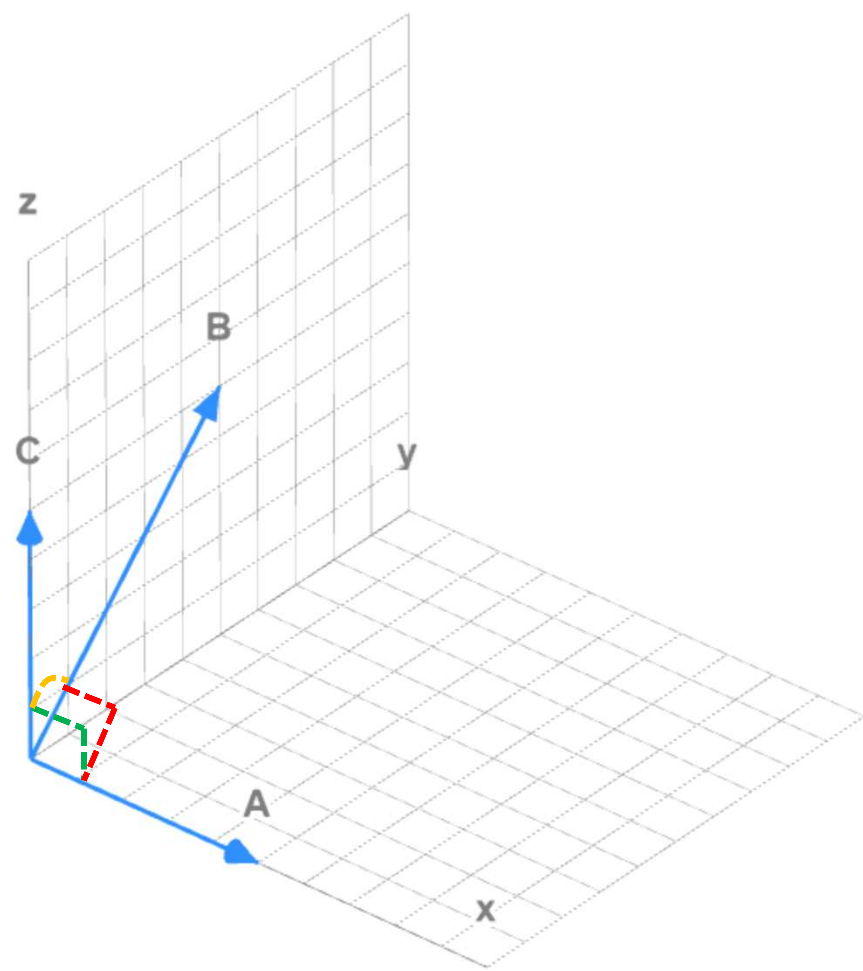
# Alternative: Cosine Similarity

**Euclidean distance**




$$D(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2)}$$

**Cosine similarity**




$$D(x, y) = cos(\theta) = \frac{x \cdot y}{||x||\, ||y||}$$

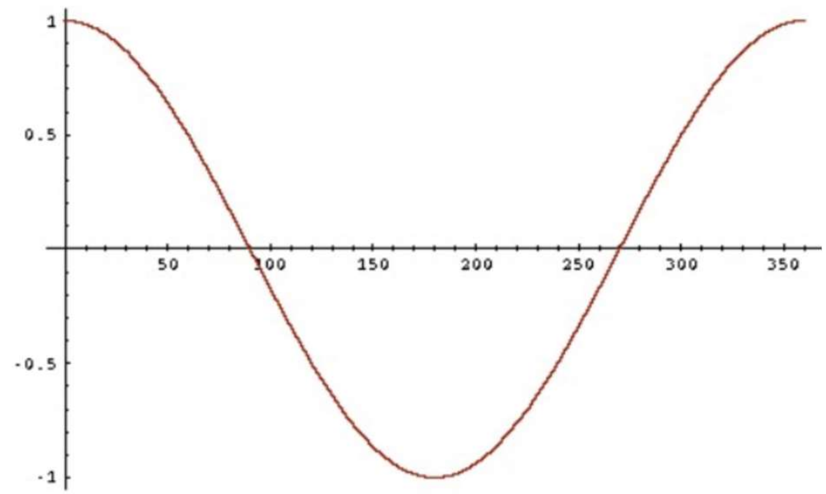
# Word Similarity | Cosine Similarity

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

**Where:** v **and** w **are two different word vectors**

# Word Similarity | Cosine Similarity

- **-1: vectors point in opposite directions**
- **+1: vectors point in same directions**
- **0: vectors are orthogonal**




**But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1**

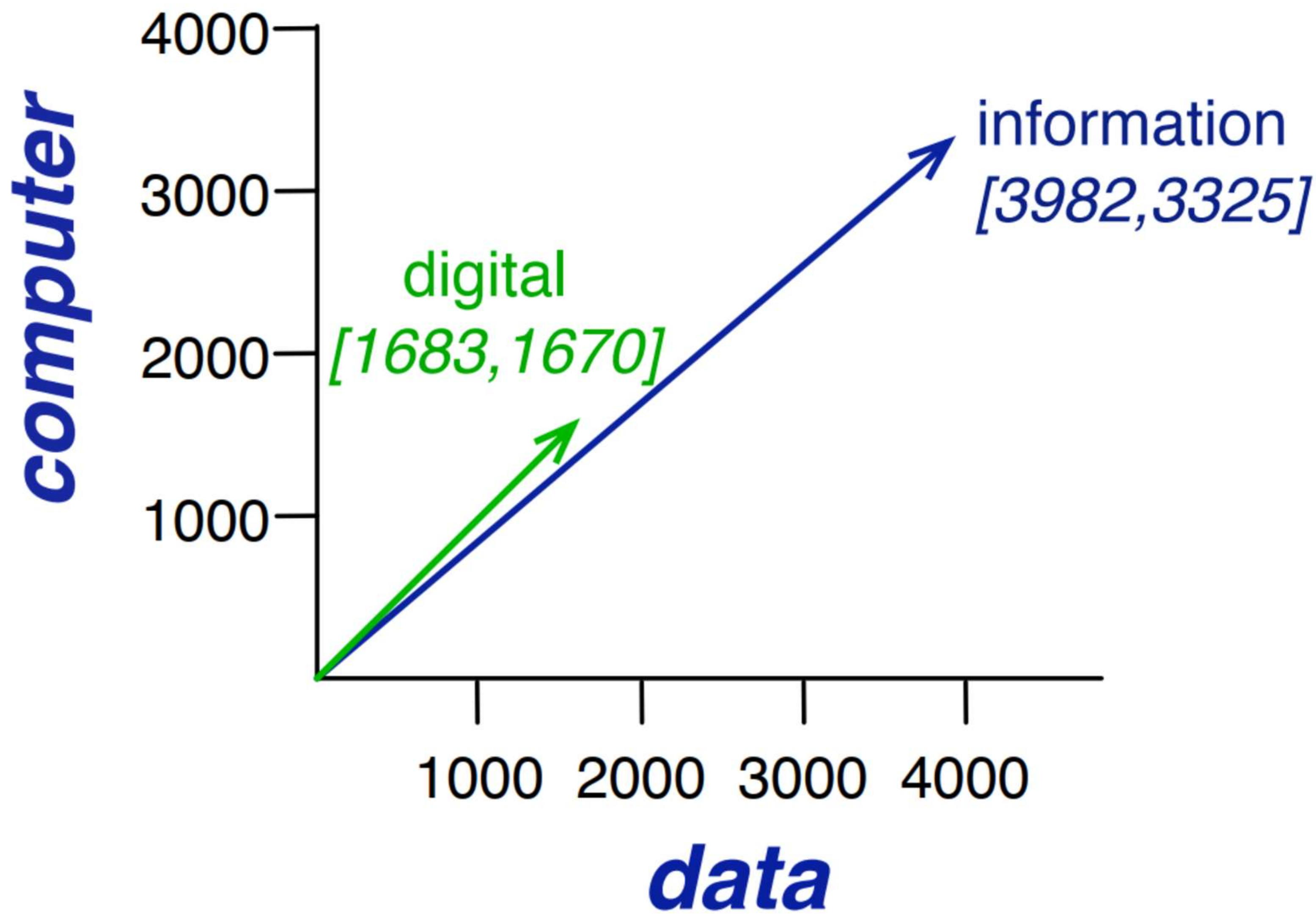
# Word Similarity

- **Two words are similar in meaning if their context vectors are similar**

| | | |
|---|---|---|
| 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 | ... |

# Word Similarity Visualization



# Word Similarity | Cosine Similarity

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

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

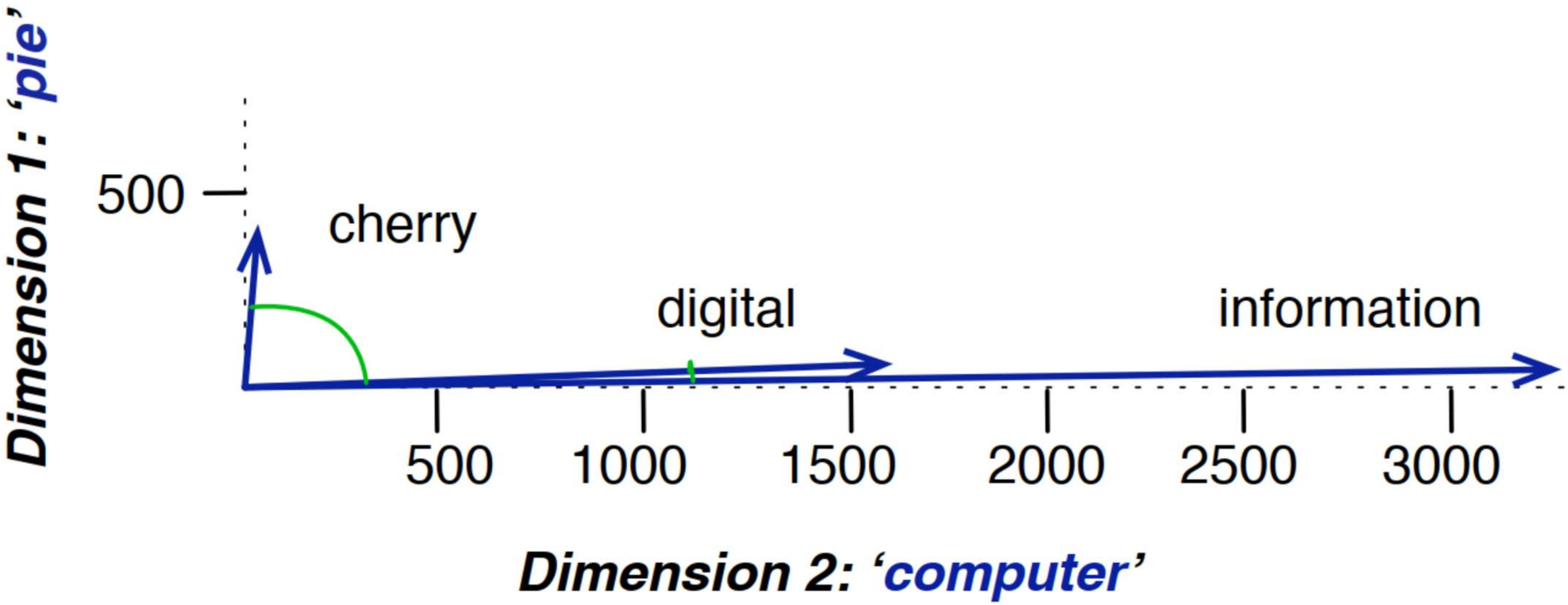
$$\cos(\text{cherry}, \text{information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

**Low similarity**

**High similarity**

# Cosine Similarity Visualization



# Vector Embeddings: Methods

- `tf-idf`
  - **popular inInformation Retrieval**
  - **sparse vectors**
  - **word represented by (a simple function of) the counts of nearby words**

- `Word2vec`
  - **dense vectors**
  - **representation is created by training a classifier to predict whether a word is likely to appear nearby**

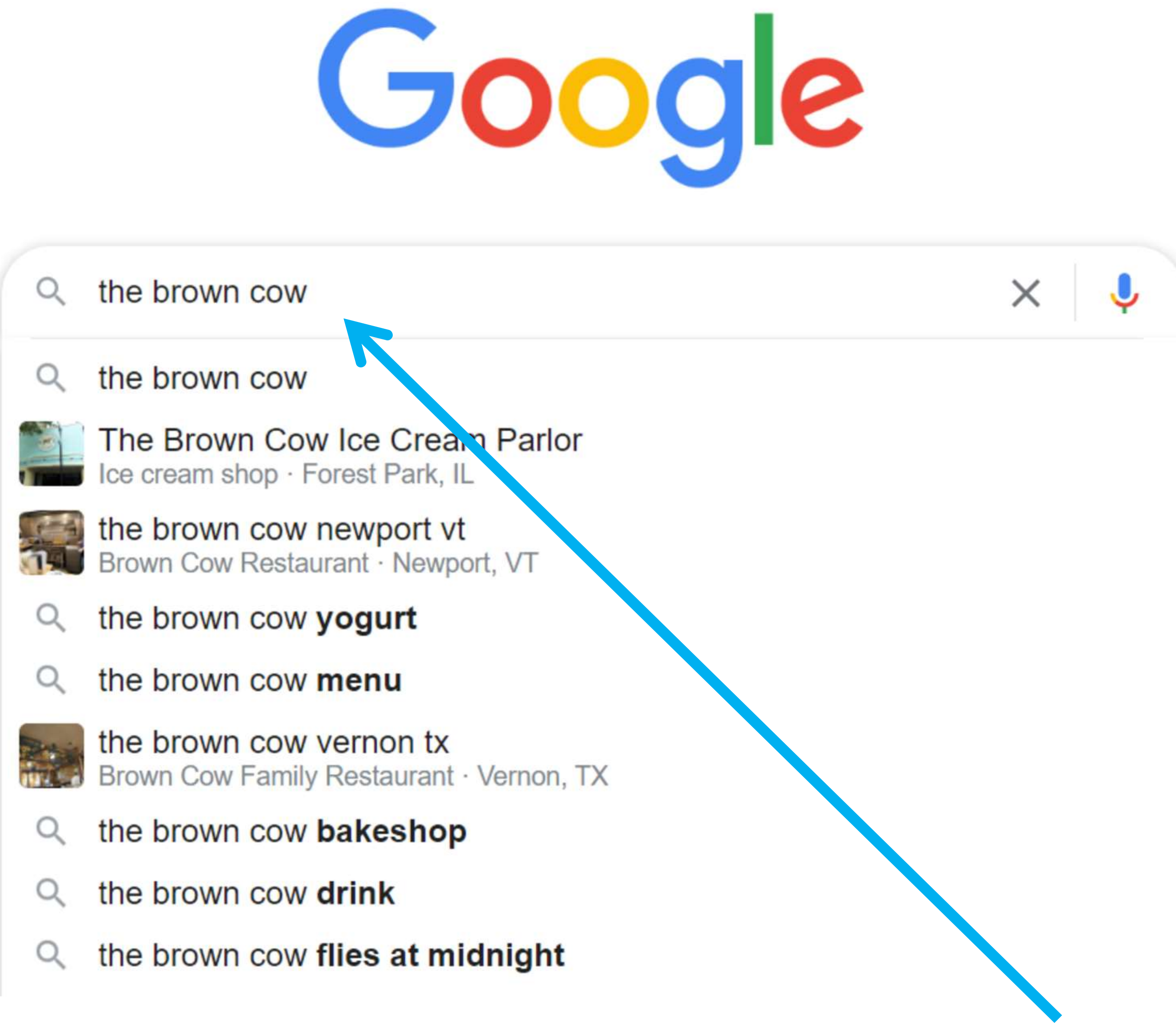
# Sparse vs. Dense Vectors

- **Sparse vectors have a lot of values set to zero.**
- **Dense vector: most of the values are non-zero.**
  - **better use of storage**
  - **carries more information**

# `tf-idf`: Frequencies Not Enough

- **The co-occurrence matrices we have seen represent each cell by word frequencies**
- **Frequency is clearly useful:**
  - **if *sugar* appears a lot near *apricot*, that's useful information**
- **But overly frequent words like *the*, *it*, or *they* are not very informative about the context**
- **It's a paradox! How can we balance these two conflicting constraints?**

# `tf-idf`: Motivation

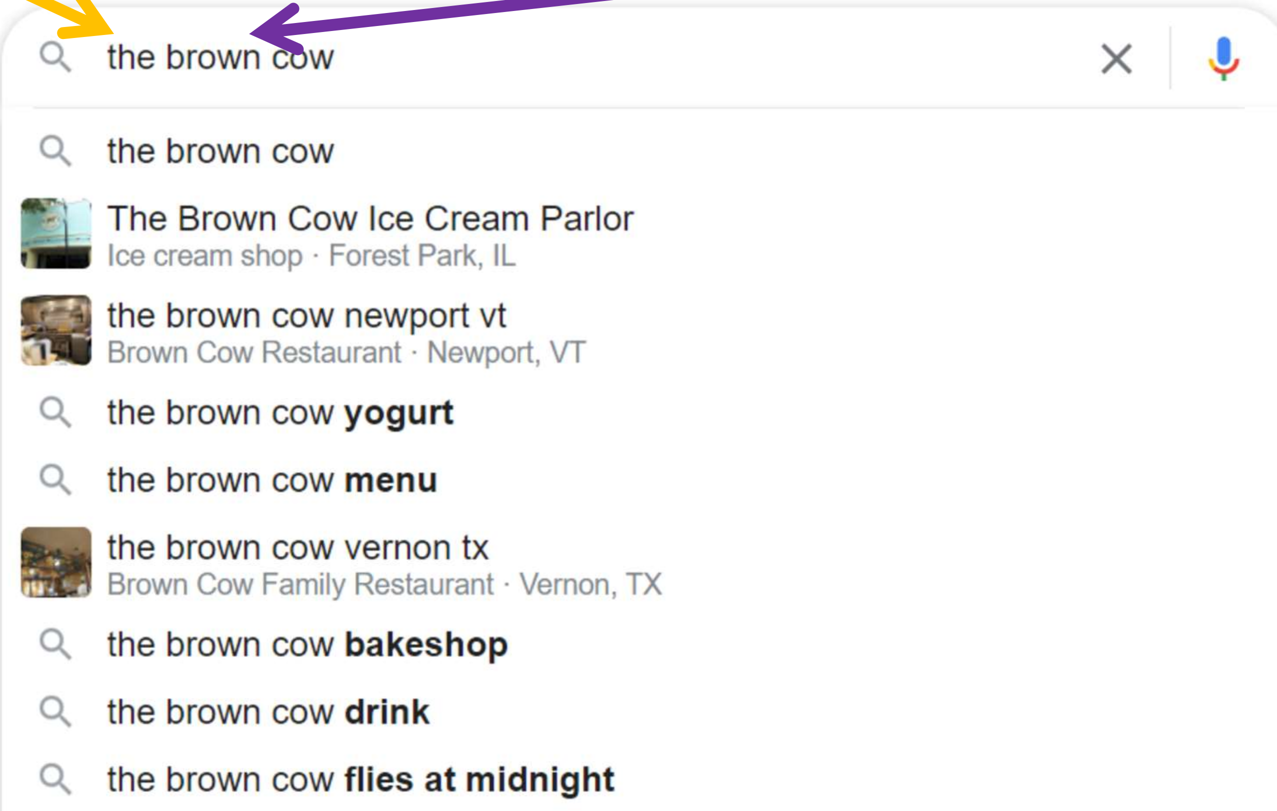



**we want to find documents most relevant to the search term**

# tf-idf: Motivation

"the" is a very common word = high frequency

"brown" and "cow" are less common words = lower frequency

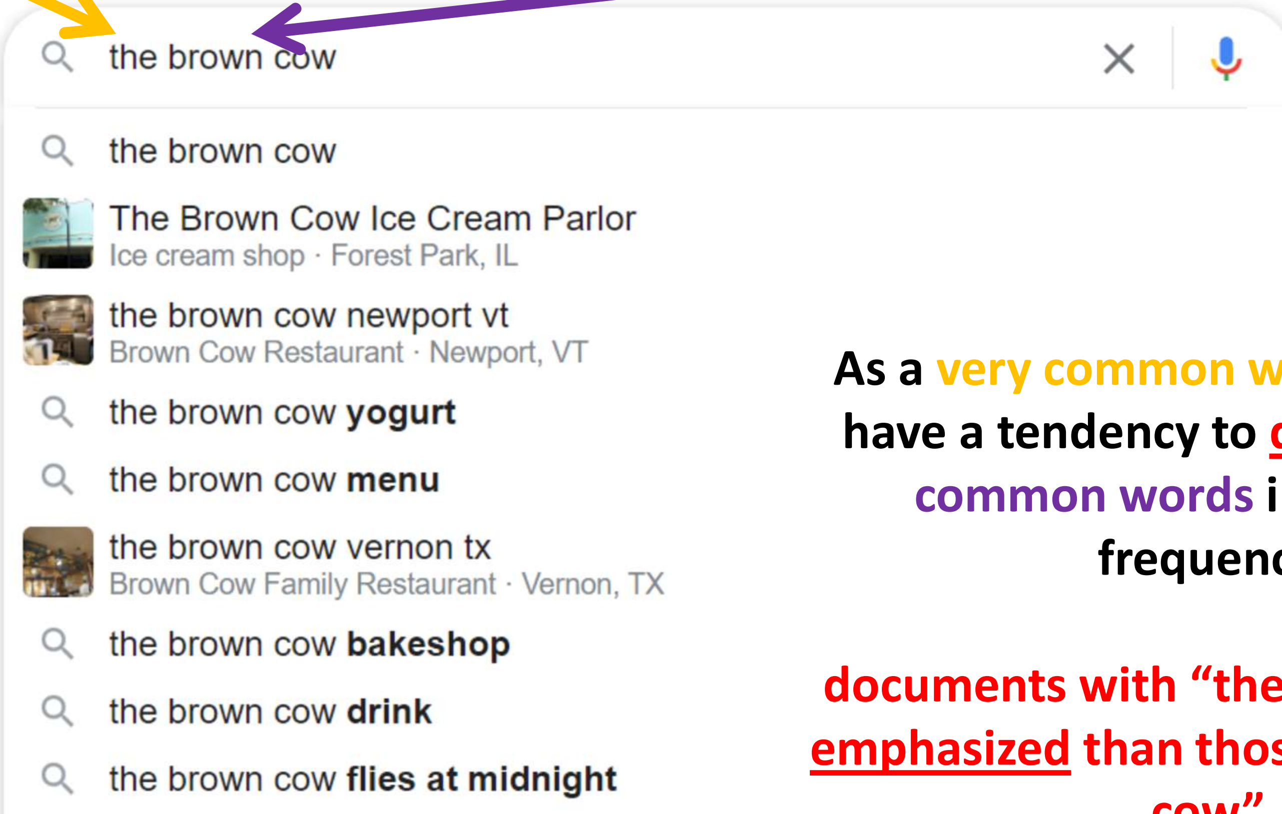



**we want to find documents most relevant to the search term**

# tf-idf: Motivation

"the" is a very common word = high frequency

"brown" and "cow" are less common words = lower frequency




As a very common word "the" will have a tendency to dominate less common words in terms of frequency

documents with "the" will be more emphasized than those with "brown cow"

we want to find documents most relevant to the search term

# `tf-idf`: Motivation

- **Problem 1: Vectorization can be "not representative"**
  - **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 drown 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

**term frequency – inverse document frequency**

# `tf-idf`: Idea and Intuition

- **Idea:**
  - **define a "score" for each component of the document vector associated with a given word w, broken down into two parts:**
    - **some measure of the frequency of w within the individual document (`tf`)**
    - **some measure of the rarity of w across all documents (corpus)(`idf`)**
- **Intuition:**
  - **`tf` can be chosen so that a lot of occurrences doesn't add up to too high a score (to address Problem 1),**
  - **`idf` will penalize the score for words that show up all over the place (to address Problem 2)**

# tf-idf

## term frequency – inverse document frequency

how often does the term (word) appear in a specific document?

1 / how often does the term (word) appear in the entire corpus?

# tf-idf

**term frequency – inverse document frequency**

**how often does the term (word) appear in a specific document?**

**how distintictive is the term (word) for a specific document?**

**1 / how often does the term (word) appear in the entire corpus?**

**1 / how common / popular is the term (word) within the entire corpus?**

# tf-idf

## term frequency – inverse document frequency

how often does the term (word) appear in a specific document?

1 / how often does the term (word) appear in the entire corpus?

how distintictive is the term (word) for a specific document?

1 / how common / popular is the term (word) within the entire corpus?

we want to balance out term frequency with inverse document frequency

# tf-idf

## term frequency – inverse document frequency

tf(word, document $d_i$) =

document $d_i$

= count( .... word .... word ... ...word ......word .... word.... )

idf(word, corpus C) =

$$= \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents with word}}\right)$$

# tf-idf

## term frequency – inverse document frequency

$tf(\text{word}, \text{document } d_i) =$

document $d_i$

$= \text{count}($

.... word ....
word ...
...word
......word ....
word....

$)$

$idf(\text{word}, \text{corpus } C) =$

$$= \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents including word}}\right)$$

$$tfidf(\text{word}, \text{document } d_i, \text{corpus } C) = tf(\text{word}, \text{document } d_i) * idf(\text{word}, \text{corpus } C) =$$

$$= \text{count}(\textit{word} \text{ in } \textit{document } d_i) \; * \; \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents including word}}\right)$$

# tf-idf

## term frequency – inverse document frequency

tfidf(word, document $d_i$, corpus C) = tf(word, document $d_i$) * idf(word, corpus C) =

$$= \text{count}(\textit{word} \text{ in } \textit{document } d_i) \; * \; \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents inluding word}}\right)$$

tfidf(word, document $d_i$, corpus C) **goes ↑**
**for words that are very specific to document $d_i$ but not common in corpus C**

tfidf(word, document $d_i$, corpus C) **goes ↓**
**for words that are NOT very specific to document $d_i$ but common in corpus C**

# `tf-idf`: Example

document $d_1$

this is a sample

Corpus $C$

document $d_1$

this is a sample

document $d_2$

this is another example

# `tf-idf`: Example

document $d_1$

this is a sample

Corpus $C$

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
| --- | --- |
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
| --- | --- | --- | --- |
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

# `tf-idf`: Example

Corpus C

document $d_1$

this is a sample

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("this", document $d_1$, corpus C) = tf("this", document $d_1$) * idf("this", corpus C) =

$$= \text{count}(\text{“}\mathit{this}\text{” in } \mathit{document}\ d_1) * \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents including “}\mathit{this}\text{”}}\right)$$

# tf-idf: Example

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

document $d_1$

this is a sample

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("this", document $d_1$, corpus C) = tf("this", document $d_1$) * idf("this", corpus C) =

$$= 1 * \log\left(\frac{2}{2}\right) = 1 * 0 = 0$$

# tf-idf: Example

document $d_1$

this is a sample

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("sample", document $d_1$, corpus C) = tf("sample", document $d_1$) * idf("sample", corpus C) =

$$= \text{count}(\text{“sample” in } document\ d_1) \ * \ \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents including “sample”}}\right)$$

# tf-idf: Example

document $d_1$

this is a sample

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("sample", document $d_1$, corpus C) = tf("sample", document $d_1$) * idf("sample", corpus C) =

$$= 1 * \log(\frac{2}{1}) = 1 * 1 = 1$$

# tf-idf: Example

document $d_1$

this is a sample

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("is", document $d_1$, corpus C) = tf("is", document $d_1$) * idf("is", corpus C) =

$$= \text{count}(\text{"is" in } document\ d_1) * \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents including "is"}}\right)$$

# tf-idf: Example

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

document $d_1$

this is a sample

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("is", document $d_1$, corpus C) = tf("is", document $d_1$) * idf("is", corpus C) =

$$= 1 * \log(\frac{2}{2}) = 1 * 0 = 0$$

# tf-idf: Example

document $d_1$

this is a sample

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("a", document $d_1$, corpus C) = tf("a", document $d_1$) * idf("a", corpus C) =

$$= \text{count}(\text{“a” in } \textit{document } d_1) * \log\left(\frac{\text{N: number of all documents in C}}{\text{number of all documents including “a”}}\right)$$

# tf-idf: Example

document $d_1$

this is a sample

Corpus C

document $d_1$

this is a sample

document $d_2$

this is another example

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

| word | count | word | count |
|---|---|---|---|
| this | 2 | another | 1 |
| is | 2 | example | 1 |
| a | 1 | | |
| sample | 1 | | |

tfidf("a", document $d_1$, corpus C) = tf("a", document $d_1$) * idf("a", corpus C) =

$$= 1 * \log(\frac{2}{1}) = 1 * 1 = 1$$

# tf-idf: Example

document $d_1$

this is a sample

| word | count |
|---|---|
| this | 1 |
| is | 1 |
| a | 1 |
| sample | 1 |

tf-idf

| word | tfidf |
|---|---|
| this | 0 |
| is | 0 |
| a | 1 |
| sample | 1 |

# tf-idf: Alternative Measures

## term frequency - inverse document frequency

| scheme | tf weight |
|---|---|
| binary | 0, 1 |
| raw count (used in example) | count(word in document) |
| term frequency | $\frac{count(word\ in\ document)}{\sum_i count(\text{anyWord}_i\ in\ document)}$ |
| log normalization | log(1 - count(word in document)) |

| scheme | idf weight |
|---|---|
| unary | 1 |
| inverse document frequency (used in example) | $\log(\frac{\text{N: number of all documents in Corpus}}{\text{number of all documents including word}})$ $= -\log(\frac{\text{number of all documents including word}}{\text{N: number of all documents in Corpus}})$ |
| inverse document frequency smooth | $1 + \log(\frac{\text{N: number of all documents in Corpus}}{\text{1+number of all documents includin word}})$ |

# `tf-idf`: Alternative Measures

## term frequency - inverse document frequency

| scheme | `tf` weight |
|---|---|
| binary | 0, 1 |
| raw count | count(word in document) |
| term frequency | $\frac{count(word\ in\ document)}{\sum_i count(\text{anyWord}_i\ in\ document)}$ |
| log normalization (typical) | log(1 - count(word in document)) |

| scheme | `idf` weight |
|---|---|
| unary | 1 |
| inverse document frequency (typical) | $\log(\frac{\text{N: number of all documents in Corpus}}{\text{number of all documents including word}}) = -\log(\frac{\text{number of all documents including word}}{\text{N: number of all documents in Corpus}})$ |
| inverse document frequency smooth | $1 + \log(\frac{\text{N: number of all documents in Corpus}}{\text{1+number of all documents including word}})$ |

# Words as Vectors: Issues

- We saw how to build vectors to represent words:
  - one-hot encoding:
    - binary, count, tf*idf
- Some problems
  - Large dimensionality of word vectors
  - Lack of meaningful relationships between words

# Vector Embeddings: Methods

- `tf-idf`
  - **popular in Information Retrieval**
  - **sparse vectors**
  - **word represented by (a simple function of) the counts of nearby words**

- `Word2vec`
  - **dense vectors**
  - **representation is created by training a classifier to predict whether a word is likely to appear nearby**

# Sparse vs. Dense Vectors

- Sparse vectors have a lot of values set to zero.

  [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2]

- Dense vector: most of the values are non-zero.
  - better use of storage
  - carries more information

  [3, 1, 5, 0, 1, 4, 9, 8, 7, 1, 1, 2, 2, 2, 2]

# Sparse vs. Dense Vectors

- `tf-idf` **vectors are typically:**
  - **long (length 20,000 to 50,000)**
  - **sparse (most elements are zero)**

- **What if we could learn vectors that are**
  - **short (length 50-1000)**
  - **dense (most elements are non-zero)**

# Short / Dense Vectors: Benefits

- Why short/dense vectors?
  - short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
  - dense vectors may **generalize** better than explicit counts
  - dense vectors may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are distinct dimensions
      - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

# Short/Dense Vectors: Methods

- “Neural Language Model”-inspired models
  - Word2vec, GloVe
- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- Alternative to these "static embeddings":
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

# Word2Vec: Idea

## DON'T count - Predict!

# Word2Vec: Idea

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
  - but we'll take the learned classifier weights as the word embeddings
- Use **self-supervision**:
  - A word c that occurs near "*apricot*" in the corpus acts as the gold "correct answer" for supervised learning
  - **No need for human labels**

# Available Tools

- Word2vec (Mikolov et al)

https://code.google.com/archive/p/word2vec/

- GloVe (Pennington, Socher, Manning)

http://nlp.stanford.edu/projects/glove/

# Word2Vec: the Approach

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get **negative examples**
3. Use **logistic regression** to train a classifier to distinguish those two cases
4. Use the **learned classifier weights** as the **embeddings**

# Word2Vec: the Approach

Given the set of **positive** and **negative** training instances, and an **initial set of embedding vectors**

The goal of learning is to adjust those word vectors such that we:

- **maximize** the similarity of the target word, context word pairs ($w$, $c_{pos}$) drawn from the **positive** data
- **minimize** the similarity of the ($w$, $c_{neg}$) pairs drawn from the **negative** data.

# Word2Vec: the Approach

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

For each positive example we'll grab k negative examples, sampling by frequency

# Word2Vec: the Approach

…lemon, a [tablespoon of apricot jam, a] pinch…
c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

**negative examples -**

| t | c | t | c |
|---|---|---|---|
| apricot | aardvark | apricot | seven |
| apricot | my | apricot | forever |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | if |