Lecture 4:
Word Embedding (cont.)

Instructor: Xiang Ren
USC CSCI 444 NLP
2026 Spring

Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Announcements

Announcements + Logistics

- Project team finalized by end of Jan → submit your team by Feb 2 and no more adjustment!
- HW1 is due by Feb 4, 11:59 PM PT
- Project proposal is due by Feb 11, 11:59 PM PT

Project Pitch (cont.)

Words as Vectors

“You shall know a word by the company it keeps.”

- Firth (1957)

Word Meaning via Language Use

- The meaning of a word can be given by its distribution in language usage:
  - One way to define "usage": words are defined by their environments
    - Neighboring words or grammatical environments
- Intuitions: Zellig Harris (1954):
  - “oculist and eye-doctor … occur in almost the same environments”
  - “If A and B have almost identical environments we say that they are synonyms.”

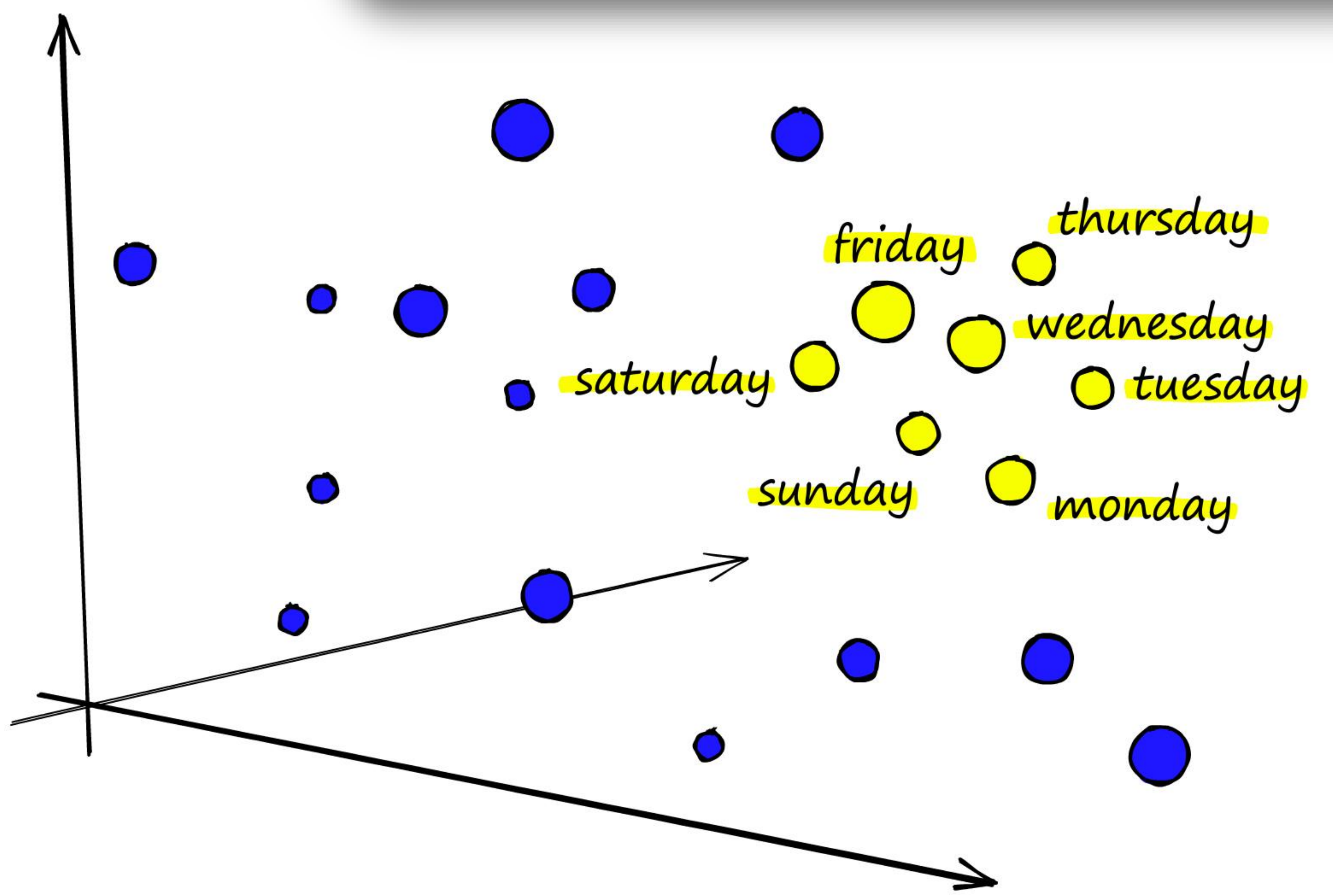
A bottle of tesgüino is on the table
Everybody likes tesgüino
Tesgüino makes you drunk
We make tesgüino out of corn.

Two words are similar if they have similar word contexts

Word Embeddings

Vector Semantics

- Represent a word as a point in a multidimensional semantic space
    - Space itself constructed from distribution of word neighbors
- Called an "embedding" because it's embedded into a space
- Fine-grained model of meaning for similarity



Every modern NLP algorithm uses embeddings as the representation of word meaning

Image Credit: Pinecone

Intuition: Why Vectors?

- Consider generation:
  - With word strings, a feature is a word identity
    - Feature 5: ‘The previous word was “horror”’
    - Requires exact same word to be in training and test
  - With embeddings:
    - 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] … perhaps corresponding to “horrific”
    - We can generalize to similar but unseen words!!!

Cosine Similarity for Word Similarity

Cosine similarity of two vectors

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}.\vec{w}}{|\vec{v}||\vec{w}|}$$

$$= \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Based on the definition of the dot product between two vectors $\vec{a}$ and $\vec{b}$

$$\vec{v}.\vec{w} = |\vec{v}||\vec{w}|\cos\theta$$

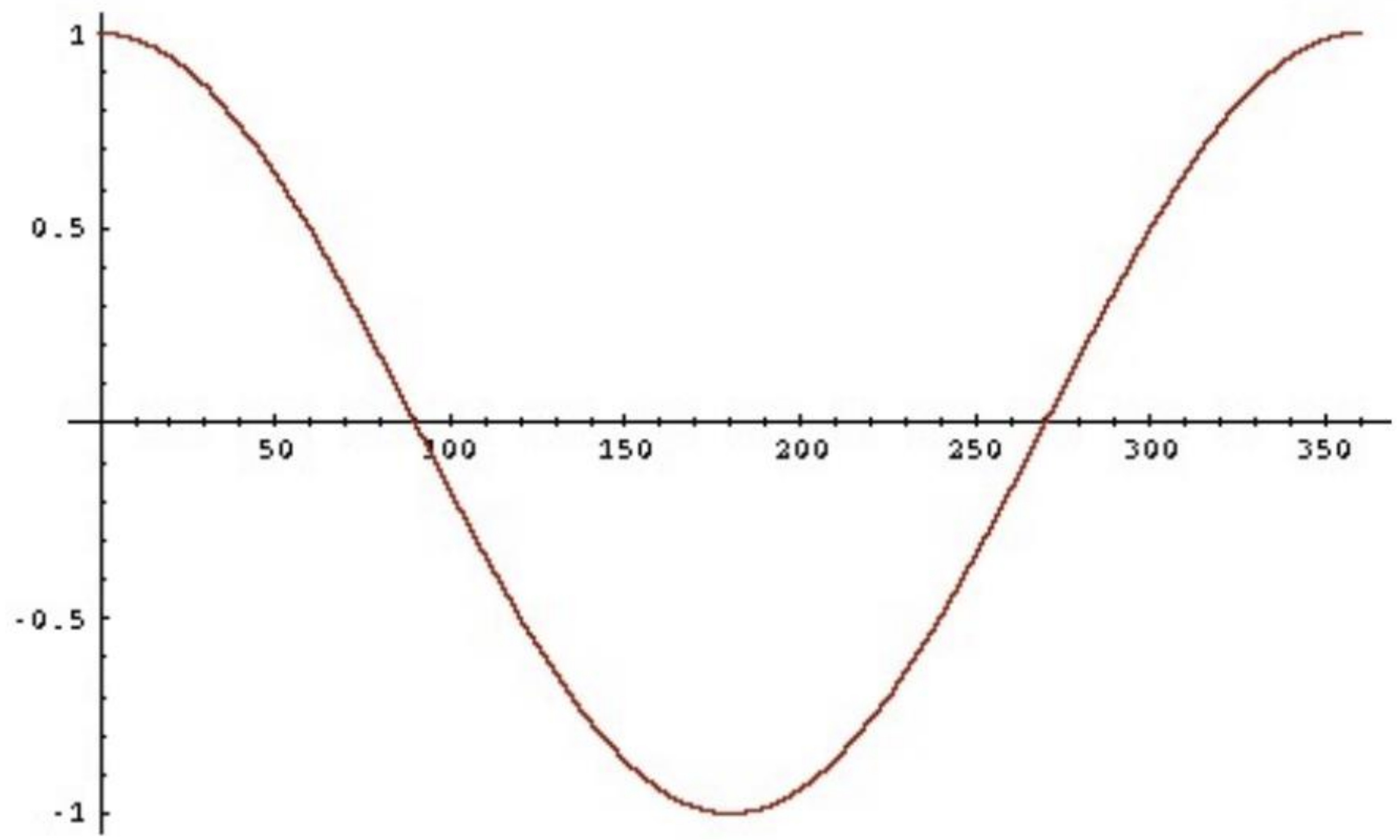
$$\cos\theta = \frac{\vec{v}.\vec{w}}{|\vec{v}||\vec{w}|}$$

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



Greater the cosine, more similar the words

n-grams as One-hot Vectors

Unigram Vectors: Represent each word as a vector of zeros with a single 1 identifying the index of the word

vocabulary
i
hate
love
the
movie
film

movie = <0, 0, 0, 0, 1, 0>
film = <0, 0, 0, 0, 0, 1>

One hot vector

How can we compute a vector representation such that the dot product correlates with word similarity?

Dot product is zero! These vectors are orthogonal

Term-document matrix

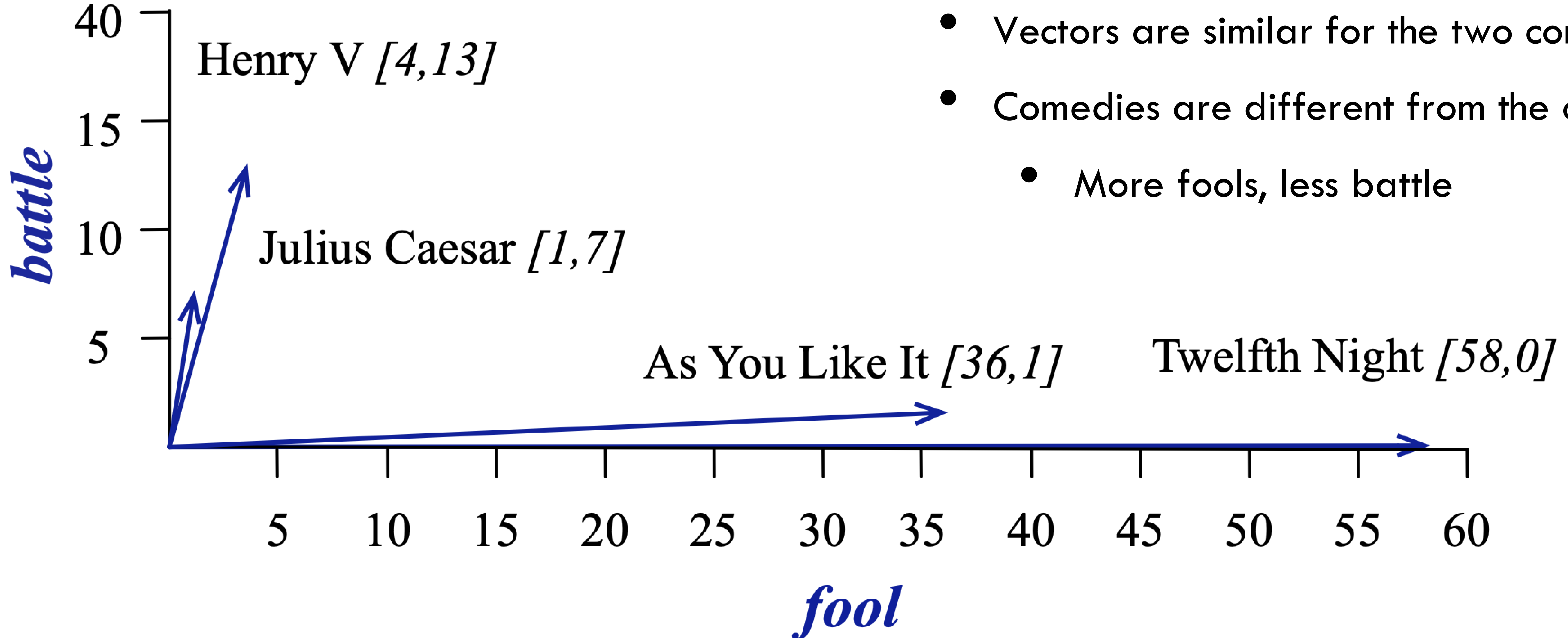
Let us consider a collection of documents and count how frequently a word (term) appears in each. A document could be a play or a Wikipedia article. In general, documents can be anything; we often call each paragraph a document!

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

Visualizing document vectors

	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



- Vectors are similar for the two comedies
- Comedies are different from the other two (tragedies)
  - More fools, less battle

USC Viterbi

Words as vectors in a co-occurrence 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

"Battle" is the kind of word that appears in Julius Caesar and Henry V

"Fool" is the kind of word that appears in As You Like It and Twelfth Night

Word-word co-occurrence matrix

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

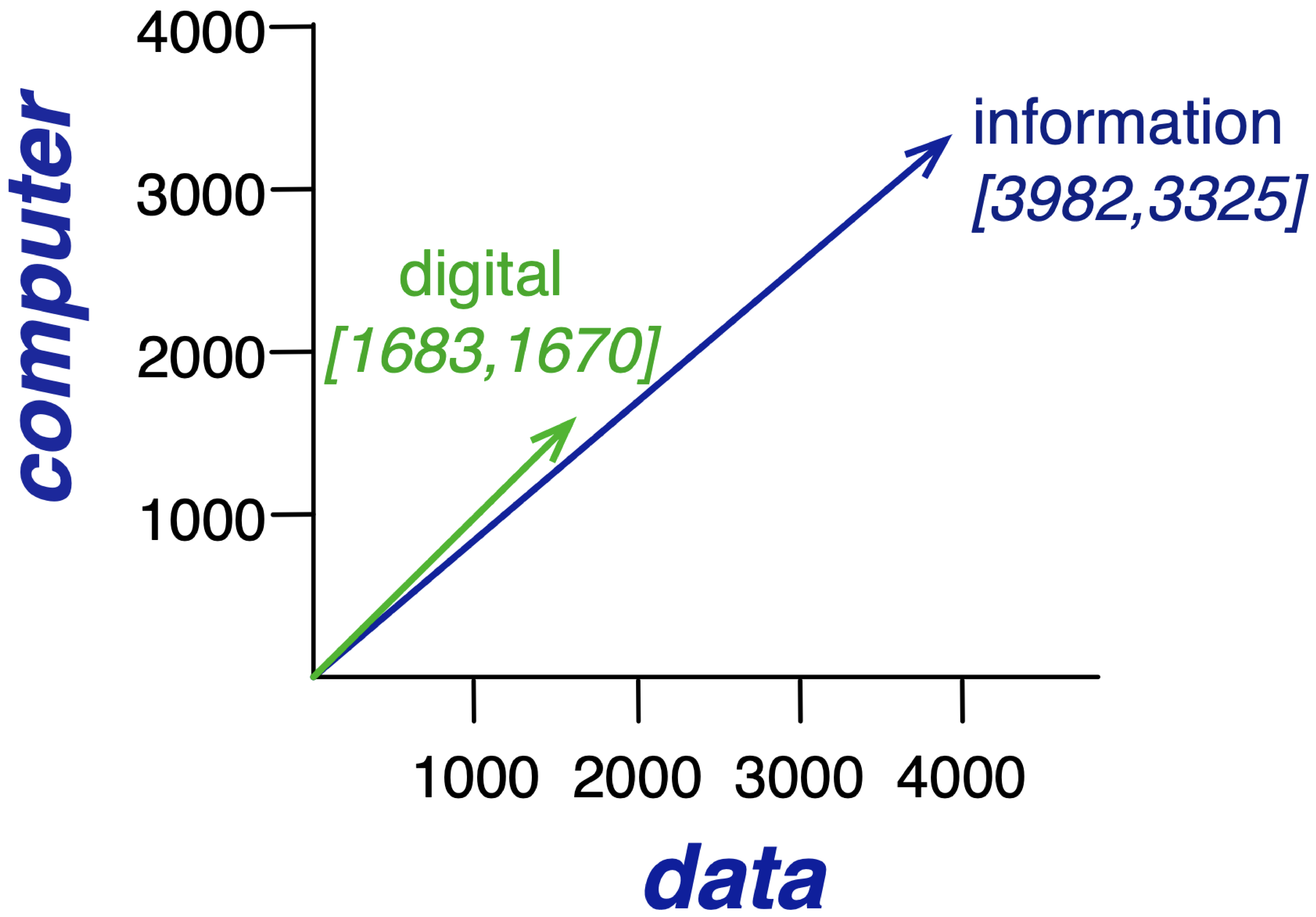
Context Window

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

Words, not documents

	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	...

	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	...



USC Viterbi

Cosine Similarity

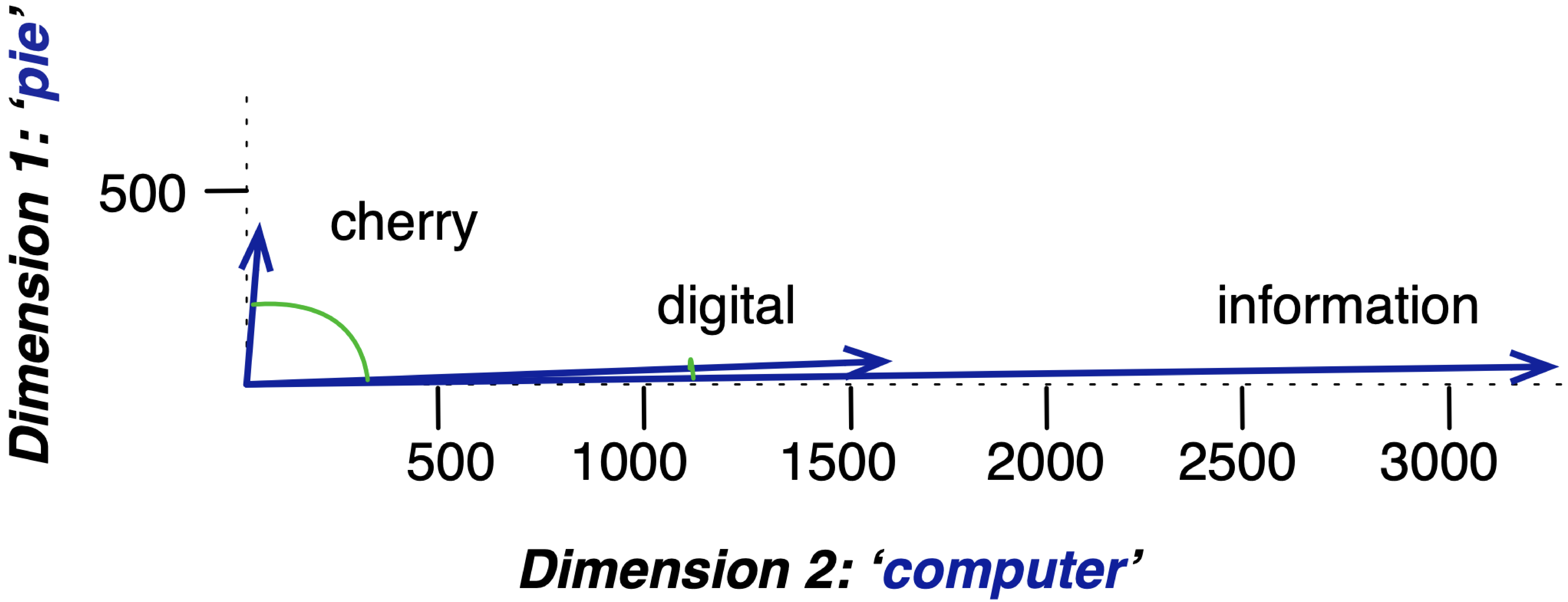
$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}.\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

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

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

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

Visualizing cosines



Raw frequencies though…

- …are a bad representation!
- 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?

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

Need some form of weighting!

Two different kinds of weighting

tf-idf: Term Frequency - Inverse Document Frequency

- Downweighting words like “the” or “if”
- Term-document matrices

PMI: Pointwise Mutual Information

- Considers the probability of words like “good” and “great” co-occurring
- Word co-occurrence matrices

Term Frequency

Term Frequency: frequency counting (usually log transformed)

$$tf_{t,d} = \begin{cases} 1 + \log(count(t,d)), & if\ count(t,d) > 0 \\ 0, & otherwise \end{cases}$$

$count(t,d)$ = # occurrences of word t in document d

Inverse Document Frequency

- Document Frequency: df_t is the number of documents t occurs in.
  - NOT collection frequency: total count across all documents
  - "Romeo" is very distinctive for one Shakespeare play
- Inverse Document Frequency: idf_t

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

N = total number of documents in the collection

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

tf-idf

Final tf-idf weighted value for a word

$$tf_{t,d} \times idf_{t,d}$$

	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

Raw Counts

tf-idf Weighted Counts

	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

Pointwise Mutual Information (PMI)

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

$$PMI(w_1, w_2) = \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

- PMI ranges from $-\infty$ to $+\infty$
  - Negative values are problematic: words are co-occurring less than we expect by chance
  - Only reliable under an enormous corpora
    - Imagine w_1 and w_2 whose probability is each 10^{-6}
    - Hard to be sure $P(w_1, w_2)$ is significantly different than 10^{-12}
  - So we just replace negative PMI values by 0

- Positive PMI

$$PPMI(w_1, w_2) = max\left(0, \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)$$

Church & Hanks 1989

Computing PPMI on a term-context matrix

Context C

Term W

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Frequency $f(w_i, c_j)$ or f_{ij} is the # times w_i occurs in context c_j

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{i,j}} \qquad P_i = \sum_{j=1}^{C} P_{ij} \qquad P_j = \sum_{i=1}^{W} P_{ij}$$

$$PPMI(w_i, c_j) = PPMI_{i,j} = max\left(0, \log \frac{P_{ij}}{P_i P_j}\right)$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{i,j}}$$

$$P_i = \sum_{j=1}^{C} P_{ij}$$

$$P_j = \sum_{i=1}^{W} P_{ij}$$

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

p(w=information,c=data) = 3982/111716 = .3399

p(w=information) = 7703/11716 = .6575

p(c=data) = 5673/11716 = .4842

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

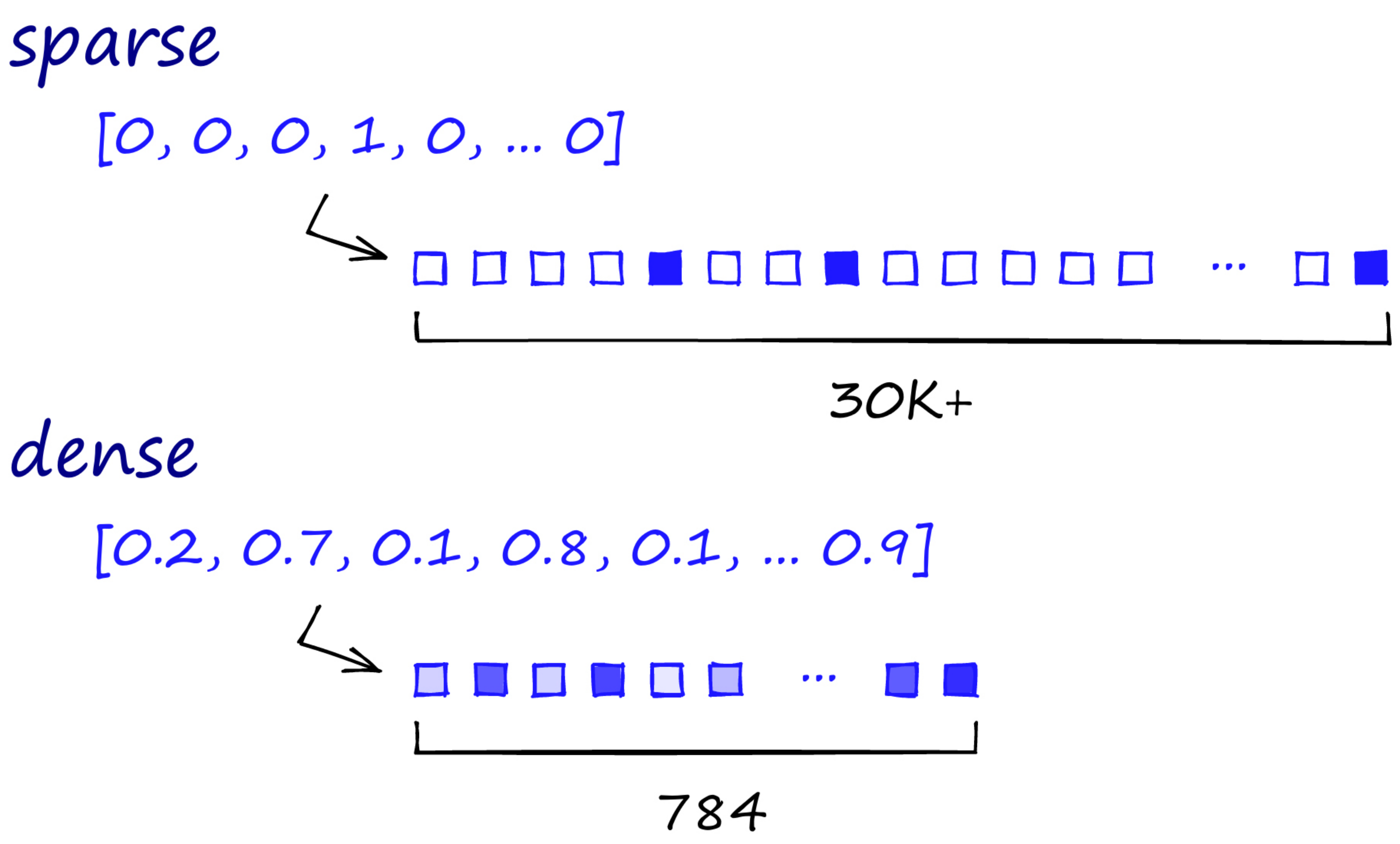
	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

pmi(information,data) = $\log_2$ (.3399 / (.6575*.4842)) = .0944

$$PPMI_{i,j} = max\left(0, \log\frac{P_{ij}}{P_i P_j}\right)$$

The problem...

- tf-idf (or PMI) vectors are
  - long (length |V| = 20,000 to 50,000)
  - sparse (most elements are zero)



- Alternative: learn vectors which are
  - short (length 50-1000)
  - dense (most elements are non-zero)

Image Credit: Pinecone

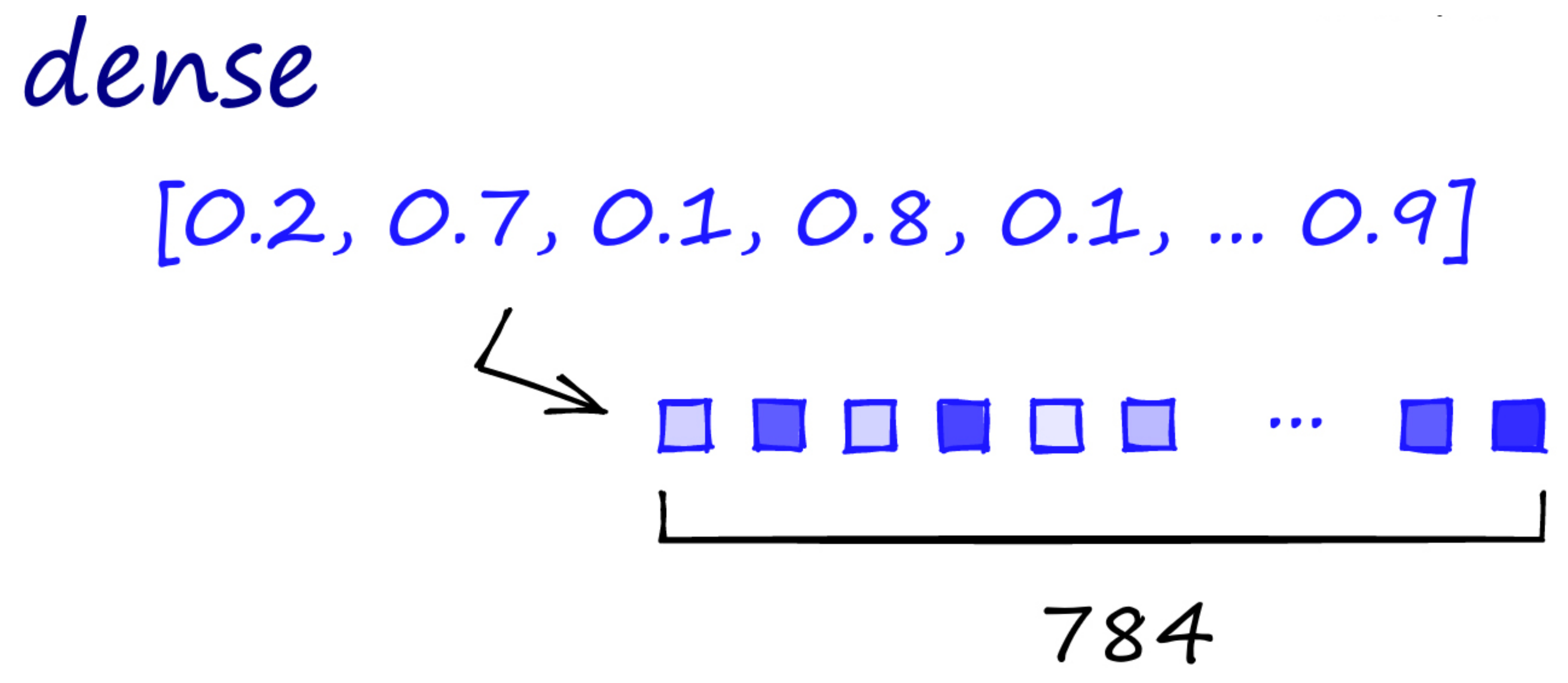
Sparse vs. Dense Vectors

- Why dense vectors?
  - Memory efficiency is not a problem for sparse 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

How to obtain dense vectors?

"Neural Language Model"-inspired models

- Word2vec (skipgram, CBOW), GloVe



Singular Value Decomposition (SVD)

- Special case: Latent Semantic Analysis (LSA)

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

Image Credit: Pinecone

USC Viterbi

word2vec

word2vec

Mikolov et al., ICLR 2013. Efficient estimation of word representations in vector space.

Mikolov et al., NeurIPS 2013. Distributed representations of words and phrases and their compositionality.

- Short, dense vector or embedding
- Static embeddings
  - One embedding per word type
  - Does not change with context change
- Two algorithms for computing:
  - Skip-Gram with Negative Sampling or SGNS
  - CBOW or continuous bag of words
  - But we will study a slightly different version…
- Efficient training
- Easily available to download and plug in

Polysemy?

word2vec : Intuition

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

Instead of counting how often each word W occurs near another, e.g. “cherry”

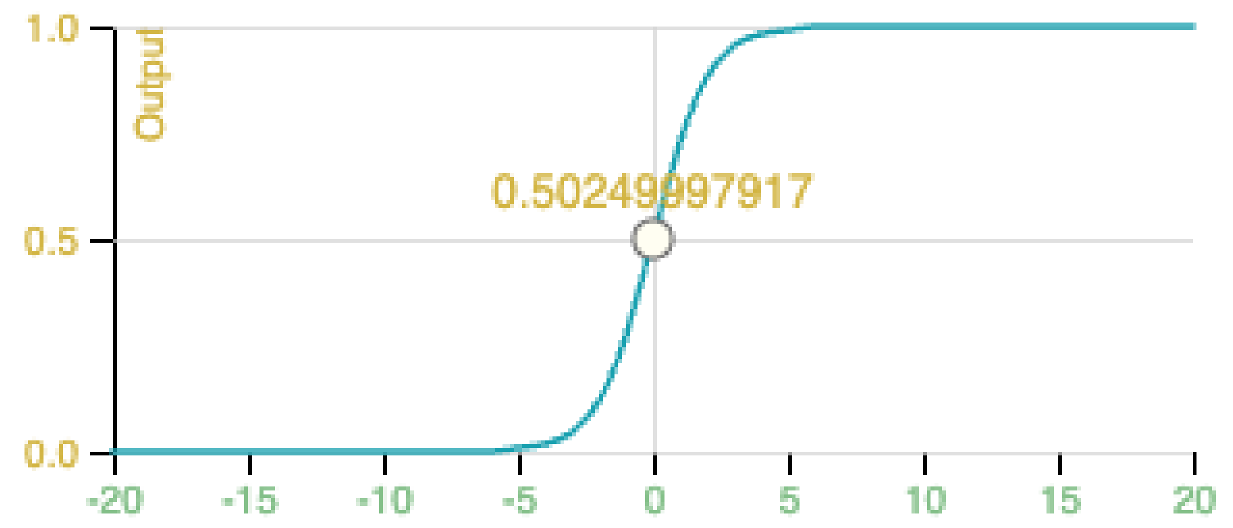
- Train a classifier on a binary prediction task:
  - Is W likely to show up near “cherry”?
- We don’t actually care about this task!!!
  - But we'll take the learned **classifier weights** as the word embeddings

x? y?

Word embedding itself is the learned parameter!

Binary Text Classification

- Goal: Given an input, predict label or class from a discrete set
  - e.g. Predict the sentiment (positive or negative) for a sentence
- Input: x represented by feature vector of size d, given by $\mathbf{x} \in \mathbb{R}^d$
- Output: $y \in \{0,1\}$ for binary classification
- Suffices to learn conditional probabilities
  - Parameterized by $\theta \in \mathbb{R}^d$
- Could estimate by cooccurrence counts, but a single feature
  - Better option: dot product (assigning a weight to every feature)
  - Returns a real value: $z \in \mathbb{R}$
- How to get a probability?
  - Consider the Sigmoid function:
- Argmax for prediction:



Logistic Regression

$$P(y|\mathbf{x};\theta)$$

$$z = \theta \cdot \mathbf{x}$$

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

$$P(y=1|\mathbf{x};\theta) = \sigma(\theta \cdot \mathbf{x})$$

$$P(y=0|\mathbf{x};\theta) = 1 - \sigma(\theta \cdot \mathbf{x}) = \sigma(-\theta \cdot \mathbf{x})$$

$$\hat{y} = \arg\max_{y' \in \{0,1\}} P(y'|\mathbf{x};\theta)$$

word2vec: Self-supervision

One missing piece: where to get the (x, y) pairs from?

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

- A word C that occurs near "cherry" in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels!

What about incorrect labels?

Bengio et al. (2003); Collobert et al. (2011)

word2vec: Goal

Assume a +/- 2 word window, given training sentence:

$P(-|w, c_k)$ …aardvark…

$P(+|w, c_1)$ $P(+|w, c_4)$

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

c_1 c_2 w c_3 c_4

Goal: train a classifier that is given a candidate (word, context) pair:

- (apricot, jam)
- (apricot, aardvark)
- …

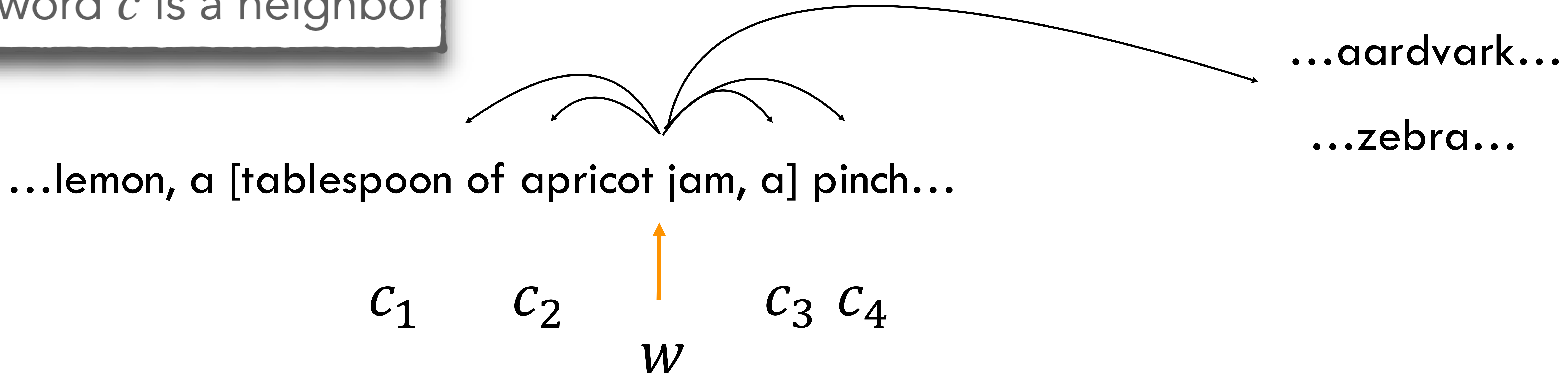
And assigns each pair a probability:

$$P(+|w, c)$$

$$P(-|w, c) = 1 - P(+|w, c)$$

word2vec: Pseudocode

Predict if candidate word c is a neighbor



1. Treat the target word w 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 weights as the embeddings