Lecture 7: Word Embeddings (I)

Instructor: Stephanie Schoch

CSCI 680: Natural Language Processing
Department of Computer Science, College of William & Mary
September 11, 2026

WILLIAM & MARY

Many slides are adapted from Jurafsky & Martin (3rd ed.)

Lecture Outline

- Continuing Text Classification:
  - Multinomial Logistic Regression
  - Evaluating Text Classification
  - Harms of Text Classification
- Word Embeddings
  - Overview
  - Count-Based Embeddings
  - Cosine Similarity
  - Weighting: TF-IDF & PMI

Text Classification: Multinomial Logistic Regression

Multinomial Logistic Regression

- Often we need more than 2 classes
  - Positive / negative / neutral sentiment of a document
  - Parts of speech of a word (noun, verb, adjective, adverb, preposition, etc.)
  - Actionable classes for emergency SMSs
- If >2 classes we use multinomial logistic regression
  - = Softmax regression
  - = Multinomial logit
  - = (defunct names : Maximum entropy modeling or MaxEnt)
- So "logistic regression" will just mean binary (2 output classes)

Multinomial Logistic Regression

- The probability of everything must still sum to 1

$$P(\text{positive} \mid \text{doc}) + P(\text{negative} \mid \text{doc}) + P(\text{neutral} \mid \text{doc}) = 1$$

$$P(+ \mid \text{doc}) + P(- \mid \text{doc}) + P(\sim \mid \text{doc}) = 1$$

- Need a generalization of the sigmoid!
- Introducing the **softmax function**, which
  - Takes a vector of arbitrary values $\mathbf{z} = [z_1, z_2, ..., z_K]$
  - Each corresponds to weighted sum of features for the Kth class
- Outputs a probability distribution
  - each value in the range $[0,1]$
  - all the values summing to 1

Multinomial Logistic Regression

Turns a vector $\mathbf{z} = [z_1, z_2, ..., z_K]$ of K arbitrary values into probabilities

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \qquad 1 \leq i \leq K$$

The denominator $\sum_{i=1}^{K} \exp(z_i)$ is used to normalize all the values into probabilities

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, \ldots, \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

Multinomial Logistic Regression: Softmax Example

Turns a vector $\mathbf{z} = [z_1, z_2, ..., z_K]$ of K arbitrary values into probabilities

$$\mathbf{z} = [0.6,\ 1.1,\ -1.5,\ 1.2,\ 3.2,\ -1.1]$$

$$\text{softmax}(\mathbf{z}) = [0.055,\ 0.090,\ 0.0067,\ 0.10,\ 0.74,\ 0.010]$$

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)},\ \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)},\ \ldots,\ \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

Softmax in Multinomial Logistic Regression

$$p(y=c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} \exp(w_j \cdot x + b_j)}$$

Input is still the dot product between weight vector **w** and input vector **x**

But now we'll need separate weight vectors for each of the K classes.

Features in Multinomial Logistic Regression

Binary: positive weight → y=1 , neg weight → y=0

$$x_5 = \begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \qquad w_5 = 3.0$$

Multinominal: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3

Learning in Multinomial Logistic Regression

Loss:

$$
\begin{aligned}
L_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) &= -\sum_{k=1}^{K} y_k \log \hat{y}_k \\
&= -\log \hat{y}_c, \quad \text{(where } c \text{ is the correct class)} \\
&= -\log \hat{p}(y_c = 1 | \mathbf{x}) \quad \text{(where } c \text{ is the correct class)} \\
&= -\log \frac{\exp(\mathbf{w_c} \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{K} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)} \quad (c \text{ is the correct class})
\end{aligned}
$$

Weight Updates:

$$
\begin{aligned}
\frac{\partial L_{\text{CE}}}{\partial w_{k,i}} &= -(y_k - \hat{y}_k) x_i \\
&= -(y_k - p(y_k = 1 | x)) x_i \\
&= -\left(y_k - \frac{\exp(\mathbf{w_k} \cdot \mathbf{x} + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)} \right) x_i
\end{aligned}
$$

Multinomial Logistic Regression: Example

$$\mathbf{y} = [0, 1, 0]$$

$$\hat{\mathbf{y}} = [0.7, 0.2, 0.1]$$

$$\mathbf{x} = [1, 2]$$

$$\eta = 0.1$$

$$\frac{\partial L_{\text{CE}}}{\partial w_{k,i}} = -(y_k - \hat{y}_k)x_i \quad \mathbf{w}_k^{\text{new}} = \mathbf{w}_k^{\text{old}} - \eta \nabla_{\mathbf{w}_k} L$$

$$L_{CE} = -\log(\hat{y_2}) = -\log(0.2) \approx 1.609$$

Text Classification: Evaluation & Harms

First Step in Evaluation: The Confusion Matrix

gold standard labels

system output labels		gold positive	gold negative
	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

Accuracy on the Confusion Matrix

		gold standard labels	
		gold positive	gold negative
system output labels	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

$$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$$

Why don’t we use accuracy?

Accuracy doesn't work well with uncommon or imbalanced classes

Suppose we look at 1,000,000 social media posts to find Delicious Pie lovers (or haters)

- 100 of them talk about Delicious Pie
- 999,900 are posts about something unrelated

Imagine the following simple classifier

Every post is "not about pie"

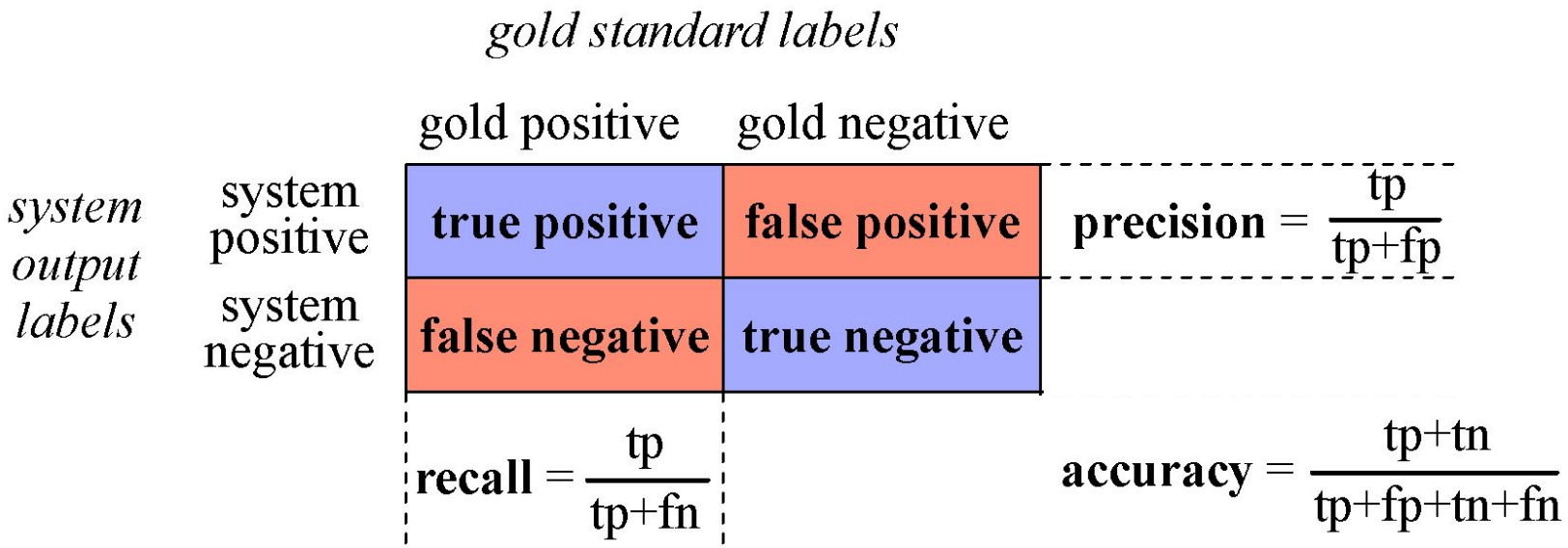
		gold standard labels	
		gold positive	gold negative
system output labels	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

$$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$$

$$= \frac{0 + 999,900}{0 + 0 + 999,900 + 100}$$

Accurate (= 99.99%), but useless for our task

Instead: Use Precision & Recall



Precision: % of selected items that are correct

Recall: % of correct items that are selected

Precision & Recall

Classifier from before: Just say no: every tweet is "not about pie"

- 100 tweets talk about pie, 999,900 tweets don't
- Accuracy = 999,900/1,000,000 = 99.99%

But the Recall and Precision for this classifier are terrible!

		gold standard labels	
		gold positive	gold negative
system output labels	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

$$\textbf{precision} = \frac{tp}{tp+fp} = \frac{0}{0+0} = 0.0\%$$

$$\textbf{recall} = \frac{tp}{tp+fn} = \frac{0}{0+100} = 0.0\%$$

A Combined Measure: F1

$$\text{precision} = \frac{tp}{tp+fp}$$

$$\text{recall} = \frac{tp}{tp+fn}$$

$$F_1 = \frac{2PR}{P+R}$$

Accuracy: % of correct predictions → performs poorly on imbalanced data (game the metric: just predict the majority class)

Precision: % of selected items that are correct → focuses only on false positives (game the metric: just predict one correct positive prediction)

Recall: % of correct items that are selected → focuses only on false negatives (game the metric: just predict "yes" for every item)

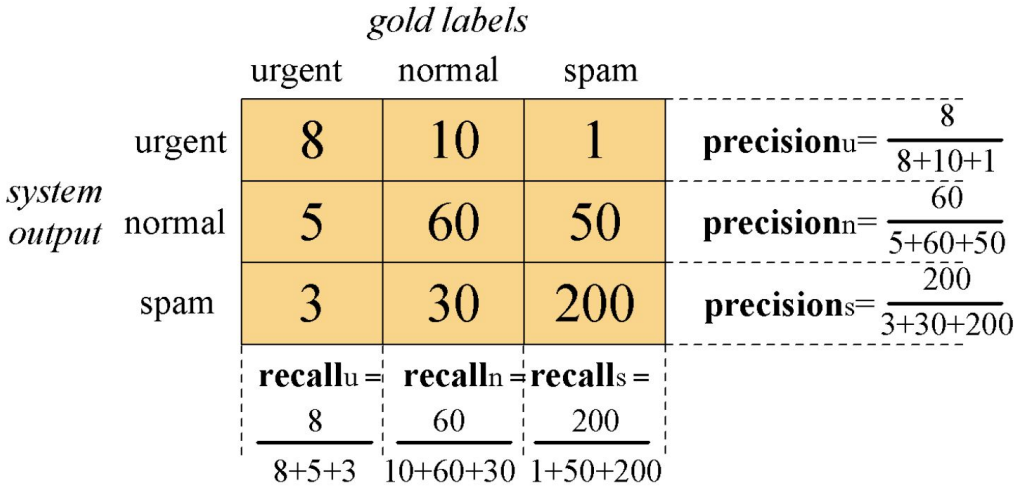
F1: harmonic mean of precision and recall → drops sharply if either P or R is low

Suppose we have more than 2 classes?

Lots of text classification tasks have more than two classes.

- Sentiment analysis (positive, negative, neutral) , named entities (person, location, organization)

We can define precision and recall for multiple classes like this 3-way email task:



How to combine P/R values for different classes:

Microaveraging vs Macroaveraging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Macroaveraging: compute performance for each class, then average over classes

- Better reflects statistics of smaller classes

Microaveraging: combine decisions for all classes into single matrix, then compute P & R

- Dominated by frequent class

Harms of Text Classification: Examples

Template	#sent.
Sentences with emotion words:	
1. <Person> feels <emotional state word>.	1,200
2. The situation makes <person> feel <emotional state word>.	1,200
3. I made <person> feel <emotional state word>.	1,200
4. <Person> made me feel <emotional state word>.	1,200
5. <Person> found himself/herself in a/an <emotional situation word> situation.	1,200
6. <Person> told us all about the recent <emotional situation word> events.	1,200
7. The conversation with <person> was <emotional situation word>.	1,200
Sentences with no emotion words:	
8. I saw <person> in the market.	60
9. I talked to <person> yesterday.	60
10. <Person> goes to the school in our neighborhood.	60
11. <Person> has two children.	60
Total	**8,640**

Representational Harm

Kititchenko and Mohammad, 2018

Toxicity classifiers incorrectly flag non-toxic sentences that simply mention minority identities:

- women (Park et al., 2018),
- disabled people (Hutchinson et al., 2020)
- gay people (Dixon et al., 2018; Oliva et al., 2021)

Toxicity Detection & Censorship

Performance disparities across **languages** & **language varieties**

- E.g. language detection “Is this English?” → worse for writers who are African American (Blodgett and O'Connor 2017) or from India (Jurgens et al., 2017)

Performance Disparities

Harms of Text Classification: Causes & Considerations

Causes:

- Issues in the data; NLP systems amplify biases in training data
- Problems in the labels
- Problems in the algorithms (like what the model is trained to optimize)
-

Prevalence: The same problems occur throughout NLP (including large language models)

Solutions: There are no general mitigations or solutions

- But harm mitigation is an active area of research
- And there are standard benchmarks and tools that we can use for measuring some of the harms

Word Embeddings

Underlying Ideas of Word Embeddings

Underlying Ideas:

1. Define the meaning by linguistic distribution
    - Distribution in language use, i.e. its neighboring words or grammatical environments.

2. Meaning as a point in multidimensional space

Meaning as a Point in Multidimensional Space

Word Connotation: Words seem to vary along 3 affective dimensions: (Osgood et al. (1957))

- **valence:** the pleasantness of the stimulus
- **arousal:** the intensity of emotion provoked by the stimulus
- **dominance:** the degree of control exerted by the stimulus

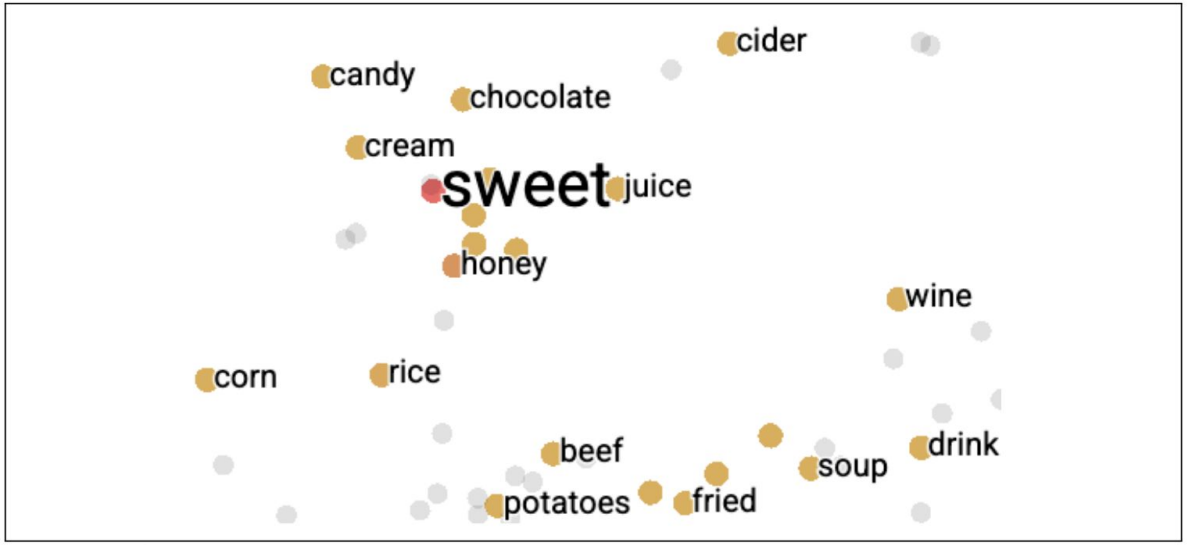
	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

Values from NRC VAD Lexicon (Mohammad 2018)

Words as Vectors: Word Embeddings

- 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 representations of word meaning!



Two Kinds of Word Embeddings

1. **Simple count embeddings**
   - **Sparse** vectors
   - Words are represented by the counts of nearby words
   - tf-idf model of weighting those counts.

2. **Word2vec**
   - **Dense** vectors
   - Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
   - Later we'll discuss extensions called **contextual embeddings**

Co-Occurrence Matrices

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

Can also treat words as vectors across documents.

	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 Co-Occurrence Matrix: 4 words, 3 contexts

Context Window (±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

	a	computer	pie
cherry	1	0	1
strawberry	0	0	2
digital	0	1	0
information	1	1	0

- This 4x3 matrix is a subset of full |V| x |V| matrix
- Each word is represented by a row vector with dimensionality [1 x |V|]
- With co-occurrence counts with each other word

Word Co-Occurrence Matrix: A Larger Matrix

Context Window (±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

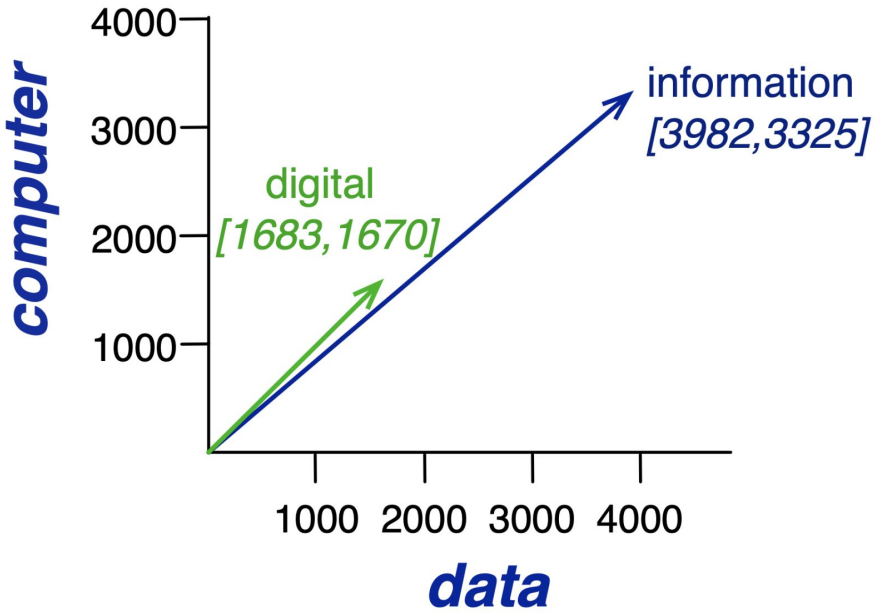
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- Word context matrix is |V| x |V|
- This could be 50,000 x 50,000

  Most of these numbers are zero: **sparse vectors**

Word Co-Occurrence Matrix: Visualization

	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	...



Computing Word Similarity: Cosine Similarity

- Dot product tends to be high when two vectors have large values in same dimension
- But: favors long vectors
  - Frequent words (e.g. functional words) have long vectors

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

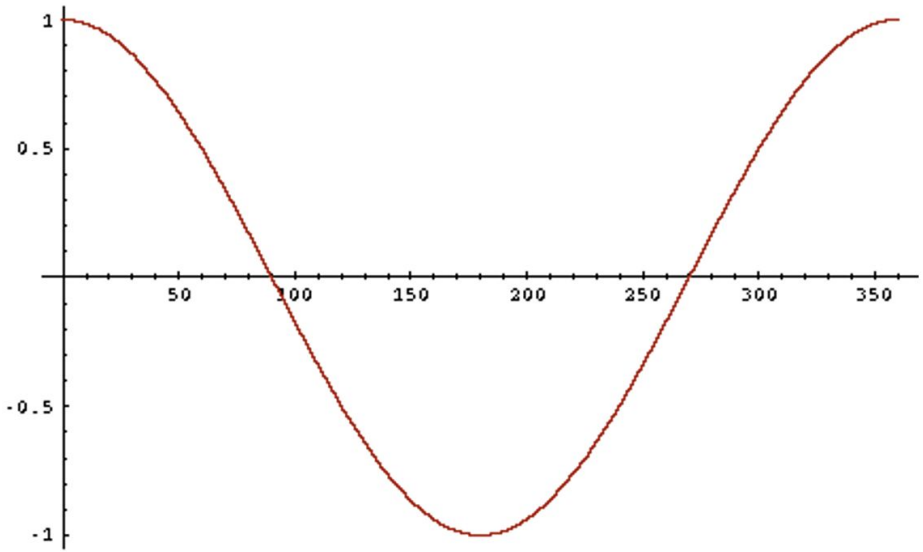
$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$
$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

Based on the definition of dot product between two vectors.

Normalize by length

Computing Word Similarity: Cosine Similarity

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



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

Cosine Example

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

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

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

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

Raw Frequency

- 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.

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

- 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?

Need a form of weighting!

Weighting for Raw Frequencies

- tf-idf: Term Frequency - Inverse Document Frequency

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

  - Downweighting words like “the” or “if”

- PMI: Pointwise Mutual Information

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

  - Consider if words like "good" appear more often with "great" than we would expect by chance

PMI: Example

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

	bark	meow	pet
dog	8	0	2
cat	0	6	4

Term Frequency (tf) in TF-IDF

We could imagine using the raw count:

$$\text{tf}_{t,d} = \text{count}(t,d)$$

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

But instead of using raw count, we usually squash it a bit

- Idea: word appearing 100 times in a document doesn't make that work 100 times more likely to be relevant to the meaning of the document

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

Inverse Document Frequency

- Document frequency df_t is the number of documents t occurs in
  - Note: this is not collection frequency: total count across all documents)
  - *Romeo* is very distinctive for one Shakespeare play

	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

- Inverse Document Frequency: idf_t

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N = total # of documents in the collection

TF-IDF

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

Raw Counts

	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

tf-idf

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.246	0	0.454	0.520
good	0	0	0	0
fool	0.030	0.033	0.0012	0.0019
wit	0.085	0.081	0.048	0.054

Sparse vs. Dense Vectors

- Count co-occurrence vectors (even if weighted by tf-idf)
  - Long (length |V|= 20,000 to 50,000)
  - High-dimensional & sparse (most elements are zero)
- Alternative: learn vectors which are
  - Short (Number of dimensions *d* ranging from 50-1000)
  - Dense (most elements are non-zero)

Why dense vectors?

- Short vectors may be easier to use as features in ML (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!**

Common Methods for Getting Short Dense Vectors

- **“Neural Language Model”-inspired models**
  - **Word2vec** (**skipgram**, CBOW), 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

TODOs:

- **Project Pitch Slide Due: 9/14 (Next Monday)**
- **Homework 1 Released: Due 9/25**

Next Lecture: Word Embeddings (II)

https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/