Introduction to Natural Language Processing

Distributional Semantics

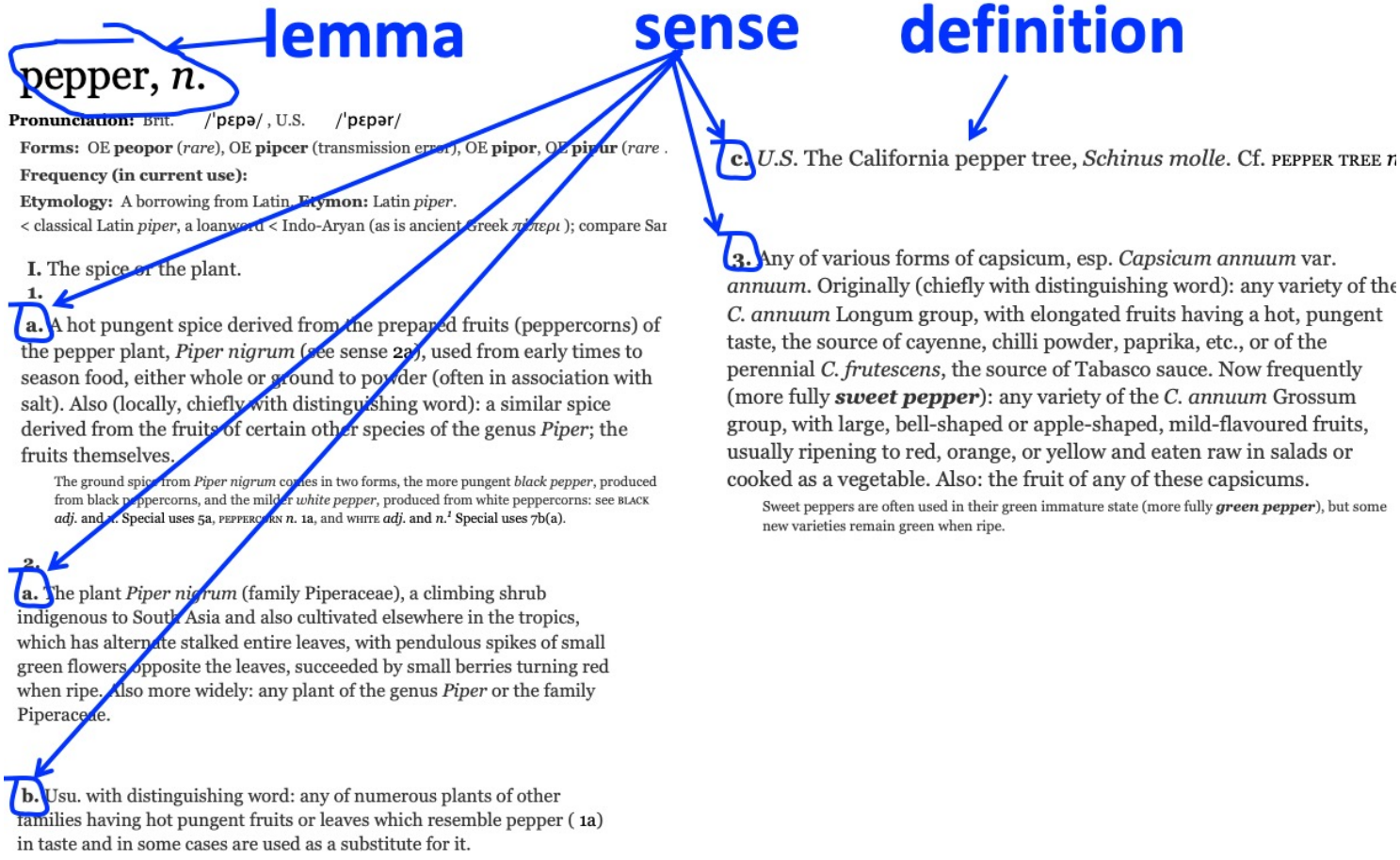
- Jurafsky, D. and Martin, J. H. (2021): Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Third Edition.: Chapter 6: https://web.stanford.edu/~jurafsky/slp3/6.pdf
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008. Chapter 18. https://nlp.stanford.edu/IR-book/pdf/18lsi.pdf

PLAN OF THE LECTURE

- Vector Representations of Words and Documents
- Sparse (Count-based) Vector Representations
- Dense Vector Representations (LSA and word2vec)

What do words mean?

- First thought: look in a dictionary, e.g.: http://www.oed.com



Ask humans how similar two words are

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

Classic ("Aristotelian") theory of concepts

The meaning of a word: a concept defined by **necessary** and **sufficient** conditions

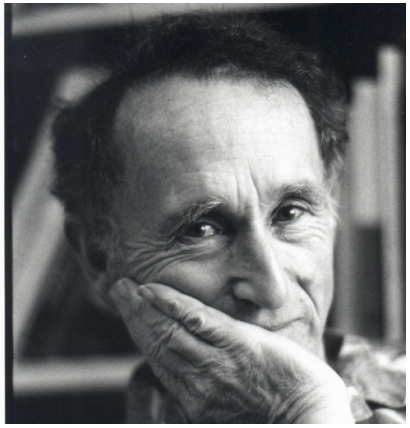
- A **necessary** condition for being an X is a condition C that X must satisfy in order for it to be an X.
  - If not C, then not X
  - "Having four sides" is necessary to be a square.

- A **sufficient** condition for being an X is condition such that if something satisfies condition C, then it must be an X.
  - If and only if C, then X
  - The following necessary conditions, jointly, are sufficient to be a square
    - x has (exactly) four sides
    - each of x's sides is straight
    - x is a closed figure
    - x lies in a plane
    - each of x's sides is equal in length to each of the others
    - each of x's interior angles is equal to the others (right angles)
    - the sides of x are joined at their ends

William Labov's (1975) definition of cup

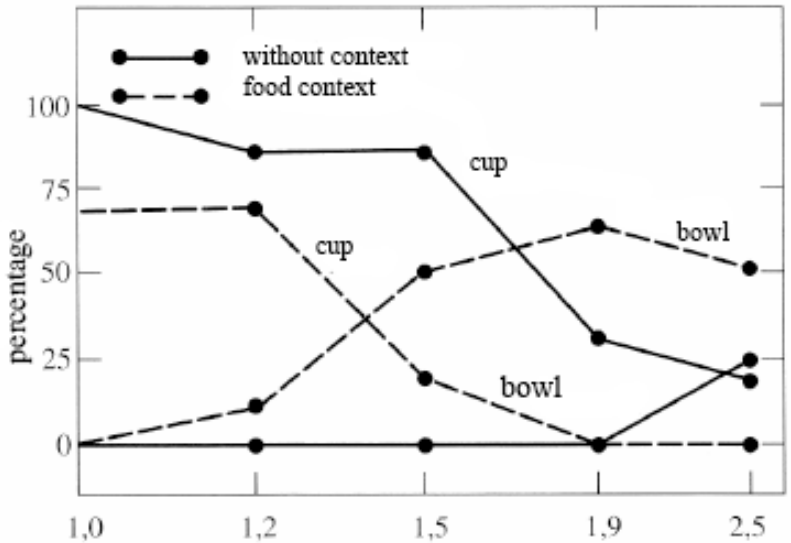
The term *cup* is used to denote round containers with a ratio of depth to width of $1 \pm r$ where $r \leq r_b$, and $r_b = \alpha_1 + \alpha_2 + \ldots \alpha_\upsilon$ and α_1 is a positive quality when the feature i is present and 0 otherwise.

feature
1 = with one handle
2 = made of opaque vitreous material
3 = used for consumption of food
4 = used for the consumption of liquid food
5 = used for consumption of hot liquid food
6 = with a saucer
7 = tapering
8 = circular in cross-section

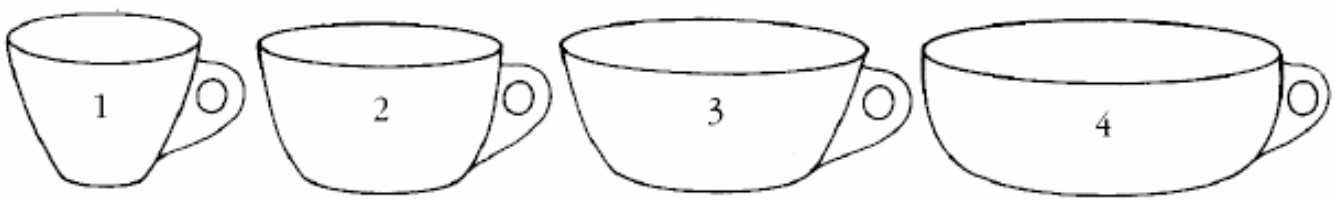


Cup is used variably to denote such containers with ratios width to depth $1 \pm r$ where $r_b \leq r \leq r_1$ with a probability of $r_1 - r/r_t - r_b$. The quantity $1 \pm r_b$ expresses the distance from the modal value of width to height.

The category depends on the context!
If there is food in it, it's a bowl

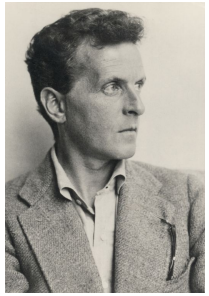


Boundaries between cups and bowls are context sensitive

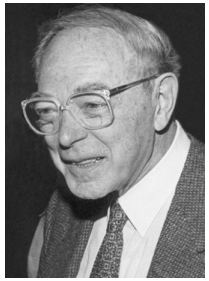


“Data-driven” approach to derivation of word meaning

- Ludwig Wittgenstein (1945): “The meaning of a word is its use in the language”



- Zellig Harris (1954): “If A and B have almost identical environments we say that they are synonyms”



- John Firth (1957): “You shall know the word by the company it keeps.”

What does “ong choi” mean?

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

Ong choi: Ipomoea aquatica "Water Spinach"

Yamaguchi, Wikimedia Commons, public domain

We'll build a model of meaning focusing on similarity

- Each word = a vector
  - Not just "word" or "word45".
- Similar words are "nearby in space"

We define a word as a vector

- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- Fine-grained model of meaning for similarity
  - NLP tasks like sentiment analysis
    - With words, requires same word to be in training and test
    - With embeddings: ok if similar words occurred!!!
  - Question answering, conversational agents, etc

We'll introduce 2 kinds of embeddings

- Sparse (TF-IDF, PPMI)
  - A common baseline model
  - Sparse vectors
  - Words are represented by a simple function of the counts of nearby words

- Dense (word2vec)
  - Dense vectors
  - Representation is created by training a classifier to distinguish nearby and far-away words

PLAN OF THE LECTURE

- Vector Representations of Words and Documents
- Sparse (Count-based) Vector Representations
- Dense Vector Representations (LSA and word2vec)

Representation of Documents: The Vector Space Model (VSM)

- (a.k.a. term-document matrix in Information Retrieval)
- word vectors: characterizing word with the documents they occur in
- document vectors: characterizing documents with their words

Documents

Words	d1	d2	…	di	…	dn
w1						
w2						
…						
wj				n(di,wj)		
…						
wm						

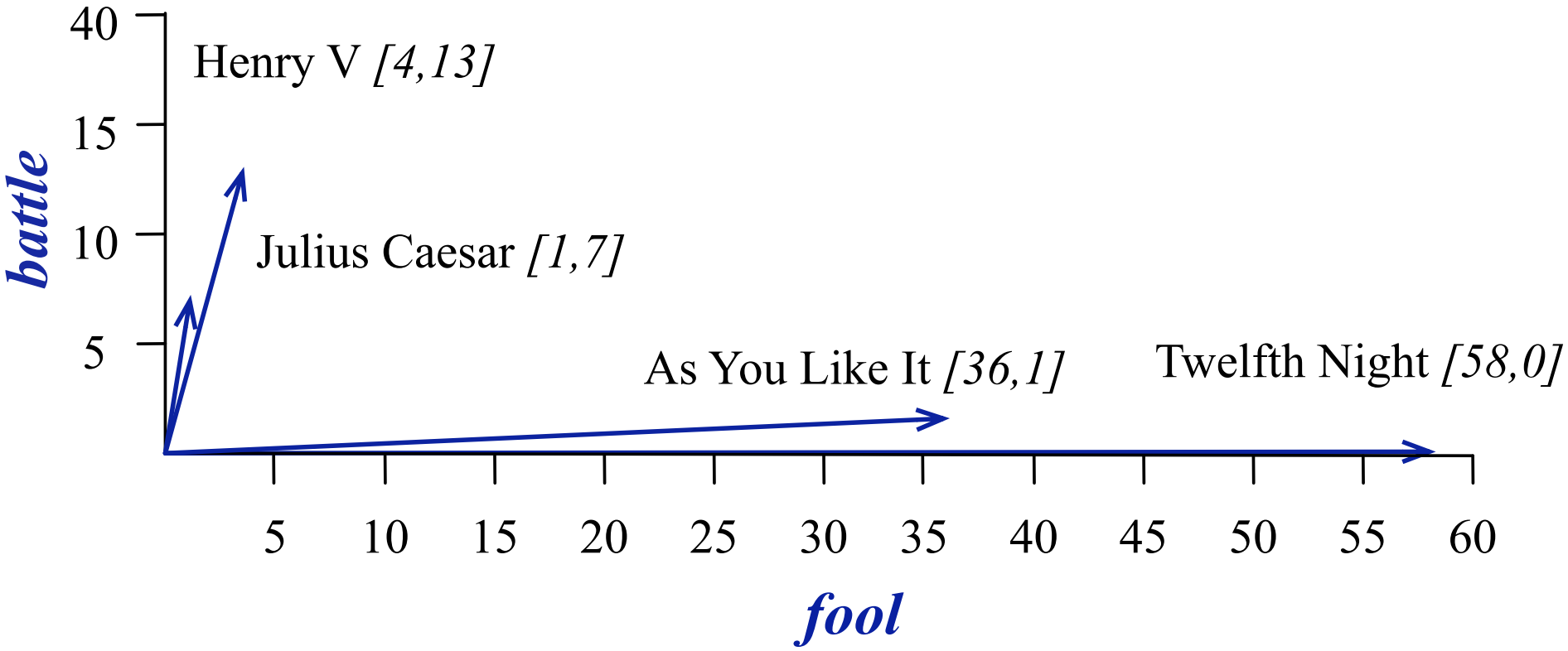
n(di, wj):= (number of words wj in document di) * term weighting

Term-document matrix

- Each document is represented by a column vector of words in the given document
  - May be lemmas, stems, named entities, etc.

	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



Vectors are the basis of information retrieval

	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
- Different than the history
- Comedies have more fools and wit and fewer battles

Words can be vectors too

	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 occurs in Julius Caesar and Henry V"
- **fool** is "the kind of word that occurs in comedies, especially Twelfth Night"
- The other matrix dimension can be composed of words too:

	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	

More common: word-word (term-context) matrix

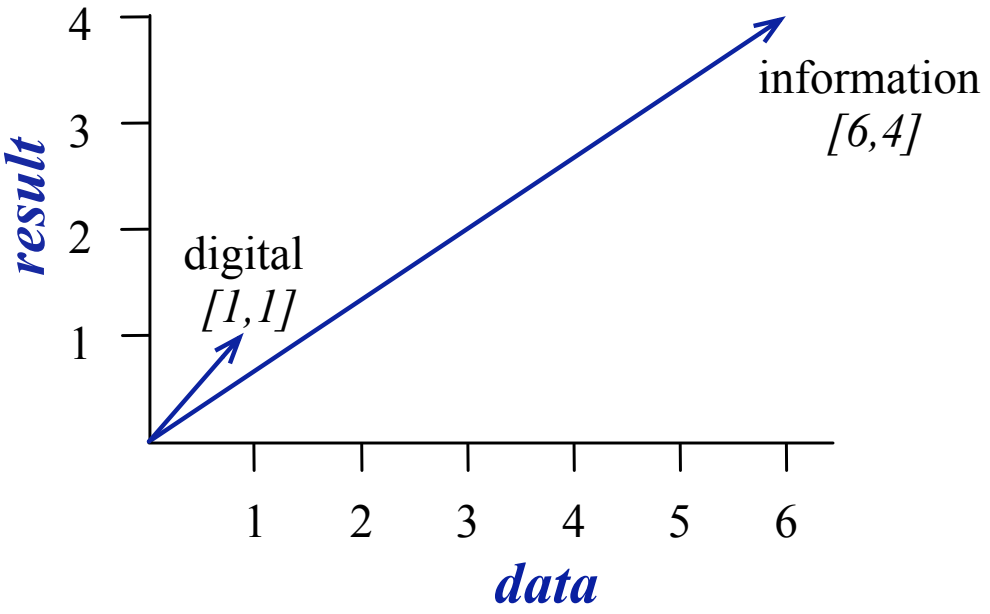
- Two words are similar in meaning if their context vectors are similar

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

More common: word-word (term-context) matrix

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	



Reminders from linear algebra

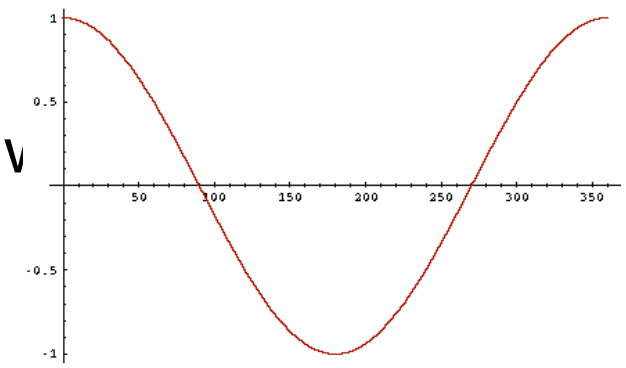
$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

$$\text{vector length} \quad |\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

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

Cosine for computing similarity

- v_i is the count for word v in context i
- w_i is the count for word w in context i.
- cos(v,w) is the cosine similarity of v and w



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

$$\vec{a}\cdot\vec{b} = |\vec{a}||\vec{b}|\cos\theta$$

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

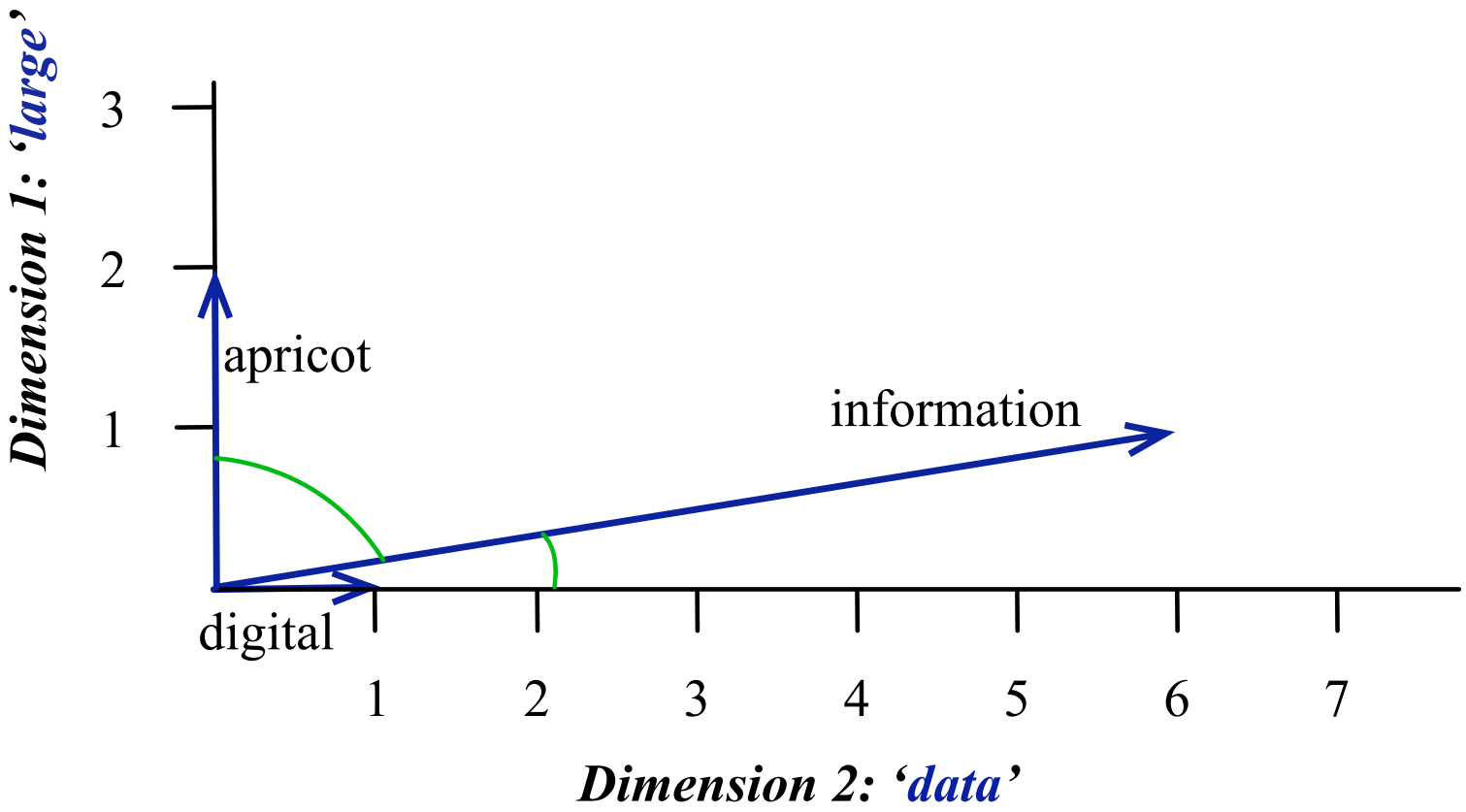
- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal
- frequency is non-negative, so cosine range 0-1

Which pair of words is more similar (based on cosine)?

	large	data	computer
apricot	1	0	0
digital	0	1	2
information	1	6	1

- cosine(apricot,information) = $\frac{1+0+0}{\sqrt{1+0+0}\sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$
- cosine(digital,information) = $\frac{0+6+2}{\sqrt{0+1+4}\sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$
- cosine(apricot,digital) = $\frac{0+0+0}{\sqrt{1+0+0}\sqrt{0+1+4}} = 0$

Visualizing cosines (angles between word vectors)



But raw frequency is a bad representation

- 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
- Need a function that resolves this frequency paradox!

TF-IDF: combine two factors

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

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

- **Idf: inverse document frequency: tf**

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

Total # of docs in collection

# of docs that have word i

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

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

An alternative to tf-idf: pointwise mutual information

- Ask whether a context word is **particularly informative** about the target word.

Pointwise mutual information:

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

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

PMI between two words: (Church & Hanks 1989)

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

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

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring less than we expect by chance
  - Unreliable without enormous corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
    - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
  - Plus it's not clear people are good at "unrelatedness"
- So we just replace negative PMI values by 0
- Positive PMI (PPMI) between word1 and word2:

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

Computing PPMI on a term-context matrix

- Matrix F with W rows (words) and C columns (contexts) f_{ij} is # of times w_i occurs in context c_j

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Weighting PMI: Giving Rare Words Higher Probability

- Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

- This helps because $P_\alpha(c) > P(c)$ for rare c

- Consider two events, P(a) = .99 and P(b)=.01

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97 \quad P_\alpha(b) = \frac{.01^{.75}}{.01^{.75}+.01^{.75}} = .03$$

PLAN OF THE LECTURE

- Vector Representations of Words and Documents
- Sparse (Count-based) Vector Representations
- Dense Vector Representations (LSA and word2vec)

Challenges in the Plain VSM

- TF-IDF and PPMI vectors are long (length |V|= 20,000 to 50,000) and sparse (most elements are zero)
- High dimensionality
  - large storage needed
  - slow processing
- Sparse Matrix
  - Power Law Distribution of frequencies: many words occur only once
  - many zeros
- Lack of Generalization
  - different words are distinct symbols, no matter how similar in meaning
  - e.g. synonyms are represented as different words (same concept)

Alternative: dense vectors

- Vectors which are:
  - short (length 50-1000)
  - dense (most elements are non-zero)

- Why dense vectors?
  - Short vectors may be easier to use as features in machine learning (less weights to tune)
  - Dense vectors may generalize better than storing explicit counts
  - They 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

LSA – Latent Semantic Analysis
What if the matrix was smaller?

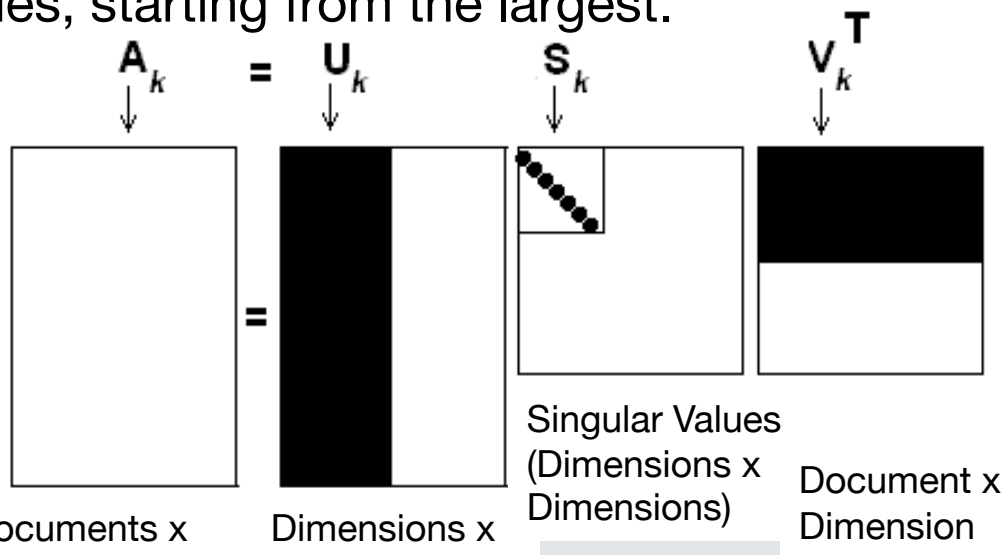
- Map Vector Space Model to a fixed number of low-dimensions
- Mapping reflects semantic association
  - words with similar context should have a similar profile in the reduced dimension space

➔Map terms with similar meaning (in best case synonyms) to similar locations in a low dimensional space

➔ Extract topics and remove noisy topics

Singular Value Decomposition (SVD) The Method used by LSA

- term-document matrix A can be decomposed by Singular Value Decomposition (SVD) into:
  $A = U \bullet S \bullet V^T$ where
  - U: columns are the Eigenvectors of $A \bullet A^T$ (“left” EVs)
  - S: diagonal matrix of singular values: square roots of Eigenvalues
  - V: columns are the “right” Eigenvectors of $A^T \bullet A$ (“right” EVs)
- can order the singular values in decreasing order. There are methods to iteratively compute singular values, starting from the largest.
- approximating the original matrix by the k most important dimensions (topics) in the SVD representation:
  $A_k = U_k \bullet S_k \bullet V_k^T$
  effectively, set all other values of s to 0.

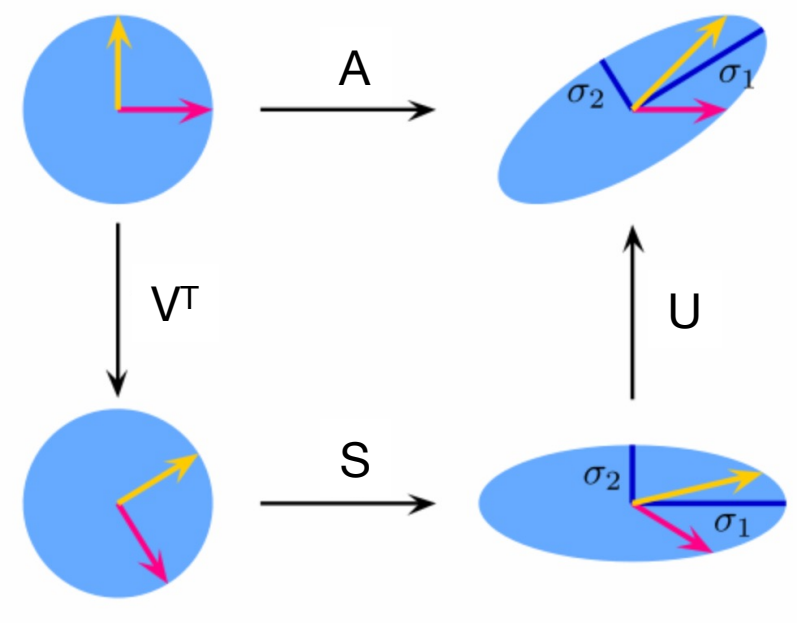


SVD Explained

https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d

$A = U \bullet S \bullet V^T$

- U: rotate the axes into the direction of maximal variance
- S: rescale the axes to achieve equal variance
- V : rotate the input vectors according to the new axes



https://technologiesrunning.blogspot.de/2016/12/10-algoritmos-de-aprendizaje-automatico.html

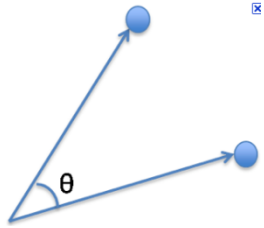
SVD is an orthogonal transform that decorrelates the variables.

Approximation to k dimensions keeps the ones with the largest variance.

Retain only U for word representations

LSA in Practice

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



- Compare documents
  - VSM: e.g. compute similarity between document vectors using the cosine similarity
  - In LSA: Project context vector into reduced vector space using: $q_{LSA}=q^T \bullet U_k \bullet S_k^{-1}$ (with q^T being the word vector space of a new document)
- Build LSA matrices is typically done using a large background corpus with k=100~1000 dims (subject to hyperparameter tuning)

Why this works:

- LSA can generalize over similar words
- LSA serves as a noise reduction technique
- LSA can fold in knowledge about lexical similarities beyond what is in smaller labeled training data

Example CORPUS

words

documents

b	b	b	b	m	m	m	m	m	m	l	l	l	l	l	l
b	b	b	b	b	m	m	m	m	m	m	m	l	l	l	l
b	b	b	b	b	b	b	m	m	m	m	m	l	l	l	l
b	b	b	b	b	b	b	m	m	m	m	m	m	l	l	l
b	b	b	b	b	b	b	m	m	l	l	l	l	l	l	l
b	b	b	b	b	b	b	b	b	m	m	m	l	l	l	l
r	b	b	b	b	m	m	m	m	m	m	m	l	l	l	l
r	s	s	b	b	b	b	b	b	m	m	m	m	l	l	l
r	s	s	s	b	b	b	b	b	b	m	m	m	m	l	l
r	r	s	s	s	b	b	b	b	b	b	m	l	l	l	l
r	r	s	s	s	b	b	b	b	b	b	b	m	m	m	l
r	r	r	s	s	s	s	s	s	b	b	b	b	b	b	m
r	r	r	r	r	r	s	s	s	b	b	b	b	b	b	l
r	r	s	s	s	s	s	s	s	s	b	b	b	b	b	b
r	r	r	r	s	s	s	s	s	s	s	b	b	b	b	b
r	r	r	r	r	s	s	s	s	s	s	s	b	b	b	b

With:

b: bank
m: money
l: loan
r: river
s: stream

LSA Example

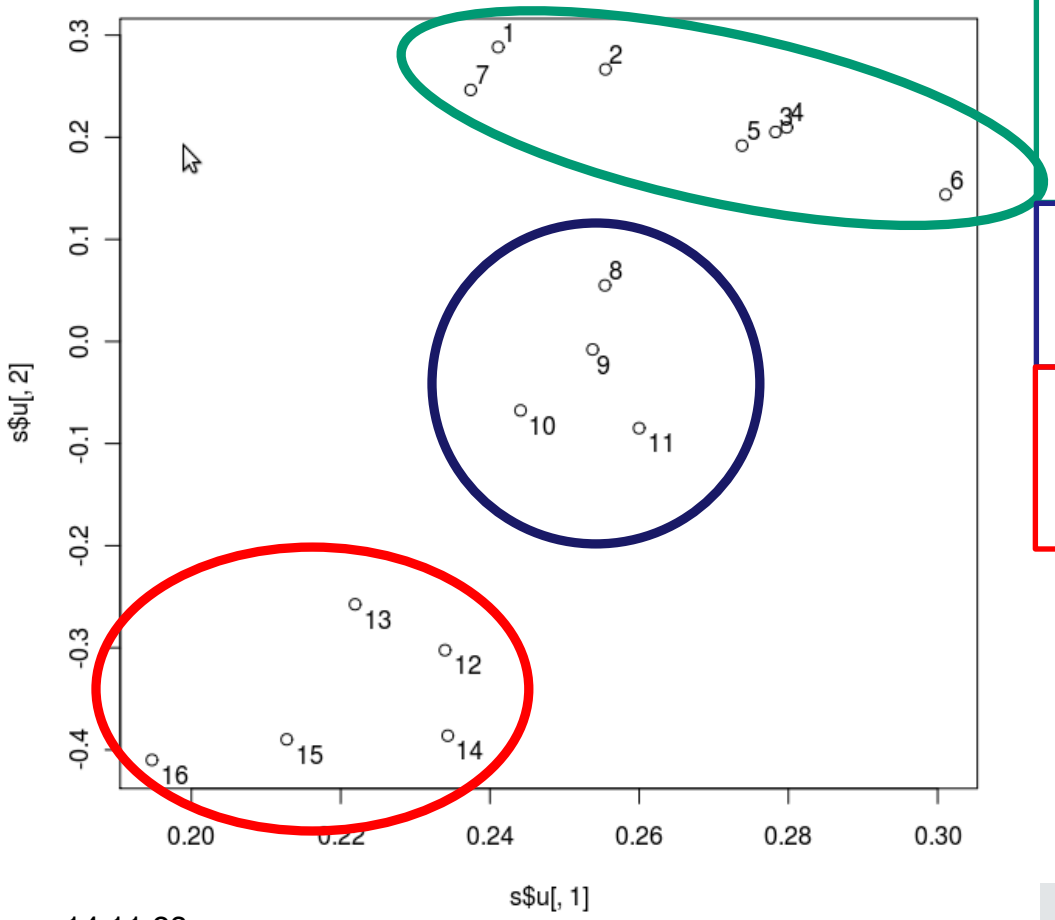
- Perceive Corpus as VSM

ID	b	r	s	m	l
1	4	0	0	6	6
2	5	0	0	7	4
3	7	0	0	5	4
4	7	0	0	6	3
5	7	0	0	2	7
6	9	0	0	3	4
7	4	1	0	7	4
8	6	1	2	4	3
9	6	1	3	4	2
10	6	2	3	1	4
11	7	2	3	3	1
12	6	3	6	1	0
13	6	6	3	0	1
14	6	2	8	0	0
15	5	4	7	0	0
16	4	5	7	0	0

With:
b: bank
m: money
l: loan
r: river
s: stream

LSA Example

Plot of the first two dimensions of V




```
1     b b b b m m m m m m l l l l l l
2     b b b b b m m m m m m m l l l l
3     b   b b b b b b m m m m m l l l l
4     b b b b b b b m m m m m m l l l
5     b b b b b b b m m l l l l l l l
6     b b b b b b b b b m m m l l l l
7     r b b b b m m m m m m m l l l l
8     r s s b b b b b b m m m m l l l
9     r s s s b b b b b b m m m m l l
10    r r s s s b b b b b b m l l l l
11    r r s s s b b b b b b b m m m l
12    r r r s s s s s s b b b b b b m
13    r r r r r r s s s b b b b b b l
14    r r s s s s s s s s b b b b b b
15    r r r r s s s s s s s b b b b b
16    r r r r r s s s s s s s b b b b
```

With:

b: bank
m: money
l: loan
r: river
s: stream

Summary on LSA

- Advantages:
  - Exact mathematical computation – however, expensive for large collections
  - Comparison between documents/words is possible
  - Reduction to n dimensions (“semantic categories”)

- Disadvantages:
  - Unclear how many dimensions are good a priori
  - Each word can have only one representation: polysemy not modeled

PLAN OF THE LECTURE

- Vector Representations of Words and Documents
- Sparse (Count-based) Vector Representations
- Dense Vector Representations (LSA and word2vec)

Some popular pre-trained dense word embeddings

- word2vec (Mikolov et al., 2013)
  - https://code.google.com/archive/p/word2vec/

- fastText (Bojanowski et. al., 2017)
  - http://www.fasttext.cc

- GloVe (Pennington et al., 2014)
  - http://nlp.stanford.edu/projects/glove

word2vec

- Idea: predict rather than count

- 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

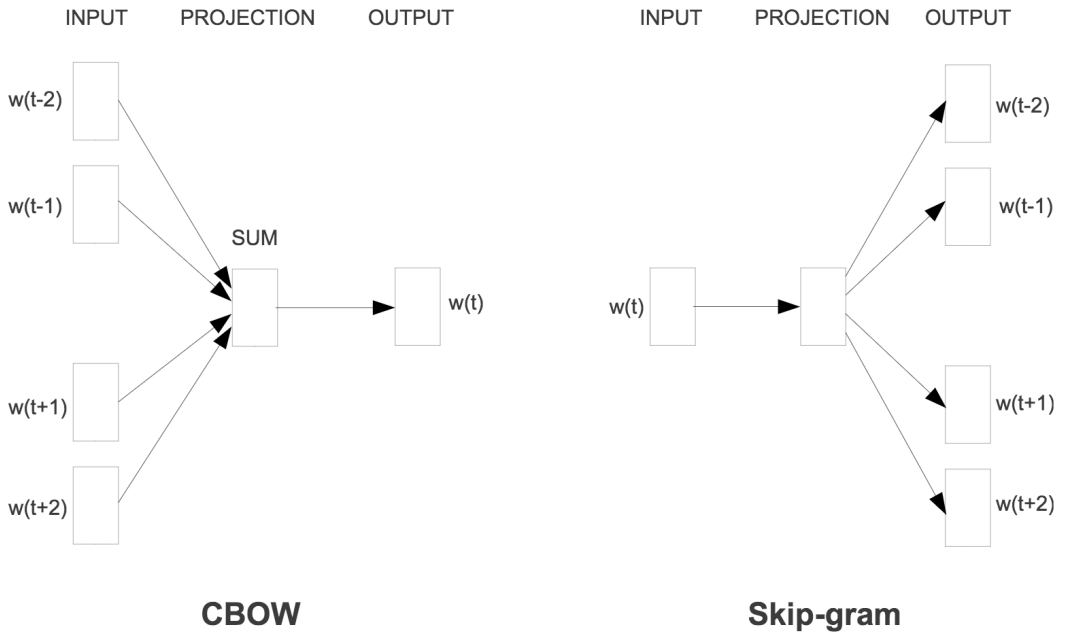
Use running text as implicitly supervised training data!

- A word *s* near *apricot*
  - Acts as gold ‘correct answer’ to the question
  - “Is word *w* likely to show up near *apricot*?”

- No need for hand-labeled supervision

- The idea comes from neural language modeling
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Word2Vec



Two architectures; both work:

- CBOW: predict word, given its close context. Bag-of-words within context
- Skip-gram: predict context, given a word. Takes order into account.

Mikolov, T., Chen, K., Conrado, G., Dean, J. (2013) Efficient Estimation of Word Representations in Vector Space. Proceedings of the Workshop at ICLR, Scottsdale, pp. 1-12.

Word2Vec: Skip-Gram Task

- word2vec provides a variety of options. Let's do
  - “skip-gram with negative sampling” (SGNS)

- Skip-gram algorithm

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

Skip-Gram Training Data

Training sentence: Asssume context words are those in +/- 2 word window.

... lemon, a tablespoon of apricot jam a pinch ...
c1 c2 target c3 c4

Given a tuple (t,c) = target, context

- (apricot , jam)
- (apricot, aadvark)

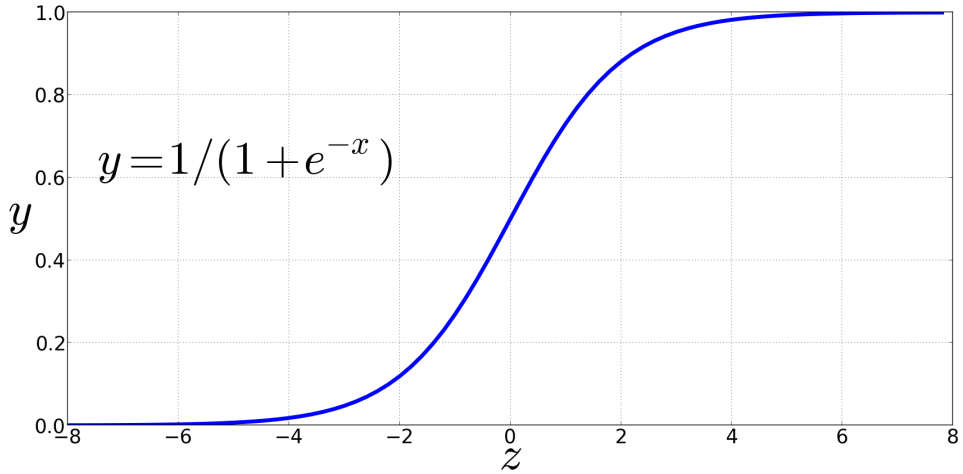
Return probability that c is a real context word:

$P(+|t,c)$

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

How to compute p(+|t,c)?

- Intuition:
  - Words are likely to appear near similar words
  - Model similarity with dot-product!
  - Similarity(t,c) $\propto t \cdot c$
  - Turning dot product into a probability



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Computing probabilities

Turning dot product into a probability:

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$\begin{aligned} P(-|t,c) &= 1 - P(+|t,c) \\ &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned}$$

Assume all context words are independent:

$$P(+|t,c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1+e^{-t\cdot c_i}}$$

$$\log P(+|t,c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t\cdot c_i}}$$

Skip-gram training

Training sentence: Asssume context words are those in +/- 2 word window.

... lemon, a tablespoon of apricot jam a pinch ...
c1 c2 target c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Choosing noise words

- Could pick w according to their unigram frequency P(w)
- More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

- α= ¾ works well because it gives rare noise words slightly higher probability
- To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Objective function

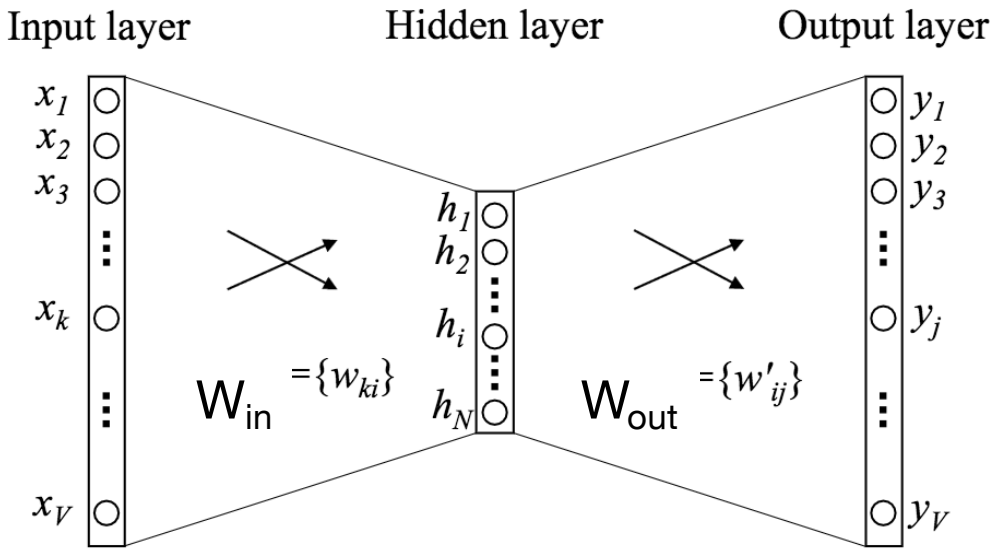
- We want to maximize…

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

- Maximize the + label for the pairs from the positive training data, and the – label for the negative samples.

$$\begin{aligned} L(\theta) &= \log P(+|t,c) + \sum_{i=1} \log P(-|t,n_i) \\ &= \log \sigma(c\cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\ &= \log \frac{1}{1+e^{-c\cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}} \end{aligned}$$

Embeddings: weights to/from projection layer



- W_{in} and W_{out}^T: V x N matrices
- every word is embedded in N dimensions, which is the size of the hidden layer
- Note: embeddings for words and contexts differ

Training word2vec model: summary

- Start with V random 300-dimensional vectors as initial embeddings

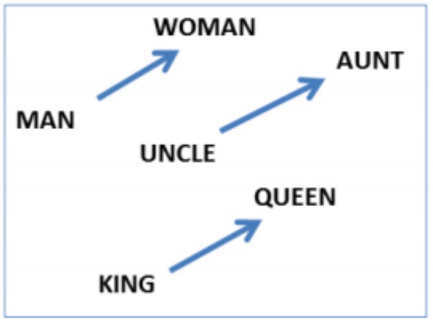
- Use logistic regression, the second most basic classifier used in machine learning after naïve bayes
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

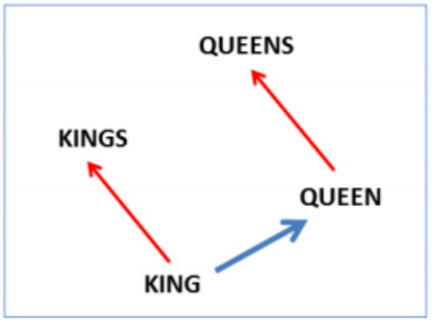
Vector Algebra for Analogy Questions

- Observation: words in the same relation have similar vector differences

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

- Syntactic analogy questions:
  "a is to b as c is to ..."
  (rough is to rougher as tough is to ...)





Mikolov, T., Yih, W., Zweig, G. (2013): Linguistic Regularities in Continuous Space Word Representations. Proc. HLT-NAACL '13, pp. 746-751

Count or Predict?

- Much previous debate why prediction-based models seem to perform better than count-based models

- However, they are similar, as shown by Levy & Goldberg (2014):
  - Skip-gram negative sampling performs implicit matrix factorization with PMI weighting
  - This is equivalent to an approximation of SVD, shifted by a global constant

- Advances on semantic similarity task is largely attributed to more parameter tuning

Levy, O. and Goldberg, Y. (2014): Neural word embedding as implicit matrix factorization. Advances in neural information processing systems, pp. 2177-2185

Summary on Dense Vector Representations

- Core idea: make documents or words comparable by real-valued representations
- Similar items should receive similar representations, which serves as a layer of abstraction
- Representations can be mathematically computed (LSA) or learned (word2vec)
- Dense vectors required as input layer for neural methods, but also suitable as features in any ML-based NLP architecture
- Principles have been known in the NLP community for over 25 years
- Neural embeddings gained a lot of attention is due to more efficient approximate implementations and the use for pre-training in deep neural networks