Lecture 9
Vector Semantics

CS 6320

Outline

- Lexical Semantics
- Vector Semantics
- Words and Vectors
- Word Similarity
- TF-IDF
- word2vec
- Word Embeddings

Lexical Semantics

- **Lexical Semantics -** is the study of word meanings and relations between word meanings.
- **Lexeme -** is an entry in a vocabulary or lexicon.
- **Wordform** – is the word as it appears in the text.

- **Lexicon -** has a finite list of lexemes. It includes individual words, compound nouns, idioms, and others.

- **Lexeme (word) sense -** refers to the meaning of that particular lexeme.
- **Lemma –** lexical forms having the same stem, same part of speech and same word sense ; cat and cats have lemma cat.

Lexical and Semantic Relations

- **Polysemy –** when one word has multiple meanings.

  table has 6 noun senses in WordNet

  1 verb sense

- **Word Sense Disambiguation** – is the NLP task that when given a lexicon with word meanings it finds the correct meaning of a word in a context.

- **Synonymy** – two words are synonymous if the substitution of one for the other does not change the truth value of a sentence in which the substitution is made.

  In WordNet these are called **synsets**.

  { telephone, phone, telephone set}

Lexical Semantics

- **Antonyms** - words with opposite meaning.
  Examples: long/short, big/little, in/out.
- **Reversives** - a group of antonyms that describe change in movement in opposite directions. Examples: rise/fall, up/down.
- **Word similarity** - words that are related via similarity relation, have properties in common. Example: cat and dog.
- **Word relatedness** - words that appear together in sentence. Example: coffee and cup.
- Semantic relations-studied later.

Osgood dimensions

- **Connotations** - affective meanings of words.
  Positive connotations (happy)
  Positive evaluations (great)

- **Osgood's dimensions of word meaning** (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.

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

Idea: Each word is represented as a point in this 3-dimensional space.

Word Distributions

- Word distribution is the set of contexts in which that word occurs;

  Example: the neighboring or grammatical environment.

- Intuition is that words likely to occur in very similar distributions are likely to have the same meaning.

Vector semantics

- Combines two ideas:
  - distributionalist intuition.
  - vector intuition.
- Idea: Represent a word as a point in some multidimensional semantic space.
- Embeddings are vectors that represent word meanings.

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for a sentiment analysis task. Simplified from Li et al. (2015).

Vector semantics

- Two models to represent vector semantics.
  - tf - idf model - word is defined by a function of counts of nearby words.
  - word2vec model - short vectors that carry semantic properties.

Vectors and documents

- Distributional models are based on co-occurrence matric-how often words co-occur.
- Term-document 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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Vectors and documents

- Each row represents a word in vocabulary, and each column represents a document from some collection of 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

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Vectors and documents

- The vector for a document is identifying a point in |V| dimensional space.

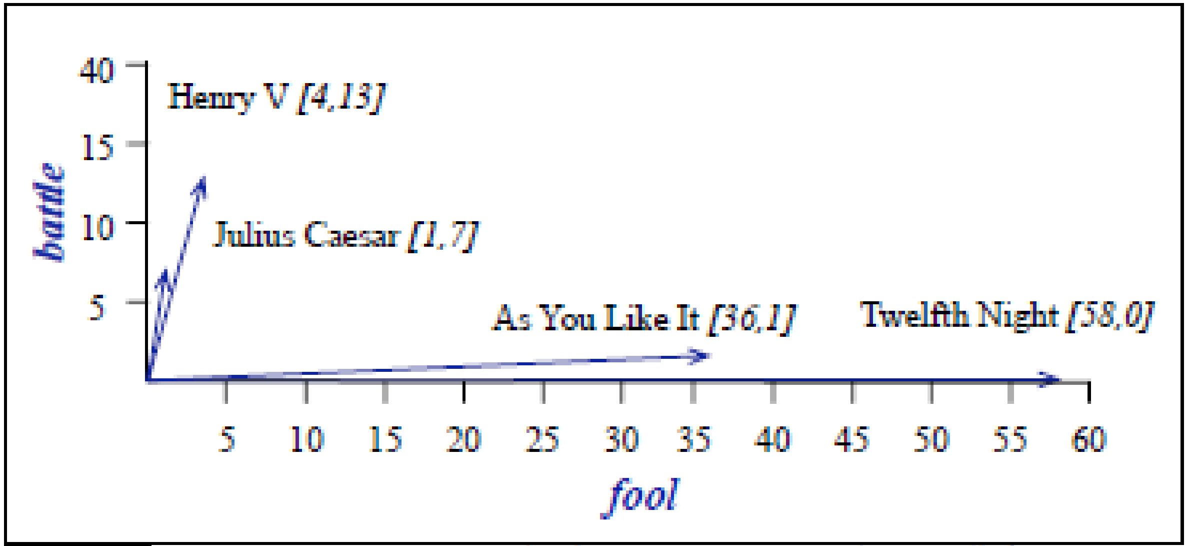

Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Model is useful in Informational Retrieval.

Words as Vectors

- Associate each word with a vector to represent word meanings.
- Use term-term matrix or word-word matrix, or term-context matrix.

|V| x |V| matrix, each entry represents the number of times the target (row) word and the context (column) word co-occur in some context training corpus.

Context:

- Can be a document (nr. of times the two words occur in document.)
- A window of n words to the left and n words to the right.

Words as Vectors

	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	

Figure 6.5 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

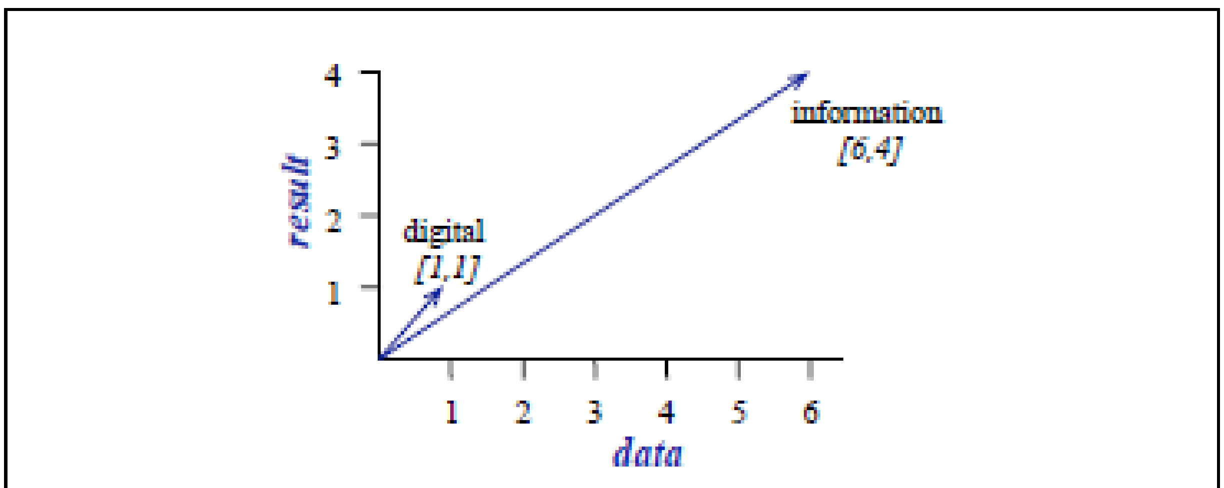

Figure 6.6 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *result*.

Cosine Similarity

- Need to measure similarity between two vectors.

  Dot product is a measure of similarity between vectors

$$\bar{v} \cdot \bar{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_N w_N$$

- Cosine similarity – normalized dot product $|\bar{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$

$$\bar{v} \cdot \bar{w} = |\bar{v}||\bar{w}| cos\theta$$

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

Example

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

$$\cos(apricot, information) = \frac{2+0+0}{\sqrt{4+0+0}\,\sqrt{1+36+1}} = \frac{2}{2\sqrt{38}} = .16$$

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

information is closer to *digital* than it is to *apricot.*

TF-IDF

- tf - term frequency - the frequency of the word in the document

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

- df_t − document frequency of a term t is simply the number of documents it occurs in.
- idf_t - inverse document frequency.

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

N - total number of documents in the collection.

df_t- total number of documents in which t occurs.

The fewer documents df_t the higher idf_t .

TD-iDF

Example from Shakespeare corpus
(37 plays)

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.074
fool	36	0.012
good	37	0
sweet	37	0

The tf- idf weighting $w_{t,d}$ is the value for word t in document d

$$w_{t,d} = tf_{t,d} \times idf_t$$

	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

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

Word2vec

- Idea: Instead of counting how often each word w occurs, say *apricot*, we will train a classifier to predict how likely is word w to show up near *apricot*.

  Then take the learned classifier weights as the word embeddings.

The Classifier- skip-gram with negative sampling

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

$c_1 \quad c_2 \quad t \quad c_3 \quad c_4$

pick (t, c), ie $(apricot, jam)$

Train classifier such that $P(+|t, c)$ is high.

Method:

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 regression weights as the embeddings

The classifier

Idea: Calculate the probabilities based on similarity. A word is likely to occur near a target if its embedding is similar to the target embedding.

$$similarity\,(t\,,\,c) \approx t \cdot c$$

Use sigmoid function $\sigma(c \cdot t)$ to transform dot product into probability

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

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

The classifier

- Need to take into account multiple context words in the window of k words

- Skip-gram model assumes that all contexts are independent.

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

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

Learning skip-gram embeddings

- Pick up a target word, example: *apricot.* Take context words as positive training examples, and also negative examples at random; called noise words.

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

Ratio negative/positive examples k/1.

Learning skip-gram embedding

- Goal:
  - Maximize the similarity of the target word, context (t, c) from positive examples.
  - Maximize the similarity of (t, c) pairs from negative examples.

$$L(\theta) = \sum_{(t,c)\in +} \log P(+|t,c) + \sum_{(t,c)\in -} \log P(-|t,c)$$

  - Focus on one pair with k noise words $n_1, \dots, n_k$

$$L(\theta) = \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)$$

$$= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

$$= \log \frac{1}{1+e^{-c\cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}$$

Learning skip-gram embedding

- We want to maximize the similarity between (t , c); thus maximize $L(\theta)$
- Use stochastic gradient descent to train to this objective. This means to modify the parameters, the embedding for each target word t and each context or noise word c in vocabulary.
- Skip- gram model learns two separate embeddings for each word w, the target embedding t and the context embedding c .

Learning skip-gram embeddings

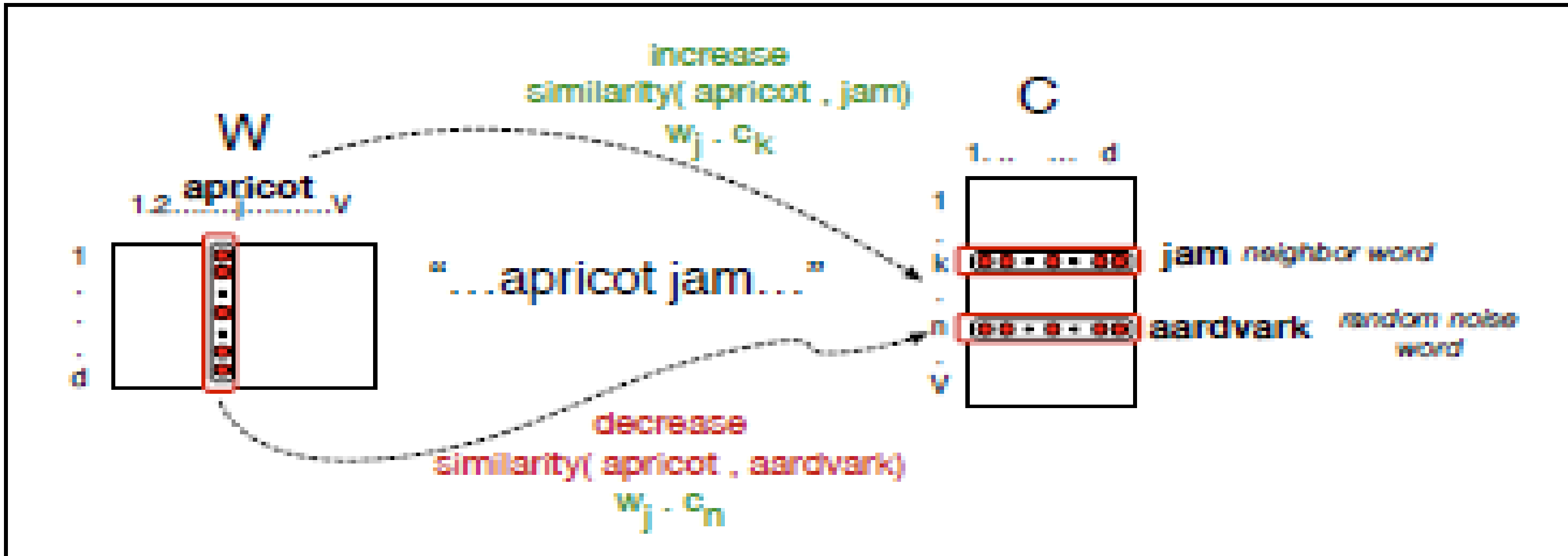

Figure 6.13 The skip-gram model tries to shift embeddings so the target embedding (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (here *aardvark*).

Learning skip-gram embeddings

- Each word w produces two embeddings.
  - Target embedding $t_i - vector\ 1 \text{ x } d$
  - Context embedding $c_i - vector\ 1 \text{ x } d$

- Finally for each word w_i
  - Add $t_i + c_i$ $1 \text{ x } d\ vector$

Or

  - Concatenate

    t_i, c_i $1 \text{ x } 2d$ $vector$