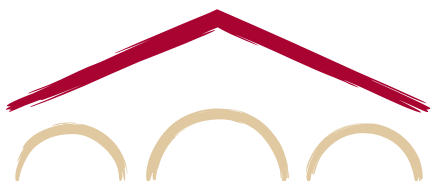
Distributed word representations: High-level goals and guiding hypotheses

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding



Meaning latent in co-occurrence patterns

	:)	:/	:D	:|	;p	abandon	abc	ability	able	⋯
:)	74	1	0	0	0	1	0	2	2	
:/	1	306	0	0	0	0	0	0	17	
:D	0	0	16	0	0	0	6	1	1	
:|	0	0	0	120	0	0	0	1	9	
;p	0	0	0	0	516286	0	0	0	0	⋯
abandon	1	0	0	0	0	370	24	65	235	
abc	0	0	6	0	0	24	7948	77	291	
ability	2	0	1	1	0	65	77	4820	1807	
able	2	17	1	9	0	235	291	1807	14328	
⋮					⋮					

Meaning latent in co-occurrence patterns

Class	Word
neg	awful
neg	terrible
neg	lame
neg	worst
neg	disappointing
pos	nice
pos	amazing
pos	wonderful
pos	good
pos	awesome

A hopeless learning scenario

Meaning latent in co-occurrence patterns

Class	Word
neg	awful
neg	terrible
neg	lame
neg	worst
neg	disappointing
pos	nice
pos	amazing
pos	wonderful
pos	good
pos	awesome

$\Pr(\text{Class} = \text{pos})$	Word
?	w_1
?	w_2
?	w_3
?	w_4

A hopeless learning scenario

Meaning latent in co-occurrence patterns

Class	Word	excellent	terrible
neg	awful	6	113
neg	terrible	8	309
neg	lame	1	69
neg	worst	9	202
neg	disappointing	19	29
pos	nice	118	2
pos	amazing	91	6
pos	wonderful	66	7
pos	good	21	9
pos	awesome	67	2

A promising learning scenario

Meaning latent in co-occurrence patterns

Class	Word	excellent	terrible
neg	awful	6	113
neg	terrible	8	309
neg	lame	1	69
neg	worst	9	202
neg	disappointing	19	29
pos	nice	118	2
pos	amazing	91	6
pos	wonderful	66	7
pos	good	21	9
pos	awesome	67	2

Pr(Class=pos)	Word	excellent	terrible
≈0	w_1	4	82
≈0	w_2	5	84
≈1	w_3	49	3
≈1	w_4	41	5

A promising learning scenario

High-level goals

1. Begin thinking about how vectors can encode the meanings of linguistic units.

2. Foundational concepts for vector-space model (VSMs) a.k.a. embeddings.

3. A foundation for deep learning NLU models.

Guiding hypotheses

Firth (1957)

"You shall know a word by the company it keeps."

Harris (1954)

"distributional statements can cover all of the material of a language without requiring support from other types of information."

Wittgenstein (1953)

"the meaning of a word is its use in the language"

Turney and Pantel (2010)

"If units of text have similar vectors in a text frequency matrix, then they tend to have similar meanings."

Great power, a great many design choices

tokenization
annotation
tagging
parsing
feature selection
⋮ cluster texts by date/author/discourse context/. . .
⇓ ⇙

Matrix design	Reweighting	Dimensionality reduction	Vector comparison
word × document	probabilities	LSA	Euclidean
word × word	length norm.	PLSA	Cosine
word × search proximity	TF-IDF	LDA	Dice
adj. × modified noun	PMI	PCA	Jaccard
word × dependency rel.	Positive PMI	NNMF	KL
⋮	⋮	⋮	⋮

(Nearly the full cross-product to explore; only a handful of the combinations are ruled out mathematically. Models like GloVe and word2vec offer packaged solutions to design/weighting/reduction and reduce the importance of the choice of comparison method. Contextual embeddings dictate many preprocessing choices.)

References I

John R. Firth. 1957. A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*, pages 1–32. Blackwell, Oxford.

Zellig Harris. 1954. Distributional structure. *Word*, 10(23):146–162.

Peter D. Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37:141–188.

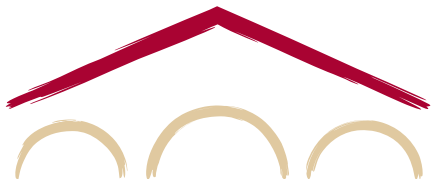
Ludwig Wittgenstein. 1953. *Philosophical Investigations*. The MacMillan Company, New York. Translated by G. E. M. Anscombe.

Distributed word representations: Matrix designs

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding



word x word

	:)	:/	:D	:|	;p	abandon	abc	ability	able	⋯
:)	74	1	0	0	0	1	0	2	2	
:/	1	306	0	0	0	0	0	0	17	
:D	0	0	16	0	0	0	6	1	1	
:|	0	0	0	120	0	0	0	1	9	
;p	0	0	0	0	516286	0	0	0	0	⋯
abandon	1	0	0	0	0	370	24	65	235	
abc	0	0	6	0	0	24	7948	77	291	
ability	2	0	1	1	0	65	77	4820	1807	
able	2	17	1	9	0	235	291	1807	14328	
⋮					⋮					

word x document

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
against	0	0	0	1	0	0	3	2	3	0
age	0	0	0	1	0	3	1	0	4	0
agent	0	0	0	0	0	0	0	0	0	0
ages	0	0	0	0	0	2	0	0	0	0
ago	0	0	0	2	0	0	0	0	3	0
agree	0	1	0	0	0	0	0	0	0	0
ahead	0	0	0	1	0	0	0	0	0	0
ain’t	0	0	0	0	0	0	0	0	0	0
air	0	0	0	0	0	0	0	0	0	0
aka	0	0	0	1	0	0	0	0	0	0

word x discourse context

Upper left corner of an interjection × dialog-act tag matrix derived from the Switchboard Dialog Act Corpus:

	Reject-part	Hedge	Completion	Tag question	Hold	Quotation	Accept	…
absolutely	0	2	0	0	0	0	95	
actually	17	12	0	0	1	0	4	
anyway	23	14	0	0	0	0	0	
boy	5	3	1	0	5	2	1	
bye	0	1	0	0	0	0	0	
bye-bye	0	0	0	0	0	0	0	…
dear	0	0	0	0	1	0	0	
definitely	0	2	0	0	0	0	56	
exactly	2	6	1	0	0	0	294	
gee	0	3	0	0	2	1	1	
goodness	1	0	0	0	2	0	0	
⋮				⋮				

Other designs

- adj. × modified noun
- word × syntactic context
- word × search query
- person × product
- word × person
- word × word × pattern
- verb × subject × object
- ⋮

Feature representations of data

- *the movie was horrible* becomes $[4, 0, 1/4]$.
- The complex, real-world response of an experimental subject to a particular example becomes $[0, 1]$ or $[118, 1]$.
- A human is modeled as a vector $[24, 140, 5, 12]$.
- A continuous, noisy speech stream is reduced to a restricted set of acoustic features.

Windows and scaling: What is a co-occurrence?

Windows and scaling: What is a co-occurrence?

from	swerve	of	shore	to	bend	of	bay	,	brings
4	3	2	1	0	1	2	3	4	5

Windows and scaling: What is a co-occurrence?

	from	swerve	of	shore	to	bend	of	bay	,	brings
	4	3	2	1	0	1	2	3	4	5

	from	swerve	of	shore	to	bend	of	bay	,	brings
Window: 3	4	3	2	1	0	1	2	3	4	5
Scaling: flat	0	1	1	1	1	1	1	1	0	0

Windows and scaling: What is a co-occurrence?

	from	swerve	of	shore	to	bend	of	bay	,	brings
	4	3	2	1	0	1	2	3	4	5

	from	swerve	of	shore	to	bend	of	bay	,	brings
Window: 3	4	3	2	1	0	1	2	3	4	5
Scaling: flat	0	1	1	1	1	1	1	1	0	0
Scaling: $\frac{1}{n}$	0	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{1}$	1	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	0	0

Windows and scaling: What is a co-occurrence?

from	swerve	of	shore	to	bend	of	bay	,	brings
4	3	2	1	0	1	2	3	4	5

- Larger, flatter windows capture more semantic information.
- Small, more scaled windows capture more syntactic (collocational) information.
- Textual boundaries can be separately controlled; core unit as the sentence/paragraph/document will have major consequences.

Code snippets

```
import os
import pandas as pd

DATA_HOME = os.path.join('data', 'vsmdata')

# Yelp: Window size = 5; scaling = 1/n
yelp5 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window5-scaled.csv.gz'), index_col=0)

# Yelp: Window size = 20; scaling = flat
yelp20 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window20-flat.csv.gz'), index_col=0)

# Gigaword: Window size = 5; scaling = 1/n
giga5 = pd.read_csv(
    os.path.join(DATA_HOME, 'giga_window5-scaled.csv.gz'), index_col=0)

# Gigaword: Window size = 20; scaling = flat
giga20 = pd.read_csv(
    os.path.join(DATA_HOME, 'giga_window20-flat.csv.gz'), index_col=0)
```

Distributed word representations: Vector comparison

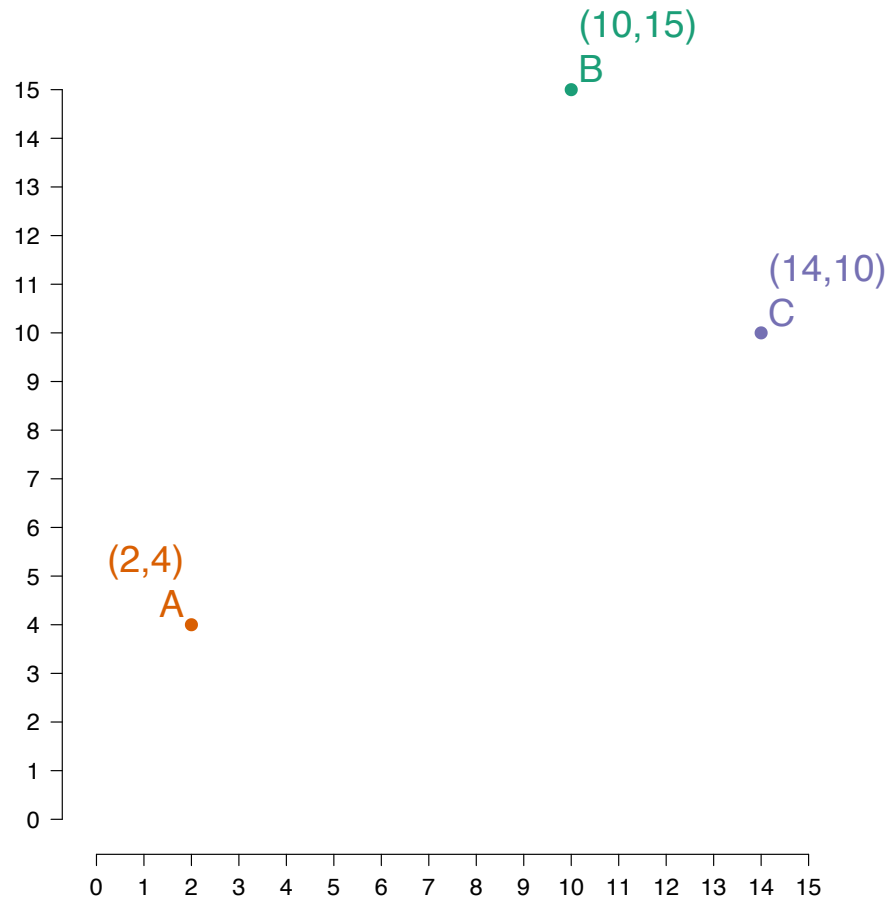
Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

Running example

	d_x	d_y
A	2	4
B	10	15
C	14	10



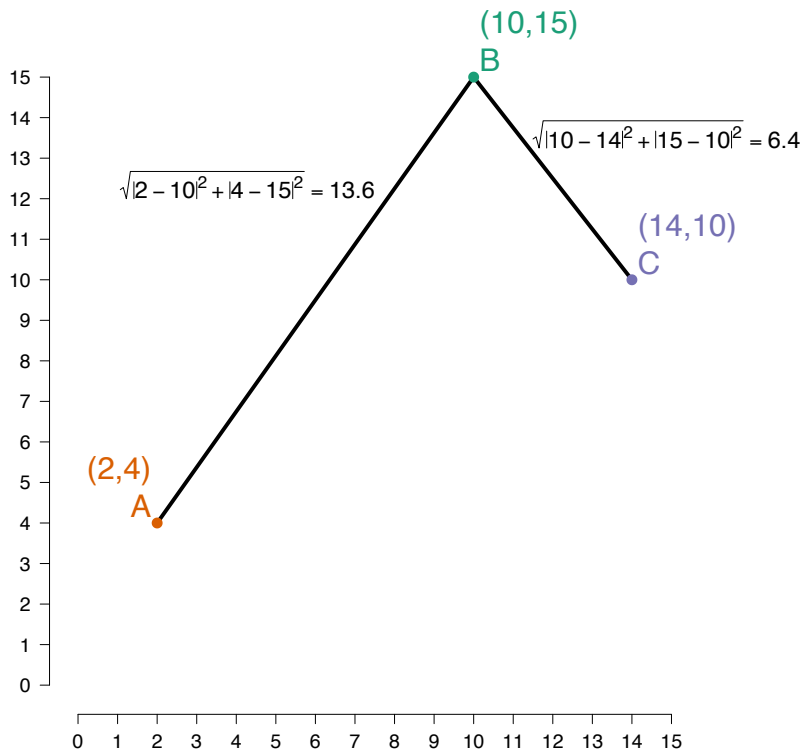
- Focus on distance measures
- Illustrations with row vectors

Euclidean

Between vectors u and v of dimension n:

$$\textbf{euclidean}(u, v) = \sqrt{\sum_{i=1}^{n} |u_i - v_i|^2}$$

	d_x	d_y
A	2	4
B	10	15
C	14	10



Length normalization

Given a vector u of dimension n, the L2-length of u is

$$||u||_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$$

and the length normalization of u is

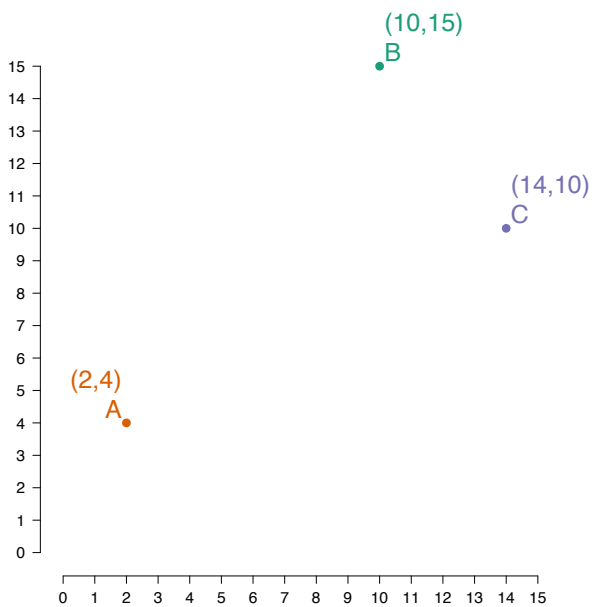
$$\left[\frac{u_1}{||u||_2}, \frac{u_2}{||u||_2}, \cdots, \frac{u_n}{||u||_2} \right]$$

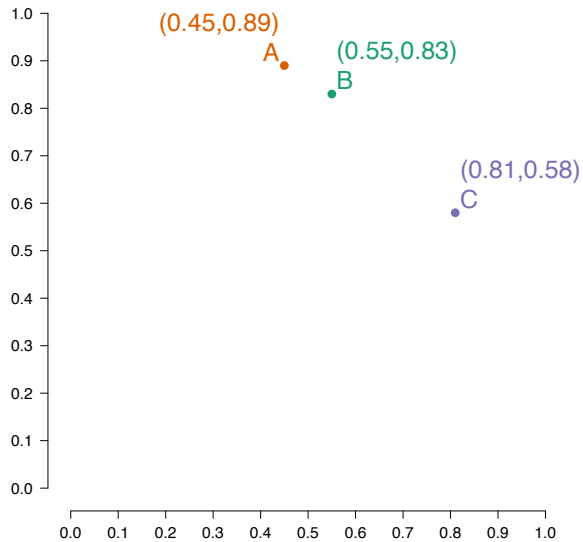
Length normalization

	d_x	d_y	$\|\|u\|\|_2$
A	2	4	4.47
B	10	15	18.03
C	14	10	17.20

row L2 norm $\Rightarrow$

	d_x	d_y
A	$\frac{2}{4.47}$	$\frac{4}{4.47}$
B	$\frac{10}{18.03}$	$\frac{15}{18.03}$
C	$\frac{14}{17.20}$	$\frac{10}{17.20}$





Cosine distance

Between vectors u and v of dimension n:

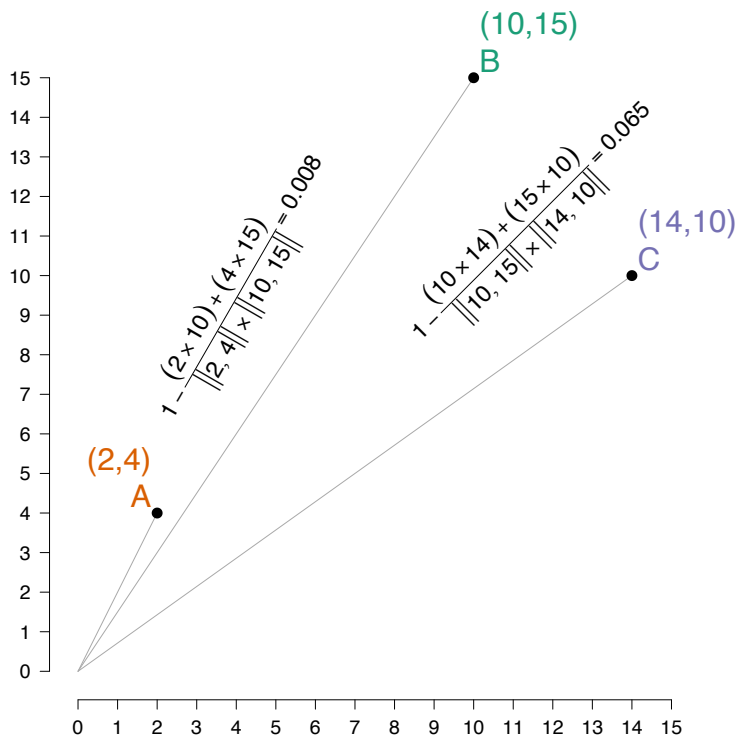
$$\textbf{cosine}(u, v) = 1 - \frac{\sum_{i=1}^{n} u_i \times v_i}{||u||_2 \times ||v||_2}$$

Cosine distance

Between vectors u and v of dimension n:

$$\textbf{cosine}(u, v) = 1 - \frac{\sum_{i=1}^{n} u_i \times v_i}{\|u\|_2 \times \|v\|_2}$$

	d_x	d_y
A	2	4
B	10	15
C	14	10

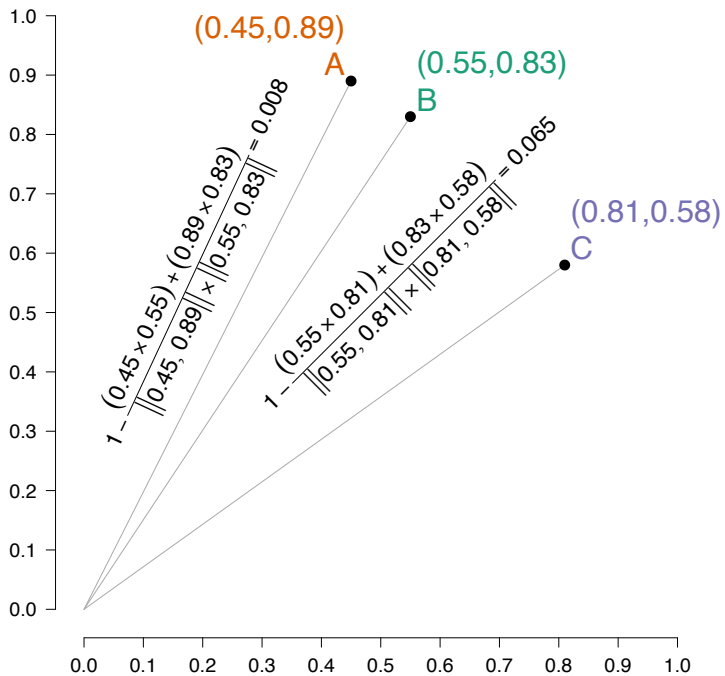


Cosine distance

Between vectors u and v of dimension n:

$$\textbf{cosine}(u, v) = 1 - \frac{\sum_{i=1}^{n} u_i \times v_i}{\|u\|_2 \times \|v\|_2}$$

	d_x	d_y
A	2	4
B	10	15
C	14	10



Matching-based methods

Matching coefficient

$$\textbf{matching}(u, v) = \sum_{i=1}^{n} \min(u_i, v_i)$$

Jaccard distance

$$\textbf{jaccard}(u, v) = 1 - \frac{\textbf{matching}(u, v)}{\sum_{i=1}^{n} \max(u_i, v_i)}$$

Dice distance

$$\textbf{dice}(u, v) = 1 - \frac{2 \times \textbf{matching}(u, v)}{\sum_{i=1}^{n} u_i + v_i}$$

Overlap

$$\textbf{overlap}(u, v) = 1 - \frac{\textbf{matching}(u, v)}{\min\left(\sum_{i=1}^{n} u_i \ , \ \sum_{i=1}^{n} v_i\right)}$$

KL divergence and variants

KL divergence

Between probability distributions p and q:

$$D(p \parallel q) = \sum_{i=1}^{n} p_i \log\left(\frac{p_i}{q_i}\right)$$

p is the reference distribution. Before calculation, smooth by adding ϵ.

Symmetric KL

$$D(p \parallel q) + D(q \parallel p)$$

Jensen–Shannon distance

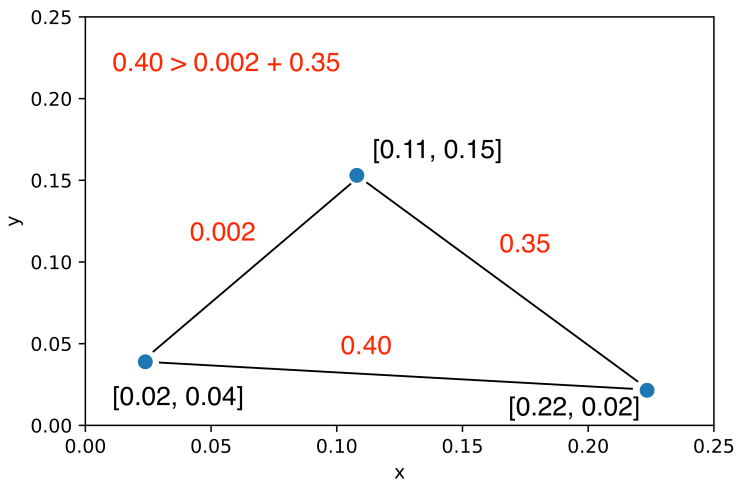
$$\sqrt{\frac{1}{2}D\left(p \parallel \frac{p+q}{2}\right) + \frac{1}{2}D\left(q \parallel \frac{p+q}{2}\right)}$$

Proper distance metric?

To qualify as a distance metric, a vector comparison method d has to be symmetric ($d(x, y) = d(y, x)$), assign 0 to identical vectors ($d(x, x) = 0$), and satisfy the **triangle inequality**:

$$d(x, z) \leqslant d(x, y) + d(y, z)$$

Cosine distance as I defined it doesn't satisfy this:



Distance metric?

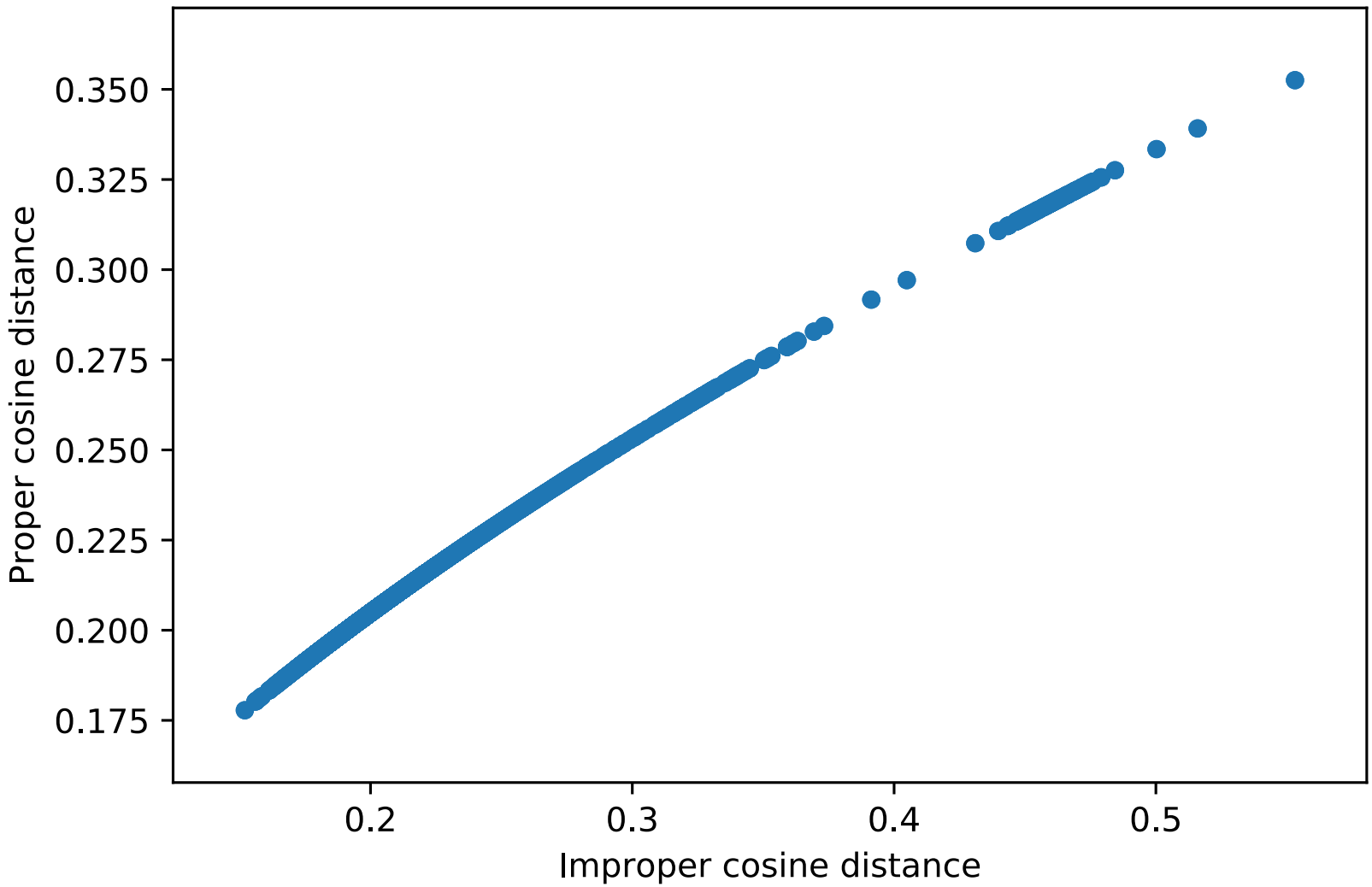
Yes: Euclidean, Jaccard for binary vectors, Jensen–Shannon, cosine as

$$\frac{\cos^{-1}\left(\frac{\sum_{i=1}^{n} u_i \times v_i}{||u||_2 \times ||v||_2}\right)}{\pi}$$

No: Matching, Jaccard, Dice, Overlap, KL divergence, Symmetric KL

Comparing the two versions of cosine

Random sample of 100 vectors from our `giga20` count matrix. Correlation is 99.8.



Relationships and generalizations

1. Euclidean, Jaccard, and Dice with raw count vectors will tend to favor raw frequency over distributional patterns.

2. Euclidean with L2-normed vectors is equivalent to cosine w.r.t. ranking.

3. Jaccard and Dice are equivalent w.r.t. ranking.

4. Both L2-norms and probability distributions can obscure differences in the amount/strength of evidence, which can in turn have an effect on the reliability of cosine, normed-euclidean, and KL divergence. These shortcomings might be addressed through weighting schemes.

Code snippets

[1]:
```
import os
import pandas as pd
import vsm
```

[2]:
```
ABC = pd.DataFrame([
    [ 2.0,  4.0],
    [10.0, 15.0],
    [14.0, 10.0]], index=['A', 'B', 'C'], columns=['x', 'y'])
```

[3]:
```
vsm.euclidean(ABC.loc['A'], ABC.loc['B'])
```

[3]: 13.601470508735444

[4]:
```
vsm.vector_length(ABC.loc['A'])
```

[4]: 4.47213595499958

[5]:
```
vsm.length_norm(ABC.loc['A']).values
```

[5]: array([0.4472136 , 0.89442719])

[6]:
```
vsm.cosine(ABC.loc['A'], ABC.loc['B'])
```

[6]: 0.007722123286332261

[7]:
```
vsm.matching(ABC.loc['A'], ABC.loc['B'])
```

[7]: 6.0

[8]:
```
vsm.jaccard(ABC.loc['A'], ABC.loc['B'])
```

[8]: 0.76

Code snippets

[9]:
```
DATA_HOME = os.path.join('data', 'vsmdata')

yelp5 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window5-scaled.csv.gz'), index_col=0)
```

[10]:
```
vsm.cosine(yelp5.loc['good'], yelp5.loc['excellent'])
```

[10]:
```
0.1197421543700451
```

[11]:
```
vsm.cosine(yelp5.loc['good'], yelp5.loc['bad'])
```

[11]:
```
0.14118253033888817
```

[12]:
```
vsm.neighbors('bad', yelp5).head()
```

[12]:
```
bad              0.000000
unfortunately    0.116183
memorable        0.120179
…               0.122024
obviously        0.123120
dtype: float64
```

[13]:
```
vsm.neighbors('bad', yelp5, distfunc=vsm.jaccard).head(3)
```

[13]:
```
bad       0.000000
..        0.452427
though    0.484269
dtype: float64
```

Distributed word representations: Basic reweighting

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

Goals of reweighting

- Amplify the important, the trustworthy, the unusual; deemphasize the mundane and the quirky.
- Absent a defined objective function, this will remain fuzzy.
- The intuition behind moving away from raw counts is that frequency is a poor proxy for the above values.
- So we should ask of each weighting scheme: How does it compare to the raw count values?
- What overall distribution of values does it deliver?
- We hope to do no feature selection based on counts, stopword dictionaries, etc. Rather, we want our methods to reveal what's important without these ad hoc interventions.

Normalization

L2 norming (repeated from earlier)

Given a vector u of dimension n, the L2-length of u is

$$||u||_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$$

and the length normalization of u is

$$\left[\frac{u_1}{||u||_2}, \frac{u_2}{||u||_2}, \cdots, \frac{u_n}{||u||_2} \right]$$

Probability distribution

Given a vector u of dimension n containing all positive values, let

$$\mathbf{sum}(u) = \sum_{i=1}^{n} u_i$$

and then the probability distribution of u is

$$\left[\frac{u_1}{\mathbf{sum}(u)}, \frac{u_2}{\mathbf{sum}(u)}, \cdots, \frac{u_n}{\mathbf{sum}(u)} \right]$$

Observed/Expected

$$\textbf{rowsum}(X, i) = \sum_{j=1}^{n} X_{ij} \quad \textbf{colsum}(X, j) = \sum_{i=1}^{m} X_{ij} \quad \textbf{sum}(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij}$$

$$\textbf{expected}(X, i, j) = \frac{\textbf{rowsum}(X, i) \cdot \textbf{colsum}(X, j)}{\textbf{sum}(X)}$$

$$\textbf{oe}(X, i, j) = \frac{X_{ij}}{\textbf{expected}(X, i, j)}$$

Observed/Expected

$$\textbf{rowsum}(X,i) = \sum_{j=1}^{n} X_{ij} \quad \textbf{colsum}(X,j) = \sum_{i=1}^{m} X_{ij} \quad \textbf{sum}(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}$$

$$\textbf{expected}(X,i,j) = \frac{\textbf{rowsum}(X,i) \cdot \textbf{colsum}(X,j)}{\textbf{sum}(X)}$$

$$\textbf{oe}(X,i,j) = \frac{X_{ij}}{\textbf{expected}(X,i,j)}$$

	a	*b*	**rowsum**
x	34	11	45
y	47	7	54
colsum	81	18	**99**

oe ⇒

	a	*b*
x	$\frac{34}{\frac{45\cdot 81}{99}}$	$\frac{11}{\frac{45\cdot 18}{99}}$
y	$\frac{47}{\frac{54\cdot 81}{99}}$	$\frac{7}{\frac{54\cdot 18}{99}}$

Observed/Expected

$$\textbf{rowsum}(X, i) = \sum_{j=1}^{n} X_{ij} \quad \textbf{colsum}(X, j) = \sum_{i=1}^{m} X_{ij} \quad \textbf{sum}(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}$$

$$\textbf{expected}(X, i, j) = \frac{\textbf{rowsum}(X, i) \cdot \textbf{colsum}(X, j)}{\textbf{sum}(X)}$$

$$\textbf{oe}(X, i, j) = \frac{X_{ij}}{\textbf{expected}(X, i, j)}$$

Observed

	tabs	reading	birds
keep	20	20	20
enjoy	1	20	20

keep and *tabs* co-occur more than expected given their frequencies, *enjoy* and *tabs* less than expected

Expected

	tabs	reading	birds
keep	$\frac{60 \cdot 21}{101}$	$\frac{60 \cdot 40}{101}$	$\frac{60 \cdot 40}{101}$
enjoy	$\frac{41 \cdot 21}{101}$	$\frac{41 \cdot 40}{101}$	$\frac{41 \cdot 40}{101}$

=

	tabs	reading	birds
keep	12.48	23.76	23.76
enjoy	8.5	16.24	16.24

Pointwise Mutual Information (PMI)

PMI is observed/expected in log-space (with $\log_e(0) = 0$):

$$\textbf{pmi}(X, i, j) = \log_e\left(\frac{X_{ij}}{\textbf{expected}(X, i, j)}\right) = \log_e\left(\frac{P(X_{ij})}{P(X_{i*}) \cdot P(X_{*j})}\right)$$

	d_1	d_2	d_3	d_4
A	10	10	10	10
B	10	10	10	0
C	10	10	0	0
D	0	0	0	1

$\Rightarrow$

	$P(w,d)$				$P(w)$
A	0.11	0.11	0.11	0.11	0.44
B	0.11	0.11	0.11	0.00	0.33
C	0.11	0.11	0.00	0.00	0.22
D	0.00	0.00	0.00	0.01	0.01
$P(d)$	0.33	0.33	0.22	0.12	

PMI

$\Downarrow$

	d_1	d_2	d_3	d_4
A	−0.28	−0.28	0.13	0.73
B	0.01	0.01	0.42	0.00
C	0.42	0.42	0.00	0.00
D	0.00	0.00	0.00	2.11

Positive PMI

The issue

PMI is actually undefined when $X_{ij} = 0$. The usual response is the one given above: set PMI to 0 in such cases. However, this is arguably not coherent (Levy and Goldberg 2014):

- Larger than expected count ⇒ large PMI
- Smaller than expected count ⇒ small PMI
- 0 count ⇒ placed right in the middle!?

Other weighting/normalization schemes

- t-test: $\frac{P(w,d)-P(w)P(d)}{\sqrt{P(w)P(d)}}$

- TF-IDF: For a corpus of documents D:
  - Term frequency (TF):
  $$\frac{x_{ij}}{\textbf{colsum}(X,j)}$$
  - Inverse document frequency (IDF):
  $$\log_e\left(\frac{|D|}{\left|\{d \in D : w \in d\}\right|}\right) \qquad \log_e(0) = 0$$
  - TF-IDF: TF $\cdot$ IDF

- Pairwise distance matrices:

	d_x	d_y
A	2	4
B	10	15
C	14	10

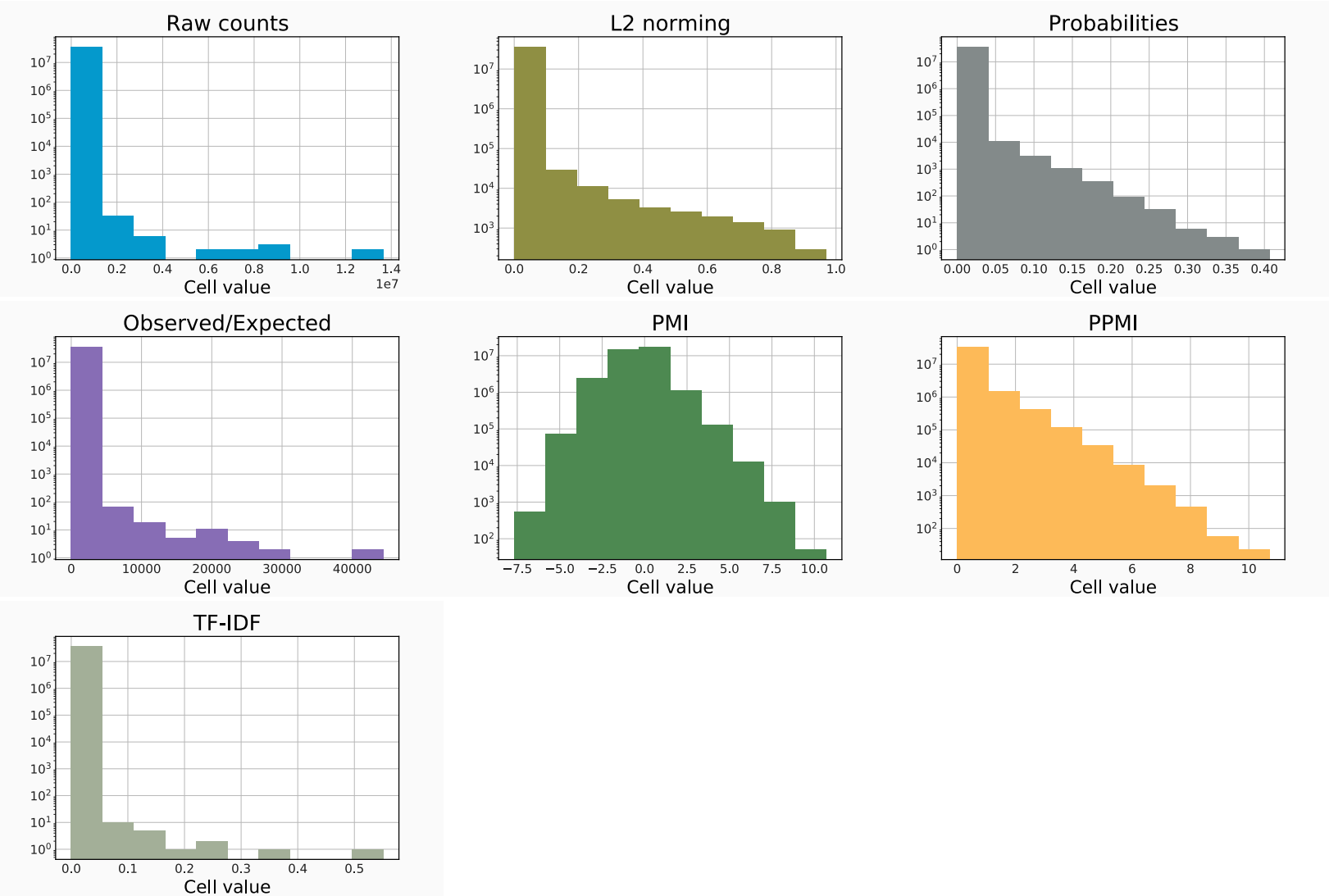
cosine $\Rightarrow$

	A	B	C
A	0	0.008	0.116
B	0.008	0	0.065
C	0.116	0.065	0

High-level effects

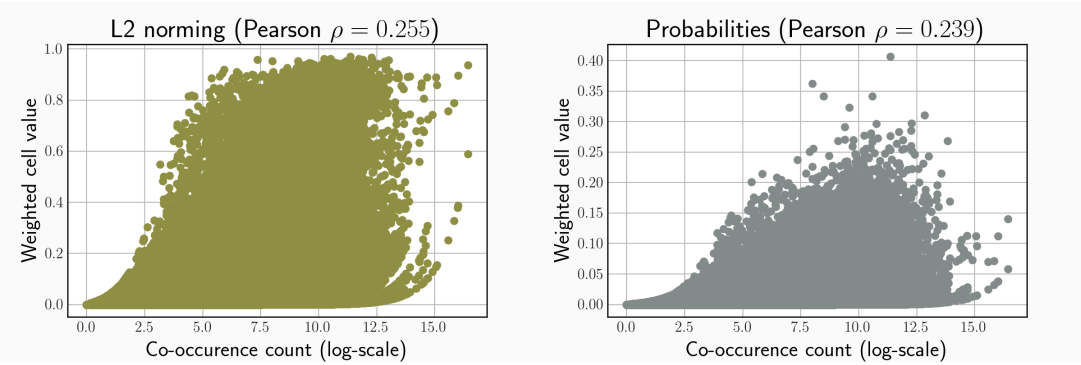
- Amplify the important, the trustworthy, the unusual; deemphasize the mundane and the quirky.
- Absent a defined objective function, this will remain fuzzy.
- So we should ask of each weighting scheme: How does it compare to the raw count values?
- What overall distribution of values does it deliver?
- We hope to do no feature selection based on counts, stopword dictionaries, etc. Rather, we want our methods to reveal what's important without these ad hoc interventions.

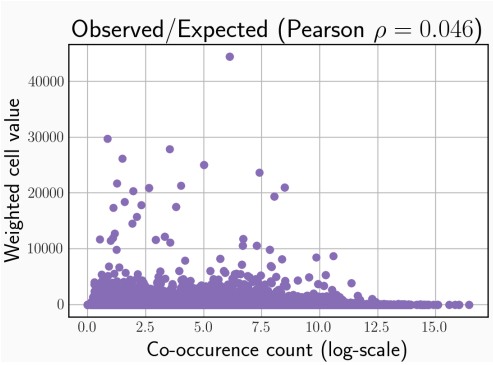
Weighting scheme cell-value distributions

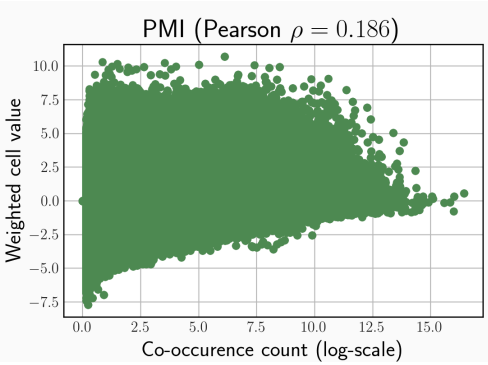


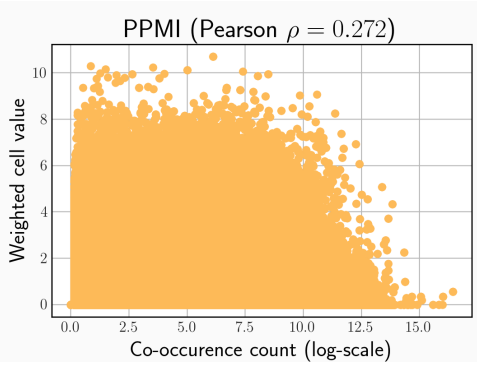
Uses the `giga5` matrix loaded earlier. Others look similar.

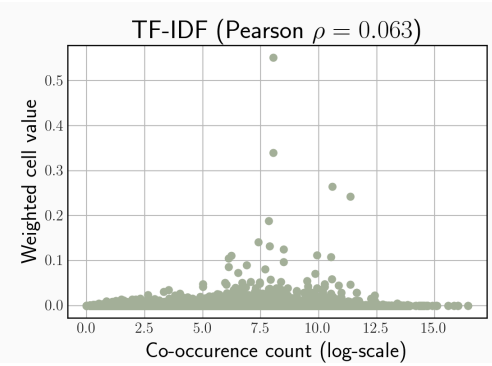
Weighting scheme relationships to counts











Uses the `giga5` matrix loaded earlier. Others look similar.

Relationships and generalizations

- The theme running through nearly all these schemes is that we want to weight a cell value X_{ij} relative to the value we expect given X_{i*} and X_{*j}.
- The magnitude of counts can be important; $[1, 10]$ and $[1000, 10000]$ might represent very different situations; creating probability distributions or length normalizing will obscure this.
- PMI and its variants will amplify the values of counts that are tiny relative to their rows and columns. Unfortunately, with language data, these might be noise noise.
- TF-IDF severely punishes words that appear in many documents – it behaves oddly for dense matrices, which can include word × word matrices.

Code snippets

[1]:
```
import os
import pandas as pd
import vsm
```

[2]:
```
DATA_HOME = os.path.join('data', 'vsmdata')
```

[3]:
```
yelp5 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window5-scaled.csv.gz'), index_col=0)
```

[4]:
```
yelp_oe = vsm.observed_over_expected(yelp5)
```

[5]:
```
yelp_norm = yelp5.apply(vsm.length_norm, axis=1)
```

[6]:
```
yelp5_ppmi = vsm.pmi(yelp5)
```

[7]:
```
yelp5_pmi = vsm.pmi(yelp5, positive=False)
```

[8]:
```
yelp5_tfidf = vsm.tfidf(yelp5)
```

Code snippets

```
[9]: vsm.neighbors('bad', yelp5).head()

[9]: bad              0.000000
     unfortunately    0.116183
     memorable        0.120179
     …                0.122024
     obviously        0.123120
     dtype: float64

[10]: vsm.neighbors('bad', yelp5_ppmi).head()

[10]: bad          0.000000
      terrible     0.471554
      horrible     0.516562
      awful        0.571104
      poor         0.599081
      dtype: float64
```

References I

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*.

Distributed word representations: Dimensionality reduction

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

Overview

1. Latent Semantic Analysis
2. Autoencoders
3. GloVe
4. Visualization

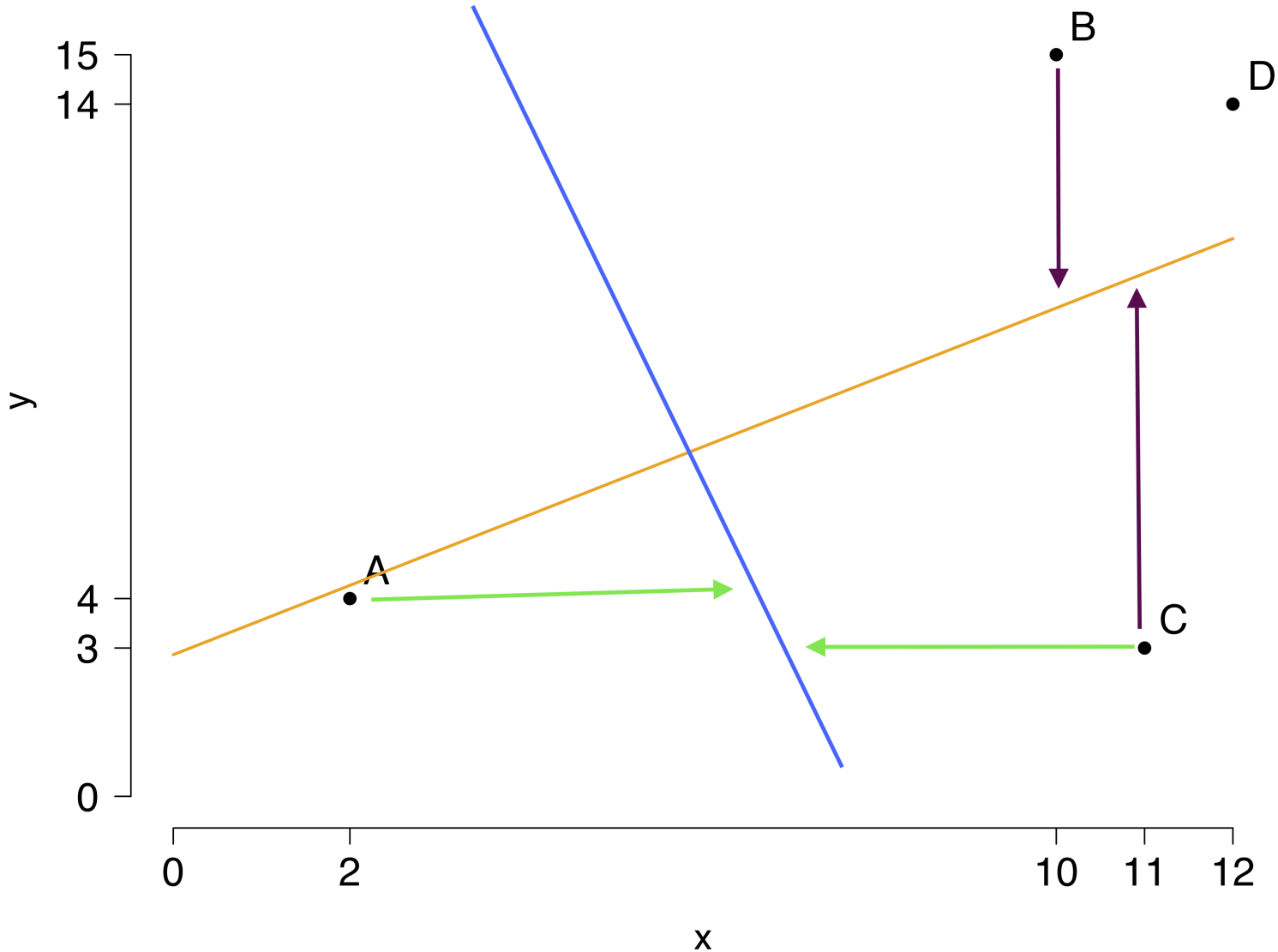
Latent Semantic Analysis (LSA)

1. Latent Semantic Analysis
2. Autoencoders
3. GloVe
4. Visualization

Overview

- Due to Deerwester et al. 1990.
- One of the oldest and most widely used dimensionality reduction techniques.
- Also known as Truncated Singular Value Decomposition (Truncated SVD).
- Standard baseline, often very tough to beat.

Guiding intuitions for LSA



The LSA method

Singular value decomposition

For any matrix of real numbers A of dimension $(m \times n)$ there exists a factorization into matrices T, S, D such that

$$A_{m \times n} = T_{m \times m} S_{m \times m} D^T_{n \times m}$$

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot & & \\ & \cdot & \\ & & \cdot \end{pmatrix} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}^T$$

$$A_{3 \times 4} \quad = \quad T_{3 \times 3} \qquad S_{3 \times 3} \qquad D^T_{4 \times 3}$$

Idealized LSA example

	d1	d2	d3	d4	d5	d6
gnarly	1	0	1	0	0	0
wicked	0	1	0	1	0	0
awesome	1	1	1	1	0	0
lame	0	0	0	0	1	1
terrible	0	0	0	0	0	1

Distance from *gnarly*
1. gnarly
2. awesome
3. terrible
4. wicked
5. lame

⇓⇑

T(erm)					S(ingular values)					D(ocument)					
gnarly	0.41	0.00	0.71	0.00	-0.58	2.45	0.00	0.00	0.00	0.00	d1	0.50	-0.00	0.50	0.00	-0.71
wicked	0.41	0.00	-0.71	0.00	-0.58	0.00	1.62	0.00	0.00	0.00	d2	0.50	0.00	-0.50	0.00	0.00
awesome	0.82	-0.00	-0.00	-0.00	0.58	0.00	0.00	1.41	0.00	0.00	d3	0.50	-0.00	0.50	0.00	0.71
lame	0.00	0.85	0.00	-0.53	0.00	0.00	0.00	0.00	0.62	0.00	d4	0.50	-0.00	-0.50	-0.00	0.00
terrible	0.00	0.53	0.00	0.85	0.00	0.00	0.00	0.00	0.00	-0.00	d5	-0.00	0.53	0.00	-0.85	0.00
											d6	0.00	0.85	0.00	0.53	0.00

$\times$ $\times$ T

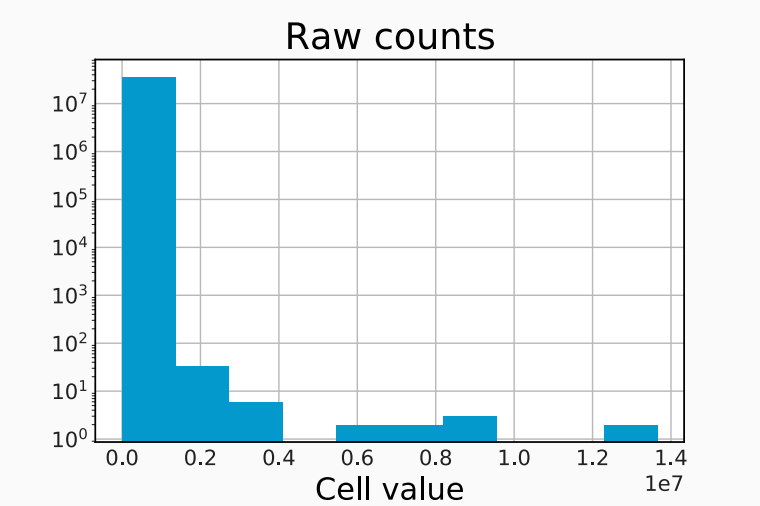
gnarly	0.41	0.00			
wicked	0.41	0.00			
awesome	0.82	-0.00	$\times$	2.45	0.00
lame	0.00	0.85		0.00	1.62
terrible	0.00	0.53			

$=$

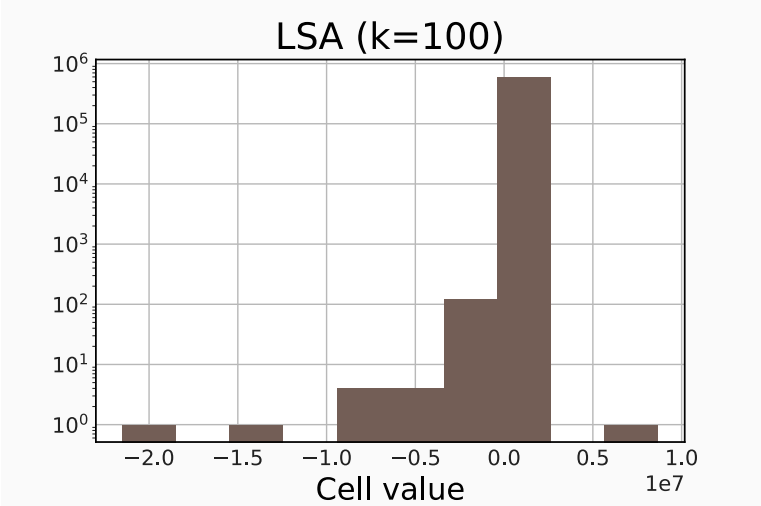
gnarly	1.00	0.00
wicked	1.00	0.00
awesome	2.00	0.00
lame	0.00	1.38
terrible	0.00	0.85

Distance from *gnarly*
1. gnarly
2. wicked
3. awesome
4. terrible
5. lame

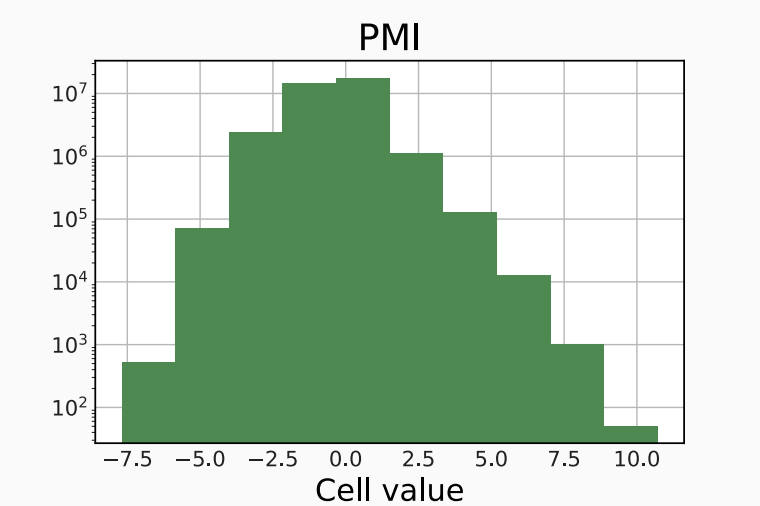
Cell-value comparisons ($k = 100$)



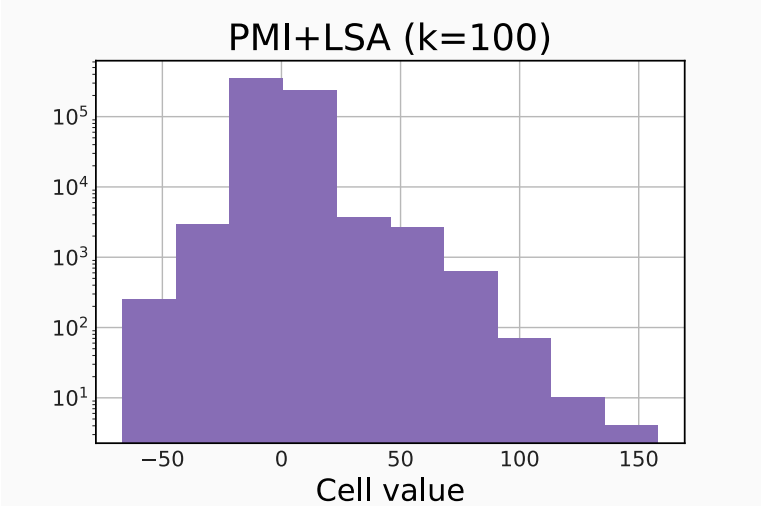
⇒



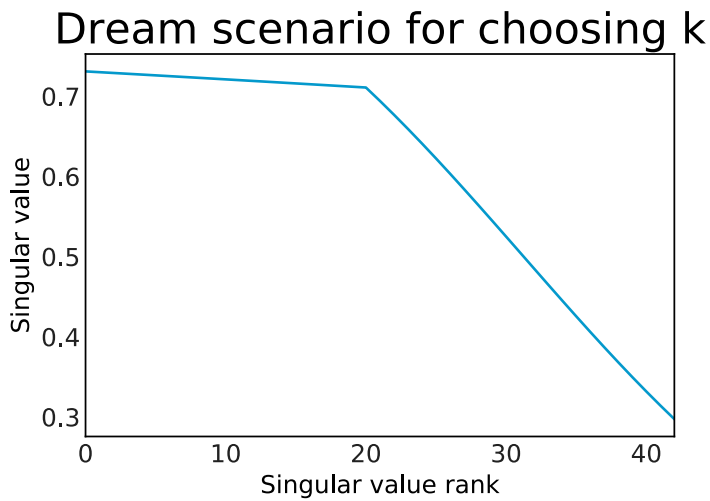
⇓

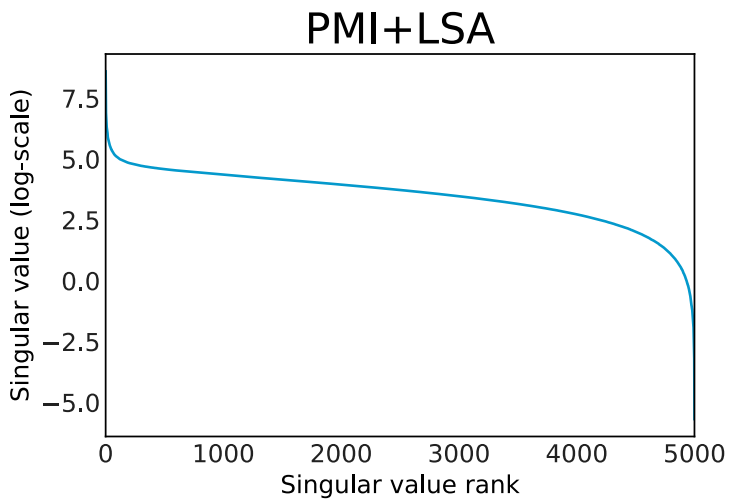


⇒



Choosing the LSA dimensionality





Related dimensionality reduction techniques

- Principal Components Analysis (PCA)
- Non-negative Matrix Factorization (NMF)
- Probabilistic LSA (PLSA; Hofmann 1999)
- Latent Dirichlet Allocation (LDA; Blei et al. 2003)
- t-SNE (van der Maaten and Hinton 2008)

See `sklearn.decomposition` and `sklearn.manifold`

Code snippets

[1]:
```
import os
import pandas as pd
import vsm
```

[2]:
```
DATA_HOME = os.path.join('data', 'vsmdata')

giga5 = pd.read_csv(
    os.path.join(DATA_HOME, 'giga_window5-scaled.csv.gz'), index_col=0)
```

[3]:
```
giga5.shape
```

[3]: (5000, 5000)

[4]:
```
giga5_lsa100 = vsm.lsa(giga5, k=100)
```

[5]:
```
giga5_lsa100.shape
```

[5]: (5000, 100)

Autoencoders

1. Latent Semantic Analysis
2. Autoencoders
3. GloVe
4. Visualization

Overview

- Autoencoders are a flexible class of deep learning architectures for learning reduced dimensional representations.
- Chapter 14 of Goodfellow et al. (2016) is an excellent discussion.

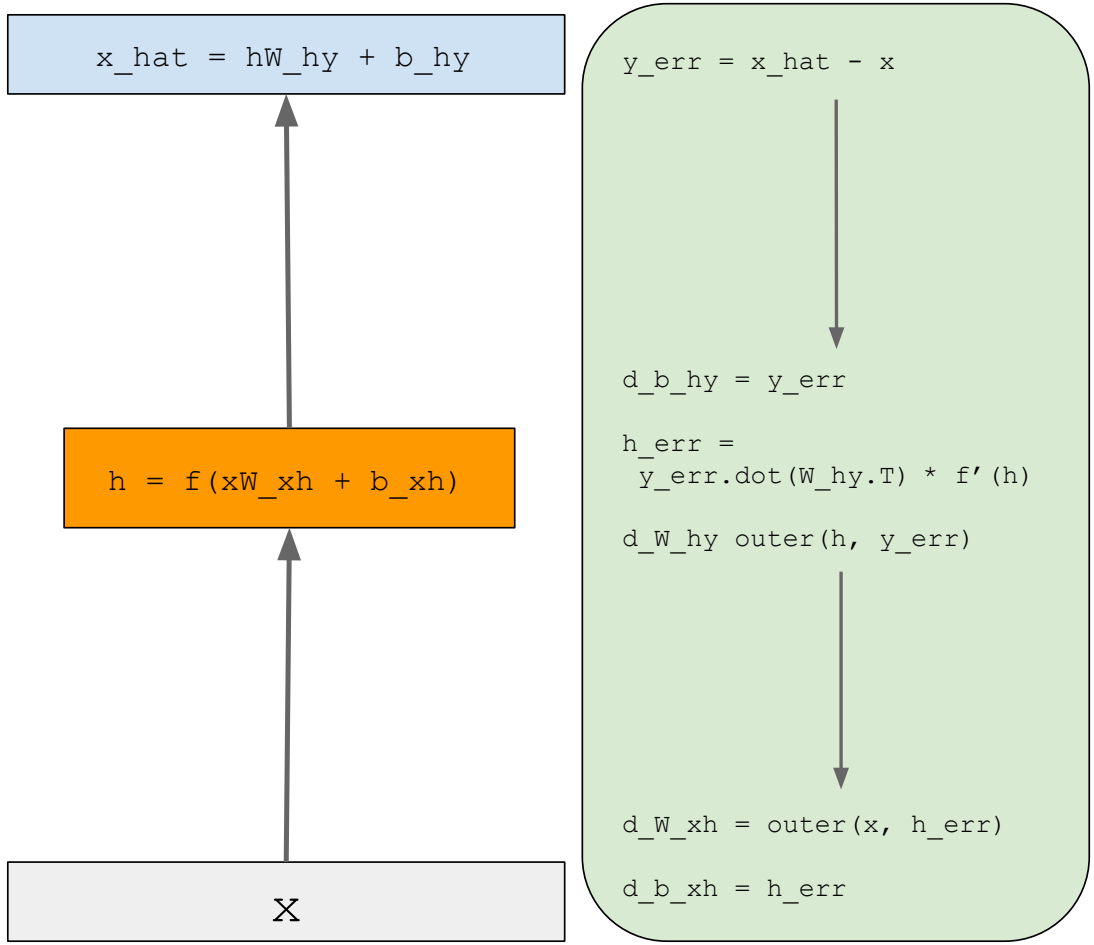
The basic autoencoder model

Assume `f = tanh` and so `f'(z) = 1.0 - z^2`. Per example error is $\sum_i$ `0.5 * (x_hat_i - x_i)^2`

Seeks to predict its own input.

High-dimensional inputs are fed through a narrow hidden layer (or multiple hidden layers). **This is the representation of interest – akin to LSA output.**

This might be preceded by a separate dimensionality reduction step (e.g., LSA)



Autoencoder code snippets

[1]:
```
from np_autoencoder import Autoencoder
import os
import pandas as pd
from torch_autoencoder import TorchAutoencoder
import vsm
```

[2]:
```
DATA_HOME = os.path.join('data', 'vsmdata')

giga5 = pd.read_csv(
    os.path.join(DATA_HOME, 'giga_window5-scaled.csv.gz'), index_col=0)
```

[3]:
```
# You'll likely need a larger network, trained longer, for good results.
ae = Autoencoder(max_iter=10, hidden_dim=50)
```

[4]:
```
# Scaling the values first will help the network learn:
giga5_l2 = giga5.apply(vsm.length_norm, axis=1)
```

[5]:
```
# The `fit` method returns the hidden reps:
giga5_ae = ae.fit(giga5_l2)
```

```
Finished epoch 10 of 10; error is 0.4883386066987744
```

[6]:
```
torch_ae = TorchAutoencoder(max_iter=10, hidden_dim=50)
```

[7]:
```
# A potentially interesting pipeline:
giga5_ppmi_lsa100 = vsm.lsa(vsm.pmi(giga5), k=100)
```

[8]:
```
giga5_ppmi_lsa100_ae = torch_ae.fit(giga5_ppmi_lsa100)
```

```
Finished epoch 10 of 10; error is 1.2230274677276611
```

Autoencoder code snippets

```
[9]: vsm.neighbors("finance", giga5).head()

[9]: finance     0.000000
     minister    0.870300
     .           0.880074
     </p>        0.896013
     ministry    0.897051
     dtype: float64

[10]: vsm.neighbors("finance", giga5_ae).head()

[10]: finance          0.000000
      article          0.504076
      style            0.526473
      domain           0.538920
      investigators    0.548903
      dtype: float64

[11]: vsm.neighbors("finance", giga5_ppmi_lsa100_ae).head()

[11]: finance       0.000000
      affairs       0.232635
      management    0.248080
      commerce      0.255099
      banking       0.256428
      dtype: float64
```

Global Vectors (GloVe)

1. Latent Semantic Analysis
2. Autoencoders
3. GloVe
4. Visualization

Overview

- Pennington et al. (2014)
- Roughly speaking, the objective is to learn vectors for words such that their dot product is proportional to their log probability of co-occurrence.
- We'll use the implementation in `torch_glove.py` in the course repo. There is a reference implementation in `vsm.py`. For really big vocabularies, the GloVe team's C implementation is probably the best choice.
- We'll make use of the GloVe team's pretrained representations throughout this course.

The GloVe objective

Equation (6):

$$w_i^\top \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

Allowing different rows and columns:

$$w_i^\top \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_{i*} \cdot X_{*k})$$

That's PMI!

$$\textbf{pmi}(X, i, j) = \log\left(\frac{X_{ij}}{\textbf{expected}(X, i, j)}\right) = \log\left(\frac{P(X_{ij})}{P(X_{i*}) \cdot P(X_{*j})}\right)$$

By the equivalence $\log(\frac{x}{y}) = \log(x) - \log(y)$

The weighted GloVe objective

Original

$$w_i^\top \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

Weighted

$$\sum_{i,j=1}^{|V|} f\left(X_{ij}\right)\left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

where V is the vocabulary and f is

$$f(x) \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

Typically, α is set to 0.75 and $x_{\max}$ to 100.

GloVe hyperparameters

- Learned representation dimensionality.
- $x_{\max}$, which flattens out all high counts.
- α, which scales the values as $(x/x_{\max})^{\alpha}$.

$$f(x) \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

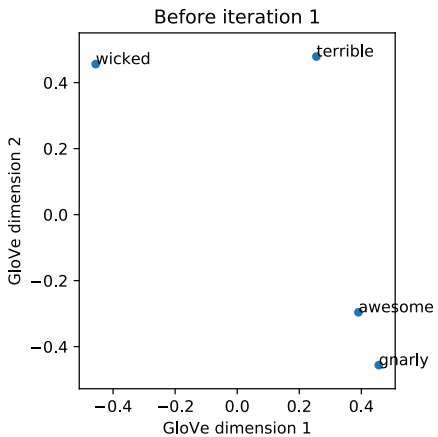
$$f\left(\begin{bmatrix} 100 & 99 & 75 & 10 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1.00 & 0.99 & 0.81 & 0.18 & 0.03 \end{bmatrix}$$

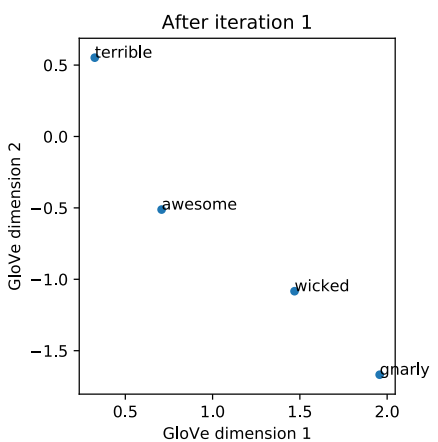
GloVe learning

The loss calculations

$$f(X_{ij})\left(w_i^\top \tilde{w}_j - \log X_{ij}\right)$$

show how *gnarly* and *wicked* are pulled toward *awesome*. Bias terms left out for simplicity. *gnarly* and *wicked* deliberately far apart in w_0 and $\tilde{w}_0$.





Counts	gnarly	wicked	awesome	terrible
gnarly	10	0	9	1
wicked	0	10	9	1
awesome	9	9	19	1
terrible	1	1	1	3

Weights($x_{\max} = 10, \alpha = 0.75$)	gnarly	wicked	awesome	terrible
gnarly	1.00	0.00	0.92	0.18
wicked	0.00	1.00	0.92	0.18
awesome	0.92	0.92	1.00	0.18
terrible	0.18	0.18	0.18	0.41

w_0		
gnarly	0.27	−0.27
wicked	−0.27	0.27
awesome	0.36	−0.50
terrible	0.08	0.16

$\tilde{w}_0$		
gnarly	0.18	−0.18
wicked	−0.18	0.18
awesome	0.03	0.20
terrible	0.17	0.32

$$0.92\left([\,0.27 \quad -0.27\,]^\top [\,0.03 \quad 0.20\,] - \log(9)\right) = -2.06$$

$$0.92\left([\,-0.27 \quad 0.27\,]^\top [\,0.03 \quad 0.20\,] - \log(9)\right) = -1.98$$

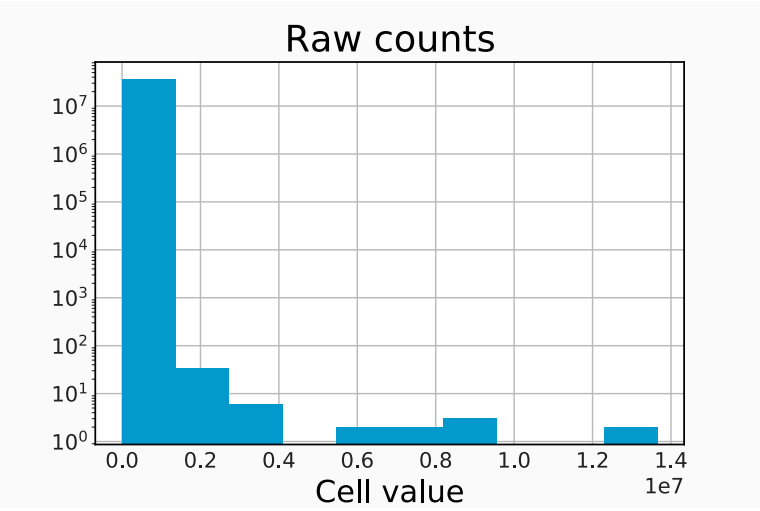
w_1		
gnarly	0.99	−0.85
wicked	0.74	−0.54
awesome	0.37	−0.26
terrible	0.12	0.21

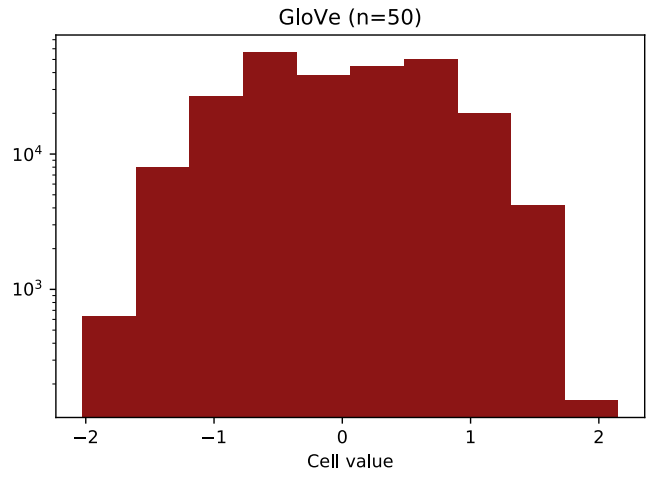
$\tilde{w}_1$		
gnarly	0.97	−0.82
wicked	0.73	−0.54
awesome	0.34	−0.25
terrible	0.20	0.34

$$0.92\left([\,0.99 \quad -0.85\,]^\top [\,0.34 \quad -0.25\,] - \log(9)\right) = -1.51$$

$$0.92\left([\,0.74 \quad -0.54\,]^\top [\,0.34 \quad -0.25\,] - \log(9)\right) = -1.66$$

GloVe cell-value comparisons ($n = 50$)





GloVe code snippets

[1]:
```
from torch_glove import TorchGloVe
import os
import pandas as pd
```

[2]:
```
DATA_HOME = os.path.join('data', 'vsmdata')
yelp5 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window5-scaled.csv.gz'), index_col=0)
yelp20 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window20-flat.csv.gz'), index_col=0)
```

[3]:
```
# What percentage of the non-zero values are being mapped to 1 by f?
def percentage_nonzero_vals_above(df, n=100):
    v = df.values.reshape(1, -1).squeeze()
    v = v[v > 0]
    above  = v[v > n]
    return len(above) / len(v)
```

[4]:
```
percentage_nonzero_vals_above(yelp5)
```

[4]:
```
0.049558084774404466
```

[5]:
```
percentage_nonzero_vals_above(yelp20)
```

[5]:
```
0.20425339735840817
```

[6]:
```
glv = TorchGloVe(max_iter=100, embed_dim=50)
```

[7]:
```
yelp5_glv = glv.fit(yelp5)
```

```
Finished epoch 100 of 100; error is 2361281.46875
```

[8]:
```
# Are dot products of learned vectors proportional
# to the log co-occurrence probabilities?
glv.score(yelp5)
```

[8]:
```
0.32520973952703197
```

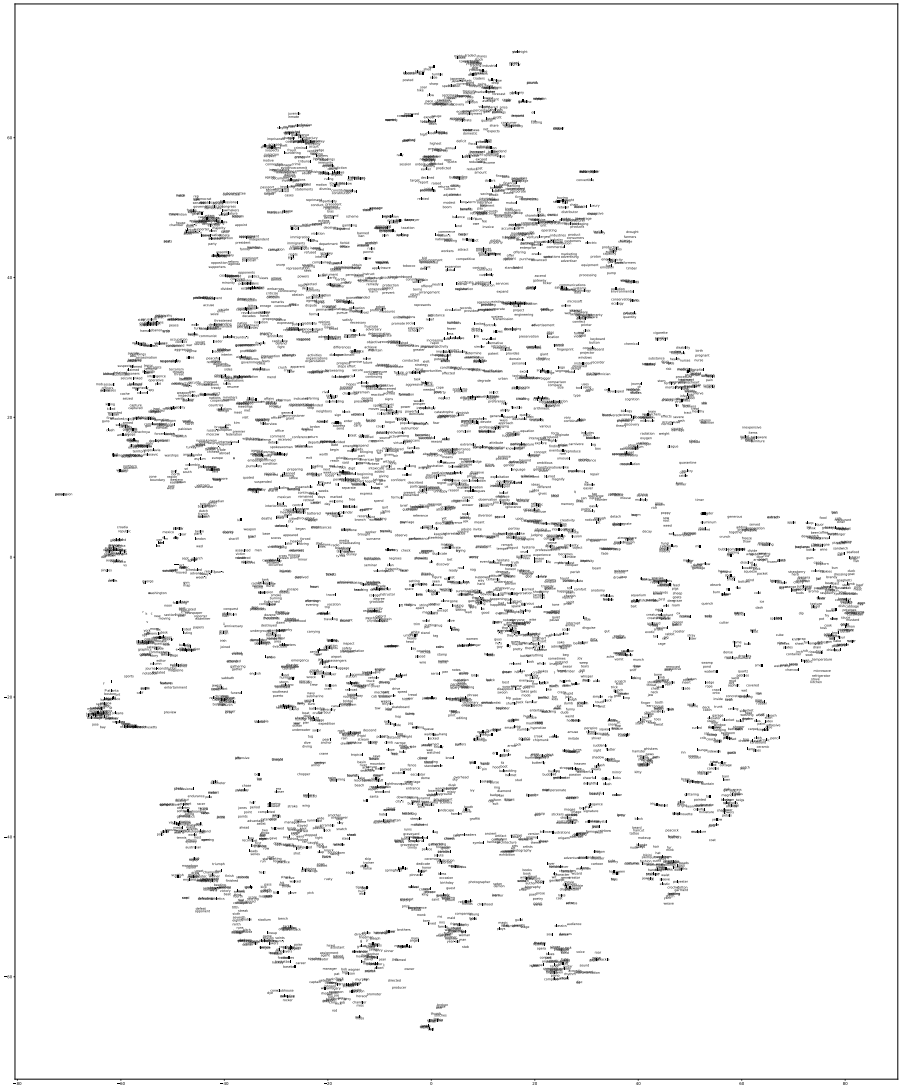
Visualization

1. Latent Semantic Analysis
2. Autoencoders
3. GloVe
4. Visualization

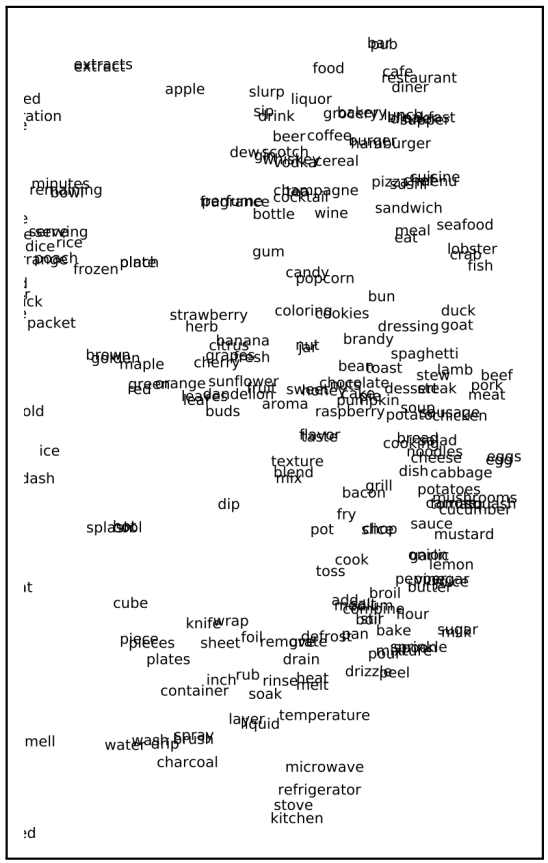
Techniques

- Our goal is to visualize very high-dimensional spaces in two or three dimensions. **This will inevitably involve compromises.**
- Still, visualization can give you a feel for what is in your VSM, especially if you pair it with other kinds of qualitative exploration (e.g., using `vsm.neighbors`).
- There are many visualization techniques implemented in `sklearn.manifold`; see this user guide for an overview and discussion of trade-offs.

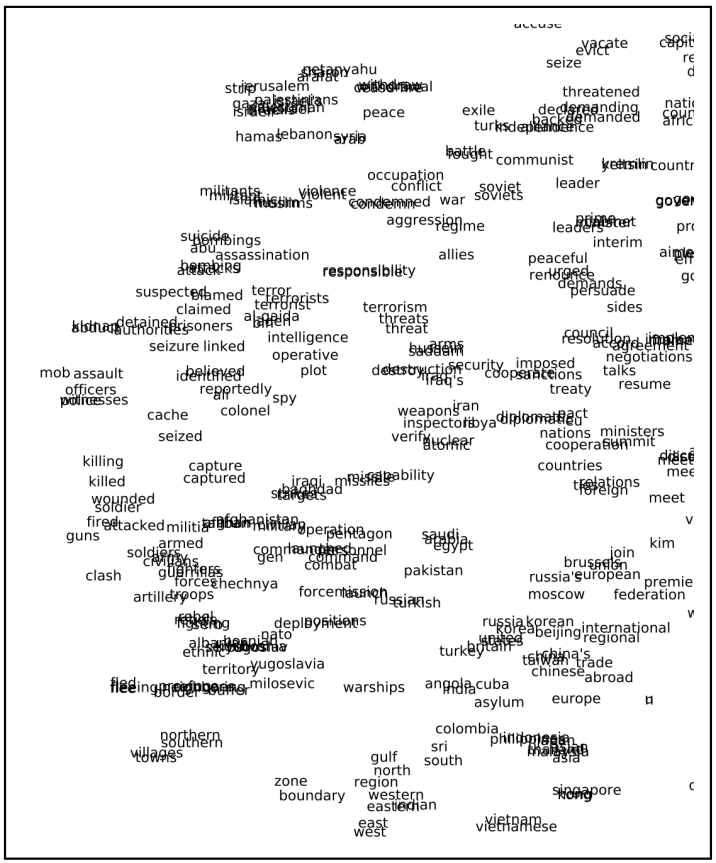
t-SNE on the giga20 PPMI VSM



t-SNE on the giga20 PPMI VSM

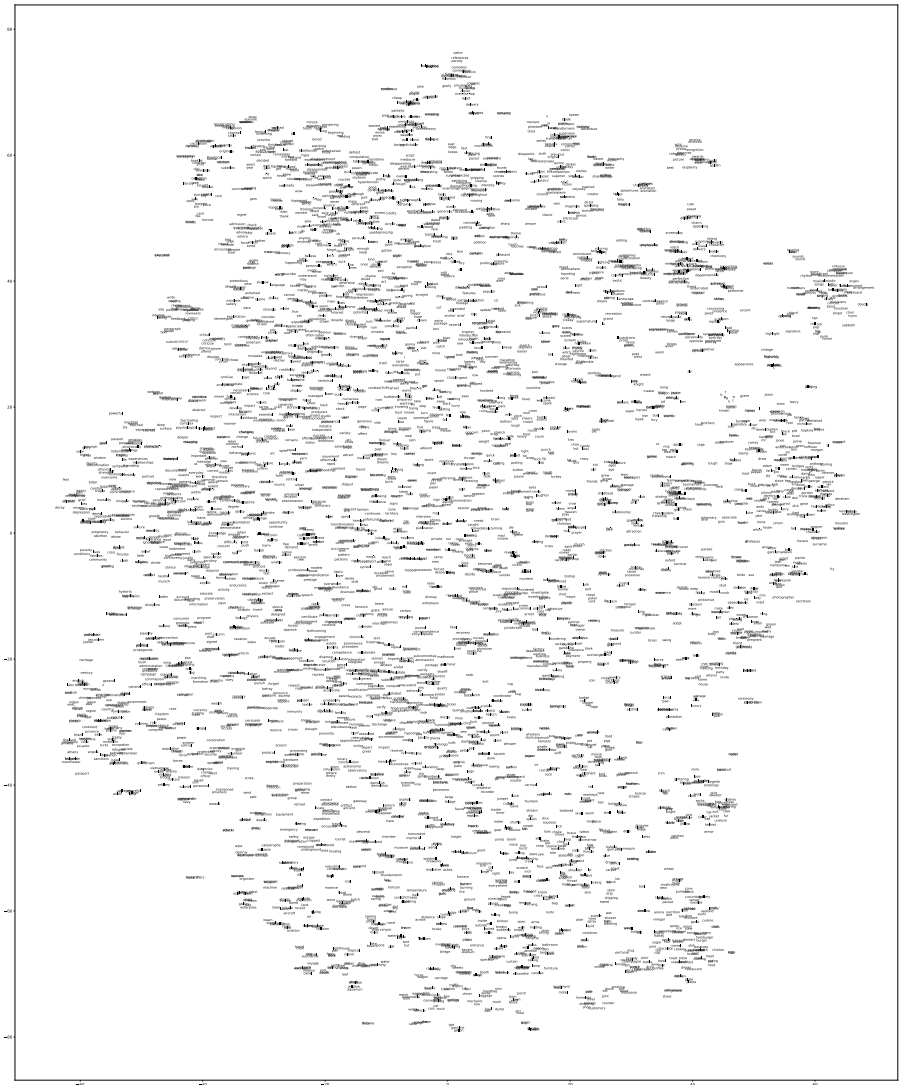


cooking

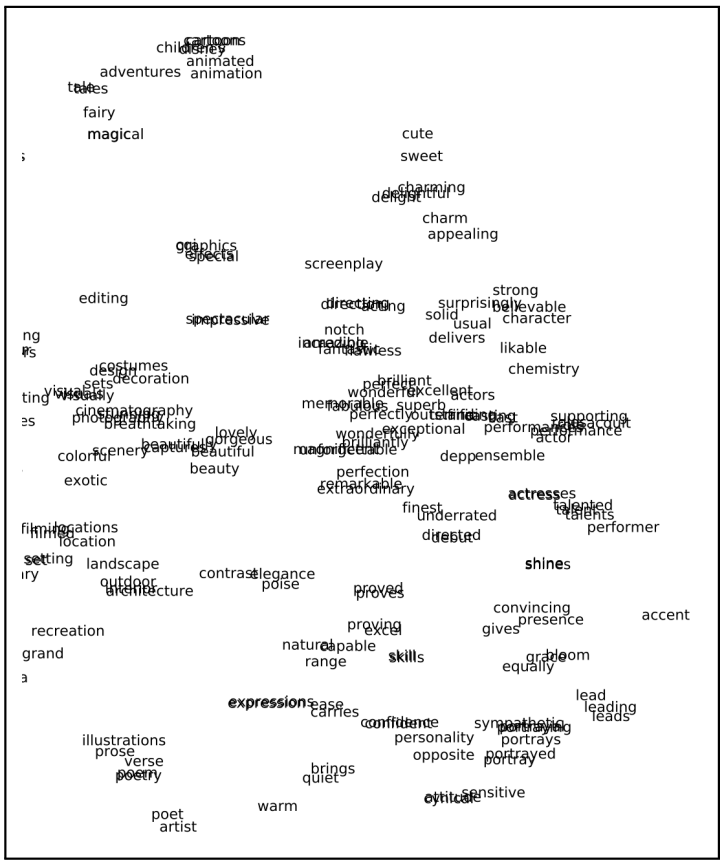


conflict

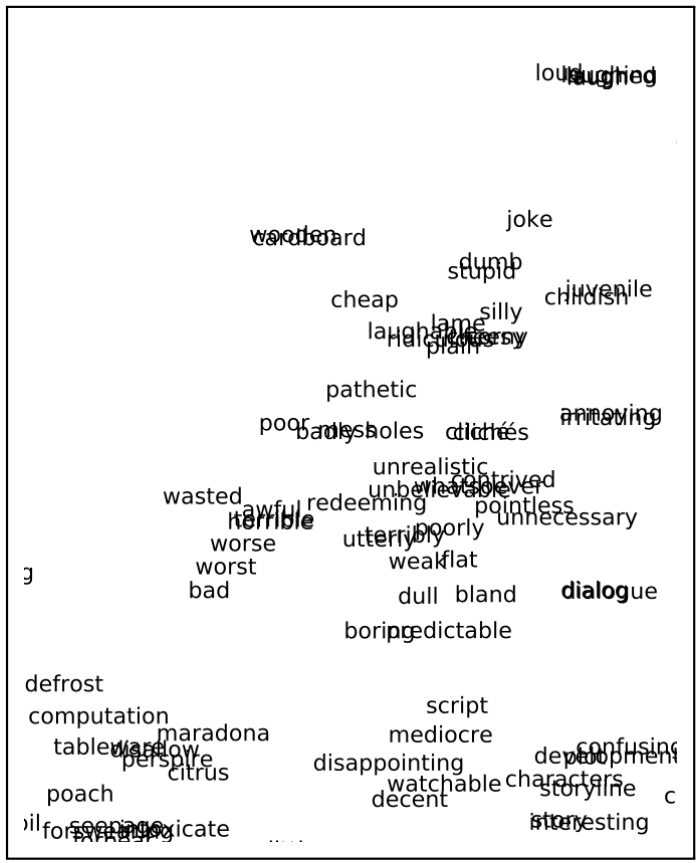
t-SNE on the yelp20 PPMI VSM



t-SNE on the yelp20 PPMI VSM



positivity



negativity

Code snippets

[1]:
```
from nltk.corpus import opinion_lexicon
import os
import pandas as pd
import vsm
```

[2]:
```
DATA_HOME = os.path.join('data', 'vsmdata')

yelp5 = pd.read_csv(
    os.path.join(DATA_HOME, 'yelp_window5-scaled.csv.gz'), index_col=0)
```

[3]:
```
yelp5_ppmi = vsm.pmi(yelp5)
```

[4]:
```
# Supply a str filename to write the output to a file:
vsm.tsne_viz(yelp5_ppmi, output_filename=None)
```

[5]:
```
# To display words in different colors based on external criteria:
positive = set(opinion_lexicon.positive())
negative = set(opinion_lexicon.negative())

colors = []
for w in yelp5_ppmi.index:
    if w in positive:
        color = 'red'
    elif w in negative:
        color = 'blue'
    else:
        color = 'gray'
    colors.append(color)

vsm.tsne_viz(yelp5_ppmi, colors=colors)
```

References I

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Thomas Hofmann. 1999. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57, New York. ACM.

Laurens van der Maaten and Geoffrey E. Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

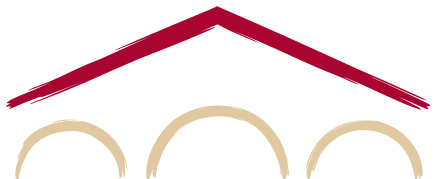
Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Distributed word representations: Retrofitting

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding



Central goals

- Distributional representations are powerful and easy to obtain, but they tend to reflect only similarity (synonymy, connotation).
- Structured resources are sparse and hard to obtain, but they support learning rich, diverse semantic distinctions.
- Can we have the best aspects of both? Retrofitting is one way of saying, “Yes”.
- Retrofitting is due to Faruqui et al. (2015).

The retrofitting model

$$\sum_{i \in \mathbf{V}} \alpha_i \left\| \boldsymbol{q}_i - \hat{\boldsymbol{q}}_i \right\|^2 + \sum_{(i,j,r) \in \mathbf{E}} \beta_{ij} \left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|^2$$

- Balances fidelity to the original vector $\hat{\boldsymbol{q}}_i$
- against looking more like one's graph neighbors.
- Forces are balanced with $\alpha = 1$ and $\beta = \frac{1}{\textbf{Degree}(i)}$

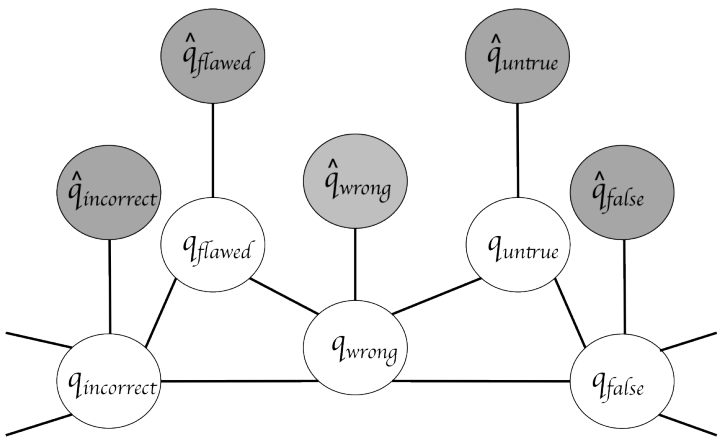
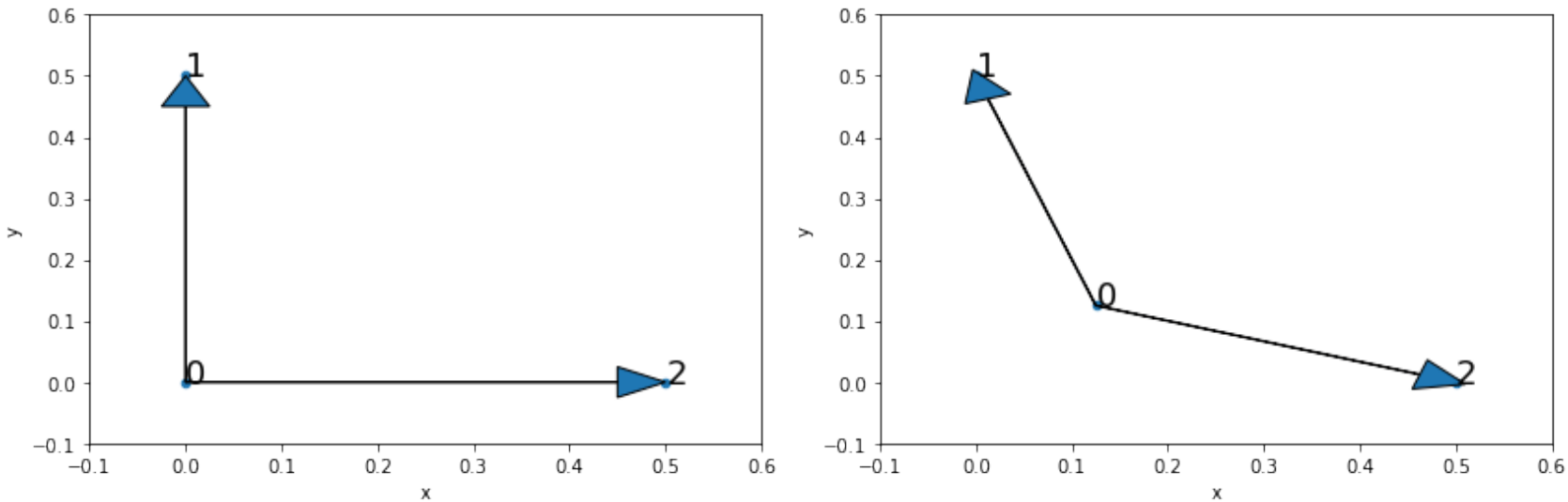


Figure 1: Word graph with edges between related words showing the observed (grey) and the inferred (white) word vector representations.

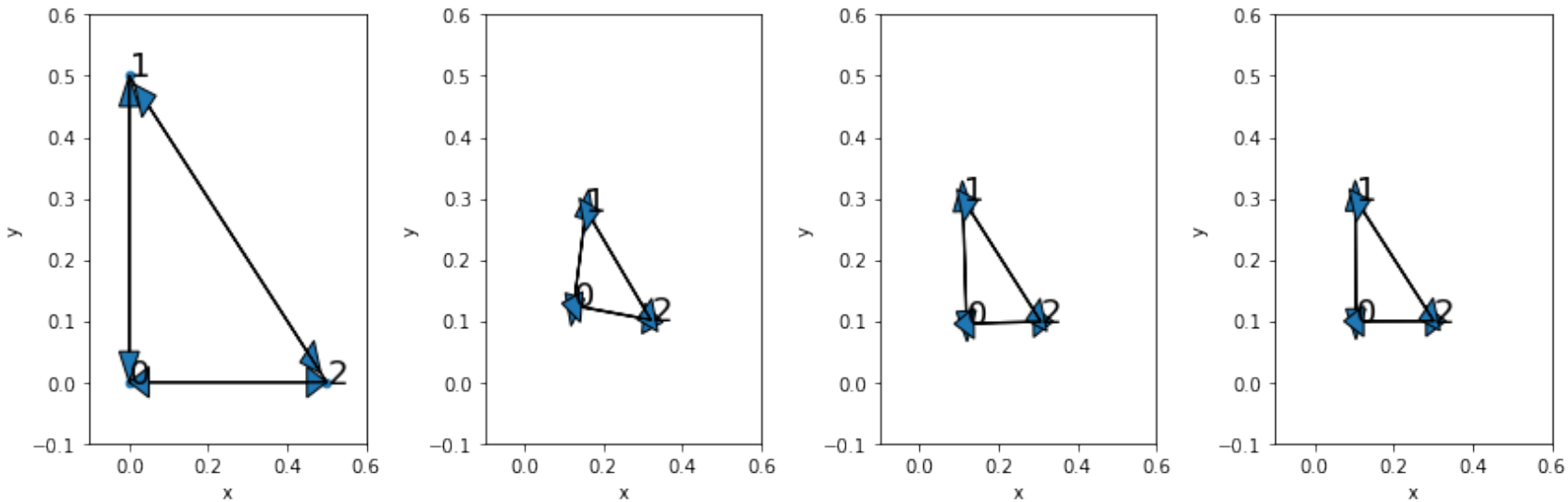
Simple retrofitting examples

$$\sum_{i \in \mathbf{V}} \alpha_i \left\| \boldsymbol{q}_i - \hat{\boldsymbol{q}}_i \right\|^2 + \sum_{(i,j,r) \in \mathbf{E}} \beta_{ij} \left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|^2$$



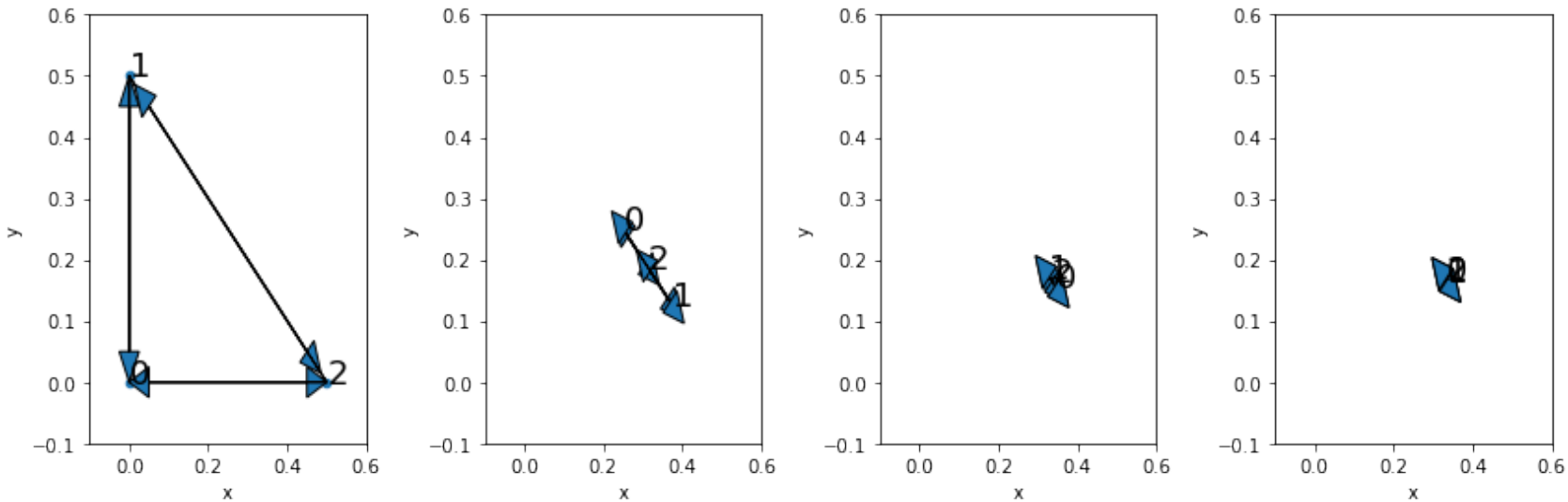
Simple retrofitting examples

$$\sum_{i \in \mathbf{V}} \alpha_i \left\| \boldsymbol{q}_i - \hat{\boldsymbol{q}}_i \right\|^2 + \sum_{(i,j,r) \in \mathbf{E}} \beta_{ij} \left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|^2$$



Simple retrofitting examples

$$\sum_{i \in \mathbf{V}} \alpha_i \left\| \boldsymbol{q}_i - \hat{\boldsymbol{q}}_i \right\|^2 + \sum_{(i,j,r) \in \mathbf{E}} \beta_{ij} \left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|^2$$



$\alpha = 0$

Extensions

Drop the assumption that every edge means ‘similar’:

- Mrkšić et al. (2016) AntonymRepel, SynonymAttract, and VectorSpacePreservation for different edge types.
- Lengerich et al. (2018): functional retrofitting to learn the semantics of any edge types.
- This work is closely related to **graph embedding** (learning distributed representations for nodes), for which see Hamilton et al. 2017.

Code snippets

[1]:
```
import pandas as pd
from retrofitting import Retrofitter
```

[2]:
```
Q_hat = pd.DataFrame(
    [[0.0, 0.0],
     [0.0, 0.5],
     [0.5, 0.0]],
    columns=['x', 'y'])

edges = {0: {1, 2}, 1: set(), 2: set()}
```

[3]:
```
Q_hat
```

[3]:

	x	y
0	0.0	0.0
1	0.0	0.5
2	0.5	0.0

[4]:
```
retro = Retrofitter(verbose=True)
```

[5]:
```
X_retro = retro.fit(Q_hat, edges)
```

```
Converged at iteration 2; change was 0.0000
```

[6]:
```
X_retro
```

[6]:

	x	y
0	0.125	0.125
1	0.000	0.500
2	0.500	0.000

[7]:
```
# For an application to WordNet, see `vsm_03_retrofitting`.
```

References I

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615, Stroudsburg, PA. Association for Computational Linguistics.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. In *IEEE Data Engineering Bulletin*, pages 52–74. IEEE Press.

Benjamin J. Lengerich, Andrew L. Maas, and Christopher Potts. 2018. Retrofitting distributional embeddings to knowledge graphs with functional relations. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2423–2436, Stroudsburg, PA. Association for Computational Linguistics.

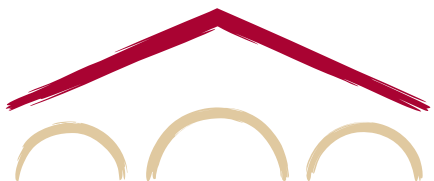
Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–148. Association for Computational Linguistics.

Distributed word representations: Static representations from contextual models

Christopher Potts

Stanford Linguistics

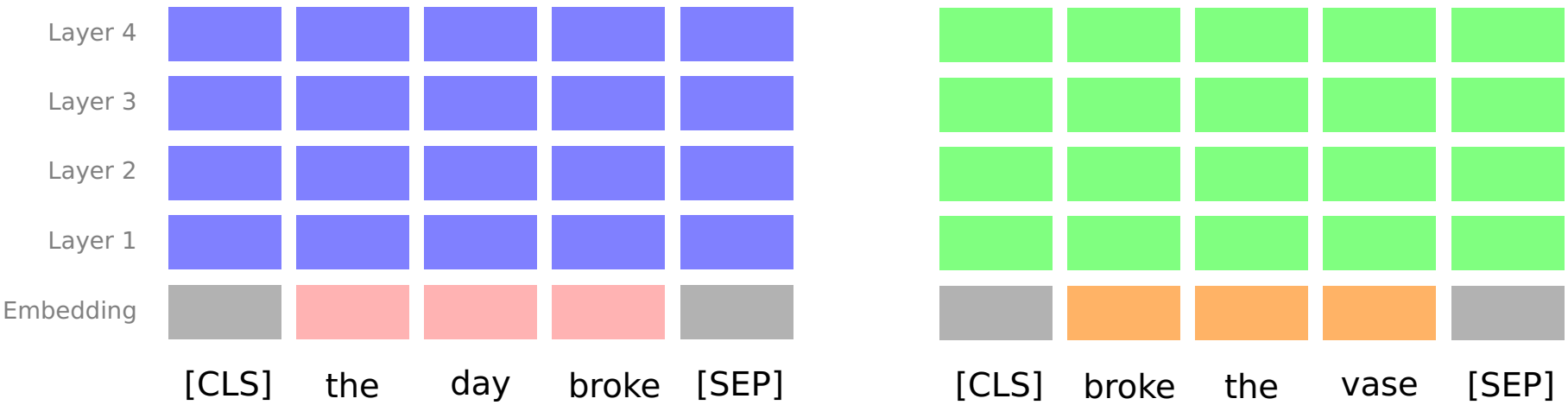
CS224u: Natural language understanding



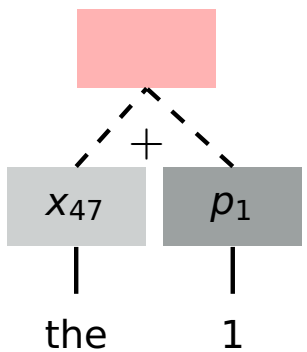
Overview

1. How can I use BERT (RoBERTa, XLNet, ELECTRA, . . .)?
2. Tension: we've been developing *static* representations, whereas BERT delivers *contextual* representations.
3. Are there good methods for deriving static representations from contextual ones?
4. Yes! Bommasani et al. (2020)!
5. This lecture:
   - Hands-on, high-level overview of these models. (We will analyze them in detail later in the quarter.)
   - Overview of the methods of Bommasani et al.

The structure of BERT



- The rectangles are vectors: the outputs of each layer of the network.
- Different sequences deliver different vectors for the same token, even in the embedding layer if the positions vary.



Tokenization

```
[1]: from transformers import BertTokenizer

[2]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

[3]: tokenizer.tokenize("This isn't too surprising.")

[3]: ['This', 'isn', "'", 't', 'too', 'surprising', '.']

[4]: tokenizer.tokenize("Encode me!")

[4]: ['En', '##code', 'me', '!']

[5]: tokenizer.tokenize("Snuffleupagus?")

[5]: ['S', '##nu', '##ffle', '##up', '##agu', '##s', '?']

[6]: tokenizer.vocab_size

[6]: 28996
```

Basic Hugging Face interfaces

[1]:
```
import torch
from transformers import BertModel, BertTokenizer
```

[2]:
```
bert_weights_name = 'bert-base-cased'
```

[3]:
```
tokenizer = BertTokenizer.from_pretrained(bert_weights_name)
```

[4]:
```
model = BertModel.from_pretrained(bert_weights_name)
```

[5]:
```
ex = tokenizer.encode(
    "the day broke",
    add_special_tokens=True,
    return_tensors='pt')
ex
```

[5]:
```
tensor([[ 101, 1103, 1285, 2795,  102]])
```

[6]:
```
with torch.no_grad():
    reps = model(ex, output_hidden_states=True)
```

[7]:
```
# Embedding and then 12 layers:
len(reps.hidden_states)
```

[7]:
```
13
```

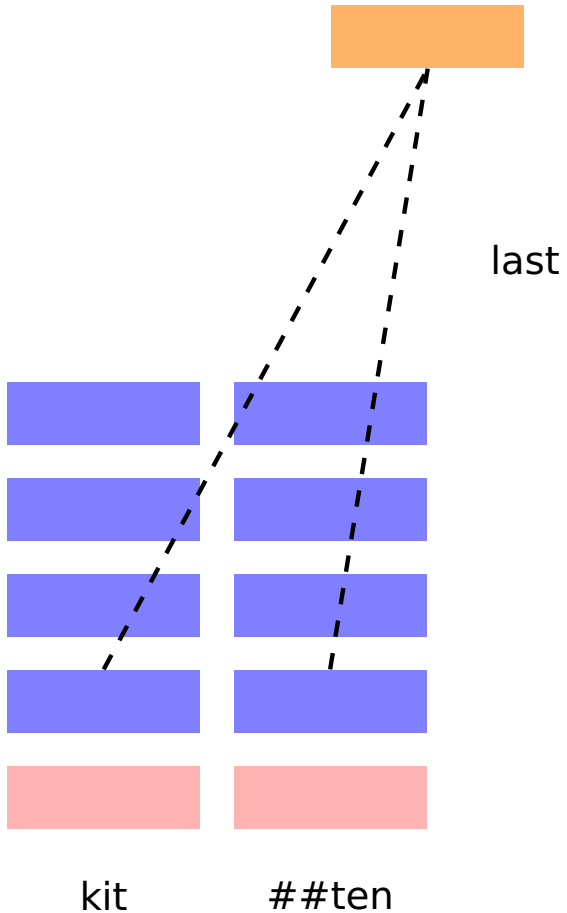
[8]:
```
# Embedding: batch of 1 example, 5 tokens, each represented
# by a vector of dimension 768:
reps.hidden_states[0].shape
```

[8]:
```
torch.Size([1, 5, 768])
```

[9]:
```
# Final output layer:
reps.hidden_states[-1].shape
```

[9]:
```
torch.Size([1, 5, 768])
```

The Decontextualized approach



Very simple! Potentially unnatural, though.

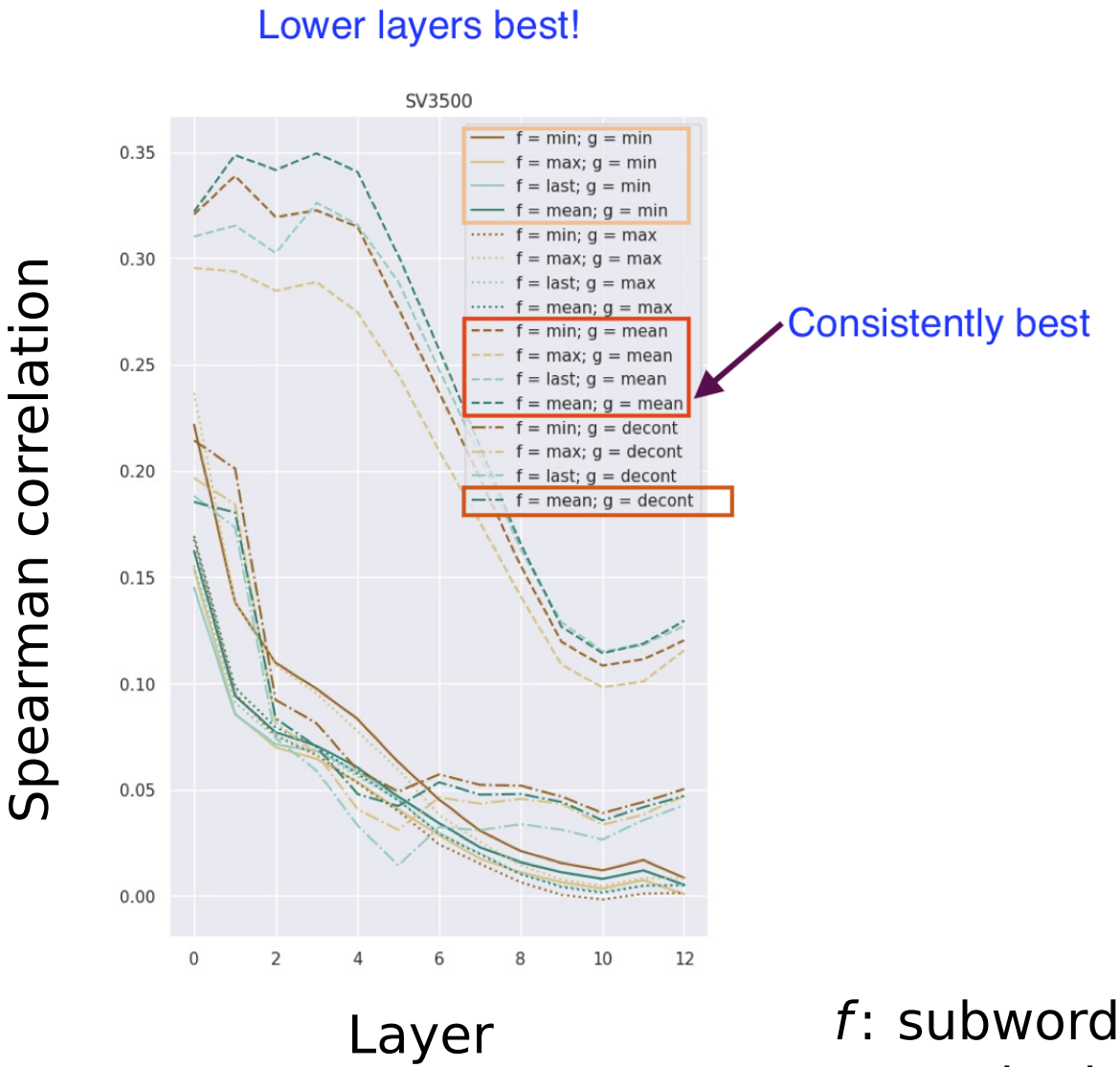
The Aggregated approach

Process *lots* of corpus examples containing the target word:

1. The kit ##ten yawned.
2. Where is my kit ##ten ?
3. A kit ##ten is a young cat.
4. The puppy and the kit ##ten are playing.
5. . . .

Pool sub-word tokens and pool the different contextuals reps.

A few results from Bommasani et al. (2020)



f: subword pooling
g: context pooling

References I

Rishi Bommasani, Kelly Davis, and Claire Cardie. 2020. Interpreting Pretrained Contextualized Representations via Reductions to Static Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4758–4781, Online. Association for Computational Linguistics.