Distributed word representations

Christopher Potts

Stanford Linguistics

CS 224U: Natural language understanding
April 3–10

Plan

1. High-level goals and guiding hypotheses
2. Matrix designs
3. Vector comparison
4. Basic reweighting
5. Subword information
6. Visualization
7. Dimensionality reduction
8. Retrofitting

Meaning latent in co-occurrence patterns

	against	age	agent	ages	ago	agree	ahead	ain’t	air	aka	al
against	2003	90	39	20	88	57	33	15	58	22	24
age	90	1492	14	39	71	38	12	4	18	4	39
agent	39	14	507	2	21	5	10	3	9	8	25
ages	20	39	2	290	32	5	4	3	6	1	6
ago	88	71	21	32	1164	37	25	11	34	11	38
agree	57	38	5	5	37	627	12	2	16	19	14
ahead	33	12	10	4	25	12	429	4	12	10	7
ain’t	15	4	3	3	11	2	4	166	0	3	3
air	58	18	9	6	34	16	12	0	746	5	11
aka	22	4	8	1	11	19	10	3	5	261	9
al	24	39	25	6	38	14	7	3	11	9	861

Meaning latent in co-occurrence patterns

Class	Word
0	awful
0	terrible
0	lame
0	worst
0	disappointing
1	nice
1	amazing
1	wonderful
1	good
1	awesome

$\Pr(\text{Class} = 1)$	Word
?	w_1
?	w_2
?	w_3
?	w_4

A hopeless learning scenario

Meaning latent in co-occurrence patterns

Class	Word	excellent	terrible
0	awful	−0.69	1.13
0	terrible	−0.13	3.09
0	lame	−1.00	0.69
0	worst	−0.94	1.04
0	disappointing	0.19	0.09
1	nice	0.08	−0.07
1	amazing	0.71	−0.06
1	wonderful	0.66	−0.76
1	good	0.21	0.11
1	awesome	0.67	0.26

Pr(Class=1)	Word	excellent	terrible
≈ 0	w_1	−0.47	0.82
≈ 0	w_2	−0.55	0.84
≈ 1	w_3	0.49	−0.13
≈ 1	w_4	0.41	−0.11

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).
3. A foundation for deep learning NLU models.
4. In your assignment and projects, you're likely to use representations like these:
   - to understand and model linguistic and social phenomena; and/or
   - as inputs to other machine learning models.

Associated materials

1. Code
   a. `vsm.py`
   b. `vsm_01_distributional.ipynb`
   c. `vsm_02_dimreduce.ipynb`
   d. `vsm_03_retrofitting.ipynb`
2. Homework 1 and bake-off 1: `hw1_wordsim.ipynb`
3. Screencasts:
   a. Overview [link]
   b. Vector comparison [link]
   c. Reweighting [link]
   d. Dimensionality reduction [link]
4. Core readings: Turney & Pantel 2010; Smith 2019; Pennington et al. 2014; Faruqui et al. 2015

Guiding hypotheses

Firth (1957)

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

Firth (1957)

"the complete meaning of a word is always contextual, and no study of meaning apart from context can be taken seriously."

Wittgenstein (1953)

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

Harris (1954)

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

Turney & 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

Great power, a great many design choices

Matrix design
word × document
word × word
word × search proximity
adj. × modified noun
word × dependency rel.
⋮

Great power, a great many design choices

tokenization
annotation
tagging
parsing
feature selection
$\vdots$ cluster texts by date/author/discourse context/...
$\Downarrow$ $\swarrow$

Matrix design
word × document
word × word
word × search proximity
adj. × modified noun
word × dependency rel.
$\vdots$

Great power, a great many design choices

tokenization
annotation
tagging
parsing
feature selection

$\vdots$ cluster texts by date/author/discourse context/...

$\Downarrow$ $\Swarrow$

Matrix design	Reweighting
word × document	probabilities
word × word	length norm.
word × search proximity	TF-IDF
adj. × modified noun	PMI
word × dependency rel.	Positive PMI
$\vdots$	$\vdots$

Great power, a great many design choices

tokenization
annotation
tagging
parsing
feature selection

$\vdots$ cluster texts by date/author/discourse context/...

$\Downarrow$ $\Swarrow$

Matrix design	Reweighting	Dimensionality reduction
word × document	probabilities	LSA
word × word	length norm.	PLSA
word × search proximity	TF-IDF	LDA
adj. × modified noun	PMI	PCA
word × dependency rel.	Positive PMI	NNMF
$\vdots$	$\vdots$	$\vdots$

Great power, a great many design choices

tokenization
annotation
tagging
parsing
feature selection
$\vdots$ cluster texts by date/author/discourse context/...
$\Downarrow$ $\swarrow$

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
$\vdots$	$\vdots$	$\vdots$	$\vdots$

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.)

Designs

word x word

	against	age	agent	ages	ago	agree	ahead	ain't	air	aka	al
against	2003	90	39	20	88	57	33	15	58	22	24
age	90	1492	14	39	71	38	12	4	18	4	39
agent	39	14	507	2	21	5	10	3	9	8	25
ages	20	39	2	290	32	5	4	3	6	1	6
ago	88	71	21	32	1164	37	25	11	34	11	38
agree	57	38	5	5	37	627	12	2	16	19	14
ahead	33	12	10	4	25	12	429	4	12	10	7
ain't	15	4	3	3	11	2	4	166	0	3	3
air	58	18	9	6	34	16	12	0	746	5	11
aka	22	4	8	1	11	19	10	3	5	261	9
al	24	39	25	6	38	14	7	3	11	9	861

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:

	%	+	ˆ2	ˆg	ˆh	ˆq	aa
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

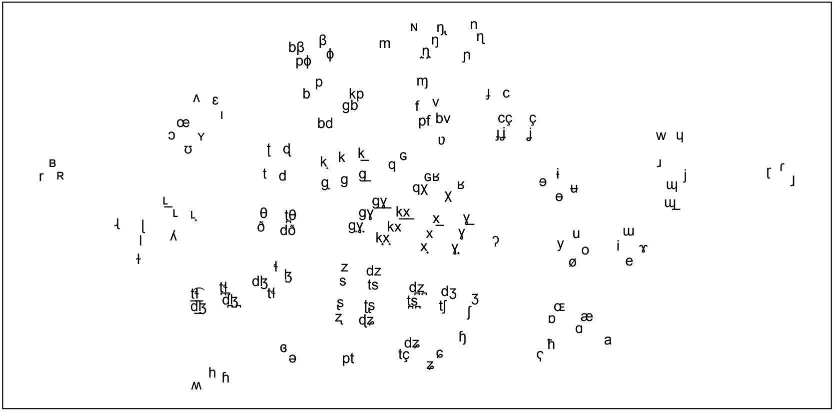
phonological segment × feature values

Derived from http://www.linguistics.ucla.edu/people/hayes/120a/.
Dimensions: (141 × 28).

	syllabic	stress	long	consonantal	sonorant	continuant	delayed.release	approximant	tap	trill	…
ɒ	1	−1	−1	−1	1	1	0	1	−1	−1	
ɑ	1	−1	−1	−1	1	1	0	1	−1	−1	
ɶ	1	−1	−1	−1	1	1	0	1	−1	−1	
a	1	−1	−1	−1	1	1	0	1	−1	−1	
æ	1	−1	−1	−1	1	1	0	1	−1	−1	…
ʌ	1	−1	−1	−1	1	1	0	1	−1	−1	
ɔ	1	−1	−1	−1	1	1	0	1	−1	−1	
o	1	−1	−1	−1	1	1	0	1	−1	−1	
ɤ	1	−1	−1	−1	1	1	0	1	−1	−1	
ə	1	−1	−1	−1	1	1	0	1	−1	−1	
⋮					⋮						

phonological segment × feature values

Derived from http://www.linguistics.ucla.edu/people/hayes/120a/.
Dimensions: (141 × 28).



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.

Other designs

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

  ⋮

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
Window: 3	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
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
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
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

- 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

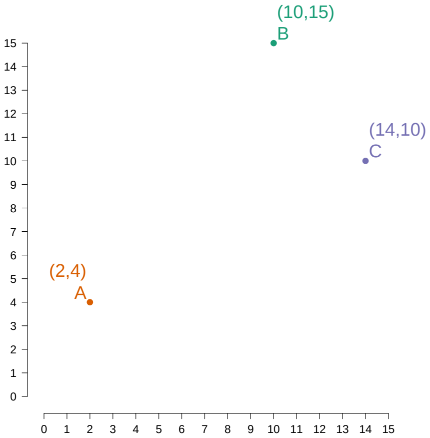
```
import os
import pandas as pd

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

# IMDB: Window size = 5; scaling = 1/n
imdb5 = pd.read_csv(
    os.path.join(DATA_HOME, 'imdb_window5-scaled.csv.gz'), index_col=0)

# IMDB: Window size = 20; scaling = flat
imdb20 = pd.read_csv(
    os.path.join(DATA_HOME, 'imdb_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)
```

Vector comparison

1. High-level goals and guiding hypotheses
2. Matrix designs
3. Vector comparison
4. Basic reweighting
5. Subword information
6. Visualization
7. Dimensionality reduction
8. Retrofitting

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:

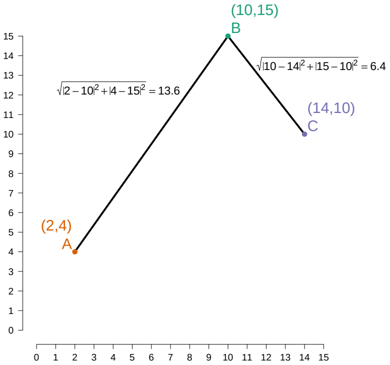
$$\textbf{euclidean}(u, v) = \sqrt{\sum_{i=1}^{n} |u_i - v_i|^2}$$

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

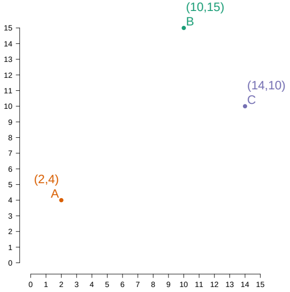
$$\left[\frac{u_1}{\|u\|_2}, \frac{u_2}{\|u\|_2}, \cdots, \frac{u_n}{\|u\|_2}\right]$$

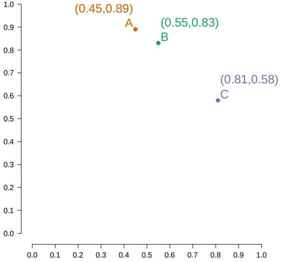
Length normalization

	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:

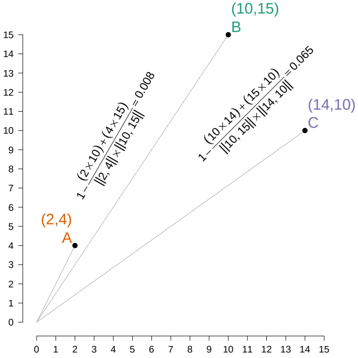
$$\textbf{cosine}(u, v) = 1 - \frac{\sum_{i=1}^{n} u_i \times v_i}{\|u\|_2 \times \|v\|_2}$$

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

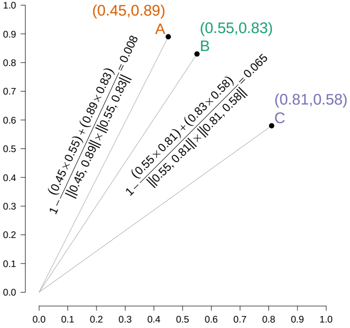


Cosine distance

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

Between probability distributions p and q:

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

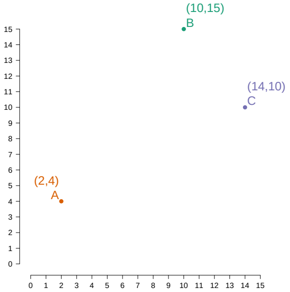
p is the reference distribution. Before calculation, smooth by adding ϵ.

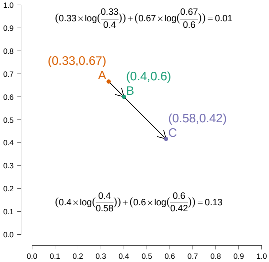
KL divergence

	d_x	d_y
A	2	4
B	10	15
C	14	10

Normalize the rows $\Rightarrow$

	d_x	d_y
A	0.33	0.67
B	0.40	0.60
C	0.58	0.42





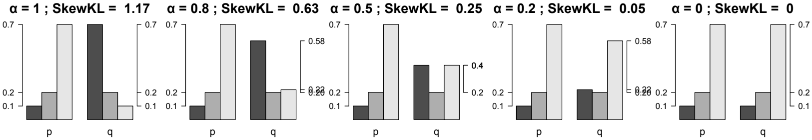
KL variants

Symmetric KL

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

KL-divergence with skew

$$D(p \parallel \alpha q + (1-\alpha)p) \qquad 0 \leq \alpha \leq 1$$



Jensen–Shannon distance

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

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 (Manning & Schütze 1999:301).

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.

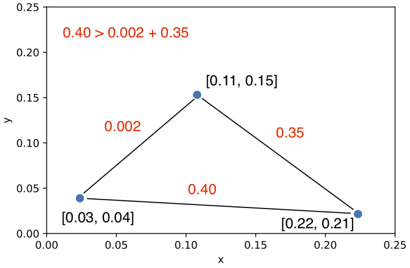
5. Skew is KL but with a preliminary step that gives special credence to the reference distribution.

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 satify this:



Distance metric?

Yes: Euclidean, 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, KL with skew

Code snippets

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

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

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

Out[3]: 13.601470508735444

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

Out[4]: 4.47213595499958

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

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

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

Out[6]: 0.007722123286332261

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

Out[7]: 6.0

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

Out[8]: 0.76
```

Code snippets

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

        imdb5 = pd.read_csv(
            os.path.join(DATA_HOME, 'imdb_window5-scaled.csv.gz'), index_col=0)

In [10]: vsm.cosine(imdb5.loc['good'], imdb5.loc['excellent'])

Out[10]: 0.9644382411451131

In [11]: vsm.cosine(imdb5.loc['good'], imdb5.loc['bad'])

Out[11]: 0.9480014759326252

In [12]: vsm.neighbors('bad', imdb5).head()

Out[12]: bad      0.000000
         guys     0.823744
         .        0.844851
         taste    0.893747
         guy      0.896312
         dtype: float64

In [13]: vsm.neighbors('bad', imdb5, distfunc=vsm.jaccard).head(3)

Out[13]: bad       0.000000
         think     0.783744
         better    0.788782
         dtype: float64
```

Basic reweighting

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?
  - How does it compare to the word frequencies?
  - 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

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

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

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

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**

$\overset{\textbf{oe}}{\Rightarrow}$

	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 $\log(0) = 0$):

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

Pointwise Mutual Information (PMI)

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

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

	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

$\Rightarrow$

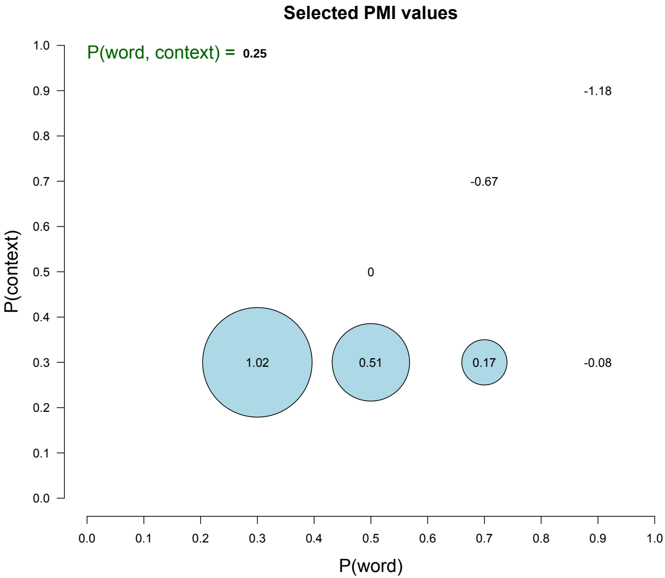
	$P(w,d)$				$P(w)$
A	0.11	0.11	0.11	0.11	0.44
B	0.11	0.11	0.11	0.00	0.33
C	0.11	0.11	0.00	0.00	0.22
D	0.00	0.00	0.00	0.01	0.01
$P(d)$	0.33	0.33	0.22	0.12	

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

Selected PMI values



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 & Goldberg 2014):

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

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 & Goldberg 2014):

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

PPMI

$$\mathbf{ppmi}(X, i, j) = \max(0, \mathbf{pmi}(X, i, j))$$

TF-IDF

For a corpus of documents D:

- Term frequency (TF): $P(w|d)$
- Inverse document frequency (IDF): $\log\left(\frac{|D|}{\left|\{d \in D : w \in d\}\right|}\right)$ $(\log(0) = 0)$
- TF-IDF: TF × IDF

TF-IDF

For a corpus of documents D:

- Term frequency (TF): $P(w|d)$
- Inverse document frequency (IDF): $\log\left(\frac{|D|}{\left|\{d \in D : w \in d\}\right|}\right)$ $(\log(0) = 0)$
- TF-IDF: TF × IDF

	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

$\Rightarrow$

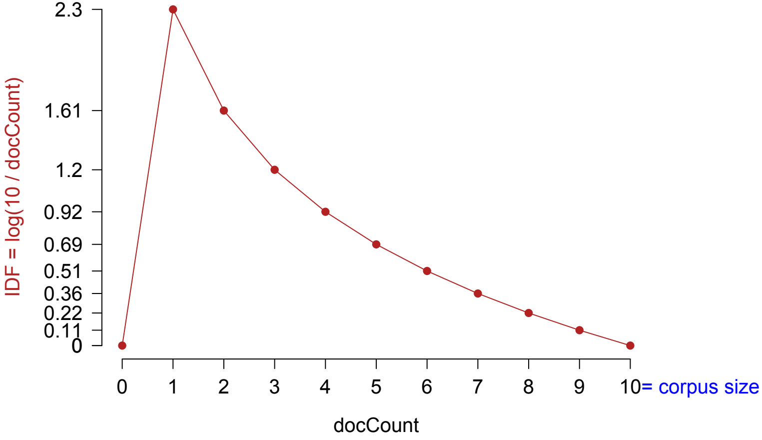
	IDF
A	0.00
B	0.29
C	0.69
D	1.39

$\Downarrow$

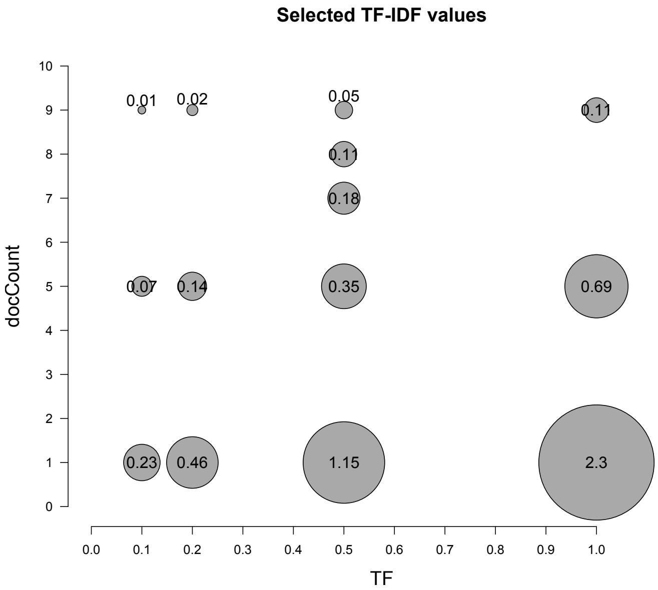
	TF			
	d_1	d_2	d_3	d_4
A	0.33	0.33	0.50	0.91
B	0.33	0.33	0.50	0.00
C	0.33	0.33	0.00	0.00
D	0.00	0.00	0.00	0.09

	TF-IDF			
	d_1	d_2	d_3	d_4
A	0.00	0.00	0.00	0.00
B	0.10	0.10	0.14	0.00
C	0.23	0.23	0.00	0.00
D	0.00	0.00	0.00	0.13

IDF values


2.3
1.61
1.2
0.92
0.69
0.51
0.36
0.22
0.11
0
IDF = log(10 / docCount)
0
1
2
3
4
5
6
7
8
9
10= corpus size
docCount

Selected TF-IDF values



Other weighting/normalization schemes

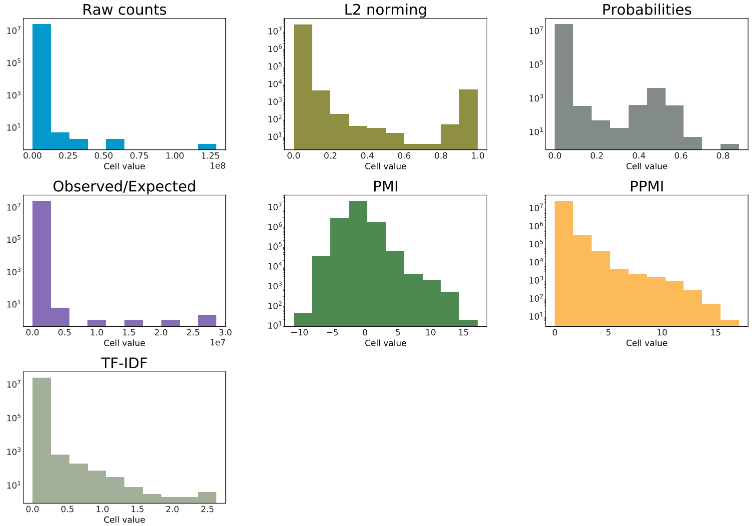
- t-test: $\frac{P(w,d)-P(w)P(d)}{\sqrt{P(w)P(d)}}$
- TF-IDF variants that seek to be sensitive to the empirical distribution of words (For discussion and references, Manning & Schütze 1999:553.)
- Pairwise distance matrices:

	d_x	d_y
A	2	4
B	10	15
C	14	10

$\overset{\text{cosine}}{\Rightarrow}$

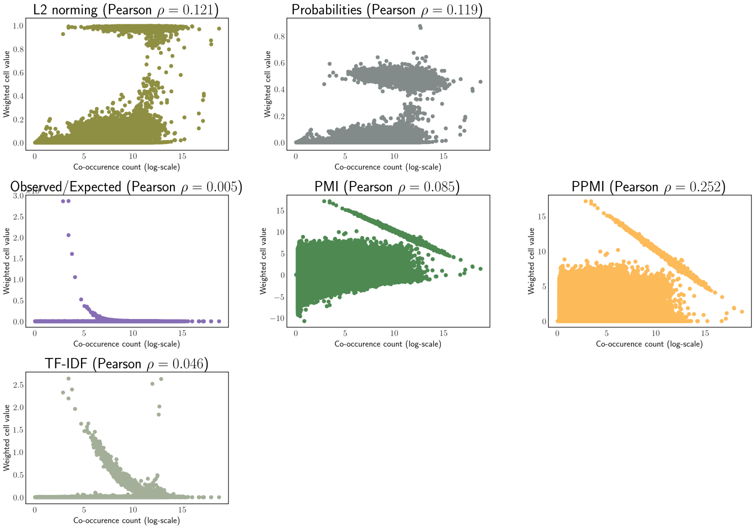
	A	B	C
A	0	0.008	0.116
B	0.008	0	0.065
C	0.116	0.065	0

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}.
- Many weighting schemes end up favoring rare events that may not be trustworthy.
- 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 are often 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

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

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

        imdb5 = pd.read_csv(
            os.path.join(DATA_HOME, 'imdb_window5-scaled.csv.gz'), index_col=0)

        imdb5_oe = vsm.observed_over_expected(imdb5)

        imdb5_norm = imdb5.apply(vsm.length_norm, axis=1)

        imdb5_ppmi = vsm.pmi(imdb5)

        imdb5_pmi = vsm.pmi(imdb5, positive=False)

        imdb5_tfidf = vsm.tfidf(imdb5)
```

Code snippets

```
In [2]: vsm.neighbors('bad', imdb5).head()

Out[2]: bad      0.000000
        guys     0.823744
        .        0.844851
        taste    0.893747
        guy      0.896312
        dtype: float64

In [3]: vsm.neighbors('bad', imdb5_ppmi).head()

Out[3]: bad         0.000000
        good        0.701241
        awful       0.757309
        terrible    0.763324
        horrible    0.763637
        dtype: float64
```

Subword information

Motivation

1. Schütze (1993) pioneered subword modeling to improve representations by reducing sparsity, thereby increasing the density of connections in a VSM.

2. Subword modeling will also
   a. Pull morphological variants closer together
   b. Facilitate modeling out-of-vocabulary items
   c. Reduce the importance of any particular tokenization scheme

Technique

Bojanowski et al. (2016) (the fastText team) motivate a straightforward approach:

1. Given a word-level VSM, the vector for a character-level *n*-gram *x* is the sum of all the vectors of words containing *x*.
2. Represent each word *w* as the sum of its character-level *n*-grams.
3. Add in the representation of *w* if available

A linguistically richer variant might use sequences of morphemes rather than characters.

Example with 4-grams

superbly becomes
[<w>sup, supe, uper, perb, erbl, rbly, bly</w>]

Code snippets

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

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

        imdb5 = pd.read_csv(
            os.path.join(DATA_HOME, 'imdb_window5-scaled.csv.gz'), index_col=0)

In [2]: imdb5_ngrams = vsm.ngram_vsm(imdb5, n=4)

In [3]: imdb5_ngrams.loc['<w>sup'].values

Out[3]: array([3.41545000e+03, 3.70000000e+01, 4.95458333e+04, ...,
               2.23950000e+02, 4.64833333e+01, 3.12166667e+01])

In [4]: imdb5_ngrams.shape

Out[4]: (9806, 5000)

In [5]: vsm.get_character_ngrams("superbly", n=4)

Out[5]: ['<w>sup', 'supe', 'uper', 'perb', 'erbl', 'rbly', 'bly</w>']

In [6]: def character_level_rep(word, cf, n=4):
            ngrams = vsm.get_character_ngrams(word, n)
            ngrams = [n for n in ngrams if n in cf.index]
            reps = cf.loc[ngrams].values
            return reps.sum(axis=0)

In [7]: superbly = character_level_rep("superbly", imdb5_ngrams)

In [8]: superbly.shape

Out[8]: (5000,)
```

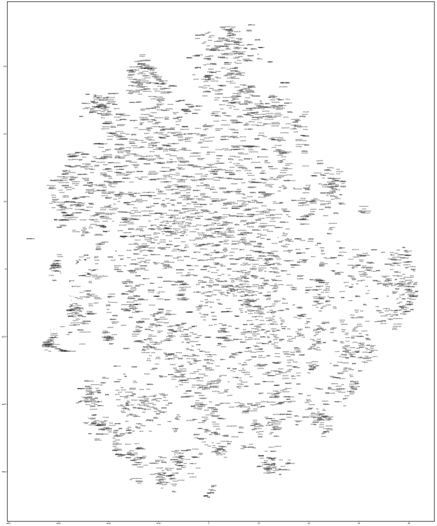
Visualization

1. High-level goals and guiding hypotheses
2. Matrix designs
3. Vector comparison
4. Basic reweighting
5. Subword information
6. Visualization
7. Dimensionality reduction
8. Retrofitting

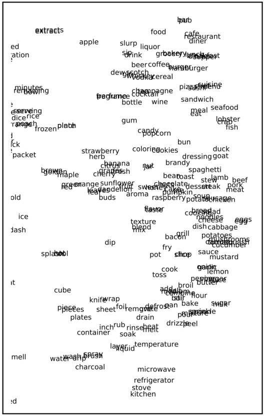
Techniques

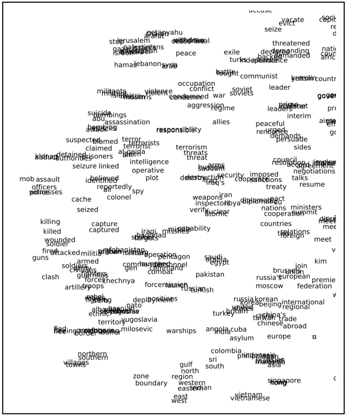
- Our goal is to visualize very high-dimensional spaces in two or three dimensions. **This will inevitably involve compromises.**
- Still, visualization can give you a feel for what is in your VSM, especially if you pair it with other kinds of qualitative exploration (e.g., using `vsm.neighbors`).
- 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



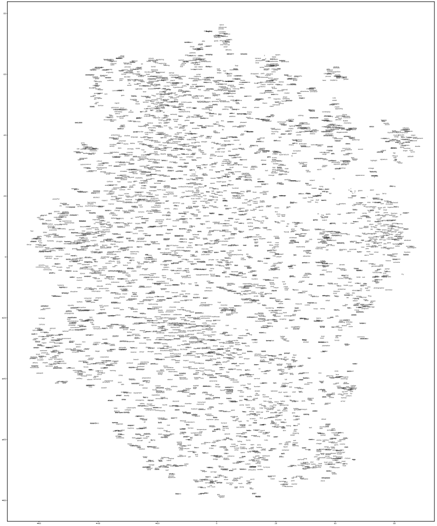
t-SNE on the giga20 PPMI VSM



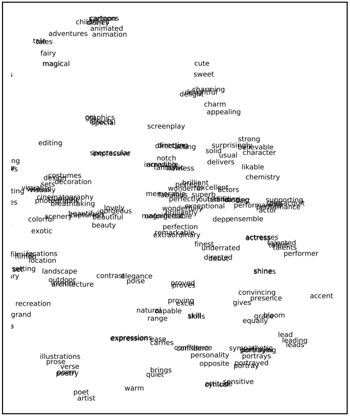


cooking **conflict**

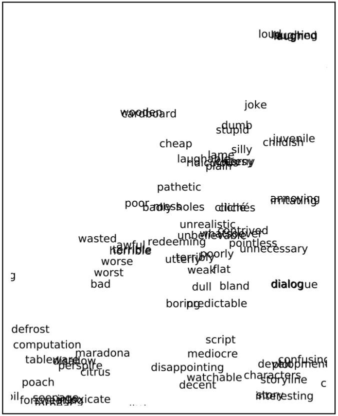
t-SNE on the imdb20 PPMI VSM



t-SNE on the imdb20 PPMI VSM



positivity



negativity

Code snippets

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

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

        imdb5 = pd.read_csv(
            os.path.join(DATA_HOME, 'imdb_window5-scaled.csv.gz'), index_col=0)

In [3]: imdb5_ppmi = vsm.pmi(imdb5)

In [4]: # Supply a str filename to write the output to a file:
        vsm.tsne_viz(imdb5_ppmi, output_filename=None)

In [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 imdb5_ppmi.index:
            if w in positive:
                color = 'red'
            elif w in negative:
                color = 'blue'
            else:
                color = 'gray'
            colors.append(color)

        vsm.tsne_viz(imdb5_ppmi, colors=colors)
```

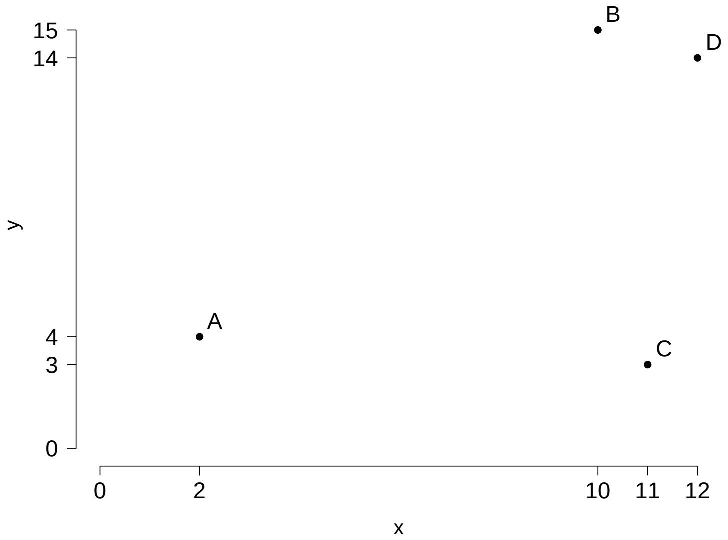
Dimensionality reduction

1. High-level goals and guiding hypotheses
2. Matrix designs
3. Vector comparison
4. Basic reweighting
5. Subword information
6. Visualization
7. Dimensionality reduction
8. Retrofitting

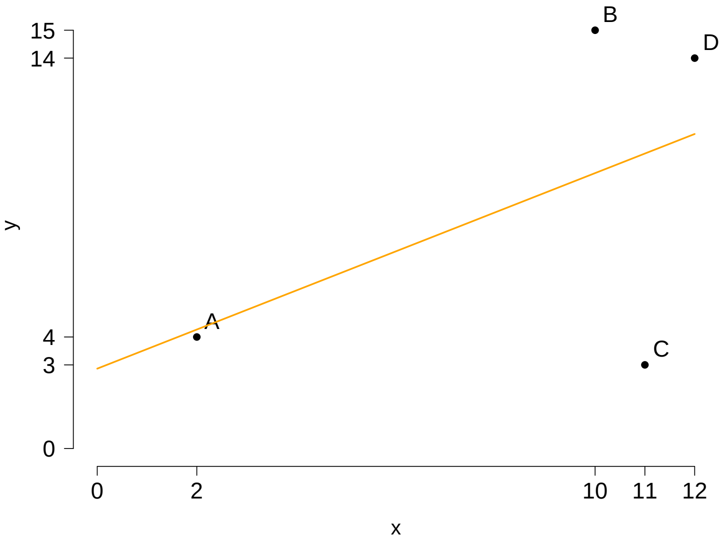
Latent Semantic Analysis (LSA)

- 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.

LSA: Guiding intuitions



LSA: Guiding intuitions



LSA: The 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

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

LSA: The 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

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

$$\underset{A_{3\times 4}}{\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}} = \underset{T_{3\times 3}}{\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}} \underset{S_{3\times 3}}{\begin{pmatrix} \cdot & & \\ & \cdot & \\ & & \cdot \end{pmatrix}} \underset{D^T_{4\times 3}}{\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}^T}$$

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

$\Downarrow\Uparrow$

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

$\Downarrow\Uparrow$

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

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

$\Downarrow\Uparrow$

T(erm)					
gnarly	0.41	0.00	0.71	0.00	-0.58
wicked	0.41	0.00	-0.71	0.00	-0.58
awesome	0.82	-0.00	-0.00	-0.00	0.58
lame	0.00	0.85	0.00	-0.53	0.00
terrible	0.00	0.53	0.00	0.85	0.00

×

S(ingular values)				
2.45	0.00	0.00	0.00	0.00
0.00	1.62	0.00	0.00	0.00
0.00	0.00	1.41	0.00	0.00
0.00	0.00	0.00	0.62	0.00
0.00	0.00	0.00	0.00	-0.00

×

D(ocument)					
d1	0.50	-0.00	0.50	0.00	-0.71
d2	0.50	0.00	-0.50	0.00	0.00
d3	0.50	-0.00	0.50	0.00	0.71
d4	0.50	-0.00	-0.50	-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

T

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)					
gnarly	0.41	0.00	0.71	0.00	-0.58
wicked	0.41	0.00	-0.71	0.00	-0.58
awesome	0.82	-0.00	-0.00	-0.00	0.58
lame	0.00	0.85	0.00	-0.53	0.00
terrible	0.00	0.53	0.00	0.85	0.00

×

S(ingular values)				
2.45	0.00	0.00	0.00	0.00
0.00	1.62	0.00	0.00	0.00
0.00	0.00	1.41	0.00	0.00
0.00	0.00	0.00	0.62	0.00
0.00	0.00	0.00	0.00	-0.00

×

T

D(ocument)					
d1	0.50	-0.00	0.50	0.00	-0.71
d2	0.50	0.00	-0.50	0.00	0.00
d3	0.50	-0.00	0.50	0.00	0.71
d4	0.50	-0.00	-0.50	-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

gnarly	0.41	0.00
wicked	0.41	0.00
awesome	0.82	-0.00
lame	0.00	0.85
terrible	0.00	0.53

×

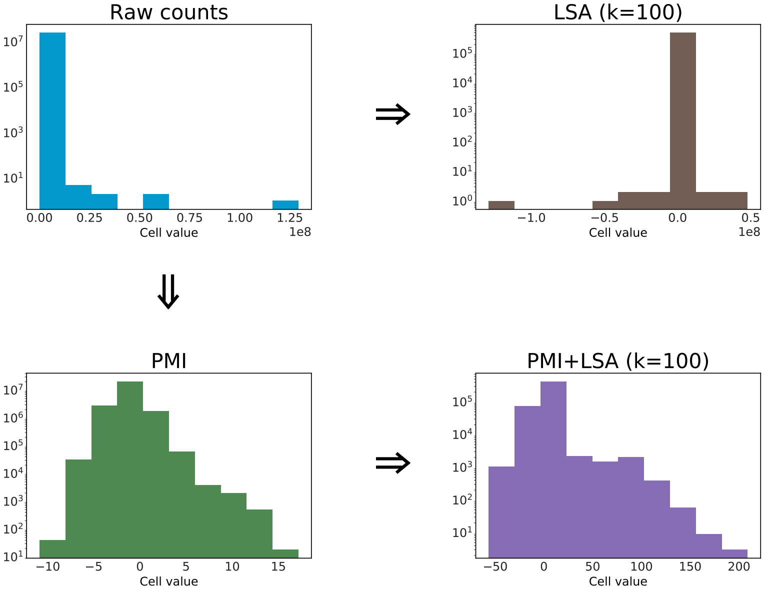
2.45	0.00
0.00	1.62

=

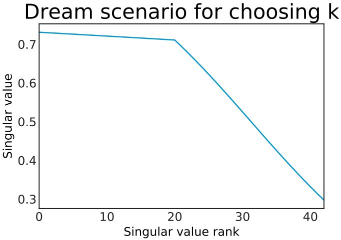
gnarly	1.00	0.00
wicked	1.00	0.00
awesome	2.00	0.00
lame	0.00	1.38
terrible	0.00	0.85

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

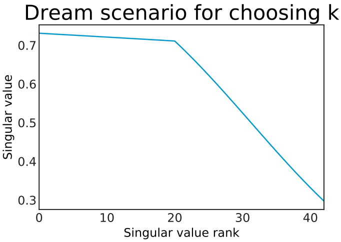
LSA: Cell-value comparisons ($k = 100$)

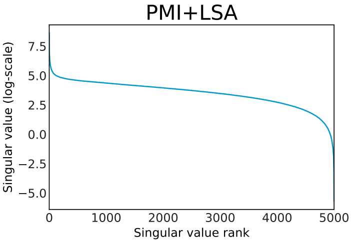


LSA: Choosing the dimensionality



LSA: Choosing the 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 & Hinton 2008)

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

LSA code snippets

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

In [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)

In [3]: giga5.shape

Out[3]: (5000, 5000)

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

In [5]: giga5_lsa100.shape

Out[5]: (5000, 100)
```

Autoencoders

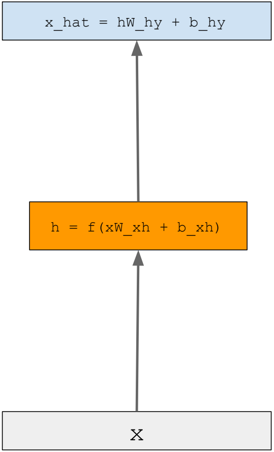
- Autoencoders are a flexible class of deep learning architectures for learning reduced dimensional representations.
- Chapter 14 of Goodfellow et al. (2016) is an excellent discussion.

The basic autoencoder model

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)



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.

`x_hat = hW_hy + b_hy`

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

`h = f(xW_xh + b_xh)`

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

`x`

```
y_err = x_hat - x

d_b_hy = y_err

h_err =
 y_err.dot(W_hy.T) * f′(h)

d_W_hy outer(h, y_err)

d_W_xh = outer(x, h_err)

d_b_xh = h_err
```

Autoencoder code snippets

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

In [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)

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

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

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

Finished epoch 10 of 10; error is 0.4883386066987744

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

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

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

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

Autoencoder code snippets

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

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

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

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

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

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

Global Vectors (GloVe)

- Pennington et al. (2014)
- Roughly speaking, the objective is to learn vectors for words such that their dot product is proportional to their probability of co-occurrence.
- We'll use the implementation in the `mittens` package (Dingwall & Potts 2018). 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

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

Equation (6):

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

Allowing different rows and columns:

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

That's PMI!

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

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

The weighted GloVe objective

Original

$$w_i^{\mathsf{T}} \widetilde{w}_k + b_i + \widetilde{b}_k = \log(X_{ik})$$

Weighted

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

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 $(x/x_{\max})^{\alpha}$.

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

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

GloVe learning

The loss calculations

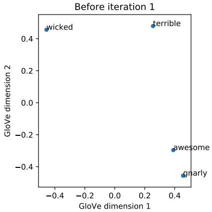
$$f(X_{ij})\left(w_i^\top \widetilde{w}_j - \log X_{ij}\right)$$

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 $\widetilde{w}_0$.

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

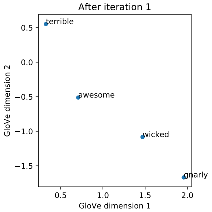


w_0		
gnarly	0.27	−0.27
wicked	−0.27	0.27
awesome	0.36	−0.50
terrible	0.08	0.16

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

$$0.92\left(\begin{bmatrix}0.27 & -0.27\end{bmatrix}^\top \begin{bmatrix}0.03 & 0.20\end{bmatrix} - \log(9)\right) = -2.06$$

$$0.92\left(\begin{bmatrix}-0.27 & 0.27\end{bmatrix}^\top \begin{bmatrix}0.03 & 0.20\end{bmatrix} - \log(9)\right) = -1.98$$



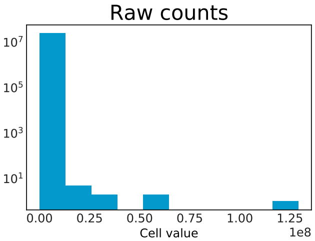
w_1		
gnarly	0.99	−0.85
wicked	0.74	−0.54
awesome	0.37	−0.26
terrible	0.12	0.21

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

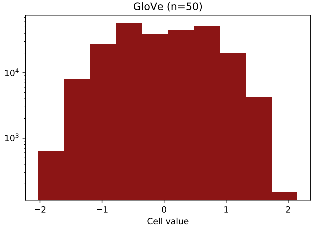
$$0.92\left(\begin{bmatrix}0.99 & -0.85\end{bmatrix}^\top \begin{bmatrix}0.34 & -0.25\end{bmatrix} - \log(9)\right) = -1.51$$

$$0.92\left(\begin{bmatrix}0.74 & -0.54\end{bmatrix}^\top \begin{bmatrix}0.34 & -0.25\end{bmatrix} - \log(9)\right) = -1.66$$

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



⇒



GloVe code snippets

```
In [1]: from mittens import GloVe
        import numpy as np
        import os
        import pandas as pd
        import vsm

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

        imdb5 = pd.read_csv(
            os.path.join(DATA_HOME, 'imdb_window5-scaled.csv.gz'), index_col=0)

        imdb20 = pd.read_csv(
            os.path.join(DATA_HOME, 'imdb_window20-flat.csv.gz'), index_col=0)

In [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)

In [4]: percentage_nonzero_vals_above(imdb5)

Out[4]: 0.017534398942316464

In [5]: percentage_nonzero_vals_above(imdb20)

Out[5]: 0.1519065095882084
```

GloVe code snippets

```
In [6]: glv = GloVe(max_iter=100, n=50)

In [7]: imdb5_glv = glv.fit(imdb5)

Iteration 100: loss: 536157.755

In [8]: glv.sess.close()

In [9]: imdb20_glv = glv.fit(imdb20)

Iteration 100: loss: 1043351.625

In [10]: # Restore the original `pd.DataFrame` structure:
         imdb20_glv = pd.DataFrame(imdb20_glv, index=imdb20.index)

In [11]: # To what a degree is the GloVe objective achieved?
         def correlation_test(true, pred):
             mask = true > 0
             M = pred.dot(pred.T)
             with np.errstate(divide='ignore'):
                 log_cooccur = np.log(true)
                 log_cooccur[np.isinf(log_cooccur)] = 0.0
                 row_prob = np.log(true.sum(axis=1))
                 row_log_prob = np.outer(row_prob, np.ones(true.shape[1]))
                 prob = log_cooccur - row_log_prob
             return np.corrcoef(prob[mask], M[mask])[0, 1]

In [12]: correlation_test(imdb5.values, imdb5_glv)

Out[12]: 0.38032242586515264

In [13]: correlation_test(imdb20.values, imdb20_glv.values)

Out[13]: 0.484126476892789
```

wordvec

- Introduced by Mikolov et al. (2013).
- Goldberg & Levy (2014) identify the relationship between word2vec and PMI.
- The TensorFlow tutorial Vector representations of words is very clear and links to code.
- Gensim package has a highly scalable implementation.

word2vec: From corpus to labeled data

it was the best of times, it was the worst of times, . . .

With window size 2:

x	y
it	was
it	the
was	it
was	the
was	best
the	was
the	it
the	best
the	of
. . .	

word2vec: Basic skip-gram

The basic skip-gram model estimates the probability of an input–output pair (a, b) as

$$P(b \mid a) = \frac{\exp(x_a w_b)}{\sum_{b' \in V} \exp(x_a w_{b'})}$$

where x_a is the row-vector representation of word a and w_b is the column vector representation of word b. Minimize:

$$-\sum_{i=1}^{m} \sum_{k=1}^{|V|} \mathbf{1}\{c_i = k\} \log \frac{\exp(x_i w_k)}{\sum_{j=1}^{|V|} \exp(x_i w_j)}$$

where V is the vocabulary and c is a one-hot encoded vector of the same length as V. This gives rise to a classifier:

$$C = \textbf{softmax}(XW + b)$$

We're back to our core insight for this unit: word and context matrix, pushing their dot products in a specific direction.

word2vec: Noise contrastive estimation

Training the basic skip-gram model directly is expensive for large vocabularies, because W, b, and C get so large. Noise contrastive estimation addresses that:

$$\sum_{a,b \in \mathbf{D}} -\log \sigma(x_a w_b) + \sum_{a,b \in \mathbf{D}'} \log \sigma(x_a w_b)$$

with σ the sigmoid activation function $\frac{1}{1+\exp(-x)}$. $\mathbf{D}'$ is a sample of pairs that don't appear in the training data.

Retrofitting

Central goals

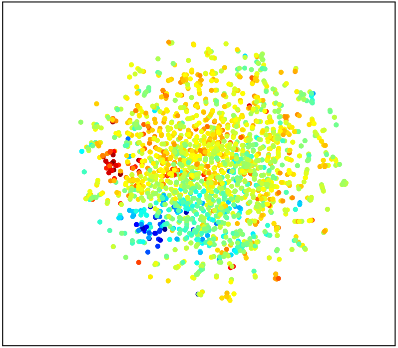
- Distributional representations are powerful and easy to obtain, but they tend to reflect only similarity (synonymy, connotation).
- Structured resources are sparse and hard to obtain, but they support learning rich, diverse semantic distinctions.
- Can we have the best aspects of both? Retrofitting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

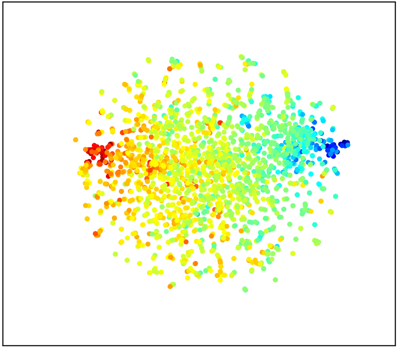
Purely distributional representations

- High-dimensional 😃
- Meaning from dense linguistic inter-relationships 😃
- Meaning *solely* from (*n*th-order) co-occurrence 😟
- No grounding in physical or social contexts 😟
- Not symbolic 😕

Grounding via supervision

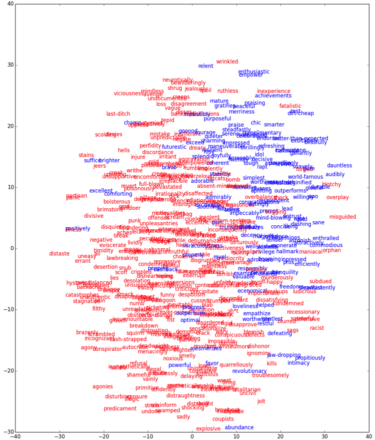
Word vectors to maximize unsupervised log-likelihood of words given documents and supervised prediction accuracy:

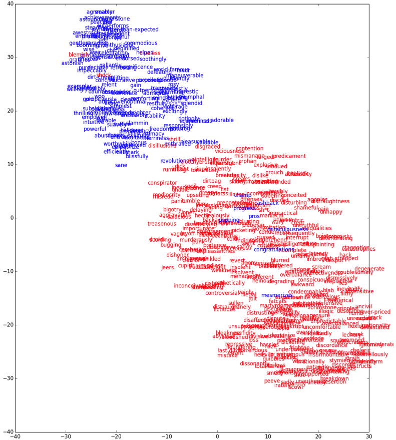




(Maas et al. 2011)

Hidden representations from a deep classifier





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 $\beta = \frac{1}{\textbf{Degree}(i)}$

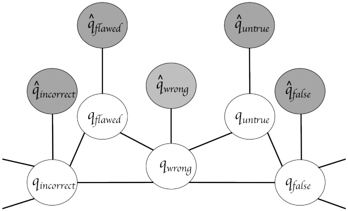
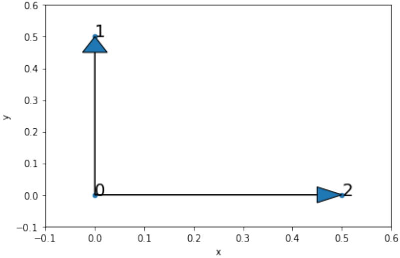
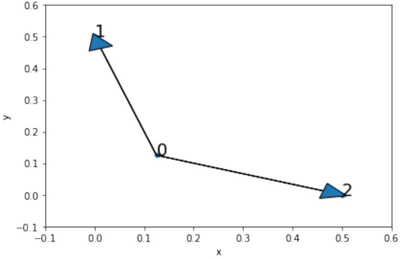


Figure 1: Word graph with edges between related words showing the observed (grey) and the inferred (white) word vector representations.

Simple retrofitting examples

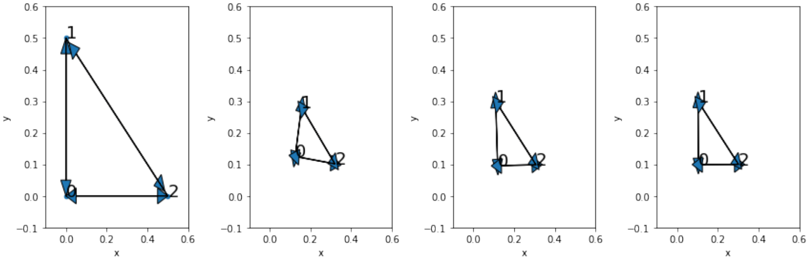
$$\sum_{i \in \mathbf{V}} \alpha_i \left\| \boldsymbol{q}_i - \hat{\boldsymbol{q}}_i \right\|^2 + \sum_{(i,j,r) \in \mathbf{E}} \beta_{ij} \left\| \boldsymbol{q}_i - \boldsymbol{q}_j \right\|^2$$





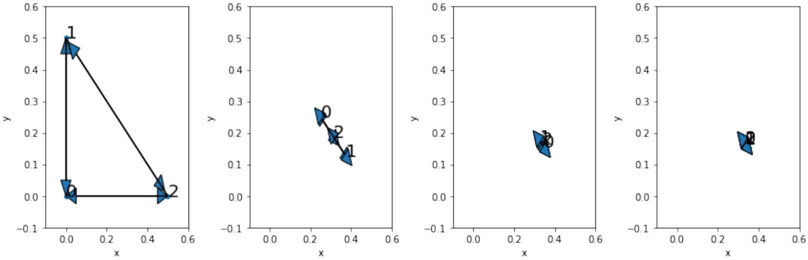
Simple retrofitting examples

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



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.

Retrofitting code snippets

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

In [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()}

In [3]: Q_hat

Out[3]:      x    y
        0  0.0  0.0
        1  0.0  0.5
        2  0.5  0.0

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

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

Converged at iteration 2; change was 0.0000

In [6]: X_retro

Out[6]:       x      y
        0  0.125  0.125
        1  0.000  0.500
        2  0.500  0.000

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

References I

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

Bojanowski, Piotr, Edouard Grave, Armand Joulin & Tomas Mikolov. 2016. Enriching word vectors with subword information. ArXiv:1607.04606.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer & R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41(6). 391–407. doi:10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

Dingwall, Nicholas & Christopher Potts. 2018. Mittens: An extension of GloVe for learning domain-specialized representations. In *Proceedings of the 2018 conference of the North American chapter of the Association for Computational Linguistics: Human language technologies*, 212–217. Stroudsburg, PA: Association for Computational Linguistics.

Faruqui, Manaal, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy & 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*, 1606–1615. Stroudsburg, PA: Association for Computational Linguistics. http://www.aclweb.org/anthology/N15-1184.

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

Goldberg, Yoav & Omer Levy. 2014. word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. Tech. Rep. arXiv:1402.3722v1 Bar Ilan University.

Goodfellow, Ian, Yoshua Bengio & Aaron Courville. 2016. *Deep learning*. MIT Press. http://www.deeplearningbook.org.

Hamilton, William L., Rex Ying & Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. In *Ieee data engineering bulletin*, 52–74. IEEE Press.

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

Hofmann, Thomas. 1999. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval*, 50–57. New York: ACM. doi:http://doi.acm.org/10.1145/312624.312649. http://doi.acm.org/10.1145/312624.312649.

Lengerich, Benjamin J., Andrew L. Maas & Christopher Potts. 2018. Retrofitting distributional embeddings to knowledge graphs with functional relations. In *Proceedings of the 27th international conference on computational linguistics*, 2423–2436. Stroudsburg, PA: Association for Computational Linguistics.

Levy, Omer & Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, .

Maas, Andrew L., Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng & Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the Association for Computational Linguistics*, 142–150. Portland, Oregon: Association for Computational Linguistics. http://www.aclweb.org/anthology/P11-1015.

References II

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

Manning, Christopher D. & Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado & Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Leon Bottou, Max Welling, Zoubin Ghahramani & Kilian Q. Weinberger (eds.), *Advances in neural information processing systems 26*, 3111–3119. Curran Associates, Inc. https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.

Mrkšić, Nikola, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen & 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*, 142–148. Association for Computational Linguistics. doi:10.18653/v1/N16-1018. http://aclanthology.coli.uni-saarland.de/pdf/N/N16/N16-1018.pdf.

Pennington, Jeffrey, Richard Socher & Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)*, 1532–1543. Doha, Qatar: Association for Computational Linguistics. http://www.aclweb.org/anthology/D14-1162.

Schütze, Hinrich. 1993. Word space. In S. J. Hanson, J. D. Cowan & C. L. Giles (eds.), *Advances in neural information processing systems 5*, 895–902. Morgan-Kaufmann. http://papers.nips.cc/paper/603-word-space.pdf.

Smith, Noah A. 2019. Contextual word representations: A contextual introduction. ArXiv:1902.06006v2.

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

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