Distributed word representations: Dimensionality reduction

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

Overview

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

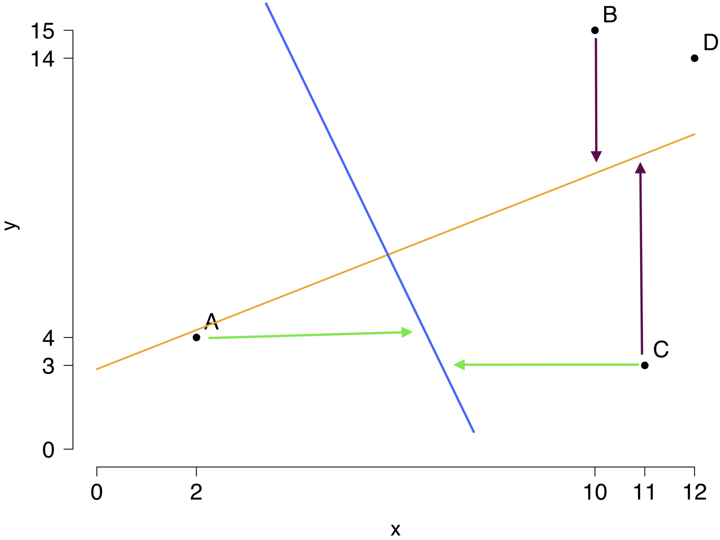
Latent Semantic Analysis (LSA)

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

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

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

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

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

$\Downarrow\Uparrow$

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

×

$\left(\text{D(ocument)}\right)^T$

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

×

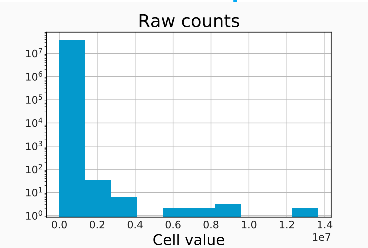
2.45	0.00
0.00	1.62

=

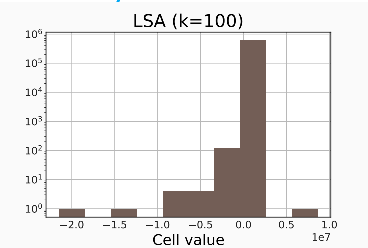
gnarly	1.00	0.00
wicked	1.00	0.00
awesome	2.00	0.00
lame	0.00	1.38
terrible	0.00	0.85

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

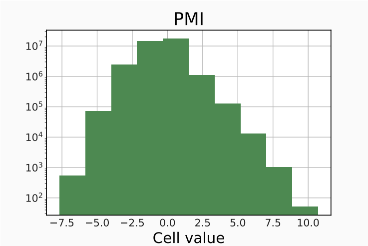
Cell-value comparisons ($k = 100$)



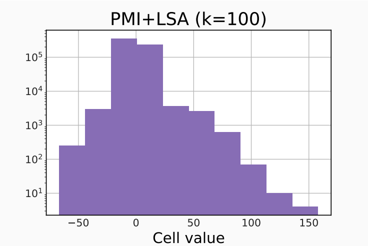
$\Rightarrow$



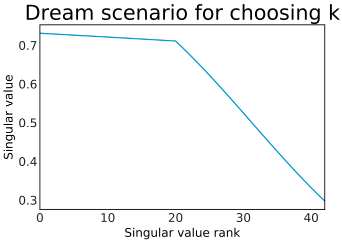
$\Downarrow$

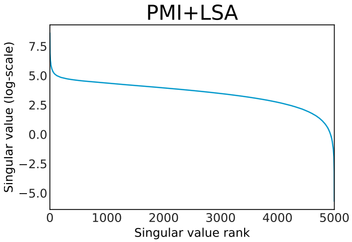


$\Rightarrow$



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.

`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

[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):

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

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

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

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

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

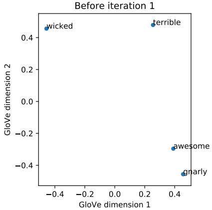
$$f\left(\begin{bmatrix} 100 & 99 & 75 & 10 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1.00 & 0.99 & 0.81 & 0.18 & 0.03 \end{bmatrix}$$

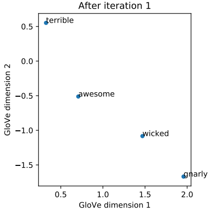
GloVe learning

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

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

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

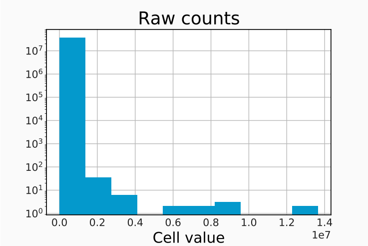
w_1		
gnarly	0.99	−0.85
wicked	0.74	−0.54
awesome	0.37	−0.26
terrible	0.12	0.21

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

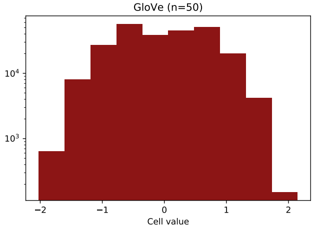
$$0.92\left(\begin{bmatrix}0.99 & -0.85\end{bmatrix}^{\top}\begin{bmatrix}0.34 & -0.25\end{bmatrix} - \log(9)\right) = -1.51$$

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

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



$\Rightarrow$



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

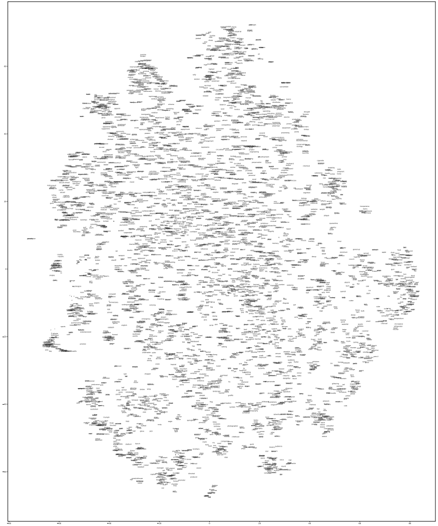
Visualization

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

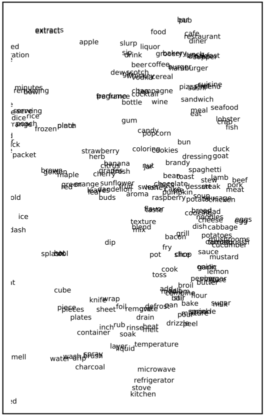
Techniques

- 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.

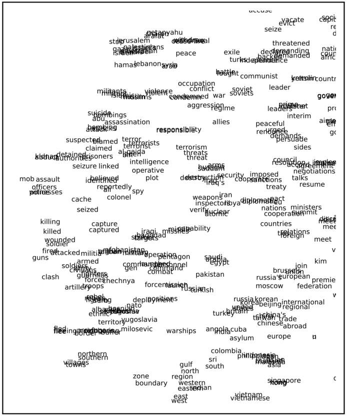
t-SNE on the giga20 PPMI VSM



t-SNE on the giga20 PPMI VSM

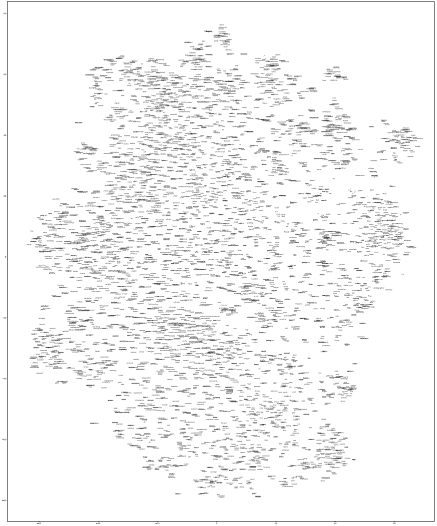


cooking

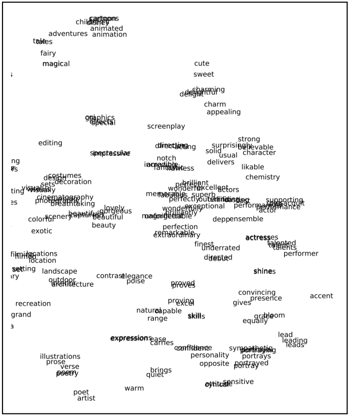


conflict

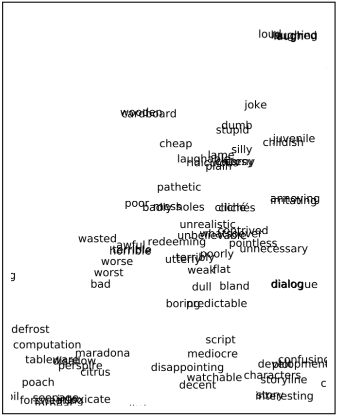
t-SNE on the yelp20 PPMI VSM



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.