

# Computational Social Science

## Word embeddings I

Dr. Thomas Davidson

Rutgers University

October 20, 2025

# Plan

1. Course updates
2. The vector-space model review
3. Latent semantic analysis
4. Language models

# Course updates

- ▶ Project proposal assignment on Canvas, due Thursday
  - ▶ A short description of the planned project
    - ▶ Details on data source and collection
    - ▶ Team information

# The vector-space model review

## Vector representations

- ▶ Last week we looked at how we can represent texts as numeric vectors
  - ▶ Documents as vectors of words
  - ▶ Words as vectors of documents
- ▶ A document-term matrix ( $DTM$ ) is a matrix where documents are represented as rows and tokens as columns

# The vector-space model review

## Weighting schemes

- ▶ We can use different schemes to weight these vectors
  - ▶ Binary (Does word  $w_i$  occur in document  $d_j$ ?)
  - ▶ Counts (How many times does word  $w_i$  occur in document  $d_j$ ?)
  - ▶ TF-IDF (How many times does word  $w_i$  occur in document  $d_j$ , accounting for how often  $w_i$  occurs across all documents  $d \in D$ ?)
    - ▶ Recall *Zipf's Law*: a handful of words account for most words used; such words do little to help us to distinguish between documents

# The vector-space model review

## Cosine similarity

$$\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} = \frac{\sum_i \vec{u}_i \vec{v}_i}{\sqrt{\sum_i \vec{u}_i^2} \sqrt{\sum_i \vec{v}_i^2}}$$

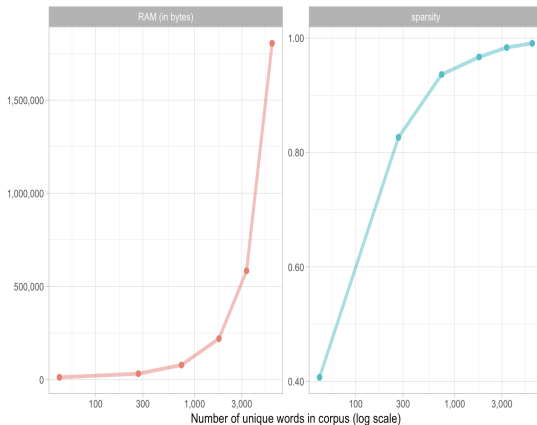
# The vector-space model review

## Limitations

- ▶ These methods produce *high-dimensional, sparse* vector representations
  - ▶ Given a vocabulary of unique tokens  $N$  the length of each vector  $|V| = N$ .
  - ▶ Many values will be zero since most documents only contain a small subset of the vocabulary.

# The vector-space model review

## Limitations



Source: <https://smlltar.com/embeddings.html>



# Latent semantic analysis

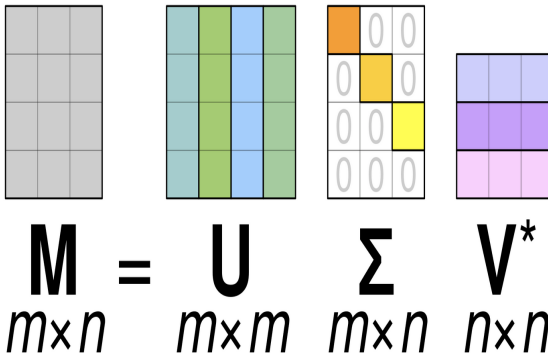
## Latent Semantic Analysis

- ▶ One approach to reduce dimensionality and better capture semantics is called **Latent Semantic Analysis (LSA)**
  - ▶ We can use a process called *singular value decomposition* to find a *low-rank approximation* of a DTM.
- ▶ This provides *low-dimensional, dense* vector representations
  - ▶ Low-dimensional, since  $|V| \ll N$
  - ▶ Dense, since vectors contain real values, with few zeros
- ▶ In short, we can “squash” a big matrix into a much smaller matrix while retaining important information.

$$DTM = U\Sigma V^T$$

# Latent semantic analysis

## Singular Value Decomposition



The diagram illustrates the Singular Value Decomposition (SVD) process. It shows four matrices:  $M$  (a 4x4 gray matrix),  $U$  (a 4x4 matrix with colored columns: teal, green, blue, green),  $\Sigma$  (a 4x4 matrix with diagonal elements in orange, yellow, and light green, and zeros elsewhere), and  $V^*$  (a 4x4 matrix with colored rows: light blue, purple, purple, pink). Below the matrices is the equation  $M = U \Sigma V^*$  with dimensions  $m \times n$ ,  $m \times m$ ,  $m \times n$ , and  $n \times n$  respectively.

$$\begin{matrix} M & = & U & \Sigma & V^* \\ m \times n & & m \times m & m \times n & n \times n \end{matrix}$$

See the Wikipedia page for video of the latent dimensions in a sparse TDM.

# Latent semantic analysis

## Example: Political tweets

```
pol_tweets <- read.csv("../data/politics_twitter.csv") %>% sample_frac(
tweet_words <- pol_tweets %>% unnest_tokens(word, text)
word_counts <- tweet_words %>% count(word)
tweet_words <- tweet_words %>% left_join(word_counts) %>%
  filter(n >= 50) %>% anti_join(stop_words)

tweet_words_tfidf <- tweet_words %>% bind_tf_idf(word, status_id, n)
DTM <- tweet_words_tfidf %>%
  cast_dtm(status_id, word, tf)

X <- as.matrix(DTM)
```

# Latent semantic analysis

## Creating a lookup dictionary

We can construct a list to allow us to easily find the index of a particular token.

```
lookup.index.from.token <- list()

for (i in 1:length(colnames(X))) {
  lookup.index.from.token[colnames(X)[i]] <- i
}
```

# Latent semantic analysis

## Using the lookup dictionary

This easily allows us to find the vector representation of a particular word. Note how most values are zero.

```
lookup.index.from.token["america"]
```

```
## $america
```

```
## [1] 63
```

```
round(as.numeric(X[,unlist(lookup.index.from.token["america"])]),5)[1:10]
```

```
## [1] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [10] 0.03179 0.00000 0.22934 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [19] 0.00000 0.00000 0.00000 0.00000 0.00000 0.02929 0.00000 0.00000 0.00000 0.00000
```

```
## [28] 0.00000 0.00000 0.00000 0.00000 0.00000 0.02954 0.00000 0.00000 0.00000 0.00000
```

```
## [37] 0.00000 0.00000 0.03037 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [46] 0.03347 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [55] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [64] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [73] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
## [82] 0.00000 0.00000 0.00000 0.03098 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

# Latent semantic analysis

## Calculating similarities

The following code normalizes each *column* (rather than row normalization seen last lecture) and constructs a word-word cosine-similarity matrix.

```
normalize <- function(X) {  
  columnNorms <- sqrt(colSums(X^2))  
  Xn <- X / matrix(columnNorms,  
                    nrow = nrow(X),  
                    ncol = ncol(X), byrow = TRUE)  
  return(Xn)  
}  
  
X.n <- normalize(X)  
  
sims <- crossprod(X.n) # Optimized routine for t(X.n) %*% X.n  
dim(sims)  
  
## [1] 947 947
```

# Latent semantic analysis

## Most similar function

For a given token, this function allows us to find the  $n$  most similar tokens in the similarity matrix, where  $n$  defaults to 10.

```
get.top.n <- function(token, sims, n=10) {  
  top <- sort(sims[unlist(lookup.index.from.token[token]),],  
              decreasing=T)[1:n]  
  return(top)  
}
```

# Latent semantic analysis

## Finding similar words

```
get.top.n("democracy", sims, n = 10)
```

##	democracy	january	elections	forthepeople	election	0
##	1.00000000	0.16643919	0.14827262	0.14067815	0.11648251	0
##	strengthen	supreme	republican	violent		
##	0.10245299	0.10044865	0.10000214	0.08846004		



# Latent semantic analysis

## Finding similar words

Choose another word to inspect.

```
get.top.n("", sims, n = 10)
```

# Latent semantic analysis

## Singular value decomposition

The `svd` function allows us to decompose the DTM. We can then easily reconstruct it using the formula shown above.

```
# Computing the singular value decomposition
lsa <- svd(X)

# We can easily recover the original matrix from this representation
X.2 <- lsa$u %*% diag(lsa$d) %*% t(lsa$v) # X = U \Sigma V^T

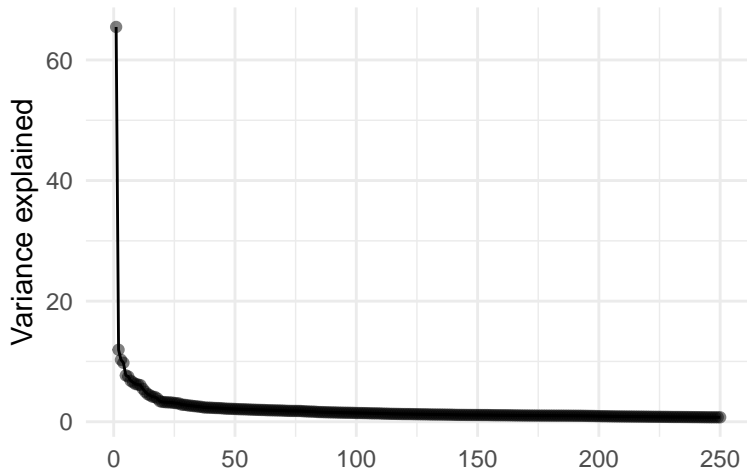
# Verifying that values are the same, example of first column
sum(round(X-X.2,5))

## [1] 0
```

# Latent semantic analysis

## Singular value decomposition

Variance explained by singular values



# Latent semantic analysis

## Truncated singular value decomposition

In the example above retained the original matrix dimensions. The point of latent semantic analysis is to compute a *truncated* SVD such that we have a new matrix in a sub-space of  $X$ . In this case we only want to retain the first  $k$  dimensions of the matrix.

```
k <- 50 # Dimensions in truncated matrix

# We can take the SVD of X but only retain the first k singular values
lsa.2 <- svd(X, nu=k, nv=k)

# In this case we reconstruct X just using the first k singular values
X.trunc <- lsa.2$u %*% diag(lsa.2$d[1:k]) %*% t(lsa.2$v)

# But the values will be slightly different since it is an approximation
# Some information is lost due to the compression
sum(round(X-X.trunc,2))

## [1] 1665.51
```

# Latent semantic analysis

## Inspecting the LSA matrix

```
words.lsa <- t(lsa.2$v)
colnames(words.lsa) <- colnames(X)

round(as.numeric(words.lsa[,unlist(lookup.index.from.token["democracy"])]
## [1] -0.00029  0.00519 -0.00507 -0.00323 -0.00487  0.00555 -0.01013
## [9]  0.00041 -0.00142 -0.01060  0.00007 -0.00091  0.01427 -0.00258
## [17]  0.00477  0.00509  0.00174 -0.00042 -0.00435  0.00622  0.00314
## [25]  0.00543 -0.00246 -0.00419 -0.02141  0.02371 -0.01952 -0.00359
## [33] -0.01974  0.00518 -0.00706 -0.01359  0.00451 -0.00654  0.01165
## [41] -0.00259 -0.00976  0.00119 -0.00148 -0.00221  0.00740  0.01332
## [49]  0.02503 -0.01066
```

# Latent semantic analysis

## Recalculating similarities using the LSA matrix

```
words.lsa.n <- normalize(words.lsa)
sims.lsa <- t(words.lsa.n) %*% words.lsa.n
```

# Latent semantic analysis

## Comparing similarities

```
bind_cols(names(get.top.n("democracy",sims)), names(get.top.n("democrac
```

```
## # A tibble: 10 x 2
##   ...1      ...2
##   <chr>    <chr>
## 1 democracy democracy
## 2 january  stand
## 3 elections communist
## 4 forthepeople lgbtq
## 5 election  word
## 6 free      breaking
## 7 strengthen integrity
## 8 supreme   election
## 9 republican taliban
## 10 violent  rights
```

# Latent semantic analysis

## Comparing similarities

```
get.bottom.n <- function(token, sims, n=10) {  
  bottom <- sort(sims[unlist(lookup.index.from.token[token]),],  
                 decreasing=F)[1:n]  
  return(bottom)  
}
```

```
get.bottom.n("democracy", sims)
```

##	told	alaska	aren't	happy	jobs
##	0	0	0	0	0
##	kamalaharris	peace	vital	vulnerable	
##	0	0	0	0	



# Latent semantic analysis

## Comparing similarities

```
bind_cols(names(get.top.n("",sims, n = 5)), names(get.top.n("",sims.lsa  
## # A tibble: 0 x 0
```

# Latent semantic analysis

## Comparing similarities

```
get.top.n("", sims, n = 5)
```

```
## [1] NA NA NA NA NA
```

```
get.top.n("", sims.lsa, n = 5)
```

```
## [1] NA NA NA NA NA
```

# Latent semantic analysis

## Comparing similarities

```
get.top.n("", sims)
```

```
## [1] NA NA NA NA NA NA NA NA NA NA
```

```
get.top.n("", sims.lsa)
```

```
## [1] NA NA NA NA NA NA NA NA NA NA
```

# Latent semantic analysis

## Exercise

Re-run the code above with a different value of  $k$  (try lower or higher). Compare some terms in the original similarity matrix and the new matrix. How does changing  $k$  affect the results?

```
get.top.n("", sims)
get.top.n("", sims.lsa)
```

# Latent semantic analysis

## Inspecting the latent dimensions

We can analyze the meaning of the latent dimensions by looking at the terms with the highest weights in each row. In this case I use the raw LSA matrix without normalizing it. What do you notice about the dimensions?

```
for (i in 1:dim(words.lsa)[1]) {  
  top.words <- sort(words.lsa[i,], decreasing=T)[1:5]  
  print(paste(c("Dimension: ",i), collapse=" "))  
  print(round(top.words,3))  
}
```

```
## [1] "Dimension: 1"
```

```
##          tens          grow          mental discrimination  
##          0           0           0           0
```

```
## [1] "Dimension: 2"
```

```
## president    people    trump    biden    american  
##    0.916    0.240    0.175    0.144    0.128
```

```
## [1] "Dimension: 3"
```

```
##          president          https          t.co realdonaldtrump
```

# Latent semantic analysis

## Limitations of Latent Semantic Analysis

- ▶ Bag-of-words assumptions and document-level word associations
  - ▶ We still treat words as belonging to documents and lack finer context about their relationships
    - ▶ Although we could theoretically treat smaller units like sentences as documents
- ▶ Matrix computations become intractable with large corpora
- ▶ A neat linear algebra trick, but no underlying *language model*

# Language models

## Intuition

- ▶ A language model is a probabilistic model of language use
- ▶ Given some string of tokens, what is the most likely token?
  - ▶ Examples
    - ▶ Auto-complete
    - ▶ Google search completion

# Language models

## Bigram models

- ▶  $P(w_i|w_{i-1})$  = What is the probability of word  $w_i$  given the last word,  $w_{i-1}$ ?
  - ▶  $P(\textit{Jersey}|\textit{New})$
  - ▶  $P(\textit{Brunswick}|\textit{New})$
  - ▶  $P(\textit{York}|\textit{New})$
  - ▶  $P(\textit{Sociology}|\textit{New})$



# Language models

## Bigram models

- ▶ We use a corpus of text to calculate these probabilities from word co-occurrence.
  - ▶  $P(\text{Jersey}|\text{New}) = \frac{C(\text{New Jersey})}{C(\text{New})}$ , e.g. proportion of times “New” is followed by “Jersey”, where  $C()$  is the count operator.
- ▶ More frequently occurring pairs will have a higher probability.
  - ▶ We might expect that  $P(\text{York}|\text{New}) \approx P(\text{Jersey}|\text{New}) > P(\text{Brunswick}|\text{New}) \gg P(\text{Sociology}|\text{New})$

# Language models

## Incorporating more information

- ▶ We can also model the probability of a word, given a sequence of words
- ▶  $P(x|S)$  = What is the probability of some word  $x$  given a partial sentence  $S$ ?
- ▶  $A = P(\text{Jersey} | \text{Rutgers University is in New})$
- ▶  $B = P(\text{Brunswick} | \text{Rutgers University is in New})$
- ▶  $C = P(\text{York} | \text{Rutgers University is in New})$
- ▶ In this case we have more information, so “York” is less likely to be the next word. Hence,
  - ▶  $A \approx B > C$ .

# Language models

## Estimation

We can compute the probability of an entire sequence of words by using considering the *joint conditional probabilities* of each pair of words in the sequence. For a sequence of  $n$  words, we want to know the joint probability of  $P(w_1, w_2, w_3, \dots, w_n)$ . We can simplify this using the chain rule of probability:

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\dots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^n P(w_k|w_{1:k-1}) \end{aligned}$$

# Language models

## Estimation

The bigram model simplifies this by assuming it is a first-order Markov process, such that the probability  $w_k$  only depends on the previous word,  $w_{k-1}$ .

$$P(w_{1:n}) \approx \prod_{k=1}^n P(w_k | w_{k-1})$$

These probabilities can be estimated by using Maximum Likelihood Estimation on a corpus.

See <https://web.stanford.edu/~jurafsky/slp3/3.pdf> for an excellent review of language models

# Language models

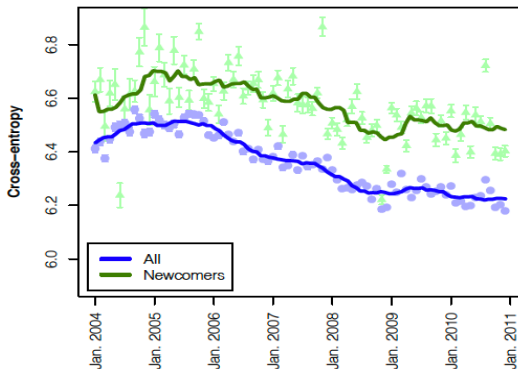
## Empirical applications

- ▶ Danescu-Niculescu-Mizil et al. 2013 construct a bigram language model for each month on *BeerAdvocate* and *RateBeer* to capture the language of the community
  - ▶ For any given comment or user, they can then use a measure called *cross-entropy* to calculate how “surprising” the text is, given the assumptions about the language model
- ▶ The theory is that new users will take time to assimilate into the linguistic norms of the community

[https://en.wikipedia.org/wiki/Cross\\_entropy](https://en.wikipedia.org/wiki/Cross_entropy)

# Language models

## Empirical applications



(a) BeerAdvocate

Danescu-Niculescu-Mizil, Cristian, Robert West, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. "No Country for Old Members: User Lifecycle and Linguistic Change in Online Communities." In Proceedings of the 22nd International Conference on World Wide Web, 307–18. ACM. <http://dl.acm.org/citation.cfm?id=2488416>.

# Language models

## Limitations of N-gram language models

- ▶ Language use is much more complex than N-gram language models
- ▶ Three limitations
  1. Insufficient data to sufficiently model language generation
  2. Complex models become intractable to compute
  3. Limited information on word order

# Summary

- ▶ Limitations of sparse representations of text
  - ▶ LSA allows us to project sparse matrix into a dense, low-dimensional representation
- ▶ Probabilistic language models allow us to directly model language use



## Next lecture

- ▶ How neural language models allow us to create more meaningful semantic representations of texts