Computational Sociology

Word embeddings

Dr. Thomas Davidson

Rutgers University

March 7, 2024

Plan

1. Course updates
2. The vector-space model review
3. Latent semantic analysis
4. Language models
5. Word embeddings
6. Contextualized embeddings

Course updates

- Project proposals due 5pm tomorrow
  - Submit via email
- Mid-semester evaluation
  - Please complete by Friday
- Homework 2 and proposal feedback coming soon

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

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

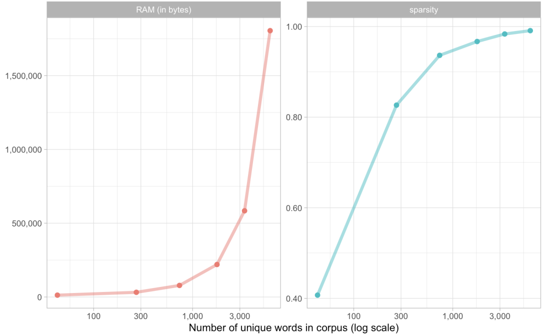
The vector-space model review

Limitations

- These methods produce *high-dimensinal*, *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://smltar.com/embeddings.html

Latent semantic analysis

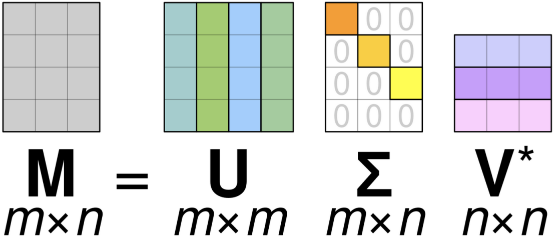
Latent Semantic Analysis

- 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 $|V| << N$
  - 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



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

Latent semantic analysis

Loading data and packages

```
library(tidyverse)
library(tidytext)
library(ggplot2)
library(stringr)

df <- read_csv("../data/politics_twitter.csv")
dim(df)
```

```
## [1] 34220    13
```

```
unique(df$screen_name)
```

```
##  [1] "JoeBiden"        "KamalaHarris"    "SpeakerPelosi"   "BernieSa
##  [5] "AOC"             "SenSchumer"      "LeaderMcConnell" "LindseyG
##  [9] "tedcruz"         "Mike_Pence"      "MarshaBlackburn" "lisamurk
```

Latent semantic analysis

Text preprocessing

```
texts.tidy <- df %>%
    mutate(text = gsub("@\\w+", "", text)) %>%
    mutate(text = gsub("#\\w+", "", text)) %>%
    mutate(text = gsub("’", "'", text)) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(status_id, word) %>%
    group_by(word) %>%
    mutate(tweet_count = n_distinct(status_id),
           text_count = sum(n)) %>%
    ungroup() %>%
    filter(tweet_count >= 100 & text_count < 1E4) %>%
    filter(!str_detect(word, "^[0-9]+$")) %>%
    bind_tf_idf(word, status_id, n)
```

Latent semantic analysis

Creating a DTM

```
X <- texts.tidy %>% cast_dtm(status_id, word, tf_idf) %>% as.matrix()
```

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["president"]
```

```
## $president
## [1] 26
```

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

```
##   [1] 0.00000 0.00000 0.00000 0.00000 0.31424 0.00000 0.00000 0.0000
##  [10] 0.13748 0.00000 0.00000 0.00000 0.00000 0.54992 0.00000 0.7332
##  [19] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.62848 0.0000
##  [28] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000
##  [37] 0.00000 0.00000 0.00000 0.21997 0.00000 0.00000 0.00000 0.0000
##  [46] 0.43994 0.00000 0.00000 0.31424 0.00000 0.00000 0.00000 0.3142
##  [55] 0.00000 0.00000 0.36661 0.36661 0.00000 0.00000 0.24441 0.2749
##  [64] 0.19997 0.00000 0.12939 0.16921 0.00000 0.00000 0.00000 0.4399
##  [73] 0.00000 0.00000 0.43994 0.00000 0.00000 0.00000 0.00000 0.0000
##  [82] 0.39994 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000
```

Latent semantic analysis

Calculating similarties

The following code normalizes each *column* 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] 830 830
```

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("", sims, n = 5)
```

Latent semantic analysis

Singular value decomposition

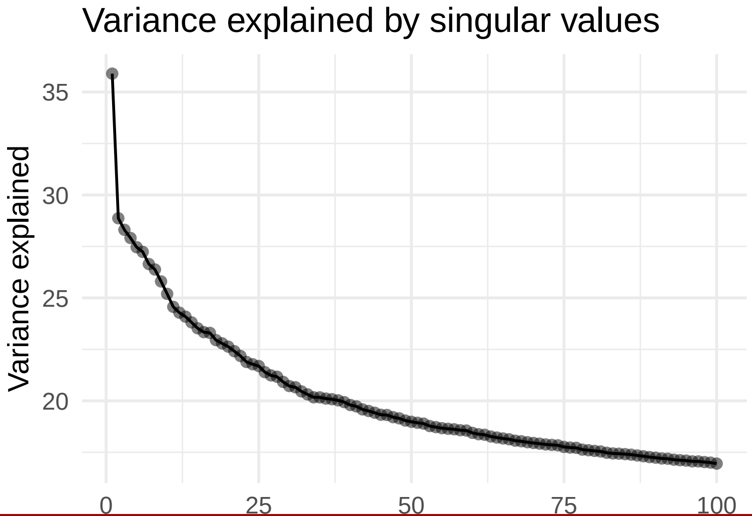
The `svd` function allows us to decompose the DTM. We can then easily reconstruct it using the formula shown above.

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

# We can 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



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 approximatio
# Some information is lost due to the compression
sum(round(X-X.trunc,2))
```

```
## [1] 23105.19
```

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["president"]
```

```
##  [1] -0.316 -0.131 -0.190 -0.168  0.249  0.236  0.078 -0.097  0.588
## [11] -0.036 -0.023 -0.007 -0.019  0.008  0.078  0.054 -0.263 -0.189
## [21] -0.084 -0.121 -0.040 -0.043 -0.025  0.024 -0.001 -0.005  0.023
## [31]  0.018  0.058 -0.026 -0.015 -0.030 -0.002 -0.048  0.055  0.065
## [41]  0.011  0.018  0.015 -0.008  0.017 -0.012 -0.004 -0.008 -0.004
```

Latent semantic analysis

Recalculating similarties 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("economy",sims)), names(get.top.n("economy",s
```

```
## # A tibble: 10 x 2
##    ...1     ...2
##    <chr>    <chr>
##  1 economy  economy
##  2 build    paying
##  3 rebuild  invest
##  4 wealth   economic
##  5 growing  rebuild
##  6 jobs     jobs
##  7 create   create
##  8 alaska's investments
##  9 virus    investment
## 10 paycheck build
```

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("health", sims)
```

```
##    region    allies    israel   victims    speech    served happened      f
##         0         0         0         0         0         0        0
##        dc    russia
##         0         0
```

Latent semantic analysis

Comparing similarities

```
bind_cols(names(get.top.n("health",sims, n = 10)), names(get.top.n("hea
```

```
## # A tibble: 10 x 2
##    ...1       ...2
##    <chr>      <chr>
##  1 health     health
##  2 care       care
##  3 insurance  insurance
##  4 public     affordable
##  5 affordable medicare
##  6 system     quality
##  7 medicare   existing
##  8 pandemic   pre
##  9 crisis     conditions
## 10 access     expand
```

Latent semantic analysis

Execise

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. What do you notice about the dimensions?

```
lsa_dimensions <- tibble(Word1 = character(),
                         Word2 = character(),
                         Word3 = character())

for (i in 1:nrow(words.lsa)) {
  top.words <- sort(words.lsa[i,], decreasing = TRUE)[1:3]
  temp <- tibble(Word1 = names(top.words)[1],
                 Word2 = names(top.words)[2],
                 Word3 = names(top.words)[3])
  lsa_dimensions <- bind_rows(lsa_dimensions, temp)
}
```

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}?
  - $P(Jersey|New)$
  - $P(Brunswick|New)$
  - $P(York|New)$
  - $P(Sociology|New)$

Language models

Bigram models

- We use a corpus of text to calculate these probabilities from word co-occurrence.
  - $P(Jersey|New) = \frac{C(New\ \ Jersey)}{C(New)}$, 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 $P(York|New) \approx P(Jersey|New) > P(Brunswick|New) >> P(Sociology|New)$

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?
- $A = P(Jersey|Rutgers\ University\ is\ in\ New)$
- $B = P(Brunswick|Rutgers\ University\ is\ in\ New)$
- $C = P(York|Rutgers\ University\ is\ in\ New)$
- 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 $P(w_1, w_2, w_3, ..., w_n)$. We can simplify this using the chain rule of probability:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n\breve{}1})$$

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

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

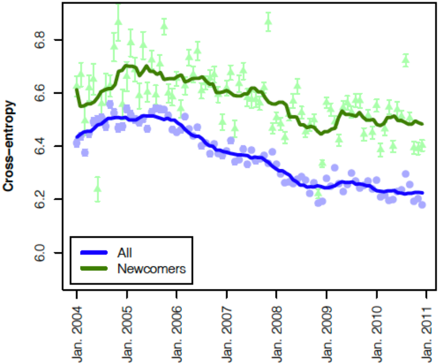
Empirical applications

- 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

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 for meaningful language generation
    2. More complex models become intractable to compute
    3. Limited information on word order

Language models

Neural language models

- Recent advances in both the availability of large corpora of text *and* the development of neural network models have resulted in new ways of computing language models.
- By using machine-learning to train a language model, we can construct better, more meaningful vector representations

Word embeddings

Intuition

- We use the context in which a word occurs to train a language model
  - The model learns by viewing millions of short snippets of text (e.g 5-grams)
- This model outputs a vector representation of each word in k-dimensional space, where $k << |V|$.
  - Like LSA, these vectors are *dense*
    - Each element contains a real number and can be positive or negative

Word embeddings

Word2vec: Skip-gram and continuous bag-of-words (CBOW)

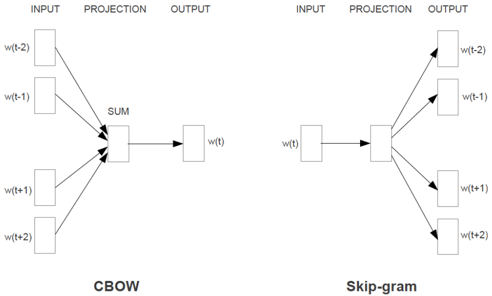


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Word embeddings

Word2vec: CBOW intuition

- We start with a string where the focal word is known, but hidden from the model, but we know the context within a window, in this case two words on either side of the focal word
  - e.g. "The cat ? on the", where ? = "sat"
- The model is trained using a process called *negative sampling*, where it must distinguish between the true sentence and "fake" sentences where ? is replaced with another token.
  - Each "guess" allows the model to begin to learn the correct answer
- By repeating this for millions of text snippets the model is able to "learn" which words go with which contexts

Word embeddings

Word2vec: Skip-gram intuition

- We start with a string where the focal is known, but the context within the window is hidden
  - e.g. "$?_1$ $?_2$ sat $?_3$ $?_4$"
- The model tests different words in the vocabulary to predict the missing context words
  - Each "guess" allows the model to begin to learn the correct answer
- By repeating this for millions of text snippets the model is able to "learn" which contexts go with which words

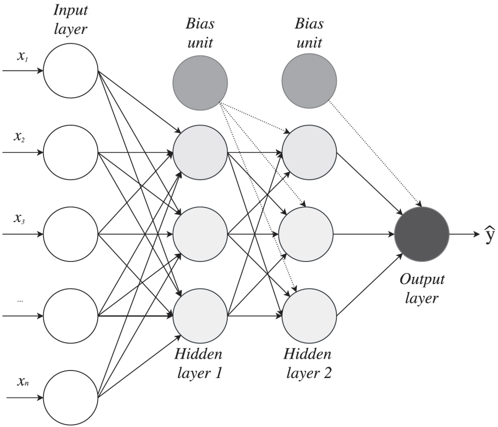
Word embeddings

Word2vec: Model

- Word2vec uses a shallow neural-network to predict a word given a context (CBOW) or a context given a word (skip-gram)
  - But we do not care about the prediction itself, only the *weights* the model learns
- It is a self-supervised method since the model is able to update using the correct answers
  - e.g. In CBOW the model knows when the prediction is wrong and updates the weights accordingly

Word embeddings

Word2vec: Feed-forward neural network



This example shows a two-layer feed-forward neural network.

Word embeddings

Word2vec: Estimation procedure

- Batches of text are passed through the network
  - After each batch, weights are updated using *back-propagation*
    - The model updates its weights in the direction of the correct answer (the objective is to improve predictive accuracy)
    - Optimization via *stochastic gradient descent*

Word embeddings

Vector representations of words

- Each word is represented as a vector of weights learned by the neural network
  - Word embeddings are *byproduct* of training a neural language model
  - Each element of this vector represents how strongly the word activates a neuron in the hidden layer of the network
  - This represents the association between the word and a given dimension in semantic space

Word embeddings

Distributional semantics

- The word vectors in the embedding space capture information about the context in which words are used
  - Words with similar meanings are situated close together in the embedding space
- *Distributional semantics* is the theory that the meaning of a word is derived from its context in language use
  - "You shall know a word by the company it keeps", linguist J.R. Firth (1957)
- This is consistent with philosopher Ludwig Wittgenstein's *use theory of meaning*
  - "the meaning of a word is its use in the language", *Philosophical Investigations* (1953)

Word embeddings

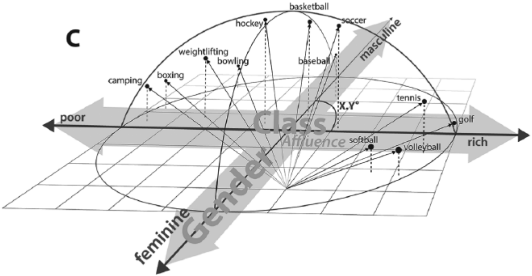
Analogies

- The most famous result from the initial word embedding paper is the ability of these vectors to capture analogies:
  - $king - man + woman \approx queen$
  - $Madrid - Spain + France \approx Paris$

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient Estimation of Word Representations in Vector Space." arXiv Preprint *arXiv*:1301.3781 / Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. "Distributed Representations of Words and Phrases and Their Compositionality." In *Advances in Neural Information Processing Systems*, 3111–19.

Word embeddings

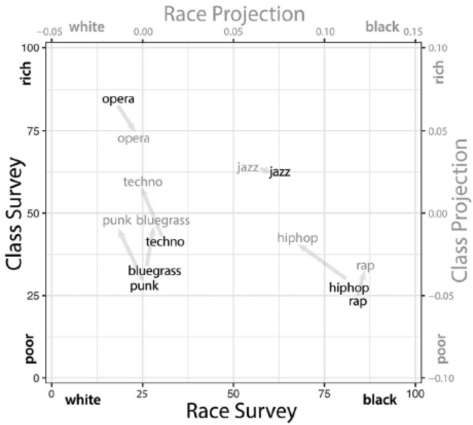
Sociological applications: Understanding social class



Kozlowski, Austin C., Matt Taddy, and James A. Evans. 2019. "The Geometry of Culture: Analyzing the Meanings of Class through Word Embeddings." American Sociological Review, September, 000312241987713.

Word embeddings

Sociological applications: Understanding social class



Word embeddings

Sociological applications: Latent dimensions

Table 4
Term pairs for immigration-citizenship cultural dimension.

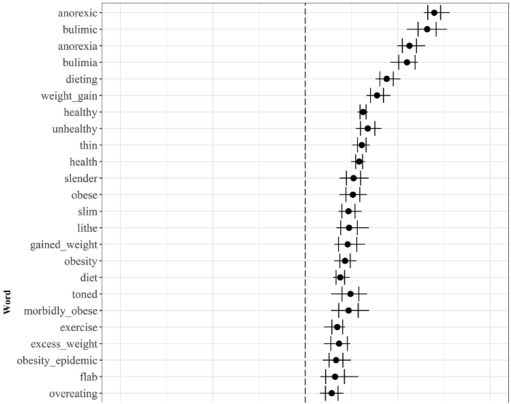
Immigrants	Citizens
Immigration	Citizenship
Immigrant	Citizen
Foreign	Domestic
Foreigner	Native
Outsider	Insider
Stranger	Local
Alien	Resident
Foreigner	Resident
Alien	Native
Immigrant	Local
Foreign	Familiar

Stoltz, Dustin S., and Marshall A. Taylor. 2021. "Cultural Cartography with Word Embeddings." *Poetics* 88 (October): 101567.

Word embeddings

Sociological applications: Understanding cultural schemas

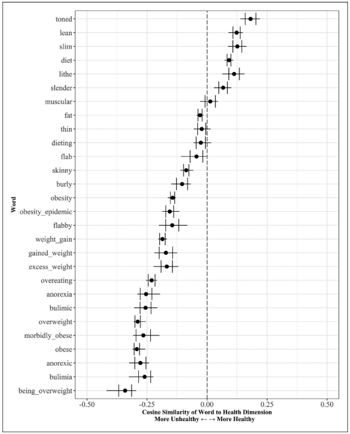
Figure 4: Gendering of Obesity-Related Words



Arseniev-Koehler, Alina, and Jacob G. Foster. 2022. "Machine Learning as a Model for Cultural Learning: Teaching an Algorithm What It Means to Be Fat." *Sociological Methods & Research* 51 (4): 1484–1539.

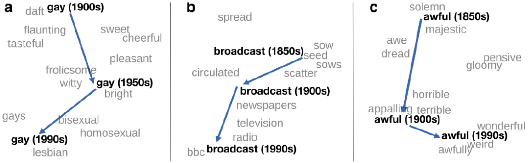
Word embeddings

Sociological applications: Understanding cultural schemas



Word embeddings

Sociological applications: Semantic change



Hamilton, William L., Jure Leskovec, and Dan Jurafsky. 2016. "Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change." In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 1489–1501.

Word embeddings

Sociological applications: Semantic change

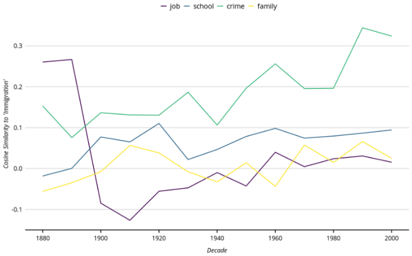
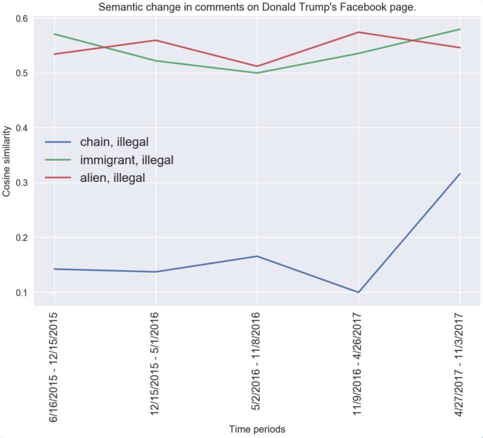


Fig. 1. Cosine Similarity of 'Immigration' and Key Terms by Decade, 1880 to 2000.

Stoltz, Dustin S., and Marshall A. Taylor. 2021. "Cultural Cartography with Word Embeddings." *Poetics* 88 (October): 101567.

Word embeddings

Sociological applications: Semantic change



Davidson 2017, unpublished.

Word embeddings

Sociological applications: Semantic change

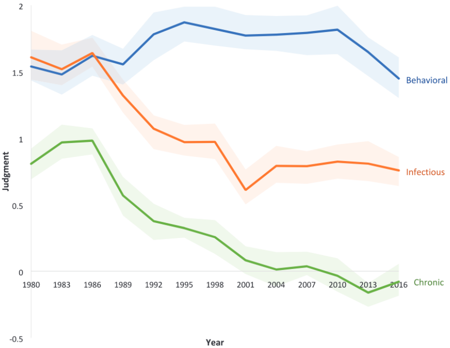


Figure 3. Judgment Scores for Behavioral, Infectious, and Chronic Diseases over Time
Note: More positive scores indicate stronger connotations of immorality and bad personality traits. More negative scores indicate stronger connotations of morality and good personality traits.

Best, Rachel Kahn, and Alina Arseniev-Koehler. 2023. "The Stigma of Diseases: Unequal Burden, Uneven Decline." *American Sociological Review* 88 (5): 938–69.

Word embeddings

Pre-trained word embeddings

- In addition to `word2vec` there are several other popular variants including `GloVe` and `Fasttext` (see Stolz and Taylor 2021)
  - Pre-trained embeddings are available to download so you don't need to train your own
- When to train your own embeddings?
  - The underlying language model / data generating process differ from that represented by existing corpora
    - e.g. Arseniev-Koehler and Foster interested in news coverage specific to health
  - Requires large numbers of documents (> 10k)

Word embeddings

Word embeddings in R

We're going to use the library `word2vec`. The library is a R wrapper around a `C++` library. The the original library can be found here and the R version wrapper here.

```
#install.packages("word2vec")
library(word2vec)
set.seed(8901) # random seed
```

Word embeddings

Training a word2vec model

```
model <- word2vec::word2vec(x = tolower(df$text),
                  type="cbow",# continuous bag of words
                  dim=100, # 100-dim output vectors
                  window = 3L, # window size 3
                  iter = 10L, # train over 10 iterations
                  negative= 10L,# 10 negative samples for each positive
                  min_count = 10L) # Drop words less than 10 times
```

Word embeddings

Getting embeddings for words

We can use the `predict` function to find the nearest words to a given term.

```
predict(model, c("economy"), type = "nearest", top_n = 10)
```

```
## $economy
##      term1     term2 similarity rank
## 1  economy   society  0.7709113    1
## 2  economy    nation  0.7290791    2
## 3  economy    system  0.7249526    3
## 4  economy   country  0.6843896    4
## 5  economy  industry  0.6818392    5
## 6  economy democracy  0.6782054    6
## 7  economy fisheries  0.6745494    7
## 8  economy    planet  0.6675637    8
## 9  economy    future  0.6648591    9
## 10 economy      goal  0.6581267   10
```

Word embeddings

Testing analogical reasoning

```r
emb <- as.matrix(model) # Extracting embedding matrix
vector <- emb["king", ] - emb["man", ] + emb["woman", ]
predict(model, vector, type = "nearest", top_n = 10)
```

```
##         term similarity rank
## 1       king  0.9791069    1
## 2     martin  0.8425124    2
## 3     luther  0.8341991    3
## 4         jr  0.8260822    4
## 5   formally  0.8239747    5
## 6       ross  0.8197786    6
## 7     ahmaud  0.8192527    7
## 8        rbg  0.8139659    8
## 9   williams  0.8111396    9
## 10       tom  0.8102375   10
```

Word embeddings

Testing analogical reasoning

```
vector <- emb["austin", ] - emb["texas", ] + emb["illinois", ]
predict(model, vector, type = "nearest", top_n = 10)
```

```
##          term similarity rank
## 1       lloyd  0.9420799    1
## 2      deputy  0.9087285    2
## 3    illinois  0.9057994    3
## 4   assistant  0.9025223    4
## 5     haaland  0.8996096    5
## 6    mayorkas  0.8974388    6
## 7   inspector  0.8915766    7
## 8      milley  0.8896881    8
## 9     mondale  0.8822240    9
## 10     powell  0.8746590   10
```

Word embeddings

Loading a pre-trained embedding

Let's try another example. I downloaded a pre-trained word embedding model trained on a much larger corpus of English texts. The file is 833MB in size. Following the documentation we can load this model into R.

```
model.pt <- read.word2vec(file = "../data/sg_ns_500_10.w2v", normalize
```

Word embeddings

Similarities

Find the top 10 most similar terms to “economy” in the embedding space.

```
predict(model.pt, c("economy"), type = "nearest", top_n = 10)
```

Word embeddings

Similarities

Find the top 10 most similar terms to “immigration” in the embedding space.

```
predict(model.pt, c("immigration"), type = "nearest", top_n = 10)
```

Word embeddings

Repeating the analogy test

Let's re-try the analogy test. We still don't go great but now queen is in the top 5 results.

```
emb <- as.matrix(model.pt)
vector <- emb["king", ] - emb["man", ] + emb["woman", ]
predict(model.pt, vector, type = "nearest", top_n = 10)
```

Word embeddings

Repeating the analogy test

Let's try another analogy. The correct answer is second. Not bad.

```
vector <- emb["madrid", ] - emb["spain", ] + emb["france", ]
predict(model.pt, vector, type = "nearest", top_n = 10)
```

Word embeddings

Repeating the analogy test

Let's try another slightly more complex analogy. Not bad overall.

```
vector <- (emb["new", ] + emb["jersey", ])/2 - emb["trenton", ] + emb["
predict(model.pt, vector, type = "nearest", top_n = 10)
```

Word embeddings

Representing documents

Last week we focused on how we could represent documents using the rows in the DTM. So far we have just considered how words are represented in the embedding space. We can represent a document by averaging over its composite words.

```
descartes <- (emb["i", ] +
                emb["think", ] +
                emb["therefore", ] +
                emb["i", ] +
                emb["am", ])/5
predict(model.pt, descartes, type = "nearest", top_n = 10)
```

Word embeddings

Representing documents

The package has a function called `doc2vec` to do this automatically. This function includes an additional scaling factor (see documentation) so the results are slightly different.

```
descartes <- doc2vec(model.pt, "i think therefore i am")
predict(model.pt, descartes, type = "nearest", top_n = 10)
```

Word embeddings

Visualizing high-dimensional embeddings in low-dimensional space

- There are various algorithms available for visualizing word-embeddings in low-dimensional space
  - `PCA`, `t-SNE`, `UMAP`
- There are also browser-based interactive embedding explorers
  - See this example on the Tensorflow website

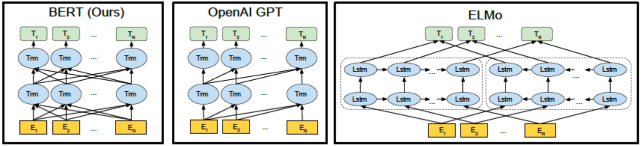
Contextualized embeddings

Limitations of existing approaches

- Word2vec and other embedding methods run into problems when dealing with *polysemy*
  - e.g. The vector for "crane" will be learned by averaging across different uses of the term: bird, construction equipment, movement
  - "She had to crane her neck to see the crane perched on top of the crane".
- New methods have been developed to allow the vector for "crane" to vary according to different contexts
  - Intuition: Meaning varies depending on context
    - e.g. Proximity to "neck" implies "crane" used to describe movement

Contextualized embeddings

Architectures



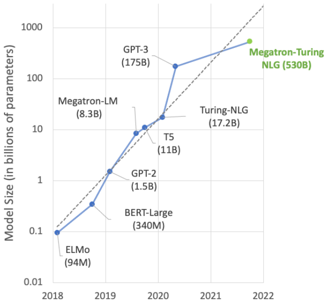
Source: Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." In *Proceedings of NAACL-HLT* 2019, 4171–86. ACL.

Contextualized embeddings

Methodological innovations

- More complex, deeper neural networks
  - *Attention* mechanisms, *LSTM* architecture, *transformers*
- Optimization over multiple tasks (not just a simple prediction problem like Word2vec)
- Character-level tokenization and embeddings
- Much more data and enormous compute power required
  - e.g. BERT trained on a 3.3 billion word corpus over 40 epochs, taking over 4 days to train on 64 TPU chips (each chip costs ~$10k).

Large language models



See Nvidia blog on Megatron-Turing NLG.

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
- Word embeddings use a neural language model to represent texts as dense vectors
  - Distributional semantics
- Recent methodological advances better incorporate context

Next week

- Spring break
- After break
  - Topic modeling