Computational Social Science

Word embeddings I

Dr. Thomas Davidson

Rutgers University

March 4, 2024

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 at 5pm
  - 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

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

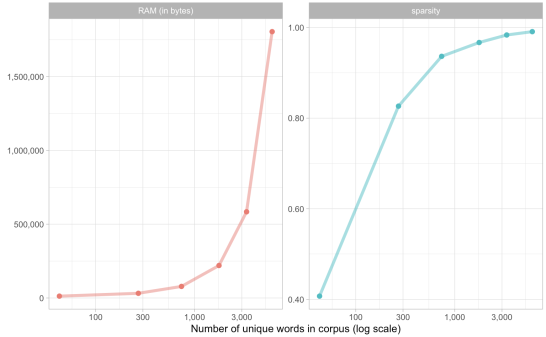
The vector-space model review

Limitations

- These methods produce *high-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

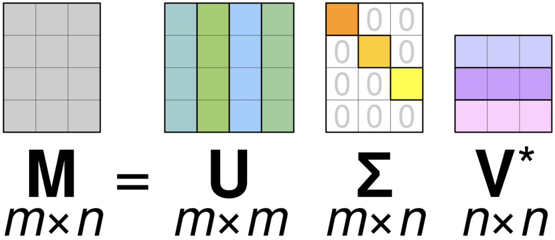
Latent Semantic Analysis

- 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

Example: Sci-fi texts

X is a TF-IDF weighted Document-Term Matrix of sci-fi from Project Gutenberg.

```r
X <- read.csv("../data/scifi_100.csv")
rownames(X) <- X[,1] # Setting rownames
X <- X[, -1] # Deleting rownames as a column
X <- as.matrix(X) # Transforming to matrix
X <- X[, which(colSums(X) != 0)] # Drop zero columns

dim(X)
```

```
## [1] 1129 5853
```

Latent semantic analysis

Creating a lookup dictionary

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

```r
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["monster"]
```

```
## $monster
## [1] 3778
```

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

```
##   [1] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000
##  [10] 0.00000 0.00000 0.00000 0.00000 0.00041 0.00029 0.00000 0.0000
##  [19] 0.00000 0.00107 0.00024 0.00000 0.00088 0.00000 0.00000 0.0020
##  [28] 0.00000 0.00000 0.00000 0.00096 0.00000 0.00027 0.00000 0.0000
##  [37] 0.00000 0.00000 0.00000 0.00000 0.00007 0.00000 0.00000 0.0005
##  [46] 0.00000 0.00000 0.00265 0.00000 0.00000 0.00000 0.00000 0.0002
##  [55] 0.00041 0.00000 0.00000 0.00000 0.00000 0.00015 0.00000 0.0000
##  [64] 0.00000 0.00000 0.00000 0.00000 0.00032 0.00286 0.00000 0.0000
##  [73] 0.00000 0.00000 0.00040 0.00159 0.00116 0.00000 0.00000 0.0000
##  [82] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00116 0.00000 0.0000
```

Latent semantic analysis

Calculating similarties

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] 5853 5853
```

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("fight", sims, n = 3)
```

```
##     fight  fighting    battle
## 1.0000000 0.5201471 0.4623335
```

Latent semantic analysis

Finding similar words

Choose another word to inspect.

```r
get.top.n("", sims, n = 3)
```

Latent semantic analysis

Singular value decomposition

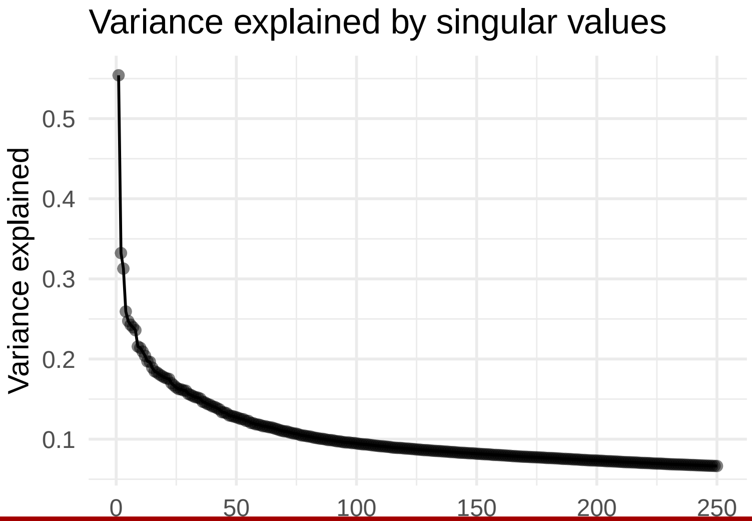
The `svd` function allows us to decompose the DTM. We can then easily reconstruct it using the formula shown above.

```r
# 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



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] 138.3
```

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["fight"])]),
```

```
##  [1] -0.02  0.00  0.00 -0.01  0.00  0.00  0.00  0.00  0.00  0.00  0.
## [13]  0.00  0.00 -0.01  0.01  0.00 -0.01  0.00  0.00 -0.01  0.00  0.
## [25]  0.00  0.00 -0.01 -0.01 -0.02 -0.01  0.00  0.00  0.00  0.01  0.
## [37] -0.01  0.01  0.00  0.00  0.01  0.01  0.01 -0.01 -0.01  0.02  0.
## [49] -0.01  0.01
```

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

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

```
##     fight  fighting    battle    called      left      half       har
## 1.0000000 0.5201471 0.4623335 0.4493276 0.4476139 0.4327410 0.419437
##      feet      head
## 0.4161047 0.4103359
```

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

```
##     fight     ropes   laughed    killed      bars     hairy      laug
## 1.0000000 0.7134000 0.7127284 0.7021199 0.6962668 0.6848260 0.680569
##       red    haired
## 0.6516037 0.6501807
```

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

```
## # A tibble: 10 x 2
##    ...1     ...2
##    <chr>    <chr>
##  1 fight    fight
```

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("fight", sims)
```

```
##       nest    bending      grant      stark      ruler       soap
## 0.01316518 0.01362281 0.01446432 0.01460876 0.01470185 0.01567309 0.
##        toe      apple     trunks
## 0.01837433 0.01887232 0.02017182
```

Latent semantic analysis

Comparing similarities

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

```
## # A tibble: 5 x 2
##   ...1    ...2
##   <chr>   <chr>
## 1 ghost   ghost
## 2 ghosts  adventures
## 3 haunted tale
## 4 tale    issue
## 5 boiling columns
```

Latent semantic analysis

Comparing similarities

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

```
##      love     loved   passion      eyes      life
## 1.0000000 0.4707218 0.4325244 0.4000985 0.3957413
```

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

```
##      love     loved     women childhood    loving
## 1.0000000 0.8438132 0.8314148 0.7949752 0.7757434
```

Latent semantic analysis

Comparing similarities

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

```
##       run    looked      time      half      head      left     calle
## 1.0000000 0.5605993 0.5584137 0.5491457 0.5407950 0.5395652 0.538103
##   started     found
## 0.5307708 0.5276992
```

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

```
##       run   started    pulled      hear      stay      left       har
## 1.0000000 0.8536205 0.8104711 0.8015017 0.7845183 0.7824391 0.779399
##       ten    listen
## 0.7588223 0.7584098
```

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 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"
##        xi      note  evidence willingly withstand
##    -0.002    -0.002    -0.002    -0.003    -0.003
## [1] "Dimension:  2"
##    joe george  james    sam   bill
##  0.779  0.562  0.054  0.045  0.042
## [1] "Dimension:  3"
##     joe     sam   james    mike colonel
```

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

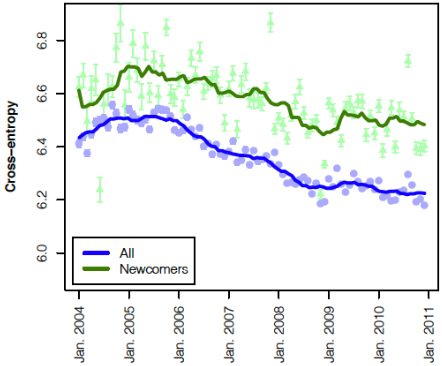
Empirical applications

- 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 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