Lesson 9 - Topics Models

Erin M. Buchanan

02/28/2019

Language Topics Discussed

- Expansion of semantic vector models into Topics Models
- Types of relations
- How to differentiate topics and other models

Topics Background

- What does it take to understand a sentence?
  - Retrieving concepts from memory
  - Dynamic process based on incoming information
  - Use the semantic context to create a “gist” representation

Topics Background

- Pulling the right information from memory can be improved by predicting what concepts are going to be relevant (expectancy generation)
  - For example, *bank* might prime *federal* and *reserve*
  - However, multiple senses can sometimes make this difficult
  - Gist representation allows us to create an overarching topic to disambiguate sense

Topics Background

- Four types of ways to think about relation:
  - Word-concept: knowledge that a word refers to some concept (physical letters dog refer to dog)
  - Concept-concept: knowledge that a concept is related to some other concept (dog is a type of animal)
  - Concept-precept/action: knowledge about what a concept looks like or does (dogs are furry and bark)
  - Word-word: knowledge that the word co-occurs with another word (dog-cat)

Topics Background

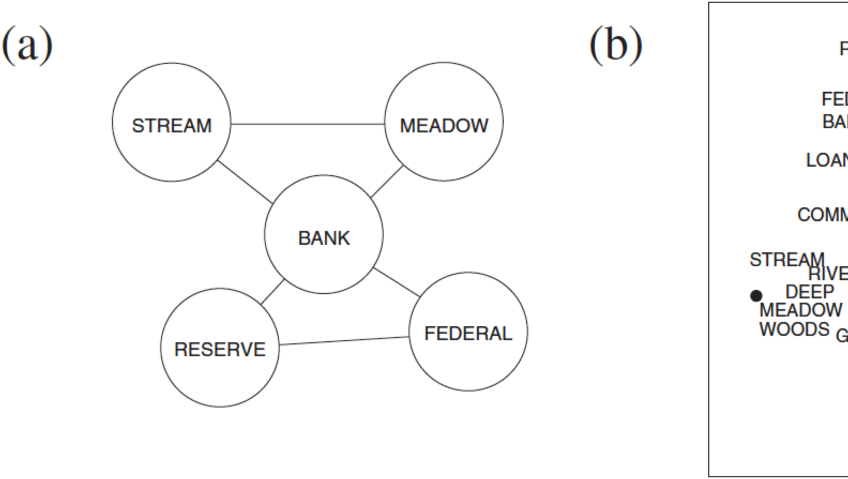


Figure 1. Approaches to semantic represen

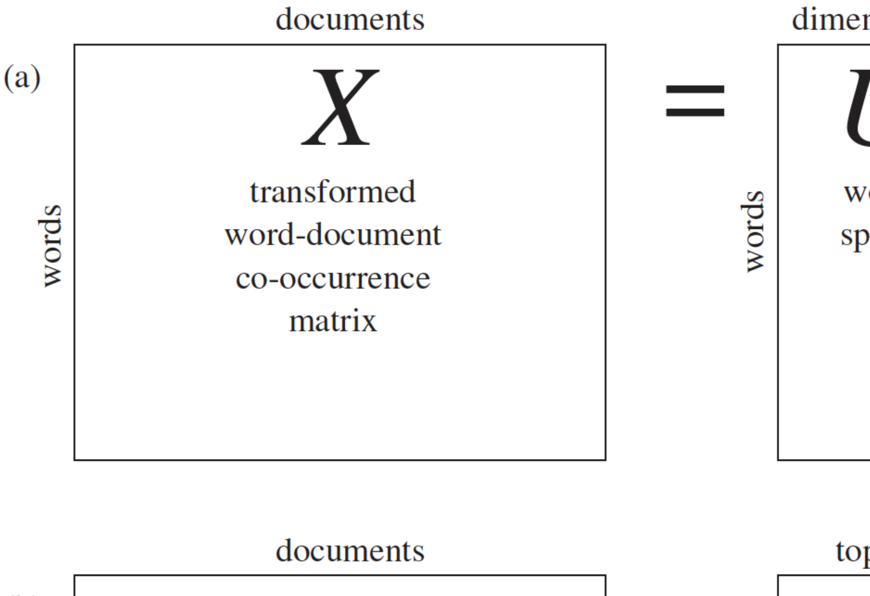
Topics Background

- These are useful to understand, because they predict different ways to think about semantic memory.
- What are people doing when they read a sentence and how can we represent that?
  - Translating words to concepts and using background knowledge to pull in other related concepts
  - Using word co-occurrence to predict the next words

Topics Background

- What do topics models propose people do?
  - Predict: people predict the next word or concept because it helps with retrieval
  - Disambiguation: of senses or meanings of words
  - Gist: creating a coherent representation of the text (mental model not individual words)

Topics Background



Other thoughts

- Topics modeling could allow us to reveal topics present in text
- OR find ways to sort various texts into different groups
- Similar to clustering, classification, finds the natural groups in the corpus

A little bit of math

- Latent Dirichlet allocation (LDA) is the most popular math
- Estimates topics based on the idea that every document includes a mix of topics, and every topic includes a mix of words
- That specification allows topics to overlap, such that they might have some of the same words/content
- LDA is the middle group that finds both the words for each topic and the topics for each document

Getting started with raw data

- First, you would need to load the libraries for the Topic Modeling packages:

```
library(tm)
library(topicmodels)
library(tidyverse)
library(tidytext)
library(slam)
```

Load a dataset or corpus

- Then, you could load a dataset you are interested in working with:

```
importdf = read.csv('exam_answers.csv', header = F, strings
```

Convert to a Corpus

- From these documents, we will create a corpus (a set of text documents).
- Because our data is in one column in our dataset, we will use `VectorSource()` to create the corpus:

```
import_corpus = Corpus(VectorSource(importdf$V1))
```

Clean up the text

- When you perform these analyses, you usually have to edit the text.
- Therefore, we are going to lower case the words, take out the punctuation, and remove English stop words (like *the, an, a*).
- This step will also transform the documents in a term (words) by document matrix.

Clean up the text

```
import_mat =
  DocumentTermMatrix(import_corpus,
           control = list(stemming = TRUE, #create root wo
                          stopwords = TRUE, #remove stop wo
                          minWordLength = 3, #cut out smal
                          removeNumbers = TRUE, #take out t
                          removePunctuation = TRUE)) #take
```

Weight the matrix

- Then you would want to weight that matrix to help control for the sparsity of the matrix.
- That means you are controlling for the fact that not all words are in each document, as well as the fact that some words are very frequent.
- Then you usually ignore very frequent words and words with zero frequency.

Weight the matrix

```
#weight the space
import_weight = tapply(import_mat$v/row_sums(import_mat)[in
                       import_mat$j,
                       mean) *
  log2(nDocs(import_mat)/col_sums(import_mat > 0))

#ignore very frequent and 0 terms
import_mat = import_mat[ , import_weight >= .1]
import_mat = import_mat[ row_sums(import_mat) > 0, ]
```

Parameters and terms

- Alpha: a measure of the number of topics; low scores indicate a few dominant topics per document, high scores indicate more
- Beta: a measure of the number of words, low scores indicate each topic only composes of a few words
- Gamma: probability of that topic in that document
- Entropy: a measure of randomness

A bit more math

- There are several model types:
  - The LDA Fit model is an analysis with VEM (variational expectation-maximization) algorithm and estimating an alpha.
    - The LDA Fixed model using the VEM algorithm with a fixed alpha value.
    - Last, the LDA Gibbs option uses a Gibbs (Bayesian) algorithm to fit the data.
    - CTM stands for correlated topics models, which allows the correlation between topics, and this method uses a VEM algorithm.

Run the models

- First, you will pick a number of expected topics - which is the k option.
- The SEED should be a random number to start the analysis on.

```
k = 3 #set the number of topics

SEED = 2010 #set a random number

LDA_fit = LDA(import_mat, k = k,
              control = list(seed = SEED))

LDA_fixed = LDA(import_mat, k = k,
                control = list(estimate.alpha = FALSE, seed
```

Run the models

```
LDA_gibbs = LDA(import_mat, k = k, method = "Gibbs",
                control = list(seed = SEED, burnin = 1000,
                               thin = 100, iter = 1000))

CTM_fit = CTM(import_mat, k = k,
              control = list(seed = SEED,
                             var = list(tol = 10^-4),
                             em = list(tol = 10^-3)))
```

Get the alpha values

- You can then get the alpha values, and smaller alpha values indicate higher percentages of documents that were classified to one single topic.

```
LDA_fit@alpha
```

```
## [1] 0.05846617
```

```
LDA_fixed@alpha
```

```
## [1] 16.66667
```

```
LDA_gibbs@alpha
```

```
## [1] 16.66667
```

Get the entropy values

- You can also get entropy values where higher values indicate that topics are evenly spread.

```
sapply(list(LDA_fit, LDA_fixed, LDA_gibbs, CTM_fit),
       function (x)
         mean(apply(posterior(x)$topics, 1, function(z) - s
```

```
## [1] 0.2402260 1.0920326 1.0946644 0.5894953
```

The actual topics

- The topic matrix indicates the rank of the number of topics for each document.
- For instance, if you select to estimate 5 topics, you will see see which topic is covered most in each document, with less covered topics ranked lower.
- Therefore, a score set of 5, 3, 1, 2, 4 indicates that the 5th topic was covered most in that document, and the 4th topic was covered least.

The actual topics

```
topics(LDA_fit, k)
```

```
##      1 2 3 4 5 6 7 8 9 10 11 14 15 16 17 18 19 20 21 22
## [1,] 3 3 1 3 3 1 1 1 2  1  1  2  3  2  3  1  3  3  1  2
## [2,] 2 1 3 1 1 2 3 2 1  3  2  1  1  1  2  2  1  1  2  1
## [3,] 1 2 2 2 2 3 2 3 3  2  3  3  2  3  1  3  2  2  3  3
##      30 31 32 33 34 35 36 37 38 39 40 41 42
## [1,]  1  1  1  3  2  2  3  2  2  1  2  2  2
## [2,]  2  3  3  1  3  1  1  1  1  2  1  1  3
## [3,]  3  2  2  2  1  3  2  3  3  3  3  3  1
```

The actual topics

```r
topics(LDA_gibbs, k)
```

```
##       1 2 3 4 5 6 7 8 9 10 11 14 15 16 17 18 19 20 21 22
## [1,] 2 3 1 2 3 1 1 2 2  2  2  2  2  1  1  1  3  1  2  2
## [2,] 1 1 2 1 1 2 2 1 1  1  3  1  1  2  2  2  1  2  3  1
## [3,] 3 2 3 3 2 3 3 3 3  3  1  3  3  3  3  3  2  3  1  3
##      30 31 32 33 34 35 36 37 38 39 40 41 42
## [1,]  1  3  1  1  2  3  2  2  2  3  3  1  2
## [2,]  2  1  2  3  1  1  1  1  3  1  1  2  1
## [3,]  3  2  3  2  3  2  3  3  1  2  2  3  3
```

```r
##you can do all of them saving space
#topics(LDA_fixed, k)
#topics(CTM_fit, k)
```

The terms for topics

- You can get the most frequent terms for each of the topics that were estimated.

```r
terms(LDA_fit,10)
```

```
##       Topic 1    Topic 2    Topic 3
##  [1,] "top"      "element"  "bike"
##  [2,] "actual"   "observ"   "mean"
##  [3,] "previous" "gorilla"  "success"
##  [4,] "surround" "black"    "ride"
##  [5,] "environ"  "line"     "gorilla"
##  [6,] "dont"     "screen"   "red"
##  [7,] "discuss"  "demonstr" "dont"
##  [8,] "therefor" "flash"    "your"
##  [9,] "hand"     "lead"     "environ"
## [10,] "rememb"   "monitor"  "effici"
```

The terms for topics

```
terms(LDA_gibbs,10)
```

```
##       Topic 1    Topic 2            Topic 3
##  [1,] "mean"     "gorilla"          "bike"
##  [2,] "dont"     "observ"           "success"
##  [3,] "environ"  "surround"         "ride"
##  [4,] "element"  "hand"             "actual"
##  [5,] "top"      "therefor"         "previous"
##  [6,] "rememb"   "your"             "red"
##  [7,] "screen"   "attentioncontrol" "black"
##  [8,] "act"      "demonstr"         "discuss"
##  [9,] "although" "lead"             "line"
## [10,] "close"    "neutral"          "monitor"
```

```
##again you can do all of them
#terms(LDA_fixed, 10)
#terms(CTM_fit,10)
```

Make some pretty plots

```r
#use tidyverse to clean up the the fit
LDA_fit_topics = tidy(LDA_fit, matrix = "beta")

#create a top terms
top_terms = LDA_fit_topics %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)
```

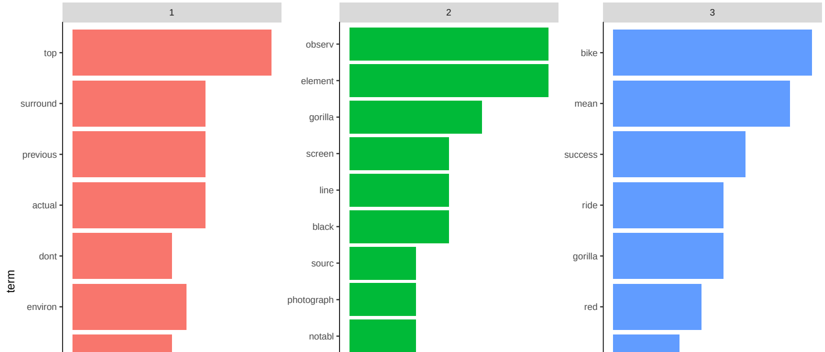
Make some pretty plots

- Some code to clean up the ggplot2 defaults

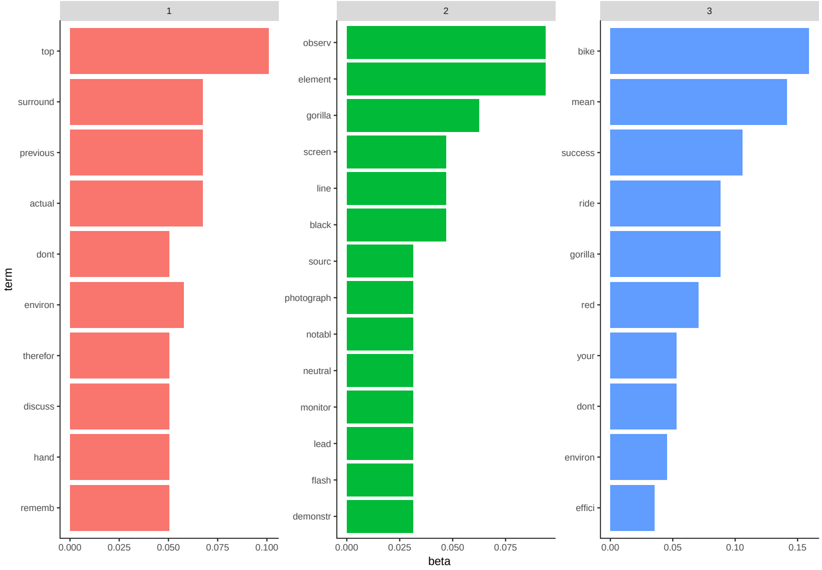
```
cleanup = theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank(),
                panel.background = element_blank(),
                axis.line.x = element_line(color = "black"),
                axis.line.y = element_line(color = "black"),
                legend.key = element_rect(fill = "white"),
                text = element_text(size = 10))
```

Make some pretty plots

```
#make the plot
top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  cleanup +
  coord_flip()
```



The plot



1
2
3
term
beta
top
surround
previous
actual
dont
environ
therefor
discuss
hand
rememb
observ
element
gorilla
screen
line
black
sourc
photograph
notabl
neutral
monitor
lead
flash
demonstr
bike
mean
success
ride
gorilla
red
your
dont
environ
effici
0.000
0.025
0.050
0.075
0.100
0.000
0.025
0.050
0.075
0.00
0.05
0.10
0.15

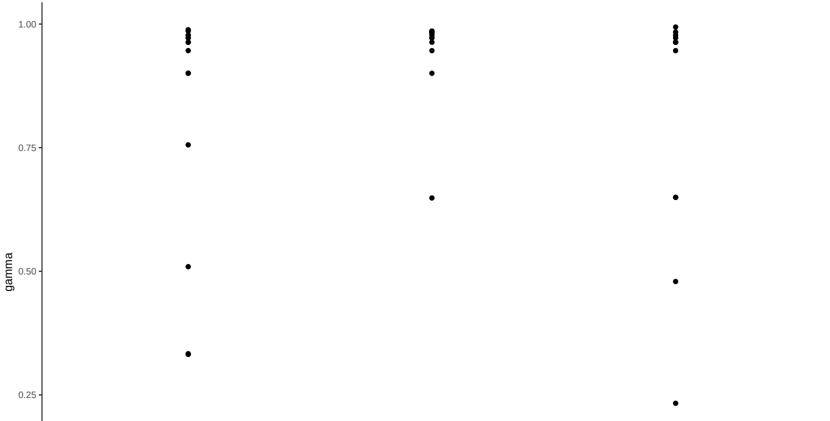
Document classification

- We saw earlier with the `topics()` function, we could figure out the most to least likely topics.
- This matrix is organized by gamma, which is the probability of that topic in for each document.
- Let's visualize those numbers.

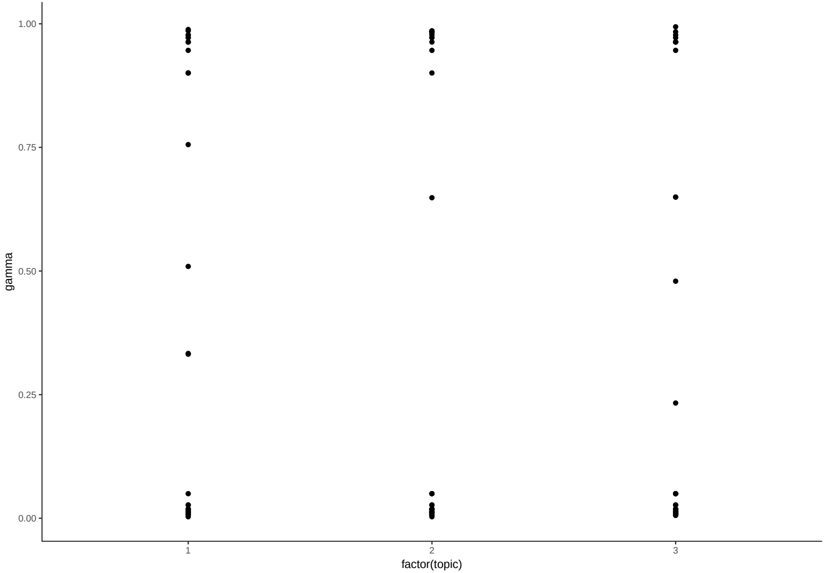
Document classification

```
LDA_gamma = tidy(LDA_fit, matrix = "gamma")

LDA_gamma %>%
  ggplot(aes(factor(topic), gamma)) +
  geom_point() +
  cleanup
```



Document classification



Summary

- We explored how the theoretical background for topics models is different than other semantic vector space models.
- We talked about how to build topics models with various settings.
- We talked about the output you can pull from a topic model.
- Extensions can be made to unsupervised classification and clustering.