LDA

Noel Johnson

2023-03-31

Topic modeling

In text mining, we often have collections of documents, such as blog posts or news articles, that we'd like to divide into natural groups so that we can understand them separately. Topic modeling is a method for unsupervised classification of such documents, similar to clustering on numeric data, which finds natural groups of items even when we're not sure what we're looking for.

Latent Dirichlet allocation (LDA) is a particularly popular method for fitting a topic model. It treats each document as a mixture of topics, and each topic as a mixture of words. This allows documents to "overlap" each other in terms of content, rather than being separated into discrete groups, in a way that mirrors typical use of natural language.

In this chapter, we'll learn to work with `LDA` objects from the topicmodels package, particularly tidying such models so that they can be manipulated with ggplot2 and dplyr. We'll also explore an example of clustering chapters from several books, where we can see that a topic model "learns" to tell the difference between the four books based on the text content.

Latent Dirichlet allocation

Latent Dirichlet allocation is one of the most common algorithms for topic modeling. Without diving into the math behind the model, we can understand it as being guided by two principles.

- **Every document is a mixture of topics.** We imagine that each document may contain words from several topics in particular proportions. For example, in a two-topic model we could say "Document 1 is 90% topic A and 10% topic B, while Document 2 is 30% topic A and 70% topic B."
- **Every topic is a mixture of words.** For example, we could imagine a two-topic model of American news, with one topic for "politics" and one for "entertainment." The most common words in the politics topic might be "President", "Congress", and "government", while the entertainment topic may be made up of words such as "movies", "television", and "actor". Importantly, words can be shared between topics; a word like "budget" might appear in both equally.

LDA is a mathematical method for estimating both of these at the same time: finding the mixture of words that is associated with each topic, while also determining the mixture of topics that describes each document. There are a number of existing implementations of this algorithm, and we'll explore one of them in depth.

the `AssociatedPress` dataset provided by the topicmodels package, as an example of a DocumentTermMatrix. This is a collection of 2246 news articles from an American news agency, mostly published around 1988.

```
library(topicmodels)

data("AssociatedPress")
AssociatedPress
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

We can use the `LDA()` function from the topicmodels package, setting `k = 2`, to create a two-topic LDA model.

Almost any topic model in practice will use a larger `k`, but we will soon see that this analysis approach extends to a larger number of topics.

This function returns an object containing the full details of the model fit, such as how words are associated with topics and how topics are associated with documents.

```r
# set a seed so that the output of the model is predictable
ap_lda <- LDA(AssociatedPress, k = 2, control = list(seed =
ap_lda
```

```
## A LDA_VEM topic model with 2 topics.
```

Fitting the model was the “easy part”: the rest of the analysis will involve exploring and interpreting the model using tidying functions from the tidytext package.

Word-topic probabilities

The tidytext package provides this method for extracting the per-topic-per-word probabilities, called β ("beta"), from the model.

```
library(tidytext)

ap_topics <- tidy(ap_lda, matrix = "beta")
ap_topics
```

```
## # A tibble: 20,946 x 3
##    topic term           beta
##    <int> <chr>         <dbl>
##  1     1 aaron      1.69e-12
##  2     2 aaron      3.90e- 5
##  3     1 abandon    2.65e- 5
##  4     2 abandon    3.99e- 5
##  5     1 abandoned  1.39e- 4
##  6     2 abandoned  5.88e- 5
##  7     1 abandoning 2.45e-33
##  8     2 abandoning 2.34e- 5
##  9     1 abbott     2.13e- 6
```

Notice that this has turned the model into a one-topic-per-term-per-row format. For each combination, the model computes the probability of that term being generated from that topic. For example, the term "aaron" has a 1.686917×10^{-12} probability of being generated from topic 1, but a 3.8959408×10^{-5} probability of being generated from topic 2.

We could use dplyr's `slice_max()` to find the 10 terms that are most common within each topic. As a tidy data frame, this lends itself well to a ggplot2 visualization (Figure @ref(fig:aptoptermsplot)).

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
ap_top_terms <- ap_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

ap_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
```

As an alternative, we could consider the terms that had the *greatest difference* in β between topic 1 and topic 2. This can be estimated based on the log ratio of the two: $\log_2(\frac{\beta_2}{\beta_1})$ (a log ratio is useful because it makes the difference symmetrical: β_2 being twice as large leads to a log ratio of 1, while β_1 being twice as large results in -1). To constrain it to a set of especially relevant words, we can filter for relatively common words, such as those that have a β greater than 1/1000 in at least one topic.

```r
library(tidyr)

beta_wide <- ap_topics %>%
  mutate(topic = paste0("topic", topic)) %>%
  pivot_wider(names_from = topic, values_from = beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2(topic2 / topic1))

beta_wide
```

```
## # A tibble: 198 x 4
##    term              topic1      topic2 log_ratio
##    <chr>              <dbl>       <dbl>     <dbl>
##  1 administration 0.000431  0.00138          1.68
##  2 ago            0.00107   0.000842        -0.339
##  3 agreement      0.000671  0.00104          0.630
##  4 aid            0.0000476 0.00105          4.46
##  5 air            0.00214   0.000297        -2.85
##  6 american       0.00203   0.00168         -0.270
##  7 analysts       0.00109   0.000000578   -10.9
```

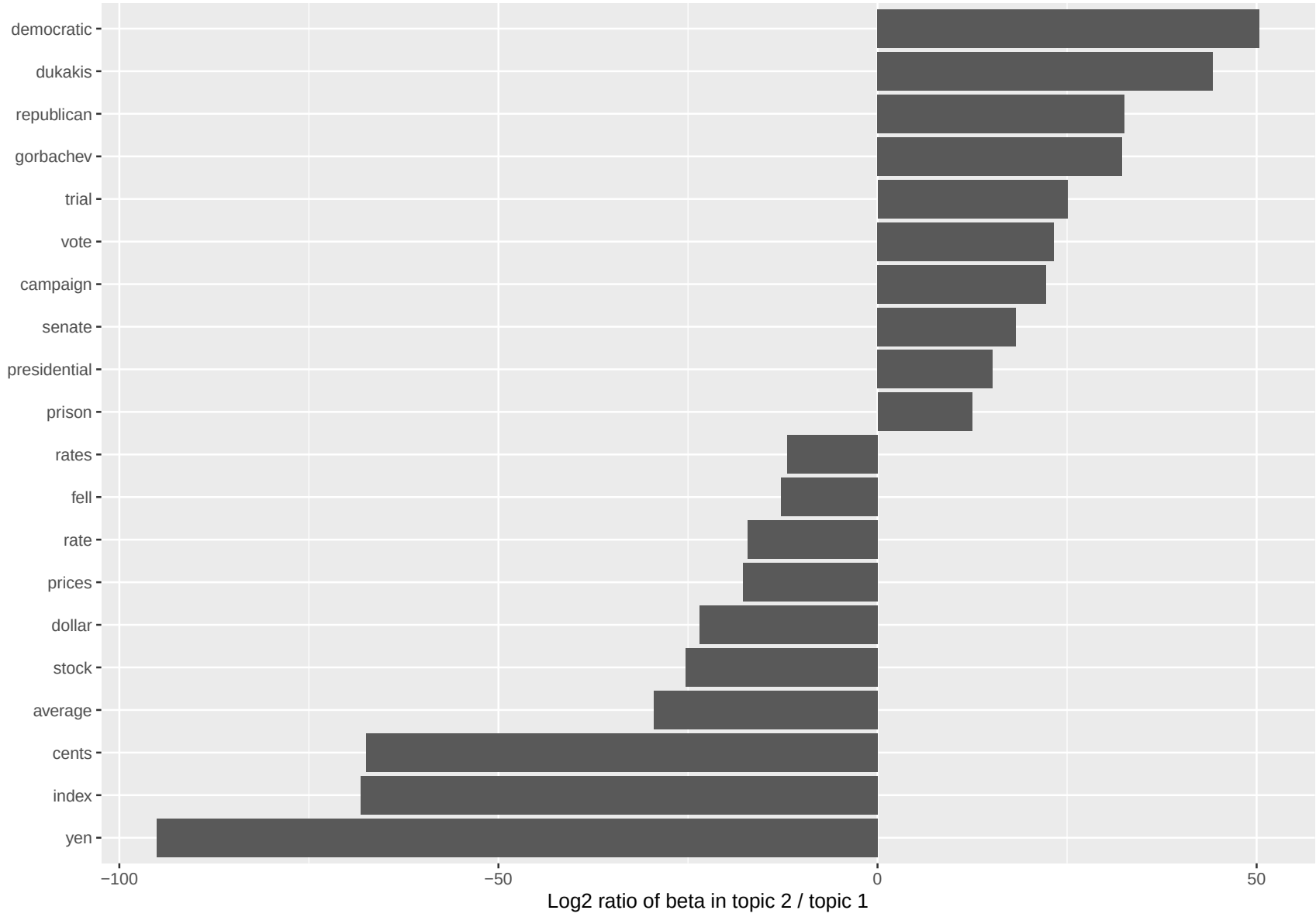

Figure 2: (ref:topiccap)

Document-topic probabilities

Besides estimating each topic as a mixture of words, LDA also models each document as a mixture of topics. We can examine the per-document-per-topic probabilities, called γ ("gamma"), with the `matrix = "gamma"` argument to `tidy()`.

```
ap_documents <- tidy(ap_lda, matrix = "gamma")
ap_documents
```

```
## # A tibble: 4,492 x 3
##    document topic    gamma
##       <int> <int>    <dbl>
##  1        1     1 0.248
##  2        2     1 0.362
##  3        3     1 0.527
##  4        4     1 0.357
##  5        5     1 0.181
##  6        6     1 0.000588
##  7        7     1 0.773
##  8        8     1 0.00445
##  9        9     1 0.967
## 10       10     1 0.147
## # ... with 4,482 more rows
```

We can see that many of these documents were drawn from a mix of the two topics, but that document 6 was drawn almost entirely from topic 2, having a γ from topic 1 close to zero. To check this answer, we could `tidy()` the document-term matrix (see Chapter @ref(tidy-dtm)) and check what the most common words in that document were.

```
tidy(AssociatedPress) %>%
  filter(document == 6) %>%
  arrange(desc(count))
```

```
## # A tibble: 287 x 3
##    document term           count
##       <int> <chr>          <dbl>
##  1        6 noriega           16
##  2        6 panama            12
##  3        6 jackson            6
##  4        6 powell             6
##  5        6 administration     5
##  6        6 economic           5
##  7        6 general            5
##  8        6 i                  5
##  9        6 panamanian         5
## 10        6 american           4
## # ... with 277 more rows
```

Based on the most common words, this appears to be an article about the relationship between the American government and Panamanian dictator Manuel Noriega, which means the algorithm was right to place it in topic 2 (as political/national news).

Example: the great library heist

When examining a statistical method, it can be useful to try it on a very simple case where you know the "right answer". For example, we could collect a set of documents that definitely relate to four separate topics, then perform topic modeling to see whether the algorithm can correctly distinguish the four groups. This lets us double-check that the method is useful, and gain a sense of how and when it can go wrong. We'll try this with some data from classic literature.

Suppose a vandal has broken into your study and torn apart four of your books:

- *Great Expectations* by Charles Dickens
- *The War of the Worlds* by H.G. Wells
- *Twenty Thousand Leagues Under the Sea* by Jules Verne
- *Pride and Prejudice* by Jane Austen

end code