Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 16

28 April 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO | Dipartimento di Scienze Economiche

Outline

Converting to and from Nontidy Formats

Topic Modeling

Latent Dirichlet Allocation

Packages: **tidytext, tm, topicmodels**

Functions: `tidytext::cast_dtm()`, `tidytext::tidy()`

`tm::Terms()`, `tm::inspect()`, `topicmodels::LDA()`

Converting to and from Nontidy Formats

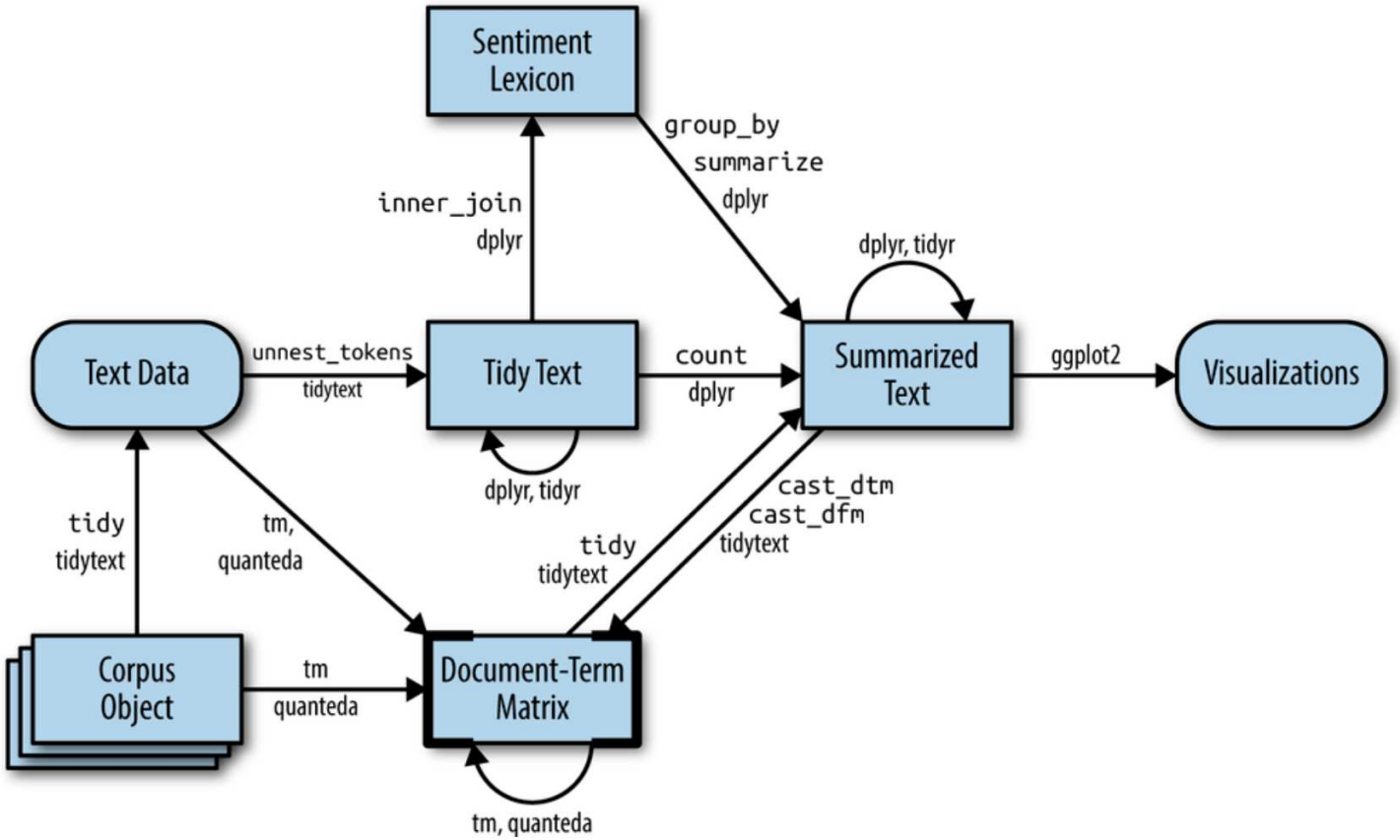
Most of existing R tools for natural language processing (besides the tidytext package) are not compatible with the tidy text format. They take other structures of input and provide nontidy outputs.

In general, an analysis might switch between tidy and nontidy data structures and tools.

Corpus These types of objects typically contain raw strings annotated with additional metadata and details.

Document-term matrix (DTM) This is a sparse matrix describing a collection of documents with one row for each document and one column for each term. The value in the matrix is typically word count. Typically, this matrix contains a lot of zeros

Converting to and from Nontidy Formats



Converting to and from Nontidy Formats

The tidytext package provides two verbs that convert between the two formats:

`tidy()` turns a DTM into a tidy data frame. This verb comes from the **broom** package, which provides several tidying functions for many statistical models and objects.

`cast_dtm()` turns a tidy data frame into a DTM

`cast_dfm()` turns a tidy data frame into a DFM

Converting to and from Nontidy Formats

Consider the collection of Associated Press newspaper articles included in the topicmodels package. This is a collection of 2246 news articles from an American news agency, mostly published around 1988.

```
> library(tm)
> library(tidyverse)
> library(tidytext)
> data("AssociatedPress", package = "topicmodels")
> AssociatedPress
<<DocumentTermMatrix (documents: 2246, terms: 10473)>>
Non-/sparse entries: 302031/23220327
Sparsity           : 99%
Maximal term length: 18
Weighting          : term frequency (tf)
```

Converting to and from Nontidy Formats

To display the DTM, we shall use the function **tm**::`inspect()`

```
> inspect(AssociatedPress[1:10,1:10])
<<DocumentTermMatrix (documents: 10, terms: 10)>>
Non-/sparse entries: 0/100
Sparsity           : 100%
Maximal term length: 10
Weighting          : term frequency (tf)
Sample             :
       Terms
Docs    aaron abandon abandoned abandoning abbott abboud abc abcs abctvs abdomen
   [1,]     0       0         0          0      0      0   0    0      0       0
   [2,]     0       0         0          0      0      0   0    0      0       0
   [3,]     0       0         0          0      0      0   0    0      0       0
   [4,]     0       0         0          0      0      0   0    0      0       0
   [5,]     0       0         0          0      0      0   0    0      0       0
   [6,]     0       0         0          0      0      0   0    0      0       0
   [7,]     0       0         0          0      0      0   0    0      0       0
   [8,]     0       0         0          0      0      0   0    0      0       0
   [9,]     0       0         0          0      0      0   0    0      0       0
  [10,]     0       0         0          0      0      0   0    0      0       0
```

Converting to and from Nontidy Formats

To access the terms we shall use the function **tm**::`Terms()`, which is a function used to access terms of a DTM

```
> terms = Terms(AssociatedPress)
> head(terms)
[1] "aaron"      "abandon"    "abandoned"  "abandoning" "abbott"     "abboud"
```

In order to analyze this data with tidy tools, we need first to turn it into the tidy text format (one token per document per row).

Converting to and from Nontidy Formats

The function `tidytext::tidy()` turns an object into a tidy tibble.

```
tidy(x,...)
```

`x`	An object to be converted into a tidy
…	Additional arguments to tidying method

Converting to and from Nontidy Formats

```
> ap_td = tidy(AssociatedPress)
> ap_td
# A tibble: 302,031 x 3
   document term        count
      <int> <chr>       <dbl>
 1        1 adding          1
 2        1 adult           2
 3        1 ago             1
 4        1 alcohol         1
 5        1 allegedly       1
 6        1 allen           1
 7        1 apparently      2
 8        1 appeared        1
 9        1 arrested        1
10        1 assault         1
# ... with 302,021 more rows
```

We obtain a tidy three column tibble, with variables document, term and count. Notice that only nonzero values are included in the tidied output.

Converting to and from Nontidy Formats

We might also need to transform tidy data into DTM. To this purpose, we need to use the **tidytext**::`cast_dtm()` function.

```
cast_dtm(data, document, term, value, ...)
```

`data`	Table with one-term-per-document-per-row
`document`	Column containing document IDs as string or symbol
`term`	Column containing terms as string or symbol
`value`	Column containing values as string or symbol

Converting to and from Nontidy Formats

For example, assume we want to transform the tidied AP dataset into a DTM.

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

This casting process allows for reading, filtering, and processing to be done using tidy tools, after which data can be converted into a DTM for other applications.

Topic Modeling

Problem: we often have large collections of documents (articles, blog posts), that we would like to divide into **natural groups**. This would allow effectively using such collections in a more structured way: finding documents similar to those of interest, and exploring the collection through the underlying topics.
This structure is not readily available and the size preclude us from building it by hand. **Automated** methods of organizing, managing, and delivering contents are needed.

Topic models are probabilistic models for uncovering the underlying structure of a document collection.
It is a suite of algorithms that aims to discover and annotate large archives of documents with thematic information. These algorithms are based on statistical methods that analyze the words of the original texts to discover the themes that run through them.

Topic modeling is a method for **unsupervised classification** as it does not require any prior annotations or labeling of the documents – the topics emerge from the analysis of the original texts.

Topic Modeling

The text miner inputs the number of topics as the «K» parameter. The topic modeling algorithm assigns a probability from all observed topics to each document. With K=4, each document would get four probabilities.

The topic modeling approach seeks to answer two basic questions:

1) How to decide if a specific word is part of a particular topic among other topics?

2) How common is a particular topic within a document?

Latent Dirichlet allocation (LDA)

LDA is a popular method for fitting a topic model. It treats each document as a mixture of topics, and each topic as a mixture of words.

LDA is 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. 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 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.

Latent Dirichlet allocation (LDA)

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.

It is considered **latent** because the modeling technique identifies hidden topics that are not explicitly defined by the text miner. It seeks to find the hidden group of words that represent each topic.

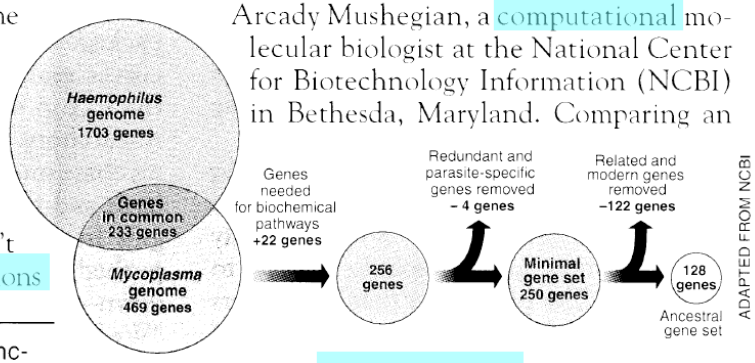
Dirichlet distributions are used to model the word **allocations**.

Latent Dirichlet allocation (LDA)

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an



Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

By hand we have highlighted different words that are used in the article.

Words about data analysis are highlighted in blue; words about evolutionary biology in pink; words about genetics in yellow.

LDA is a statistical model of document collections that tries to capture this intuition.

Latent Dirichlet allocation (LDA)

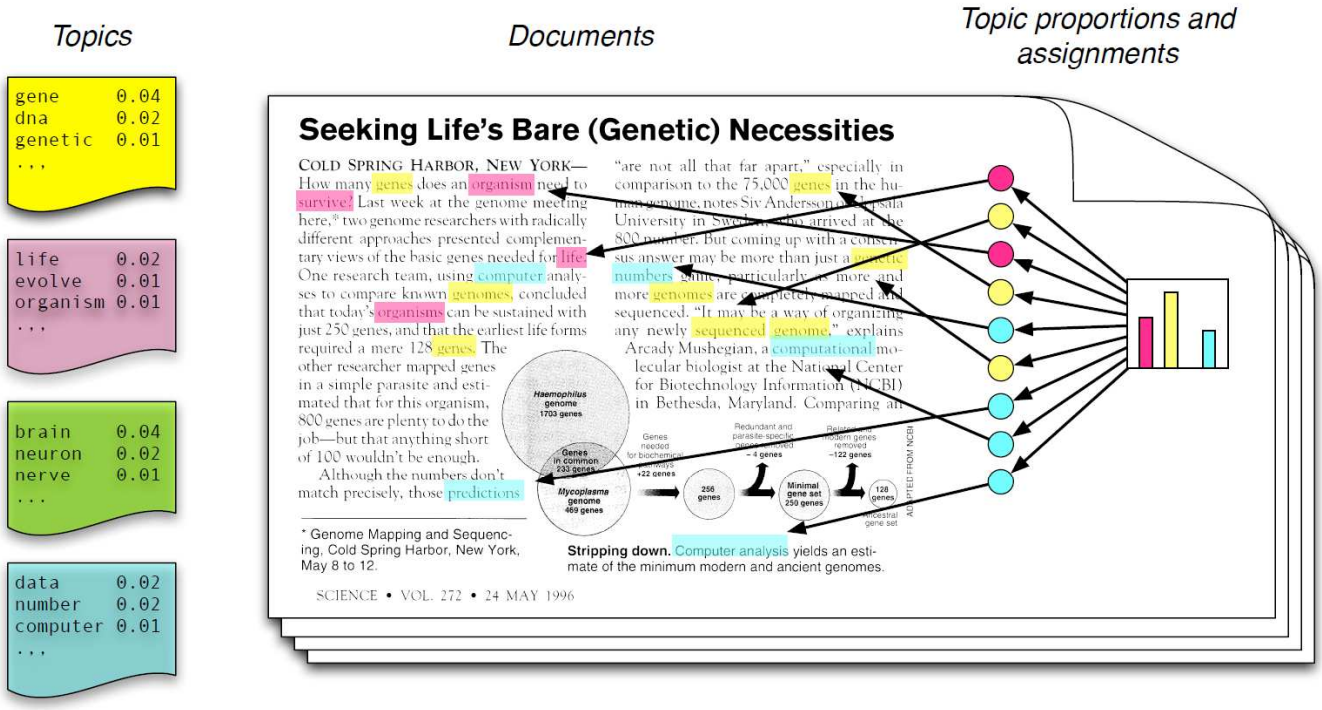
LDA is most easily described by its **generative process**, the imaginary random process by which the model assumes the documents arose.

Topic is defined as a distribution over a fixed vocabulary. It is assumed that *K* topics are associated with a collection and that each document exhibits these topics with different proportions.
The challenge is that these topics are not known in advance. The goal is to learn them from the data.

LDA casts this intuition into a **hidden variable model** of documents. Hidden variable models are structured distributions in which observed data interact with hidden random variables. With a hidden variable model, the researcher posits a hidden structure on the observed data and then learns about that structure using posterior probabilistic inference.

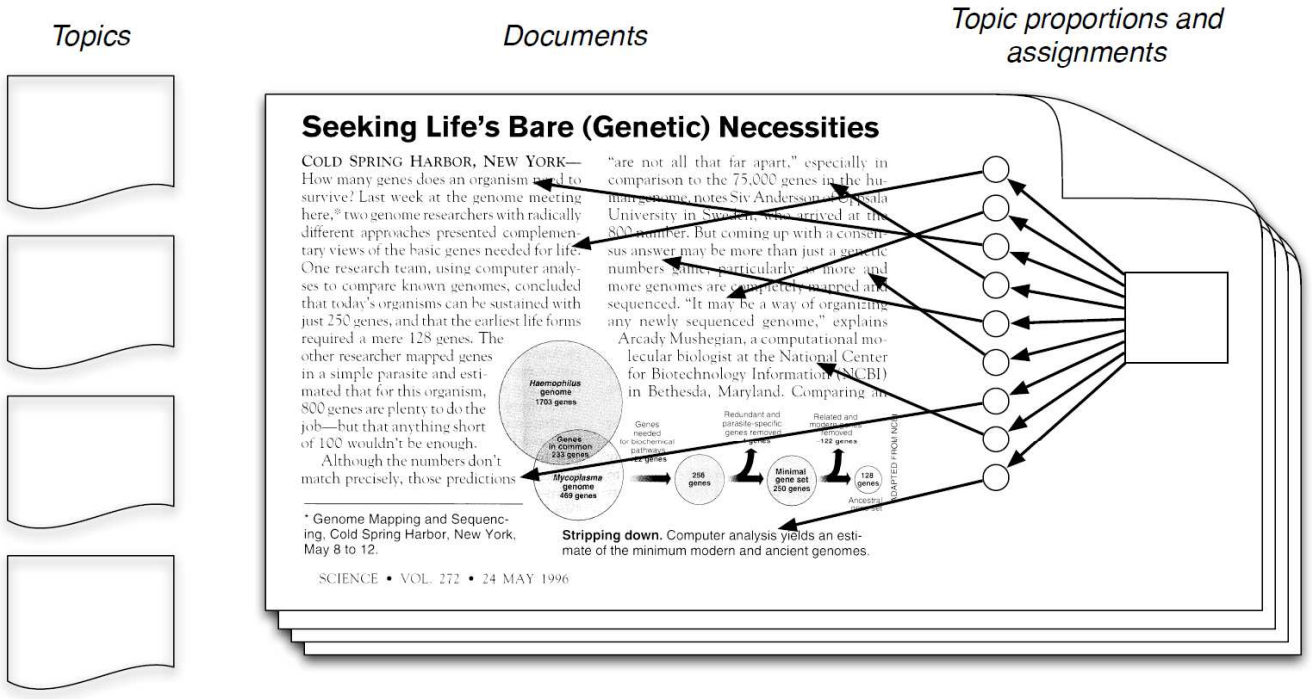
In LDA, the observed data are the words of each document and the hidden variables represent the latent topical structure (i.e. the topics themselves and how each document exhibits them). Given a collection, the posterior distribution of the hidden variables given the observed documents determines a hidden topical decomposition of the collection.

Latent Dirichlet allocation (LDA)



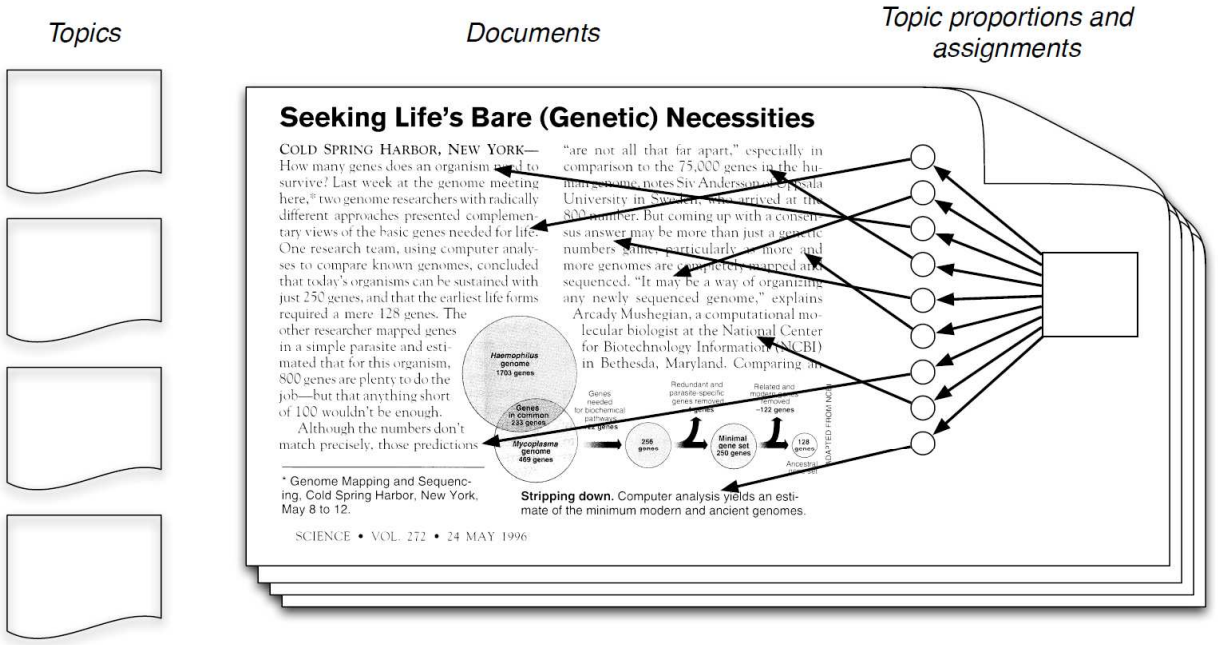
- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

Latent Dirichlet allocation (LDA)



- In reality, we only observe the documents
- The other structure are **hidden variables**

Latent Dirichlet allocation (LDA)

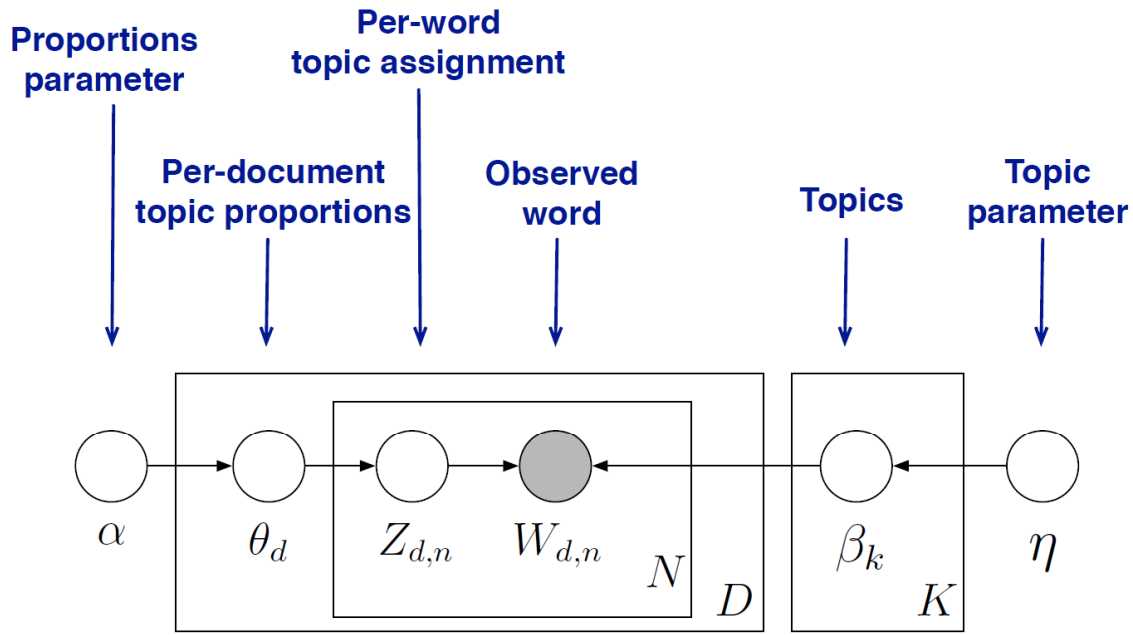


- Our goal is to **infer** the hidden variables
- I.e., compute their distribution conditioned on the documents

$$p(\text{topics, proportions, assignments} \mid \text{documents})$$

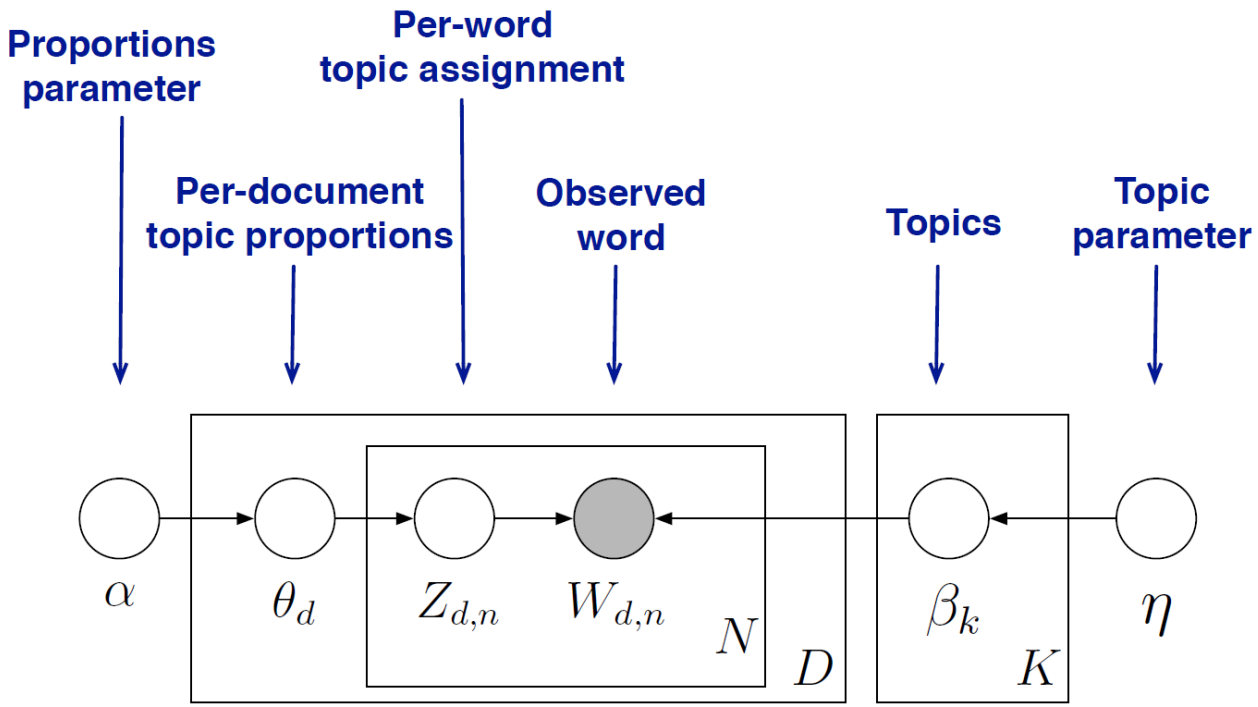
Latent Dirichlet allocation (LDA)

Probabilistic graphical model for LDA



- Nodes are random variables; edges indicate dependence.
- Shaded nodes are observed.
- Plates indicate replicated variables.

Latent Dirichlet allocation (LDA)



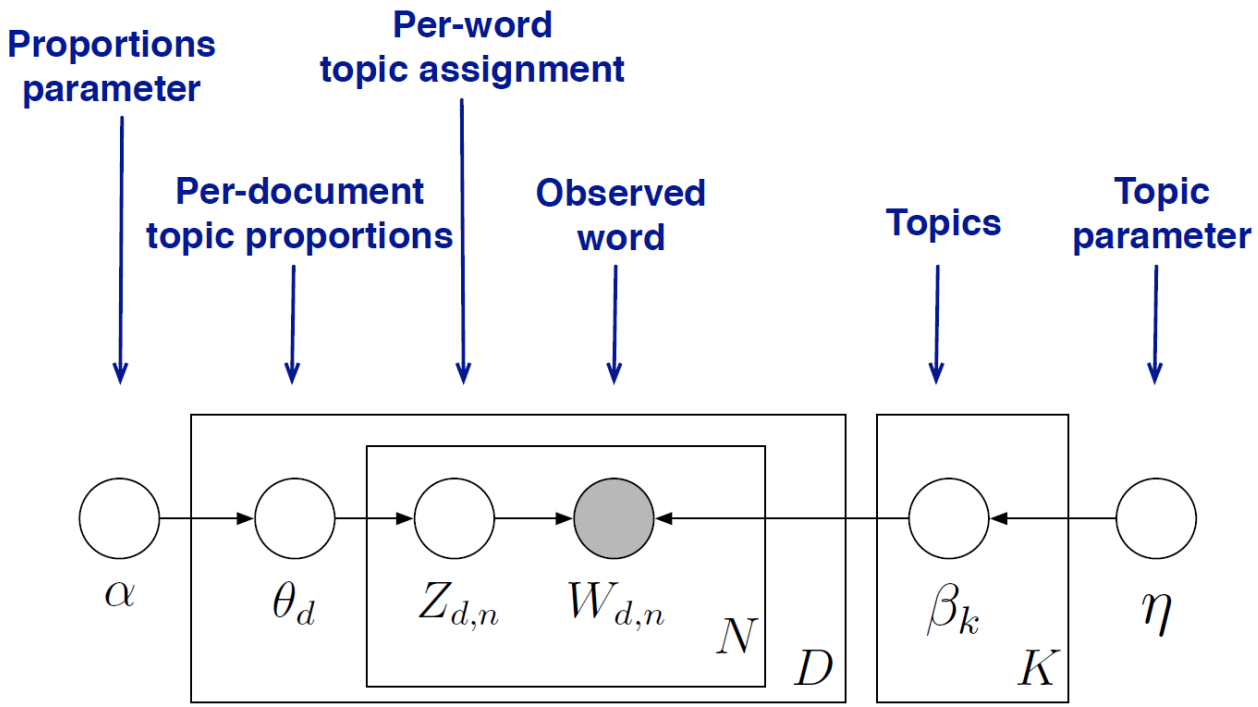
For each document, we generate the words in a two-stage process:

1) Randomly choose a distribution over topics
2) For each word in the document:

   2a) randomly choose a topic from the distribution over topics in step 1

   2b) randomly choose a word from the corresponding distribution over the vocabulary.

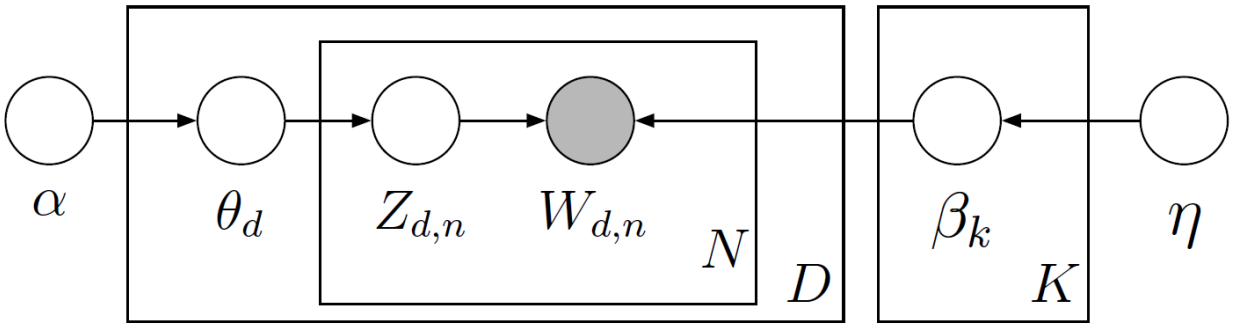
Latent Dirichlet allocation (LDA)



Joint distribution of the hidden and observed variables:

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \left(\prod_{n=1}^{N} p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right)$$

Latent Dirichlet allocation (LDA)



- This joint defines a posterior.
- From a collection of documents, infer
  - Per-word topic assignment $z_{d,n}$
  - Per-document topic proportions θ_d
  - Per-corpus topic distributions β_k

Latent Dirichlet allocation (LDA)

To perform topic modeling we use the `LDA()` function from the **topicmodels** package. This produces LDA objects. Next, we tidy such models so that they can be manipulated with tidy tools.

The function `LDA()` estimates a LDA model.

```
LDA(x, k, method = "VEM", control = NULL,...)
```

`x`	Object of class "DocumentTermMatrix" with term-frequency weighting.
`k`	Integer; number of topics.
`method`	The method to be used for fitting; currently method = "VEM" or method= "Gibbs" are supported.
`control`	A named list of the control parameters for estimation.

Data – working example

The working example that I will show uses the AssociatedPress dataset (provided by the topicmodels package). This is a collection of 2246 news articles from an American news agency, mostly published around 1988.

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

Latent Dirichlet allocation (LDA)

We shall use the `LDA()` function, setting *k=2*, to create a two-topic LDA model. The function returns an LDA object containing the details of the model fit, such as how words are associated with topics and how topics are associated with documents.

```
> ap_lda = LDA(AssociatedPress, k=2, control = list(seed = 1234))
> class(ap_lda)
[1] "LDA_VEM" attr(,"package")
[1] "topicmodels"
> ap_lda
A LDA_VEM topic model with 2 topics.
```

Let us now explore and interpret the model using tidy tools.

Tidying model objects

To tidy the results of a Latent Dirichlet Allocation we use the `tidy()` function, originally from the broom package and provided by the package tidytext.

```
tidy(x, matrix = c("beta", "gamma"), ...)
```

`x` An LDA object from the topicmodels package

`matrix` Whether to tidy the beta (per-term-per-topic, default) or gamma (per-document-per-topic) matrix

Word-Topic Probabilities

Let us first extract the per-topic-per-word probabilities, called β from the model.

```
> ap_topics = tidy(ap_lda, matrix = "beta")
> ap_topics
# A tibble: 20,946 × 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
10   2 abbott     2.97e- 5
#  20,936 more rows
```

We obtain a tibble with format: one-topic-per-term-per-row.

For each combination the model computes the probability of that term being generated from that topic.

Word-Topic Probabilities

Exercise. Verify and convince yourself that β probabilities represent probability distributions for each topic.

Word-Topic Probabilities

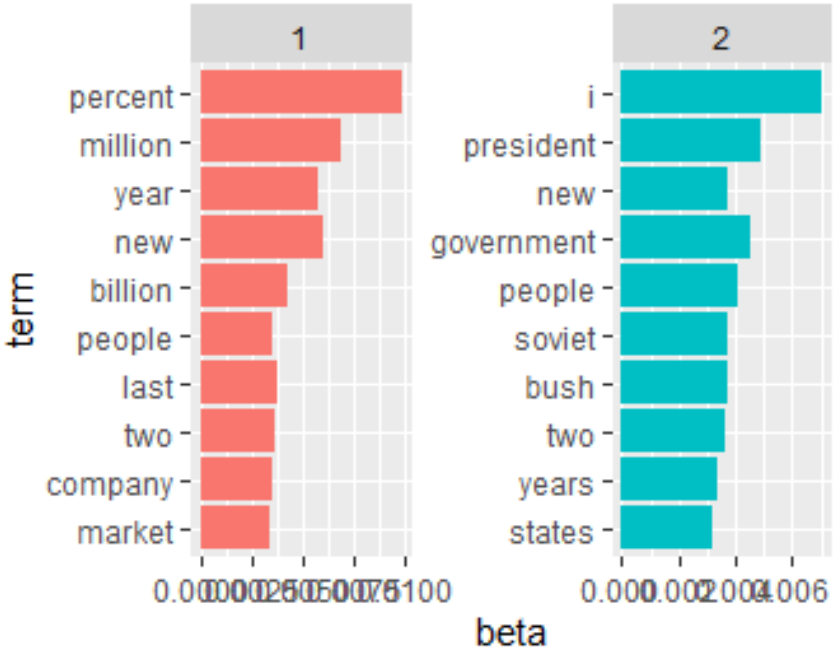
Let us find the 10 terms that are most common within each topic

```
> ap_top_terms = ap_topics |>
+ group_by(topic) |>
+ slice_max(beta, n= 10) |>
+ ungroup() |>
+ arrange(topic, -beta)
> ap_top_terms
# A tibble: 20 × 3
topic term           beta
<int> <chr>         <dbl>
1   1 percent     0.00981
2   1 million     0.00684
3   1 new         0.00594
4   1 year        0.00575
5   1 billion     0.00427
6   1 last        0.00368
7   1 two         0.00360
```

Word-Topic Probabilities

ggplot2 visualization

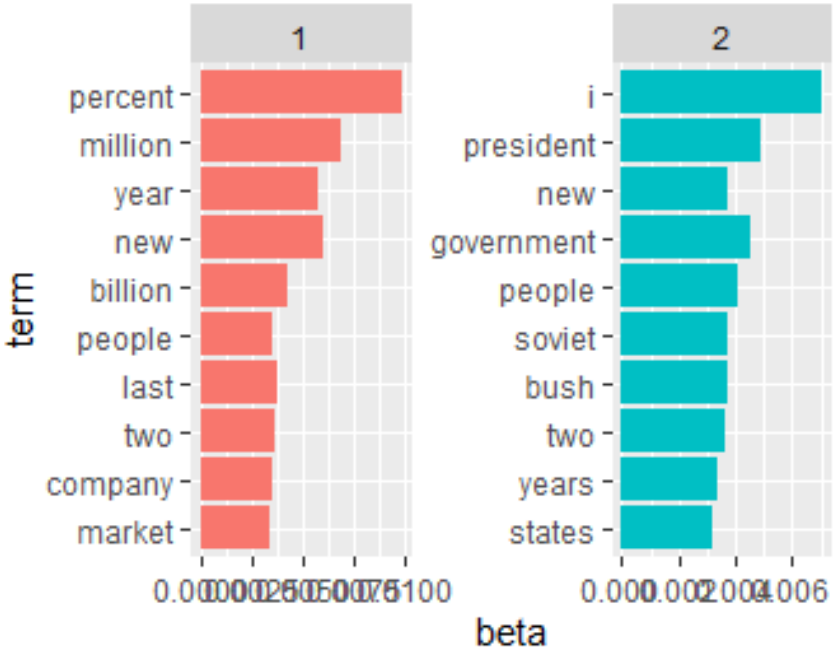
```
> ap_top_terms |>
+ mutate(term = reorder(term, beta)) |>
+ ggplot(aes(term, beta, fill = factor(topic))) +
+ geom_col(show.legend = F)+
+ facet_wrap(~topic, scales = "free")+
+ coord_flip()
```



Word-Topic Probabilities

ggplot2 visualization

```
> ap_top_terms |>
+ mutate(term = reorder(term, beta)) |>
+ ggplot(aes(term, beta, fill = factor(topic))) +
+ geom_col(show.legend = F)+
+ facet_wrap(~topic, scales = "free")+
+ coord_flip()
```



Topic 1: business or financial news
Topic 2: political news

Exercise for you

Exercise 1. An alternative way to identify the terms that best characterize each topic is to consider the terms that have the greatest difference in beta coefficients between topic1 and topic2. This can be estimated using the log ratio of the two, namely $\log_2(\beta_2 / \beta_1)$

With reference to the word-topic probabilities computed from the LDA analysis for the AcademicPress case study, write proper code to perform the following operations:

1) Replace the values 1 and 2 for the variable topic with the values topic1 and topic2
2) Create two variables, one containing beta values for topic1 and one beta values for topic2
3) Filter for words that have beta greater than 0.001 in at least one topic
4) Compute the log ratio and have a look at it
5) Produce a proper plot to visualize the most characteristic words for each topic.

Exercise for you

Exercise 2. Explore the AssociatedPress database, by performing the following operations.

1) Count the number of words per document. What is the maximum number of words in a document? And the minimum?
2) Which are the most frequent terms overall?
3) Drop stopwords.
4) Perform sentiment analysis on the newspaper articles using the lexicon bing.
5) Build a plot to visualize which words from the AP articles most often contributed to positive or negative sentiment.