Intro to R

Text-as-data with `tidytext`

Prof. Kevin Koehler
kevin.koehler@santannapisa.it

Structure

1. Basic assumptions of frequency-based approaches to quantitative text analysis

Structure

1. Basic assumptions of frequency-based approaches to quantitative text analysis
2. Sentiment analysis and topic models

Structure

1. Basic assumptions of frequency-based approaches to quantitative text analysis
2. Sentiment analysis and topic models
3. The `tidytext` approach to quantitative text analysis

Assumptions

Assumptions

Assumption 1 - Bag-of-words (BoW): Texts are a combination of tokens (words), grammar and word order do not matter.

Assumptions

Assumption 1 - Bag-of-words (BoW): Texts are a combination of tokens (words), grammar and word order do not matter.

> *"He almost failed every exam"* and
> *"He failed almost every exam"*
> are the same texts!

Assumptions

Assumption 1 - Bag-of-words (BoW): Texts are a combination of tokens (words), grammar and word order do not matter.

> *"He almost failed every exam"* and
> *"He failed almost every exam"*
> are the same texts!

Assumption 2 - Word frequency reflects importance: If a word is frequent in a document, but rare in the overall corpus, it is an important word. Term frequency-inverse document frequency (**TF-IDF**) is used to measure this.

Assumptions

Assumption 1 - Bag-of-words (BoW): Texts are a combination of tokens (words), grammar and word order do not matter.

> *"He almost failed every exam"* and
> *"He failed almost every exam"*
> are the same texts!

Assumption 2 - Word frequency reflects importance: If a word is frequent in a document, but rare in the overall corpus, it is an important word. Term frequency-inverse document frequency (**TF-IDF**) is used to measure this.

> The word "**election**" might stand out in texts on sports, but not so much in texts on politics.

Term Frequency-Inverse Document Frequency

The **TF-IDF** score for a term t in a document d from a corpus D is defined as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

The term frequency, $\text{TF}(t, d)$, measures how frequently a term occurs in a document:

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_k f_{k,d}}$$

where $f_{t,d}$ is the number of times term t appears in document d and $\sum_k f_{k,d}$ is the total number of terms in document d

The inverse document frequency, $\text{IDF}(t, D)$, reflects how important a term is across the corpus:

$$\text{IDF}(t, D) = \log\left(\frac{N}{1 + n_t}\right)$$

where N is the total number of documents in the corpus D and n_t is the number of documents containing term t. The "+1" prevents division by zero.

the final TF-IDF formula is therefore:

$$\boxed{\text{TF-IDF}(t, d, D) = \frac{f_{t,d}}{\sum_k f_{k,d}} \times \log\left(\frac{N}{1 + n_t}\right)}$$

Assumptions

Assumption 3 - Texts as vectors: Texts can be represented as vectors in high-dimensional space with each individual term a dimension.

Assumptions

Assumption 3 - Texts as vectors: Texts can be represented as vectors in high-dimensional space with each individual term a dimension.

> This allows for mathematical operations like calculating the similarity between documents based on **cosine similarity**.

Assumptions

Assumption 3 - Texts as vectors: Texts can be represented as vectors in high-dimensional space with each individual term a dimension.

> This allows for mathematical operations like calculating the similarity between documents based on **cosine similarity**.

Assumption 4 - Statistical patterns reflect meaning: Patterns in (relative) word frequency are assumed to reflect underlying themes or topics.

Assumptions

Assumption 3 - Texts as vectors: Texts can be represented as vectors in high-dimensional space with each individual term a dimension.

> This allows for mathematical operations like calculating the similarity between documents based on **cosine similarity**.

Assumption 4 - Statistical patterns reflect meaning: Patterns in (relative) word frequency are assumed to reflect underlying themes or topics.

> Topics in a set of documents can be identified by clustering words that frequently appear together (like in Latent Dirichlet Allocation, or **LDA**)

Cosine similarity

Cosine similarity measures the similarity between two vectors based on the angle between them, not their magnitude.

$$\text{Cosine similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Where:

- $\mathbf{A} \cdot \mathbf{B}$ is the **dot product** of the vectors, or $\sum_{i=1}^{n} a_i b_i$
- $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are the **Euclidean norms** (lengths) of the vectors, or $\sqrt{\sum a_i^2}$ and $\sqrt{\sum b_i^2}$, respectively.

> Value ranges from 0 (no common words) to 1 (the same words in the same proportion).
> Cosine similarity ignores vector length, focusing on **direction**.

Assumptions

Assumption 5 - Zipf's Law: Word frequencies in natural language follow Zipf's Law: a few words are very common, while many are rare.

Assumptions

Assumption 5 - Zipf's Law: Word frequencies in natural language follow Zipf's Law: a few words are very common, while many are rare.

> In English, words like *"the"* and *"is"* are extremely common, while many other words occur infrequently.

Assumptions

Assumption 5 - Zipf's Law: Word frequencies in natural language follow Zipf's Law: a few words are very common, while many are rare.

> In English, words like *"the"* and *"is"* are extremely common, while many other words occur infrequently.

Assumption 6 - Preprocessing enhances accuracy: Text preprocessing steps like tokenization, stop-word removal, and stemming are assumed to improve analysis by reducing noise.

Assumptions

Assumption 5 - Zipf's Law: Word frequencies in natural language follow Zipf's Law: a few words are very common, while many are rare.

> In English, words like *"the"* and *"is"* are extremely common, while many other words occur infrequently.

Assumption 6 - Preprocessing enhances accuracy: Text preprocessing steps like tokenization, stop-word removal, and stemming are assumed to improve analysis by reducing noise.

> Converting *"running"* and *"ran"* to the root word *"run"* to consolidate term frequencies (known as **stemming**).

Zipf's Law

George Kingsley Zipf (1902-1950)
Linguist and statistician

Zipf's Law states that in a given natural language, the frequency of any word is inversely proportional to its rank in the frequency table.

$$f(r) \propto \frac{1}{r^s}$$

Where $f(r)$ is the frequency of the word at rank r and s is a constant (with $s \approx 1$ for natural languages)

Zipf's Law

George Kingsley Zipf (1902-1950)
Linguist and statistician

Zipf's Law states that in a given natural language, the frequency of any word is inversely proportional to its rank in the frequency table.

$$f(r) \propto \frac{1}{r^s}$$

Where $f(r)$ is the frequency of the word at rank r and s is a constant (with $s \approx 1$ for natural languages)

> The second most frequent word appears about half as often as the most frequent word, the third one-third as often, and so on.

Sentiment analysis

What is Sentiment Analysis?

Sentiment analysis

Sentiment analysis is the process of **automatically identifying the emotional tone** behind a body of text.

> The goal is to classify text as **positive**, **negative**, or **neutral** — or score it on a scale.

How Does It Work?

1. **Text preprocessing**: Clean and tokenize the text
2. **Lexicon lookup or model prediction**
3. **Score aggregation and interpretation**

Lexicon-based approach

Lexicon-based sentiment analysis

Lexicon-based sentiment analysis uses a dictionary of words with associated sentiment scores.

Example words:

- love $\rightarrow +3$
- hate $\rightarrow -3$
- excellent $\rightarrow +2$
- boring $\rightarrow -2$

Sentence:

> "The movie was excellent, but a bit boring."

Score:
$+2$ (excellent) -2 (boring) $=$ **0** (neutral)

Common sentiment lexicons

- **AFINN**: Numeric scores from -5 to $+5$
- **bing**: Positive / Negative classification
- **NRC**: Emotion categories + polarity

Used via the `tidytext` package in R (or `textdata` in Python).

Example in R

```r
library(tidytext)
library(dplyr)

text_df <- tibble(line = 1, text = "I love this movie but it’s boring")
sentiment <- get_sentiments("bing")

text_df %>%
  unnest_tokens(word, text) %>%
  inner_join(sentiment, by = "word") %>%
  count(sentiment)
```

```
# A tibble: 2 x 2
  sentiment     n
  <chr>     <int>
1 negative      1
2 positive      1
```

Example in R

```r
library(tidytext)
library(dplyr)

text_df <- tibble(line = 1, text = "I love this movie but it’s boring")
sentiment <- get_sentiments("bing")

text_df %>%
  unnest_tokens(word, text) %>%
  inner_join(sentiment, by = "word") %>%
  count(sentiment)
```

```
# A tibble: 2 x 2
  sentiment     n
  <chr>     <int>
1 negative      1
2 positive      1
```

! Conclusion

We would conclude that the text is neutral.

Topic models

What is LDA?

LDA

Latent Dirichlet Allocation (LDA) is a **topic modeling** algorithm. It uncovers **hidden topics** in a collection of documents.

What is LDA?

LDA

Latent Dirichlet Allocation (LDA) is a **topic modeling** algorithm. It uncovers **hidden topics** in a collection of documents.

Assumptions:

- Each **document is a mixture of topics**.
- Each **topic is a distribution over words**.

What is LDA?

LDA

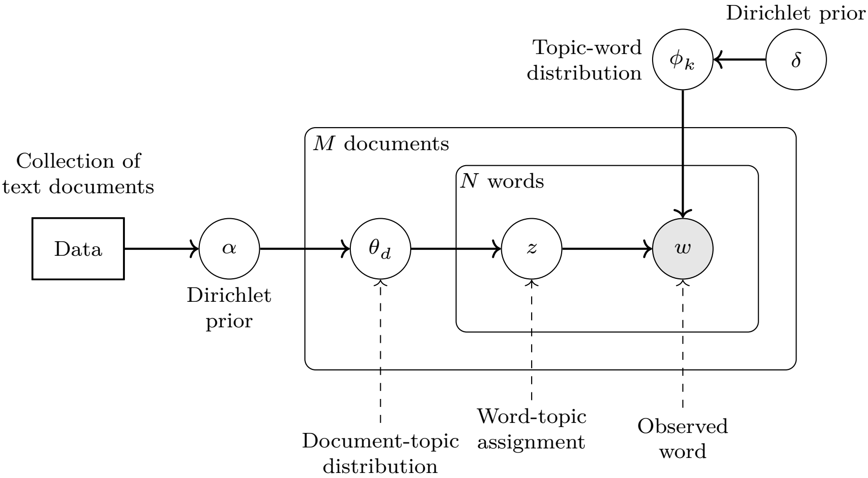
Latent Dirichlet Allocation (LDA) is a **topic modeling** algorithm. It uncovers **hidden topics** in a collection of documents.

Assumptions:

- Each **document is a mixture of topics**.
- Each **topic is a distribution over words**.

> Instead of labeling documents manually, LDA **discovers patterns** of word use. Words that **co-occur across documents** are grouped into topics. Each topic is a **bag of words** with certain probabilities.

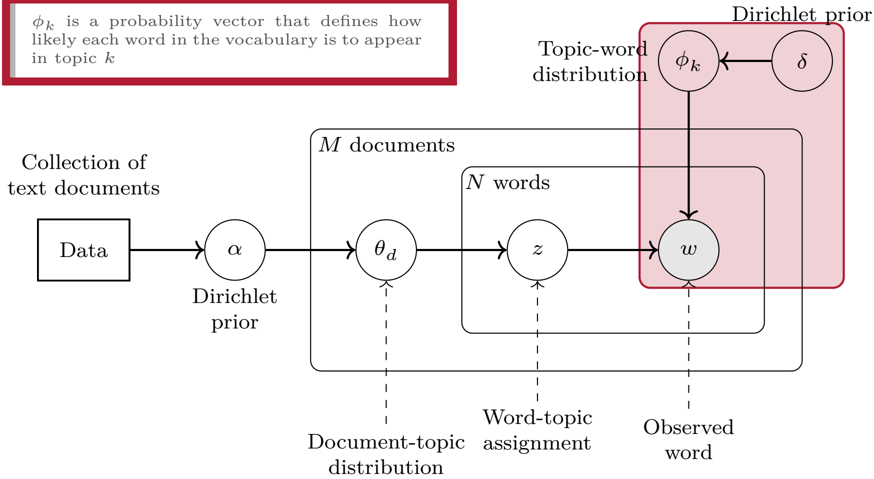
The data generation process in LDA



Step 1: Topic-word distribution

For each topic $k = 1, \dots, K$: $\phi_k \sim \mathrm{Dir}(\delta)$, where ϕ_k is a probability distribution over all words and δ is a parameter controlling sparsity.

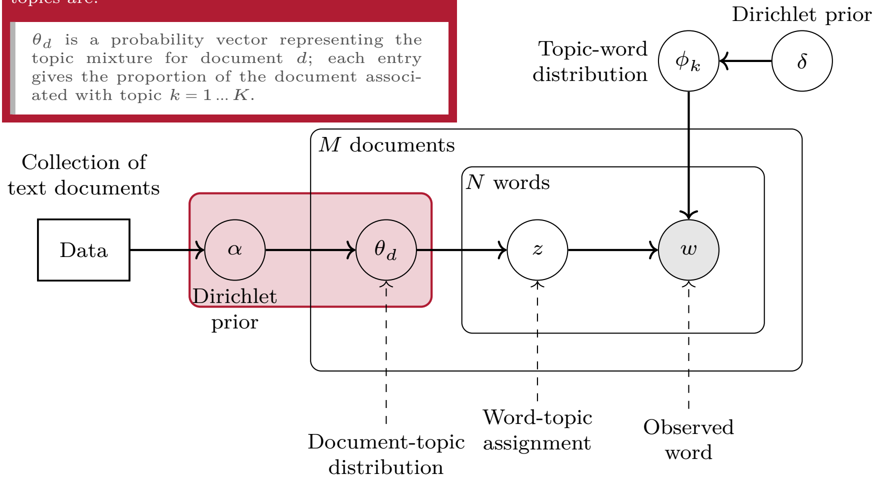
> ϕ_k is a probability vector that defines how likely each word in the vocabulary is to appear in topic k



Step 2: Document-topic distribution

For each document $d = 1, \ldots, M$: $\theta_d \sim \mathrm{Dir}(\alpha)$, where θ_d is a topic mixture specific to document d and α controls how concentrated or diverse the topics are.

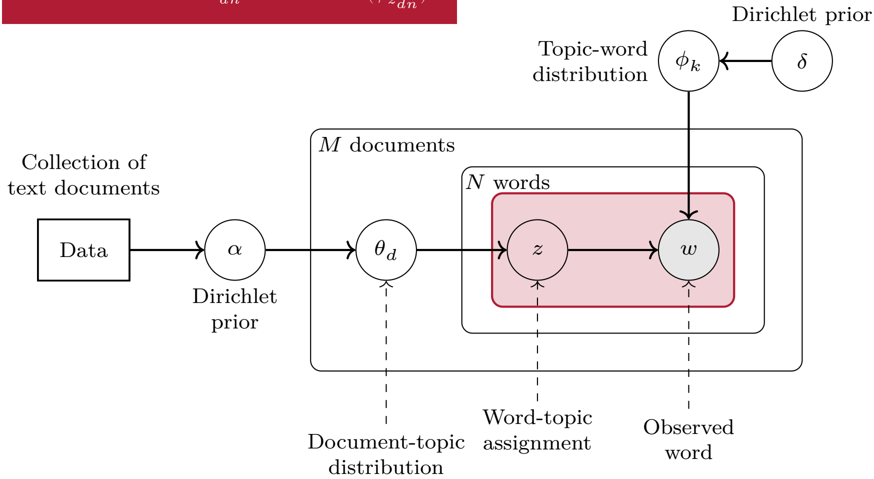
> θ_d is a probability vector representing the topic mixture for document d; each entry gives the proportion of the document associated with topic $k = 1 \ldots K$.



Step 3: Generate each word

For each word w_{dn} in document d:

1. choose a topic: $z_{dn} \sim \text{Multinomial}(\theta_d)$, and
2. choose a word: $w_{dn} \sim \text{Multinomial}(\phi_{z_{dn}})$.



Summary: LDA

Step 1: Topic-word distribution

For each topic $k = 1, \ldots, K$:

$$\phi_k \sim \text{Dirichlet}(\delta)$$

- ϕ_k is a probability distribution over all words.
- δ is a parameter controlling sparsity.

> This gives us the most likely words per topic.

Summary: LDA

Step 1: Topic-word distribution

For each topic $k = 1, \dots, K$:

$$\phi_k \sim \text{Dirichlet}(\delta)$$

- ϕ_k is a probability distribution over all words.
- δ is a parameter controlling sparsity.

> This gives us the most likely words per topic.

Step 2: Doc-topic distribution

For each document $d = 1, \dots, D$:

$$\theta_d \sim \text{Dirichlet}(\alpha)$$

- θ_d is a topic mixture specific to document d.
- α controls how concentrated or diverse the topics are.

> This gives us the most likely topic per document

Summary: LDA

Step 1: Topic-word distribution

For each topic $k = 1, \dots, K$:

$$\phi_k \sim \text{Dirichlet}(\delta)$$

- ϕ_k is a probability distribution over all words.
- δ is a parameter controlling sparsity.

> This gives us the most likely words per topic.

Step 2: Doc-topic distribution

For each document $d = 1, \dots, D$:

$$\theta_d \sim \text{Dirichlet}(\alpha)$$

- θ_d is a topic mixture specific to document d.
- α controls how concentrated or diverse the topics are.

> This gives us the most likely topic per document

Step 3: Generate each word

For each word w_{dn} in document d:

1. Choose a topic:

$$z_{dn} \sim \text{Multinomial}(\theta_d)$$

2. Choose a word:

$$w_{dn} \sim \text{Multinomial}(\phi_{z_{dn}})$$

> This gives us the most likely word, given the document, its topic, and the distribution of words in the topic.

Output of LDA

Given documents (only words are observed), LDA infers:

- ϕ_k: Word distributions per topic
- θ_d: Topic distributions per document
- z_{dn}: Topic assignment per word
- In some specifications, α and δ are estimated as well

Output of LDA

Given documents (only words are observed), LDA infers:

- ϕ_k: Word distributions per topic
- θ_d: Topic distributions per document
- z_{dn}: Topic assignment per word
- In some specifications, α and δ are estimated as well

You get:

- For each **document**: a vector of **topic proportions** (θ_d)
- For each **topic**: a list of **most likely words** (ϕ_k)
- For each **word**: its **most likely topic** (z_{dn})

Take-home

- LDA is a **generative probabilistic model**.
- It uses **Dirichlet priors** to control topic and word sparsity.
- Useful for **unsupervised discovery** of themes in large text corpora.

Text analysis with `tidytext`

What is `tidytext`?

- `tidytext` brings **text mining** into the **tidyverse**.
- It treats text as tidy data: one word per row, one document per group.
- Works seamlessly with `dplyr`, `ggplot2`, `tidyr`, etc.

Why Use `tidytext`?

`tidytext` offers easy integration with `tidyverse` workflows

It Simplifies:

- Tokenization
- Stopword removal
- Sentiment analysis
- Word frequency + TF-IDF
- Topic modeling

Tokenization with `unnest_tokens()`

```r
text_df <- tibble(doc_id = c(1,2),
                  text = c("Text mining is fun!", "I love using tidytext."))
tokens <- text_df %>%
  unnest_tokens(word, text)
tokens
```

```
# A tibble: 8 x 2
  doc_id word
   <dbl> <chr>
1      1 text
2      1 mining
3      1 is
4      1 fun
5      2 i
6      2 love
7      2 using
8      2 tidytext
```

Tokenization

`unnest_tokens()` splits text into one token (word) per row; you can also tokenize by sentence, n-grams, etc.

Stopword Removal

```r
data("stop_words")
tokens_clean <- tokens %>%
  anti_join(stop_words, by = "word")
tokens_clean
```

```
# A tibble: 5 x 2
  doc_id word
   <dbl> <chr>
1      1 text
2      1 mining
3      1 fun
4      2 love
5      2 tidytext
```

Stopword removal

Removes common "stopwords" like "the", "is", "and" using standard lexicons (`onix`, `snowball`, `smart`). You can also add your own custom stopwords.

Word counts and TF-IDF

```
word_counts <- tokens %>%
  count(word, sort = TRUE)
word_counts
```

```
# A tibble: 8 x 2
  word         n
  <chr>    <int>
1 fun          1
2 i            1
3 is           1
4 love         1
5 mining       1
6 text         1
7 tidytext     1
8 using        1
```

Word counts and TF-IDF

```
tf_idf <- tokens %>%
  count(doc_id, word, sort = TRUE) %>%
  bind_tf_idf(term = word, document = doc_id, n = n)
tf_idf
```

```
# A tibble: 8 x 6
  doc_id word         n    tf   idf tf_idf
   <dbl> <chr>    <int> <dbl> <dbl>  <dbl>
1      1 fun          1  0.25 0.693  0.173
2      1 is           1  0.25 0.693  0.173
3      1 mining       1  0.25 0.693  0.173
4      1 text         1  0.25 0.693  0.173
5      2 i            1  0.25 0.693  0.173
6      2 love         1  0.25 0.693  0.173
7      2 tidytext     1  0.25 0.693  0.173
8      2 using        1  0.25 0.693  0.173
```

TF-IDF

Combining `count()` and `bind_tf_idf()` gives you the TF-IDF for each word.

gutenbergr

We will use example data provided by the `gutenbergr` package. This package gives you access to the full text of 72,569 books and other documents by 23,980 authors in 68 languages. We will download **H.G. Wells'** *The War of the Worlds* and **Jane Austin's** *Pride and Prejudice* and use parts of these texts.

In particular, we divide both books into chapters which we treat as separate documents. This gives us 88 documents. We then run a topic model to see whether the model can distinguish begtween Austin and Wells.

Topic models with `topicmodels`

```
library(topicmodels)
library(SnowballC)

tokens <- chapters %>%
  unnest_tokens(word, text) %>%
  filter(!grepl("[0-9]", word)) %>%
  filter(!str_detect(word, "^[[:punct:]]+$")) %>%
  anti_join(stop_words, by = "word") %>%
  mutate(word = wordStem(word)) %>%
  count(doc, word, sort = TRUE) %>%
  ungroup

dtm <- tokens %>%
  cast_dtm(doc, word, n)

lda_model <- LDA(dtm, k = 2,
                 method = "Gibbs",
                 control = list(
                   seed = 1234),
                 alpha = 0.01,
                 delta = 0.01)
```

We have 38,362 words (tokens) in our texts, containing 6,652 unique words.

We set $\alpha = 0.01$ and $\delta = 0.01$ to make topics as consistent as possible.

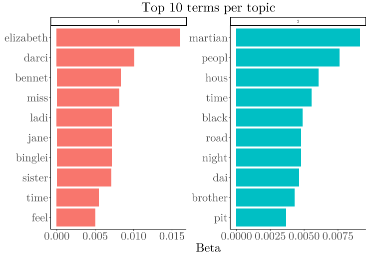
The `seed` option ensures reproducibility. `method = "Gibbs"` uses random sampling to assign words to topics iteratively. Setting a seed makes sure that the model produces consistent results over multiple iterations.

Topic models with `topicmodels`

```r
library(latex2exp)
topics <- tidy(lda_model, matrix = "beta")

# View top terms per topic
top_terms <- topics %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

top_terms %>%
  mutate(term = reorder_within(term,
                               beta,
                               topic)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_x_reordered() +
  coord_flip() +
  labs(title = "Top 10 terms per topic",
       x = NULL,
       y = TeX("$\\beta$"))
```



$$\beta = P(\text{word}|\text{topic})$$

Hence, if we randomly draw a word from Topic 1, there is about a 1.5% chance that it will be “Elizabeth”; if we do the same from Topic 2, there is about a 0.8% chance that it will be “Martians”.

Exercises (on GitHub)