Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 7

10 March 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO | Dipartimento di Scienze Economiche

Outline

Key basic concepts

Types of text mining

Tidy text format

Tidy text format vs other data structures

Preprocessing

Frequent terms

Packages: **tidytext**

Functions: `unnest_tokens()`, `anti_join()`, `count()`

Some key basic concepts

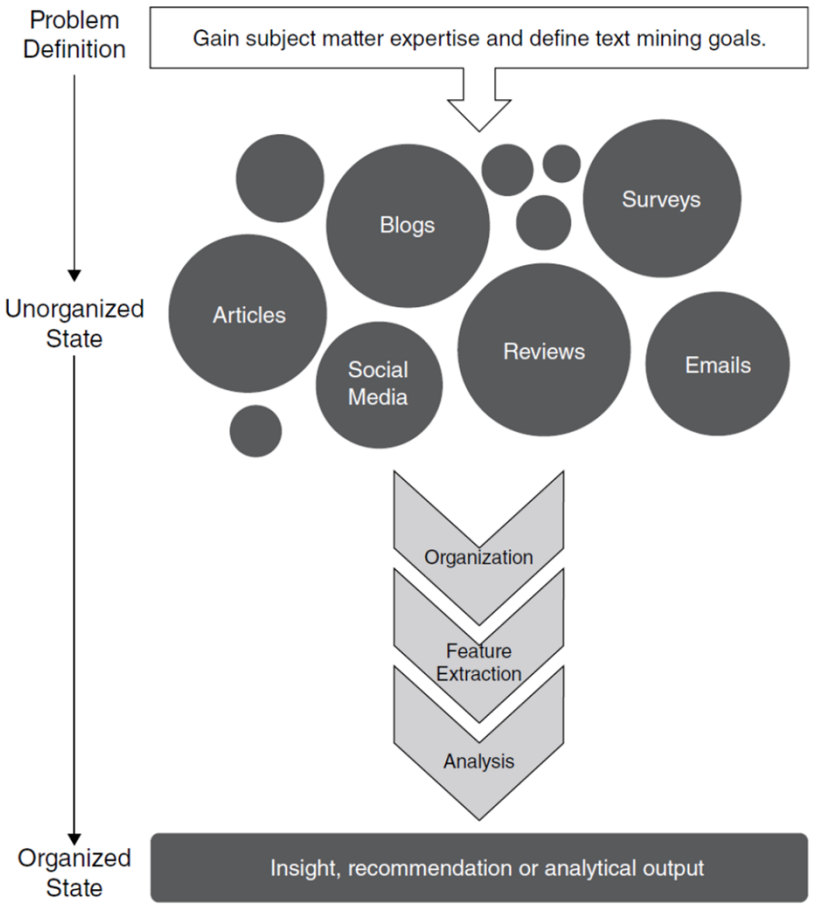
(text) corpus	a large and structured set of texts for analysis
Document	each of the units of a corpus (e.g. a tweet or a FB post)
Token	piece/part of a document (word, phrase, sentence,…)
Tokenization	process of splitting text into tokens

e.g. A corpus is a set of documents.
This is the 2nd document in the corpus.

is a corpus with 2 documents, where each document is a sentence. The first document has 7 tokens. The second has 8 tokens.

stop words	common words that often do not provide any additional insight, such as articles. They are designated for exclusion from any analysis of a text

Text mining project workflow



From Kwartler T. (2017), Text Mining in practice with R, Wiley.

The figure shows a high level **workflow** of a text mining project, with structured predefined steps that are applied to the unorganized text to reach the final output or conclusion.

Text mining project workflow

1) **Define the problem and specific goals**. As the practitioner, you need to acquire subject matter expertise sufficient to define the problem and the outcome in an appropriate manner.
2) **Identify the text that needs to be collected**. Care must be taken to explicitly select text that is appropriate to the problem definition. Typical sources are web scraping and the use of APIs.
3) **Organize the text**. Once the appropriate text is identified, it is collected and organized into a corpus or collection of documents.
4) **Preprocessing**. Clear and prepare the text for subsequent analyses. Examples include making all text lowercase, or removing punctuation.
5) **Analyze**. Apply the analytical technique to the prepared text. The goal of applying an analytical methodology is to gain an insight or a recommendation or to confirm existing knowledge about the problem. The analysis can be relatively simple, such as searching for a keyword, or it may be an extremely complex algorithm.
6) **Reach an insight or recommendation**. The end result of the analysis is to apply the output to the problem definition or expected goal.

Types of Text Mining

Overall there are two types of text mining, one called “bag of words” and the other “syntactic parsing,” each with its benefits and shortcomings.

Bag of words treats every word (or group of words) as a unique feature of the document. Word order and grammatical word type are not captured in a bag of word analysis. This approach disregards grammar and word order and uses word frequencies as features.

One benefit of the bag of words approach is that it is generally **not computationally expensive or too technical.**

Syntactic parsing differs from the bag of words approach in its complexity and approach. It is based on word syntax.

At its root, syntax represents a set of rules that define the components of a sentence that then combine to form the sentence itself (similar to building blocks). Syntactic parsing uses **part of speech tagging** techniques to identify the words themselves in a grammartical or useful context. It creates the building blocks that make up a sentence. Then the blocks are anlysed to draw out the insight. The building block methodologies can become relatively complicated.

Text Mining in R

tidytext: uses tidy data principles, which can make many text mining tasks easier, more effective, and consistent with tools already in wide use. One of its benefits is that it works very well in tandem with other tidy tools in R such as dplyr or tidyr.

tm: provides a comprehensive text mining framework for R, with some powerful functions which will aid in text-processing steps and techniques for count-based analysis methods, text clustering, text classification and string kernels.

openNLP: provides an R interface to OpenNLP, a machine learning based toolkit for the processing of natural language text written in Java.

udpipe: toolkit providing language-agnostic tokenization, tagging, lemmatization and dependency parsing of raw text.

quentada: fast, flexiblem and comprehensive framework for quantitative text analysis in R. It provides functionalities to perform several tasks from NLP – corpus management, preprocessing, exploring and analysing keyworks, computing feature similarities and distances – to more advanced statistical analyses, such as wordscores, document classification (Naive Bayes) and topic modeling.

Tidy text format

We start using the **tidytext** package

Tidy data principles can make text mining tasks easier, more effective, and consistent with tools already in wide use.

We define the **tidy text format** as a table with one token per row. For tidy text mining, the token that is stored in each row is most often a single *word*, but can also be an *n-gram*, *sentence*, or *paragraph*.

Treating text as data frames of individual words allows to manipulate, summarise, and visualise the characteristics of text easily.

Tidy Text Format vs Other Data Structures

String	text can, of course, be stored as strings (i.e. character vectors) within R, and often text data is first read into memory in this form
Corpus	These types of objects typically contain raw strings annotated with additional metadata and details.
Document-term matrix	This is a sparse matrix describing a collection (i.e., a corpus) 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

Tidy Text Format vs Other Data Structures

Document term matrix (DTM). Consider the following three tweets:

- *@hadleywickham:* "How do I hate thee stringsAsFactors=TRUE? Let me count the ways #rstats"
- *@recodavid:* "R the 6th most popular programming language in 2015 IEEE rankings #rstats"
- *@dtchimp:* "I wrote an #rstats script to download, prep, and merge @ACLEDINFO's historical and realtime data."

Abbreviated document term matrix, showing simple word counts contained in the three-tweet corpus

Tweet	@ acledinfo's	#rstats	2015	6th	And	Count	Data	Download	…
1	0	1	0	0	0	1	0	0	…
2	0	1	1	1	0	0	0	0	…
3	1	1	0	0	2	0	1	1	…

Tidy text format

At the same time, the **tidytext** package does not expect a user to keep text data in a tidy form at all times during an analysis. The package also includes functions to tidy objects from other popular text mining R packages.

This allows, for example, a workflow where importing, filtering, and processing is done using **dplyr** and other tidy tools, after which the data is converted into a DTM for other applications. The models eventually can then be reconverted into a tidy form for interpretation and visualization.

Dataset

We will work with the tibble 'text.tbl' (created in Lecture 6) containing Delta tweets from October 1 to October 15, 2015. Variables:

weekday	day of the week
month	month
data	day of the month
year	year
text	tweet text
agents	agent initial letters

```
> library(tidyverse)
> library(tidytext)
```

Preprocessing

The cleaning steps outlined here represent common and foundational steps.
You can custom your preprocessing steps, depending on the analysis. For instance, in *X* you may want to preprocess specific tokens such as 'RT' or '#' by either removing retweets or explicitly identifying hashtag tokens as providing more context in the analysis.

We will need to keep track of the tweets. Since this tibble does not have unique Ids, we create them in the new tibble called tweets

```
> tweets = text.tbl |>
+ mutate(
+ ID = seq_along(text)
+ )
> tweets
# A tibble: 1,377 × 7
weekday month date  year  text          agents ID
<chr>   <chr> <dbl> <dbl> <chr>         <chr> <int>
1 Thu     Oct     1  2015 @mjdout I know t… AA    1
2 Thu     Oct     1  2015 @rmarkerm Terrib… AA    2
```

Preprocessing

`unnest_tokens()` Split a column into tokens, transforming the table into one-token-per-row.

```
unnest_tokens(
  tbl,
  output,
  input,
  token = "words",
  to_lower = TRUE,
  ...
)
```

`tbl`	A data frame
`output`	Output column to be created as string
`input`	Input column that gets split as string
`token`	Unit for tokenizing, or a custom tokenizing function. Some built-in options are "**words**" (default), "characters","ngrams", "sentences", "lines", "paragraphs", "regex", "tweets" (tokenization by word that preserves usernames, hashtags, and URLS).
`to_lower`	Whether to convert tokens to lowercase. If tokens include URLS (such as with token = "tweets"), such converted URLs may no longer be correct.

Preprocessing

```
> tidy.tweets = tweets |>
+ unnest_tokens(word, text)
> class(tidy.tweets)
[1] "tbl_df" "tbl" "data.frame"
> dim(tidy.tweets)
[1] 21758 7
> tidy.tweets
# A tibble: 21,758 × 7
weekday month date  year  agents ID    word
<chr>   <chr> <dbl> <dbl> <chr>  <int> <chr>
1 Thu    Oct     1  2015  AA     1     mjdout
2 Thu    Oct     1  2015  AA     1     i
3 Thu    Oct     1  2015  AA     1     know
4 Thu    Oct     1  2015  AA     1     that
5 Thu    Oct     1  2015  AA     1     can
6 Thu    Oct     1  2015  AA     1     be
7 Thu    Oct     1  2015  AA     1     frustrating
….
```

Check the dimensions of the data frame: there are 21 758 rows and 7 columns. This means that in total there are 21 758 words (not distinct) after the tokenization

Preprocessing

After using `unnest_tokens()` each row is split so that there is one token (word) in each row of the new tibble. The default tokenization is for single words.

Further:

- Other columns, such as the line number each word came from, are retained
- Punctuation is removed
- By default, `unnest_tokens()` converts the tokens to lowercase

Having the text in this format (one-word-per-row), it is possible to manipulate, process, and visualize it using the standard set of tidytools.

Preprocessing

Typically in text analysis, we want to remove **stopwords**. In the tidytext package, stopwords are kept in the dataset `stop_words`, which contains English stop words from three lexicons, as a data frame.
This dataframe has 1149 rows and 2 variables (`word` = An English word, `lexicon` = The source of the stop word).

```
> class(stop_words)
[1] "tbl_df" "tbl" "data.frame"
> dim(stop_words)
[1] 1149 2
> str(stop_words)
tibble [1,149 × 2] (S3: tbl_df/tbl/data.frame)
$ word : chr [1:1149] "a" "a's" "able" "about" ...
$ lexicon: chr [1:1149] "SMART" "SMART" "SMART" "SMART" ...
```

Preprocessing

```
> stop_words
# A tibble: 1,149 × 2
word lexicon
<chr>        <chr>
1 a            SMART
2 a's          SMART
3 able         SMART
4 about        SMART
5 above        SMART
6 according    SMART
7 accordingly SMART
8 across       SMART
9 actually     SMART
10 after       SMART
# … 1,139 more rows #
> View(stop_words)
```

Preprocessing

We shall remove stopwords using the `dplyr::anti_join()` function. This function filters rows from `x` based on the presence of absence of matches in `y`

```
anti_join(x, y)
```

`x, y` A pair of data frames, data frame extensions (e.g. a tibble)

`anti_join()` returns all rows from `x` without a match in `y`.

Preprocessing

```
> tidy.tweets.2 = tidy.tweets |>
+ anti_join(stop_words)
Joining with `by = join_by(word)`
> tidy.tweets.2
# A tibble: 10,231 × 7
weekday month date  year agents ID    word
<chr> <chr> <dbl>  <dbl> <chr> <int> <chr>
1 Thu   Oct     1   2015   AA     1   mjdout
2 Thu   Oct     1   2015   AA     1   frustra…
3 Thu   Oct     1   2015   AA     1   hope
4 Thu   Oct     1   2015   AA     1   parked
5 Thu   Oct     1   2015   AA     1   deplaned
6 Thu   Oct     1   2015   AA     1   shortly
7 Thu   Oct     1   2015   AA     1   patience
8 Thu   Oct     1   2015   AA     1   aa
9 Thu   Oct     1   2015   AA     2   rmarkerm
10Thu   Oct     1   2015   AA     2   terribly
# … 10,221 more rows
```

Exercise

1) Remove stopwords from the tibble tidy.tweets in an equivalent way to the above using one usual verb in **dplyr**.
2) Check equivalence of the results. Hint: you may use the function `base::identical`. Check its usage in the help.

Preprocessing

Beyond stopwords contained in `stop_words`, there might be other words we want to drop from the dataset. We can do that simply using the `filter()` verb in **dplyr**.

```
> View(tidy.tweets.2)
```

	weekday	month	date	year	agents	ID	word
32	Thu	Oct	1	2015	/3	4	3
33	Thu	Oct	1	2015	/3	4	3
34	Thu	Oct	1	2015	/3	5	nealaa
35	Thu	Oct	1	2015	/3	5	advisory
36	Thu	Oct	1	2015	/3	5	issued
37	Thu	Oct	1	2015	/3	5	bahamas
38	Thu	Oct	1	2015	/3	5	change
39	Thu	Oct	1	2015	/3	5	check

Preprocessing

```
> tidy.tweets.2 = tidy.tweets.2 |>
+ filter( + !str_detect(word, "\\d"))
> tidy.tweets.2
# A tibble: 8,695 × 7
weekday month date year agents ID   word
<chr> <chr> <dbl> <dbl> <chr> <int> <chr>
1 Thu  Oct    1     2015  AA      1  mjdout
2 Thu  Oct    1     2015  AA      1  frustra…
3 Thu  Oct    1     2015  AA      1  hope
4 Thu  Oct    1     2015  AA      1  parked
5 Thu  Oct    1     2015  AA      1  deplaned
6 Thu  Oct    1     2015  AA      1  shortly
7 Thu  Oct    1     2015  AA      1  patience
8 Thu  Oct    1     2015  AA      1  aa
9 Thu  Oct    1     2015  AA      2  rmarkerm
10Thu  Oct    1     2015  AA      2  terribly
# … 8,685 more rows
```

Frequent terms

Performing a **frequency analysis** is often a good place to start when presented with a text mining problem. To this purpose, we shall use the `count()` function, which lets you quickly count the unique values of one or more variables

```
count(x, ..., sort = FALSE, name = NULL)
```

`x` A data frame, data frame extension (e.g. a tibble)

`sort` If TRUE, will show the largest groups at the top

`name` The name of the new column in the output. If omitted, it will default to `n`. If there's already a column called n, it will error, and require you to specify the name.

A new data frame object is created by putting the original words and the corresponding frequencies next to each other,

Frequent terms

```
> freq.df = tidy.tweets.2 |>
+ count(word, sort = TRUE)
> freq.df
# A tibble: 2,213 × 2
word              n
<chr>         <int>
1 dm              181
2 follow          155
3 pls             154
4 hear            151
5 team            137
6 confirmation    127
7 flight          109
8 pl              104
9 ng               96
10 assistance      91
# … 2,203 more rows
```

We see a lot of tweets have please, flight, confirmation, assistance.
As an airline, this may not be surprising, but it can still be insightful to understand common issues and help to draw inferences.

Reviewing the most frequent terms can provide some insight into typical customer services issues this company encountered

Exercise for you

Exercise 1

1) Produce the frequency tables of words in tidy.tweets.2 in an equivalent way to the above using the verb `summarise()` in **dplyr**.
2) Check equivalence of the results.

Exercise 2

With reference to the Delta customer service dataset,

1) Build the frequency distribution of words of length 2 in the tibble tidy.tweets.2 and inspect it. Do you think we should retain any of these words?
2) Remove words of length 2 from the tibble tidy.tweets.2 (except those that you think should be retained) and name the new tibble tidy.tweets.3.
3) Produce the frequency table of words in tidy.tweets.3.
4) Remove the word 't.co' from tidy.tweets.3 and produce a new frequency table of words.