NLP 1 - From text to (big) matrices

SICSS Paris 2023

Julien Boelaert

CERAPS, Université de Lille

23/6/2023

Outline

1. Introduction: from text corpus to matrix (DTM)
2. Preprocessing: building an efficient matrix
3. Documents as term-vectors: distance, weighting
4. Work with DTMs: reduction, prediction

1/ Introduction

In order to make text corpora amenable to statistical analysis, we have to transform the texts into numbers.

DTM: Document-Term Matrix

	fellow-citizens	of	the	senate	and	house	repre
1790_G_Washington1	1	69	97	2	41	3	
1790_G_Washington2	1	89	122	2	45	3	
1791_G_Washington3	1	158	240	3	73	3	
1792_G_Washington4	1	139	195	2	56	3	
1793_G_Washington5	1	132	180	2	49	3	
1794_G_Washington6	1	187	273	2	86	4	
1795_G_Washington7	1	130	174	4	73	4	
1796_G_Washington8	1	193	262	2	94	3	
1797_J_Adams9	0	128	185	2	87	3	
1798_J_Adams10	0	151	214	2	91	3	
1799_J_Adams11	0	106	148	3	53	3	
1800_J_Adams12	0	86	123	2	50	3	
1801_T_Jefferson13	0	178	220	1	100	1	
1802_T_Jefferson14	0	126	175	1	80	1	
1803_T_Jefferson15	0	139	184	3	92	1	
1804_T_Jefferson16	0	137	184	1	70	1	
1805_T_Jefferson17	0	191	235	1	95	1	
1806_T_Jefferson18	0	152	225	1	83	1	
1807_T_Jefferson19	0	126	178	1	81	1	
1808_T_Jefferson20	0	159	249	1	94	1	
1809_J_Madison21	1	105	172	1	51	1	
1810_J_Madison22	1	172	252	1	77	1	
1811_J_Madison23	1	135	204	1	73	1	

- rows = documents
- columns = terms (set = **vocabulary**)
- elements = counts

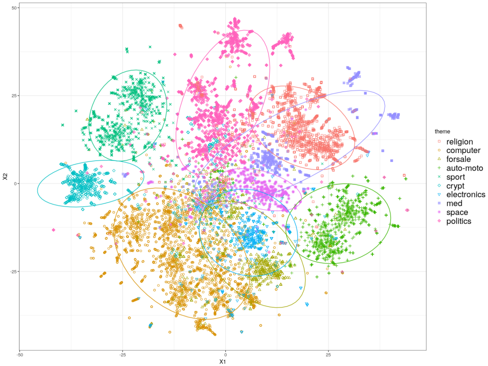
1/ Introduction

The **DTM** is the basis for all classical text mining methods

- efficiently computed
- stored as a *sparse* matrix: don't store the 90%+ of 0s

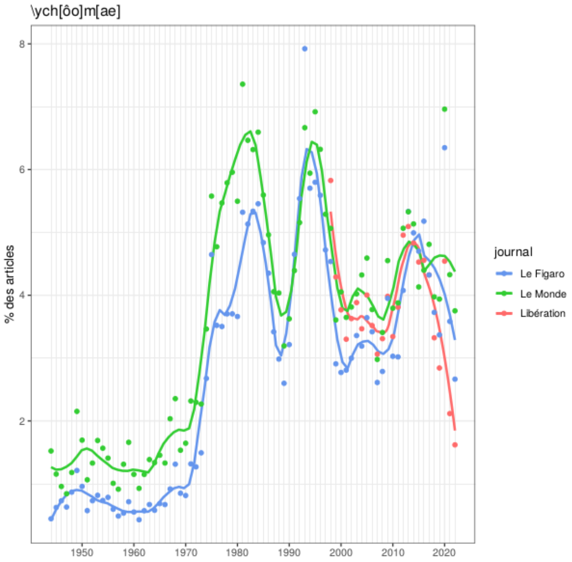
Many possible uses:

- descriptive stats
- dim. reduction
- prediction
- document retrieval
- word similarity
- variations: transpose (TDM), cross-product: term co-occurrence matrix (TCM)



t-SNE on a DTM of 20news corpus

1/ Introduction

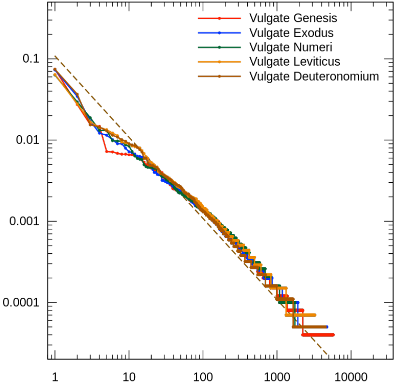


Mentions of unemployment in French newspapers over time

1/ Introduction

Why so sparse?

- **Zipf's law**: words' frequency $\simeq$ inverse of its rank
- most words will rarely appear, hence many 0s in the DTM
- extreme case: *hapax* (1 occurrence, 40-60% in large corpora)



Term counts (proportion) vs. rank, on Vulgate Pentateuch

1/ Introduction

Limitations of the DTM paradigm:

- "bag of words" (BOW) representation:
  - "The cat is on the mat" = "The mat is on the cat"
  - (possible alleviation: n-grams)
- "one-hot" coding limitations:
  - similar words are different columns
    ("cat" $\neq$ "cats" just as much as "cat" $\neq$ "computer")
  - polysemic words are a single column ("march", "mouse", ...)
- very big matrix -> curse of dimensionality
  - several preprocessing steps will help reduce the vocabulary size

1/ Introduction

There are several steps in the creation of a useful DTM:

1. Corpus creation
2. Tokenization (word detection, special symbols, n-grams...)
3. Standardization (lowercase, lemmatization, named entities...)
4. Filtering (on counts, POS, ...)
5. DTM computation
6. Weighting (binary, tf-idf)

We will cover these steps in some detail today.

2/ Preprocessing

Preprocessing = choosing what the rows and columns of the DTM will represent.

1/ Determining the **corpus**, *ie* what is a **document**?

- Unit: A book? chapter? paragraph? sentence? tweet-answer couple?
- Each document will end up as a **row** in the DTM.
- Correct choice depends on your research goals, and feasibility
- In practice, in R, a corpus will be *character vector*, or a column in a data frame (other columns being metadata: author, date, ...).

2/ Preprocessing

Several available R tools for splitting into paragraphs or sentences

- strsplit(x, split = "\n")
- tokenizers::tokenize_paragraphs(x), tokenizers::tokenize_sentences(x)
- tidytext::unnest_sentences(x), tidytext::unnest_paragraphs(x) are wrappers around the tokenizers functions
- quanteda::tokens(x, what = "sentence")
- spacyr::spacy_tokenize(x, what = "sentence")

2/ Preprocessing

2/ **Tokenization** is the process of splitting documents into tokens (usually words), that will end up as **columns** in the DTM.

Usually easy in Western languages: split on space or punctuation (but "U.S.A", URLs, hashtags...)

Some usual choices to be made:

- remove punctuation, symbols?
- remove numbers? replace them with "XXX"?
- split hyphens? (*eg* "pre-processing", 1 or 2 tokens?)

In practice, the tokenized object will either be a list of character vectors, or a data.frame with one row per token and special columns indicating which document it comes from and at what place.

2/ Preprocessing

Many available R tools for tokenization

- strsplit(x, split = "\\\\W") (fast but risky)
- tokenizers::tokenize_words(), tokenizers::tokenize_ptb()
- tidytext::unnest_tokens() usually calls the tokenizers functions
- quanteda::tokens() : several options for speed and accuracy (careful: doesn't split apostrophies)
- spacyr::spacy_tokenize() : powerful, language-tuned models
- spacyr::spacy_parse() : even more powerful, tokenizes and detects POS, NER etc.
- udpipe::udpipe() : very powerful, language-tuned neural-network models, also detect POS, NER etc.

Once tokenization is done, we can already build a (rough) dtm: quanteda::dfm(toks)

2/ Preprocessing

We might want to include **token n-grams** as tokens too
→ richer representation of the documents (but larger vocabulary).

Example: "The cat is on the mat."

- 2-grams: "The_cat", "cat_is", "is_on", "on_the", "the_mat", "mat_."
- 3-grams: "The_cat_is", "cat_is_on", "is_on_the", "on_the_mat", "the_mat_."

This is done after the first tokenization, *eg* with

- ```
  toks <- quanteda::tokens(corpus, "word")
  ngrams <- quanteda::tokens_ngrams(toks, n = 1:3)
  ```
  ```

2/ Preprocessing

3/ **Standardization** is the process of grouping tokens together to form an efficient vocabulary $\rightarrow$ smaller vocabulary, but loss of information

3a/ A common and simple standardization is transforming all documents to **lower case**

- "Cat" $=$ "cat"
- But: "White House" $\stackrel{?}{=}$ "white house"

NB: with *quanteda* we will transform to lowercase at the final DTM construction step, so as not to interfere with other preprocessing steps

2/ Preprocessing

3b/ Another important standardization is **lemmatization**: grouping tokens according to their common root

- *eg* "sings", "sang", "sung" -> "sing"

Stemming is a rough version of this, relying on cutting off suffixes such as "-ation", "-s", etc.

- quanteda::tokens_wordstem() does this for several languages, but will not harmonize *eg* "ate" and "eat".

More refined lemmatizers will first infer POS info, then exploit this with machine learning to produce lemmas:

- **TreeTagger** (koRpus in R): good language-wise results using 90s tech (HMM + decision trees)
- modern packages such as **spacyr** and **udpipe** use language-wise pre-trained neural networks for even better results

2/ Preprocessing

Example of the spacy parser in action:

doc_id	sentence_id	token_id	token	lemma	pos	entity
text1	1	1	Joe	Joe	PROPN	PERSON_B
text1	1	2	Newman	Newman	PROPN	PERSON_I
text1	1	3	ate	eat	VERB	
text1	1	4	apples	apple	NOUN	
text1	1	5	when	when	ADV	
text1	1	6	he	he	PRON	
text1	1	7	went	go	VERB	
text1	1	8	to	to	ADP	
text1	1	9	San	San	PROPN	GPE_B
text1	1	10	Francisco	Francisco	PROPN	GPE_I
text1	1	11	with	with	ADP	
text1	1	12	the	the	DET	ORG_B
text1	1	13	Count	Count	PROPN	ORG_I
text1	1	14	Basie	Basie	PROPN	ORG_I
text1	1	15	Orchestra	Orchestra	PROPN	ORG_I
text1	1	16	in	in	ADP	
text1	1	17	1964	1964	NUM	DATE_B
text1	1	18	.	.	PUNCT	

2/ Preprocessing

3c/ Yet another interesting standardization concerns **named entities** (names of people, places, organizations)

- Harmonization: "USA" = "U.S.A."
- Nounphrase consolidation: merge multi-token entities "San Francisco" → "San_Francisco"
- Both: "Jacques Chirac" = "Jack Chirac" → "Jacques_Chirac"

For this, we will typically:

1. use spacyr::spacy_parse() to detect named entities, and spacyr::entity_extract() to extract them
2. use spacyr::nounphrase_consolidate() to consolidate them
3. after close inspection, manually build a dictionary of other things to harmonize, (*eg* the Chirac case) and apply quanteda::dfm_lookup() after the DFM is built.

2/ Preprocessing

4/ Finally, it is common practice to filter out some tokens:

- Based on dictionaries, pre-built or hand-crafted
  eg stopwords ("I", "me", "am", "and", ...)

- Based on counts, *eg* exclude tokens that appear less than X times in total, or in less than X documents, or in more than X% of the documents
  ex: quanteda::dfm_trim(the_dfm, min_docfreq = 10)

- Based on POS
  eg keep only NEs, verbs, nouns, adjectives, adverbs
  Here again, use advanced parsers: TreeTag, spacy, udpipe

2/ Preprocessing

Recap: full pre-processing DTM pipeline - **quanteda only**

```
library(quanteda)
texts <- ... ## choose document unit (ie number of rows)

## Tokenize
toks <- tokenize(texts)

## Consolidate on dictionary
toks <- tokens_lookup(toks, dictionary = ...)
## Filter on dictionary
toks <- tokens_remove(toks, stopwords("en"))
## Include n-grams
toks <- tokens_ngrams(toks, n = 1:2)

## DTM (lowercase?)
dtm <- dfm(toks, tolower = TRUE)
## Filter on counts
dtm <- dfm_trim(dtm, min_docfreq = 5)
```

2/ Preprocessing

Recap: quick pre-processing DTM pipeline - **tidytext only**

```
library(dplyr); library(stringr); library(SnowballC)
library(tidytext)

## First put the texts in a data.frame (or tibble)
corpus_df <- data.frame(content = text, doc = docnames, year = ...)

t_toks <- corpus_df %>%
    filter(between(year, 1900, 2000)) %>%                   ## Filter on year
    unnest_tokens(output = token, input = content) %>%      ## Tokenize (case?)
    anti_join(get_stopwords(), by = c("token" = "word")) %>% ## Rm stopwords
    filter(!str_detect(token, "[:digit:]")) %>%             ## Remove numbers
    mutate(token = wordStem(token, language = "en")) %>%    ## Stemming
    group_by(token) %>% filter(n() > 3)                     ## Filter on counts

t_dtm <- t_toks %>%
    count(doc, token) %>% cast_dfm(doc, token, n)           ## Compute DTM

t_tfidf <- t_toks %>%
    count(doc, token) %>% bind_tf_idf(token, doc, n) %>%    ## Compute tf-idf
    cast_dfm(doc, token, tf_idf)                            ## Format as DTM
```

2/ Preprocessing

Recap: full pre-processing DTM pipeline - **spacy + quanteda**

```
library(quanteda); library(spacyr);
texts <- ... ## choose document unit (ie number of rows)

## Parse: tokenize, detect POS, NER, lemma, ...
spacy_initialize("selected_spacy_model")
parsy <- spacy_parse(texts)
spacy_finalize()
parsy <- nounphrase_consolidate(parsy) ## Consolidate NEs

select_pos <- c("ADJ", "ADV", "ENTITY", "NOUN", "PROPN", "VERB")
parsy <- parsy[parsy$pos %in% select_pos, ] ## Filter on POS

## Convert to quanteda tokens
toks <- tapply(parsy$lemma, parsy$doc_id, identity)[names(texts)]
toks <- as.tokens(toks)
## Include n-grams
toks <- tokens_ngrams(toks, n = 1:2)

## DTM (lowercase?), filter on counts, ... see quanteda pipeline
```

3/ Documents as term-vectors

Processing the DTM:

- Documents now live in term-space
- Distance: how different/similar are two documents?
- Weights: making documents more comparable

NB: we will treat DTMs, but the same applies to TDMs and TCMs

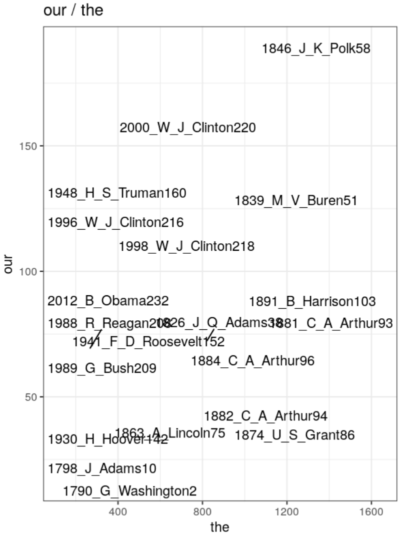
3/ Documents as term-vectors

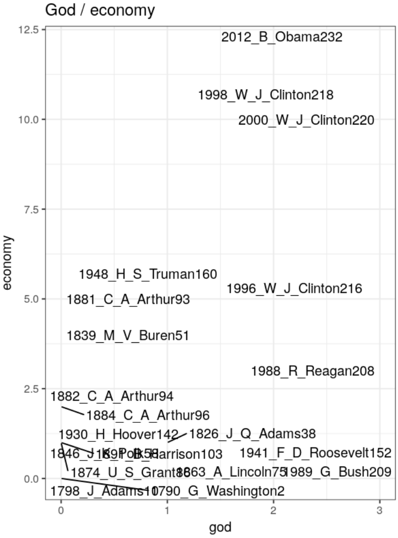
DTM: each **document** is a **vector in term-space**

"and lo, the text had become data"

	the	of	,	.	and	to	in	a	that	for	be	our	is
1790_G_Washington1	97	69	40	24	41	56	20	21	15	7	20	10	10
1790_G_Washington2	122	89	57	39	45	49	27	21	17	16	18	17	11
1791_G_Washington3	240	158	98	59	73	88	41	42	33	22	34	5	18
1792_G_Washington4	195	139	112	60	56	88	48	32	24	30	29	11	17
1793_G_Washington5	180	132	83	54	49	74	26	34	12	23	43	16	13
1794_G_Washington6	273	187	175	79	86	138	36	48	39	14	36	22	12
1795_G_Washington7	174	130	102	51	73	64	27	33	27	22	22	43	20
1796_G_Washington8	262	193	132	75	94	112	67	57	31	35	41	28	23
1797_J_Adams9	185	128	110	59	87	76	39	21	17	28	21	20	11
1798_J_Adams10	214	151	81	58	91	79	29	41	33	18	25	26	19
1799_J_Adams11	148	106	75	36	53	64	22	25	21	10	23	18	14
1800_J_Adams12	123	86	71	37	50	63	19	17	12	13	20	23	11
1801_T_Jefferson13	220	178	167	88	100	135	50	49	53	31	61	47	23
1802_T_Jefferson14	175	126	120	61	80	66	46	36	26	29	30	37	18
1803_T_Jefferson15	184	139	112	47	92	79	43	22	29	32	28	35	7
1804_T_Jefferson16	184	137	115	49	70	65	55	24	35	20	23	26	8
1805_T_Jefferson17	235	191	131	77	95	94	63	42	28	31	34	50	15

3/ Documents as term-vectors





SOTU, Counts of words for 20 random documents

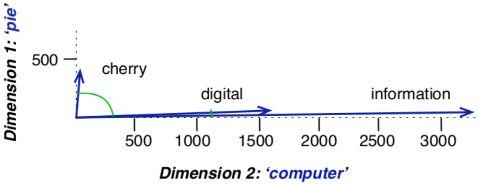
3/ Documents as term-vectors

To compare the documents, we need a **distance/similarity** metric:

- euclidean distance? too sensitive to magnitude
- cosine similarity!

0 (nothing in common) to 1 (perfectly aligned)
(no negatives, because counts are all positive)

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$



(Jurafsky and Martin, 2021, 6.2)

NB: cosine distance = 1 - similarity

3/ Documents as term-vectors

Cosine similarity:

- neutralizes the overall magnitude
  (*eg* document twice as long)
- but on raw counts, still sensitive to **relative magnitudes** of possibly uninformative words
  (*eg* "the", "and", "in" / "economy", "war", "Providence")
- need for **weighting**: we will use cosine on a weighted DTM

3/ Documents as term-vectors

Weighting: neutralize the uninformative (too frequent) terms

Transformation of the DTM:

- (filter out in DTM pre-processing)
- binary: presence/absence
- **tf-idf**: scale and neutralize
- (PPMI on TCM: how much more two words co-occur than could be expected by chance)

	rights	more	indeed
1790_G_Washington1	1	0	0
1790_G_Washington2	1	1	0
1791_G_Washington3	1	1	0
1792_G_Washington4	0	1	1
1793_G_Washington5	0	1	0
1794_G_Washington6	1	1	1
1795_G_Washington7	1	1	0
1796_G_Washington8	1	1	0
1797_J_Adams9	1	1	1
1798_J_Adams10	1	1	1
1799_J_Adams11	1	0	0
1800_J_Adams12	0	1	0

SOTU: Binary DTM

3/ Documents as term-vectors

Binary: quanteda::dfm_weight(dtm, "boolean")

tf-idf: new weighted DTM from the original counts

- def: term frequency × inverse document frequency
- one common form (Jurafsky & Martin, 2021):

$$w_{dt} = log_{10}(1 + termcount_{dt}) \times log_{10}\left(\frac{N}{doccount_t}\right)$$

- tf: weigh up (small) differences in document-term counts
- idf: weigh down terms that appear in many documents
- in R: quanteda::dfm_tfidf(dtm, "logcount", "inverse")

4/ Work with DTMs

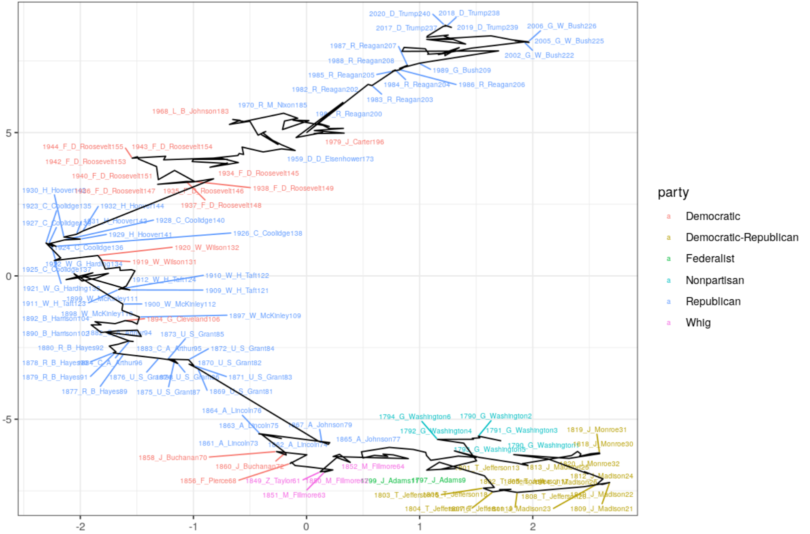
Now we have a big matrix and a reasonably scaled distance measure, we can plug it into standard algorithms:

- Dimensionality reduction
  - feature extraction: LSA (for term-doc or term-term)
  - visualization (on aggregated groups): CA
  - nonlinear visualization: t-SNE, UMAP
- Predictive algorithms (regression, classification)
  - penalized linear / SVM
  - random forest, neural networks...

4/ Work with DTMs

Reduction for visualization, with umap:

- uwot::umap(wdfm, metric = "cosine")



4/ Work with DTMs

Prediction of party (Dem, Rep, Other) from tf-idf on SOTU

- quanteda.textmodels::svm(wdfm, party)

Results on random holdout set: 94% accuracy

		Predicted				
		Dem	Other	Rep	*Acc.*	*F1*
	Dem	16	2	1	0.94	0.91
Truth	Other	0	6	0	0.96	0.86
	Rep	0	0	23	0.98	0.98

Coefficients: highest positive coefs. (x100) for Republican

	colored	acreage	rebellion	postal	d	ballistic	immigration
Dem	-3.83	-5.04	-3.10	-3.45	-5.19	-5.48	-4.61
Rep	6.14	5.94	5.93	5.91	5.78	5.75	5.44
Other	-2.37	-2.97	-3.28	-2.18	-1.29	-0.94	-1.92

Further reading

- Dan Jurafsky and James H. Martin, 2023, *Speech and Language Processing* (3rd ed. draft), chapter I.6
  `https://web.stanford.edu/~jurafsky/slp3`

- Powerful parsers
  `https://spacy.io`
  `https://lindat.mff.cuni.cz/services/udpipe`

- Tutorials
  `https://tutorials.quanteda.io`
  `https://www.tidytextmining.com/tidytext.html`

Questions and comments: `julien.boelaert@univ-lille.fr`