Week 2: Quantifying Texts

LSE MY459: Quantitative Text Analysis
https://lse-my459.github.io/

Ryan Hübert

Slides last updated: 27 January 2025

Don't forget

For this week's seminar:

- ➜ Bring a laptop!
- ➜ Install R (from https://www.r-project.org/)
- ➜ Install RStudio Desktop (from https://www.rstudio.com/products/rstudio-desktop/)

You will do coding activities in seminar!

Going forward, also:

- ➜ Create a GitHub account
- ➜ Install GitHub Desktop (from https://desktop.github.com/)

Quantifying texts

We want to use quantitative (read: statistical) methods, so our goal is to conceptualise *texts* as *tabular data* (read: a matrix)

```
When I presented the
supplementary budget to
this House last April, I
said we could work our
way through this period
of severe economic
distress. Today, I can
report that
notwithstanding the
difficulties of the past
eight months, we are now
on the road to economic
recovery.
```

```
In this next phase of the
Government's plan we must
stabilise the deficit in
a fair way, safeguard
those worst hit by the
recession, and stimulate
crucial sectors of our
economy to sustain and
create jobs. The worst is
over.
```

```
This Government has the
moral authority and the
well-grounded optimism
rather than the cynicism
of the Opposition. It has
the imagination to create
the new jobs in energy,
agriculture, transport
and construction that
this green budget will
```

docs	words made	because	had	into	get	some	through	next	where	many	irish
t06_kenny_fg	12	11	5	4	8	4	3	4	5	7	10
t05_cowen_ff	9	4	8	5	5	5	14	13	4	9	8
t14_ocaolain_sf	3	3	3	4	7	3	7	2	3	5	6
t01_lenihan_ff	12	1	5	4	2	11	9	16	14	6	9
t11_gormley_green	0	0	0	3	0	2	0	3	1	1	2
t04_morgan_sf	11	8	7	15	8	19	6	5	3	6	6
t12_ryan_green	2	2	3	7	0	3	0	1	6	0	0
t10_quinn_lab	1	4	4	2	8	4	1	0	1	2	0
t07_odonnell_fg	5	4	2	1	5	0	1	1	0	3	0
t09_higgins_lab	2	2	5	4	0	1	0	0	2	0	0
t03_burton_lab	4	8	12	10	5	5	4	5	8	15	8
t13_cuffe_green	1	2	0	0	11	0	16	3	0	3	1
t08_gilmore_lab	4	8	7	4	3	6	4	5	1	2	11
t02_bruton_fg	1	10	6	4	4	3	0	6	16	5	3

Descriptive statistics on words

Scaling documents

Classifying documents

Extraction of topics

Vocabulary analysis

Sentiment analysis

Outline for today

1. Document selection (rows of DFM)
2. Feature selection (columns of DFM)
3. Issues in feature selection
4. Describing texts

Maybe (probably not):

5. Two approaches to analysis with DFMs

Document selection (rows of DFM)

Strategies for selecting units of textual analysis

A **document** is the fundamental unit of analysis for quantitative text analysis, for exampe:

- n-word sequences
- Sentences
- Pages
- Paragraphs
- Natural units (a speech, a poem, a manifesto)
- Aggregation of units (e.g. all speeches by party and year)

A collection of documents is called a **corpus**

How you define documents depends on the research design

- Recall first assumption of QTA: Texts represent an observable implication of some underlying thing of interest — select documents with this in mind

Side note on terminology

In this class:

- a **document** is the chosen unit of analysis; could be a full "document" in colloquial sense, or not
- a **text** (used as a countable noun) is what we often refer to as a "document" in the colloquial sense, e.g. a novel, a speech, a legal opinion, a tweet, etc.
- **text** (used as an uncountable noun) is a general word used to label a type of data, similar to "string"

We will try our best to be consistent, but context will usually be informative

Selecting texts

Stepping back: how do you know which texts you should collect and include for your QTA task?

This is a complicated issue! But it starts with a deep substantive knowledge of your research topic

As with most research — this is part art, part science

Difference between a **sample** and a **population**

- ➜ The distinction is a little philosophical (beyond this course)
- ➜ But keep in mind potential biases that can arise from your data collection decisions

Key: make sure that what is being analysed is a valid representation of the phenomenon as a whole – again, a question of **research design** — read chs. 3 and 4 of GRS

Where to obtain textual data?

Existing datasets, e.g.

- UCD's EuroParl project
- Hansard Archive of parliamentary debates in UK
- Media archives (newspapers, TV transcripts) in databases
- Academic articles (JSTOR Data for Research)
- Open-ended responses to survey questions

Collect your own data:

- From social media and/or blogs
- Scraping other websites

Digitise your own text data using OCR

Warning: *always* scrutinise terms of service before collecting data

Alas, we're not going to cover data collection in this course

Example: US President Trump's tweets

To demonstrate concepts, we will use a corpus of US President Trump's tweets

- ➜ The corpus covers 2017 and half of 2018
- ➜ Available on the course website

Data is stored in a `.json` format based on the Twitter API

- ➜ Not formatted as a standard JSON file; can't use `jsonlite`
- ➜ Need the `streamR` package to load file

Example: US President Trump's tweets

Source: https://x.com/realDonaldTrump/status/815422340540547073

Example: US President Trump's tweets


```
{ "created_at": ["Sun Jan 01 05:00:10 +0000 2017"],
  "id": [8.15422340540547e+17],
  "id_str": ["815422340540547073"],
  "full_text": ["TO ALL AMERICANS-\n#HappyNewYear &amp; many blessings to
                you all! Looking forward to a wonderful &amp; prosperous
                2017 as we work together to #MAGA\U0001F1FA\U0001F1F8
                https://t.co/UaBFaoDYHe"],
  "truncated": [false],
  "display_text_range": [[0],[148]],
  "entities": {"hashtags": [{"text":["HappyNewYear"],"indices":[[18],[31]]},
                           {"text":["MAGA"], "indices":[[141],[146]]}],
             "symbols": [],
             "user_mentions":[],
             "urls":[],
             "media": [{"id":[8.15422333510816e+17],
                        "id_str":["815422333510815746"],
                        "indices":[[149],[172]],
                        "media_url":["http://pbs.twimg.com/media/C1D2SsLVEAIn
                        "media_url_https":["https://pbs.twimg.com/media/C1D2S
                        "url":["https://t.co/UaBFaoDYHe"],
                        ...
```

Feature selection (columns of DFM)

Defining document features

Recall second assumption of QTA: Texts can be represented by extracting their “features”

Documents contain **features**, which can be:

- characters
- words
- word “stems” or “lemmas” (more later)
- word segments, especially for languages using compound words, such as German, e.g. *Saunauntensitzer*
- “word” sequences, especially when inter-word delimiters (usually white space) are not commonly used, e.g., in Chinese
- linguistic features, such as parts of speech
- coded or annotated text segments
- word embeddings (more later)

The most common approach: bag of words

Most common approach to quantifying text: **bag of words** model

- ➜ Single *words* are the relevant *features* of each document
- ➜ Documents are quantified by counting occurences of words
- ➜ Word order does not matter
- ➜ Discards grammar and syntax

Why bag of words?

Most obvious reason: it's very simple

But also, context is often uninformative and conditional on *presence* of words

- ➜ Individual word usage tends to be associated with a particular degree of affect, position, etc. without regard to context
- ➜ So presence of words by itself captures info about context

Plus, *single* words tend to be the most informative since co-occurrences of multiple words ("n-grams") are relatively rare

But for our purposes: in social science applications bag of words works well most of the time (i.e., it's been validated *a lot*)

Why bag of words?

There are times where word order is important, e.g.

- **Text reuse**: plagiarism detection software used for social science applications, such as Corley (2007) and Grimmer (2010)
- **Parts of speech tagging**: tagging words in documents with grammatical information, with applications such as Bamman and Smith (2014) and Handler et al (2016)
- **Named entity recognition (NER)**: finding and tagging words in documents that are people, organisations, or places, with applications such as Copus, Hübert and Pellaton (2024)

As usual: whether word order matters depends on the QTA task!

Implementing bag of words

Goal: get from a set of texts to a quantitative dataset

There is a basic four step process for implementing bag of words to quantify texts:

1. Choose unit of analysis
2. Tokenise
3. Reduce complexity
4. Create **document feature matrix**

People often refer to this as **preprocessing**

The "reducing complexity" part is where most of the discretion is

➜ You will need to *justify* why the preprocessing steps you took make sense for your task!

Tokenising

You start by **tokenising** each document:

- Split into an array of words, eached called a **token**
- For English (and many other languages), use white space; trickier for logographic languages (e.g. Chinese)
- Tokens are *mostly* “words”—but not always

A list of all the distinct tokens used in an entire corpus is a **vocabulary**

- Each element of the vocabulary is a **type**

Tokenising Trump's tweet

Original (plain) text of Trump tweet:

```
TO ALL AMERICANS-\n#HappyNewYear &amp; many blessings
to you all! Looking forward to a wonderful &amp;
prosperous 2017 as we work together to
#MAGA\U0001F1FA\U0001F1F8 https://t.co/UaBFaoDYHe
```

Tokenising Trump's tweet

Use white space to tokenise:

```
c("TO", "ALL", "AMERICANS-", "#HappyNewYear", "&amp;",
  "many", "blessings", "to", "you", "all!", "Looking",
  "forward", "to", "a", "wonderful", "&amp;",
  "prosperous", "2017", "as", "we", "work", "together",
  "to", "#MAGA", "https://t.co/UaBFaoDYHe")
```

Notice some issues here:

- some useless punctuation: "AMERICANS-"
- some "non-words" included: links, ampersands
- the words "to" and "TO" are considered different words (side note: remember that in coding/computer context, CAPITALISATION IS IMPORTANT)

This will create a vocabulary that is too large and redundant

Reduce complexity: cleaning up formatting

To deal with these problems, we can: remove “non-words” and punctuation, and make text lowercase

```
c("to", "all", "americans", "#happynewyear", "many",
  "blessings", "to", "you", "all", "looking", "forward",
  "to", "a", "wonderful", "prosperous", "as", "we",
  "work", "together", "to", "#maga")
```

Note:

- Had to manually get rid of symbols written in HTML syntax (e.g. `&`) using regex pattern `"[&][#]?[A-z]+;"`
- Decided to keep Twitter hashtags intact (why?)

Reduce complexity: removing stop words

Stop words are words that occur very frequently in a language but do not provide much information

Some very common English stop words are:

`a, able, about, across, after, all, almost, also, am, among, an, and, any, are, as, at, be, because, been, but, by, can, cannot, could, dear, did, do, does, either, else, ever, every, for, from, get, got, had, has, have, he, her, hers, him, his, how, however, I, if, in, into, is, it, its, just, least, let, like, likely, may, me, might, most, must, my, neither, no, nor, not, of, off, often, on, only, or, other, our, own, rather, said, say, says, she, should, since, so, some, than, that, the, their, them, then, there, these, they, this, tis, to, too, twas, us, wants, was, we, were, what, when, where, which, while, who, whom, why, will, with, would, yet, you, your`

In QTA, it is very common to remove stop words

- ➜ But this depends on your task! See Pennebaker (2011)
- ➜ There are different lists out there, some longer, some shorter

Removing stop words from Trump's tweet

We can remove all the stop words (defined by the list above) from the Trump tweet:

```
c("americans", "#happynewyear", "many", "blessings",
  "looking", "forward", "wonderful", "prosperous",
  "work", "together", "#maga")
```

Reduce complexity: stemming and lemmatisation

Many words have multiple “forms” with different spellings, e.g.

- Tenses: “I see” and “I saw”
- Pluralisation: “family” and “families”
- Contractions: “families” and “family’s”

So far, we have treated all these as different words

But, you may wish to create **equivalence classes** of words considered to convey same meaning

- Basic idea: for every token, idenify the “root” word, and replace the token with root word
- This reduces vocabulary, and can be quite useful for comparing across documents

You may or may not want to do this depending on your task!

Reduce complexity: stemming and lemmatisation

Two common approaches:

1. **Lemmatisation**: refers to the algorithmic process of converting words to their **lemma**

➜ A word's **lemma** is its "canonical form"

2. **Stemming**: removing the ends of words using a set of rules

Basic difference: stemmers operate on single words without knowledge of the context, lemmatisers are more powerful

➜ There is a computational tradeoff

Example: `production`, `producer`, `produce`, `produces`, `produced` all replaced with `produc`

Tools for each available in `quanteda` package

Reduce complexity: stemming and lemmatisation

Lots of different ways to stem and lemmatise

Porter stemmer is most common, but gets many stems wrong:

- `policy` and `police` considered (wrongly) equivalent
- `general` becomes `gener`, `iteration` becomes `iter`

There are other corpus-based, statistical, and mixed approaches designed to overcome these limitations

Plus, sometimes stemming isn't appropriate: Schofield and Mimno (2016) find that "stemmers produce no meaningful improvement in likelihood and coherence (of topic models) and in fact can degrade topic stability"

Take away: you have to read and validate!

Reduce complexity: stemming and lemmatisation

We can use `quanteda`'s default stemming function `tokens_wordstem()` on the Trump tweet, yielding:

```
c("american", "#happynewyear", "mani", "bless", "look",
  "forward", "wonder", "prosper", "work", "togeth",
  "#maga")
```

Note:

- `quanteda` uses the Snowball stemmer by default
- The stemmer did many transformations, e.g.:
  - `americans` → `american`
  - `many` → `mani`
  - `blessing` → `bless`
  - `together` → `togeth`

The document-feature matrix (DFM)

So far, we've just "pre-processed" one example document

You repeat this process for every document

- ➜ This yields a vocabulary of types for the corpus
- ➜ These are the *features* to be analysed (again: remember we're assuming bag of words!)

Each document uses some of the features in the vocabulary

- ➜ You can count how many times

A **document-feature matrix** (math: $\mathbf{W}$) is a matrix of N documents (rows) by J features (columns) where:

- ➜ each W_{ij} counts the number of times the jth feature appears in the ith document

The document-feature matrix (DFM)

All of Trump's tweets from January 2017:

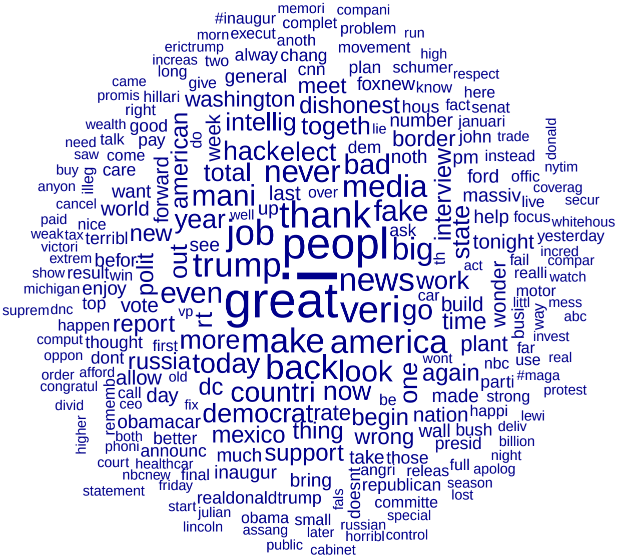
```
Document-feature matrix of: 212 documents, 1,039 features (98.94%
  sparse) and 0 docvars.
                     features
docs                  american #happynewyear mani bless look
  2017-01-01 05:00:10        1             1    1     1    1
  2017-01-01 05:39:13        0             1    0     0    0
  2017-01-01 05:43:23        0             0    0     1    1
  2017-01-01 05:44:17        0             0    0     0    0
  2017-01-01 06:49:33        0             0    0     0    0
  2017-01-01 06:49:49        0             0    0     0    0
[ reached max_ndoc ... 206 more documents, reached max_nfeat ...
  1,034 more features ]
```

This DFM has $J = 1,039$ features (words) and $N = 212$ documents (tweets)

➜ Each cell is a count, e.g. $W_{11} = 1$ and $W_{21} = 0$

Wordclouds

Wordclouds are basic visualisations of the data in a DFM



Issues in feature selection

Issues in feature selection

Recall: how you implement QTA depends on your task!

This applies at all steps of analysis, including constructing a DFM

There are some well known issues to consider when thinking about defining and selecting features

You already saw some in the context of specific documents: formatting, stop words, equivalence classes (stemming/lemmatisation)

We now consider three main types of issues that can come up when we consider entire corpuses

Issues in feature selection

1. In some languages (e.g. English), some phrases with multiple words have singular meanings as if they were single words
   - ➜ E.g., `United States` should be one token, not two
2. Relationship between word frequencies and usefulness in analysis is not obvious
   - ➜ Many highly frequent words aren't particularly meaningful
3. Bag of words creates huge vocabularies, and "sparse" DFMs
   - ➜ Infrequent words are often less useful and slow down computations
   - ➜ Assumes each word has a distinctive, independent meaning; but we know many words have similar meanings

Not all tokens *should* be unigrams

When we tokenise using white spaces, we create tokens that are **unigrams**

- (Technically: after we eliminated "non-words")

We *could* define our tokens differently by choosing a **collocation** such as:

- **bigrams**: pairs of adjacent words
- **trigrams**: triples of adjacent words
- more generally: **n-grams**: n adjacent words

Example: `capital gains tax` can be represented as

- unigrams: `c("capital", "gains", "tax")`
- bigrams: `c("capital gains", "gains tax")`
- trigrams: `c("capital gains tax")`

Not all tokens *should* be unigrams

Why would you want to depart from unigrams?

1. You might want to do your entire analysis using n-grams instead of unigrams
   - In this class, it will be less common
   - But, it could be useful for situations where word order matters like simple text completion
2. Many collocations have independent meaning that you might want to retain in your analysis
   - For example, `United Kingdom` makes more sense left as a bigram than as two unigrams
   - In cases like this: need to manually define which phrases to leave as n-grams

How do you decide which words to collocate into n-grams?

Not all tokens *should* be unigrams

Ask: does a given word occur next to another given word with a higher relative frequency than other words?

➜ If so, then it is a candidate for a collocation

We can use statistical measures of association to determine this

But the key is to distinguish "true collocations" from uninteresting word pairs/triplets/etc, such as "of the"

➜ This, again, requires validation!

Collocations example

$C(w^1\ w^2)$	w^1	w^2
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the
13183	with	the
12622	from	the
11428	New	York
10007	he	said
9775	as	a
9231	is	a
8753	has	been
8573	for	a

Table 5.1 Finding Collocations: Raw Frequency. $C(\cdot)$ is the frequency of something in the corpus.

Source: Manning and Schütze (ch. 5)

Collocations example

$C(w^1\ w^2)$	w^1	w^2
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the
13183	with	the
12622	from	the
11428	New	York
10007	he	said
9775	as	a
9231	is	a
8753	has	been
8573	for	a

Table 5.1 Finding Collocations: Raw Frequency. $C(\cdot)$ is the frequency of something in the corpus.

Identifying collocations in `quanteda`

Lots of ways to use statistical measures to find collocations

- ➜ But: you still always have to validate for your purpose!

`quanteda` has a function `textstat_collocations()` that we can use to find high frequency collocations

- ➜ Use `size=n` argument to indicate what size n-gram you want

After getting a list of common collocations:

- ➜ Filter to the most important ones – justify your choices!
- ➜ When you decide which to keep, "merge" words together so they do not get broken up when tokenising
  - ➜ Common to use underscores or periods, but careful when discarding punctuation

Identifying collocations in quanteda

Let's find the common bigrams in the Trump tweet corpus

```
# A tibble: 5,335 x 6
   collocation   count count_nested length lambda     z
   <chr>         <int>        <int>  <dbl>  <dbl> <dbl>
 1 fake news       217            0      2   8.01  40.1
 2 tax cut         130            0      2   7.11  35.8
 3 make america     99            0      2   5.60  35.6
 4 unit state      101            0      2   7.25  30.6
 5 north korea     116            0      2   9.32  30.0
 6 news media       63            0      2   5.02  28.7
 7 america great    91            0      2   4.00  28.5
 8 great again      98            0      2   4.75  28.2
 9 illeg immigr     39            0      2   6.56  26.1
10 work hard        46            0      2   5.42  25.9
# i 5,325 more rows
```

Identifying collocations in quanteda

A subjective (but reasonable) judgement for most QTA projects with this Trump tweet data:

- ➜ keep `United States` as bigram

For example:

```
Having a great time hosting Prime Minister Shinzo Abe in the
United States! https://t.co/Fvjsac89qS https://t.co/OupKmRRuTI
https://t.co/smGrnWakWQ
```

Modified to:

```
Having a great time hosting Prime Minister Shinzo Abe in the
United_States! https://t.co/Fvjsac89qS https://t.co/OupKmRRuTI
https://t.co/smGrnWakWQ
```

Reduce complexity: filter by frequency

Bag of words creates sparse DFMs

- ➜ Computationally inefficient, plus rare words are generally uninformative

So, you can filter the vocabulary by dropping certain features

Already saw examples of filtering based on intuitions:

- ➜ Removed “stop words” because they represent linguistic connectors of no substantive content
- ➜ Consolidate words into “root” forms via stemming/lemmatisation

Or just choose words to filter based on your task

There’s a more structured way to filter using *frequencies*

Reduce complexity: filter by frequency

Document frequency of term j counts how many documents contain the feature j:

$$\mathsf{df}_j = |\{i : W_{ij} > 0\}|$$

Total term frequency of term j counts how many times the feature j appears in the corpus:

$$\mathsf{tf}_j = \sum_i W_{ij}$$

Note: there is ambiguity about how these terms are used/defined

- "term frequency" can also refer to the frequency of a term j in a specific document i
- multiple ways to calculate using various normalisations

Filtering uncommon words from Trump tweet corpus

Let's filter out any feature that doesn't appear in at least two documents or that doesn't appear at least twice in the DFM

Original:

```
Document-feature matrix of: 212 documents, 1,039 features (98.94%
  sparse) and 0 docvars.
                     features
docs                  american #happynewyear mani bless look
  2017-01-01 05:00:10        1             1    1     1    1
  2017-01-01 05:39:13        0             1    0     0    0
  2017-01-01 05:43:23        0             0    0     1    1
  2017-01-01 05:44:17        0             0    0     0    0
  2017-01-01 06:49:33        0             0    0     0    0
  2017-01-01 06:49:49        0             0    0     0    0
[ reached max_ndoc ... 206 more documents, reached max_nfeat ...
  1,034 more features ]
```

Filtering uncommon words from Trump tweet corpus

Let's filter out any feature that doesn't appear in at least two documents or that doesn't appear at least twice in the DFM

Trimmed dfm with infrequent features filtered:

```
Document-feature matrix of: 212 documents, 103 features (95.86%
  sparse) and 0 docvars.
                     features
docs                  american mani look forward wonder
  2017-01-01 05:00:10        1    1    1       1      1
  2017-01-01 05:39:13        0    0    0       0      0
  2017-01-01 05:43:23        0    0    1       1      0
  2017-01-01 05:44:17        0    0    0       0      0
  2017-01-01 06:49:33        0    0    0       0      0
  2017-01-01 06:49:49        0    0    0       0      0
[ reached max_ndoc ... 206 more documents, reached max_nfeat ...
  98 more features ]
```

But raw frequency is a bad representation

Raw frequency is clearly useful: if sugar appears a lot near apricot, that's useful information

But overly frequent words like "the", "it", or "they" are not very informative about content

Some terms carry more information about contents

Creates a kind of "**word frequency paradox**"

Can create issues in analysis; high frequency words will be given a lot of weight because word counts are higher

- ➜ Filtering low frequency words still keeps all the useless high frequency words!

Zipf's law of word frequency

Zipf's law characterises relationship between word rank and word frequency

Skipping a lot of technical details, but simplest example:

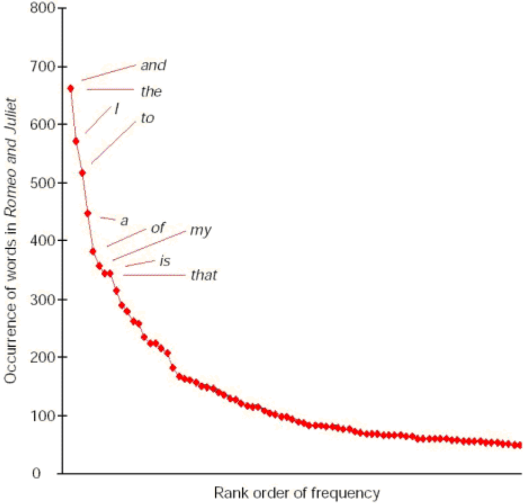
- In a corpus, word frequency is inversely related to word rank

$$\text{frequency} = 1/\text{rank}$$

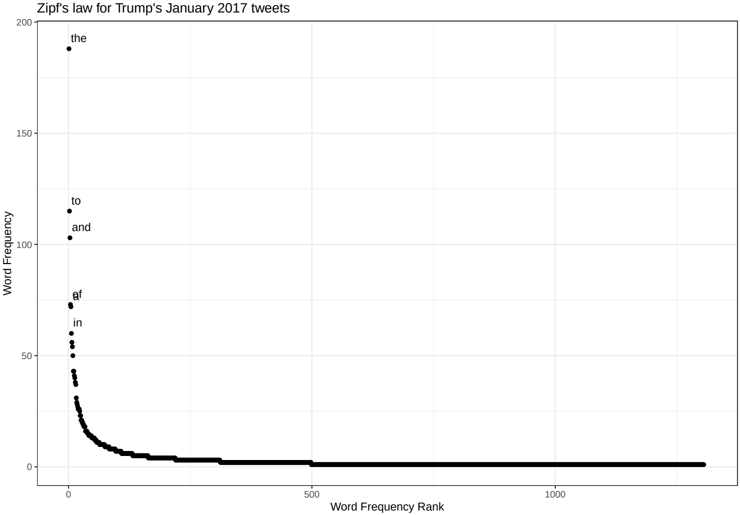
A fun fact: this relationship also holds for other measures, like population of global cities

Power-law/Zipf Distribution of Word Frequency

Where are the most informative words on this plot?

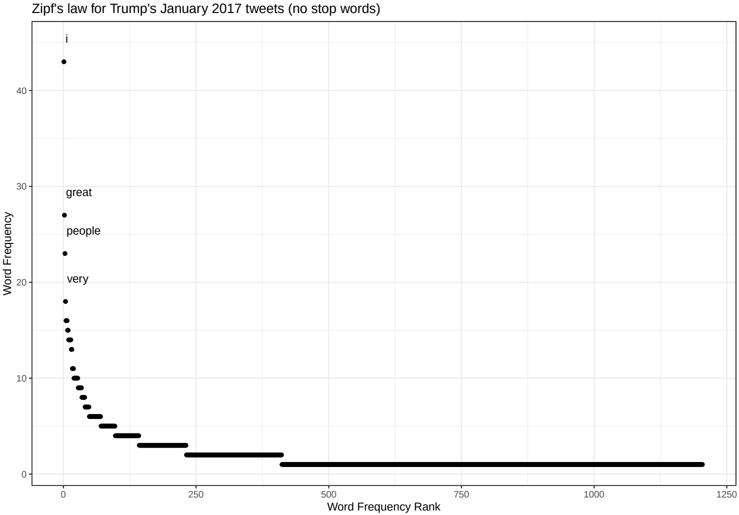


Zipf's law and Trump's tweets



Zipf's law and Trump's tweets

Removing stop words *helps*



Weighting strategies

Filtering features (based on stop words, term/document frequency) is a kind of **weighting strategy**

➜ Each filtered feature is weighted to zero!

But maybe we want to do something more subtle

In particular how might we simultaneously address the problems of:

➜ very low frequency tokens that are not informative, and
➜ very high frequency tokems that are not informative?

Weighting with tf-idf

Can *weight* the cells of a DFM using **term frequency inverse document frequency (tf-idf) weighting**

$$\text{tfidf}_{ij} = W_{ij} \times \log\left(\frac{N}{|\{i : W_{ij} > 0\}|}\right)$$

Replace each cell of a DFM (W_{ij}) with tfidf_{ij}

More weight to word counts with: high term frequency in a document AND low document frequency of the term in the whole corpus

➜ weights tend to filter out common terms

There are other weighting schemes, such as Okapi BM25 and SMART weighting scheme, mostly used in CS applications

Weighting DFM of Trump tweets

Recall the DFM of January 2017 Trump tweets

First: calculate idf (manually)

Highest idf words:

```
# A tibble: 1,039 x 2
  feature     idf
  <chr>     <dbl>
1 prosper    2.33
2 behalf     2.33
3 #potus     2.33
4 teamtrump  2.33
5 https      2.33
# i 1,034 more rows
```

Lowest idf words:

```
# A tibble: 1,039 x 2
  feature   idf
  <chr>   <dbl>
1 i       0.795
2 great   0.928
3 peopl   0.965
4 thank   1.10
5 trump   1.12
# i 1,034 more rows
```

Weighting DFM of Trump tweets

Second: replace cells of DFM with calculated tf-idf

Original DFM:

```
Document-feature matrix of: 212 documents, 1,039 features (98.94%
  sparse) and 0 docvars.
                     features
docs                  american #happynewyear mani bless look
  2017-01-01 05:00:10        1             1    1     1    1
  2017-01-01 05:39:13        0             1    0     0    0
  2017-01-01 05:43:23        0             0    0     1    1
  2017-01-01 05:44:17        0             0    0     0    0
  2017-01-01 06:49:33        0             0    0     0    0
  2017-01-01 06:49:49        0             0    0     0    0
[ reached max_ndoc ... 206 more documents, reached max_nfeat ...
  1,034 more features ]
```

Weighting DFM of Trump tweets

Second: replace cells of DFM with calculated tf-idf

Tf-idf weighted DFM using `dfm_tfidf()` function from quanteda:

```
Document-feature matrix of: 212 documents, 1,039 features (98.94%
  sparse) and 0 docvars.
                     features
docs                  american #happynewyear     mani    bless
  2017-01-01 05:00:10 1.423246      2.025306 1.284943 2.025306
  2017-01-01 05:39:13 0             2.025306 0        0
  2017-01-01 05:43:23 0             0        0        2.025306
  2017-01-01 05:44:17 0             0        0        0
  2017-01-01 06:49:33 0             0        0        0
  2017-01-01 06:49:49 0             0        0        0
[ reached max_ndoc ... 206 more documents, reached max_nfeat ...
  1,035 more features ]
```

Word embeddings

There is a core *conceptual* “problem” with bag of words that creates all these complications

- ➜ “problem” is in quotes because in reality, bag of words works pretty well most of the time

Bag of words conceives of each word as having a distinct and unique meaning

In math terms: we treat words as **one-hot encodings**

Word embeddings

Concrete example: consider a vocabulary of three words:
`c("cat", "dog", "rat")`

Can represent each word in **vector format**:

- The word `cat` is `(1,0,0)`
- The word `dog` is `(0,1,0)`
- The word `rat` is `(0,0,1)`

(Should be obvious why this is called "one-hot encoding")

These are *orthogonal*: zero similarity between the words

- Similar concept to a "dummy variable"

Word embeddings

But what if words are actually representations of a smaller group of concepts, where each word is a *mix* of concepts?

E.g., what if these words can be represented in a two dimensional **embedding** (e.g., two distinct “concepts”)

Then, you might have

- The word `cat` represented by `(0.23,0.77)`
- The word `dog` represented by `(0.39,0.61)`
- The word `rat` represented by `(0.82,0.18)`

Word embeddings have to be *estimated* from a corpus

You can then use word embeddings to represent documents in a DFM, but this is a little more complicated

More at the end of the course

Describing texts

Quantities for describing texts

- **Length**: in characters, words, lines, sentences, paragraphs, pages, sections, chapters, etc.
- **Term frequency**: counts or proportions of words
- **Lexical diversity**: measuring the diverity of the types in a vocabulary, i.e. "richness"
- **Readability**: measuring, roughly speaking, the complexity of texts

Lexical diversity

Consider a corpus that has: vocabulary with J types, and has $W = \sum_i \sum_j W_{ij}$ total tokens

- We're considering lexical diversity for a corpus
- Many applications measure lexical diversity for individual texts (and then compare across texts)
- So, for single document i, the vocabulary has J_i types (just those appearing in J) and $W_i = \sum_j W_{ij}$ total tokens

Basic measure is **Type-to-Token ratio** (**TTR**): $\mathrm{TTR} = \frac{J}{W}$

- Problem: This is very sensitive to overall document length, as shorter texts may exhibit fewer word repetitions
- Special problem: length may relate to the introduction of additional subjects, which will also increase richness

Lexical diversity

Some alternatives:

- **Guiraud's root TTR** introduces a penalty for long corupuses: $\mathrm{R} = \frac{J}{\sqrt{W}}$
- **vocd-D**: randomly sample *fixed* numbers of tokens; calculate mean TTR for each sample size; then summarize relationship between mean TTR and sample size with D measure
  - See McCarthy and Jarvis (2007)
- **Measure of textual lexical diversity (MTLD)**: partition text by sequences of words yielding a TTR of 0.72; calculate size of partition, its "factor count" (F); then calculate a MTLD value as $\mathrm{MTLD} = \frac{W}{F}$
  - See McCarthy and Jarvis (2010)

Complexity and Readability

- Use a combination of syllables and sentence length to indicate "readability" in terms of complexity
- Common in educational research, but could also be used to describe textual complexity
- Most use some sort of sample
- No natural scale, so most are calibrated in terms of some interpretable metric

Flesch-Kincaid readability index

Now consider corpus with J types, W total words, S total sentences and Y total syllables

Flesch Reading Ease Score

$$\mathrm{F} = 206.835 - 1.015\left(\frac{W}{S}\right) - 84.6\left(\frac{Y}{W}\right)$$

- Interpretation: 0-30: university level; 60-70: understandable by 13-15 year olds; and 90-100 easily understood by an 11-year old student.

Flesch-Kincaid Readability Score modifies by rescaling to the US educational grade levels (1-12):

$$\mathrm{FK} = 0.39\left(\frac{W}{S}\right) + 11.8\left(\frac{Y}{W}\right) - 15.59$$

Two approaches to analysis with DFMs

Two approaches to analysis with DFMs

Once we have a DFM in hand, what do we do with it?

➜ this is subject of the rest of this class!!

But we can zoom out at a high level and identify two broad approaches:

1. Probabilistic models
2. Vector space models

Probabilistic models

Basic idea: documents are “draws” from a probability distribution

So, a dfm is a sample with all the associated sampling properties

Think back to your stats courses:

- A **data generating process** is a probability distribution that (we assume) captures way real-life data occurs
- Real-life datasets are conceptualised as **samples** from a DGP
- In some sense, the DGP is what we care about since it tells us “how things work”
- We use real-life data to try to learn about the DGP

Language models

Language is often modeled using probabilistic models

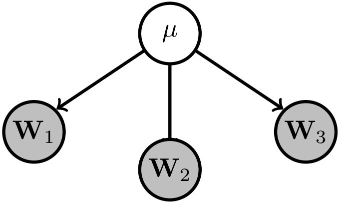
One of the simplest language models: assume each document i is a "draw" from a **multinominal distribution** with three parameters:

- ➜ $J \rightarrow$ the vocabulary size
- ➜ $\mu \rightarrow$ probability each type in the vocabulary is drawn
- ➜ $M_i \rightarrow$ the document length (number of tokens)

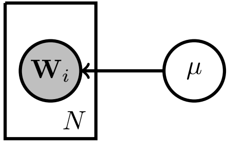
Using this distribution makes sense under bag of words, since each token is assumed to be drawn independently

Graphing probability models of language

Three-document model



N-document model



Very simple example

Suppose a vocabulary with $J = 3$ types: `cat`, `dog`, `rat`

Further suppose that `cat` is twice as likely as `dog` or `rat` (which are equally likely):

$$\mu = (\mu_{\textsf{cat}} = 0.50, \mu_{\textsf{dog}} = 0.25, \mu_{\textsf{rat}} = 0.25)$$

Notice we've made a bunch of assumption about the way documents are made

Very simple example

Simulate drawing four documents of different lengths:

```r
v <- c("cat","dog","rat") # the types
mu <- c(0.5,0.25,0.25) # the assumed type probabilities
M <- c(2,3,2,6) # draw a sample of 4 docs with varying lengths
set.seed(2025)
for(i in 1:length(M)){
  w <- rmultinom(1,M[i],mu)
  wi <- c(rep(v[1],w[1,1]), rep(v[2],w[2,1]), rep(v[3],w[3,1]))
  print(paste0("Document ",i,": ",
               paste0(sample(wi), collapse = " ")))
}
```

```
[1] "Document 1: rat cat"
[1] "Document 2: cat cat dog"
[1] "Document 3: rat cat"
[1] "Document 4: dog rat rat rat cat dog"
```

Very simple example

These documents could be put into a DFM:

	cat	dog	rat	
Document 1	1	0	1	<- "rat cat"
Document 2	2	1	0	<- "cat cat dog"
Document 3	1	0	1	<- "rat cat"
Document 4	1	2	3	<- "dog rat rat rat cat dog"

Each document can be represented mathematically in vector form

For example, Documents 1 and 4:

$$\mathbf{W}_1 = (1, 0, 1) \qquad \mathbf{W}_4 = (1, 2, 3)$$

Calculating important stuff

It is common to use the multinomial distribution to model language because it's easy to work with

➜ Again: keep in mind we're assuming bag of words!

We can look up various important formulas for multinomial distributions

E.g., calculate the probability of getting a particular document $\mathbf{W}_i$, given μ:

$$p(\mathbf{W}_i|\mu) = \frac{M_i!}{\prod_{j=1}^{J}\left(W_{ij}!\right)} \prod_{j=1}^{J} \left(\mu_j^{W_{ij}}\right)$$

➜ Again: keep in mind we're assuming bag of words!

Calculating important stuff

Back to our simple example with three word vocabulary:

$$\mu = (\mu_{\mathsf{cat}} = 0.50, \mu_{\mathsf{dog}} = 0.25, \mu_{\mathsf{rat}} = 0.25)$$

What is probability of getting a this document: `cat, rat, dog, cat`?

- ➜ In vector form, this is $\mathbf{W}_i = (2, 1, 1)$?

$$p(\mathbf{W}_i|\mu) = \frac{(2+1+1)!}{2! \times 1! \times 1!} \left[0.5^2 \times 0.25^1 \times 0.25^1\right] = \frac{12}{64} = 0.1875$$

Very simple example

We can calculate in R as well

```
v <- c("cat","dog","rat") # the types
m <- c(0.5,0.25,0.25) # the assumed type probabilities
dmultinom(c(2,1,1),prob=m)
```

```
[1] 0.1875
```

Estimating models with data

In real-world situations, we don't usually know the DGP

- We don't actually know μ and we need to estimate it from data

How do we do this?

It's simple: $\hat{\mu}_j = W_{ij}/M$

Why does this matter?

- We want to estimate the parameters of the DGP so we have a rough idea how documents show up in our data
- This then enables us to do some cool things to "fill in" gaps in our knowledge

Example: the Federalist papers

When the US was established, three people wrote a set of essays called The Federalist Papers in support of the new US Constitution

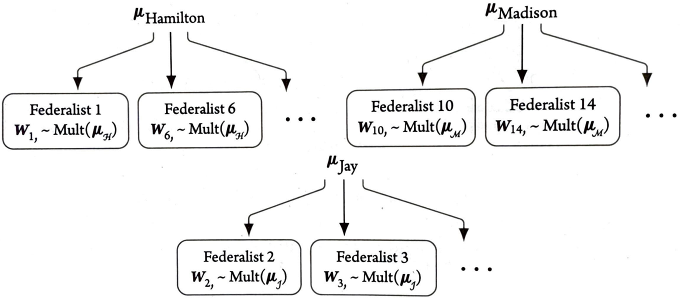
- The people: Alexander Hamilton, John Jay, James Madison
- There were 85, none of them had bylines (all “Publius”)
- Historical records tell us who wrote 73 of them; remainder are contested

Mosteller and Wallace (1963) use a probability model to try to figure out who wrote the unlabeled Federalist papers

Example: the Federalist papers

Let's look at a simplified version of this task where we set up a probabilistic model assuming each writer has distinctive style

Mathematically, each writer's μ is different: $\mu_{\mathcal{H}}$, $\mu_{\mathcal{J}}$, $\mu_{\mathcal{M}}$



Source: Figure 6.3 of GRS (p. 64)

Example: the Federalist papers

The basic process:

Step 1: estimate μ for each writer ("figure out their writing style") using real-world data, generating estimates: $\hat{\mu}_{\mathcal{H}}$, $\hat{\mu}_{\mathcal{J}}$, $\hat{\mu}_{\mathcal{M}}$

Step 2: see which model best explains the unlabeled ones by calculating probability each person was the author of the unlabeled documents

If we assume each author's style is constant across documents, things get real simple

- Just add together all the tokens — each author writes one big document
- (This is due to properties of multinomial)

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

What is the probability that Hamilton wrote the unlabeled ones?

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

What is the probability that Hamilton wrote the unlabeled ones?

Step 1: estimate Hamilton's "language model" $\hat{\mu}_{\mathcal{H}}$

$$\hat{\mu}_{\mathcal{H}} = \left(\frac{859}{859+102+374}, \frac{102}{859+102+374}, \frac{374}{859+102+374} \right) \approx (0.64, 0.08, 0.28)$$

Example: the Federalist papers

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7
Unlabeled	15	2	0

What is the probability that Hamilton wrote the unlabeled ones?

Step 1: estimate Hamilton's "language model" $\hat{\mu}_{\mathcal{H}}$

$$\hat{\mu}_{\mathcal{H}} = \left(\frac{859}{859 + 102 + 374}, \frac{102}{859 + 102 + 374}, \frac{374}{859 + 102 + 374} \right) \approx (0.64, 0.08, 0.28)$$

Step 2: calculate probability Hamilton wrote unlabeled documents

$$\Pr(\mathbf{W}_{\text{Unlabeled}} | \hat{\mu}_{\mathcal{H}}) = \frac{(15 + 2 + 0)!}{15! \times 2! \times 0!} \times 0.64^1 5 \times 0.08^2 \times 0.28^0 = 0.001$$

Vector space models

There's another way to mathematically analyse DFMs

- ➜ Treat documents as vectors in a "space"
- ➜ The space has J dimensions, one for each feature
- ➜ Each document is a vector of word counts

You already saw this, for example document `cat, rat, dog, cat`

- ➜ In vector form, this is $\mathbf{W}_i = (2, 1, 1)$

Vector space models

We are not going to spend a ton of time on this (yet)

But conceptualising words as vectors allows us to quantify

- how similar texts are
- how distant text are

Useful for a wide range of things, such as clustering (see week 7)