Topic 2: Quantifying Texts

LSE MY459: Computational Text Analysis and Large Language Models

https://lse-my459.github.io/

Ryan Hübert
Associate Professor of Methodology

Slides last updated: 26 January 2026

Don't forget

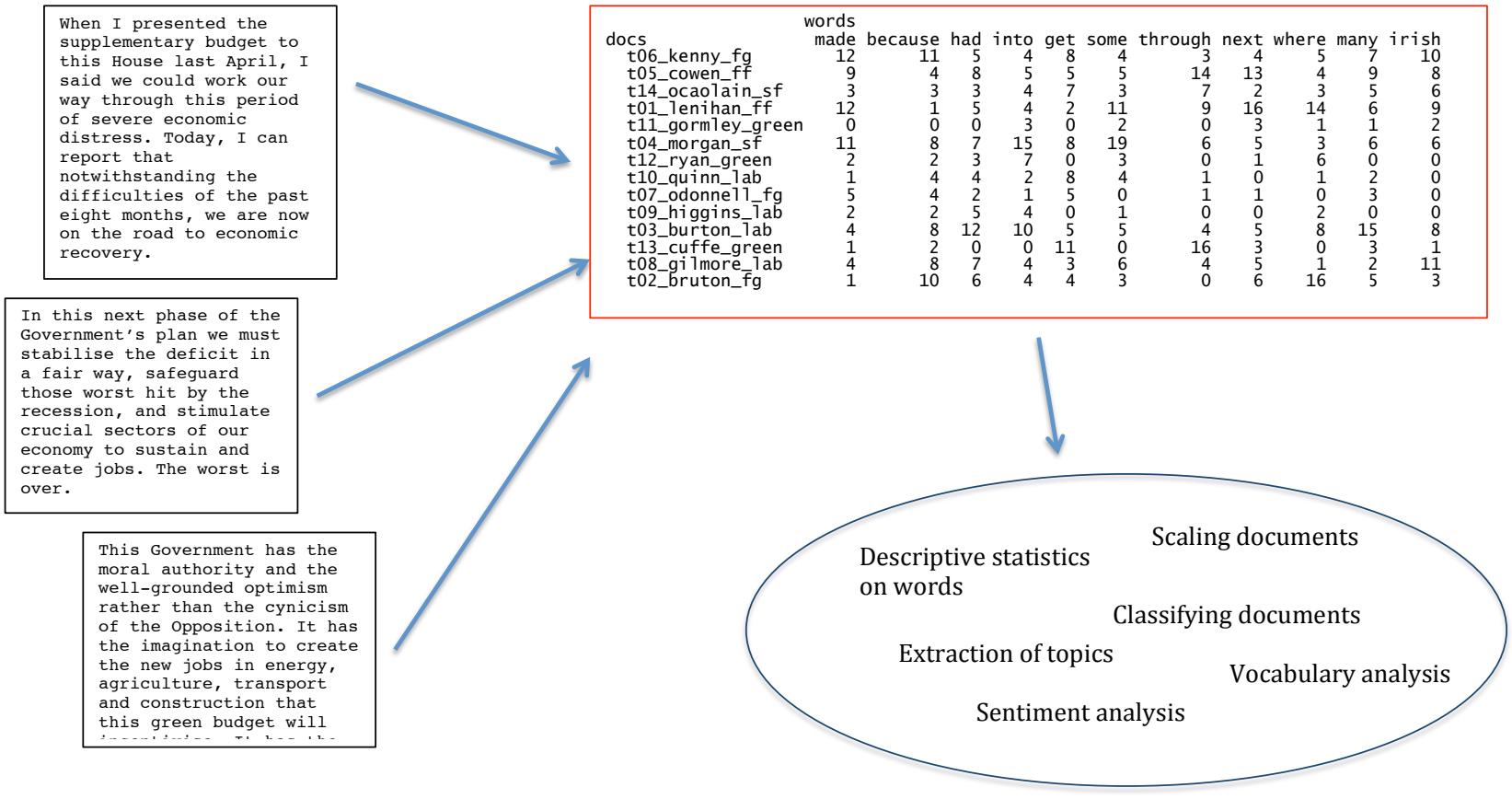
For this week's seminar:

- Bring your laptop
- Ensure that you have a working Python set up
  - See information on Moodle
  - You should be able to open an `.ipynb` notebook and run cells of python code

We are agnostic about how you set up Python on your computer, but you must get it working before seminar

Quantifying texts

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



Document selection (rows of DFM)

Strategies for selecting units of textual analysis

A **document** is the fundamental unit of analysis for analysing texts, for example:

- ➜ 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 our first assumption: 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 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 `data` repo on the course GitHub

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

- ➜ Need to do some wrangling to get into Python (seminar!)

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;amp; many blessings to
              you all! Looking forward to a wonderful &amp;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":[8.1542233510816e+17],
                         "id_str":["815422333510815746"],
                         "indices":[[149],[172]],
                "media_url":["http://pbs.twimg.com/media/C1D2SsLVEAIn1UJ.jp
                "media_url_https":["https://pbs.twimg.com/media/C1D2SsLVEAI
                         "url":["https://t.co/UaBFaoDYHe"],
                         ...
```

Defining document features

Recall our second assumption: 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

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 task!

Word embeddings

Bag of words: each feature is a word and each word has a distinct, unique meaning

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

Concrete example: consider a vocabulary of three words:
`["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

Word embeddings

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

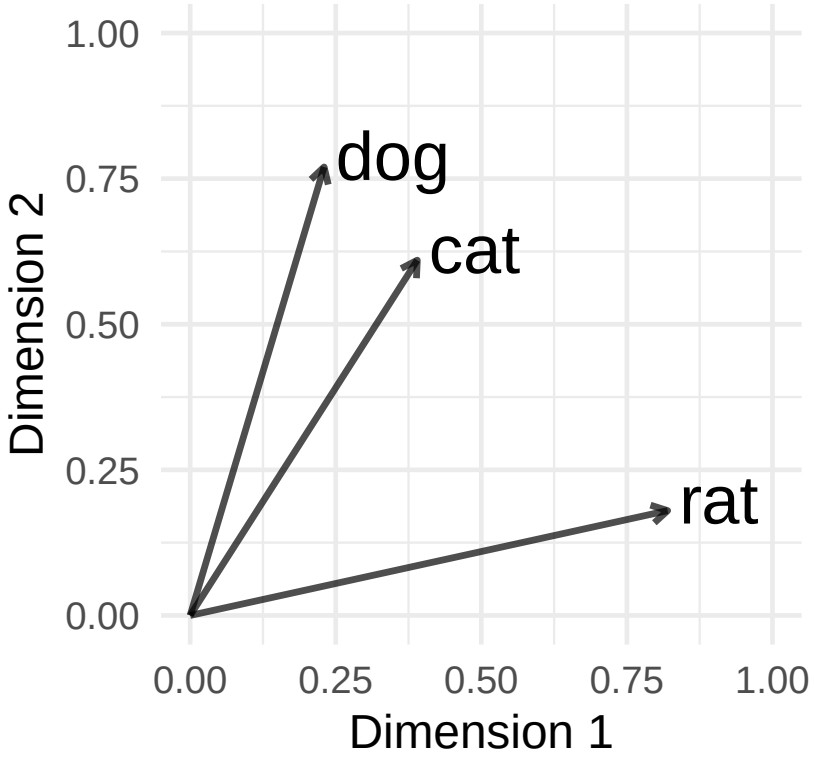
E.g., what if these words can be represented in a two dimensional **embedding** (e.g., two distinct "concepts")

Then, you might have

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

Word embeddings have to be *estimated* from a corpus

Word embeddings



Machine "learns" these dimensions from patterns in corpus

Analyst interprets their meaning

- ➜ Dimension 1 might represent something like "wildness" or "pest-like" traits.
- ➜ Dimension 2 might represent "domesticated companionship."

Allows you to measure semantic similarity of words using concepts from linear algebra, like **cosine similarity**

- ➜ Vectors pointing in same direction are very "similar"
- ➜ One-hot encodings are all perfectly dissimilar using this metric
- ➜ We'll come back to this later in course

Implementing bag of words

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

Tokenising Trump's tweet

Use white space to tokenise:

```
['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

```
['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']
```

Reduce complexity: removing stop words

It is very common to remove stop words when analysing texts

- ➜ But this depends on your task!
- ➜ See Pennebaker (2011)

There are different lists of stop words, some longer, some shorter

A commonly used list comes from the `nltk` module

- ➜ Module has a very large set of tools for working with texts, for example:

```
import nltk
# nltk.download('stopwords') # Only first time
sw = nltk.corpus.stopwords.words("english")
sw[0:8] # show the first 8
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all']
```

Removing stop words from Trump's tweet

We can remove all the stop words from the Trump tweet using the `nltk` list of stop words

```
['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`

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`
- ➜ In modern contexts, the **Snowball stemmer** is used, which is effectively Porter v2

There are other corpus-based, statistical, and mixed approaches designed to overcome these limitations, but this may be overrated:

- ➜ Schofield and Mimno (2016): "stemmers produce no meaningful improvement in likelihood and coherence [of topic models] and in fact can degrade topic stability"

Reduce complexity: stemming and lemmatisation

Again, the `nltk` module has tools for stemming and lemmatisation

We can use the `nltk.stem.snowball` submodule to stem the Trump tweet using the Snowball stemmer

```
['american', '#happynewyear', 'mani', 'bless', 'look', 'forward',
 'wonder', 'prosper', 'work', 'togeth', '#maga']
```

The stemmer did many transformations, e.g.:

- `americans` → `american`
- `many` → `mani`
- `blessing` → `bless`
- `together` → `togeth`

Reduce complexity: edit distance

As we see with stemming (and lemmatising), quantifying texts requires categorical decisions about token identity

For example, when `blessing` → `bless`, we are making a decision to treat the words "bless" and "blessing" as the same token

But raw text is often messy, and we may want to do *even more* to figure out what words should be treated the same

For example: if your corpus contains a lot of misspelled words or typos, you may want a technique for consolidating, such as:

➜ `togesther` → `together`

To identify such instances, we might turn to more general measures of **string similarity**

Reduce complexity: edit distance

Edit distance: number of operations needed to transform one string into another

- ➜ Good for finding typos and such
- ➜ Also useful for measuring **text reuse** (e.g., plagiarism)

Most common edit distance metric is **Levenshtein distance**

Example: Levenshtein distance between `kitten` and `sitting` is 3

- ➜ `kitten` → `sitten` (substitution of "s" for "k")
- ➜ `sitten` → `sittin` (substitution of "i" for "e")
- ➜ `sittin` → `sitting` (insertion of "g" at the end).

You can calculate Levenshtein distance with the `edit_distance()` function from the `nltk.metrics.distance` submodule

Source for example: Wikipedia

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:

datetime	american	#happynewyear	mani	bless	...	asham
						...
2017-01-01 05:00:10	1	1	1	1	...	0
2017-01-01 05:39:13	0	1	0	0	...	0
2017-01-01 05:43:23	0	0	0	1	...	0
...						
2017-01-31 00:45:50	0	0	0	0	...	0
2017-01-31 11:21:52	0	0	0	0	...	0
2017-01-31 11:27:02	0	0	0	0	...	1

[212 rows x 993 columns]

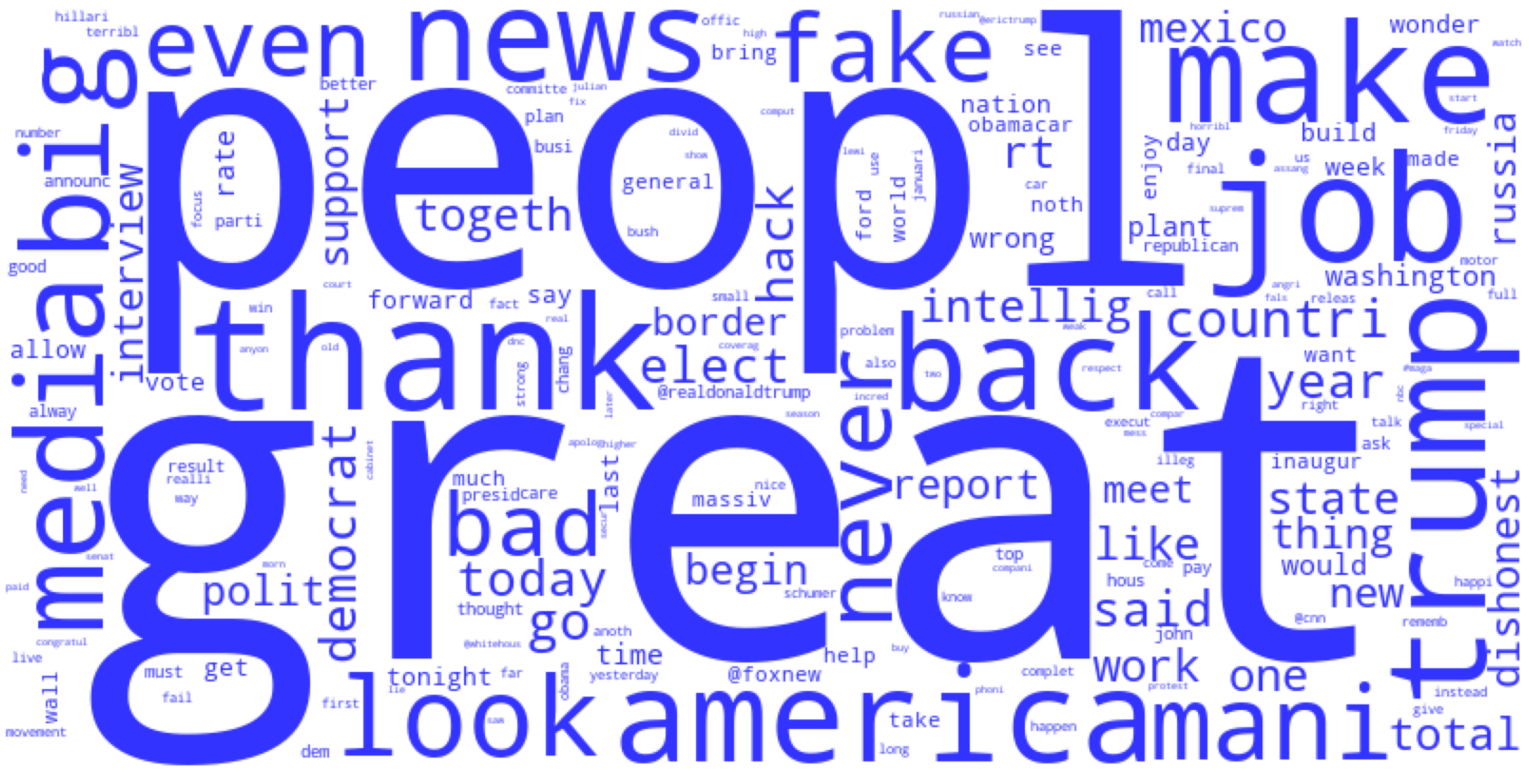
This DFM has $N = 212$ documents (tweets) and $J = 993$ features (tokens)

➜ 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

- They depict the most commonly used tokens in the dfm
- Can use raw frequency or rank (or a combination)



Issues in feature selection

How you quantify your texts 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

Collocations

Not all tokens *should* be unigrams

We are treating tokens as our document features

When we tokenise using white spaces, we get **unigrams**

We *could* define our document features differently by using **collocations** 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: `['capital', 'gains', 'tax']`
- bigrams: `['capital gains', 'gains tax']`
- trigrams: `['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!

Important collocations

$C(w^1\ w^2)$	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)

Important collocations

$C(w^1\ w^2)$	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)

Identifying collocations

Yet again, `nltk` has tools for working with collocations

E.g., define a document's features as bigrams (which `nltk` represents as `tuple` objects):

```
[('TO', 'ALL'), ('ALL', 'AMERICANS-'), ('AMERICANS-', '#HappyNewYear'),
 ('#HappyNewYear', '&amp;'), ('&amp;', 'many'), ('many', 'blessings'),
 ('blessings', 'to'), ('to', 'you'), ('you', 'all!'), ('all!', 'Looking'),
 ('Looking', 'forward'), ('forward', 'to'), ('to', 'a'), ('a', 'wonderful'),
 ('wonderful', '&amp;'), ('&amp;', 'prosperous'), ('prosperous', '2017'),
 ('2017', 'as'), ('as', 'we'), ('we', 'work'), ('work', 'together'),
 ('together', 'to'), ('to', '#MAGA\x1f '),
 ('#MAGA\x1f ', 'https://t.co/UaBFaoDYHe')]
```

You might also decide to do this after some preprocessing

```
[('americans', '#happynewyear'), ('#happynewyear', 'many'),
 ('many', 'blessings'), ('blessings', 'looking'), ('looking', 'forward'),
 ('forward', 'wonderful'), ('wonderful', 'prosperous'),
 ('prosperous', 'work'), ('work', 'together'), ('together', '#maga')]
```

Identifying collocations

You can also count the frequency of collocations across a corpus then:

- ➜ 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

Lots of ways to use statistical measures to find "important" collocations in a corpus

Identifying collocations

Most common bigrams in the Trump tweet corpus (after preprocessing):

	collocation	freq
0	('fake', 'news')	217
1	('tax', 'cut')	129
2	('north', 'korea')	114
3	('unit', 'state')	100
4	('make', 'america')	99
5	('america', 'great')	92
6	('white', 'hous')	77
7	('witch', 'hunt')	76
8	('look', 'forward')	71
9	('great', 'honor')	69
10	('news', 'media')	63
11	('prime', 'minist')	56
12	('rt', '@foxandfriend')	52
13	('work', 'hard')	51
14	('rt', '@realdonaldtrump')	49

Identifying collocations

Most common trigrams in the Trump tweet corpus (after preprocessing):

	collocation	freq
0	('make', 'america', 'great')	87
1	('fake', 'news', 'media')	60
2	('tax', 'cut', 'reform')	21
3	('great', 'honor', 'welcom')	21
4	('job', 'job', 'job')	20
5	('massiv', 'tax', 'cut')	20
6	('kim', 'jong', 'un')	19
7	('today', 'great', 'honor')	19
8	('crook', 'hillari', 'clinton')	17
9	('togeth', 'make', 'america')	16
10	('fake', 'news', 'cnn')	14
11	('tax', 'cut', 'bill')	13
12	('prime', 'minist', 'abe')	12
13	('presid', 'donald', 'j')	12
14	('donald', 'j', 'trump')	12

Identifying collocations

A subjective (but reasonable) judgement for most analysis 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
```

Weighting

Trimming by frequency

Bag of words creates sparse DFMs

- ➜ Computationally inefficient, plus rare words are generally uninformative

So, you can "trim" the vocabulary by dropping certain features

Already saw examples of trimming 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 trim based on your task

There's a more structured way to trim using *frequencies*

Trimming by frequency

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

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

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

$$\mathrm{ttf}_j = \sum_i W_{ij}$$

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

➜ "term frequency" (without "total") also refers to the frequency of a term j in a specific document i, i.e. W_{ij}

Trimming uncommon words from Trump tweet corpus

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

Original:

	american	#happynewyear	mani	bless	...	asham
datetime						...
2017-01-01 05:00:10	1	1	1	1	...	0
2017-01-01 05:39:13	0	1	0	0	...	0
2017-01-01 05:43:23	0	0	0	1	...	0
...						
2017-01-31 00:45:50	0	0	0	0	...	0
2017-01-31 11:21:52	0	0	0	0	...	0
2017-01-31 11:27:02	0	0	0	0	...	1

[212 rows x 993 columns]

Trimming uncommon words from Trump tweet corpus

Let's trim 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 trimmed:

datetime	american	#happynewyear	mani	bless	...	tear
						...
2017-01-01 05:00:10	1	1	1	1	...	0
2017-01-01 05:39:13	0	1	0	0	...	0
2017-01-01 05:43:23	0	0	0	1	...	0
...						
2017-01-31 00:45:50	0	0	0	0	...	0
2017-01-31 11:21:52	0	0	0	0	...	1
2017-01-31 11:27:02	0	0	0	0	...	0

[212 rows x 386 columns]

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

➜ Trimming 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:

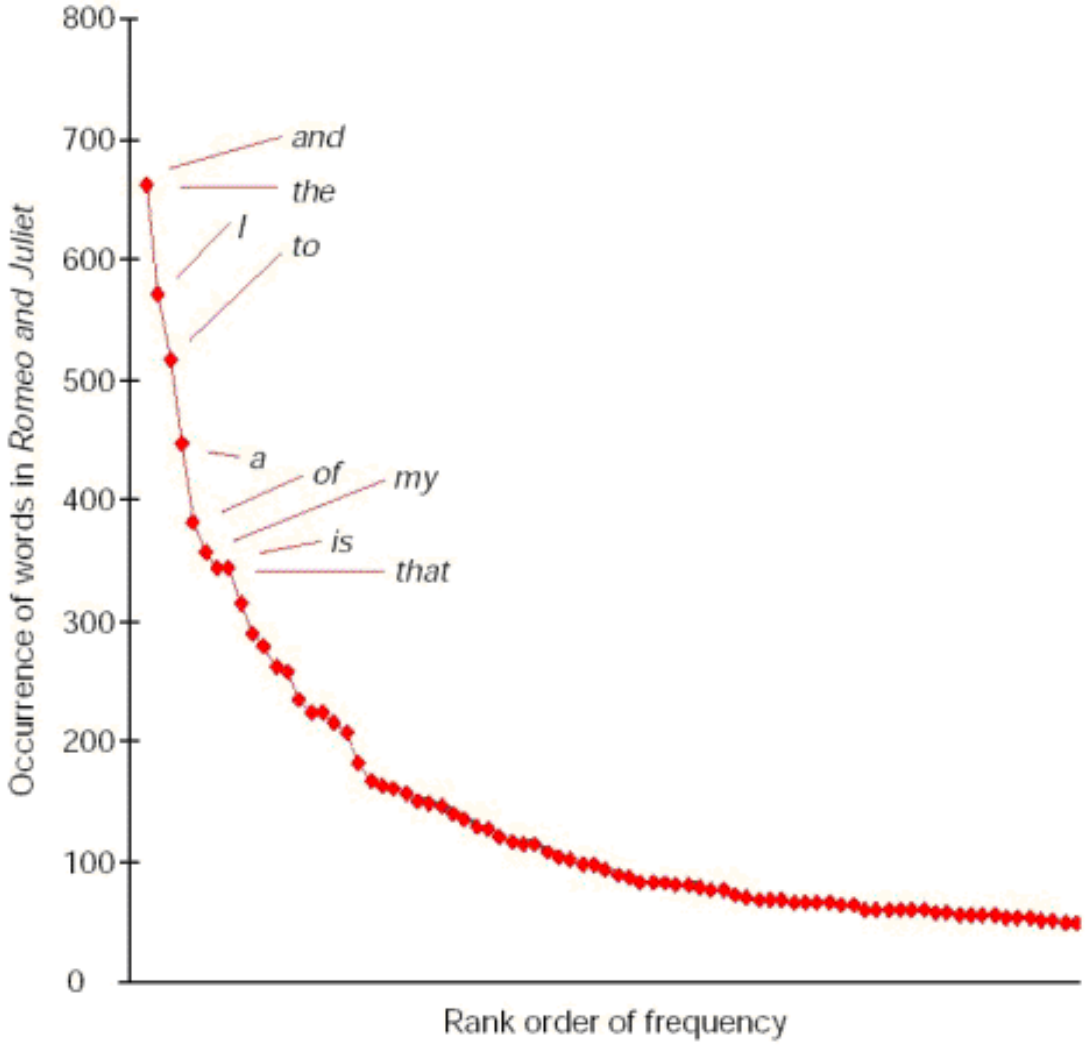
➜ In a corpus, word frequency is inversely related to word rank

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

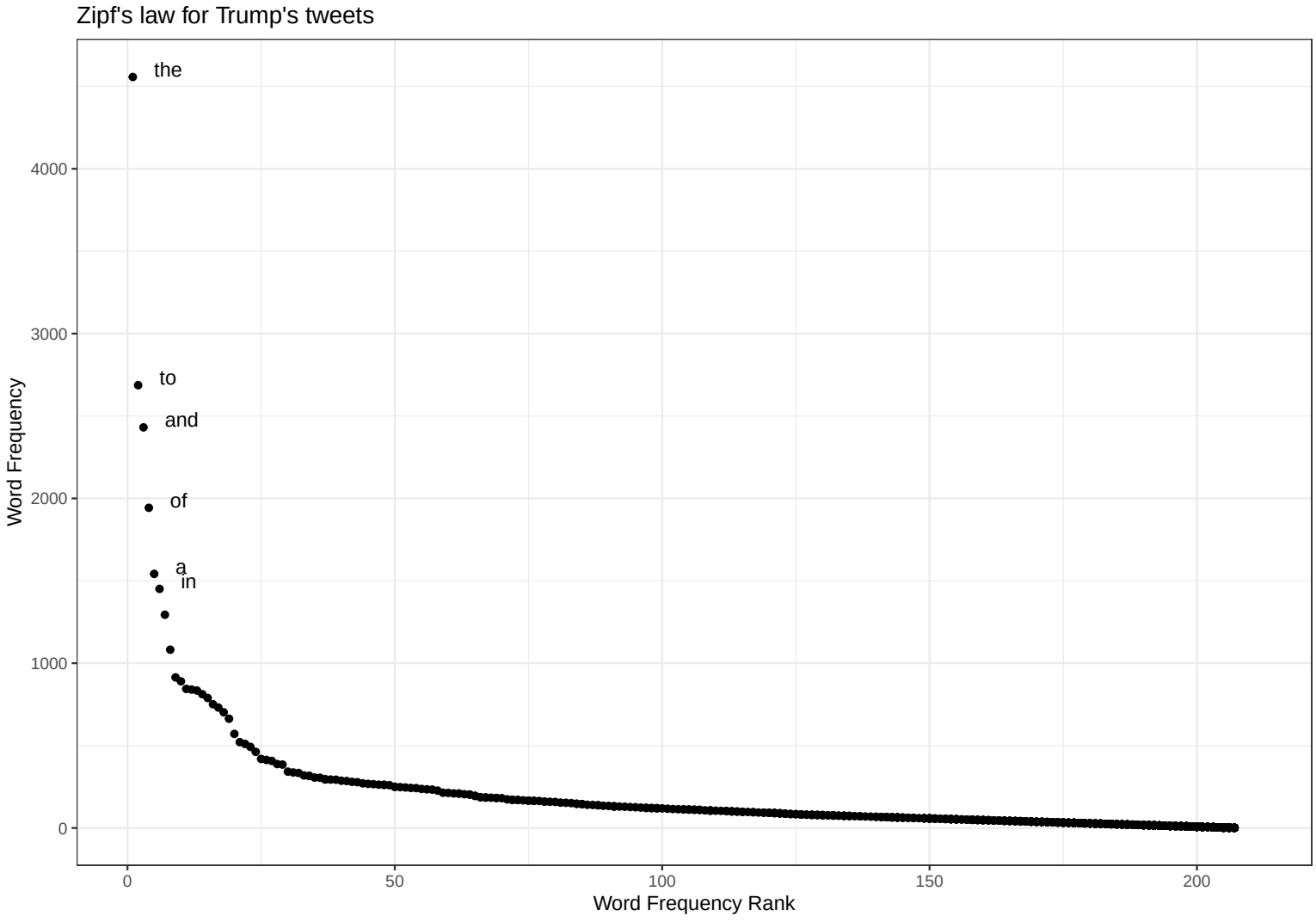
A fun fact: this relationship also holds for other measures, like population of global cities

Power-law/Zipf Distribution of Word Frequency

Where are the most informative words on this plot?

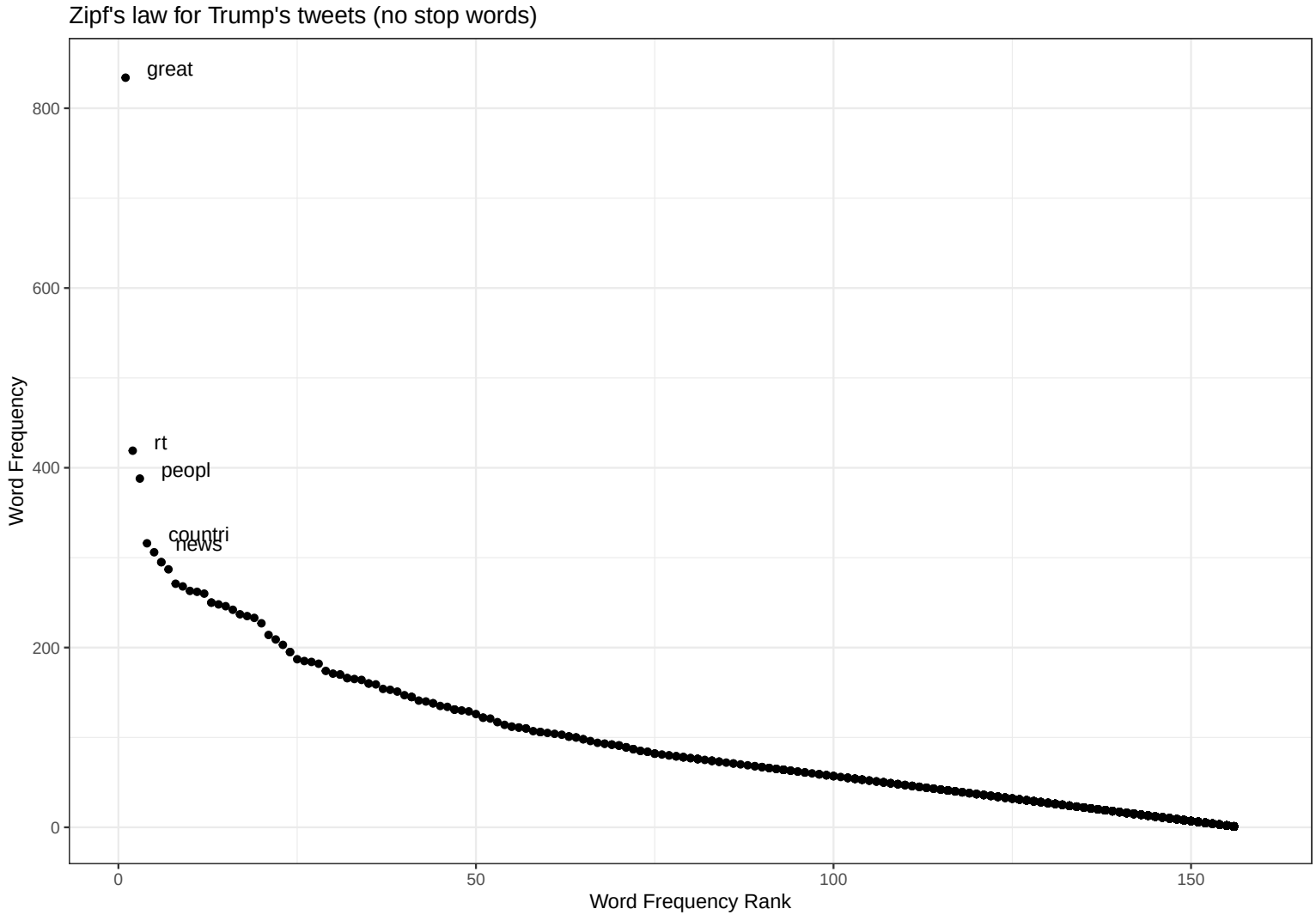


Zipf's law and Trump's tweets



Zipf's law and Trump's tweets

Removing stop words *helps*



Weighting strategies

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

- ➜ Each trimmed 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* counts in cells of a DFM using **term frequency inverse document frequency (tf-idf) weighting**

- More weight to word counts with: high term frequency in document AND low document frequency in the whole corpus
- Weights tend to "filter out" common terms

Many variations for how to calculate, but basic idea is that the count in each cell ij of the DFM is replaced by:

$$\text{tfidf}_{ij} = \text{tf}_{ij} \times \text{idf}_j$$

The "simplest" version uses:

$$\text{tf}_{ij} = W_{ij} \qquad \text{idf}_j = \frac{N}{\text{df}_j}$$

Weighting DFM of Trump tweets

Replace cells of DFM with calculated tf-idf

Original DFM:

datetime	american	#happynewyear	mani	bless	...	asham
						...
2017-01-01 05:00:10	1	1	1	1	...	0
2017-01-01 05:39:13	0	1	0	0	...	0
2017-01-01 05:43:23	0	0	0	1	...	0
...						
2017-01-31 00:45:50	0	0	0	0	...	0
2017-01-31 11:21:52	0	0	0	0	...	0
2017-01-31 11:27:02	0	0	0	0	...	1

[212 rows x 993 columns]

Weighting DFM of Trump tweets

Replace cells of DFM with calculated tf-idf

datetime	american	#happynewyear	mani	...	asham
					...
2017-01-01 05:00:10	1.423246	2.025306	1.284943	...	0.000000
2017-01-01 05:39:13	0.000000	2.025306	0.000000	...	0.000000
2017-01-01 05:43:23	0.000000	0.000000	0.000000	...	0.000000
...					
2017-01-31 00:45:50	0.000000	0.000000	0.000000	...	0.000000
2017-01-31 11:21:52	0.000000	0.000000	0.000000	...	0.000000
2017-01-31 11:27:02	0.000000	0.000000	0.000000	...	2.326336

[212 rows x 993 columns]

Weighting variations

We did simplest version if TF-IDF weighting, but this can biased toward long documents (weights them higher)

- **Sublinear term frequency scaling** reduces the impact of very frequent terms within a document by rescaling tf_{ij}:

$$\text{tf}^{\text{s}}_{ij} = (1 + \log(W_{ij}))$$

- **Binary term frequency** completely removes the impact of repetition in a textby rescaling tf_{ij} (useful for very short texts):

$$\text{tf}^{\text{b}}_{ij} = \mathbf{1}(W_{ij} > 0)$$

- **Smoothed inverse document frequency** prevents division-by-zero (if you do not remove unused tokens!) and softens extremes by rescaling idf_{ij}:

$$\text{idf}^{\text{s}}_{ij} = \log\left(\frac{N+1}{\text{df}_j + 1}\right) + 1$$

There are other weighting schemes, e.g. Okapi BM25 and SMART