Information Retrieval & Data Mining [COMP0084]

Text processing and indexing

Vasileios Lampos

Computer Science, UCL

lampos.net

@lampos

Preliminaries — About me!

- Associate Professor at the Computer Science department (2021)
- Ph.D. in Computer Science from the University of Bristol (2012)
- Been @ UCL Computer Science for almost a decade
- Main research theme: *Machine learning and natural language processing methods for health-related tasks*
- Information about my research at my personal / academic website: lampos.net
- Publications: scholar.google.com/citations?user=eXDONDEAAAAJ
- Tweets about research and society at: twitter.com/lampos
- Originally from Greece, but in the UK for more than 15 years

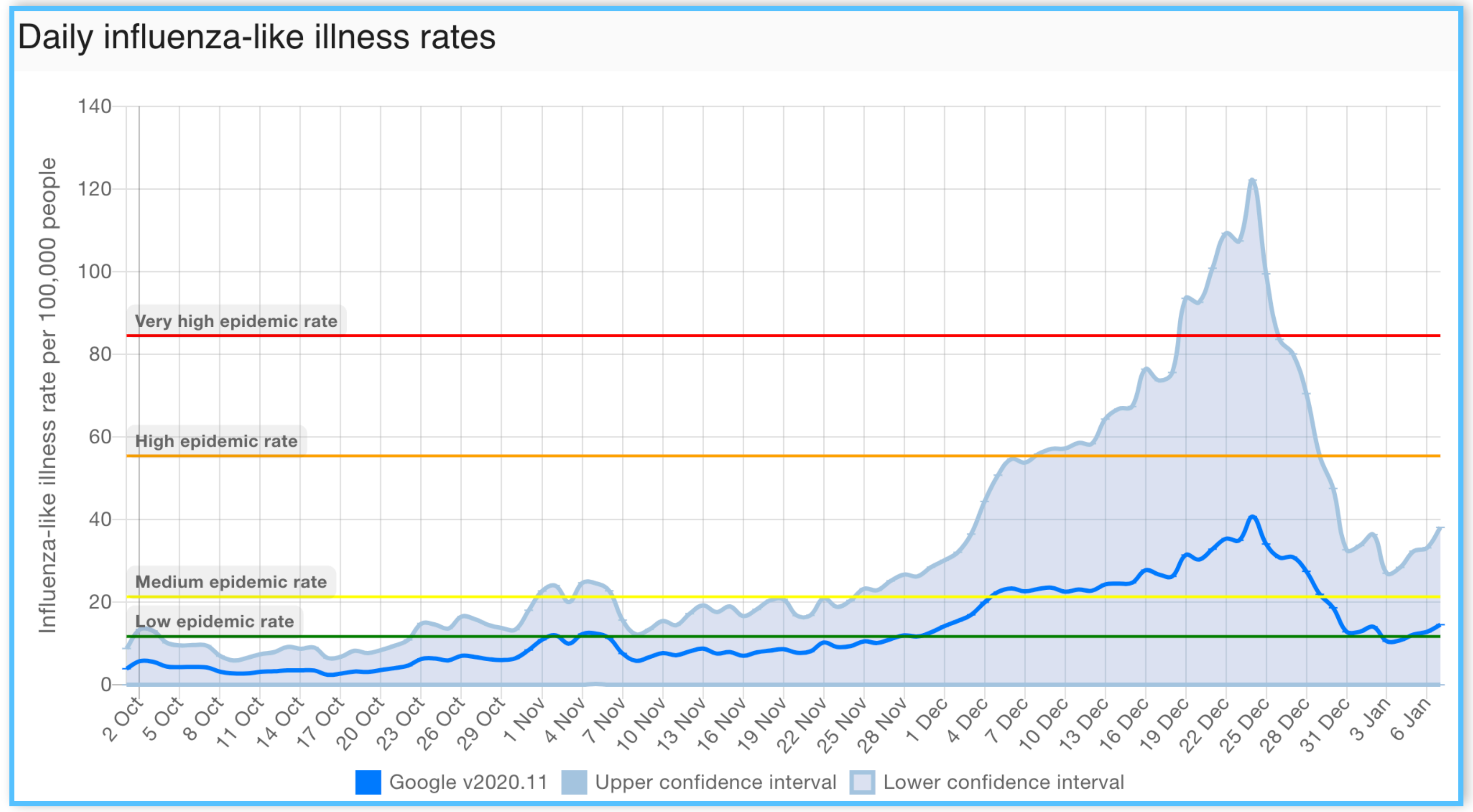
Preliminaries — PhD studentship(s) at UCL Computer Science

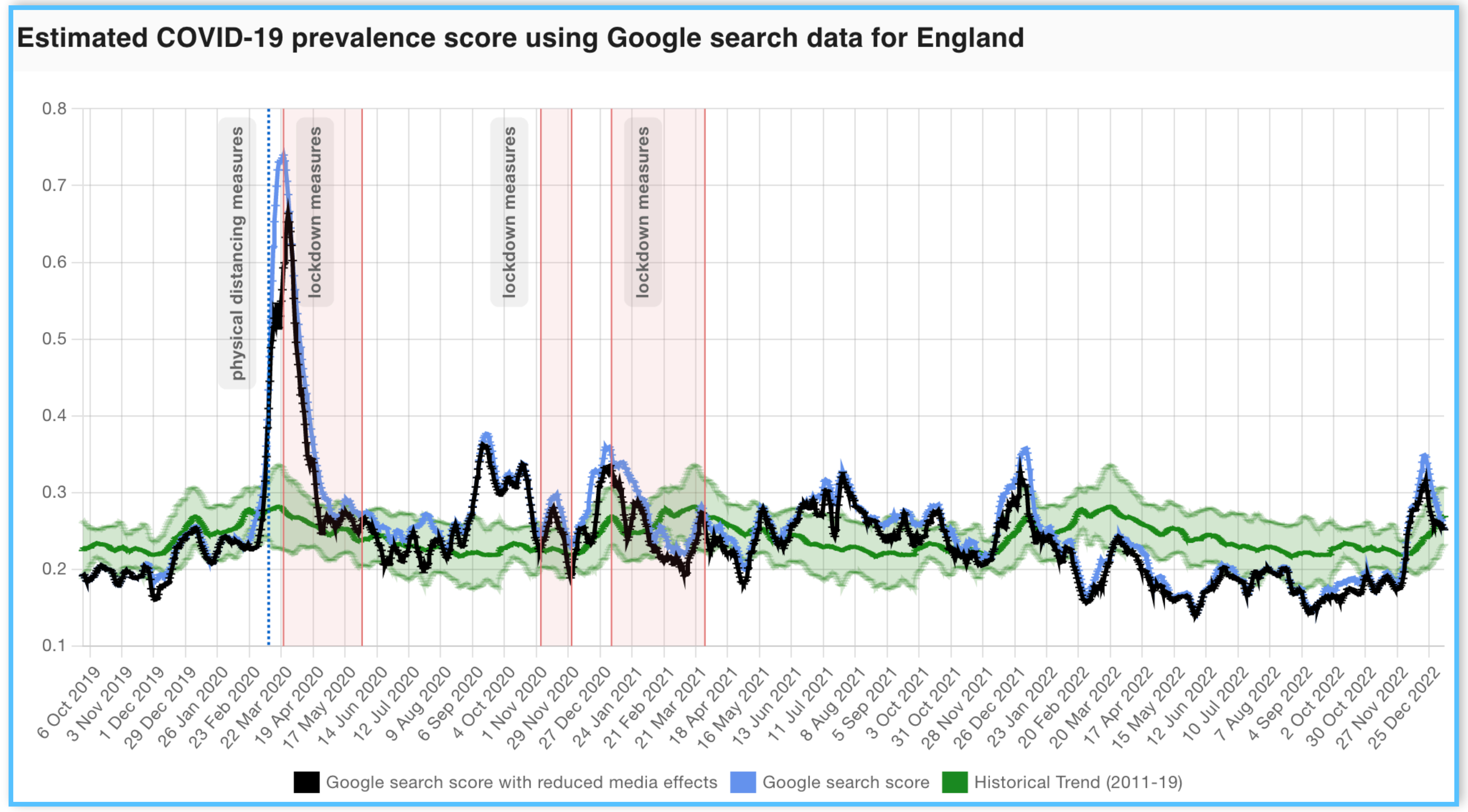
- **EPSRC DTP fully-funded studentship** (*home & international students*)
  - Apply ASAP, deadline in January 26, 2023
  - PhD project(s) description: ucl-epsrc-dtp.github.io/2023-24-project-catalogue/projects/2228bd1193.html
  - Apply @ ucl.ac.uk/epsrc-doctoral-training/prospective-students/apply-ucl-epsrc-dtp-studentship
- **Foundational AI CDT fully-funded studentship** (*home students only*)
  - Please get in touch via email to discuss your project proposal
  - Application deadlines in March and June, 2023
  - Apply @ ucl.ac.uk/foundational-ai-cdt/
- **General UCL Computer Science PhD studentship** (*home & international students*)
  - Please get in touch via email to discuss your project proposal
  - Funding not yet guaranteed; funding decision to be made in April 2023
  - Apply @ ucl.ac.uk/prospective-students/graduate/research-degrees/computer-science-4-year-programme-mphil-phd

Preliminaries — PhD studentship(s) at UCL Computer Science

- Research topics
  - machine (*deep*) learning, artificial intelligence
  - natural language processing
  - health-related tasks

- Example: Models for influenza and COVID-19 based on Google search activity
  - Flu detector, fludetector.cs.ucl.ac.uk
  - COVID-19, covid.cs.ucl.ac.uk
  - Both solutions are part of the national health surveillance in the UK, i.e. used by UKHSA





- More information @ lampos.net/join-us

Preliminaries — A few words about me and COMP0084 or IRDM

- Will do ~30% of COMP0084's lectures and will host 1 or 2 guest lectures
- Will run, support (*office hours*, *email*), and mark (*with potentially minor support from teaching assistants*) Coursework 1 which is **50%** of the final mark
- Will **not** be involved with Coursework 2 at all
- **Not** the module lead, hence when "*in crisis*" please email or cc Prof. Ingemar Cox

Preliminaries — About this lecture

- In this lecture:
  - basic text processing steps
  - inverted index
  - Zipf's law, Heaps' law (text statistics)
  - brief overview of Coursework 1
- **NB:** Topics discussed in this lecture are very relevant to Coursework 1
- **Some material can be found in** (*plus a great resource for further reading*): Chapters 1, 2, and section 5.1 of the [IIR] book: "*An Introduction to Information Retrieval*" by Manning, Raghavan, and Schütze (2009) — nlp.stanford.edu/IR-book/information-retrieval-book.html

Text processing — Applications

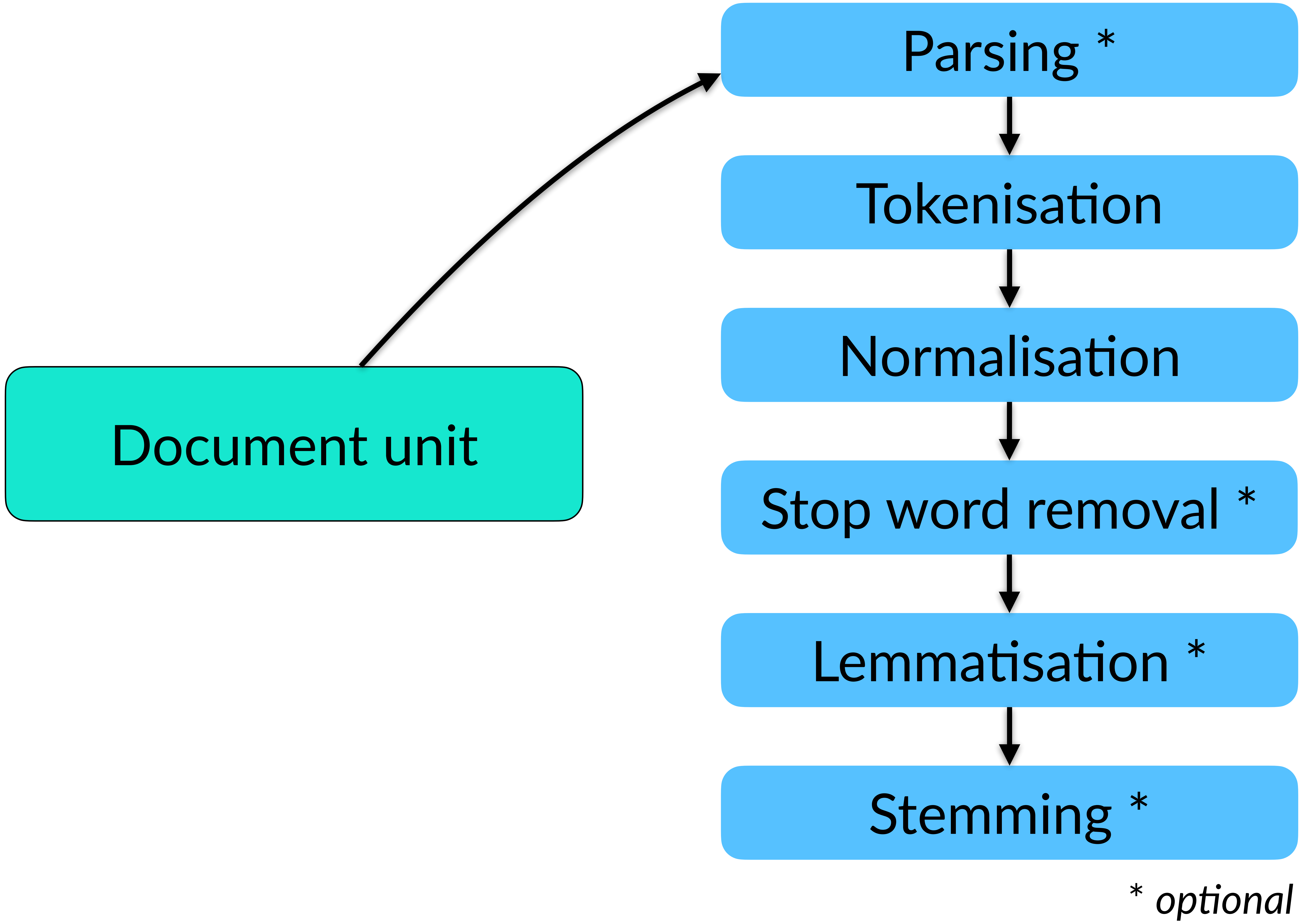
- Search engines
- Advertising
- Autocorrection, autocompletion, grammar check
- Machine translation
- Chatbots
- Email filters (spam, categorisation)
- Text-driven analytics (sentiment, opinions, health)
- News (topic models, summarisation)

Text processing — Basic steps

- **Document unit**
  book, book chapter, news article, paragraph, sentence, tweet, search query, fixed window of terms

- Depending on the task and the machine learning methods that are going to be deployed some processing steps are not applicable or may not be required

- The order in this diagram is not necessarily rigid (e.g. *parsing* can also take place after *tokenisation*)



Text processing — Parsing & tokenisation

- **Parsing**
  — if the file is not raw text, e.g. JSON, HTML
  — identify structural elements (e.g. titles, links, headings)

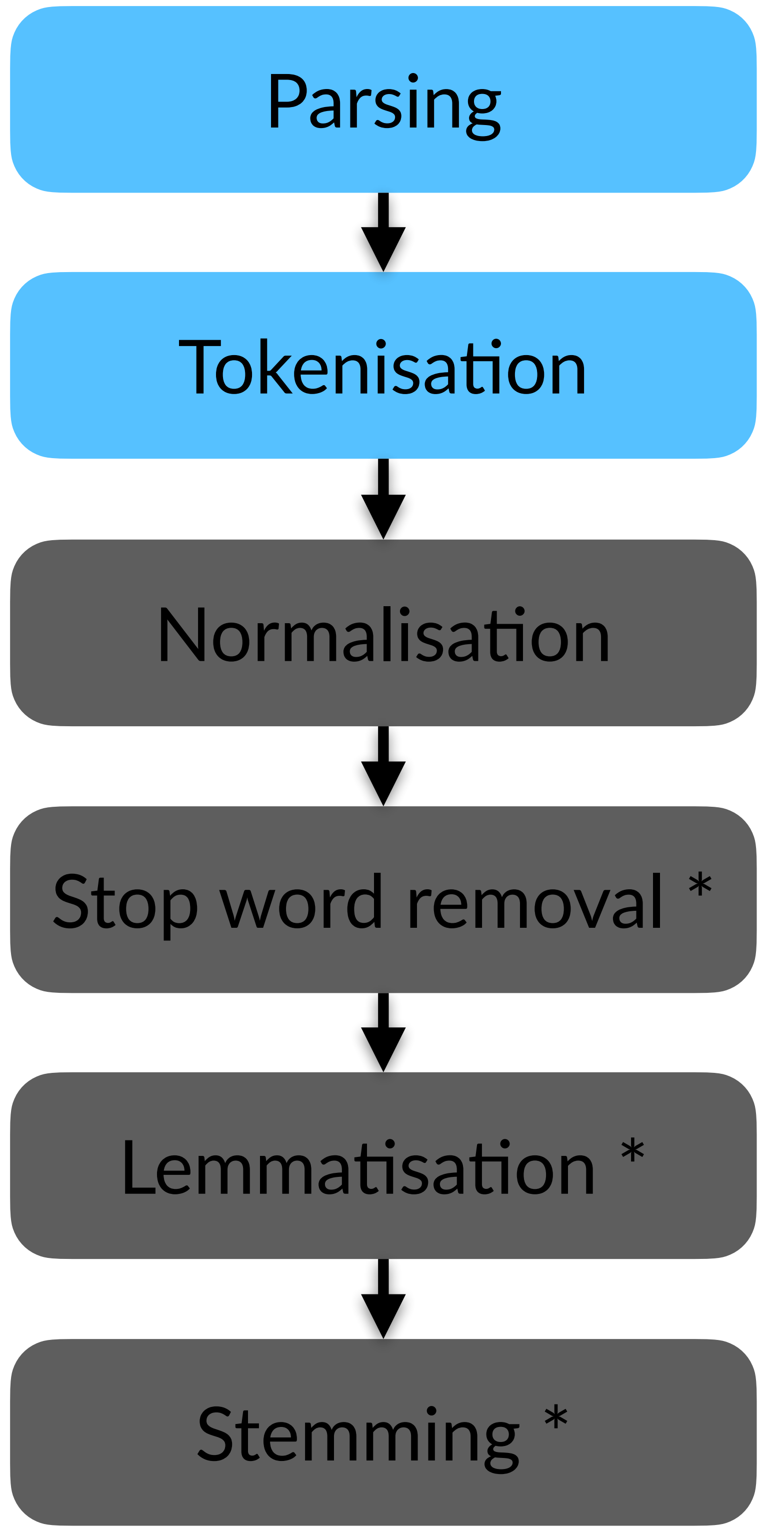
- **Tokenisation**
  the task of chopping up a document unit into pieces, called tokens

  Sentence: "They won't let you fly, but they might let you sing."

  Tokens: [They] [won't] [let] [you] [fly] [,] [but] [they] [might] [let] [you] [sing] [.]

- **Tokens** need to be turned to **terms**, i.e. *processed tokens* that will be maintained in our vocabulary index

- Not necessarily an easy task even for English and definitely harder in some other languages: punctuation, hyphens, capitalisation, numbers, separators, segmentation (*where does a word end*)



* *optional*

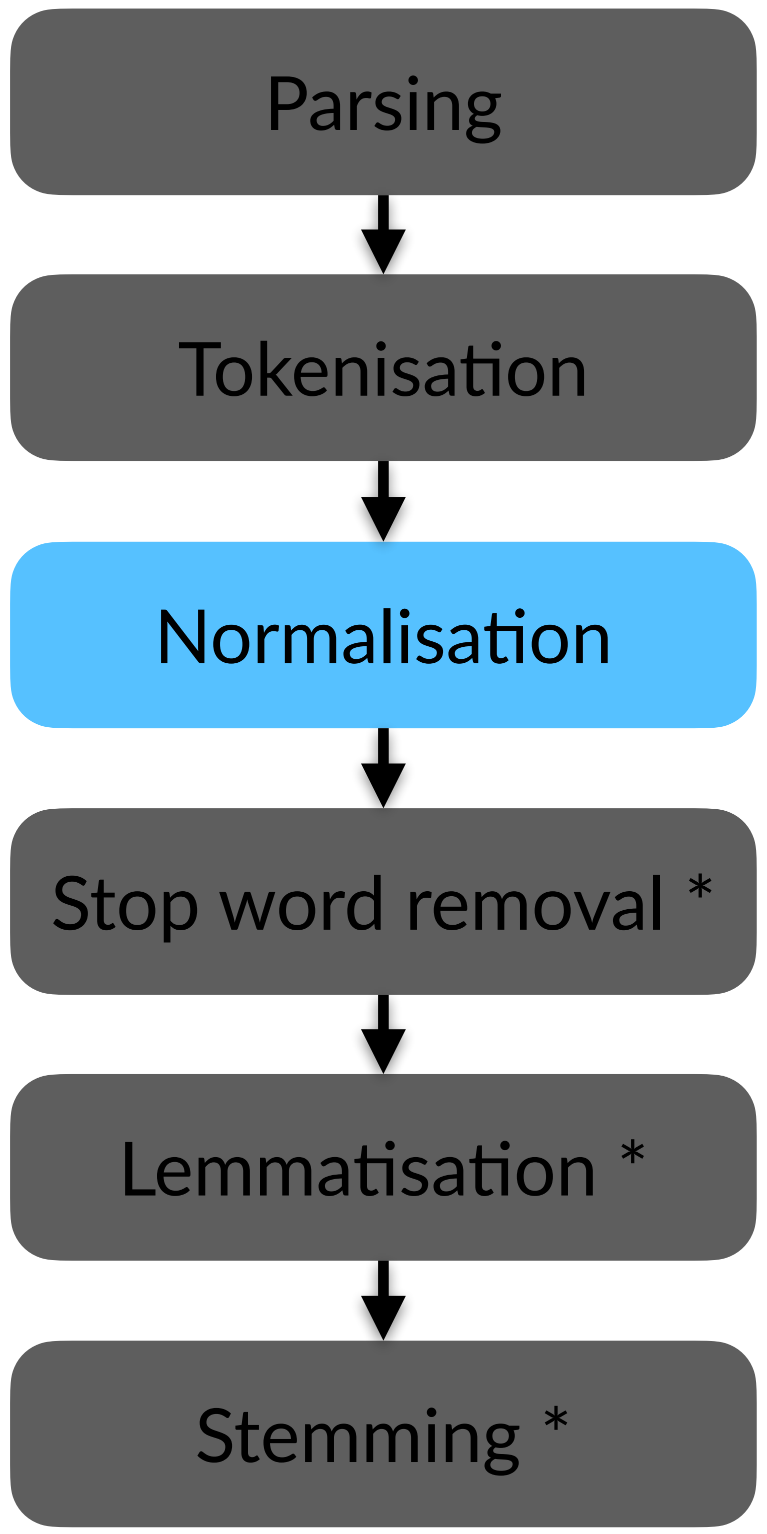
Text processing — Normalisation

- **Normalisation**
  the process of canonicalising tokens so that during indexing matches occur despite of superficial differences in the character sequences

- Try to group tokens with minor differences caused by the use of punctuation, diacritics, accents, hyphens
  “U.K.” ~ “UK” “naïve” ~ “naive”
  “don’t” ~ “dont” “co-exist” ~ “coexist”

- Maintain upper case, establish a conditional upper case, or lower case everything?
  “Windows” the operating system vs. “windows” in a house
  “Bill” the name vs. “bill” the check

- Hard task to get right — the type of each token needs to be known



* *optional*

- **Stop words**
  extremely common (*very frequent*) words that do not add to the meaning of a document unit, but exact definition depends on the set of decisions we make (*linked to the target task*) in order to identify stop words.

  Examples: **“the”, “an”, “to”, “so”, “then”**

  Benefits: **reduces number of features / dimensionality and helps derive models that can generalise better, saves storage / memory space (*perhaps not very relevant nowadays*)**

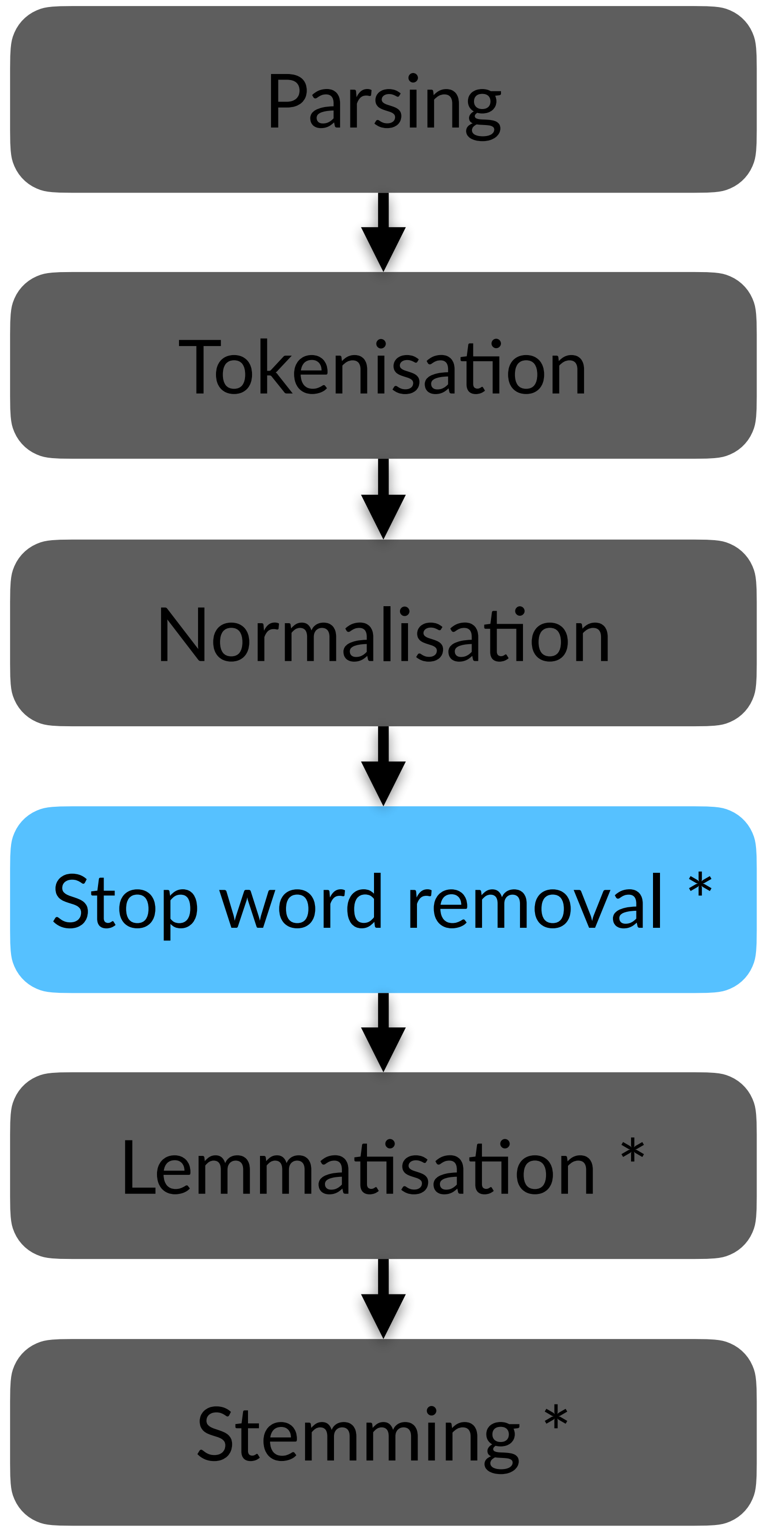
  Issues: **might remove some meaning from the text**
  e.g. “flights to London” — if we remove “to” as a stop word, then we don’t know whether this text snippet is about flights “to” or “from” London

Parsing
↓
Tokenisation
↓
Normalisation
↓
Stop word removal *
↓
Lemmatisation *
↓
Stemming *

* *optional*

Text processing — Stop word removal

- Could be determined using a predefined list and/or automatically, e.g. the most frequent terms in very large corpus
- Bag-of-words models (each term is considered in isolation) could benefit from stop word removal, but modern language models (e.g. BERT or GPT variants) might not as stop words can add to the semantic interpretation of text
- Should we remove stop words? Depends on the method used and the target task. Most of the times the downstream task accuracy can be measured, and we can actually see whether removing stop words helps or not and how much.



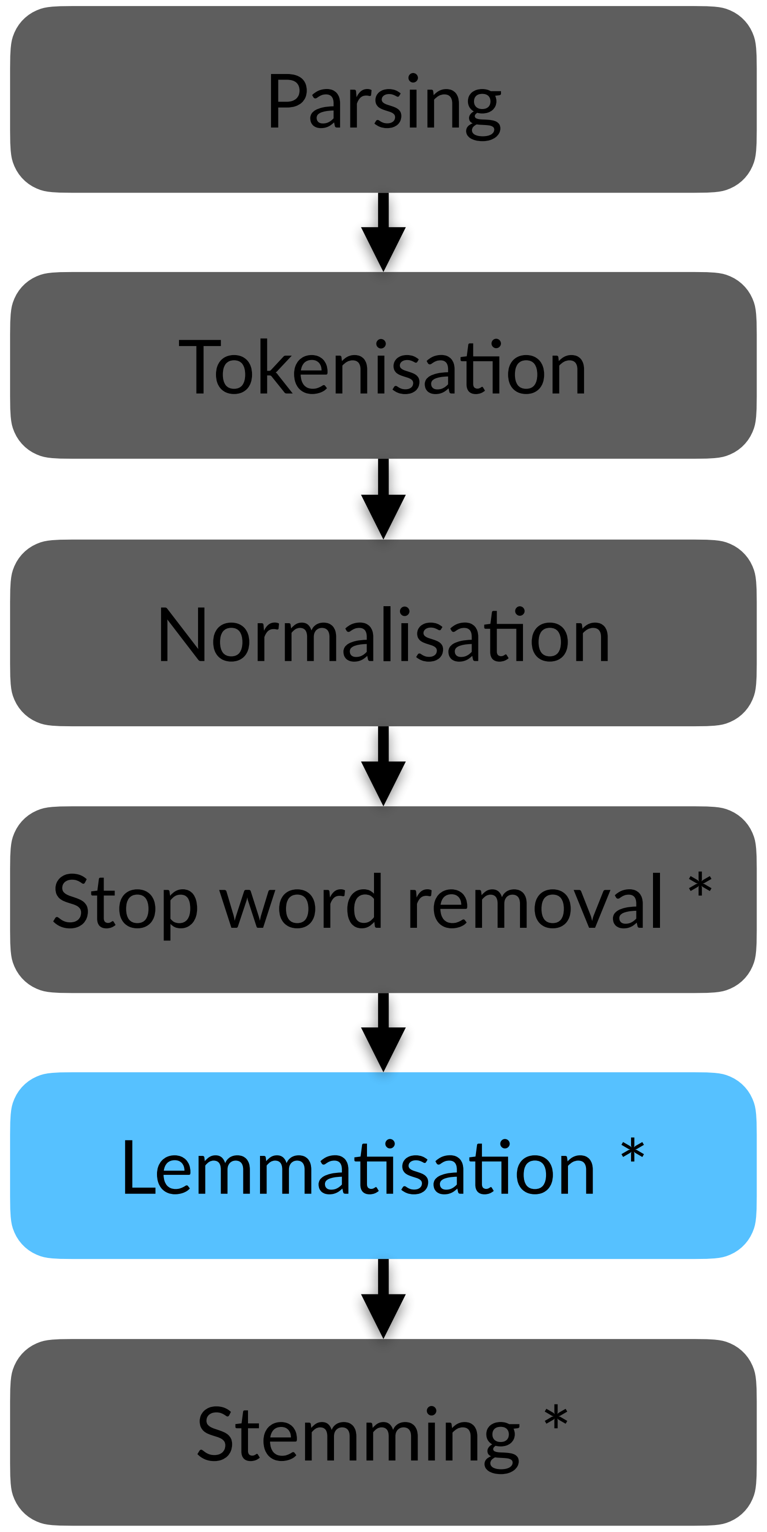
** optional*

Text processing — Lemmatisation

- **Lemmatisation**
  Returns the base (*dictionary*) form of a word, which is known as the *lemma*.

  “organises” or “organising” to “organise”
  “cars” to “car”
  “saw” to “see” (*if “saw” is a verb*)

- Does things “properly”, i.e. requires a complete vocabulary and morphological analysis (needs to know what part of speech is the target word for example), aiming to remove inflectional endings only



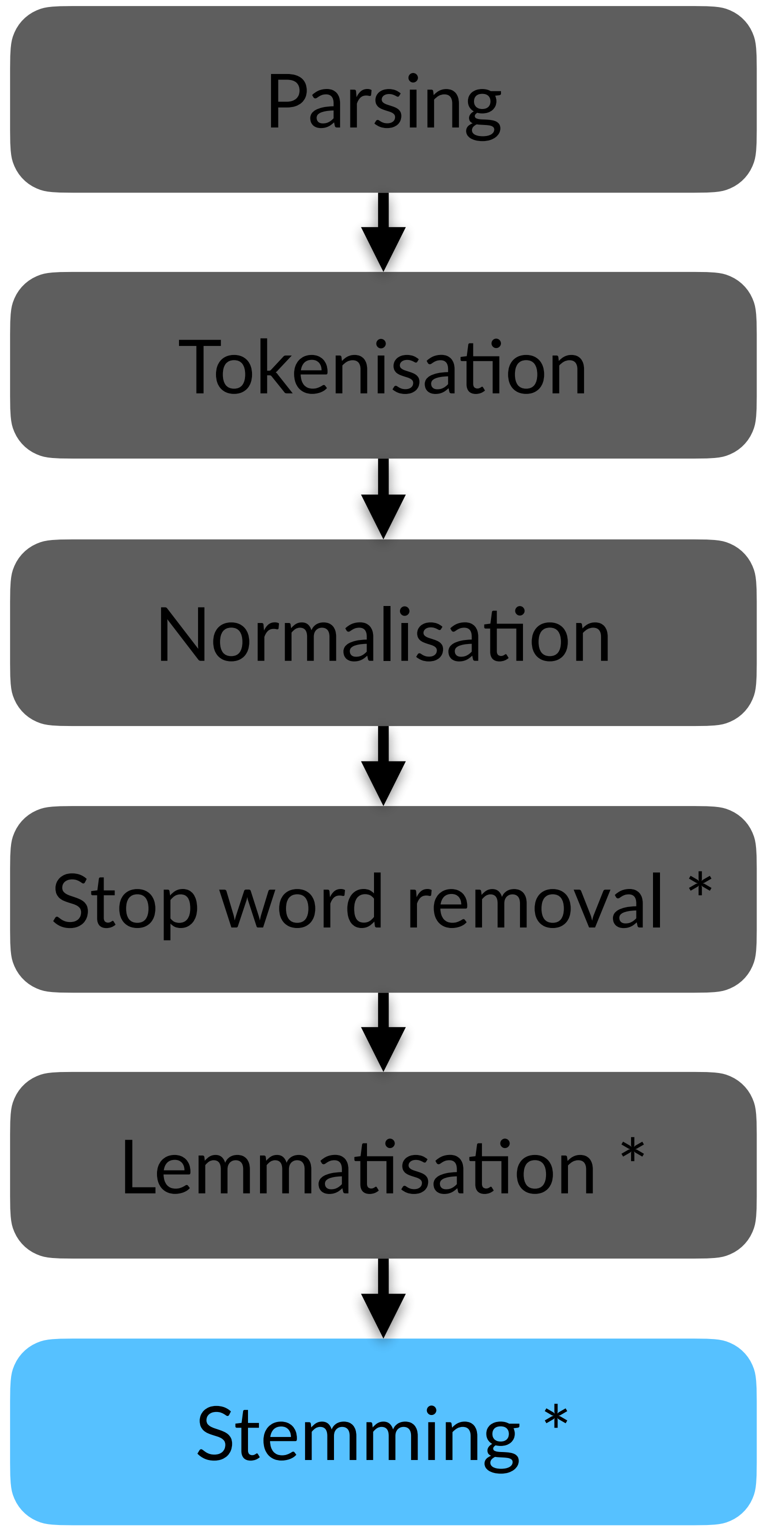
* *optional*

Text processing — Stemming

- **Stemming**
  Crude heuristic process that uses a stemmer (*stemming algorithm*) in an attempt to reduce inflected (or derived) words / tokens to their word stem (*root form*) — the stem, i.e. the output of a stemmer is very often not a vocabulary word

  "cars" to "car"
  "organises" or "organising" to "organis"
  "story" or "stories" to "stori"

- Most common algorithms: **Porter and Porter 2 (snowball) stemmer**
  tartarus.org/martin/PorterStemmer
  snowball.tartarus.org/algorithms/english/stemmer.html
  follows a set of complex rules (*easier to deploy than a lemmatiser*)
  removes the most common morphological and inflexional endings from words / tokens



* *optional*

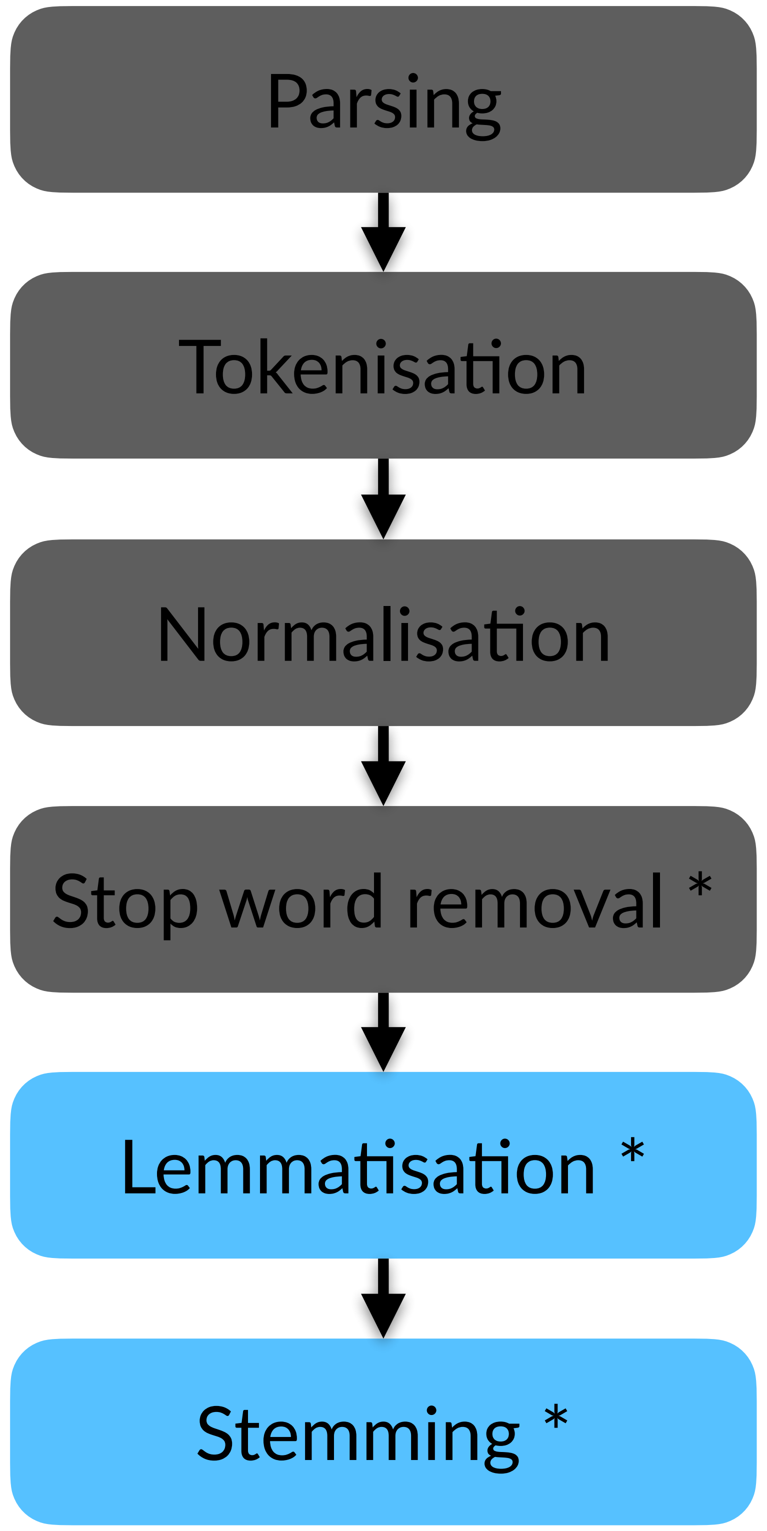
- Do lemmatisation and/or stemming significantly improve the accuracy in downstream tasks?
  — Not necessarily, at most very modest benefits for English
  — Stemming helps other languages though such as German

- Increase recall while harming precision, i.e. we will most definitely obtain all relevant documents, but together with them we will also obtain many irrelevant ones

  query: "operating" AND "system"

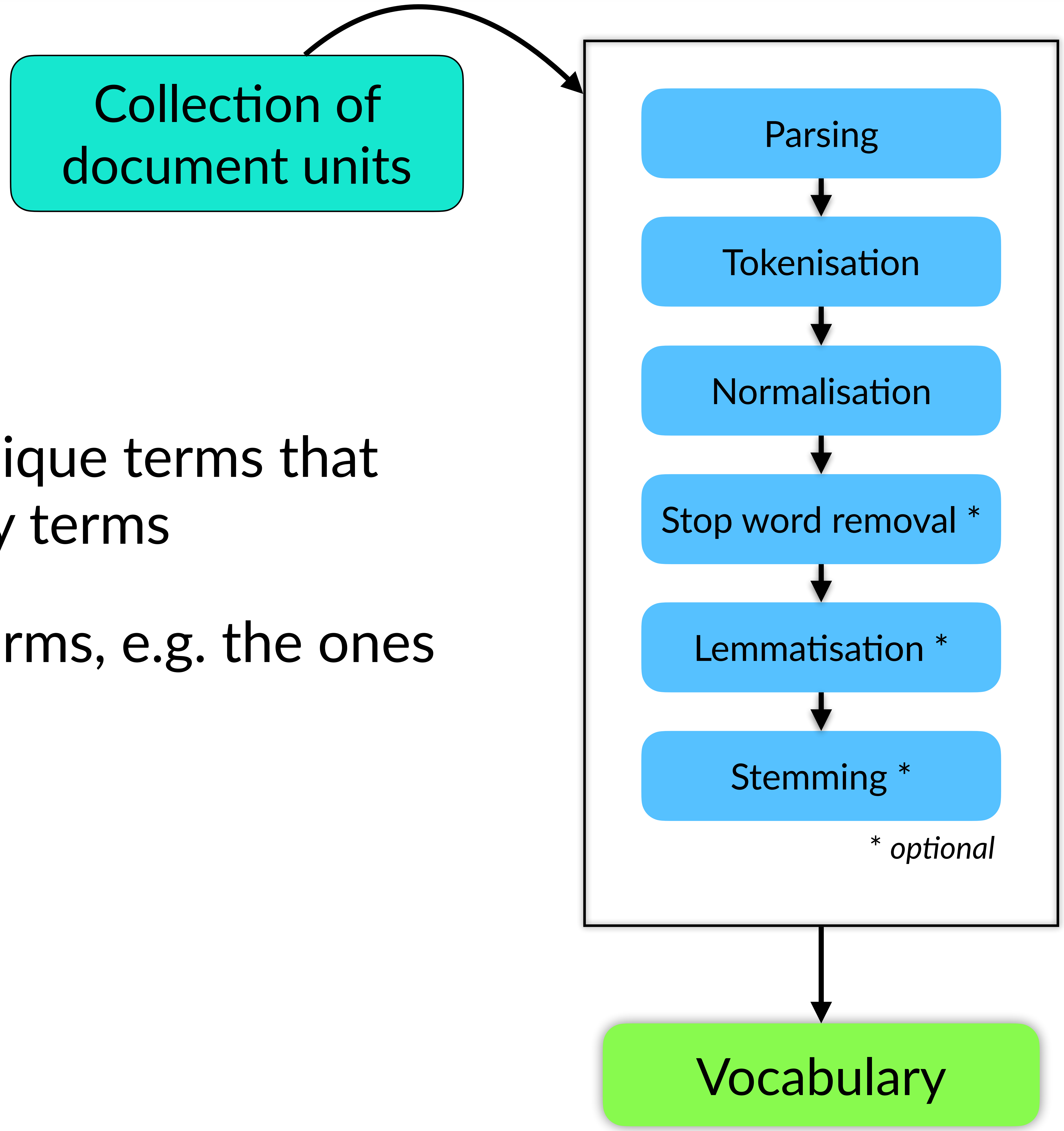
  if we assume Porter stemming is applied this will return documents that have the stems "oper" AND "system"

  however, this includes documents with the words "operational" AND "system" that are not a good match



* *optional*

Text processing — Vocabulary



- Finally, we obtain a **vocabulary**, an index of unique terms that either proper words or derived non-vocabulary terms
- Optionally, we can further remove very rare terms, e.g. the ones that appear only one time

Text processing — Inverted index

- **Index**
  A common way to think about an index is that a document in our collection will be represented as a set of indices of the terms in it. Hence:
  document —> terms —> index of terms in our vocabulary

- **Inverted index**
  works the other way around, hence the "inverted" connotation
  terms in our vocabulary —> list of documents in our collection they appear in

  Fair to say that "inverted" could be considered as redundant — we are not really inverting anything, and it is actually a common way of indexing.

  Why use it: improves search / retrieval speed
  NB: storage overhead, additional cost for adding / removing / updating documents

- Apart from a document index, an inverted index could also hold additional information
  — number of times (*count*) the term appears in a certain document
  — position in the document the term appears at

Text processing — Inverted index, an example

D_1: And you run and you run to catch up with the sun, but it is sinking, racing around to come up behind you again.

D_2: In my rear view mirror the sun is going down, sinking behind bridges in the road.

D_3: One day you find ten years have got behind you. No one told you when to run, you missed the starting gun.

Text processing — Inverted index, an example

D_1: And you run and you run to catch up with the sun, but it is sinking, racing around to come up behind you again.

D_2: In my rear view mirror the sun is going down, sinking behind bridges in the road.

D_3: One day you find ten years have got behind you. No one told you when to run, you missed the starting gun.

```
again:    1          have:     3          starting: 3
and:      1          in:       2          sun:      1,2
around:   1          is:       1,2        ten:      3
behind:   1,2,3      it:       1          the:      1,2,3
bridges:  2          mirror:   2          to:       1,3
but:      1          missed:   3          told:     3
catch:    1          my:       2          up:       1
come:     1          no:       3          view:     2
day:      3          one:      3          when:     3
down:     2          racing:   1          with:     1
find:     3          rear:     2          years:    3
going:    2          road:     2          you:      1,3
got:      3          run:      1,3
gun:      3          sinking:  1,2
```

Text processing — Inverted index, an example

D_1: And you run and you run to catch up with the sun, but it is sinking, racing around to come up behind you again.

D_2: In my rear view mirror the sun is going down, sinking behind bridges in the road.

D_3: One day you find ten years have got behind you. No one told you when to run, you missed the starting gun.

```
again:    1          have:     3          starting: 3
and:      1          in:       2          sun:      1,2
around:   1          is:       1,2        ten:      3
behind:   1,2,3      it:       1          the:      1,2,3
bridges:  2          mirror:   2          to:       1,3
but:      1          missed:   3          told:     3
catch:    1          my:       2          up:       1
come:     1          no:       3          view:     2
day:      3          one:      3          when:     3
down:     2          racing:   1          with:     1
find:     3          rear:     2          years:    3
going:    2          road:     2          you:      1,3
got:      3          run:      1,3
gun:      3          sinking:  1,2
```

Text processing — Inverted index, an example

D_1: And you run and you run to catch up with the sun, but it is sinking, racing around to come up behind you again.

D_2: In my rear view mirror the sun is going down, sinking behind bridges in the road.

D_3: One day you find ten years have got behind you. No one told you when to run, you missed the starting gun.

```
again:    1          have:     3          starting: 3
and:      1          in:       2          sun:      1,2
around:   1          is:       1,2        ten:      3
behind:   1,2,3      it:       1          the:      1,2,3
bridges:  2          mirror:   2          to:       1,3
but:      1          missed:   3          told:     3
catch:    1          my:       2          up:       1
come:     1          no:       3          view:     2
day:      3          one:      3          when:     3
down:     2          racing:   1          with:     1
find:     3          rear:     2          years:    3
going:    2          road:     2          you:      1,3
got:      3          run:      1,3
gun:      3          sinking:  1,2
```

Text processing — Inverted index, an example (*term's count*)

D_1: And you run and you run to catch up with the sun, but it is sinking, racing around to come up behind you again.

D_2: In my rear view mirror the sun is going down, sinking behind bridges in the road.

D_3: One day you find ten years have got behind you. No one told you when to run, you missed the starting gun.

```
again:    1:1
and:      1:2
around:   1:1
behind:   1:1,2:1,3:1
bridges:  2:1
but:      1:1
catch:    1:1
come:     1:1
day:      3:1
down:     2:1
find:     3:1
going:    2:1
got:      3:1
gun:      3:1
have:     3:1
in:       2:2
is:       1:1,2:1
it:       1:1
mirror:   2:1
missed:   3:1
my:       2:1
no:       3:1
one:      3:2
racing:   1:1
rear:     2:1
road:     2:1
run:      1:2,3:1
sinking:  1:1,2:1
starting: 3:1
sun:      1:1,2:1
ten:      3:1
the:      1:1,2:2,3:1
to:       1:2,3:1
told:     3:1
up:       1:2
view:     2:1
when:     3:1
with:     1:1
years:    3:1
you:      1:3,3:4
```

Text processing — Inverted index, an example (*term's count*)

query: `sun AND is AND going AND up`

```
{1:1, 2:1} + {1:1, 2:1} + {2:1} + {1:2} => D1:4, D2:3, D3:0
```

Hence the response to this query (ranked list of documents) by using a very naive retrieval approach would be D1, then D2, then D3.

```
again:    1:1
and:      1:2
around:   1:1
behind:   1:1,2:1,3:1
bridges:  2:1
but:      1:1
catch:    1:1
come:     1:1
day:      3:1
down:     2:1
find:     3:1
going:    2:1
got:      3:1
gun:      3:1
have:     3:1
in:       2:2
is:       1:1,2:1
it:       1:1
mirror:   2:1
missed:   3:1
my:       2:1
no:       3:1
one:      3:2
racing:   1:1
rear:     2:1
road:     2:1
run:      1:2,3:1
sinking:  1:1,2:1
starting: 3:1
sun:      1:1,2:1
ten:      3:1
the:      1:1,2:2,3:1
to:       1:2,3:1
told:     3:1
up:       1:2
view:     2:1
when:     3:1
with:     1:1
years:    3:1
you:      1:3,3:4
```

Text processing — Inverted index, an example (*term's position*)

D_1: And you run and you run to catch up with the sun, but it is sinking, racing around to come up behind you again.

D_2: In my rear view mirror the sun is going down, sinking behind bridges in the road.

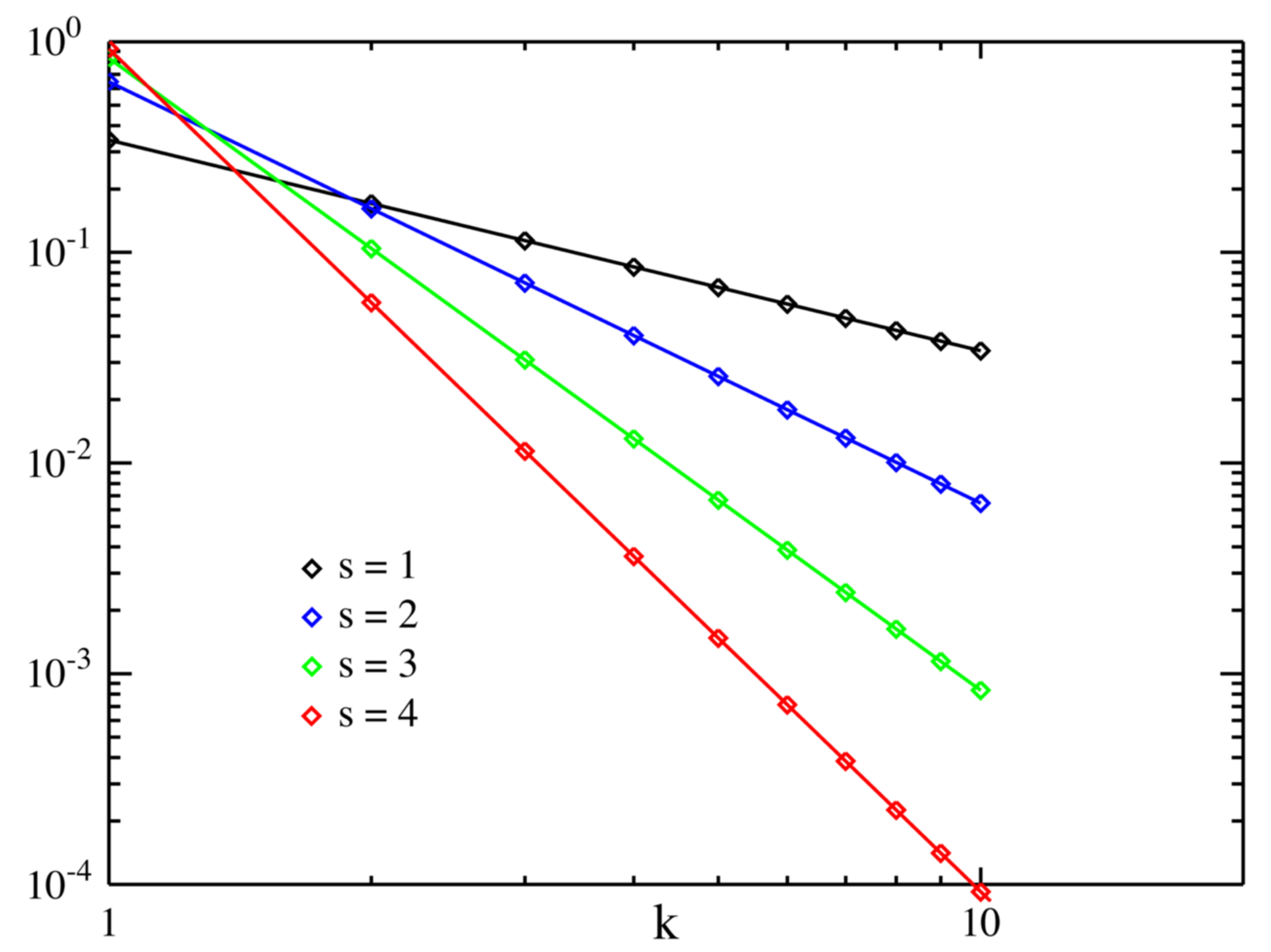
D_3: One day you find ten years have got behind you. No one told you when to run, you missed the starting gun.

```
again:    1:24               in:        2:1, 2:14          sun:    1:12, 2:7
and:      1:1, 1:4           is:        1:15, 2:8          ten:    3:5
around:   1:18               it:        1:14               the:    1:11, 2:6, 2:15
behind:   1:22, 2:12,3:9     mirror:    2:5                        3:20
bridges:  2:13               missed:    3:19               to:     1:7, 1:19, 3:16
but:      1:13               my:        2:2                told:   3:13
catch:    1:8                no:        3:11               up:     1:9, 1:21
come:     1:20               one:       3:1, 3:12          view:   2:4
day:      3:2                racing:    1:17               when:   3:15
down:     2:10               rear:      2:3                with:   1:10
find:     3:4                road:      2:16               years:  3:6
going:    2:9                run:       1:3, 1:6,          you:    1:2, 1:5, 1:23,
got:      3:8                           3:17                       3:3, 3:10, 3:14,
gun:      3:22               sinking:   1:16, 2:11                 3:18
have:     3:7                starting:  3:21
```

Text statistics — Zipfian distribution

$$f(k; s, N) = \frac{k^{-s}}{\sum_{i=1}^{N} i^{-s}}$$

- Power law, named after linguist G. K. Zipf
- [*top*] The (*normalised*) frequency (f) of a variable is inversely related to the variable's frequency rank (k) in a set of N variables, controlled by parameter $s \geq 0$.
- [*right*] The log-log plot of the probability mass function (PMF) defined for (*discrete*) values of k for $s = \{1,2,3,4\}$, and $N = 10$.



Source: Wikipedia (en.wikipedia.org/wiki/Zipf%27s_law)

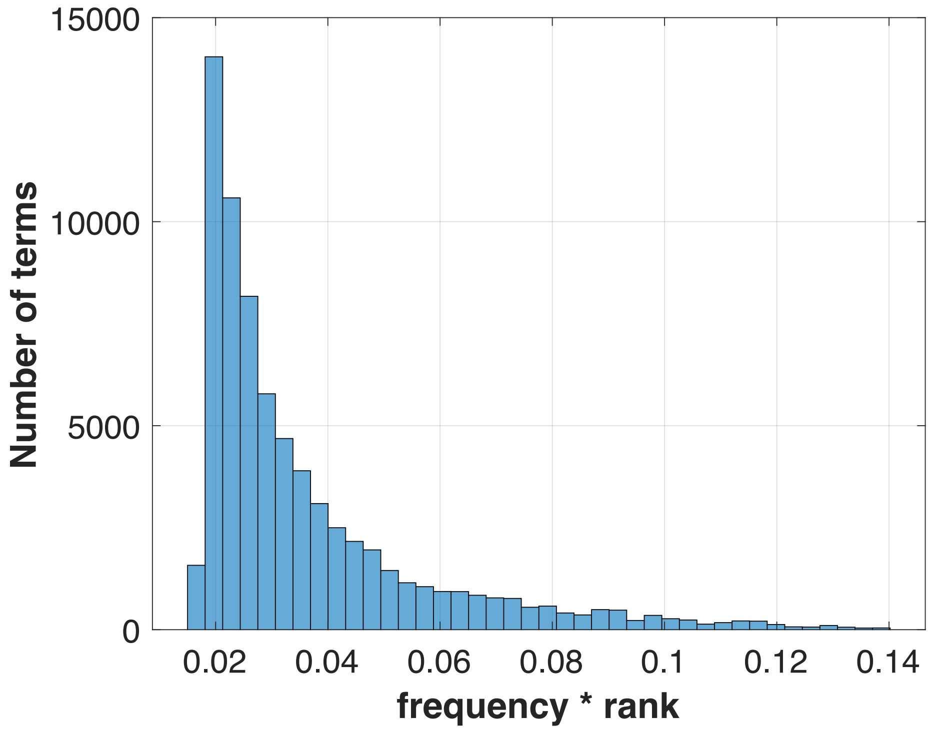
$$\text{Zipfian distribution: } f(k; s, N) = \frac{k^{-s}}{\sum_{i=1}^{N} i^{-s}}$$

$$\textbf{Zipf's law} \text{ sets } s = 1\text{, hence: } f(k; N) = \frac{1}{k \underbrace{\sum_{i=1}^{N} i^{-1}}_{H_N}}$$

- a few words occur very often, and many words hardly ever occur
- Zipf's law characterises the frequency distribution of terms in a (*large*) collection of documents (*corpus*)
- Specifically, it suggests that the rank of a term times its frequency $(k * f)$ is constant

Text statistics — Zipf's law, an example

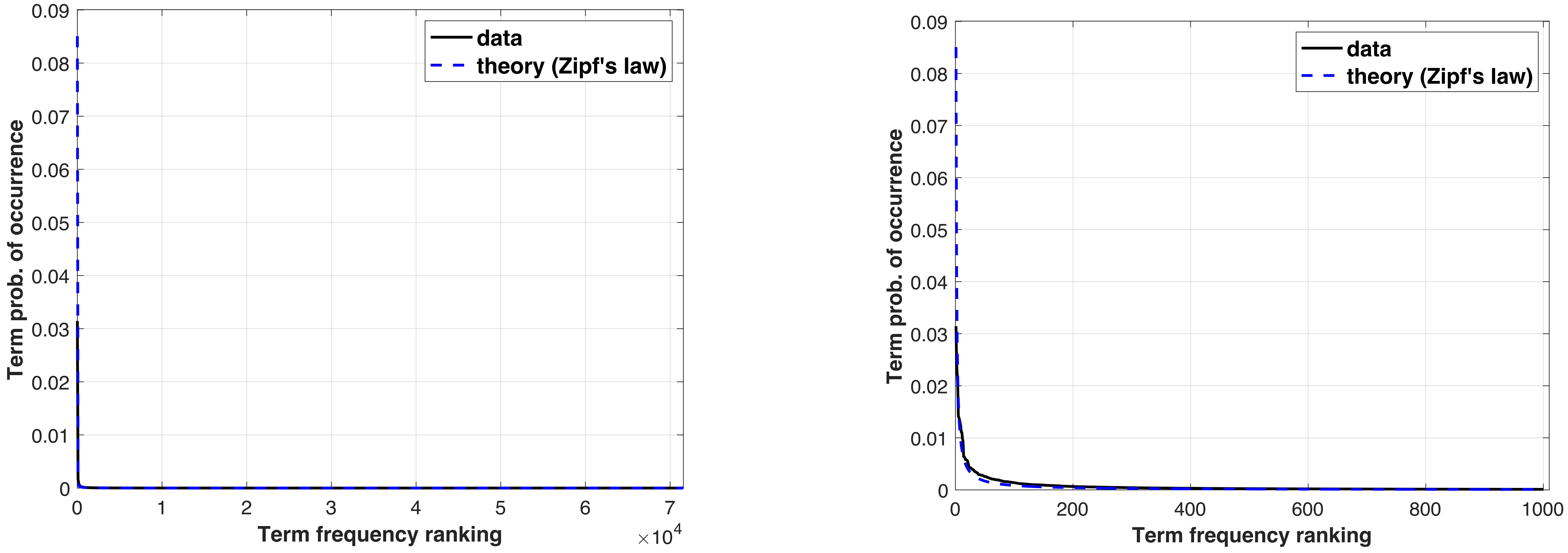
- Based on the Twitter data used in a paper of ours aclanthology.org/E14-1043.pdf
- ~50 million tweets
- 71,555 terms in the vocabulary
- Top-40 terms based on their normalised frequency
- μ(rank*frequency) = 0.036
  σ(rank*frequency) = 0.021



word	rank	frequency	rank*frequency
the	1	0.03145	0.03145
to	2	0.02441	0.04882
a	3	0.02224	0.06672
i	4	0.01976	0.07903
you	5	0.01418	0.07091
and	6	0.01384	0.08306
in	7	0.01347	0.09427
of	8	0.01285	0.10281
for	9	0.01228	0.11055
is	10	0.01108	0.11076
on	11	0.01103	0.12133
it	12	0.00985	0.11826
my	13	0.00933	0.12131
at	14	0.00638	0.08930
that	15	0.00633	0.09498
with	16	0.00621	0.09933
be	17	0.00584	0.09923
this	18	0.00581	0.10457
me	19	0.00574	0.10903
have	20	0.00567	0.11332

word	rank	frequency	rank*frequency
just	21	0.00548	0.11510
so	22	0.00493	0.10850
not	23	0.00446	0.10259
are	24	0.00432	0.10358
your	25	0.00426	0.10652
out	26	0.00407	0.10980
was	27	0.00402	0.11256
but	28	0.00398	0.11531
all	29	0.00386	0.11569
up	30	0.00385	0.11924
good	31	0.00358	0.11460
get	32	0.00357	0.11779
like	33	0.00349	0.11859
from	34	0.00341	0.11924
what	35	0.00332	0.11945
now	36	0.00329	0.12182
do	37	0.00318	0.12086
today	38	0.00297	0.11594
if	39	0.00296	0.11846
new	40	0.00290	0.11875

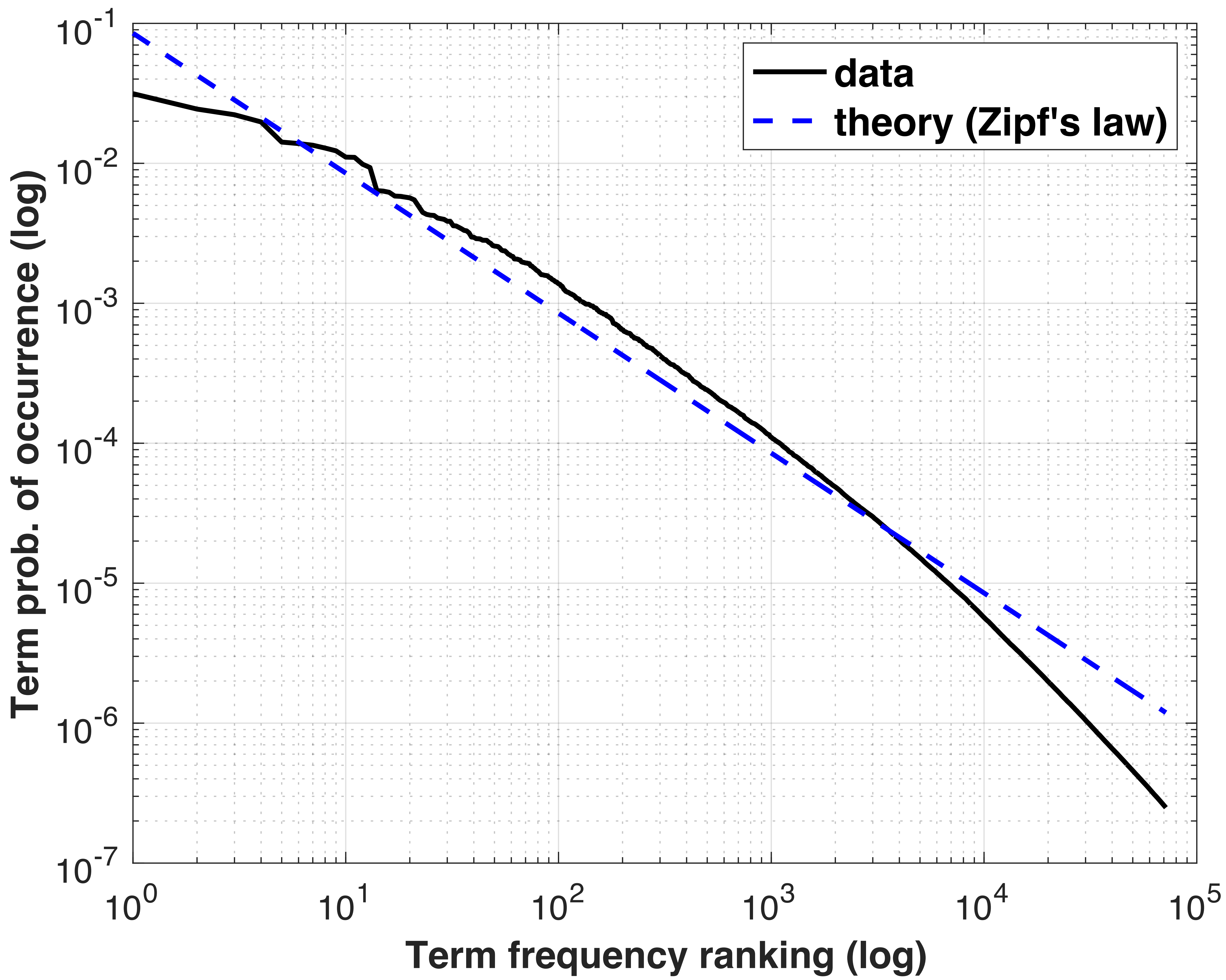
Text statistics — Zipf's law, an example



- probability of occurrence (*normalised frequency*) of a term vs. the term's ranking
- all 71,555 terms (*left*), top-1000 most frequent terms (*right*)
- practice seems to be following theory, but from these plots it is quite unclear

Text statistics — Zipf's law, an example

- Log-log plot provides a much better visual confirmation
- Zipf's law proposes that this relationship is "constant" (straight line in the log space)
- Practice follows theory quite well, but not entirely
- *What will happen to this plot if we remove stop words from our vocabulary?*



Text statistics — Zipf's law, an example

- Zipf's law suggests that rank $*$ frequency $= C$
- What is the proportion of terms with a certain frequency $\geq f \in [0,1]$?
- A term that has a frequency f has an estimated rank $k_f = C/f$ and hence the proportion of terms with frequency higher or equal to f is k_f/N where N is the size of our vocabulary
- Similarly, the proportion of terms with a frequency $a \leq x \leq b$ is given by $\left(k_a - k_b + 1\right)/N$
- If we set $a = 10^{-5}$ and $b = 10^{-3}$ then Zipf's law indicates that our corpus should have 11.9% of terms within that range (*when empirically we have 9.3%*)

word	rank	frequency	rank*frequency
the	1	0.03145	0.03145
to	2	0.02441	0.04882
a	3	0.02224	0.06672
i	4	0.01976	0.07903
you	5	0.01418	0.07091
and	6	0.01384	0.08306
in	7	0.01347	0.09427
of	8	0.01285	0.10281
for	9	0.01228	0.11055
is	10	0.01108	0.11076
on	11	0.01103	0.12133
it	12	0.00985	0.11826
my	13	0.00933	0.12131
at	14	0.00638	0.08930
that	15	0.00633	0.09498
with	16	0.00621	0.09933
be	17	0.00584	0.09923
this	18	0.00581	0.10457
me	19	0.00574	0.10903
have	20	0.00567	0.11332

Text statistics — Heaps' law

$$M = kT^{\beta} \quad \text{or} \quad \log(M) = \log(k) + \beta \log(T)$$

M is the size of the vocabulary and T is the number of tokens

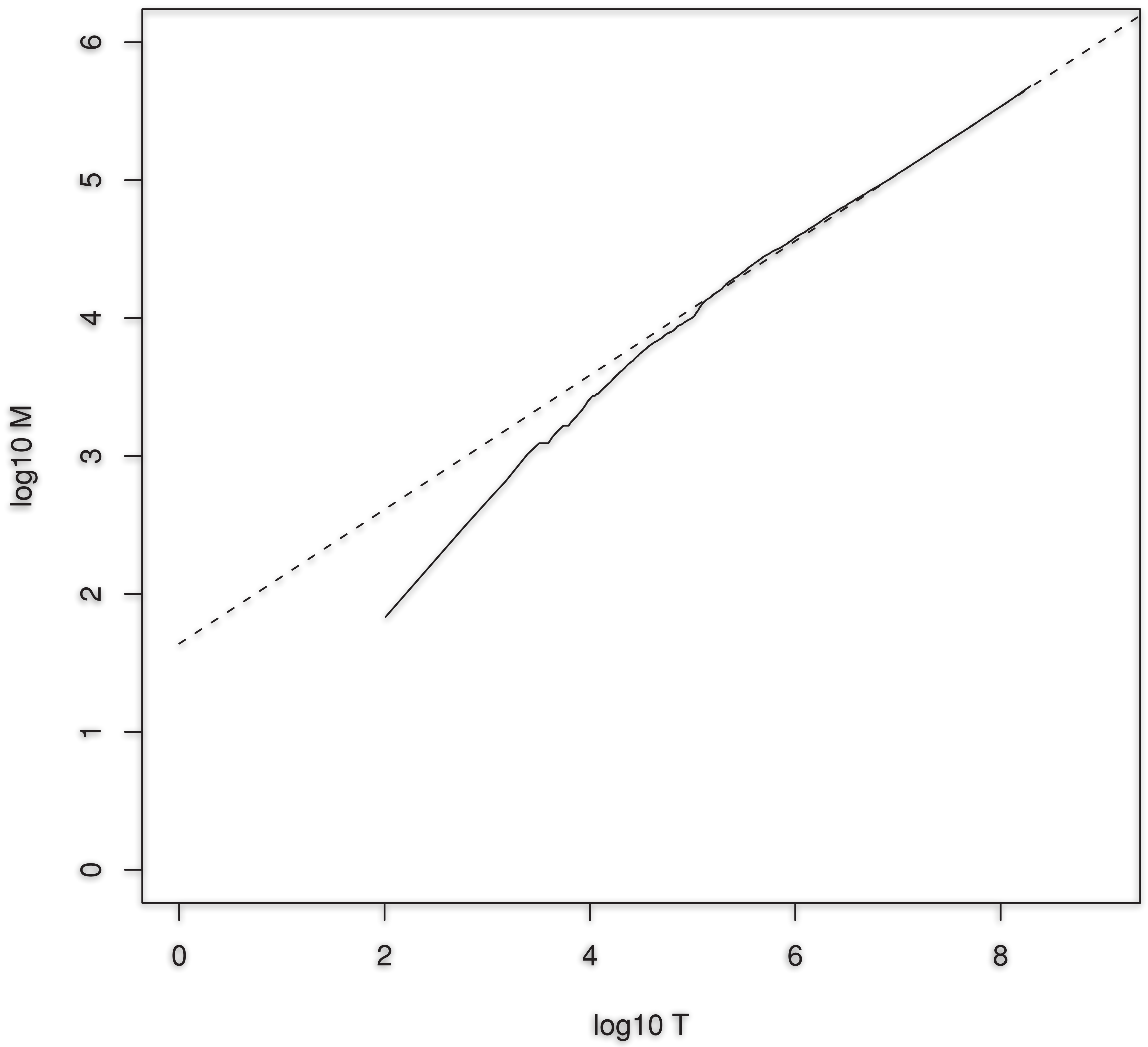
common parameter values: $k \in [10,100]$, $\beta \in [0.4,0.6]$

- Heaps' law captures how the size of the vocabulary (*unique terms*) grows with the size of the corpus (*number of tokens*)
  — no upper bound because of typos, novel terms (e.g. social media hashtags)
  — however, new terms occur less frequently as the vocabulary grows
  — still, the vocabulary size will become very large for very large corpora

- Heaps' law can be derived from Zipf's law by assuming documents are generated by randomly sampling words from a Zipfian distribution

Text statistics — Heaps' law, an example (IIR, Chapter 5)

- Corpus: 800K news articles from the Reuters RCV1 data set
  jmlr.csail.mit.edu/papers/volume5/lewis04a/
- Best least squares fit
  $\log_{10} M = 0.49 \times \log_{10} T + 1.64 \Rightarrow$
  $M \approx 44\, T^{0.49}$
- Hence $k = 44$ and $\beta = 0.49$
- For the first 1,000,020 tokens Heaps' law predicts a vocabulary size of 38,323 terms — the actual number is 38,365 (*very close!*)



Source: Fig. 5.1 of IIR (2009 edition)

Text statistics — Heaps' law, another example

- Vocabulary of size 100,000 — term frequencies follow Zipf's law
- We first draw 25K terms (T) from the Zipfian distribution of 100,000 terms (recall, we set $s = 1$). Because we are sampling, the terms we draw most likely are not going to be unique. We see how many unique terms exist in this draw (M). This pair $\{T, M\}$ of variables is our first sample, i.e. number of drawn terms (tokens) and the number of unique terms, respectively. Then we repeat by increasing the number of terms we draw by 25K (=50K), and continue doing so until we reach 1.5 million tokens (60 samples obtained).
- Best least squares fit in these 60 samples
  $\ln(M) = 0.5107 \times \ln(T) + 4.252 \Rightarrow M \approx 70.248\, T^{0.5107}$
  Hence $k = 70.248$ and $\beta = 0.5107$
- If we assume an exponential relationship between M and T, then this is captured well by Heaps' law.
- **Optional exercise:** *Can you replicate this experiment? What can go wrong with this derivation?*

About Coursework 1

- **50%** of the final mark
- **Data set:** 200 search queries, for each one ≤1,000 passages that were returned
- **Tasks:** text processing and analysis, inverted index implementation, (re-)rank the passages for each query based on basic retrieval and query likelihood language models
- Give **extra attention** the following
  - marking will be partially automated — *please follow instructions to the letter*!
  - Python (recommended), Java (*permitted*), no notebook submissions, each task asks for specific output — filename/type, a submission will consist of 10 or 11 files exactly
  - your answers will have a level of stochasticity
  - do not use external functions that can solve end-to-end the tasks of building an inverted index, retrieval and language models
  - only use unigram (1-gram) text representations
  - use the ACL LaTeX template for your report

About Coursework 1 — Questions, support, basic code of conduct

- **Deadline:** March 3, 2023 at 4pm
- **Q & A** about Coursework 1 on February 1, time slot TBA
- **Office hours:** Tuesdays 9-10am starting from Jan. 24, MS Teams, no group calls
- **Send me an email any time** [v.lampos@ucl.ac.uk]
- Do **not** post anything about Coursework 1 on the course's forum or in any public medium (*to avoid unintentional "spoilers"*)
- Based on the discussions during the office hours and the emails, I might send announcements with clarifications to the entire class. **These will be posted on the forum, so please check your emails and the COMP0084 forum regularly.**
- Do **not** send me questions about Coursework 2 — this is run by Prof. Emine Yilmaz and a team of teaching assistants
- Marks are expected to be released by April 4, 2023. Please note that if there are many EC extensions, the mark release date might be delayed (by 1 to 3 weeks).

About Coursework 1 — Hints

- My not very optimal code that solves Coursework 1 runs on my 1st generation Macbook Pro M1 (8 CPU/GPU cores, 16GB RAM) in about 12 minutes
- Having said that, the inverted index implementation might need some extra care to avoid getting out-of-memory and some parallelisation to make it fast enough
- Please make sure you have access to the right CPU resources (*especially for Coursework 2*)
- I tried to provide very specific instructions for Coursework 1, but please be aware that further clarifications might be required along the way

Next lectures with me

Q & A about Coursework 1

- February 1, time slot TBD

Introduction to machine learning and data mining

- February 8 and 9 (3 hours)

Topic models and vector semantics (*word embeddings*)

- Lectures on March 1 and 2 (2 hours)
- March 1, 12pm to 1pm, guest lecture by Dr Adam Tsakalidis

Guest lecture by me on modelling COVID-19 using web search activity

- March 15, 11am to 12pm