# The Term Vocabulary

Chien Chin Chen

Department of Information Management
National Taiwan University

# Information Retrieval (1/2)

- ***Information Retrieval*** (IR):
  - Is finding **material of an unstructured nature** that satisfy an **information need** from within **large collections**.
    - Unstructured material: usually documents.
      - Structured material, e.g., database tuples.
    - Information need: user queries, a set of keywords.
    - Large collections: a collection of documents, usually stored on computers.
- The simplest form of document retrieval is for a computer to do **linear scan** through documents.
  - Read through all the text.
  - Note for each document whether it contains desired information.
  - Unix command `grep` can perform this process efficiently.

# Information Retrieval (2/2)

- But What if …
  - you are dealing with a large document collections?
  - Flexible query operations, such as or, not …
  - **Ranking!!**
- A better way than linear scanning is to **index** the documents in advance.
  - Recall the index part of a book where you can find the desired information easily.
  - Probably create a term-document incidence matrix.

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |

1 if the document Contains the word

A column: a vector of the document

# Major Steps in Index Construction

1. Collect the documents to be indexed.
2. ***Tokenize*** the text.
3. Do ***linguistic preprocessing*** of tokens.
4. ***Index*** the documents that each term occurs in.

- In this chapter, we will talk about:
  - Characteristics of documents.
  - Tokenization.
  - Token normalization.
  - Stemming and lemmatization.
  - Stop word dropping.
  - Statistical properties of terms in Information Retrieval.

# Document Delineation and Parsing (1/4)

- Digital documents are typically **bytes** in a file or on a web server.
- The first step of document parsing is to convert this **byte sequence** into a linear **sequence of characters**.
- To parse a document, we have to know the **properties** of the document:
  - The **encoding mechanism** of the text.
    - ASCII/Unicode/vendor-specific standards.
    - Then can decode the bytes to a character sequence.

# Document Delineation and Parsing (2/4)

- The documents may be in some **binary** representation.
  - doc, pdf, ps …
- Or the documents are **semi-structure** text.
  - XML, SGML, html, …
  - The textual part of the document may need to be extracted out of other material (markup).

- These document properties have to be determined, and then an appropriate decoder has to be used.
  - Especially for Commercial products, which usually need to support a broad range of document types and encoding.
  - Usually, licensed software libraries can handle these issues.

# Document Delineation and Parsing (3/4)

- For some languages, such as Arabic, where text takes on some two dimensional and mixed order characteristics.
  - استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.
  - ← → ← → ← start
  - But there is an underlying sequence of sounds being represented and hence an essentially linear structure remains.

# Document Delineation and Parsing (4/4)

- ***Document unit*** and ***indexing granularity***:
  - Many email messages can contain attached documents.
    - We should split them into separate documents.
  - A ppt put on the Web can contain more than one html file.
    - We should combine multiple files into a single document.
  - An units can be a set of documents, a document, a paragraph, or a sentence.
    - **Indexing granularity affects IR performance !!**
    - If units are too large, difficult to get relevant information.
      - E.g., index an entire book and the query terms appear in the first and last chapter.
    - If units are too small, users can be overwhelmed by too many returns.
  - For a good choice of granularity, the person who is deploying the IR system must have a good understanding of the document collection, the users, and usage patterns.

# Tokenization (1/5)

- ***Tokenization*** is the task of
  - Chopping a character sequence up into pieces, called **tokens**.
  - Perhaps at the same time throwing away certain characters, such as punctuation.
  - Example:
    - Input: *Friends, Romans, Countrymen, lend me your ears*.
    - Output: <*Friends*, *Romans*, *Countrymen*, *lend*, *me*, *your*, *ears*>
  - ***Token*** — ***type*** — ***term***
    - A **token** is an instance of a sequence of characters in some particular document, and is useful semantic unit for processing.
    - A **type** is the class of all tokens containing the same character sequence.
    - A **term** is a (normalized) type that is indexed in the IR system's dictionary.
      - Usually derived by various *normalization* processes.
      - Can be entirely distinct from the tokens (e.g., stemming).

# Tokenization (2/5)

- Token — type — term example:
  - *"to sleep perchance to dream"*
  - 5 tokens: *to*, *sleep*, *perchance*, *to*, and *dream*.
  - 4 types, since there are 2 instances of *to*.
  - 3 terms, if *to* is omitted from the index as a **stopword**.
- **The major question of the tokenization phase is what are the correct tokens to emit?**
- Simple method: just split on all **non-alphanumeric** characters.
  - *Chile's* → *Chile* and *s*.
  - *O'Neal* → *O* and *Neal*.
  - C++ → C.
  - *r13579@ntu.edu.tw* → *r13579*, *ntu*, *edu*, *tw*

# Tokenization (3/5)

- **Hyphens**:
  - In English, hyphens are used for various purposes, e.g., joining nouns as names, *Hewlett-Packard*.
  - Should we split it or regard it as one token?
  - Some useful heuristic rules:
    - Allowing short hyphenated prefixes on words.
      - E.g., *co-worker*.
    - But not longer hyphenated forms.
      - *the hold-him-back-and-drag-him-away manner*.
    - Some systems encourage users to enter hyphens wherever they may be possible, and the systems will generalize the query to cover all three of the one word, hyphenated, and two word forms.
      - E.g., over-eager → over-eager or over eager or overeager.
      - Depends on user training.

# Tokenization (4/5)

- Splitting on white space can also split what should be regarded as a single token.
  - Names: Los Angles.
  - Compounds are sometimes space separated: whitespace vs. white space.
  - Dates: Mar 11, 1983.
- Moreover, hyphens and non-separating whitespace can interact.
  - Air fares: San Francisco-Los Angles.
- You always do the exact same tokenization of document and query words to guarantee the a sequence of characters in a text will always match the same sequence typed in a query.

# Tokenization (5/5)

- Each language presents some new issues:
  - German writes compound nouns without space:
    - Computerlinguistik → computational linguistics.
    - Require a *compound-splitter* module to see if a word can be subdivided into multiple words that appear in a vocabulary.
  - East Asian Language (e.g., Chinese, Japanese, Korean, and Tai) are written without any spaces between words.
    - Require a *word segmentation* preprocessing.
      - Having a large vocabulary and taking the **longest** vocabulary match with some heuristics for unknown words.
        - 我要去總統府 → 總統?? or 總統府??
      - Machine learning sequence models, such as HMMs.
    - Or do indexing with short subsequence of characters (*n*-grams), regardless of whether particular sequences cross word boundaries or not.
      - Is appealing because:
        - Single Chinese character usually has semantic content.
        - Most words are short (2 characters).
        - Word boundaries are not clear and probably no exact locations.

# Normalization (1/5)

- Having broken up documents into tokens, the easy case of information retrieval is if query tokens just match document tokens.
- However, there are many cases when two character sequences are not quite the same but you would like a match to occur.
  - *'on-line'* and '*online'*.
  - *'USA'* and '*U.S.A'.*
- (Token) ***normalization*** is a process of canonicalizing tokens so that matches occur despite **superficial differences** in the character sequences of tokens.
  - Need to "normalize" terms in indexed text as well as query terms into the same form.

# Normalization (2/5)

- The most standard way to normalize is to create **equivalence classes** using **rules**.
  - For example, *removing hyphens and periods*.
    - Tokens '*on-line'* and '*online'* belong to the same class named 'online'.
    - Then searches for one term will retrieve documents that contain either.
  - The classing process of this approach is done **implicitly**.
- An alternative is to maintain relations between unnormalized tokens and do **token expansion**.
  - For example, hand-constructed lists of synonyms.
    - '*car*' and '*automobile*' belong to the same class.
  - With (explicit) equivalence classes, we can:
    - Index unnormalized tokens and do query expansion with equivalence classes.
    - Or, perform the expansion during index construction.
      - When a document contains '*automobile*', we index it under '*car*' as well.

# Normalization (3/5)

- The **best amount** of equivalence classing or query expansion to do is a fairly open question.
  - Doing some definitely seems a good idea.
  - But doing a lot can easily have unexpected consequences.
    - For example, '*C.A.T.*' → '*cat*'.
- Some normalization strategies:
  - **Remove accents and diacritics**, such as '*naïve*' → '*naive*', because users are used to enter queries for words without diacritics.
  - **Case-folding** by **reducing all letters to lower case**.
    - '*Automobile*' at the beginning of a sentence to match with a query of '*automobile*'.
    - However, many proper nouns are distinguished only by case.
      - E.g., person names, Bush or bush.
    - A heuristic for English: to lowercase words at the beginning of a sentence; mid-sentence capitalized words are left as capitalized.
    - As users usually use lowercase regardless of the correct case of words, **lowercasing everything often remains the most practical solution**.

# Normalization (4/5)

- Other languages present distinctive issues in normalization.
- Properties of other languages:
  - Approximately 60% of web pages are in English.
  - But … less than one third of Internet users and less than 10% of the world's population primarily speak English.
  - So … the non-English portion are expected to grow over time.
    - E.g., only one third of blog posts are in English.

# Normalization (5/5)

- The French word for '*the*' has distinctive forms based on gender/number of the following noun …
  - *the* → *le*, *la*, *l'*, *les*.
- Japanese is even more difficult.
  - Japanese is mixed with multiple alphabets.
    - Chinese, hiragana(平假名), katakana(片假名).
  - Even a word may be written with multiple writing systems.
  - Retrieval systems thus require complex equivalence classing across the writing systems.
- Other issues:
  - Foreign name translation, such as *Beijing* and *Peking*.
  - A document including many different languages requires more than one tokenizer and language-specific normalization.

# Stemming and Lemmatization (1/5)

- For grammatical reasons, documents are going to use different forms of a word.
  - Such as *'organize'*, *'organizes'*, and *'organizing'*.
- It would be useful for a search for one of these words to return documents that contain another word in the set.
- ***Stemming*** and ***lemmatization*** are to reduce terms to their "**roots**" before indexing.
  - For instance: *car*, *cars*, *car's*, *cars'* → *car*.

# Stemming and Lemmatization (2/5)

- Differences between stemming and lemmatization:
  - **Stemming:** usually a **crude heuristic process** that chops off the **ends** of words.
    - E.g., '*automate*(*s*)*'*, *'automatic', 'automation'* all reduced to '*automat*'.
  - **Lemmatization:** doing things more properly with the use of a vocabulary and morphological analysis of words.
    - To return the base or dictionary form of a word, ***lemma***.
    - E.g., *'am'*, *'are'*, *'is'* → *'be'*.
    - *'saw'* would return either *'see'* or *'saw'* depending on whether the use of the token was as a verb or a noun.
- Stemming or lemmatization is often done by an additional plug-in component to the indexing process.
  - A number of commercial and open-source components.

# Stemming and Lemmatization (3/5)

- ***Porter's algorithm***, the most common algorithm for stemming English.
  - Have repeatedly been shown to be very effective.
- Consist of 5 phases of word reductions, applied sequentially.
  - Each phase consists of a set of **rules** and with **conventions** to select rules.
    - Examples of rules and results:
      - (rule) *'tional'* → *'tion'* and (example) *'national'* → *'nation'*
    - Convention example: *select the rule from each phase that applies to the longest suffix.*

# Stemming and Lemmatization (4/5)

- Many of the later rules use the ***measure*** of a word to see whether a stemmed word is long enough.
  - Measure loosely checks the number of syllables.
  - (***m* >1**) *'ement'* → *' '*
    - *'replacement'* → *'replac'*
    - *'cement'* → *'cement'*
- Other stemming algorithms:
  - Lovins stemmer, Paice stemmer.

# Stemming and Lemmatization (5/5)

- You can use a **lemmatizer** which does full morphological analysis to accurately identify the **lemma** for each word.
  - An NLP application.
- However … doing morphological analysis produces very modest benefits for retrieval.
- Stemming increases recall while harming precision sometimes.
  - *'operate'* and *'operating'* are all stemmed to *'oper'*.
  - Query: *'operating system'* would find sentence with *operate* and *system*.

# Stop Words (1/2)

- Some **common words** (**stop words**) appear to be of little value in helping select documents matching a user need.
  - Should be excluded from the vocabulary entirely.
  - Example: *a, an, and, are, as, …, was, were, with, …*
- The general strategy for determining stop words:
  - **Sort** the terms (in a document collection) by ***collection frequency***.
    - *Collection frequency*: the number of times each term appears in the collection.
  - Take the **most frequent terms** as a ***stop list***.
    - Often with the help of human experts.

# Stop Words (2/2)

- A lot of the time, not indexing stop words does little harm:
  - "*flights to London*" → "*fights London*"
  - Titles of books or songs generally consists of a lot of words on stop lists.
    - "*let it be*", "*to be or not to be*", "*As we may think*"…
- The trend in IR systems over times has been from standard use of quite large stop list (200-300 terms) to very small stop list (7-12 terms) to no stop list whatsoever.
  - Web search engines generally do not use stop lists.
  - Several works on how the statistics of language can be used to cope with common words in better ways.
    - TF-IDF term weighting leads to very common words have little impact on document rankings.

# Basic Indexing Process

Documents to be indexed.

Friends, Romans, countrymen.

Tokenizer

Token stream.

Friends | Romans | Countrymen

Linguistic modules

Modified tokens, or terms

friend | roman | countryman

Indexer

Inverted index.

***friend*** → 2 → 4 →

***roman*** → 1 → 2 →

***countryman*** → 13 → 16

# Statistical Properties of Terms in Information Retrieval (1/8)

- In many information retrieval tasks, knowing (or estimating) the number of terms *M* in a collection is necessary.
- The second edition of the Oxford English Dictionary (OED) defines more than 600,000 words.
  - Big enough?? … **No!!**
  - OED does not include most names of people, locations, products and scientific entities (like genes).
  - These names still need to be included in most of information retrieval systems, so users can search for them.
- Is there any convenient method to estimate *M* for a text collection?

# Statistical Properties of Terms in Information Retrieval (2/8)

- ***Heaps' Law*** – estimating the number of terms.
  - Heaps' law estimates vocabulary size as a function of collection size:

$$M = kT^b$$

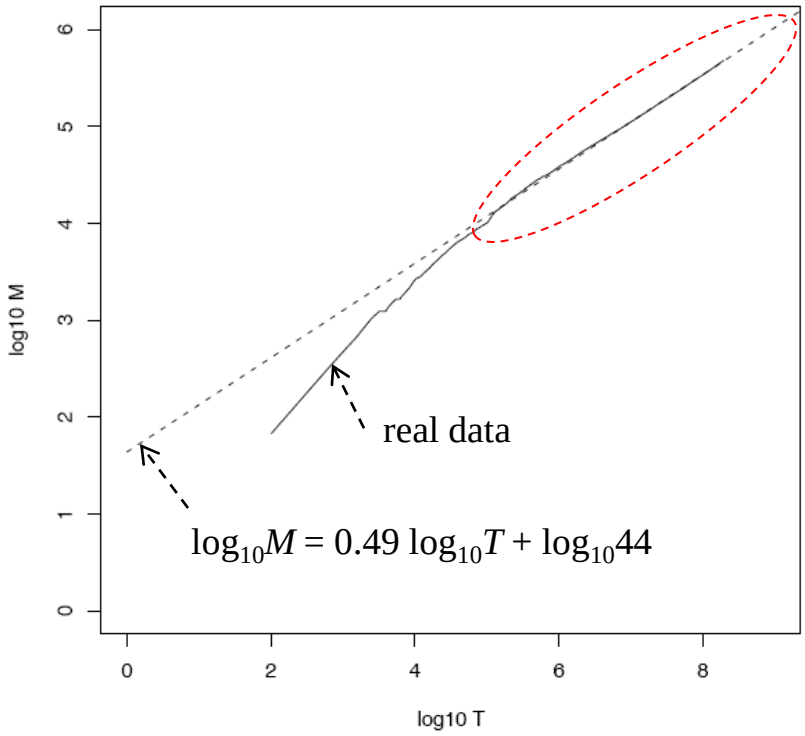
$T$: the number of tokens in the collection

  - Typical values for the parameters are:
    - $30 \leq k \leq 100$
    - $b \approx 0.5$.
  - The law indicates that the relationship between **collection size and vocabulary size is linear in log-log space**.

$$\log M = \log kT^b$$
$$= \log T^b + \log k$$
$$= b \log T + \log k$$

$\log k$: constant

# Statistical Properties of Terms in Information Retrieval (3/8)




- For Reuters-RCV1. Heaps' law has an excellent fit for $T > 10^5$, for $b = 0.49$ and $k = 44$.
- For example, for the first 1,000,020 tokens, Heaps' law predicts

  $44 \times 1{,}000{,}020^{0.49} \approx 38{,}323$ terms.
  - The actual number is 38,365 terms, very close to the prediction.

# Statistical Properties of Terms in Information Retrieval (4/8)

- The parameter *k* is quite variable and depends a lot on how we process the collection.
  - Case-folding and stemming reduce the growth rate of the vocabulary.
  - Including numbers and spelling errors will significantly increase it.
- Anyway … Heaps' law suggest that:
  - **The dictionary size will continue to increase** with more documents in the collection.
    - No maximum vocabulary size can be reached.
  - The size of the dictionary will be quite large for large collections.
- These two hypotheses have been shown to be true empirically.

# Statistical Properties of Terms in Information Retrieval (5/8)

- Sometimes … we also want to know how terms are distributed across documents.

- ***Zipf's law*** – modeling the distribution of terms:
  - If we rank terms according to their collection frequency.
    - Term 1 is the most common term in the collection; term 2 is the next most common etc.
  - Then the collection frequency $cf_i$ of the $i$th most common term is proportional to 1/$i$.

$$cf_i \propto \frac{1}{i} \quad \text{or} \quad cf_i \cdot i = c$$

a constant

# Statistical Properties of Terms in Information Retrieval (6/8)

- So if the most frequent term occurs $cf_1$ times …
  - Then the second most frequent term has half as many occurrences.
  - The third most frequent terms a third as many occurrences.
  - The 50th most frequent terms should occur with three times the frequency of the 150th most frequent terms.

| Term | Freq. | Rank | $cf_i * i$ | Term | Freq. | Rank | $cf_i * i$ |
|---|---|---|---|---|---|---|---|
| the | 3332 | 1 | 3332 | two | 104 | 100 | 10400 |
| and | 2972 | 2 | 5944 | turned | 51 | 200 | 10200 |
| a | 1775 | 3 | 5235 | you'll | 30 | 300 | 9000 |
| he | 887 | 10 | 8770 | name | 21 | 400 | 8400 |
| but | 410 | 20 | 8200 | comes | 16 | 500 | 8000 |
| be | 294 | 30 | 8820 | group | 13 | 600 | 7800 |
| there | 222 | 40 | 8880 | lead | 11 | 700 | 7700 |

Empirical evaluation of Zipf's law on Tom Sawyer

# Statistical Properties of Terms in Information Retrieval (7/8)

- Equivalently, we can write Zipf's law as

$$cf_i = ci^k$$

$k = -1$

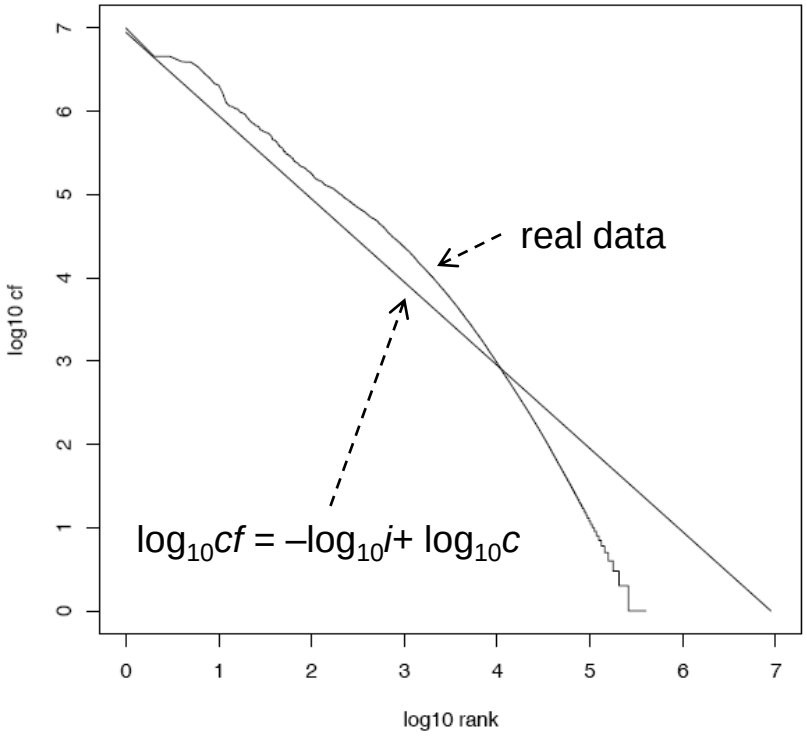
a constant

and

$$\log cf_i = \log ci^{-1}$$
$$= -\log i + \log c$$

- For Reuters-RCV1, the fit of the data to the law is not particularly good.

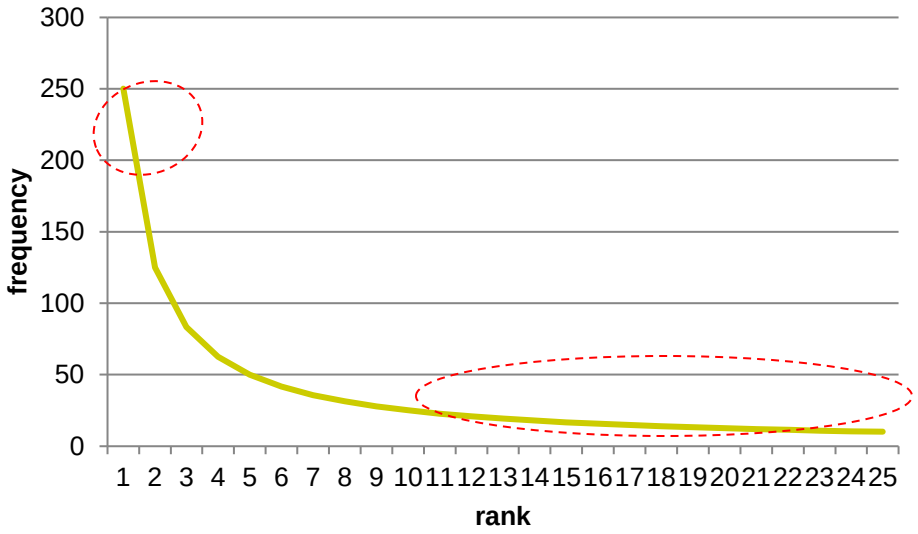
- But, in many cases, Zipf's law is good enough to estimate the distribution of terms in a text collection.




Zipf's law is also a power law with exponent (or slope) $k = -1$.

# Statistical Properties of Terms in Information Retrieval (8/8)

- Zipf's law is useful as a **rough description** of the frequency distribution of words in human language:
  - **There are a few very common words**
  - **And many low frequency words.**




a serious problem in statistic-based text mining, e.g., language modeling

for most words, their use will be extremely **sparse**!!

only for a few words will have lots of examples.