

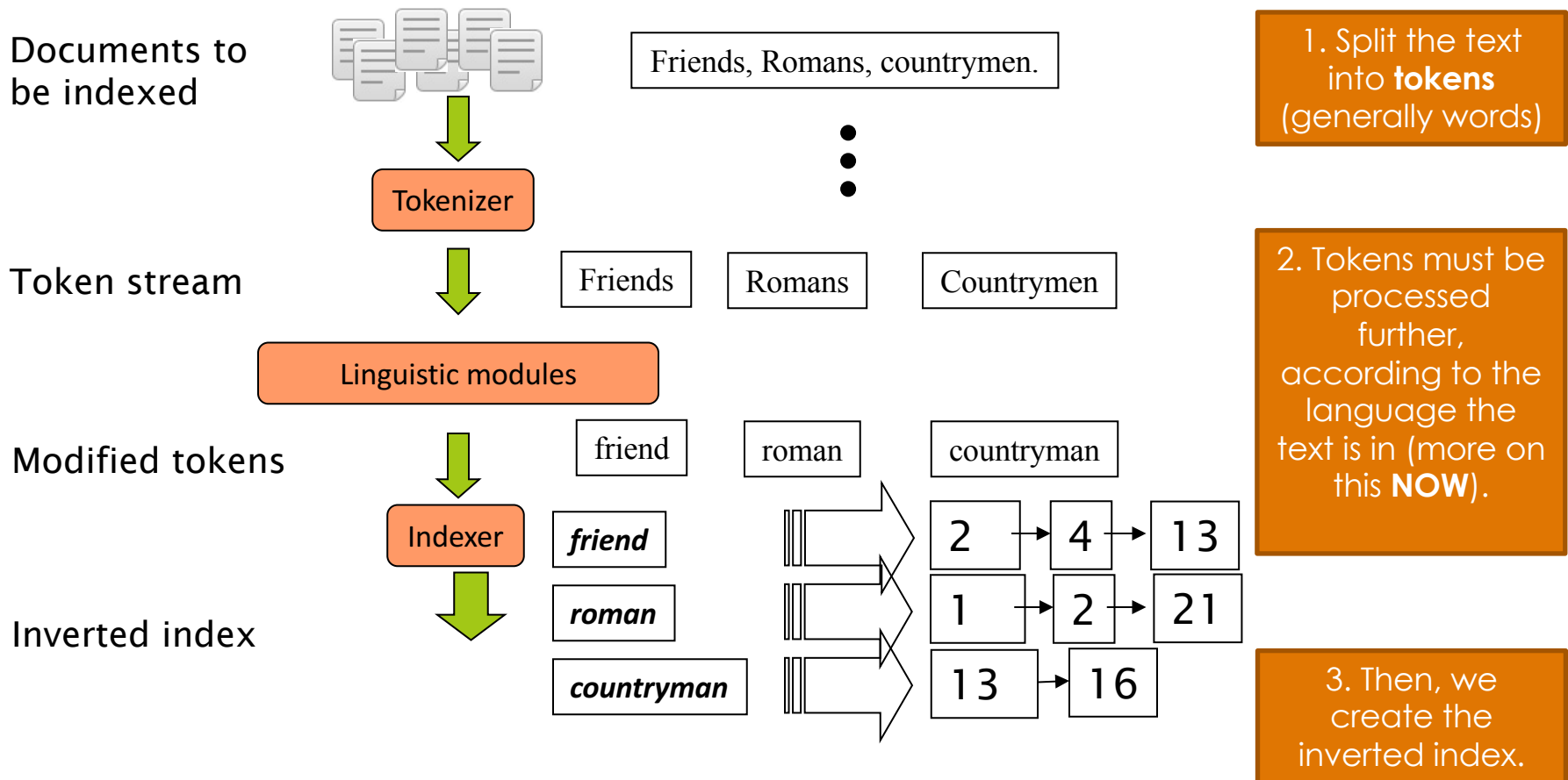
# Topic 3: Preprocessing

## **COMP3009J: Information Retrieval**

Dr. David Lillis ([david.lillis@ucd.ie](mailto:david.lillis@ucd.ie))

UCD School of Computer Science  
Beijing Dublin International College

# Remember: creating an Inverted Index



# Tokenisation (or “Tokenization” if you’re American)

- ▣ **Input:** “Friends, Romans and Countrymen”
- ▣ **Output:** Tokens:
  - ▣ *Friends*
  - ▣ *Romans*
  - ▣ *Countrymen*
- ▣ A **token** is an instance of a sequence of characters. In the previous lecture we said they were similar to words, but they are not the same.
- ▣ Each such token is now a candidate for storing in as an index entry, after **further processing**. We refer to this as **preprocessing** as it occurs before queries are processed by the system.
  - ▣ When we store a token in an index, we call it a **term**.
- ▣ How can we turn tokens into terms?
  - ▣ Libraries are available to deal with most of these situations.

# Tokenisation

- Issues in tokenisation:
  - ***Finland's capital*** → *Finland* AND *s*? *Finlands*? *Finland's*?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***lowercase, lower-case, lower case*** ?
    - It can be effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

# Numbers

■ 3/20/91

*Mar. 12, 1991*

20/3/91

■ 55 B.C.

■ B-52

■ *My PGP key is 324a3df234cb23e*

■ (800) 234-2333

- Often have embedded spaces

- Older IR systems may not index numbers

- But often very useful: think about things like looking up error codes/stacktraces on the web

- Will often index “meta-data” separately

- Creation date, format, etc.

# Tokenisation: language issues

- French
  - **L'ensemble** → one token or two?
    - **L ? L' ? Le ?**
    - Want **l'ensemble** to match with **un ensemble**
      - Until at least 2003, it didn't on Google
        - Internationalization!
- German noun compounds are not segmented
  - **Lebensversicherungsgesellschaftsangestellter**
  - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module
    - Can give a 15% performance boost for German

# Tokenisation: language issues

- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization.
- Further complicated in Japanese, with multiple alphabets intermingled.

フォーチュン500社は情報不足のため時間あた\$500K(約6,000万円)

Katakana   Hiragana   Kanji   Romaji

End-user can express query entirely in hiragana!

# Tokenisation: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right.
- Words are separated, but letter forms within a word form complex ligatures.

استقلت الجزائر في سنة 1962 بعد 132 عام من الاحتلال الفرنسي.

← →      ← →      ← start

- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’
- With Unicode, the surface presentation is complex, but the stored form is straightforward.



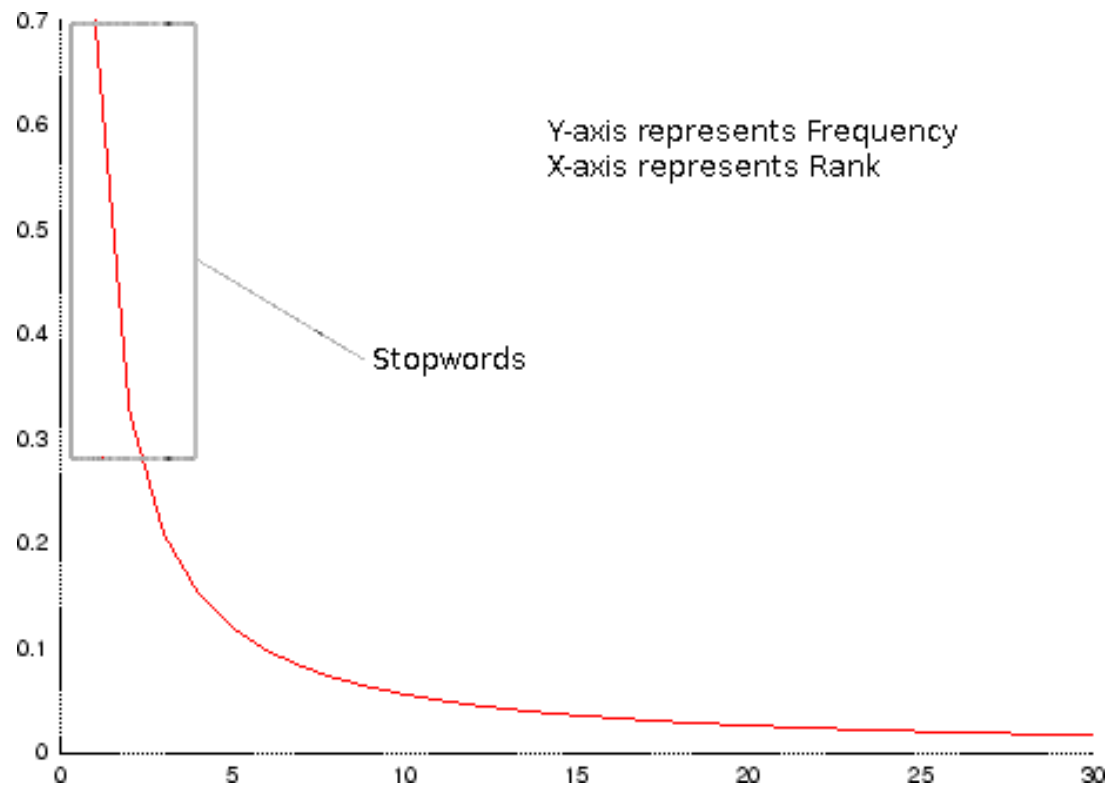
# Stopword Removal

- A **stopword** is a commonly occurring term that appears in so many documents that it does not add to the meaning of the document, and appear in most English texts (except for very short ones).
- For example, in English, terms such as *the*, *and*, *of*, etc. tell a reader nothing about a document's meaning.
- They are so common that they appear in almost all documents, and of little use when differentiating between documents.
- The fact that they are so common also means that more processing power is required to deal with these terms than others that are less common.
- This additional processing, combined with their relative lack of usefulness means that they are often removed from the documents at the preprocessing stage.

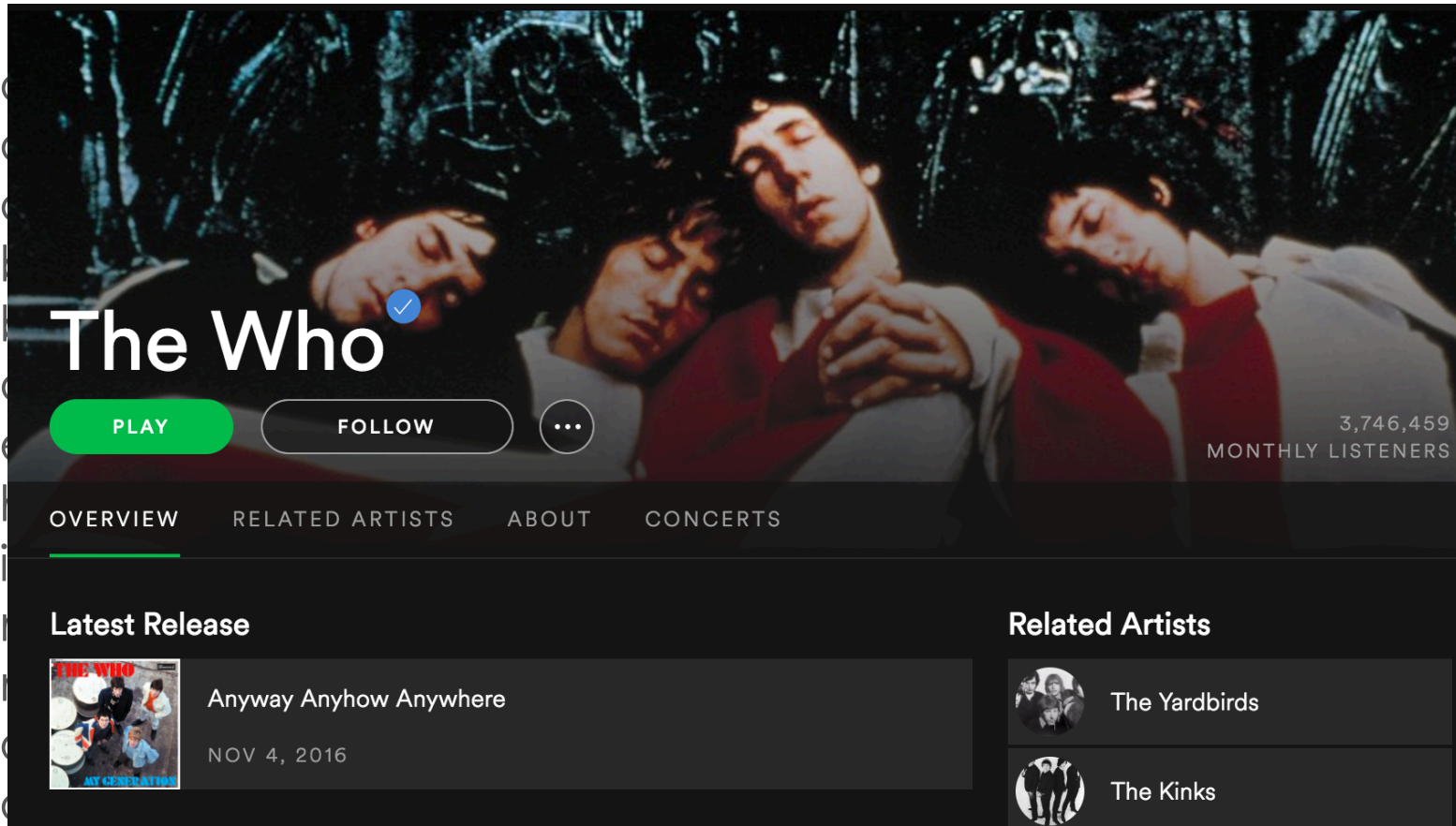
# Stopword Removal

- Stopwords can be estimated using **Zipf's Law**.
- George Kingsley Zipf analysed the statistical occurrence of words (terms) in text.
- He noted that while only a few terms are used very often, more are used only rarely.
- If we rank terms from the most commonly used to the least commonly-used in a large text collection, the second term is approximately  $\frac{1}{2}$  as common as the first, the third is approximately  $\frac{1}{3}$  as common as the first, etc.
- We use this to identify what terms count as stopwords in various languages.

# Stopword Removal: Zipf's Law



a about above across adj after again against all almost alone along  
also although always am among an and another any anybody  
anyone anything anywhere apart are around as aside at away be  
because been before behind being below besides between beyond  
both but by can cannot could deep did do does doing done down  
downwards during each either else enough etc even ever every  
everybody everyone except far few for forth from get gets got had  
hardly has have having her here herself him himself his how however i  
if in indeed instead into inward is it its itself just kept many maybe  
might mine more most mostly much must myself near neither next no  
nobody none nor not nothing nowhere of off often on only onto or  
other others ought our ours out outside over own p per please plus pp  
quite rather really said seem self selves several shall she should since so  
some somebody somewhat still such than that the their theirs them  
themselves then there therefore these they this thorough thoroughly  
those through thus to together too toward towards under until up  
upon v very was well were what whatever when whenever where  
whether which while who whom whose will with within without would  
yet young your yourself




The Who

PLAY FOLLOW ...


3,746,459 MONTHLY LISTENERS


OVERVIEW RELATED ARTISTS ABOUT CONCERTS

Latest Release

 Anyway Anyhow Anywhere  
NOV 4, 2016

Related Artists

 The Yardbirds

 The Kinks

e along  
dy  
ay be  
beyond  
e down  
very  
ot had  
however i  
ybe  
next no  
nto or  
e plus pp  
d since so

some somebody somewhat still such than that **the** their theirs them  
themselves then there therefore these they this thorough thoroughly  
those through thus to together too toward towards under until up  
upon v very was well were what whatever when whenever where  
whether which while **who** whom whose will with within without would  
yet young your yourself

# Stopword Removal

- ❑ There is a **tradeoff** required when using stopwords.
- ❑ The trend is away from removing them:
  - ❑ Good compression means that space required is small.
  - ❑ Good optimisation means processing them takes little time.
- ❑ Some queries will become impossible if all stopwords are removed.
  - ❑ Phrases: “King **of** Denmark”
  - ❑ Song titles, etc. “Let **it be**”, “**To be or not to be**”
  - ❑ Relational queries: “Flights **to** London”
- ❑ Some solutions:
  - ❑ Don't remove stopwords. Use all words as terms.
  - ❑ Recognise when combinations of stopwords are meaningful, and include these as terms in the index.

# Stemming

- Many terms in natural language appear different to a computer, but represent the same concept.
- In English, terms such as “connect”, “connecting”, “connects”, “connected”, “connector”, etc. represent a similar concept.
- Many IR systems believe that a search for “computing” should also include documents that contain the word “computer”, for example.
- **Stemming** is the process that maps these terms to a **common root**.
- The end of a term is called a **suffix** (-er, -s, -ing, -ed, etc.)
- Hence stemming is also known as **suffix stripping**.

# Stemming

- While there have been many algorithms to accomplish stemming, the most famous (for English text) is **Porter's Stemming Algorithm**.
- The rules that it follows are detailed in:
  - Martin Porter, An Algorithm for Suffix Stripping, Program, 14 no. 4, pp. 130-137, July 1980.
- Numerous implementations are available at:
  - <http://tartarus.org/~martin/PorterStemmer>
- It should be noted that Porter's stemming algorithm is not necessarily the most accurate, though it is the most widely used. Snowball and Porter2 are often considered to be more effective.



# Stemming Algorithms

- Stemming Example: each of the following news headlines include words with a common stem:
  - accept: Congress decides to **accept** healthcare bill.
  - acceptable: Congress deems healthcare bill **acceptable**.
  - acceptance: Obama pleased after healthcare bill **acceptance**.
  - accepted: Congress has **accepted** President Obama's healthcare bill.
  - accepts: Congress **accepts** healthcare bill.
- Clearly, these would be relevant to the same query.

# Stemming: Problems

- Sometimes suffixes can be removed from words that are **not related**, but that end up being the same afterwards.
- This is known as **overstemming**.
- Stemming is a very useful process to help words of similar meanings to be matched, but it is not a perfect process.
- **Example:** consider the words “prove” and “provable”
- Now what about “probe” and “probable”?
- Using a dictionary can help avoid this type of problem

# Stemming: Problems

- Although not specifically a problem with stemming, there is also an issue with words that do not have the same meaning, but are spelled the same (known as **homographs**).
  - At the Battle of Waterloo, Napoleon's forces were **routed**.
    - The army was heavily defeated in battle.
  - The cars were **routed** off the motorway.
    - The cars were sent in a different direction.
- It is very difficult to tell the difference between these without using Natural Language Processing, which is a much slower process.

# Normalisation

- We may need to “normalise” tokens so that they become terms in the same form.
  - e.g. we want *USA* and *U.S.A.* to match.
- Some common approaches:
  - Changing all tokens to lowercase.
    - Even if something should have uppercase letters in it, users often type in lowercase anyway.
  - Delete full stops to form terms: *USA, U.S.A.* → *usa*
  - Delete hyphens:  
*anti-discriminatory, antidiscriminatory* → *antidiscriminatory*

# Thesauri and soundex

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
    - **car** = **automobile**      **color** = **colour**
  - We can rewrite to form equivalence-class terms
    - When the document contains **automobile**, index it under **car-automobile** (and vice-versa)
  - Or we can expand a query
    - When the query contains **automobile**, look under **car** as well
- What about spelling mistakes?
  - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics (i.e. it indexes terms using their sounds rather than their spelling).

# Phrase Queries

- Want to be able to answer queries such as “**stanford university**” – as a phrase.
- Thus the sentence “*I went to university at Stanford*” is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
- For this, it is not enough to store only  
<term : docs> entries

# A first attempt: Biword indexes

- Index every consecutive **pair** of terms in the text as a phrase.
- For example the text “Friends, Romans, Countrymen” would generate the biwords (bi- is a prefix used to mean that there are two of something).
  - *friends romans*
  - *romans countrymen*
- Each of these biwords is now a dictionary term.
- Two-word phrase query-processing is now immediate.

# Longer phrase queries

- “*stanford university palo alto*” can be broken into the Boolean query on biwords:

*stanford university AND university palo AND palo alto*

Without the documents themselves, we cannot verify that the docs matching the above Boolean query do contain the exact phrase.



Can have false positives!



# Issues for biword indexes

- False positives, as noted before.
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them.
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy.

## Solution 2: Positional indexes

- In the postings, store for each **term** the position(s) in which tokens of it appear:

<**term**, number of docs containing **term**;

doc1: position1, position2 ... ;

doc2: position1, position2 ... ;

etc.>

# Positional index example

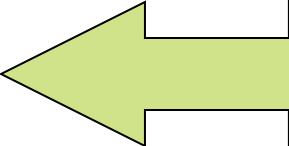
<*be*: 993427;

*1*: 7, 18, 33, 72, 86, 231;

*2*: 3, 149;

*4*: 17, 191, 291, 430, 434;

*5*: 363, 367, ...>



Which of docs *1,2,4,5*  
*could* contain “*to be*  
*or not to be*”?

- For phrase queries, we use a merge algorithm recursively at the **document** level.
- But we now need to deal with more than just equality: we look for terms that follow each other.

# Positional index size

- You can compress position values.
- **BUT**, a positional index expands postings storage *substantially*
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase queries.

# Rules of thumb

- A positional index is 2–4 times as large as a non-positional index.
- Positional index size 35–50% of volume of original text.
- Caveat: all of this holds for “English-like” languages.
- Often, a combination of selected biwords and a positional index are used, with good results.

# Conclusions

- When creating an index from a set of tokens, several challenges are presented.
- These challenges depend on the language that the document collection is written in.
- Several approaches have been proposed to address these problems.
  - For many approaches, a tradeoff is required, as they do not improve retrieval in all circumstances.