COMP6237 Data Mining

Searching and Ranking

Jonathon Hare
jsh2@ecs.soton.ac.uk

Note: portions of these slides are from those by ChengXiang “Cheng” Zhai at UIUC
https://class.coursera.org/textretrieval-001

Introduction

- Information retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources.
    - Searches can be based on metadata or on content-based indexing.
        - Content need not be textual!
- Automated information retrieval systems are used to reduce what has been called "information overload".

Problem statement: It's all about the user

- User has an **information need**
  - A **query**
    - Could be specific: What is the population of Southampton?
    - Or vague: I am going to a conference in Arizona in July. What do I need to know?
- IR system must find documents that are relevant to the users' query
  - Clearly a classic data mining problem!

Historical Context

- ~4000 years ago
  - Table of contents and the **index** data-structure developed
- 1876
  - **Dewey decimal** system
    - Classification Scheme
      - Commonly used in libraries

Historical Context

- 1960's
  - basic advances in automatic indexing using computers
- 1970's
  - Probabilistic and vector-space models
  - Clustering, relevance feedback
  - Large, on-line Boolean IR systems
- 1980's
  - Natural language processing and IR
  - Expert Systems
  - Commercial-Off-the-Shelf IR systems

Historical Context

- 1990's onwards
  - Dominance of ranking
  - The Internet!
    - Web-based IR
    - Distributed IR
  - Multimedia IR
  - Statistical language modelling
  - User feedback
- Last ~10 years
  - Really "Big data"
  - Better language modelling
  - Even better ranking

Text Retrieval

What is text retrieval

- Collection of text documents exists
- User gives a query to express the information need
- Search engine system returns relevant documents to users
- Known as “search technology” in industry

Text retrieval versus database retrieval

- Information
  - Unstructured/free text vs. structured data – Ambiguous vs. well-defined semantics
- Query
  - Ambiguous vs. well-defined semantics – Incomplete vs. complete specification
- Answers
  - Relevant documents vs. matched records
- TR is an empirically defined problem
  - Can’t mathematically prove one method is better than another
  - Must rely on **empirical evaluation** involving users!

Classical Models of Retrieval

- **Boolean Model**
  - Does the document satisfy the Boolean expression?
    - “winter” AND “Olympics” AND (“ski” OR “snowboard”)
- **Vector Space Model**
  - How similar is the document to the query?
    - [(Olympics 2) (winter 2) (ski 1) (snowboard 1)]
- **Probabilistic Model**
  - What is the probability that the document is generated by the query?

Classical Models of Retrieval

Result Selection

- **Boolean Model**
  - Does the document satisfy the Boolean expression?
    - “winter” AND “Olympics” AND (“ski” OR “snowboard”)

Result Ranking

- **Vector Space Model**
  - How similar is the document to the query?
    - [(Olympics 2) (winter 2) (ski 1) (snowboard 1)]
- **Probabilistic Model**
  - What is the probability that the document is generated by the query?

Problems of Result Selection

- The classifier is unlikely accurate
  - “Over-constrained” query -> no relevant documents to return
  - “Under-constrained” query -> over delivery
  - Hard to find the right position between these two extremes
- Even if it is accurate, all relevant documents are not equally relevant (relevance isn’t a binary attribute!)
- Prioritisation is needed
  - Thus, ranking is generally preferred

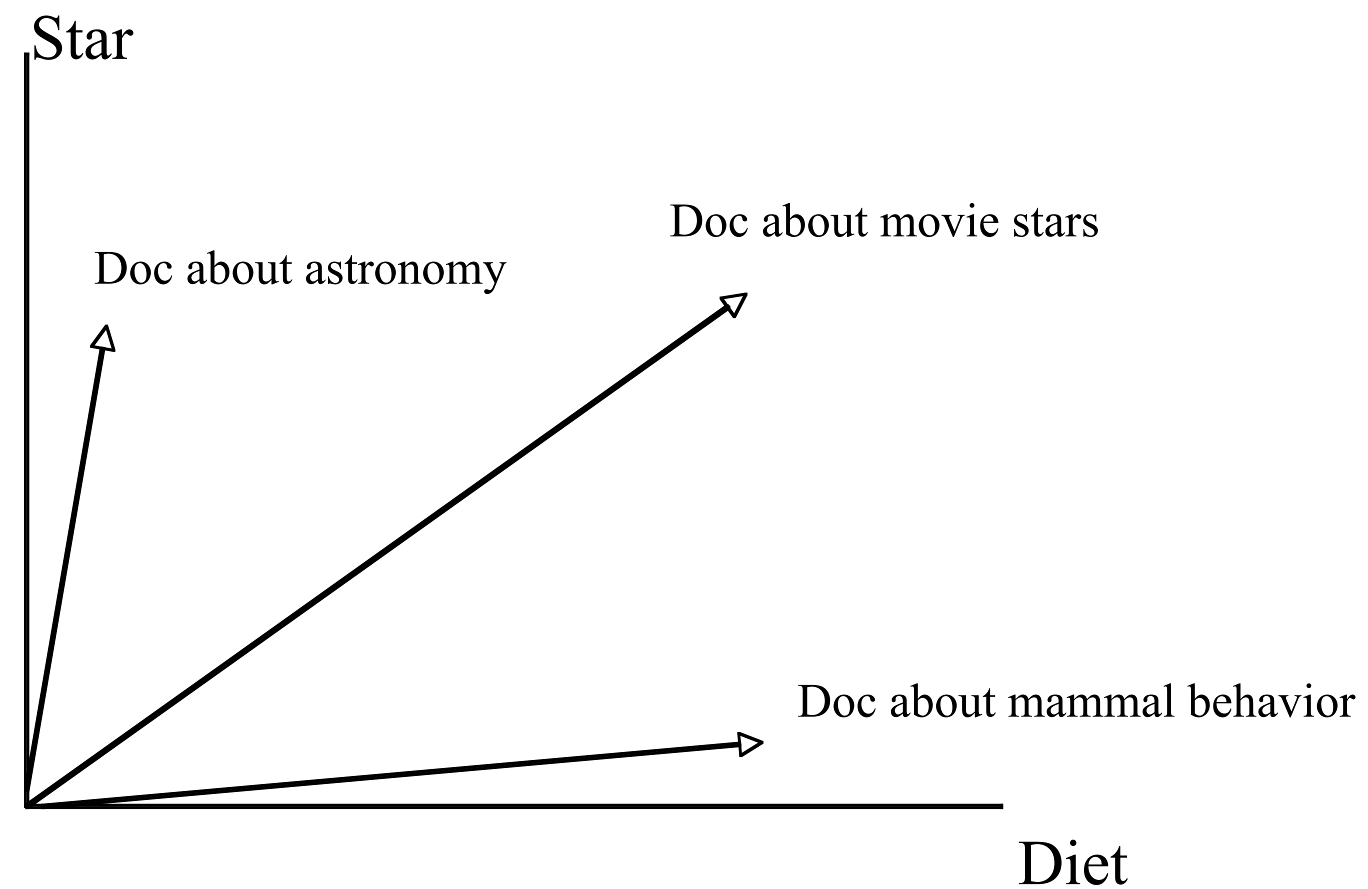
Ranking versus selecting results

- **Probability Ranking Principle** [Robertson 77]:
    - Returning a ranked list of documents in descending order of probability that a document is relevant to the query is the optimal strategy under the following two assumptions:
        - The utility of a document (to a user) is **independent** of the utility of any other document
        - A user would browse the results **sequentially**
- Do these two assumptions hold?

The Vector-Space Model of Retrieval

The Vector-Space Model (VSM)

- Conceptually simple:
  - Model each document by a vector
  - Model each query by a vector
  - Assumption: documents that are “close together” in space are similar in meaning.
  - Develop similarity measures to rank each document to a query in terms of decreasing similarity



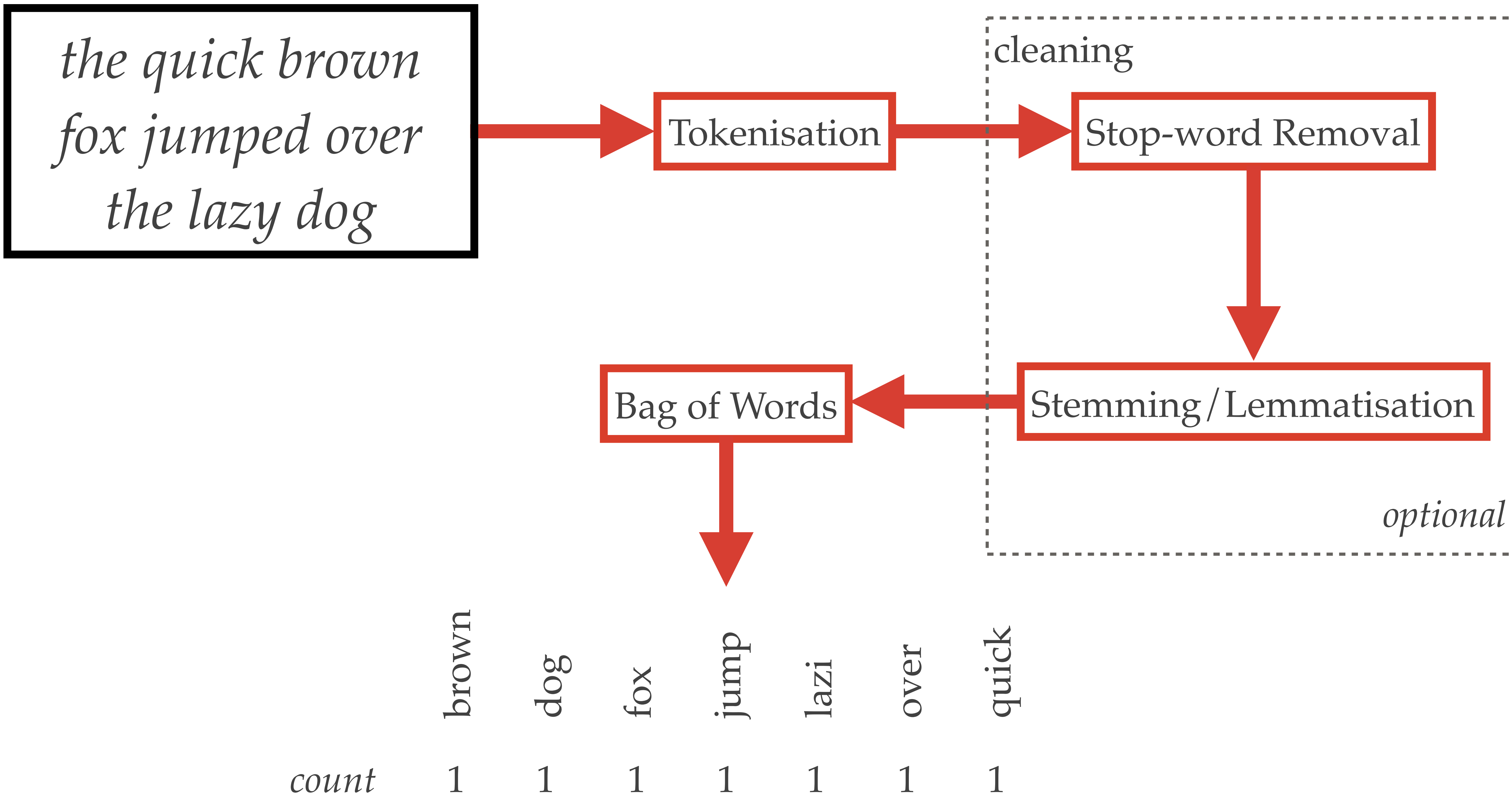
VSM Is a Framework

- Represent a doc/query by a term vector
  - **Term**: basic concept, e.g., word or phrase
  - Each term defines one dimension
  - N terms define an **N-dimensional space**
  - **Query vector**: $\mathbf{q}=(x_1,\ldots,x_N)$, $x_i \in \Re$ is query term weight
  - **Doc vector**: $\mathbf{d}=(y_1,\ldots,y_N)$, $y_j \in \Re$ is doc term weight
- relevance(d|q) $\propto$ similarity($\mathbf{q}$,$\mathbf{d}$) = f($\mathbf{q}$,$\mathbf{d}$)

What VSM Doesn’t Say

- How to define/select the “basic concept”
  - Concepts are assumed to be orthogonal
- How to place docs and query in the space (= how to assign term weights)
  - Term weight in query indicates importance of term
  - Term weight in doc indicates how well the term characterises the doc
- How to define the similarity measure, f(**q**,**d**)

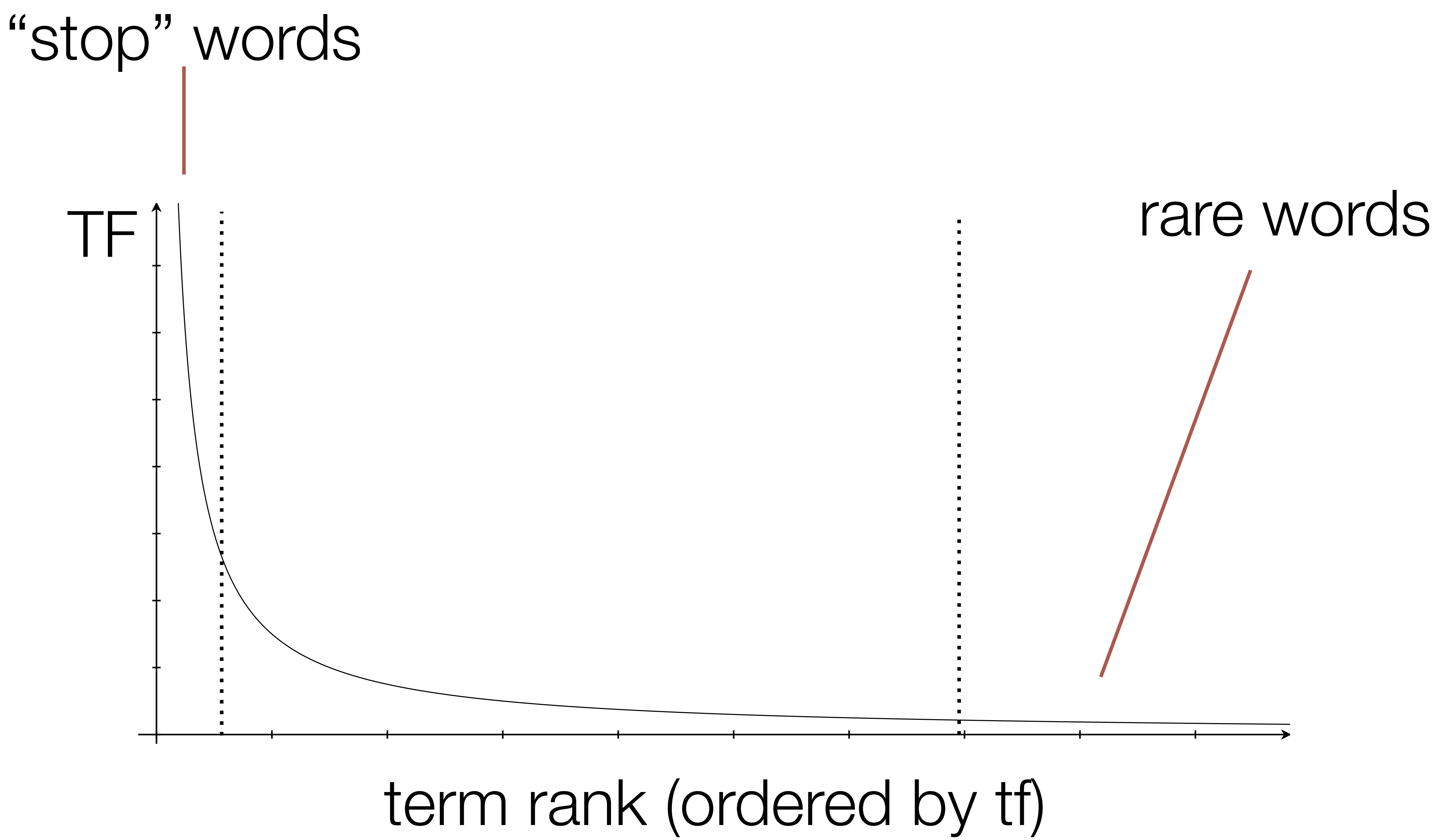
Text processing (feature extraction)



Bag of Words Vectors

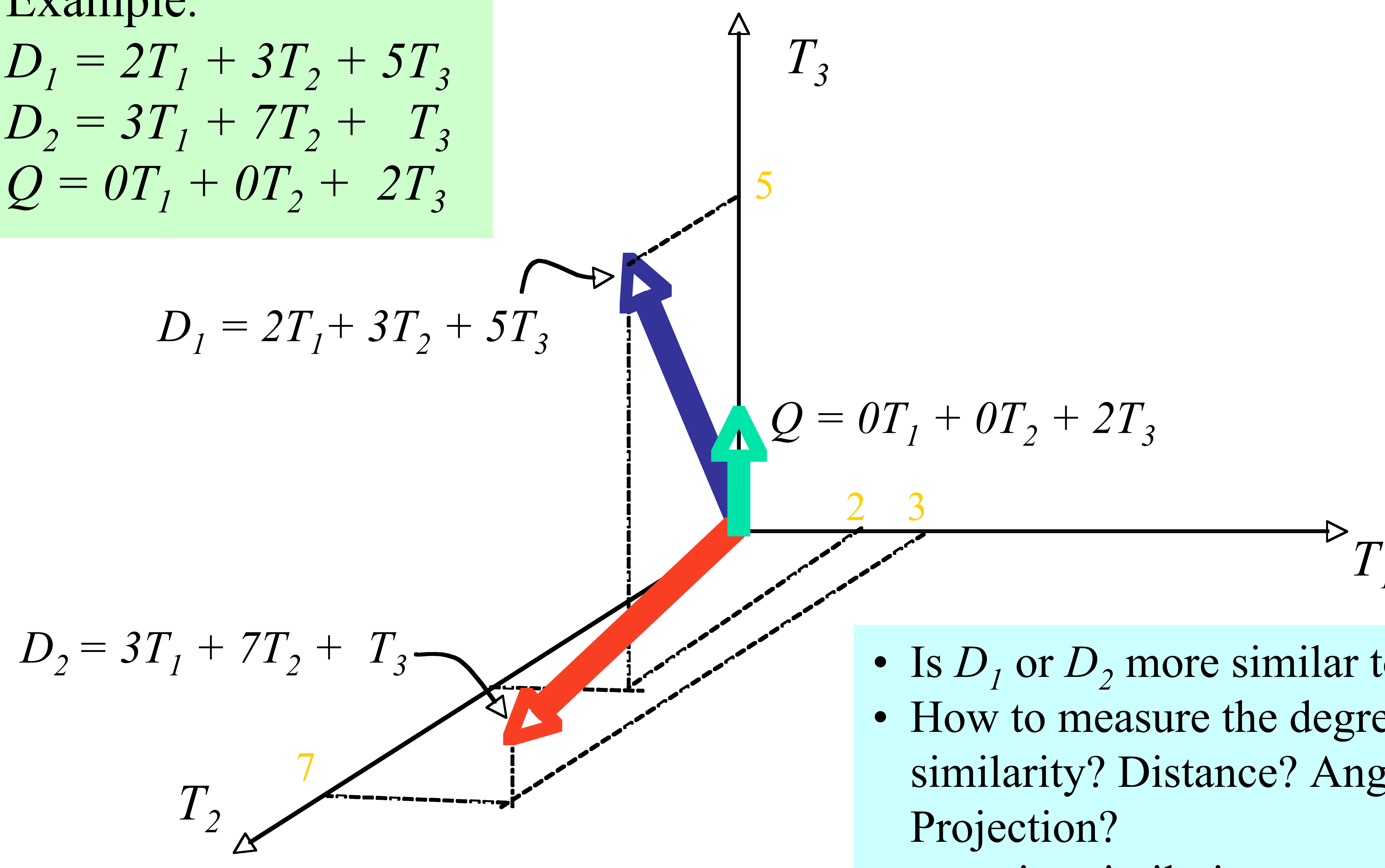
- The lexicon or vocabulary is the **set** of all (processed) words across all documents known to the system.
- We can create vectors for each document with as many dimensions as there are words in the lexicon.
  - Each word in the document's bag of words contributes a count to the corresponding element of the vector for that word.
    - In essence, each vector is a histogram of the word occurrences in the respective document.
    - **Vectors will have very high number of dimensions, but will be very sparse.**

Aside: language statistics and Zipf's Law



Searching the VSM

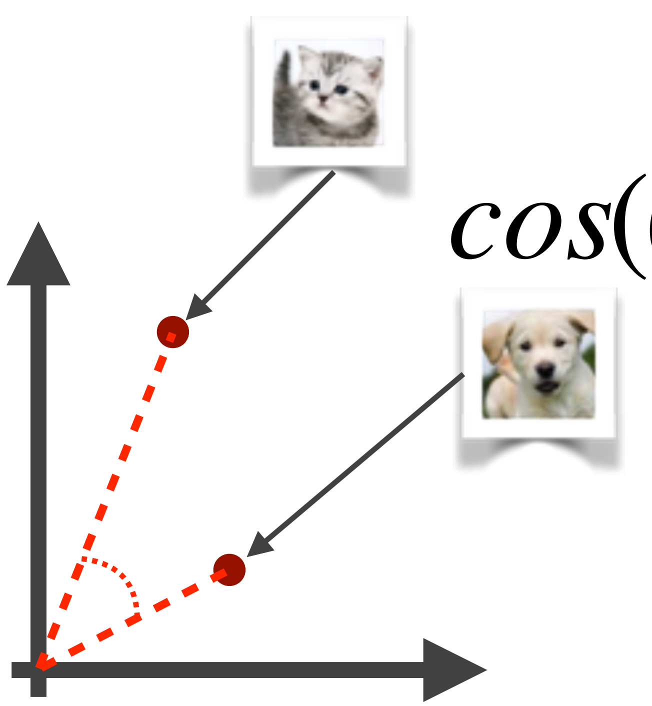
Example:
$D_1 = 2T_1 + 3T_2 + 5T_3$
$D_2 = 3T_1 + 7T_2 + T_3$
$Q = 0T_1 + 0T_2 + 2T_3$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?
- …cosine similarity

Recap: Cosine Similarity

If *p* and *q* are both high dimensional and sparse, then you’re going spend a lot of time multiplying 0 by 0 and adding 0 to the accumulator



$$cos(\theta) = \frac{p.q}{||p|| \, ||q||} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}}$$

These can be pre-computed and stored!

Inverted Indexes

Aardvark	[doc3:4]
Astronomy	[doc1:2]
Diet	[doc2:9; doc3:8]
…	
Movie	[doc2:10]
Star	[doc1:13; doc2:4]
Telescope	[doc1:15]

…A map of words to lists of postings…

Inverted Indexes

Aardvark	[doc3:4]
Astronomy	[doc1:2]
Diet	[doc2:9; doc3:8]
...	
Movie	[doc2:10]
Star	[doc1:13, doc2:4]
Telescope	[doc1:15]

A **posting** is a pair formed by a **document ID** and the **number of times** the specific word appeared in that document

Computing the Cosine Similarity

- For each word in the query, lookup the relevant postings list and accumulate similarities for only the documents seen in those postings lists
  - much more efficient than fully comparing vectors…

Query: “**Movie Star**”

Aardvark	[doc3:4]
Astronomy	[doc1:2]
Diet	[doc2:9; doc3:8]
…	
Movie	[doc2:10]
Star	[doc1:13; doc2:4]
Telescope	[doc1:15]

Query: “**Movie Star**”

Accumulation table:

doc2	10x1

Aardvark	[doc3:4]
Astronomy	[doc1:2]
Diet	[doc2:9; doc3:8]
…	
Movie	[doc2:10]
Star	[doc1:13; doc2:4]
Telescope	[doc1:15]

Query: “**Movie Star**”

Accumulation table:

doc2	10×1 + 4×1
doc1	13×1

Aardvark	[doc3:4]
Astronomy	[doc1:2]
Diet	[doc2:9; doc3:8]
…	
Movie	[doc2:10]
Star	[doc1:13; doc2:4]
Telescope	[doc1:15]

Query: "**Movie Star**"

Accumulation table:

doc2	(10×1 + 4×1) / 14.04 = **0.997**
doc1	13×1 / 19.95 = **0.652**
doc3	***0***

Aardvark	[doc3:4]
Astronomy	[doc1:2]
Diet	[doc2:9; doc3:8]
…	
Movie	[doc2:10]
Star	[doc1:13; doc2:4]
Telescope	[doc1:15]

Weighting the vectors

- The number of times a term occurs in a document reflects the importance of that term in the document.
- Intuitions:
  - A term that appears in many documents is not important: e.g., the, going, come, …
  - If a term is frequent in a document and rare across other documents, it is probably important in that document.

Possible weighting schemes

- Binary weights
  - Only presence (1) or absence (0) of a term recorded in vector.
- Raw frequency
  - Frequency of occurrence of term in document included in vector.
- *TF-IDF*
  - Term frequency is the frequency count of a term in a document.
  - Inverse document frequency (idf) provides high values for rare words and low values for common words.
  - *(Note there are many forms of TF-IDF and that it isn't one specific scheme)*

Actual scoring functions: baseline

- Key bit of the cosine similarity is the dot-product between the query and document
  - Use this as a basis for building a scoring function that incorporates TF and IDF

Total #docs in collection

Number of times *w* appears in *d*

$$f(\mathbf{q}, \mathbf{d}) = \sum_{i=1}^{N} q_i y_i = \sum_{w \in q \cap d} c(w, q) c(w, d) log \frac{M + 1}{df(w)}$$

Number of times *w* appears in *q*

document frequency (number of docs containing *w*)

Better scoring functions

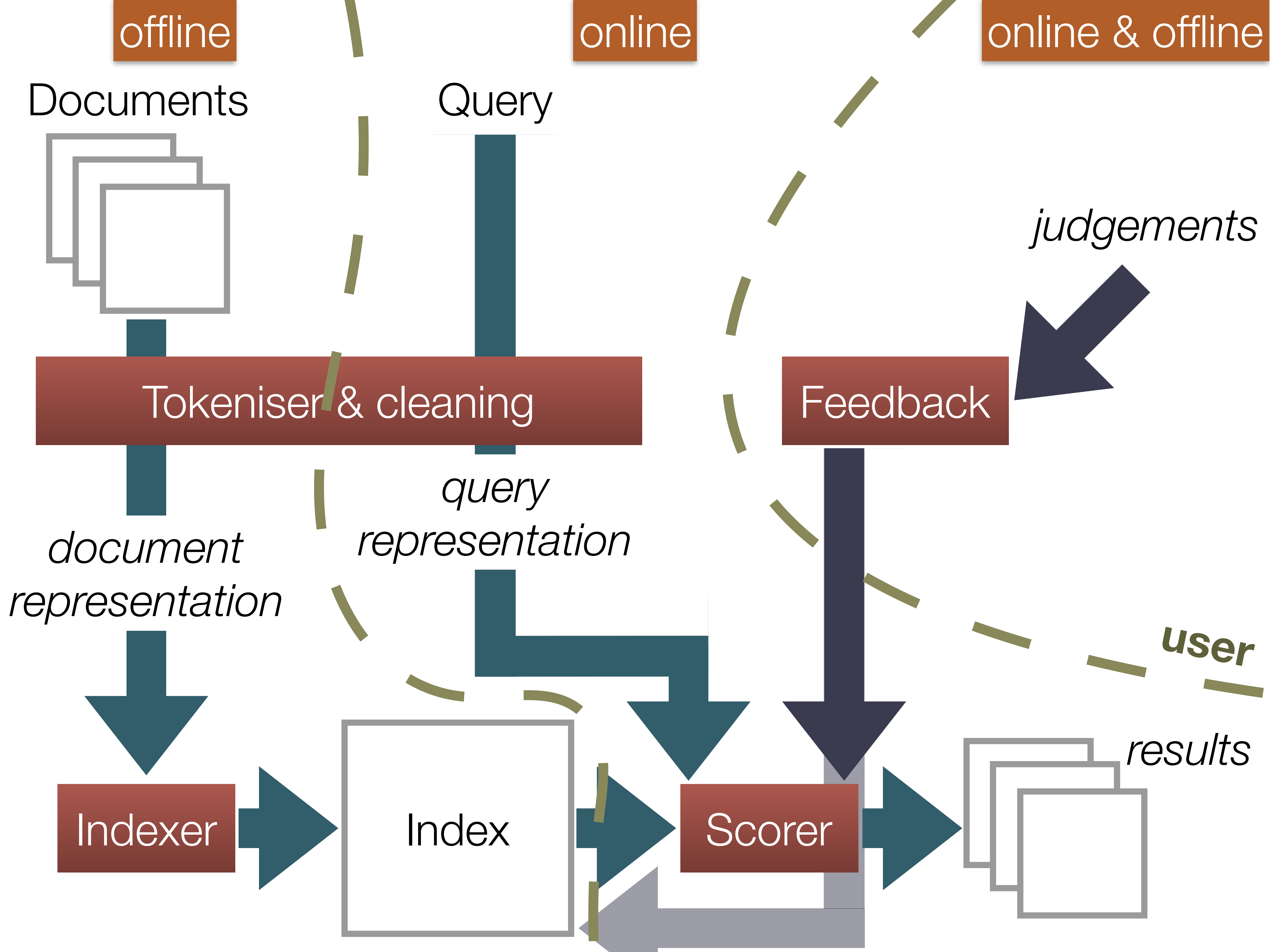
- The baseline TF-IDF scoring function has a few problems
  - skewed scores if a document contains a word many occurrences
  - what if the document is long?
- Lots of other scoring functions out there…
  - BM25 is one of the most popular families of functions
    - Developed as part of the Okapi retrieval system at City University in the 80s

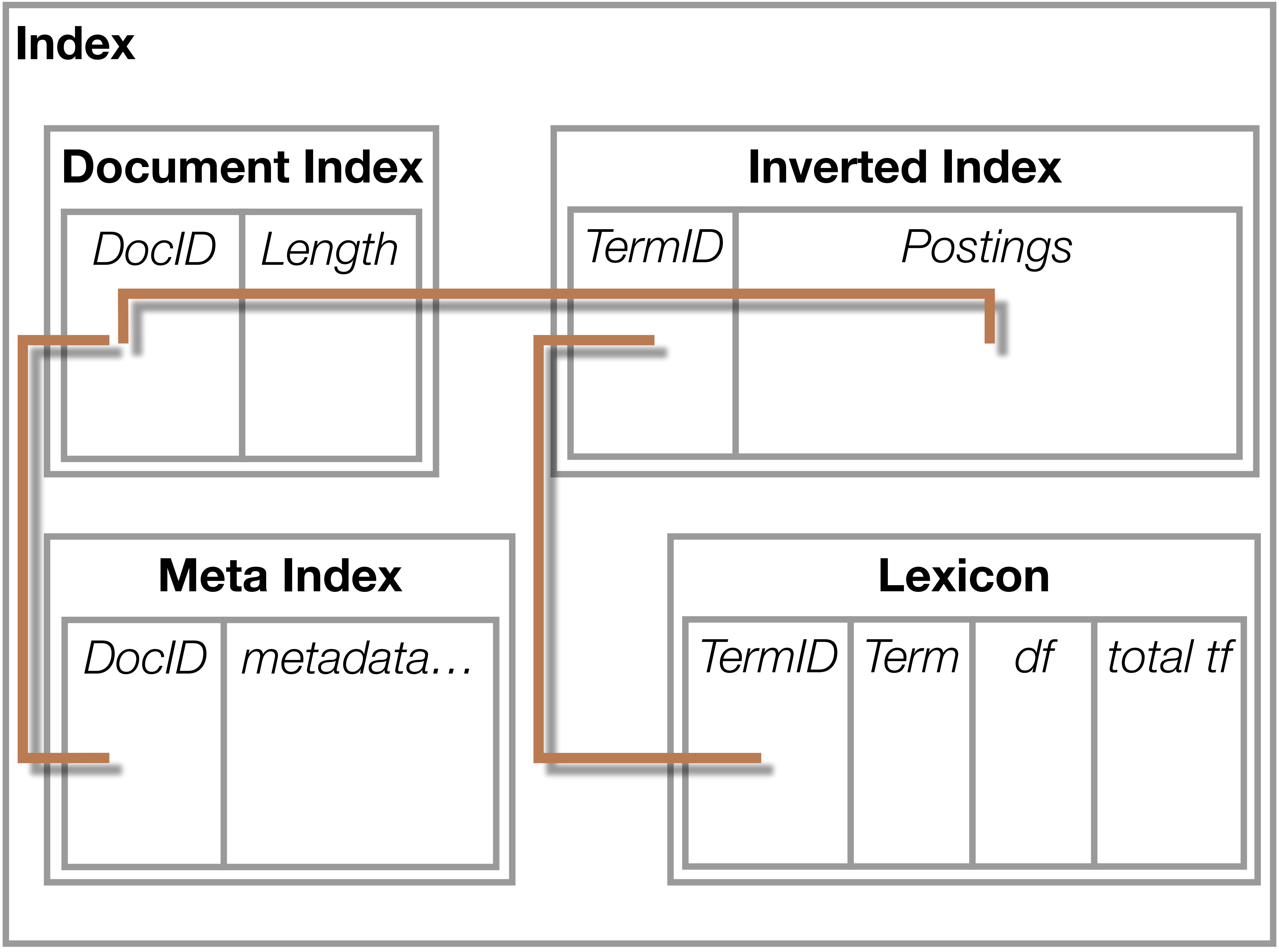
BM25

$$f(q,d) = \sum_{w \in q \cap d} \frac{c(w,q) \cdot c(w,d) \cdot (k_1 + 1)}{c(w,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)} \cdot \log \frac{M - df(w) + 0.5}{df(w) + 0.5}$$

k_1, b are constants
|d| is the length of doc d
avgdl is the average doc length in the collection

Inside a retrieval system

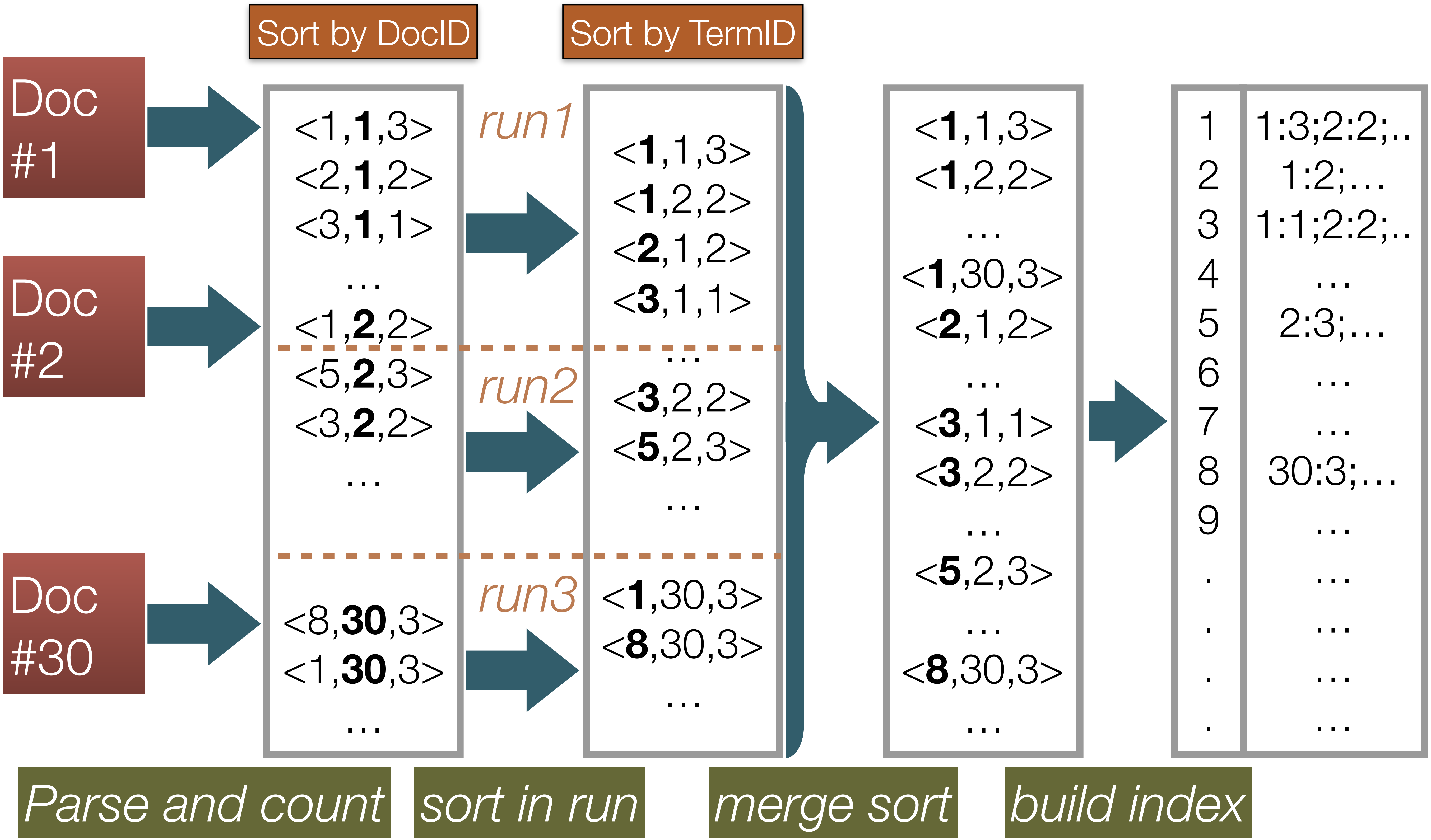

offline
online
online & offline
Documents
Query
judgements
Tokeniser & cleaning
Feedback
query
representation
document
representation
user
results
Indexer
Index
Scorer


Index
Document Index
DocID
Length
Inverted Index
TermID
Postings
Meta Index
DocID
metadata…
Lexicon
TermID
Term
df
total tf

Building an inverted index

- Difficult to build a huge index with limited memory
- Memory-based methods: not usable for large collections
- Sort-based methods:
  - Step 1: Collect local <termID, docID, freq> tuples in a *run*
  - Step 2: Sort tuples within the run and write to disk
  - Step 3: Pair-wise merge runs on disk
  - Step 4: Output inverted file

Sort-based Inversion



Inverted Index Compression

- Leverage skewed distribution of values and use variable-length integer encoding
- TF compression
  - Small numbers tend to occur far more frequently than large numbers (c.f. Zipf)
  - Fewer bits for small (high frequency) integers at the cost of more bits for large integers
- DocID compression
  - “delta-gap” or (“d-gap”) (store difference): d1, d2-d1, d3-d2,...
  - Feasible due to sequential access
- Methods:
  - Oblivious: Binary code, unary code, γ-code, δ-code, varint...
  - List-adaptive: Golomb, Rice, Frame of Reference (FOR), Patched FOR (PFOR), …

Improving search engine ranking

Location-weighting; Phrase and Proximity search

- Can we use the position of the query terms in the document to improve ranking?
  - Document more likely to be relevant if the query terms occur nearer beginning?
  - Allow for exact “phrase matching” or proximity search (query terms appearing *close* to each other)
  - For any of these to be tractable, the term position of every term in every document needs to be indexed…

Index Payloads

Positional inverted index augments the postings with a list of term offsets from the beginning of the document

term1	[doc3:4:⟨21,28,100,311⟩, …]
…	…

Obviously increases size of index dramatically, so compression is very important - typically use d-gap+γ or d-gap+FOR encoding

Retrieving hypertext: Using inbound links

- Assume we are indexing text documents with hyperlinks (i.e. webpages)
  - Can we use the hyperlinks to improve search?
    - perhaps increase the score of documents that have many other documents linking to them
      - unfortunately prone to manipulation - easy to set up new pages with lots of links…
    - Solution: **PageRank**
      - Compute the importance of a page from the importance of all the other pages that link to it and the number of links each other page has.

Making use of feedback: Click Ranking

- Can we make use of user feedback?
  - When a user performs a search, which results do they look at?
  - Could improve ranking by:
    - *increasing* the weighting of documents that are clicked (irrespective of the query)
    - *learning* associations between queries and documents and using this to re-rank documents in response to a specific query

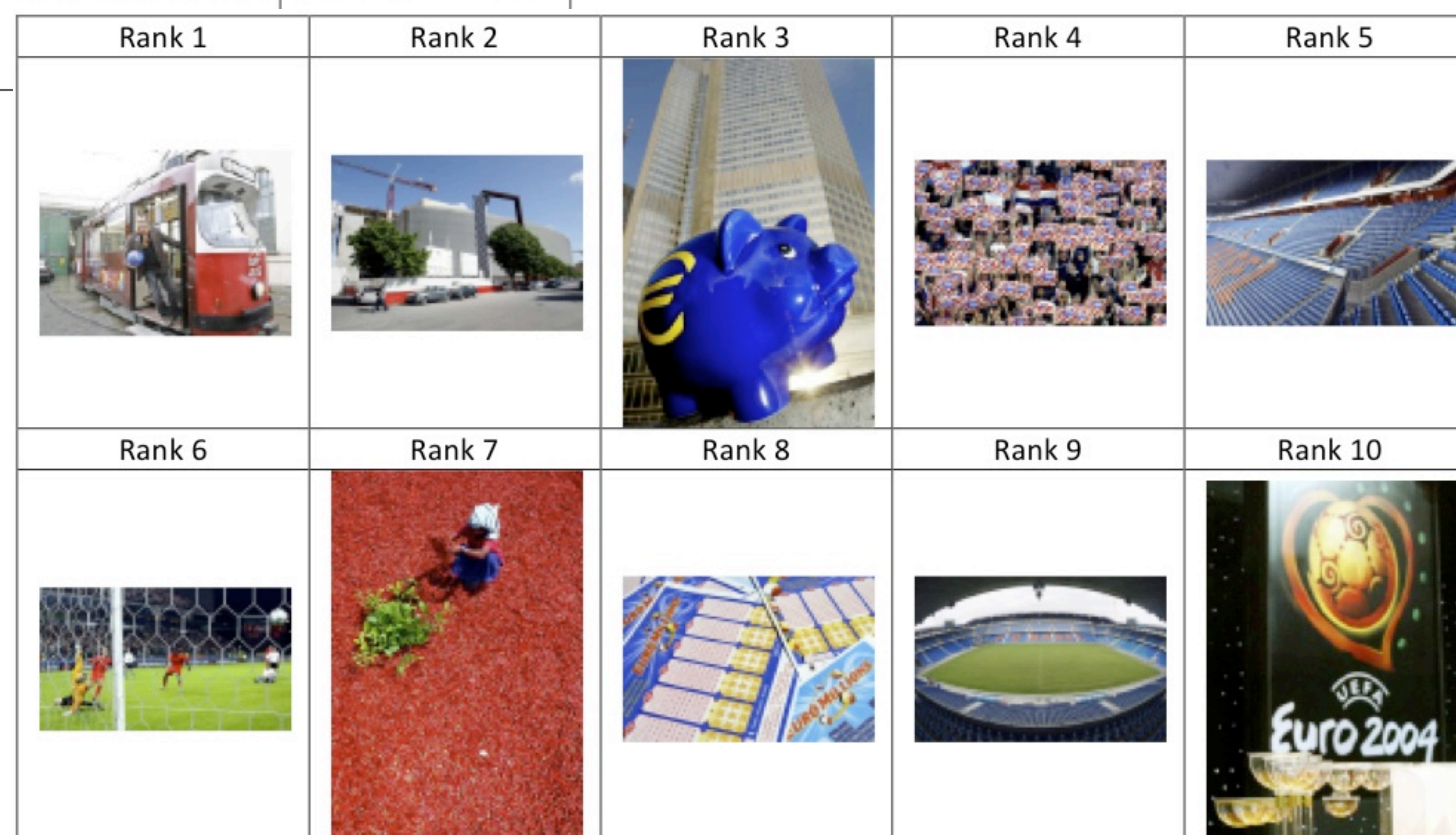
Current challenges in IR: Diverse ranking

Implicit Search Result Diversification

- Most current search systems always assume that a ranked list of results will be generated
  - Is this optimal?
- Diversity in search result rankings is needed when users' queries are poorly specified or ambiguous.
  - e.g. what if I perform a web search for “jaguar”
  - By presenting a diverse range of results covering all possible representations of a query the probability of finding relevant documents is increased
    - very much a data-mining problem!

Search for images related to "euro" (using textual metadata):

Original Ranking



Diversified Ranking

Summary

- Search engines are a key tool for data mining
  - They can help a user understand what is in a data set by effectively providing target access
- The techniques required to build a search engine are important:
  - Content analysis/feature extraction is of key importance
  - Low-level techniques for scalability and efficiency used by search engines have applications in other areas
  - Search engines can provide a base for performing other types of data mining efficiently and effectively