# Searching and Ranking

## Summary

Search engines are a key tool for data mining; they in particular help us find *relevant information* that meets an *information need*. In this lecture we'll focus on text search, although you should note the core techniques can also be applied to non-textual data. We'll look at how text documents can be processed (feature extraction) and indexed to allow for *efficient* retrieval. We'll also look in some detail about how documents can be ranked against a query to make the retrieval process *effective*.

## Key points

### Setting the scene

- Information retrieval (IR) is all about the user.
    - User has an *information need* - a query
    - IR system needs to find documents that are relevant to that specific users' query
- IR isn't new
    - Index structure (like in the back of a book) developed about 4000 years ago
    - Dewey Decimal classification developed in 1876
        - Still used in libraries today
    - Computers used for IR from the 1960's

### Text Retrieval

#### Overview

- Assume that we have a collection of text *documents*
    - often referred to as a corpus (*plural: corpora*) if they all have something in common (i.e. the "New York Times corpus")
- User wants to find relevant documents by issuing a query against the collection
- This is different to database retrieval!
    - Data is unstructured
    - Query is almost certainly ambiguous with respect to the collection
    - There is potentially no correct answer with respect to relevance (unlike database retrieval where there is a specific algebra that defines the correct answer)
        - The problem is purely empirical; no guarantee for different users issuing the same query that the same documents will be relevant
        - Can only evaluate performance empirically

#### Classical retrieval model

- Boolean model
    - Looks for presence or absence of query terms in document satisfying a Boolean expression (using the operators *and*, *or*, *not*)
    - Only performs *result selection*
        - All documents that match the query considered equally relevant

        - Can't produce a ranking of most-relevant to least relevant
- Vector Space Model
    - Represents both documents and queries as vectors
    - Uses a similarity measure to compare the query against each document
        - Document with closest similarity to query is assumed to be more relevant than a document with a smaller similarity
        - Thus can produce a ranked list of results
- Probabilistic Model
    - Attempts to determine the probability that each document could have been *generated* by the query
    - Documents that are more likely to have been generated by the query more likely to be relevant
- What is better - Result Selection or Result Ranking?
    - Problems of Result Selection:
        - Classifier used for result selection unlikely to be accurate
            - If query is over-constrained (too specific) then zero documents will match
            - If query is under-constrained (too general) then many documents will match
            - Very hard to find the middle-ground
        - Even if the classifier for result selection is accurate, can't assume that all documents are equally relevant
            - Relevance is clearly not a binary attribute
    - ...indicates that results need to be prioritised
        - ranking is needed!
    - Formalised by Robertson's Probability Ranking Principle:
        - 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

**The Vector Space Model (VSM)**

- VSM is a framework of components; the actual details of the construction of those components is not specified
- Two key parts:
    - a query/document representation
        - *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: q=($x_1$,...,$x_N$), $x_j \in \Re$ is query term weight
            - Doc vector: d=($y_1$,...,$y_N$), $y_j \in \Re$ is doc term weight
    - a relevance function
        - relevance(d|q) $\propto$ similarity(q,d) = f(q,d)
- VSM doesn't prescribe:
    - How the *basic concepts* (the terms) are defined
        - Does however assume that they are orthogonal and thus independent

        - How to assign weights to each term and thus place documents/queries into the space
            - Implicit assumption that higher weight means more importance for a term
                - Term weight in a document must measure how important that term is to that document
        - How to design the similarity measure

- Most text-retrieval systems represent the text using a bag of words.
    - The order of the words in the document is disregarded.
    - Each word becomes a term (subject to some modification or removal of certain word)
    - The weight is related to the number of times the word occurs
    - To create a bag of words from a text document, there are a two key processes:
        - Breaking the document into its constituent words (tokenisation);
        - Processing the words to reduce variability in the vocabulary
            - Often the words are processed using techniques like stemming (which removes variations in words like the letters *s* and *ing* at the end of some words).
            - Certain words are also removed (stop word removal) – words like "a", "the", "at", "which", etc., which don't have semantic meaning.
    - On this basis the vectors from the documents contain (potentially weighted) counts of the number of times each word in the lexicon (the set of all possible words) occurs in the document.
        - Essentially the vectors are just histograms of (potentially weighted) word counts.
        - The vector for any given document is highly sparse
            - a document is only likely to contain a very small proportion of all possible words!

- Searching using the vector space model is simple:
    - A query can be turned into a vector form, and all the documents in the system can be ranked by their similarity to the query.
        - Cosine similarity (i.e. the angle between the vectors) is a good starting point, as it is less affected by the vector magnitude (the query vector probably only contains a few words, so has a much lower magnitude (e.g. L1 or L2 norm) compared to the document vectors).
        - Better similarity functions below...
    - Many of the documents will have a similarity of 0 as they don't share any terms with the query.
    - In practice, actual vectors are never created (it would just be too inefficient), and the bag of words is indexed directly in a structure called an inverted index.
        - An inverted index is a map of words to postings lists.
            - A postings list contains postings.
                - A posting is a pair containing a document identifier and word count.
                - Postings are only created if the word count is bigger than 1.
        - Using an inverted index, you can quickly find out which documents a word occurs in, and how many times that word occurs in each of those documents.
        - This allows for really efficient computation of the cosine (and other) similarity measures, as you only need to perform calculations for the words that actually appear in the query, and the documents containing those words.

- Weighting terms
    - Number of occurrences of a term in a document reflects its importance in that document
    - Intuition 1: terms that appear in all documents are not very informative
        - terms should be weighted on the basis of the inverse document frequency
            - (recall the *document frequency* is the number of documents a given word

occurs in)

- Intuition 2: a term that appears frequently in a single document but rarely in other documents is highly informative
  - terms should be weighted on the basis of the term frequency
- This leads to a family of commonly used weighting schemes known as term frequency-inverse document frequency (*TF-IDF*)

- Real scoring functions
  - Build scoring function around cosine similarity to incorporate the TF-IDF weighting
    - Key part of cosine similarity is the dot-product between the vectors (the numerator of the equation)
  - Baseline function looks like:

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

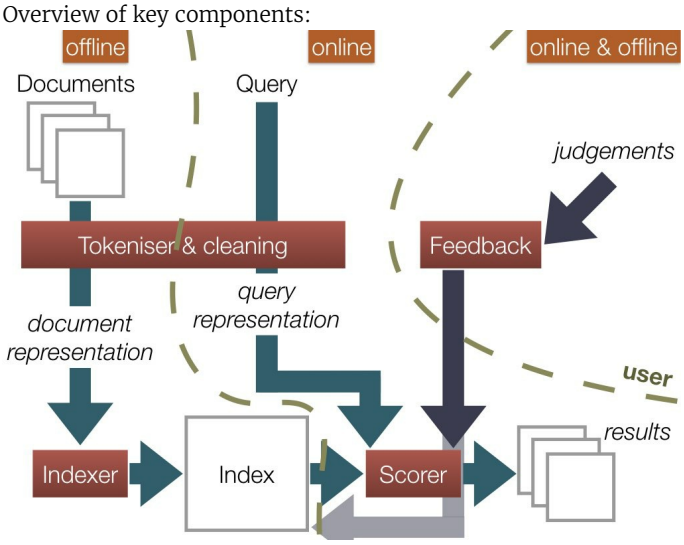
    where $M$ is the number of documents in the collection, $c(w,\cdot)$ is the count of how many times the term $w$ occurs in the query or document, and $df(w)$ is the document frequency of word $w$
    - This has a few problems though:
      - skewed scores with very high TF
      - no normalisation against long documents
  - BM25 is perhaps the most popular family of scoring functions in current usage:

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

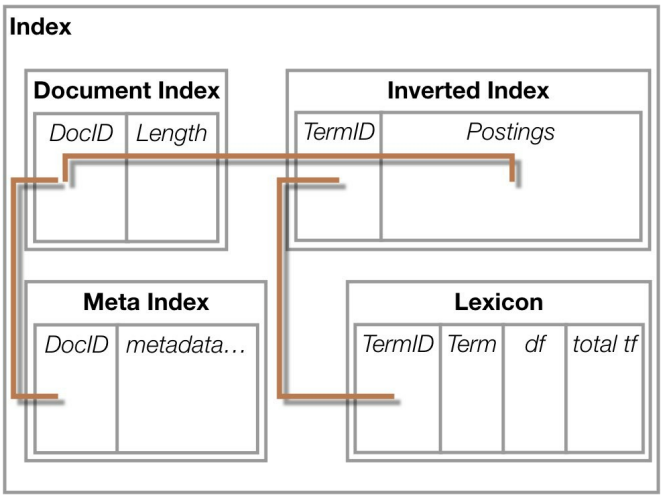
    where $k_1$ and $b$ are constants, $|d|$ is the length of document $d$ and *avgdl* is the average length of documents in the collection

## Inside a retrieval system

- Overview of key components:

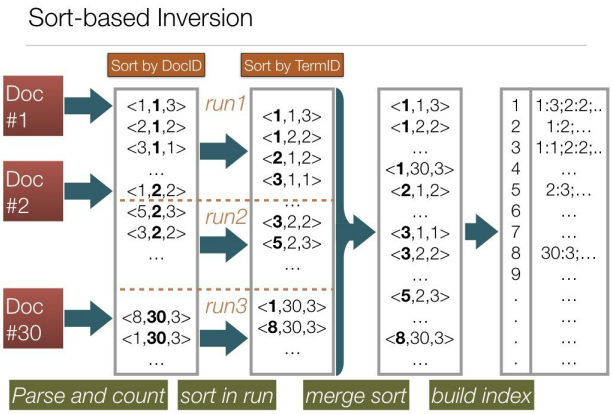



- Inside the index




- Four key components:
  - Inverted index
  - Document index
  - Lexicon
  - Meta index
- Inverted index and Meta index potentially very large
  - usually stored on disk
  - format optimised for fast sequential reads of postings/metadata
- Common for lexicon and document index to be held in memory (if possible)

- Building the inverted index
  - Assume we have 100 million documents that we want to index
    - Potentially terabytes of data
    - Can't extract terms from all of those documents and invert them in memory
  - Must use sort based method
    - Step 1: Collect local tuples in memory (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
  - Runs typically set-up to exhaust memory before sorting and flushing to disk




- Index compression is important for efficiency
  - 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, ɣ-code, δ-code, varint...
    - List-adaptive: Golomb, Rice, Frame of Reference (FOR), Patched FOR (PFOR), ...

## Improving ranking performance

- Utilising term location
    - 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*
        - Allow for *proximity search* (query terms appearing close to each other)
    - Need to efficiently incorporate the location of each term in the index
        - Augment postings with a list of term positions
            - typically in terms of word count from beginning of document
        - Obviously will create a massive increase in size of index
            - Compression is vital:
                - delta-gaps to reduce magnitude of integers being stored
                - ɣ-codes (best compression) or FOR (slightly bigger file size but faster decompression)
    - Modify scoring functions to make use of this extra information
- Retrieving hypertext
    - If the documents contain hyperlinks to other documents, can we make use of the network graph structure to improve ranking?
    - Intuition: documents that are linked to by many other documents are likely to be authoritative
    - Problem: prone to manipulation by pages with many 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.
- Feeding back into the search engine - *click ranking*
    - If we've built a search engine we could look at what result documents users actually look at based on a specific query
    - For each query we capture the users "clicks" on the documents in results list
    - Use this as a basis for improving ranking:
        - 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
    - More generally these ideas fall under the concept of relevance feedback

## Current challenges in IR: Diverse Ranking

- 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

## Further Reading

- Chapter 4 of "Programming Collective Intelligence" gives a good overview of the some of the basic concepts, although misses a lot of key information that you'd need to build an effective and scalable retrieval system
- The most authoritative textbooks are:
    - Modern Information Retrieval. Baeza-Yates & Ribeiro-Neto. Addison-Wesley, Wokingham, UK, 1999. Second edition published in 2011.
    - Managing gigabytes (2nd ed.): compressing and indexing documents and images. Witten, Moffat & Bell. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA. 1999
- For an detailed look at modern index compression:
    - On Inverted Index Compression for Search Engine Efficiency. Catena, Macdonald & Ounis. In: Proceedings of ECIR 2014. http://www.dcs.gla.ac.uk/~craigm/publications/catena14compression.pdf (http://www.dcs.gla.ac.uk/~craigm/publications/catena14compression.pdf)
- For information on ranking:
    - Wikipedia has a reasonable page on BM25 (https://en.wikipedia.org/wiki/Okapi_BM25) ,
    - The Probabilistic Relevance Framework: BM25 and Beyond. Robertson and Zaragoza. http://staff.city.ac.uk/~sb317/papers/foundations_bm25_review.pdf (http://staff.city.ac.uk/~sb317/papers/foundations_bm25_review.pdf)
    - Wikipedia's page on relevance feedback is a good starting point: https://en.wikipedia.org/wiki/Relevance_feedback (https://en.wikipedia.org/wiki/Relevance_feedback)