Search Engine Architecture and Basic Retrieval Models

[DAT640] Information Retrieval and Text Mining

Krisztian Balog
University of Stavanger

August 21, 2024

In this module

1. Introduction to Information Retrieval
2. Search engine architecture
3. Indexing and query processing
4. Retrieval models

Introduction to Information Retrieval

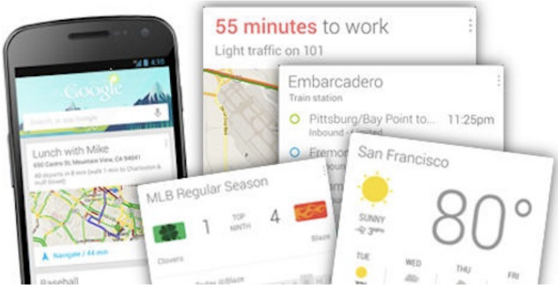
Information Retrieval (IR)

"Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information."

(Salton, 1968)

Modern definition

"Making the **right information** available to the **right person** at the **right time** in **the right form**."



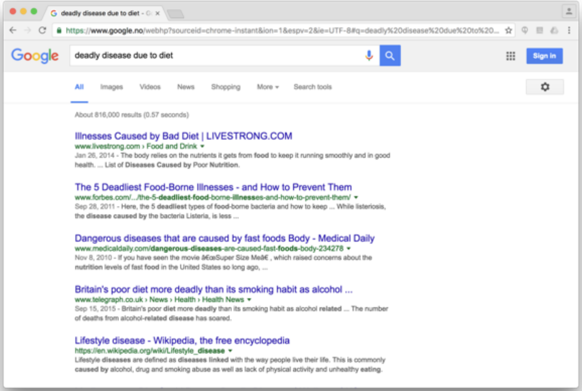
Searching in databases

Query: *records with balance* $>$ *\$50,000 in branches located in Amherst, MA.*

Name	Branch	Balance
Sam I. Am	Amherst, MA	$95,342.11
Patty MacPatty	Amherst, MA	$23,023.23
Bobby de West	Amherst, NY	$78,000.00
Xing O'Boston	Boston, MA	$50,000.01

Searching in text

Query: *deadly disease due to diet*



Which of the results are relevant?

Core problem in IR

How to match information needs (“queries”) and information objects (“documents”)

Core issues in IR

- **Relevance**
  - Simple (and simplistic) definition: A relevant document contains the information that a person was looking for when they submitted a query to the search engine
    - Many factors influence a person's decision about what is relevant (task, context, novelty, ...)
    - Distinction between *topical relevance* vs. *user relevance* (all other factors)
  - *Retrieval models* define a view of relevance
  - *Ranking algorithms* used in search engines are based on retrieval models
  - Most models are based on statistical properties of text rather than linguistic
  - Exact matching of words is not enough!

Core issues in IR

- **Evaluation**
  - Experimental procedures and measures for comparing system output with user expectations
  - Typically use test collection of documents, queries, and relevance judgments
  - *Recall* and *precision* are two examples of effectiveness measures

Core issues in IR

- **Information needs**
  - Keyword queries are often poor descriptions of actual information needs
  - Interaction and context are important for understanding user intent
  - Query modeling techniques such as query expansion, aim to refine the information need and thus improve ranking

Dimensions of IR

- IR is more than just text, and more than just web search
  - Although these are central
- Content
  - Text, images, video, audio, scanned documents, ...
- Applications
  - Web search, vertical search, enterprise search, desktop search, social search, legal search, chatbots and virtual assistants, ...
- Tasks
  - Ad hoc search, filtering, question answering, response ranking, ...

Search engines in operational environments

- Performance
  - Response time, indexing speed, etc.
- Incorporating new data
  - Coverage and freshness
- Scalability
  - Growing with data and users
- Adaptibility
  - Tuning for specific applications

Debate

Do we still need search engines (Google) when we have ChatGPT?

Search engine architecture

Software architecture

- A software architecture consists of software components, the interfaces provided by those components, and the relationships between them
  - Describes a system at a particular level of abstraction

Question

Imagine that you are tasked with creating a search engine for the university library. What would be the main architectural components?

Search engine architecture

- Architecture of a search engine determined by 2 requirements
  - Effectiveness (quality of results)
  - Efficiency (response time and throughput)
- Two main processes:
  - Indexing (offline)
  - Querying (online)

Search engine architecture

- Indexing process
- Query process

Indexing

Indexing is the process that makes a document collection searchable

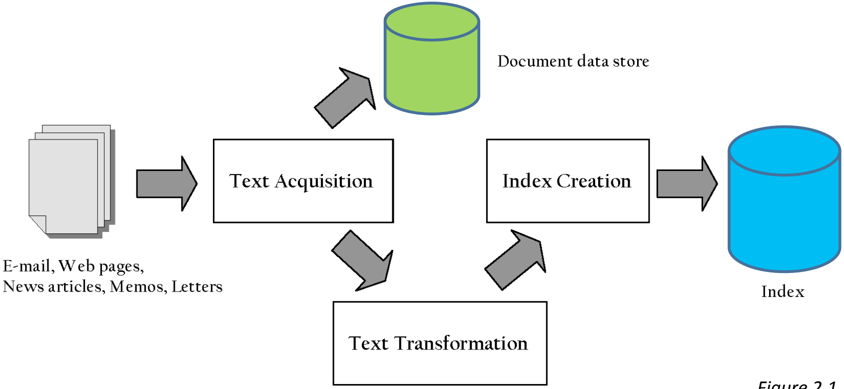


Figure 2.1

Text acquisition

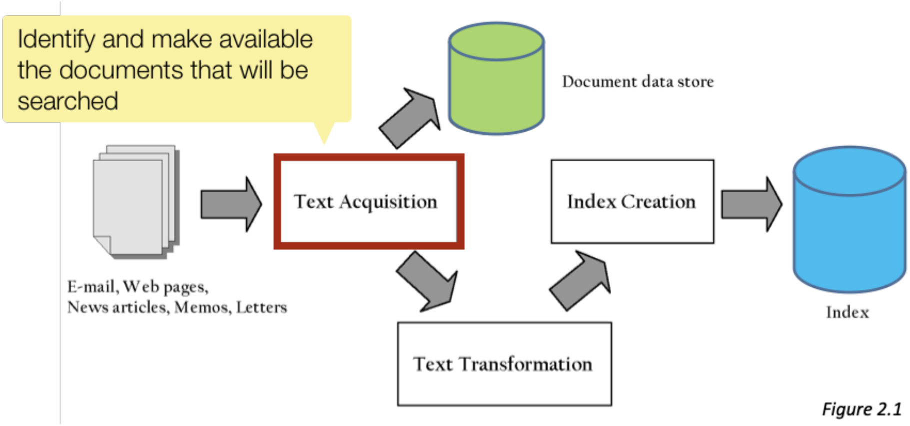


Figure 2.1

Text acquisition

- **Crawler**: identifies and acquires documents for search engine
  - Many types: web, enterprise, desktop, etc.
  - Web crawlers follow links to find documents
    - Must efficiently find huge numbers of web pages (coverage) and keep them up-to-date (freshness)
    - Single site crawlers for site search
    - Topical or focused crawlers for vertical search
  - Document crawlers for enterprise and desktop search
    - Follow links and scan directories
- **Feeds**: real-time streams of documents
  - E.g., web feeds for news, blogs, video, radio, TV
  - RSS is common standard

Document data store

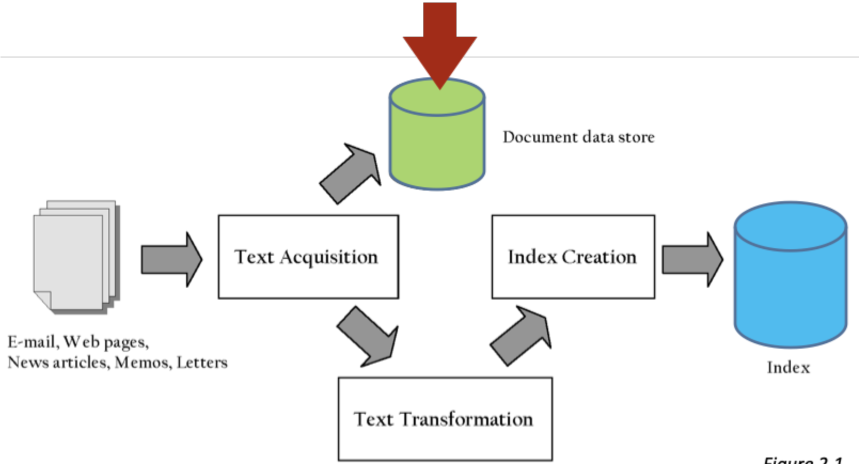


Figure 2.1

Document data store

- Stores text, metadata, and other related content for documents
  - Metadata is information about document such as type and creation date
  - Other content includes links, anchor text
- Provides fast access to document contents for search engine components
  - E.g. result list generation
- Could use relational database system
  - More typically, a simpler, more efficient storage system is used due to huge numbers of documents

Text transformation

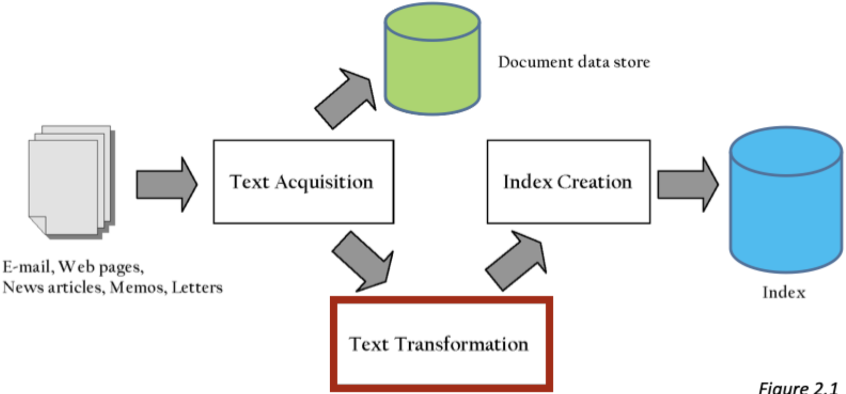


Figure 2.1

Transform documents into index terms or features

Text transformation

- Tokenization, stopword removal, stemming
- Semantic annotation
  - Named entity recognition
  - Text categorization
  - ...
- Link analysis
  - Anchor text extraction
  - ...

Index creation

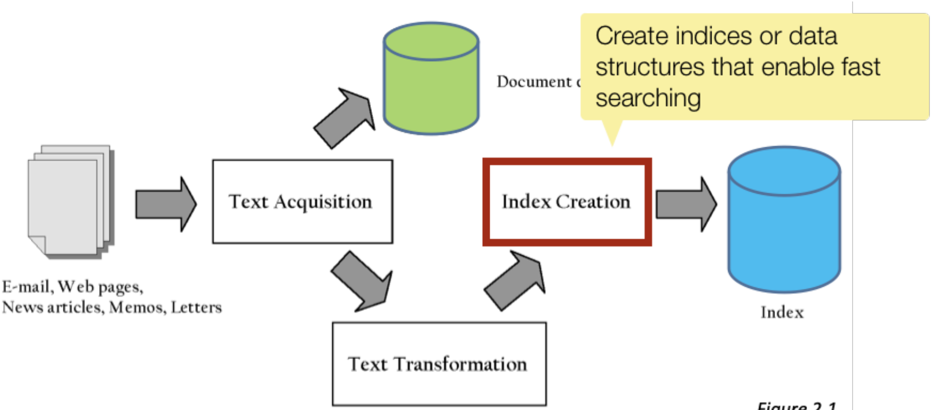


Figure 2.1

Index creation

- Gathers counts and positions of words and other features used in ranking algorithm
- Format is designed for fast query processing
- Index may be distributed across multiple computers and/or multiple sites
- (More in a bit)

Search engine architecture

- Indexing process
- Query process

Query process

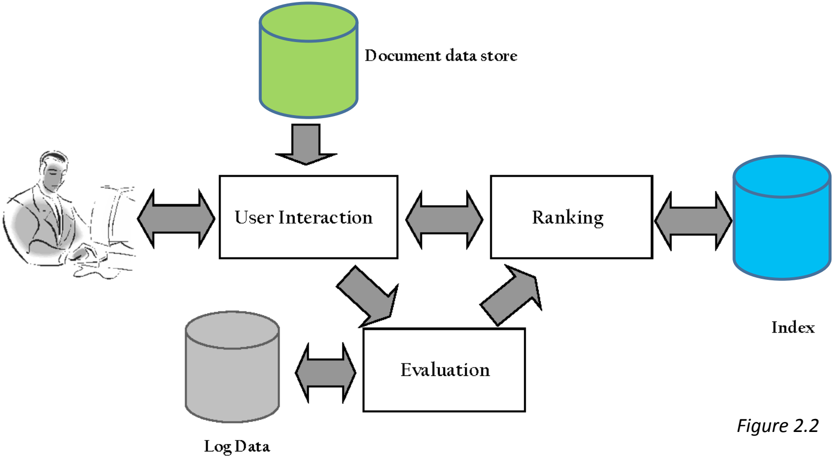


Figure 2.2

User interaction

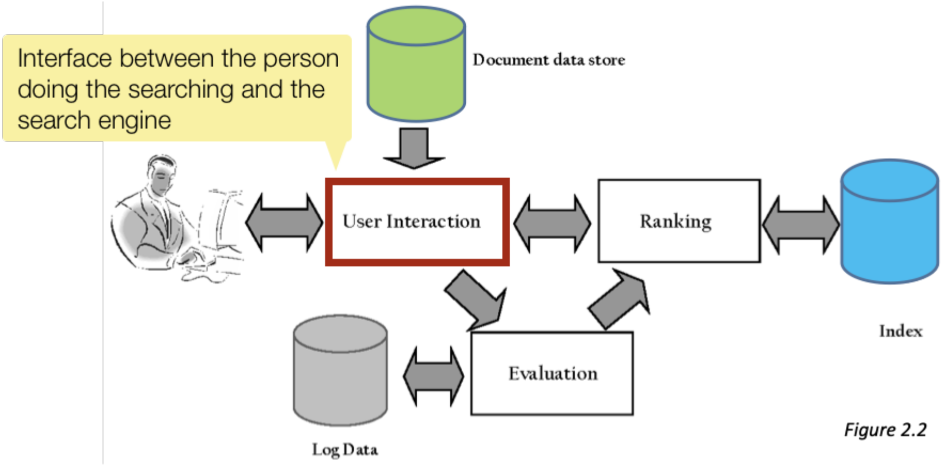


Figure 2.2

User interaction

- **Query input**: accepting the user's query and transforming it into index terms
  - Most web search query languages are very simple (i.e., small number of operators)
  - There are more complicated query languages (proximity operators, structure specification, etc.)
- **Results output**: taking the ranked list of documents from the search engine and organizing it into the results shown to the user
  - Generating *snippets* to show how queries match documents
  - *Highlighting* matching words and passages
  - May provide *clustering* of search results and other visualization tools

Ranking

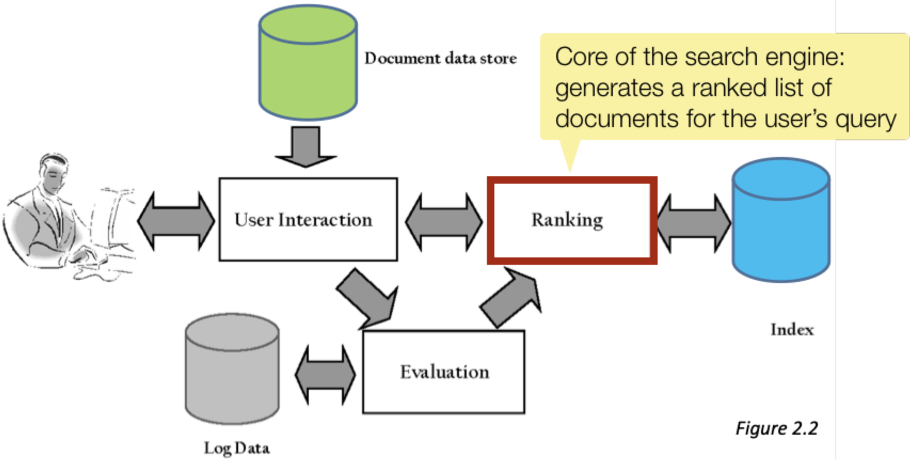


Figure 2.2

Ranking

- Calculates scores for documents using a *ranking algorithm*, which is based on a *retrieval model*
- Core component of search engine
- Many variations of ranking algorithms and retrieval models exist
- **Performance optimization**: designing ranking algorithms for efficient processing
  - *Term-at-a-time* vs. *document-at-a-time* processing
  - *Safe* vs. *unsafe* optimizations
- **Distribution**: processing queries in a distributed environment
  - *Query broker* distributes queries and assembles results

Evaluation

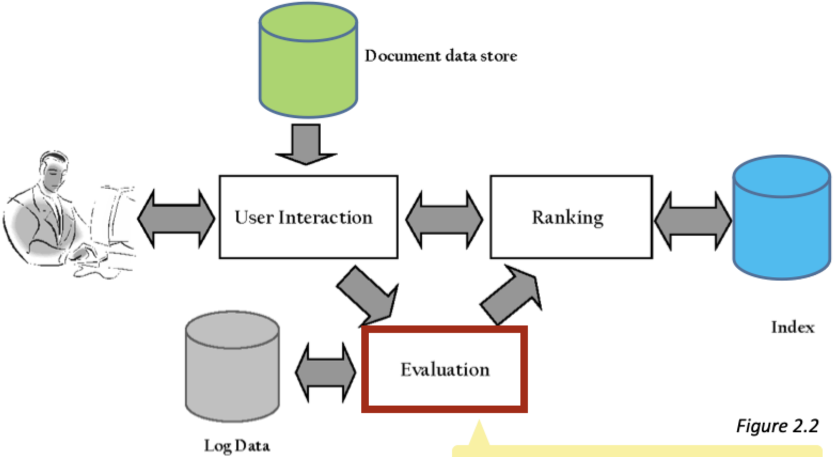


Figure 2.2

Measure and monitor effectiveness and efficiency.
Record and analyze usage data

Evaluation

- **Logging** user queries and interaction is crucial for improving search effectiveness and efficiency
  - *Query logs* and *clickthrough data* used for query suggestion, spell checking, query caching, ranking, advertising search, and other components
- **Ranking analysis**: measuring and tuning ranking effectiveness
- **Performance analysis**: measuring and tuning system efficiency

Indexing and query processing

Indexing and query processing

- Indexing
- Query processing

Indices

- Text search has unique requirements, which leads to unique data structures
- Indices are data structures designed to make search faster
- Most common data structure is the *inverted index*
  - General name for a class of structures
  - "Inverted" because documents are associated with words, rather than words with documents
  - Similar to a concordance

Motivation

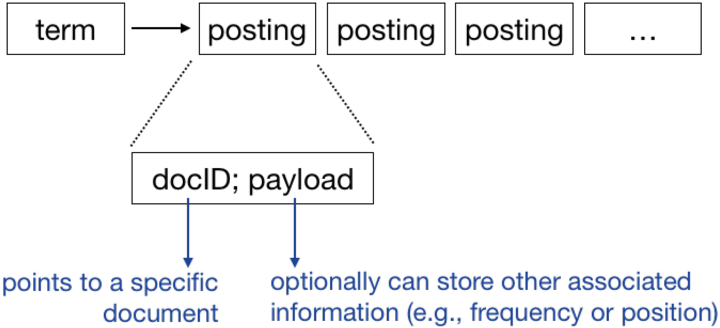
Index

<u>Note</u>: *italic* page numbers indicate specific methods, whilst **bold** page numbers indicate major sections on the subject.

Inverted Index

- Each index term is associated with a *postings list* (or *inverted list*)
    - Contains lists of documents, or lists of word occurrences in documents, and other information
    - Each entry is called a *posting*
    - The part of the posting that refers to a specific document or location is called a *pointer*
        - Each document in the collection is given a unique number (docID)
    - The posting can store additional information, called the *payload*
    - Lists are usually *document-ordered* (sorted by docID)

Postings list



Example

- S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.
- S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.
- S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.
- S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Four sentences from the Wikipedia entry for *tropical fish*

Simple inverted index

Each document that contains the term is a posting. No additional payload.

$\boxed{\text{docID}}$

Term	Postings	Term	Postings
and	1	only	2
aquarium	3	pigmented	4
are	3 4	popular	3
around	1	refer	2
as	2	referred	2
both	1	requiring	2
bright	3	salt	1 4
coloration	3 4	saltwater	2
derives	4	species	1
due	3	term	2
environments	1	the	1 2
fish	1 2 3 4	their	3
fishkeepers	2	this	4
found	1	those	2
fresh	2	to	2 3
freshwater	1 4	tropical	1 2 3
from	4	typically	4
generally	4	use	2
in	1 4	water	1 2 4
include	1	while	4
including	1	with	2
iridescence	4	world	1
marine	2		
often	2 3		

Inverted index with counts

The payload is the frequency of the term in the document.

Supports better ranking algorithms.

docID: freq

and	1:1
aquarium	3:1
are	3:1, 4:1
around	1:1
as	2:1
both	1:1
bright	3:1
coloration	3:1, 4:1
derives	4:1
due	3:1
environments	1:1
fish	1:2, 2:3, 3:2, 4:2
fishkeepers	2:1
found	1:1
fresh	2:1
freshwater	1:1, 4:1
from	4:1
generally	4:1
in	1:1, 4:1
include	1:1
including	1:1
iridescence	4:1
marine	2:1
often	2:1, 3:1
only	2:1
pigmented	4:1
popular	3:1
refer	2:1
referred	2:1
requiring	2:1
salt	1:1, 4:1
saltwater	2:1
species	1:1
term	2:1
the	1:1, 2:1
their	3:1
this	4:1
those	2:1
to	2:2, 3:1
tropical	1:2, 2:2, 3:1
typically	4:1
use	2:1
water	1:1, 2:1, 4:1
while	4:1
with	2:1
world	1:1

Inverted index with term positions

There is a separate posting for each term occurrence in the document. The payload is the term position.

Supports proximity matches. E.g., find "tropical" within 5 words of"fish"

docID. position

and	1,15								
aquarium	3,5								
are	3,3	4,14							
around	1,9								
as	2,21								
both	1,13								
bright	3,11								
coloration	3,12	4,5							
derives	4,7								
due	3,7								
environments	1,8								
fish	1,2	1,4	2,7	2,18	2,23	3,2	3,6	4,3	4,13
fishkeepers	2,1								
found	1,5								
fresh	2,13								
freshwater	1,14	4,2							
from	4,8								
generally	4,15								
in	1,6	4,1							
include	1,3								
including	1,12								
iridescence	4,9								

marine	2,22	
often	2,2	3
only	2,10	
pigmented	4,16	
popular	3,4	
refer	2,9	
referred	2,19	
requiring	2,12	
salt	1,16	4
saltwater	2,16	
species	1,18	
term	2,5	
the	1,10	2
their	3,9	
this	4,4	
those	2,11	
to	2,8	2
tropical	1,1	1
typically	4,6	
use	2,3	
water	1,17	2
while	4,10	
with	2,15	
world	1,11	

Issues

- Compression
  - Inverted lists are very large
  - Compression of indexes saves disk and/or memory space
- Optimization techniques to speed up search
  - Read less data from inverted lists
    - "Skipping" ahead
  - Calculate scores for fewer documents
    - Store highest-scoring documents at the beginning of each inverted list
- Distributed indexing

Example

Create a simple inverted index for the following document collection

Doc 1	new home sales top forecasts
Doc 2	home sales rise in july
Doc 3	increase in home sales in july
Doc 4	july new home sales rise

Solution

new	1	4		
home	1	2	3	4
sales	1	2	3	4
top	1			
forecasts	1			
rise	2	4		
in	2	3		
july	2	3	4	
increase	3			

Exercise

E2-1 Inverted index

Indexing and query processing

- Indexing
- Query processing

Scoring documents

- Objective: estimate the relevance of documents in the collection w.r.t. the input query q (so that the highest-scoring ones can be returned as retrieval results)
- In principle, this would mean scoring all documents in the collection
- In practice, we're only interested in the top-k results for each query
- Common form of a retrieval function

$$score(d, q) = \sum_{t \in q} w_{t,d} \times w_{t,q}$$

  - where $w_{t,d}$ is the weight of term t in document d and $w_{t,q}$ is the weight of that term in the query q

Question

How to compute these retrieval functions for all document in the collection?

Query processing

- Strategies for processing the data in the index for producing query results
  - We benefit from the inverted index by scoring only documents that contain at least one query term
- Term-at-a-time
  - Accumulates scores for documents by processing term lists one at a time
- Document-at-a-time
  - Calculates complete scores for documents by processing all term lists, one document at a time
- Both approaches have optimization techniques that significantly reduce time required to generate scores

Term-at-a-time query processing

$scores = \{\}$ // score accumulator maps doc IDs to scores

for $w \in q$ **do**

  for $d, count \in Idx.fetch\_docs(w)$ **do**

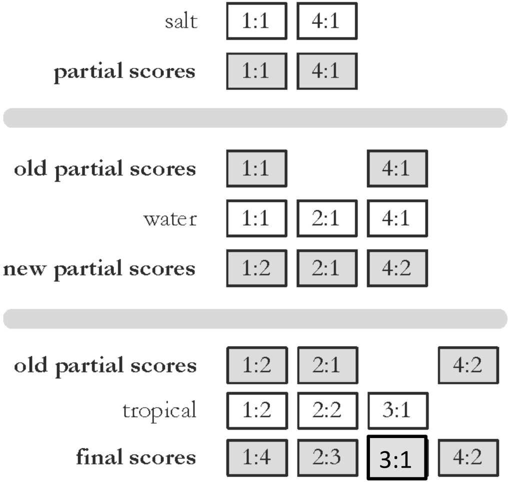
    $scores[d] = scores[d] + score\_term(count)$

  end for

end for

return top k documents from *scores*

Term-at-a-time query processing



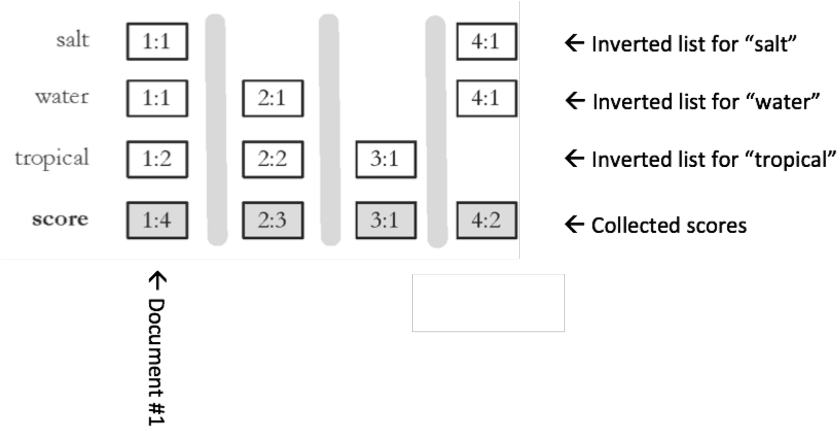
From term-at-a-time to document-at-a-time query processing

- Term-at-a-time query processing
  - Advantage: simple, easy to implement
  - Disadvantage: the score accumulator will be the size of #documents matching at least one query term
- Document-at-a-time query processing
  - Make the score accumulator data structure smaller by scoring entire documents at once. We are typically interested only in top-k results
  - Idea 1: hold the top-k best completely scored documents in a priority queue
  - Idea 2: Documents are sorted by document ID in the posting list. If documents are scored ordered by their IDs, then it is enough to iterate through each query term's posting list only once
    - Keep a pointer for each query term. If the posting equals the document currently being scored, then get the term count and move the pointer; otherwise the current document does not contain the query term

Document-at-a-time query processing

```
context = {}        // maps a document to a list of matching terms
for w ∈ q do
    for d, count ∈ Idx.fetch_docs(w) do
        context[d].append(count)
    end for
end for
priority_queue = {}        // low score is treated as high priority
for d, term_counts ∈ context do
    score = 0
    for count ∈ term_counts do
        score = score + score_term(count)
    end for
    priority_queue.push(d, score)
    if priority_queue.size() > k then
        priority_queue.pop()        // removes lowest score so far
    end if
end for
Return sorted documents from priority_queue
```

Document-at-a-time query processing



Exercise

E2-2 Query processing DaaT

Retrieval models

Retrieval models

- Bag-of-words representation
  - Simplified representation of text as a bag (multiset) of words
  - Disregards word ordering, but keeps multiplicity
- Common form of a retrieval function

$$score(d,q) = \sum_{t \in q} w_{t,d} \times w_{t,q}$$

  - Note: we only consider terms in the query, $t \in q$
  - $w_{t,d}$ is the term's weight in the document
  - $w_{t,q}$ is the term's weight in the query
- $score(d,q)$ is (in principle) to be computed for every document in the collection

Example retrieval functions

- General scoring function

$$score(d,q) = \sum_{t \in q} w_{t,d} \times w_{t,q}$$

- **Example 1**: Count the number of matching query terms in the document

$$w_{t,d} = \begin{cases} 1, & c_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

  - where $c_{t,d}$ is the number of occurrences of term t in document d

$$w_{t,q} = c_{t,q}$$

  - where $c_{t,q}$ is the number of occurrences of term t in query q

Example retrieval functions

- General scoring function

$$score(d,q) = \sum_{t \in q} w_{t,d} \times w_{t,q}$$

- **Example 2**: Instead of using raw term frequencies, assign a weight that reflects the term's importance

$$w_{t,d} = \begin{cases} 1 + \log c_{t,d}, & c_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

  - where $c_{t,d}$ is the number of occurrences of term t in document d

$$w_{t,q} = c_{t,q}$$

  - where $c_{t,q}$ is the number of occurrences of term t in query q

Retrieval models

- Vector space model
- Term weighting
- Vector space models (cont'd)
- Language models
- Summary

Vector space model

- Basis of most IR research in the 1960s and 70s
- Still used
- Provides a simple and intuitively appealing framework for implementing
    - Term weighting
    - Ranking
    - Relevance feedback

Vector space model

- Main underlying assumption: if document d_1 is more similar to the query than another document d_2, then d_1 is *more relevant* than d_2
- Documents and queries are viewed as vectors in a high dimensional space, where each dimension corresponds to a term

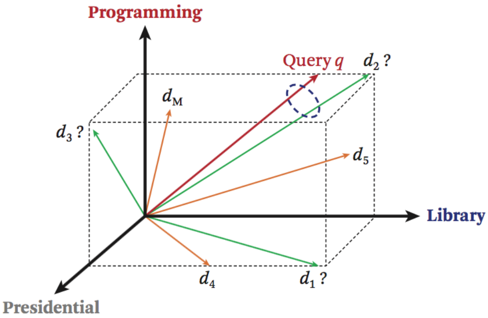


Figure: Illustration is taken from (Zhai&Massung, 2016)[Fig. 6.2]

Instantiation

- The vector space model provides a *framework* that needs to be instantiated by deciding
    - How to select terms? (i.e., vocabulary construction)
    - How to place documents and queries in the vector space (i.e., term weighting)
    - How to measure the similarity between two vectors (i.e., similarity measure)

Simple instantiation (bit vector representation)

- Each word in the vocabulary V defines a dimension
- Bit vector representation of queries and documents (i.e., only term presence/absence)
- Similarity measure is the dot product

$$sim(q,d) = \vec{q} \cdot \vec{d} = \sum_{t \in V} w_{t,q} \times w_{t,d}$$

  - where $w_{t,q}$ and $w_{t,d}$ are either 0 or 1

Question

What are potential shortcomings of this simple instantiation?

Retrieval models

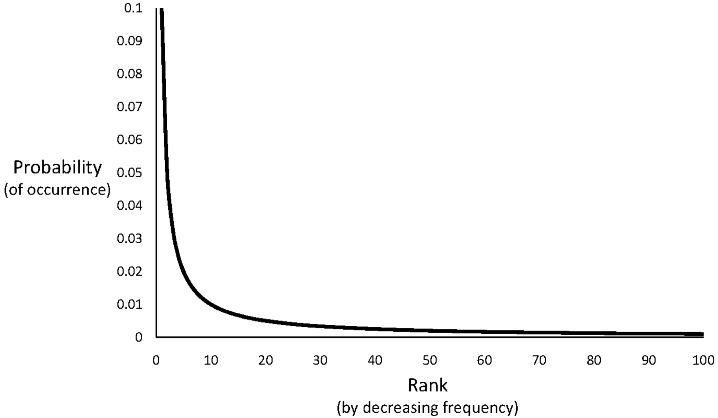
- Vector space model
- Term weighting
- Vector space models (cont'd)
- Language models
- Summary

English language

- Most frequent words
  - the (7%)
  - of (3.5%)
  - and (2.8%)
- Top 135 most frequent words account for half of the words used

Zip's law

- Given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table
  - Word number n has a frequency proportional to $1/n$



Term weighting

- Intuition 1: terms that appear often in a document should get high weights
  - E.g., The more often a document contains the term "dog," the more likely that the document is "about" dogs
- Intuition 2: terms that appear in many documents should get low weights
  - E.g., stopwords, like "a," "the," "this," etc.
- How do we capture this mathematically?
  - Term frequency
  - Inverse document frequency

Term frequency (TF)

- We write $c_{t,d}$ for the raw count of a term in a document
- **Term frequency** $tf_{t,d}$ reflects the importance of a term (t) in a document (d)
- Variants
  - Binary: $tf_{t,d} \in \{0, 1\}$
  - Raw count: $tf_{t,d} = c_{t,d}$
  - **L1-normalized**: $tf_{t,d} = \frac{c_{t,d}}{|d|}$
    - where $|d|$ is the length of the document, i.e., the sum of all term counts in d: $|d| = \sum_{t \in d} c_{t,d}$
  - L2-normalized: $tf_{t,d} = \frac{c_{t,d}}{||d||}$
    - where $||d|| = \sqrt{\sum_{t \in d} (c_{t,d})^2}$
  - Log-normalized: $tf_{t,d} = 1 + \log c_{t,d}$
  - ...
- By default, when we refer to TF we will mean the L1-normalized version

Inverse document frequency (IDF)

- **Inverse document frequency** idf_t reflects the importance of a term (t) in a collection of documents
  - The more documents that a term occurs in, the less discriminating the term is between documents, consequently, the less "useful"

$$idf_t = \log \frac{N}{n_t}$$

  - where N is the total number of documents in the collection and n_t is the number of documents that contain t
  - Log is used to "dampen" the effect of IDF

IDF illustration[1]

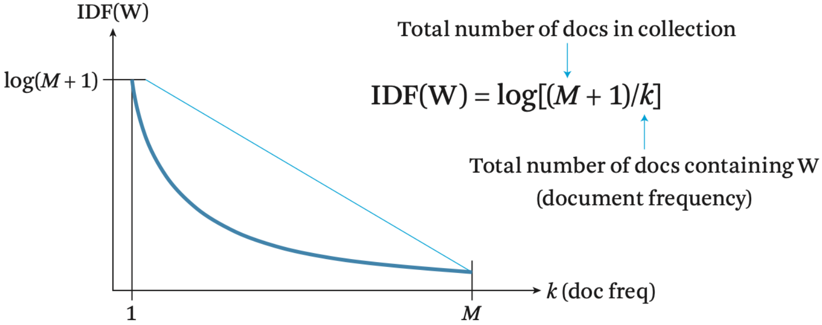


Figure: Illustration of the IDF function as the document frequency varies. Figure is taken from (Zhai&Massung, 2016)[Fig. 6.10]

[1] Note that the textbook uses a slightly different IDF formula with $+1$ in the numerator.

Term weighting (TF-IDF)

- Combine TF and IDF weights by multiplying them:

$$\textit{tfidf}_{t,d} = tf_{t,d} \cdot idf_t$$

  - Term frequency weight measures importance in document
  - Inverse document frequency measures importance in collection

Exercise

E2-3 Document-term matrix

Exercise

E2-4 TFIDF weighting

Retrieval models

- Vector space model
- Term weighting
- Vector space models (cont'd)
- Language models
- Summary

Improved instantiation (TF-IDF weighting)

- Idea: incorporate term importance by considering term frequency (TF) and inverse document frequency (IDF)
  - TF rewards terms that occur frequently in the document
  - IDF rewards terms that do not occur in many documents
- A possible ranking function using the TF-IDF weighting scheme:

$$score(d,q) = \sum_{t \in q \cap d} tf_{t,q} \times tf_{t,d} \times idf_t$$

- Note: the above formula uses raw term frequencies and applies IDF only on one of the (document/query) vectors

Many different variants out there!

- Different variants of TF and IDF
- Different TF-IDF weighting for the query and for the document
- Different similarity measure (e.g., cosine)

BM25

- BM25 was created as the result of a series of experiments (“Best Match”)
- Popular and effective ranking algorithm
- The reasoning behind BM25 is that good term weighting is based on three principles
  - Term frequency
  - Inverse document frequency
  - Document length normalization

BM25 scoring

$$score(d,q) = \sum_{t \in q} \frac{c_{t,d} \times (1 + k_1)}{c_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})} \times idf_t$$

- Parameters
  - k_1: calibrating term frequency scaling
  - b: document length normalization
- Note: several slight variations of BM25 exist!

Recall: TF transformation

- Many different ways to transform raw term frequency counts

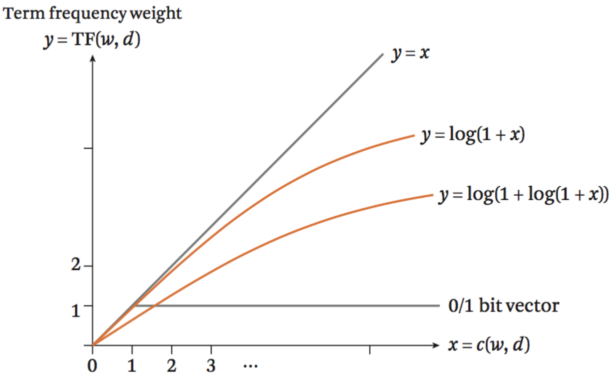


Figure: Illustration is taken from (Zhai&Massung, 2016)[Fig. 6.14]

BM25 TF transformation

- Idea: term saturation, i.e., repetition is less important after a while

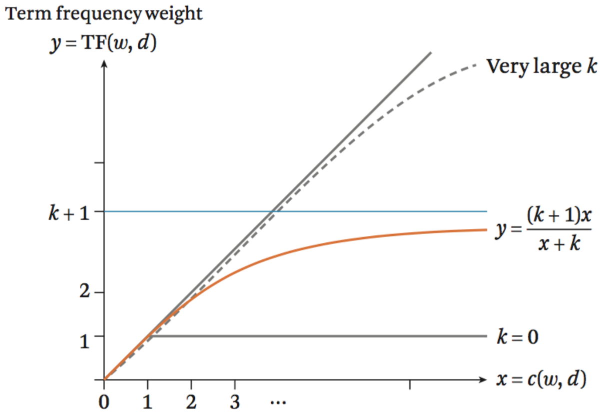


Figure: Illustration is taken from (Zhai&Massung, 2016)[Fig. 6.15]

BM25 document length normalization

- Idea: penalize long documents w.r.t. average document length (which serves as pivot)

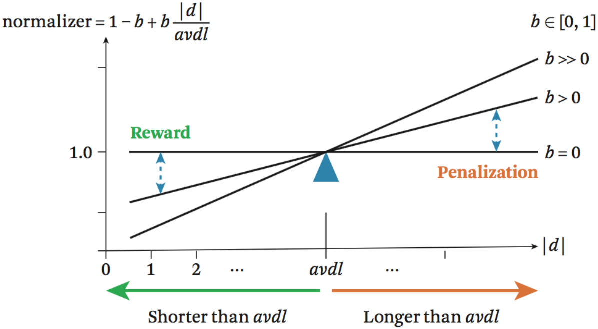


Figure: Illustration is taken from (Zhai&Massung, 2016)[Fig. 6.17]

BM25 parameter setting

- k_1: calibrating term frequency scaling
  - 0 corresponds to a binary model
  - large values correspond to using raw term frequencies
  - typical values are between 1.2 and 2.0; a common default value is 1.2
- b: document length normalization
  - 0: no normalization at all
  - 1: full length normalization
  - typical value: 0.75

Retrieval models

- Vector space model
- Term weighting
- Vector space models (cont'd)
- Language models
- Summary

Language models

- Based on the notion of probabilities and processes for generating text
- Wide range of usage across different applications
  - Speech recognition
    - "I ate a cherry" is a more likely sentence than "Eye eight uh Jerry"
  - OCR and handwriting recognition
    - More probable sentences are more likely correct readings
  - Machine translation
    - More likely sentences are probably better translations

Language models for ranking documents

- Represent each document as a multinomial probability distribution over terms
- Estimate the probability that the query was "generated" by the given document
  - How likely is the search query given the language model of the document?

Query likelihood retrieval model

- Rank documents d according to their likelihood of being relevant given a query q:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \propto P(q|d)P(d)$$

- **Query likelihood**: Probability that query q was "produced" by document d

$$P(q|d) = \prod_{t \in q} P(t|\theta_d)^{c_{t,q}}$$

- **Document prior**, $P(d)$: Probability of the document being relevant to *any* query

Query likelihood

$$P(q|d) = \prod_{t \in q} P(t|\theta_d)^{c_{t,q}}$$

- θ_d is the document language model
  - Multinomial probability distribution over the vocabulary of terms
- $c_{t,q}$ is the raw frequency of term t in the query
- **Smoothing**: ensuring that $P(t|\theta_d)$ is > 0 for all terms

Jelinek-Mercer smoothing

- Linear interpolation between the empirical document model and a collection (background) language model

$$P(t|\theta_d) = (1 - \lambda)P(t|d) + \lambda P(t|C)$$

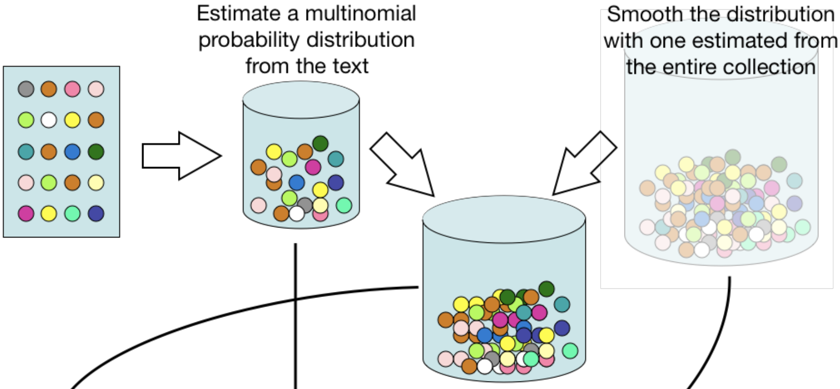
  - $\lambda \in [0, 1]$ is the smoothing parameter
  - Empirical document model (maximum likelihood estimate):

$$P(t|d) = \frac{c_{t,d}}{|d|}$$

  - Collection (background) language model (maximum likelihood estimate):

$$P(t|C) = \frac{\sum_{d'} c_{t,d'}}{\sum_{d'} |d'|}$$

Jelinek-Mercer smoothing



$$\overbrace{P(t|\theta_d)} = (1-\lambda)\overbrace{P(t|d)} + \lambda\underbrace{P(t|C)}$$

Dirichlet smoothing

- Smoothing is inversely proportional to the document length

$$P(t|\theta_d) = \frac{c_{t,d} + \mu P(t|C)}{|d| + \mu}$$

  - μ is the smoothing parameter (typically ranges from 10 to 10000)
- Notice that Dirichlet smoothing may also be viewed as a linear interpolation in the style of Jelinek-Mercer smoothing, by setting

$$\lambda = \frac{\mu}{|d| + \mu} \qquad (1 - \lambda) = \frac{|d|}{|d| + \mu}$$

Query likelihood scoring (Example)

- query: "sea submarine"

$$\begin{aligned} P(q|d) &= P(\text{sea}|\theta_d) \times P(\text{submarine}|\theta_d) \\ &= \big((1-\lambda)P(\text{sea}|d) + \lambda P(\text{sea}|C)\big) \\ &\quad \times \big((1-\lambda)P(\text{submarine}|d) + \lambda P(\text{submarine}|C)\big) \end{aligned}$$

- where
  - $P(\text{sea}|d)$ is the relative frequency of term "sea" in document d
  - $P(\text{sea}|C)$ is the relative frequency of term "sea" in the entire collection
  - ...

Practical considerations

- Since we are multiplying small probabilities, it is better to perform computations in the log space

$$
\begin{aligned}
P(q|d) &= \prod_{t \in q} P(t|\theta_d)^{c_{t,q}} \\
&\Downarrow \\
\log P(q|d) &= \sum_{t \in q} c_{t,q} \times \log P(t|\theta_d)
\end{aligned}
$$

- Notice that it is a particular instantiation of our general scoring function $score(d,q) = \sum_{t \in q} w_{t,d} \times w_{t,q}$ by setting
  - $w_{t,d} = \log P(t|\theta_d)$
  - $w_{t,q} = c_{t,q}$

Retrieval models

- Vector space model
- Term weighting
- Vector space models (cont'd)
- Language models
- Summary

BM25

- Retrieval model is based on the idea of query-document similarity. Three main components:
  - Term frequency
  - Inverse document frequency
  - Document length normalization
- Retrieval function

$$score(d, q) = \sum_{t \in q} \frac{c_{t,d} \times (1 + k_1)}{c_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})} \times idf_t$$

  - Parameters
    - k_1: calibrating term frequency scaling ($k_1 \in [1.2..2]$)
    - b: document length normalization ($b \in [0, 1]$)

Language models

- Retrieval model is based on the probability of observing the query given that document
- Log query likelihood scoring

$$score(d,q) = \log P(q|d) = \sum_{t \in q} \log P(t|\theta_d) \times c_{t,q}$$

- Jelinek-Mercer smoothing

$$score(d,q) = \sum_{t \in q} \log \left((1-\lambda)\frac{c_{t,d}}{|d|} + \lambda P(t|C) \right) \times c_{t,q}$$

- Dirichlet smoothing

$$score(d,q) = \sum_{t \in q} \log \frac{c_{t,d} + \mu P(t|C)}{|d| + \mu} \times c_{t,q}$$

Question

What other statistics are needed to compute these retrieval functions, in addition to term frequencies ($c_{t,d}$)?

BM25

- Total number of documents in the collection (for IDF computation) (int)
- Document length for each document (dictionary)
- Average document length in the collection (int)
- (optionally pre-computed) IDF score for each term (dictionary)

Language models

- Document length for each document (dictionary)
- Sum TF for each term (dictionary)
- Sum of all document lengths in the collection (int)
- (optionally pre-computed) Collection term probability $P(t|C)$ for each term (dictionary)

Exercise

E2-5 Vector space retrieval

Summary

- The problem of information retrieval and core issues (information needs, relevance, evaluation)
- Main components of search engines; indexing and querying processes
- Inverted index (posting list, posting)
- Index creation (without payload, with counts, with position information)
- Query processing (term-at-a-time and document-at-a-time scoring)
- General scoring function
- Vector space model, TF-IDF and BM25 retrieval models
- Language models (query likelihood scoring, different smoothing methods)

Reading

- Text Data Management and Analysis (Zhai&Massung)
  - Chapter 5: Sections 5.3, 5.4
  - Chapter 6
  - Chapter 8: Sections 8.2, 8.3 (optionally, 8.5, 8.6)
  - Chapter 10: Section 10.1 (optionally, Section 10.2)