Queries and Indexes

CISC489/689-010, Lecture #7
Wednesday, March 4
Ben Carterette

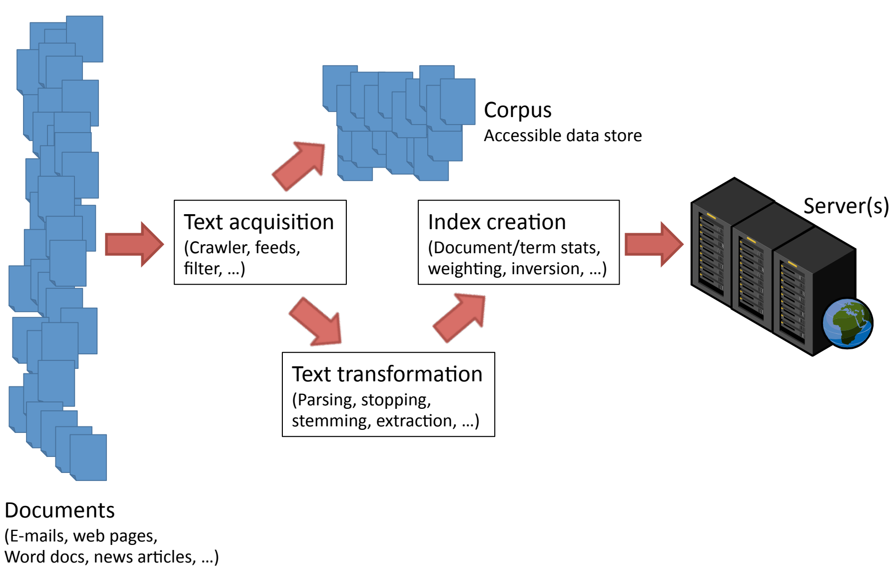
Project Notes

- Next worksheet:
  - Inverted lists for terms in the wiki000 documents.
  - For each term, store:
    - The list of document numbers it occurs in.
    - The term frequencies in those documents.
    - The document frequency (total number of documents it occurs in).
  - If inclined, you may store other information:
    - Term positions, field information, etc.

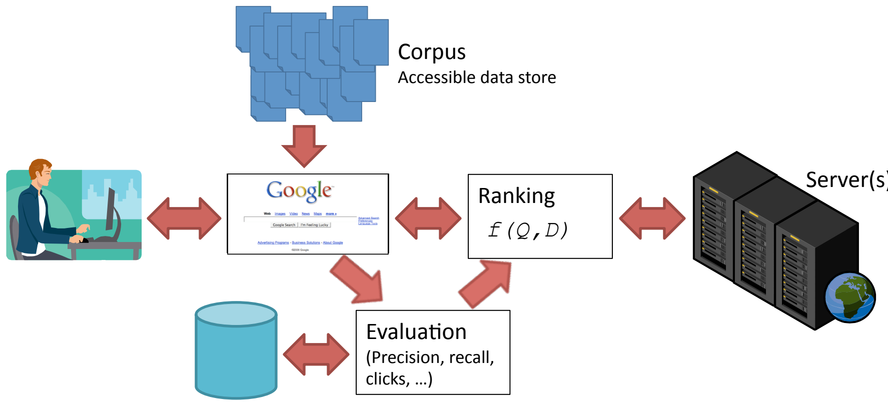
Project Notes

- Inverted list compression:
  - You should compress the inverted lists.
  - Use d-gaps for document numbers.
  - Compress integers using one of the methods discussed in class.
- Store everything in memory.
  - Writing to disk will be the next part of the project.
  - I strongly recommend using ir.cis to run your code.
    - It has a total of 128Gb of RAM (8 nodes, 16Gb per node).

Indexing Process



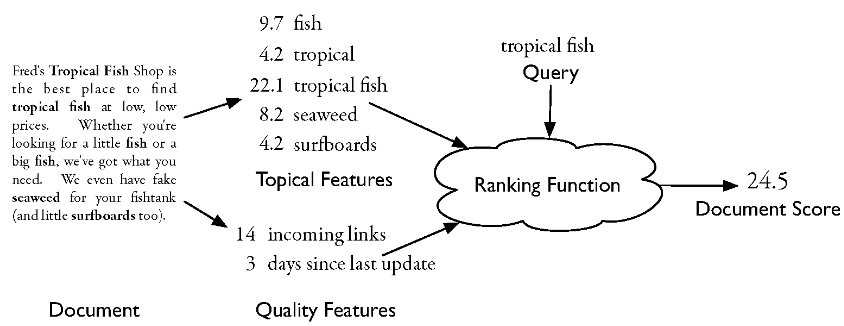
Query Process



Query Process

- We have an index stored on disk.
  - Inverted file, vocabulary, collection.
  - Contains features of terms and documents:
    - Term frequencies in documents, document frequencies, term positions, link-graph features, …
- User inputs a query.
- Engine computes features of the query.
- Engine accesses index to respond to query.
  - Matches query features to document/term features in index to score each document.
- Returns a ranked list of documents.

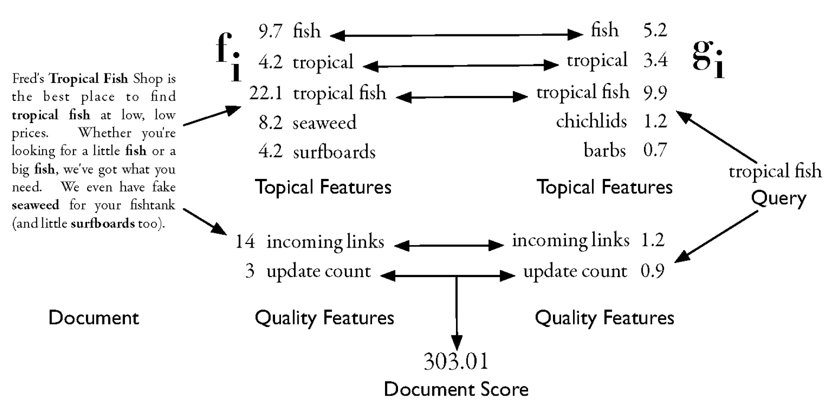
Abstract Model of Ranking



More Concrete Model

$$R(Q,D) = \sum_i g_i(Q) f_i(D)$$

f_i is a document feature function
g_i is a query feature function



Example "Collection"

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*

Query:
tropical fish

pigmented fish

saltwater species bright coloration

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		

$$R(Q, D) = \sum_i g_i(Q) f_i(D)$$

$g_i(Q)$ = # of occurrences of i in Q

$f_i(D)$ = # of occurrences of i in D

Query:
tropical fish

pigmented fish

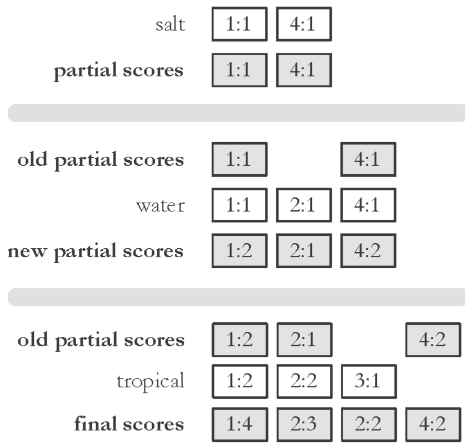
saltwater species bright coloration

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			

Query Processing

- 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

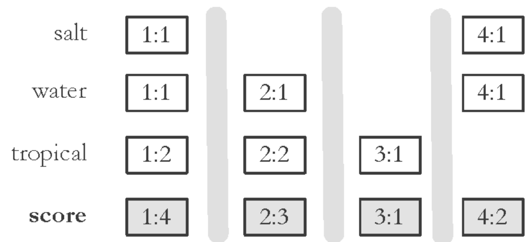
Term-At-A-Time



Term-At-A-Time

procedure TermAtATimeRetrieval(Q, I, f, g k)
  $A \leftarrow$ HashTable()
  $L \leftarrow$ Array()
  $R \leftarrow$ PriorityQueue(k)
  for all terms w_i in Q **do**
    $l_i \leftarrow$ InvertedList(w_i, I)
    L.add(l_i)
  end for
  for all lists $l_i \in L$ **do**
    while l_i is not finished **do**
      $d \leftarrow l_i$.getCurrentDocument()
      $A_d \leftarrow A_d + g_i(Q)f(l_i)$
      l_i.moveToNextDocument()
    end while
  end for
  for all accumulators A_d in A **do**
    $s_D \leftarrow A_d$ ▷ Accumulator contains the document score
    R.add(s_D, D)
  end for
  return the top k results from R
end procedure

Document-At-A-Time



Document-At-A-Time

```
procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    for all documents d ∈ I do
        for all inverted lists l_i in L do
            if l_i points to d then
                s_D ← s_D + g_i(Q)f_i(l_i)          ▷ Update the document score
                l_i.movePastDocument( d )
            end if
        end for
        R.add( s_D, D )
    end for
    return the top k results from R
end procedure
```

Optimization Techniques

- Inverted lists can be very long
  - Decompression time + processing time can add up fast
- Optimizations are used to speed up processing time
- Two classes of optimization
  - Read less data from inverted lists
    - e.g., skip lists
    - better for simple feature functions
  - Calculate scores for fewer documents
    - e.g., conjunctive processing
    - better for complex feature functions

Conjunctive Term-at-a-Time

```
procedure TermAtATimeRetrieval(Q, I, f, g, k)
    A ← HashTable()
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    for all lists l_i ∈ L do
        while l_i is not finished do
            if i = 0 then
                d ← l_i.getCurrentDocument()
                A_d ← A_d + g_i(Q)f(l_i)
            else
                d ← l_i.getCurrentDocument()
                d ← A.getNextDocumentAfter(d)
                l_i.skipForwardTo(d)
                if l_i.getCurrentDocument() = d then
                    A_d ← A_d + g_i(Q)f(l_i)
                else
                    A.remove(d)
                end if
            end if
        end while
    end for
    for all accumulators A_d in A do
        s_D ← A_d                 ▷ Accumulator contains the document score
        R.add( s_D, D )
    end for
    return the top k results from R
end procedure
```

Conjunctive Document-at-a-Time

```
procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    while all lists in L are not finished do
        for all inverted lists l_i in L do
            if l_i.getCurrentDocument() > d then
                d ← l_i.getCurrentDocument()
            end if
        end for
        for all inverted lists l_i in L do l_i.skipForwardToDocument(d)
            if l_i points to d then
                s_d ← s_d + g_i(Q)f_i(l_i)        ▷ Update the document score
                l_i.movePastDocument( d )
            else
                break
            end if
        end for
        R.add( s_d, d )
    end while
    return the top k results from R
end procedure
```

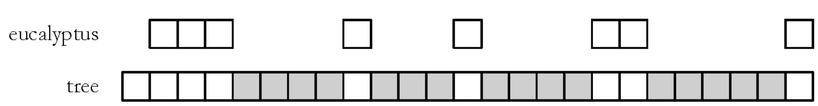
Threshold Methods

- Threshold methods use number of top-ranked documents needed (*k*) to optimize query processing
  - for most applications, *k* is small
- For any query, there is a *minimum score* that each document needs to reach before it can be shown to the user
  - score of the *k*th-highest scoring document
  - gives *threshold score* τ
  - optimization methods estimate τ' to ignore documents

Threshold Methods

- For document-at-a-time processing, use score of lowest-ranked document so far for τ'
  - for term-at-a-time, have to use k_{th}-largest score in the accumulator table
- *MaxScore* method compares the maximum score that remaining documents could have to τ'
  - *safe* optimization in that ranking will be the same without optimization

MaxScore Example



- Indexer computes μ_{tree}
  - maximum score for any document containing just “tree”
- Assume k =3, τ' is lowest score after first three docs
- Likely that $\tau' > \mu_{tree}$
  - τ' is the score of a document that contains both query terms
- Can safely skip over all gray postings

Other Approaches

- Early termination of query processing
  - ignore high-frequency word lists in term-at-a-time
  - ignore documents at end of lists in doc-at-a-time
  - *unsafe* optimization
- List ordering
  - order inverted lists by quality metric (e.g., PageRank) or by partial score
  - makes unsafe (and fast) optimizations more likely to produce good documents

Review

- Query processing:
  - Document-at-a-time
  - Term-at-a-time
  - Optimizations:
    - Conjunctive processing
    - Thresholding
- How do you define f_i and g_i in the scoring function?
  - What is the actual *goal*?

Information Needs

- An *information need* is the underlying cause of the query that a person submits to a search engine
  - sometimes called *information problem* to emphasize that information need is generally related to a task
- Categorized using variety of dimensions
  - e.g., number of relevant documents being sought
  - type of information that is needed
  - type of task that led to the requirement for information

Queries and Information Needs

- A query can represent very different information needs
  - May require different search techniques and ranking algorithms to produce the best rankings
- A query can be a poor representation of the information need
  - User may find it difficult to express the information need
  - User is encouraged to enter short queries both by the search engine interface, and by the fact that long queries don't work

Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of many ranking algorithms
  - can be implicit
- Theories about relevance

Relevance

- Complex concept that has been studied for some time
  - Many factors to consider
  - People often disagree when making relevance judgments
- Retrieval models make various assumptions about relevance to simplify problem
  - e.g., *topical* vs. *user* relevance
  - e.g., *binary* vs. *multi-valued* relevance

Retrieval Model Overview

- Older models
  - Boolean retrieval
  - Vector Space model
- Probabilistic Models
  - BM25
  - Language models
- Combining evidence
  - Inference networks
  - Learning to Rank

Boolean Retrieval

- Two possible outcomes for query processing
  - TRUE and FALSE
  - “exact-match” retrieval
  - simplest form of ranking
- Query usually specified using Boolean operators
  - AND, OR, NOT
  - proximity operators also used

Boolean Retrieval

- Advantages
  - Results are predictable, relatively easy to explain
  - Many different features can be incorporated
  - Efficient processing since many documents can be eliminated from search
- Disadvantages
  - Effectiveness depends entirely on user
  - Simple queries usually don’t work well
  - Complex queries are difficult

Searching by Numbers

- Sequence of queries driven by number of retrieved documents
  - e.g. “lincoln” search of news articles
  - president AND lincoln
  - president AND lincoln AND NOT (automobile OR car)
  - president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
  - president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

Vector Space Model

- Documents and queries represented as vectors in V-dimensional space.
  - Vector coefficients are term weights.

$$D_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1,V} \end{bmatrix}$$
$$D_2 = \begin{bmatrix} w_{21} & w_{22} & w_{23} & \dots & w_{2,V} \end{bmatrix}$$
$$\dots$$
$$D_N = \begin{bmatrix} w_{N,1} & w_{N,2} & w_{N,3} & \dots & w_{N,V} \end{bmatrix}$$
$$Q = \begin{bmatrix} w_{Q,1} & w_{Q,2} & w_{Q,3} & \dots & w_{Q,V} \end{bmatrix}$$

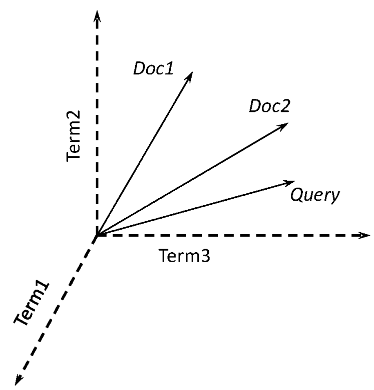
Vector Space Model

D_1 Tropical Freshwater Aquarium Fish.
D_2 Tropical Fish, Aquarium Care, Tank Setup.
D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	**Documents**			
	D_1	**D_2**	**D_3**	**D_4**
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

Vector Space Model

- Visualization:



Term Weighting

- What features are useful in term weights?
- Term frequency (tf):
  - term occurs often in a document → document more likely to be relevant.
- Inverse document frequency (idf):
  - term appears in many documents → less discriminating → documents it appears in less likely to be relevant.
- Document length:
  - Very long document → each term occurrence less important → document less likely to be relevant.
- How can we combine these into a weight w_{ik}?

Term Weighting

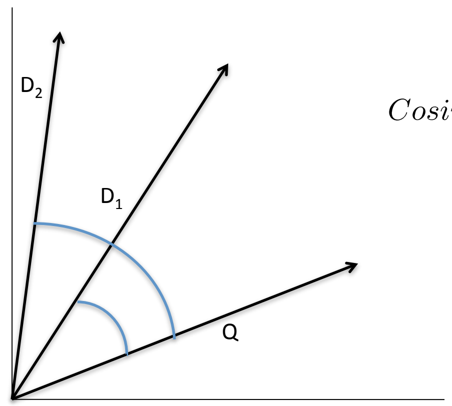
- There are many different ways to weight terms.
- tf-idf weighting is one of the most common.
  - Term frequency of term k in document i: $tf_{ik} = \frac{f_{ik}}{l_i}$
  - Inverse document frequency of term k: $idf_k = \log \frac{N}{n_k}$
  - Weight of term k in document i = tf*idf: $d_{ik} = \frac{f_{ik}}{l_i} \log \frac{N}{n_k}$

Vector Space Model

- Documents ranked by distance between vectors representing query and documents
  - e.g. Cosine correlation

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

Similarity



$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

Similarity Calculation

– Consider two documents D_1, D_2 and a query Q

- D_1 = (0.5, 0.8, 0.3), D_2 = (0.9, 0.4, 0.2), Q = (1.5, 1.0, 0)

$$\begin{aligned} Cosine(D_1, Q) &= \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}} \\ &= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87 \end{aligned}$$

$$\begin{aligned} Cosine(D_2, Q) &= \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}} \\ &= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97 \end{aligned}$$