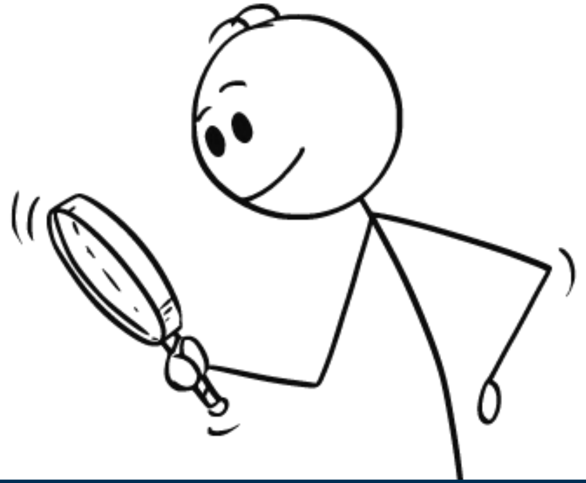


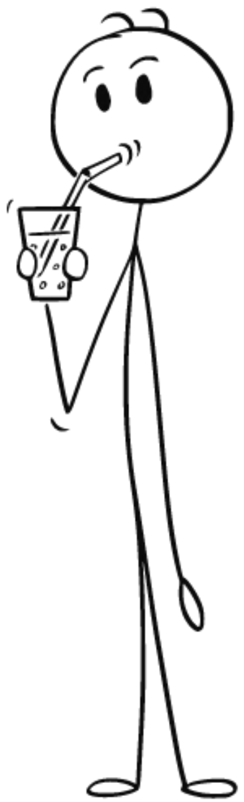
CSE 4/535 Information Retrieval

Sayantan Pal
PhD Student, Department of CSE
338Z Davis Hall

University at Buffalo
Department of CSE

Before we start

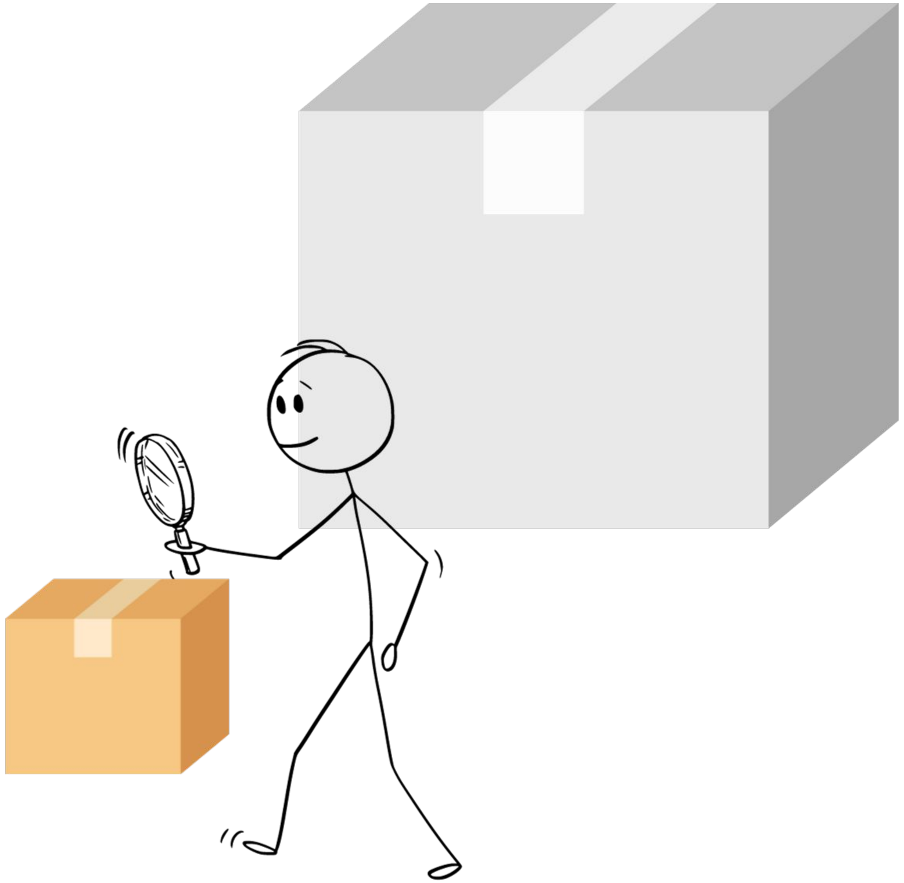
1. Project 1 due 29th September, 11:59 PM (Hope it helps)
2. Join office hours if you have questions (Thursday 8-10 AM)
3. Today's lecture
    a. Efficient Scoring in a Complete Search System
    b. Speeding up vector space ranking
4. Upto Today’s lecture - Syllabus for Mid Term
5. Last 10 mins of class - Mid Term Discussion



Recap - Previous Class

1. Term Frequency (TF)
2. Inverse Document Frequency (IDF)
3. TF-IDF score
4. VSM

Restricting the Search space



Parametric search

- Most documents have, in addition to text, some "meta-data" in fields e.g.,
  - Language = French
  - Field → Format = pdf ← Value
  - Subject = Physics etc.
  - Date = Feb 2000
- A parametric search interface allows the user to combine a full-text query with selections on these field values e.g.,
  - language, date range, etc.

Parametric search example

CarFinder.com

Over one million fictional vehicles to choose from!

We can add text search.

Choose your search criteria from the drop down menus:

Number of results to display: 50

Make BMW **Model** 5-Series **Category** Any **Year** 1997

City San Francisco **Color** Any **Price** From $10,100 to $15,000 **Description**

Search | Clear Form

Reset Filters | Reset Sorts

Make	Model	Year	City	Mileage	Price	Category	Description	Color
BMW	5-Series	1997	San Francisco	14300	13100	Luxury	5-speed, heavy-duty suspension, extra wide tires. Well-maintained by mechanic-owner. Cloth seats and upgraded stereo system.	White
BMW	5-Series	1997	San Francisco	14600	13100	Luxury	Is that price for real? You bet it is. Fully loaded with all factory options. Former floor model.	Beige
BMW	5-Series	1997	San Francisco	14900	13100	Luxury	Fun to drive. Manual 5-speed transmission, turbo charger. Garaged all winter and pampered the rest of the year. This is a steal!	Orange
BMW	5-Series	1997	San Francisco	14800	13200	Luxury	Fully loaded, automatic transmission. Power everything. Anti-lock brakes and full safety features. Must test drive. Price firm.	Green
BMW	5-Series	1997	San Francisco	14300	13200	Luxury	Formerly an executive's vehicle. Interior has been professionally maintained, engine factory serviced every 3000 miles. Great gas mileage. Price negotiable.	Maroon
BMW	5-Series	1997	San Francisco	15000	13200	Luxury	Sun roof, air, CD player, driver side air bag. 10% deposit required. Owner financing available. Best offer by end of weekend buys it	Red

Zones

- A zone is an identified region within a doc
- E.g., Title, Abstract, Bibliography
- Generally culled from marked-up input or document metadata (e.g., powerpoint)
- Contents of a zone are free text
- Not a “finite” vocabulary
- Indexes for each zone -allow queries like
- sorting in Title AND smith in Bibliography AND recur* in Body

Boosting

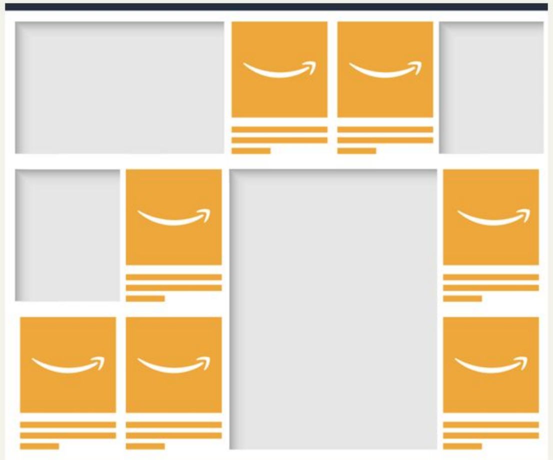
- Supported by Solr
- What to boost Query terms
  - E.g. terms appearing in title more important those those in body of document
  - Named entities
- Documents
  - E.g. more recent documents

Amazon Product Search (Sept 2019)

◆ WSJ NEWS EXCLUSIVE

Amazon Changed Search Algorithm in Ways That Boost Its Own Products

The e-commerce giant overcame internal dissent from engineers and lawyers, people familiar with the move say



Index support for zone combinations

- In the simplest version we have a separate inverted index for each zone
- Variant: have a single index with a separate dictionary entry for each term and zone
- E.g.,

bill.author	1 → 2
bill.title	3 → 5 → 8
bill.body	1 → 2 → 5 → 9

Of course, compress zone names like author/title/body.

Zone combinations index

- The above scheme is still wasteful: each term is potentially replicated for each zone
- In a slightly better scheme, we encode the zone in the postings:

bill → 1.author, 1.body → 2.author, 2.body → 3.title

As before, the zone names get compressed.

Speeding up vector space ranking

Computing cosine scores

```
CosineScore(q)
float Scores[N] = 0
float Length[N]
for each query term t
do calculate w_{t,q} and fetch postings list for t
    for each pair(d, tf_{t,d}) in postings list
    do Scores[d] += w_{t,d} × w_{t,q}
Read the array Length
for each d
do Scores[d] = Scores[d]/Length[d]
return Top K components of Scores[]
```

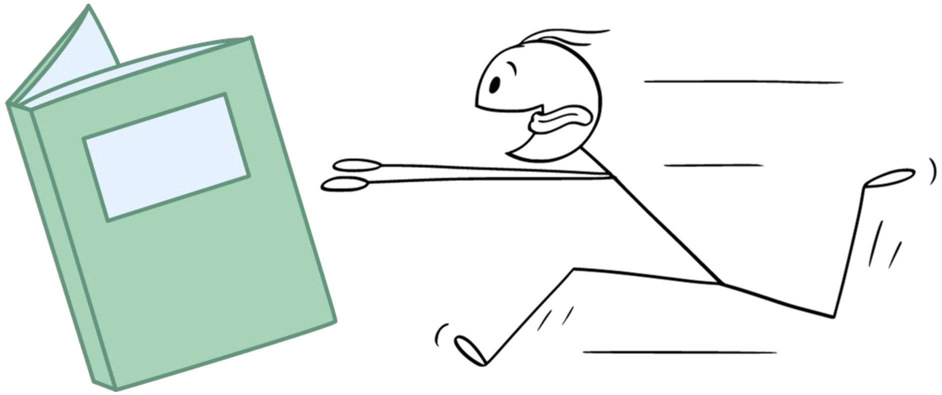
Scoring

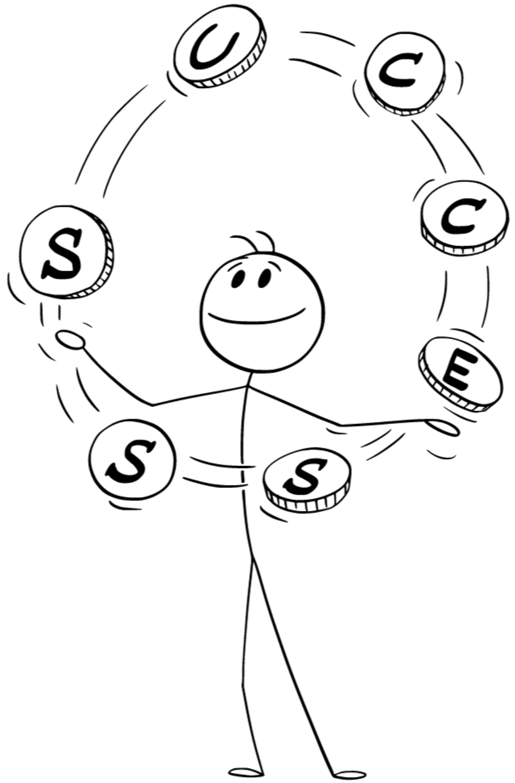
DOT PRODUCT

LENGTH NORMALIZATION

Finding the Best

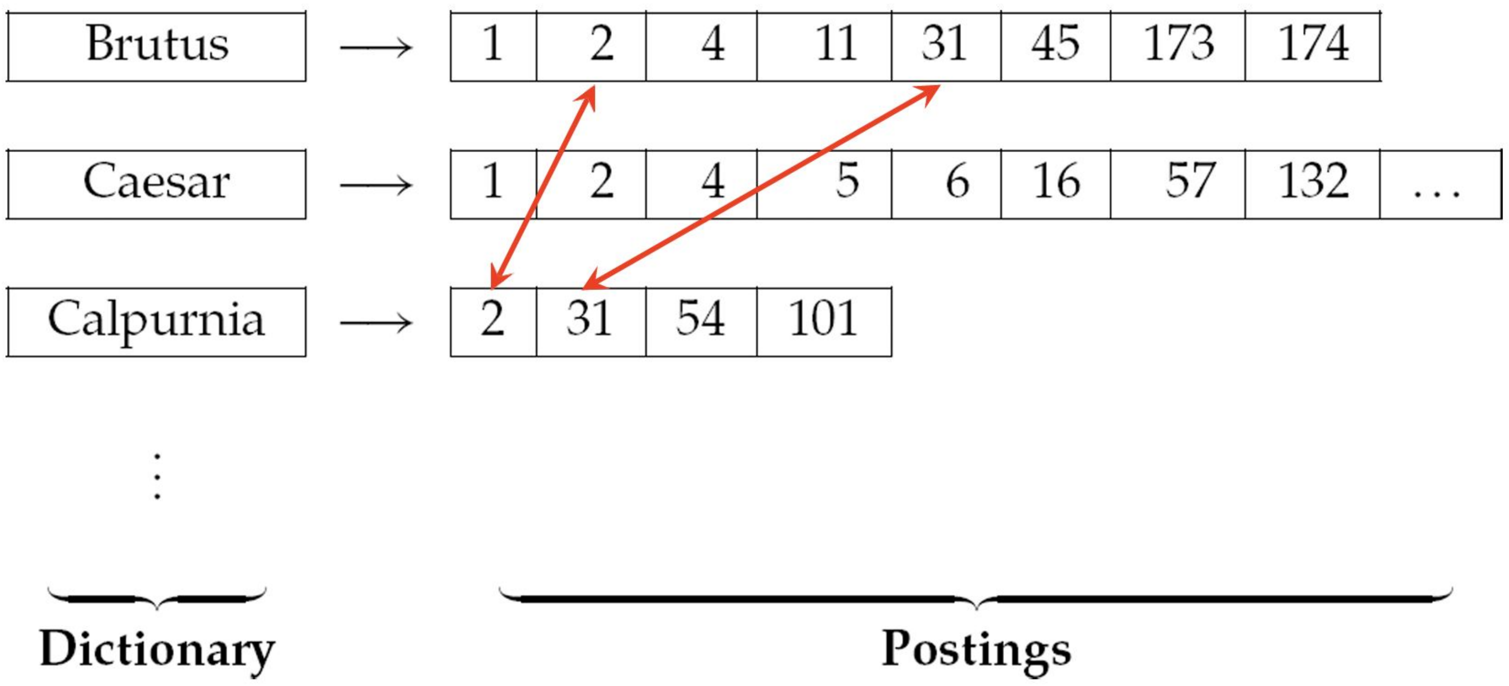
Document-at-a-Time (DaaT) vs Term-at-a-Time (TaaT) scoring





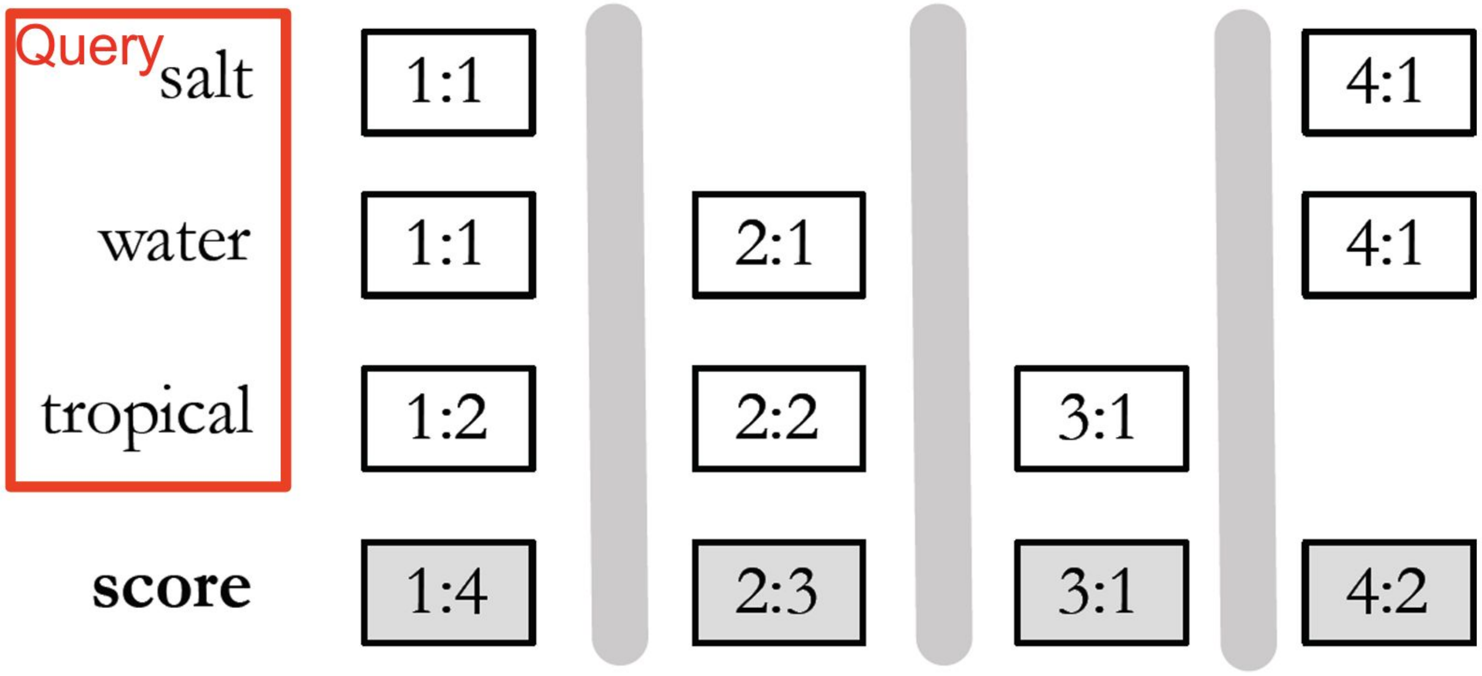
Inverted Indexes

Query “Brutus” AND “Calpurnia”



Document-at-a-time Evaluation

- The conceptually simplest query answering method



University at Buffalo
Department of CSE

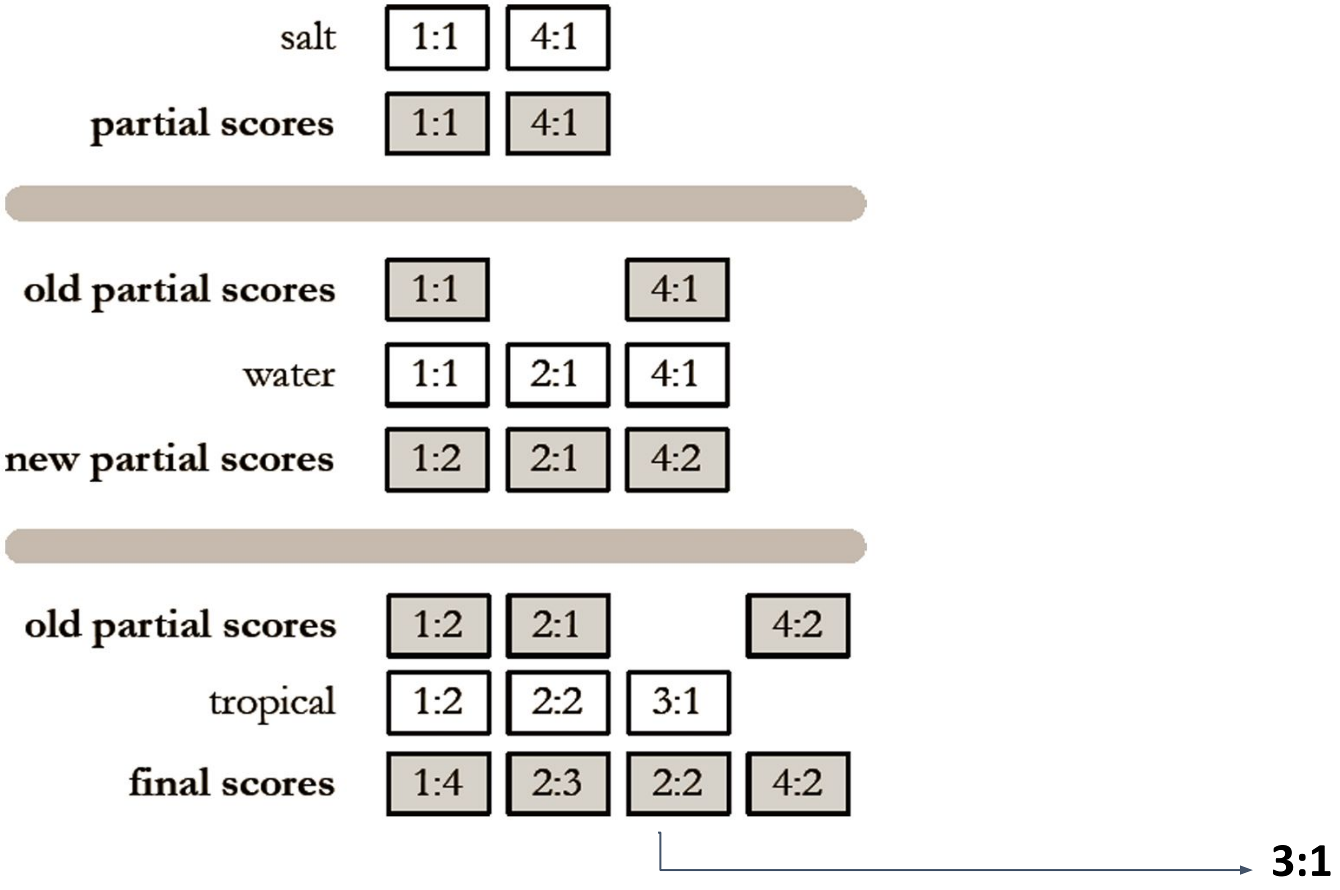
Algorithm

procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
    $L \leftarrow$ 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 documents $d \in I$ **do**
        for all inverted lists l_i in L **do**
            if l_i points to d **then**
                $s_D \leftarrow s_D + g_i(Q) f_i(l_i)$ ▷ 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

Find posting lists

Can be implemented efficiently by keeping the top-k list at anytime

Term-at-a-time Evaluation



Algorithm

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

Compute scores on one term

Can be implemented efficiently by keeping the top-k list at anytime

Comparison

- Memory usage
  - The document-at-a-time only needs to maintain a priority queue R of a limited number of results
  - The term-at-a-time needs to store the current scores for all documents
- Disk access
  - The document-at-a-time needs more disk seeking and buffers for seeking since multiple lists are read in a synchronized way
  - The term-at-a-time reads through each inverted list from start to end-requiring minimal disk seeking and buffer

EFFICIENT SCORING and SELECTING

Efficient cosine ranking

- Find the K docs in the collection "nearest" to the query => K largest query-doc cosines.
- Efficient ranking:
  - Computing a single cosine efficiently.
  - Choosing the K largest cosine values efficiently.
    - Can we do this without computing all N cosines?

Efficient cosine ranking

- Special case
  - unweighted queries
- No weighting on query terms
- Assume each query term occurs only once
  - Then for ranking, don't need to normalize query vector

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the weight of term i in the query
d_i is the weight of term i in the document

Computing the K largest cosines:

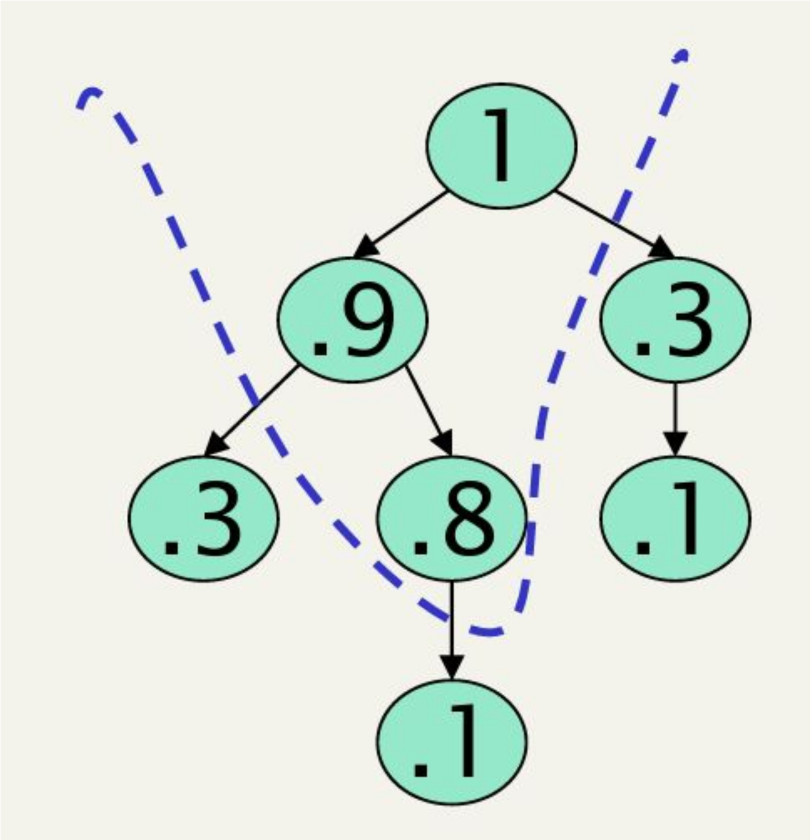
- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
  - Not to totally order all docs in the collection
- Can we get docs with K highest cosines?

Computing the K largest cosines: selection vs. sorting

- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
  - Not to totally order all docs in the collection
- Can we pick off docs with K highest cosines?
- Let J = number of docs with non-zero cosines
  - We seek the K best of these J

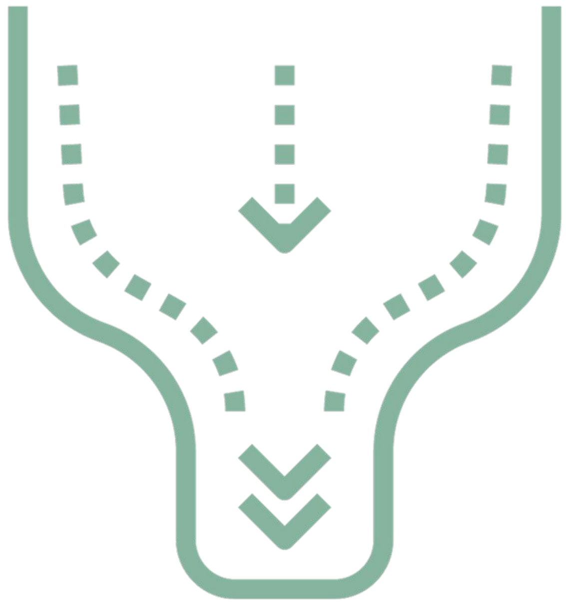
Use heap for selecting top K

- Binary tree in which each node's value > the values of children
- Takes 2J operations to construct, then each of K "winners" read off in 2(log J) steps.
- For J=1M, K=100, this is about 10% of the cost of sorting.



Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- Can we avoid all this computation?
- Yes, but may sometimes get it wrong
  - a doc not in the top K may creep into the list of K output docs
  - Is this such a bad thing?



Cosine similarity is only a proxy

- User has a task and a query formulation
- Cosine matches docs to query
- Thus cosine is anyway a proxy for user happiness
- If we get a list of K docs "close" to the top K by cosine measure, should be ok

Generic approach

- Find a set A of contenders, with $K < |A| << N$
  - A does not necessarily contain the top K, but has many docs from among the top K
  - Return the top K docs in A
- Think of A as pruning non-contenders
- The same approach is also used for other (non-cosine) scoring functions
- Will look at several schemes following this approach

Index elimination

- Only consider high-idf query terms
- Only consider docs containing many query terms

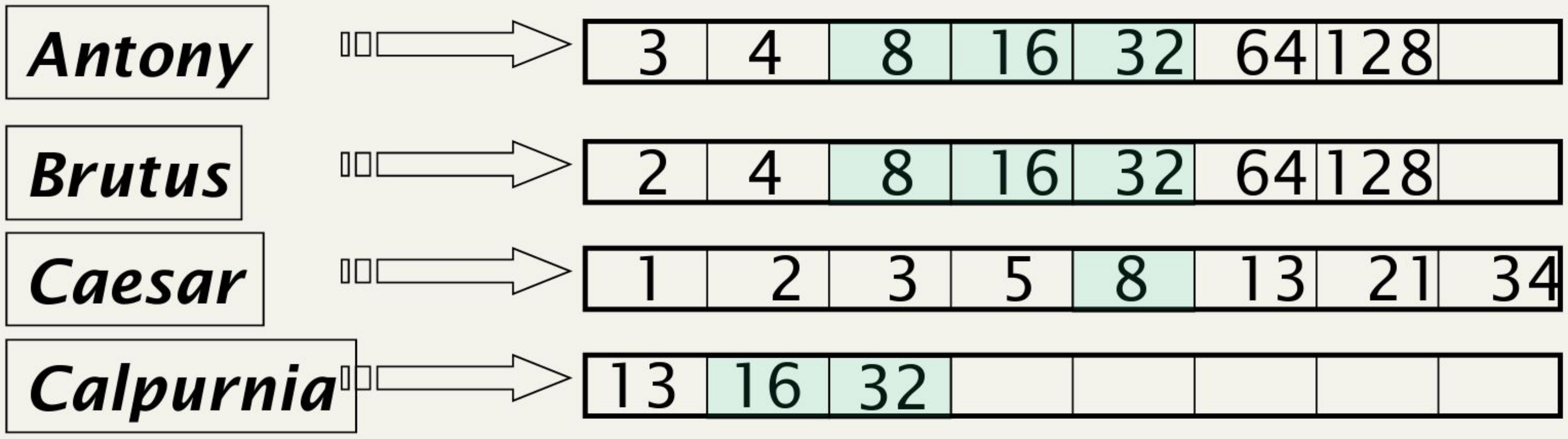
High-idf query terms only

- For a query such as catcher in the rye
- Only accumulate scores from catcher and rye
- Intuition: in and the contribute little to the scores and don’t alter rank-ordering much
    - Benefit: Postings of low-idf terms have many docs -> these (many) docs get eliminated from A

Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms
  - Say, at least 3 out of 4
  - Imposes a “soft conjunction”on queries seen on web search engines (early Google)
- Easy to implement in postings traversal

3 of 4 query terms



Scores only computed for 8, 16 and 32.

Champion lists

- Precompute for each dictionary term t, the r docs of highest weight in t's postings
    - Call this the champion list for t
    - (aka fancy list or top docs for t)
- Note that r has to be chosen at index time
- At query time, only compute scores for docs in the union of the champion lists of query term
    - Pick the K top-scoring docs from amongst these

Static quality scores

- We want top-ranking documents to be both relevant and authoritative
- Relevance is being modeled by cosine scores
- Authority is typically a query-independent property of a document
- Examples of authority signals
    - Wikipedia among websites
    - Articles in certain newspapers
    - A paper with many citations
    - How many likes
    - (Pagerank)

Modeling authority

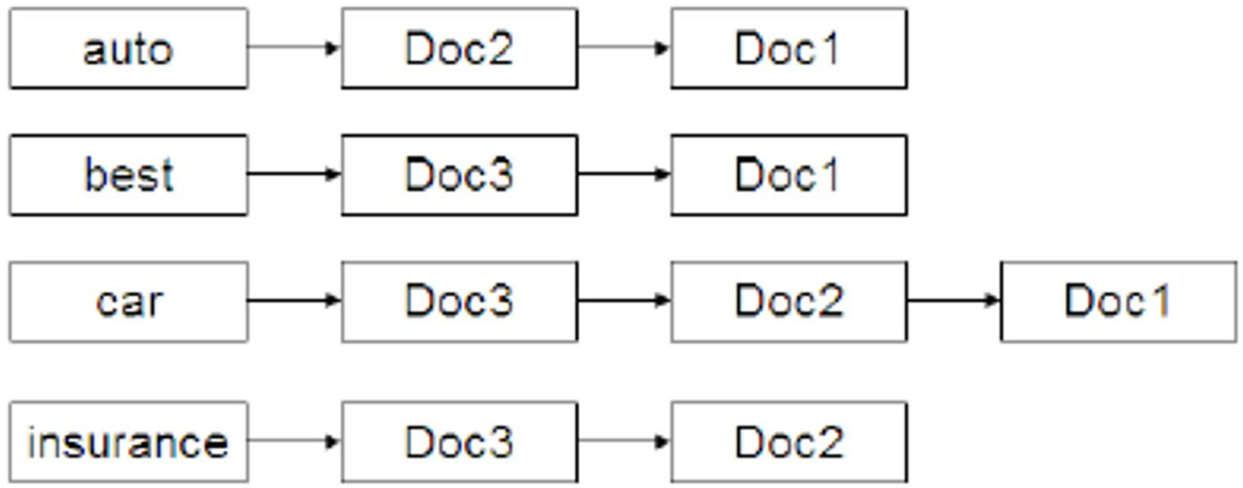
- Assign to each document a query-independent quality score in [0,1] to each document d
- Denote this by g(d)
- Thus, a quantity like the number of citations is scaled into [0,1]

Net score

- Consider a simple total score combining cosine relevance and authority
- **Net-score(q,d) = g(d) + cosine(q,d)**
  - Can use some other linear combination than an equal weighting
  - Indeed, any function of the two "signals" of user happiness –more later
- Now we seek the top K docs by net score

Top K by net score –fast methods

- First idea: Order all postings by g(d)
- Key: this is a common ordering for all postings
- Thus, can concurrently traverse query terms' postings for
  - Postings intersection
  - Cosine score computation
  - Document-at-a-time scoring
- Use accumulators to get the scores



▶ **Figure 7.2** A static quality-ordered index. In this example we assume that Doc1, Doc2 and Doc3 respectively have static quality scores $g(1) = 0.25$, $g(2) = 0.5$, $g(3) = 1$

Champion lists in g(d)-ordering

- Can combine champion lists with g(d)-ordering
- Maintain for each term a champion list of the r docs with highest g(d) + tf-idf (List is still sorted by common order, either by document id, or by static score)
- Seek top-K results from only the docs in these champion lists
  - find documents in union of these champion lists
  - Compute scores and return k highest ones

High and low lists

- For each term, we maintain two postings lists called high and low
  - Think of high as the champion list
- When traversing postings on a query, only traverse high lists first
  - If we get more than K docs, select the top K and stop
  - Else proceed to get docs from the low lists

Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!
- How do we compute scores in order to pick off top K?
  - Two ideas follow

1. Early termination

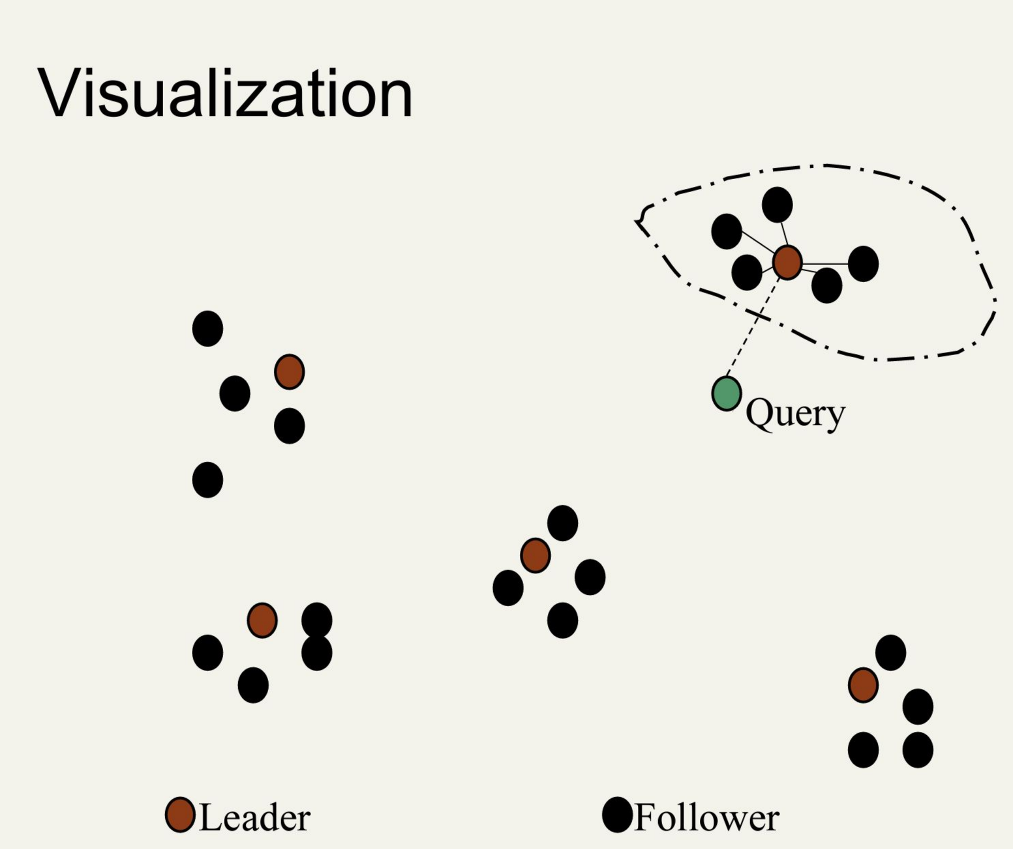
- When traversing t's postings, stop early after either
  - a fixed number of r docs
  - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
  - One from the postings of each query term
- Compute only the scores for docs in this union

2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
  - High idf terms likely to contribute most to score
- As we update score contribution from each query term
  - Stop if doc scores relatively unchanged
- Can apply to cosine or some other net scores

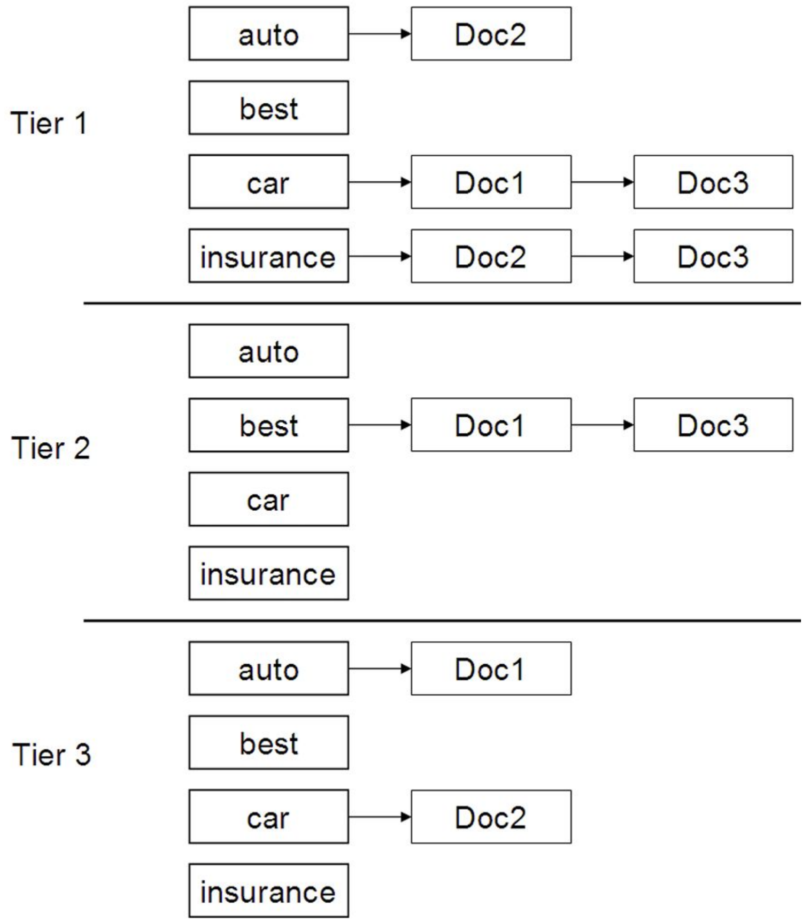
Cluster pruning: preprocessing

- Pick √N docs: call these leaders
- For every other doc, pre-compute nearest leader
  - Docs attached to a leader: its followers
  - Likely: each leader has ~ √N followers.
- Process a query as follows:
  - Given query Q, find its nearest leader L.
  - Seek K nearest docs from among L’s followers.



Tiered indexes

- Break postings up into a hierarchy of lists
- Most important
- …
- Least important
- Can be done by g(d) or another measure
- Inverted index thus broken up into tiers of decreasing importance
- At query time use top tier unless it fails to yield K docs
- If so drop to lower tiers



References

1. Slides provided by Sougata Saha (Instructor, Fall 2022 - CSE 4/535)
2. Materials provided by Dr. Rohini K Srihari
3. https://nlp.stanford.edu/IR-book/information-retrieval-book.html