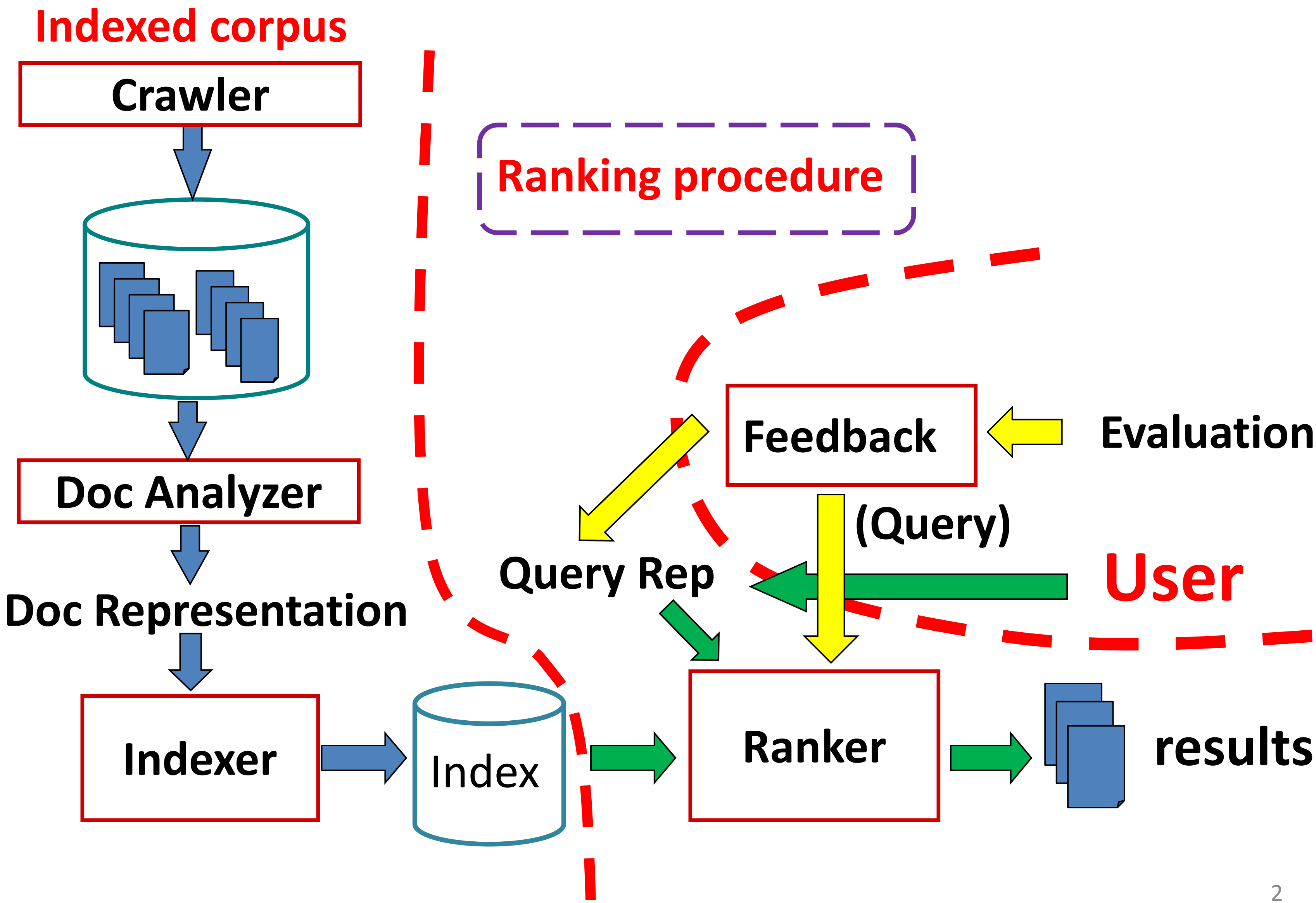
Boolean Model

Abstraction of search engine architecture

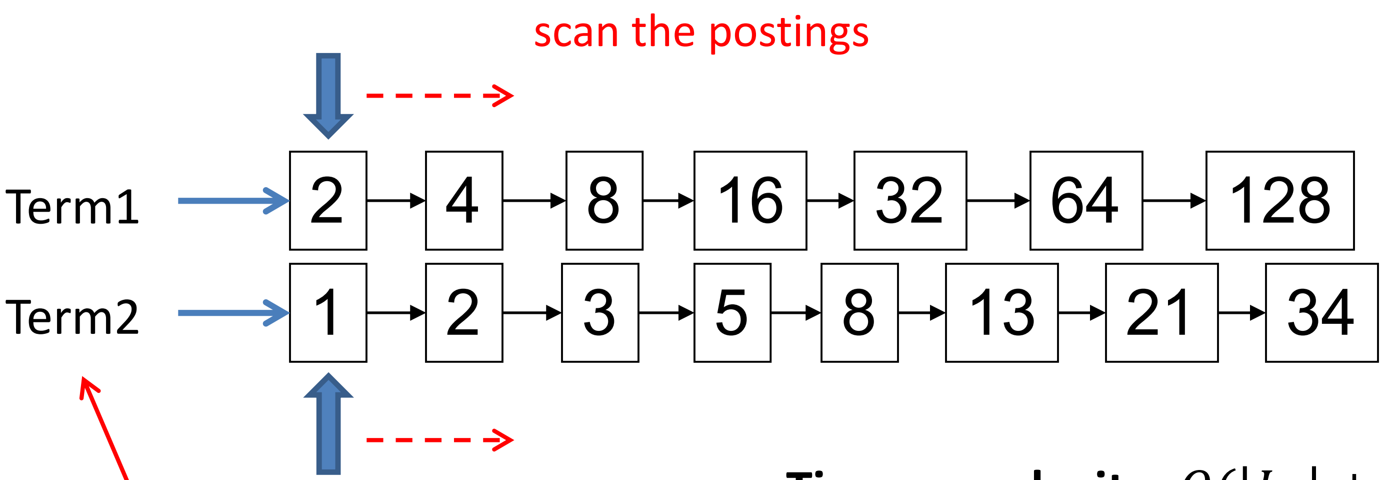


Search with Boolean query

- Boolean query
  - E.g., “obama” AND “healthcare” NOT “news”
- Procedures
  - Lookup query term in the dictionary
  - Retrieve the posting lists
  - Operation
    - AND: intersect the posting lists
    - OR: union the posting list
    - NOT: diff the posting list

Search with Boolean query

- Example: AND operation



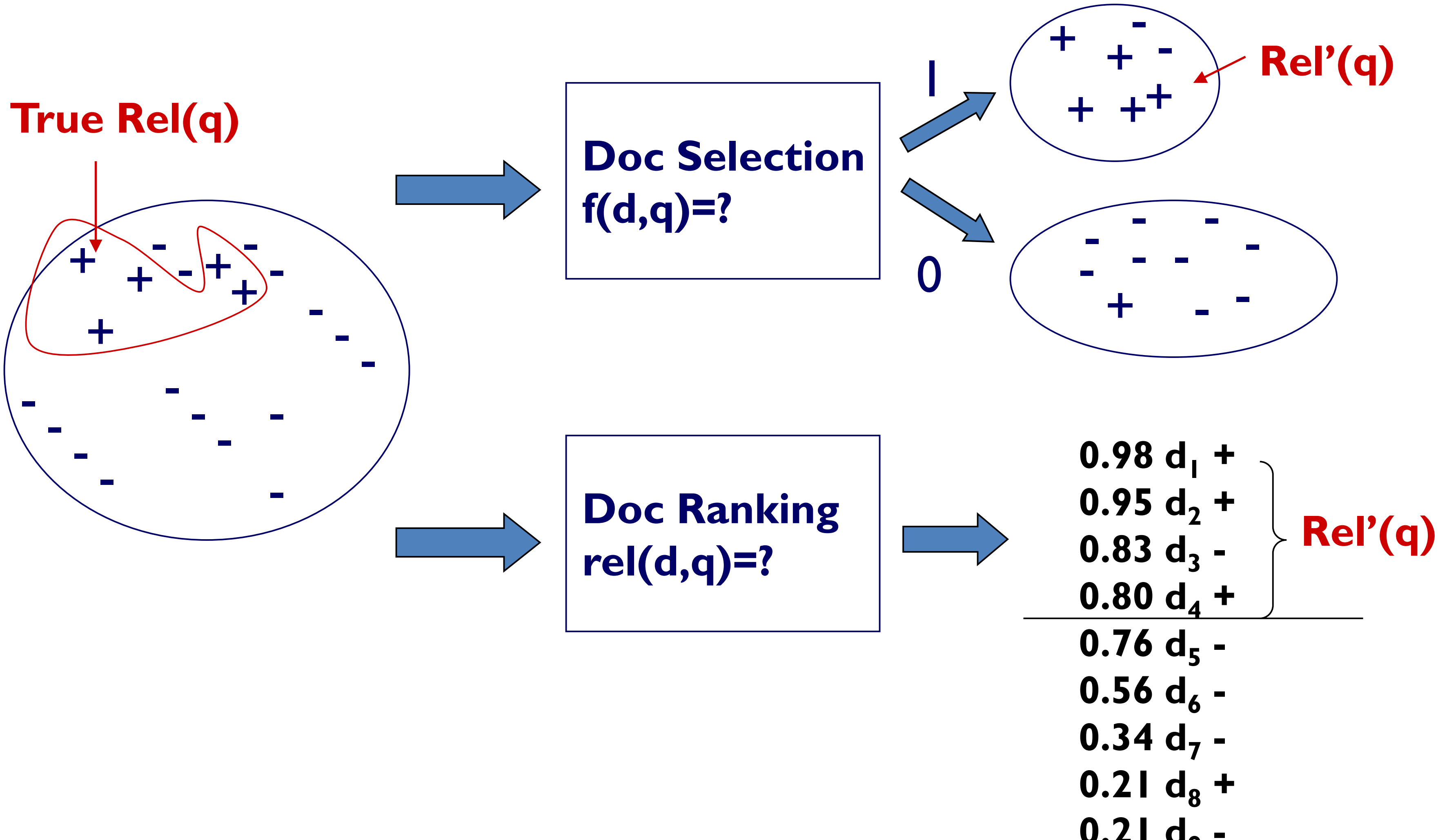
Time complexity: $O(|L_1| + |L_2|)$

Trick for speed-up: when performing multi-way join, starts from lowest frequency term to highest frequency ones

Deficiency of Boolean model

- The query is unlikely precise
  - "Over-constrained" query (terms are too specific): no relevant documents found
  - "Under-constrained" query (terms are too general): over delivery
  - It is hard to find the right position between these two extremes (hard for users to specify constraints)
- Even if it is accurate
  - Not all users would like to use such queries
  - All relevant documents are not equally relevant
    - No one would go through all the matched results
- Relevance is a matter of degree!

Document Selection vs. Ranking



Ranking is often preferred

- Relevance is a matter of degree
  - Easier for users to find appropriate queries
- A user can stop browsing anywhere, so the boundary is controlled by the user
  - Users prefer coverage would view more items
  - Users prefer precision would view only a few

Retrieval procedure in modern IR

- Boolean model provides all the ranking candidates
  - Locate documents satisfying Boolean condition
    - E.g., "obama healthcare" -> "obama" OR "healthcare"
- Rank candidates by relevance
  - Important: the notation of relevance
- Efficiency consideration
  - Top-k retrieval (Google)

Notion of relevance

Relevance

- **Δ(Rep(q), Rep(d))** **Similarity**
  - **Different rep & similarity**
    - **Vector space model** (Salton et al., 75)
    - …
    - **Prob. distr. model** (Wong & Yao, 89)
- **P(r=1|q,d) r $\in$ {0,1}** **Probability of Relevance**
  - **Regression Model** (Fox 83)
  - **Generative Model**
    - **Doc generation**
      - **Classical prob. Model** (Robertson & Sparck Jones, 76)
    - **Query generation**
      - **LM approach** (Ponte & Croft, 98) (Lafferty & Zhai, 01a)
- **P(d $\rightarrow$ q) or P(q $\rightarrow$ d)** **Probabilistic inference**
  - **Different inference system**
    - **Prob. concept space model** (Wong & Yao, 95)
    - **Inference network model** (Turtle & Croft, 91)

This lecture

Some notations

- Vocabulary $V=\{w_1, w_2, \ldots, w_N\}$ of language
- Query $q = t_1,\ldots,t_m$, where $t_i \in V$
- Document $d_i = t_{i1},\ldots,t_{in}$, where $t_{ij} \in V$
- Collection $C= \{d_1, \ldots, d_k\}$
- Rel(q,d): relevance of doc d to query q
- Rep(d): representation of document d
- Rep(q): representation of query q

Vector Space Model

Relevance = Similarity

- Assumptions
  - Query and documents are represented in the same form
    - A query can be regarded as a "document"
  - Relevance(d,q) $\propto$ similarity(d,q)
- Key issues
  - How to represent query/document?
  - How to define the similarity measure $\Delta(x,y)$?

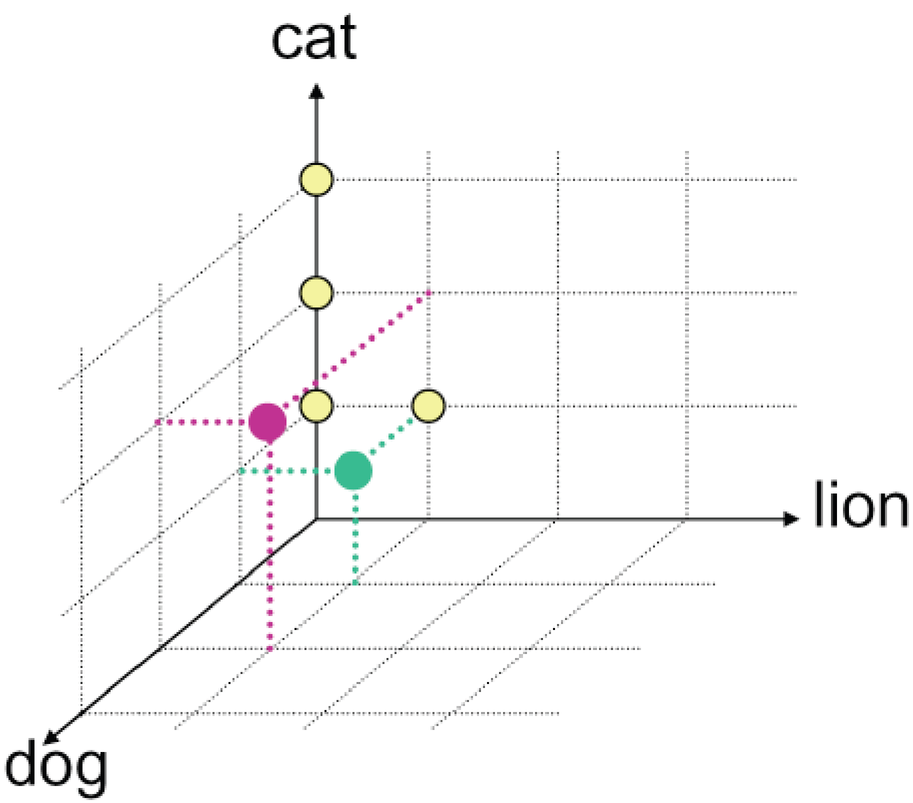
Vector space model

- Represent both doc and query by <u>concept</u> vectors
  - Each concept defines one dimension
  - *K* concepts define a high-dimensional space
  - Element of vector corresponds to concept weight
    - E.g., $d=(x_1,\ldots,x_k)$, x_i is "importance" of concept i
- Measure relevance
  - Distance between the query vector and document vector in this concept space

Documents as vectors

- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

Terms are axes of space



Queries as vectors

- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

How to assign weights?

- Important!
- Why?
  - Query side: not all terms are equally important
  - Doc side: some terms carry more information about the content
- How?
  - Two basic heuristics
    - TF (Term Frequency) = Within-doc-frequency
    - IDF (Inverse Document Frequency)

TF weighting

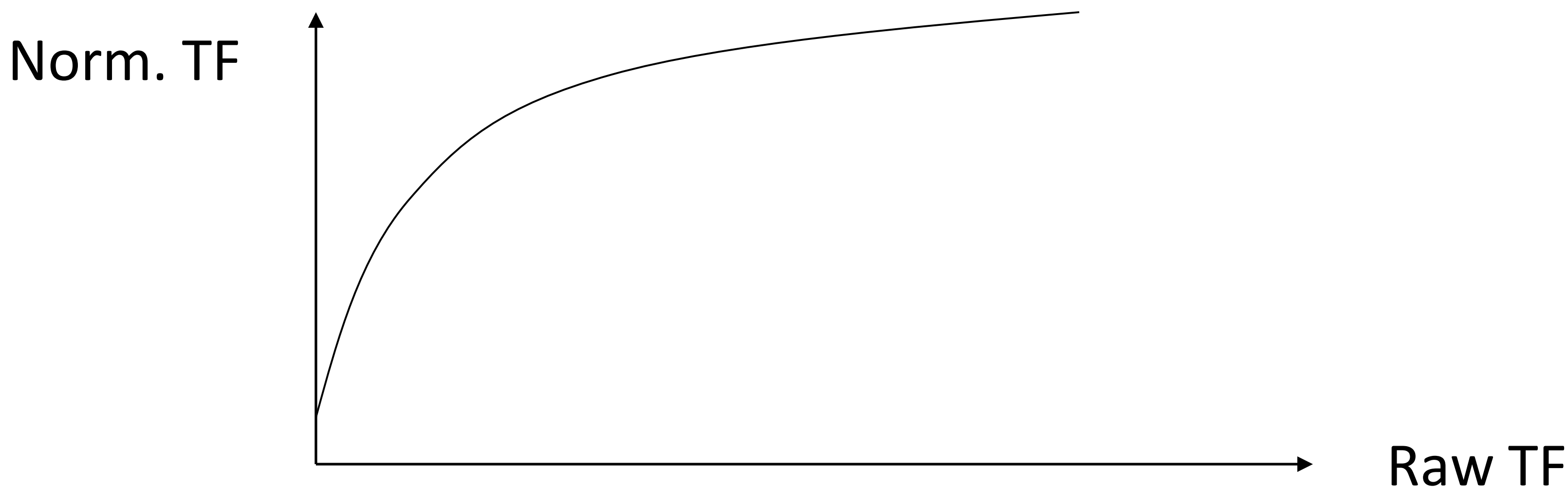
- Idea: a term is more important if it occurs more frequently in a document
- TF Formulas
  - Let $f(t, d)$ be the frequency count of term t in doc d
  - Raw TF: $tf(t, d) = f(t, d)$

TF normalization

- Two views of document length
  - A doc is long because it is verbose
  - A doc is long because it has more content
- Raw TF is inaccurate
  - Document length variation
  - "Repeated occurrences" are less informative than the "first occurrence"
  - Relevance does not increase proportionally with number of term occurrence
- Generally penalize long doc, but avoid over-penalizing
  - Pivoted length normalization

TF normalization

- Sublinear TF scaling
  - $tf(t,d) = \begin{cases} 1 + \log f(t,d), if\ f(t,d) > 0 \\ 0, otherwise \end{cases}$



tf	1 + log (tf)
0	0
1	1
2	1.3
3	1.47
4	1.6
5	1.7
6	1.77
10	2
100	3
1000	4

Document frequency

- Idea: a term is more discriminative if it occurs only in fewer documents

IDF weighting

- Solution
  - Assign higher weights to the rare terms
  - Formula
    - $IDF(t) = \log(\frac{N}{df(t)})$

      Non-linear scaling

      Total number of docs in collection

      Number of docs containing term t
  - A corpus-specific property
    - Independent of a single document

Example

Table 2. Raw frequency count used as score of documents

	disease	symptom	osteoporosis	
Doc 1	20	15	1	36
Doc 2	3	10	13	26

Table 3. log Normalized frequency (1 + log (tf)) used as score of documents

	disease	symptom	osteoporosis	
Doc 1	2.3	2.2	1	5.5
Doc 2	1.47	2	2.1	4.57

Example

Table 3. log Normalized frequency (1 + log (tf)) used as score of documents

	disease	symptom	osteoporosis	
Doc 1	2.3	2.2	1	5.5
Doc 2	1.47	2	2.1	4.57

Table 4. Idf score of words

	Document frequency	$IDF(t) = 1 + \log(\frac{N}{df(t)})$
disease	2000	1+ log (100,000/2000) = 2.7
symptom	300	1+ log (100,000/300) = 3.5
osteoporosis	10	1+ log (100,000/10) = 5

TF-IDF score of Doc 1 = 2.3*2.7 + 2.2*3.5 + 1*5 = 18.9

TF-IDF score of Doc 2 = 1.47*2.7 + 2*3.5 + 2.1*5 = 21.5

Why document frequency

- How about total term frequency?
  - $ttf(t) = \sum_d f(t,d)$

Table 1. Example total term frequency v.s. document frequency in Reuters-RCV1 collection.

Word	ttf	df
try	10422	8760
insurance	10440	3997

TF-IDF weighting

- Combining TF and IDF
  - Common in doc → high tf → high weight
  - Rare in collection → high idf → high weight
  - $w(t,d) = TF(t,d) \times IDF(t)$
- Most well-known document representation schema in IR! (G Salton et al. 1983)

"Salton was perhaps the leading computer scientist working in the field of information retrieval during his time." - wikipedia

Gerard Salton Award – highest achievement award in IR

Question

- Suppose a user enters a single word query to two different search engines. One search engine uses normalized TF as score of documents and other search engine uses normalizedTF*IDF as score of documents. Will the two search engines produce different rankings of documents ?