Information Retrieval
CS 6900

Lecture 06

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Boolean Retrieval vs. Ranked Retrieval

- Many users (professionals) prefer Boolean query models:
  - Boolean queries are precise: a document either matches the query or it does not.
    - Greater control and transparency over what is retrieved.
  - Some domains allow an effective ranking criterion:
    - Westlaw returns documents in reverse chronological order.

- Hard to tune precision vs. recall:
  - AND operator tends to produce high precision but low recall.
  - OR operator gives low precision but high recall.
  - Difficult/impossible to find satisfactory middle ground.

Boolean Retrieval vs. Ranked Retrieval

- Need an effective method to **rank** the matched documents.
  - Give more weight to documents that mention a token several times vs. documents that mention it only once.
    - record **term frequency** in the postings list.

- Web search engines implement ranked retrieval models:
  - Most include at least partial implementations of Boolean models:
    - Boolean operators.
    - Phrase search.
  - Still, improvements are generally focused on free text queries.

Ranked Retrieval vs. Text Queries

- **Ranked retrieval**: the system returns an ordering over the (top) documents in the collection for a query.
- **Free text queries**: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language.
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa.

Scoring for Ranked Retrieval

- We wish to return in order the documents most likely to be useful to the searcher.
- How can we rank-order the documents in the collection with respect to a query?
- Retrieval based on similarity between query and documents:
  - Output documents are ranked according to similarity to query.

Criteria for Query-Document Similarity

1. The more frequent a query term is in the document, the higher the similarity.
   - need to compute **term frequency** (tf).

2. Rare terms in a collection are more informative than frequent terms.
   - need to compute **inverse document frequency** (idf).

=> the **tf.idf** weighting scheme in the **vector space model**.

The Vector Space Model

- Vocabulary V = the set ot terms left after pre-processing the text (*tokenization*, *stemming*, *case folding*, ...).
- Each document or query is represented as a |V| = T dimensional vector:
  - $d_j = [w_{1j}, w_{2j}, ..., w_{Tj}]$.
  - w_{ij} is the weight of term *i* in document *j*.

  $\Rightarrow$ the terms in V form the orthogonal dimensions of a vector space.
- Document = **Bag of words**:
  - Vector representation doesn't consider the ordering of words:
    - *John is quicker than Mary* vs. *Mary is quicker than John.*

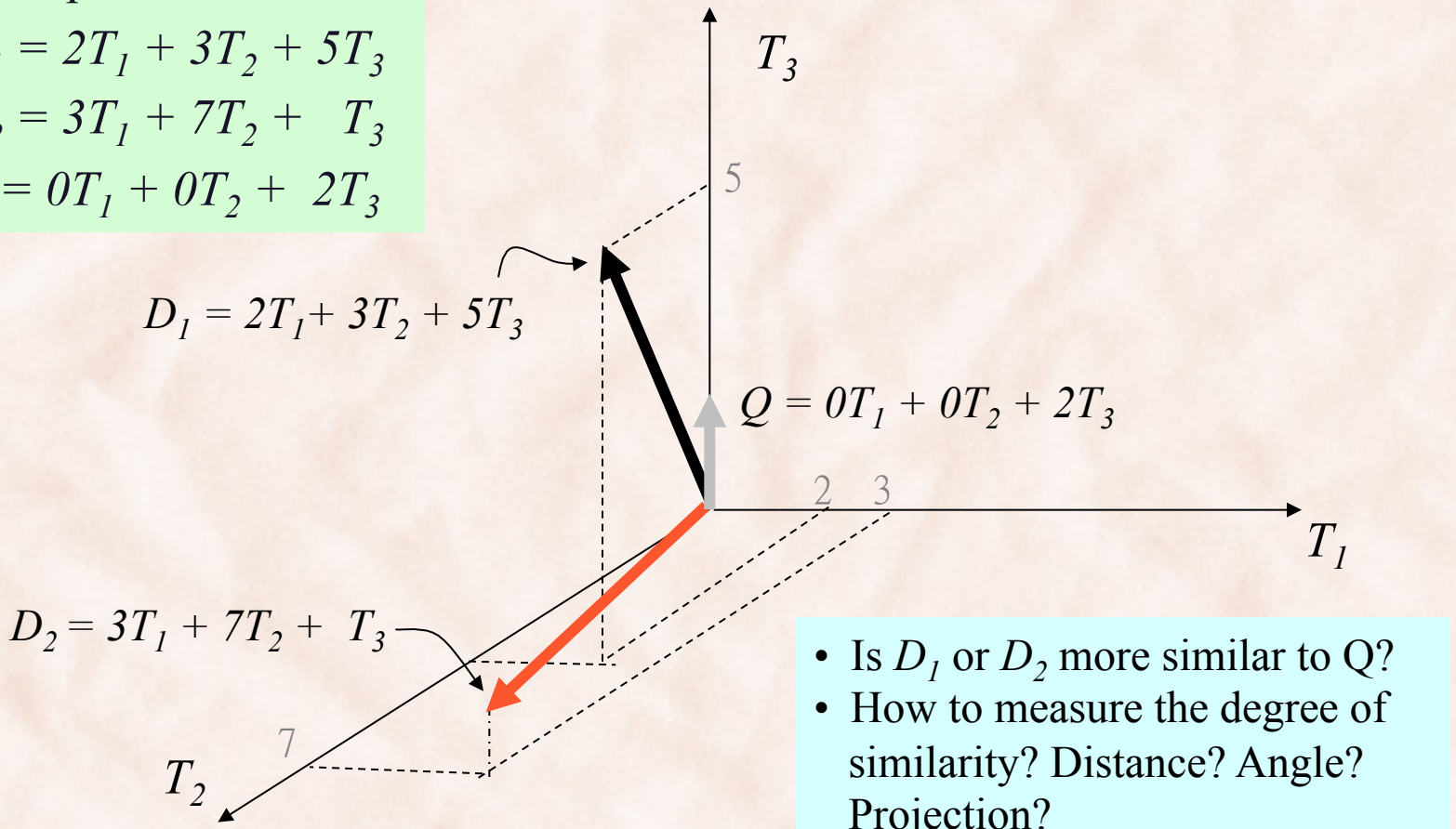
Graphical Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

The Term-Document Weight Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

- Each document is represented as a vector of real valued term weights in a |V| dimensional space.
  - what are these weights?

Term Weights: TF

- More frequent terms in a document are more important, i.e. more indicative of the topic.

  f_{ij} = frequency of term i in document j.

- May want to normalize *term frequency* (*tf*) by dividing by the frequency of the most common term in the document:

  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

- Or sublinear *tf* scaling:

  $tf_{ij} = 1 + log\ f_{ij}$

Term Weights: IDF

- Terms that appear in many *different* documents are *less* indicative of overall topic.

  df_i = document frequency of term i

  = number of documents containing term i

  idf_i = inverse document frequency of term i,

  = $\log_2 (N/ df_i)$

  (N: total number of documents)

- An indication of a term's *discrimination* power.
  - Log used to dampen the effect relative to *tf*.
- Why not use inverse collection frequency?

TF.IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N/ df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

TF.IDF Example

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

A: tf = 3; idf = $\log_2$(10000/50) = 7.6; tf-idf = 3*7.6

B: tf = 2; idf = $\log_2$ (10000/1300) = 2.9; tf-idf = 2*2.0

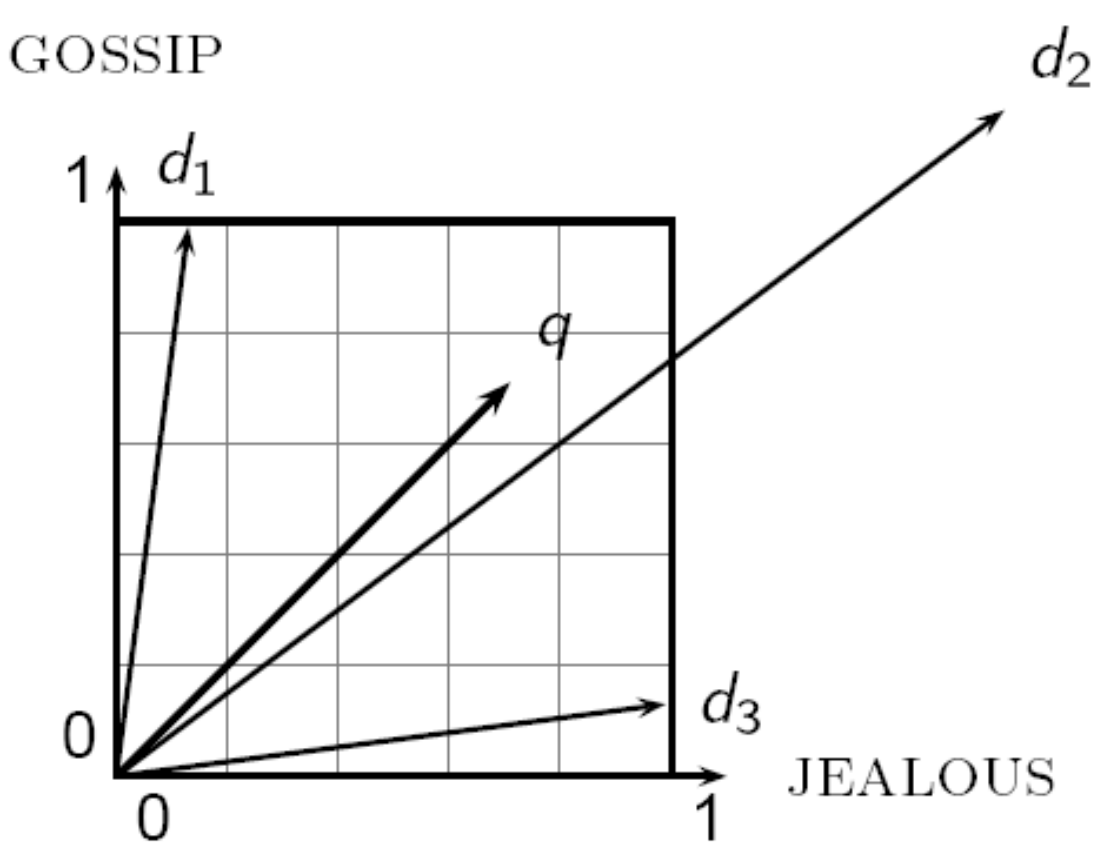
C: tf = 1; idf = $\log_2$ (10000/250) = 5.3; tf-idf = 1*1.8

Ranking with VSM

- So we have a |V|-dimensional vector space
  - Terms are axes of the space.
- Documents are points or vectors in this space.
- Queries too are represented as vectors in the space

- Rank documents according to their "proximity" or "similarity" to the query in this space.
  - How do we define similarity?
    - The inverse of distance?

Euclidean Distance



Euclidean Distance

- Euclidean Distance is large for vectors of different lengths:
  - Take a document *d* and append it to itself. Call this document *d*′.
    - "Semantically" d and d′ have the same content
  - The Euclidean distance between *d* and *d*′ can be quite large.
    - But the angle between the two documents is 0.

=> Rank documents in **decreasing order of angle** to query.

<=> Rank documents in **increasing order of cosine of angle** to query.

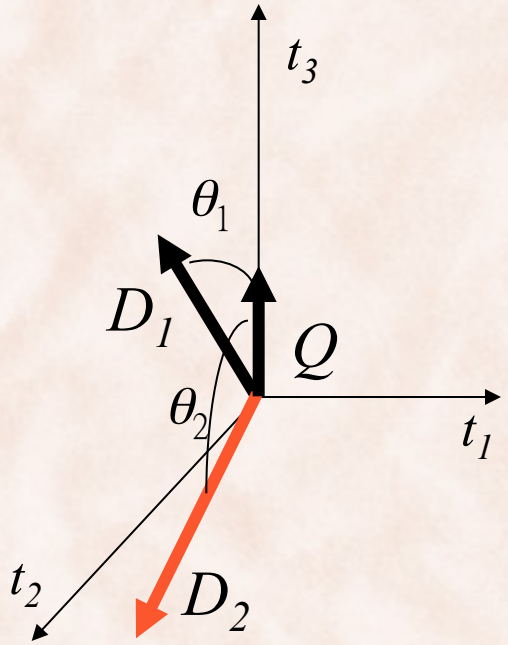
Cosine Similarity

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 tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document



$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = 10 / √(4+9+25)(0+0+4) = 0.81
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = 2 / √(9+49+1)(0+0+4) = 0.13
$Q = 0T_1 + 0T_2 + 2T_3$

Cosine Similarity

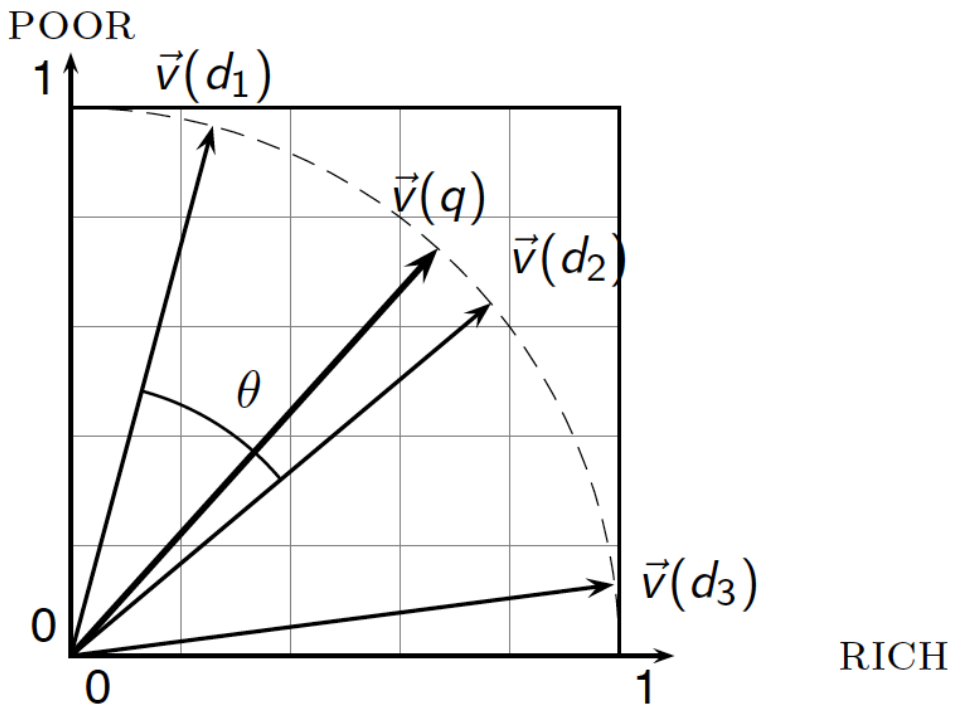
- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

  - Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
  - Effect on the two documents d and d′ (d appended to itself) from earlier slide?
  - Long and short documents now have comparable weights.

- What is the cosine *cos*(*q*,*d*) of two normalized vectors?

Cosine Similarity vs. Euclidean Distance



Compare ranking given by cos(d_i,q) with ranking given by dist(d_i,q).

Alternative TF.IDF Weighting Schemes

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

- Prove that base of the log in *idf* is immaterial for ranking.

Weighting Schems: Documents vs. Queries

- Many search engines allow for different weightings for queries vs. documents:
  - SMART Notation: *ddd*.*qqq* denotes the combination in use in an engine, using the acronyms from the previous table.

- A very standard weighting scheme is lnc.ltc
  - Document: logarithmic tf (l as first character), no idf, and cosine normalization.
  - Query: logarithmic tf (l as first character), idf (t in second column), no normalization.

Query Processing: Document-At-A-Time

1. Convert all documents in collection D to tf-idf weighted vectors $\boldsymbol{d_j}$, for keyword vocabulary V.
2. Convert query to a tf-idf-weighted vector $\boldsymbol{q}$.
3. For each $\boldsymbol{d_j}$ in D:

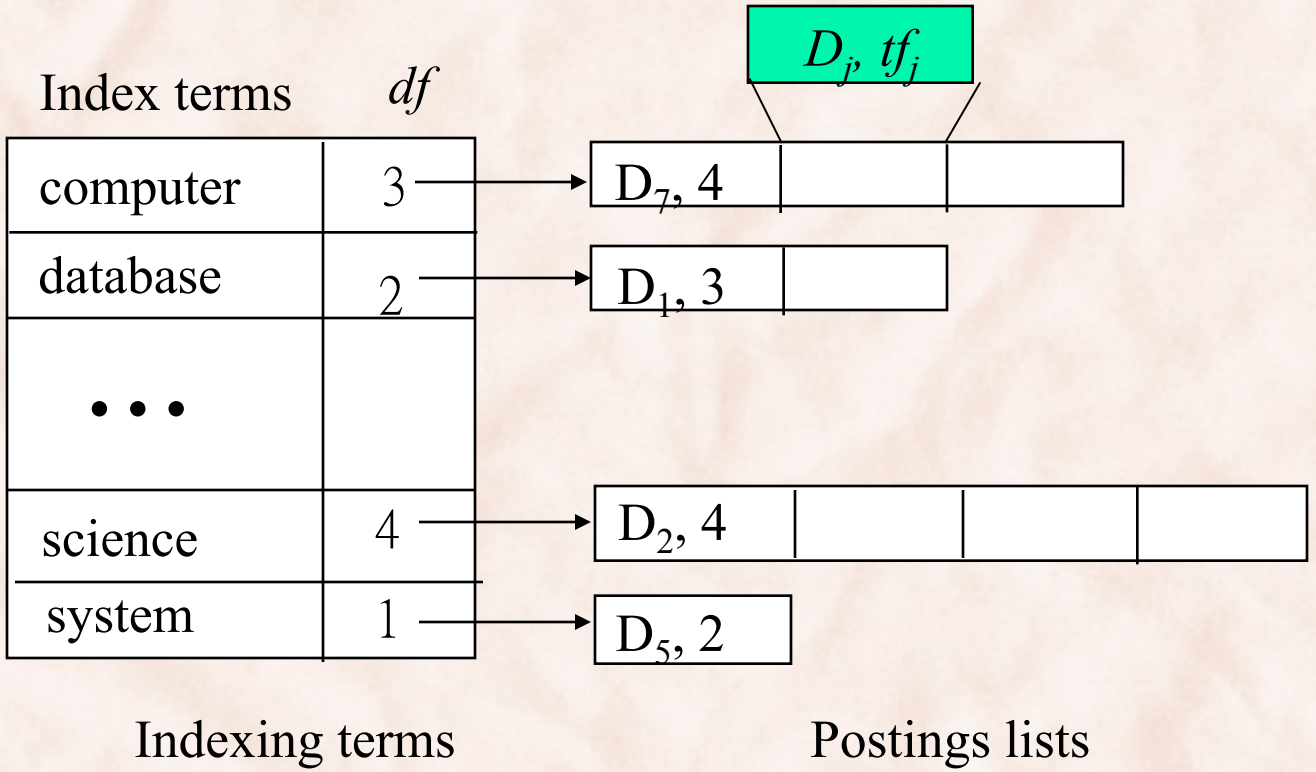
   Compute score $\mathbf{s_j} = \text{cosSim}(\boldsymbol{d_j}, \boldsymbol{q})$
4. Sort documents by decreasing score.
5. Present top **k** ranked documents to the user.

=> Time Complexity?

Query Processing: Term-At-A-Time

- Based on the observation that documents containing none of the query keywords do not affect the final ranking.
- Try to identify only those documents that contain at least one query keyword.
- Actual implementation with an **inverted index**:
  - Input: Tokens obtained from the preprocessing module.
  - Output: An inverted index for fast access.
  - Do not store tf.idf weights directly, because floating point numbers waste storage:
    - Store document frequencies (df) and term frequencies (tf).

Indexing Scheme



Building the Index

- TF and IDF for each token can be computed in one pass.
- Cosine similarity also required document lengths.
- Need a second pass to compute document vector lengths
  - Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.
  - Remember the weight of a token is: TF * IDF.
  - Therefore, must wait until IDF's are known (i.e. until all documents are indexed) before document lengths can be determined.
- Do a second pass over all documents:
  - keep a list or hashtable with all document id-s, and for each document determine its length.

Time Complexity?

Query Processing: Term-At-A-Time

- Incrementally compute cosine similarity of each indexed document as query words are processed one by one.
  - To accumulate a total score for each retrieved document, store retrieved documents in a hashtable, where the document id is the key, and the partial accumulated score is the value.
- Ranking of the retrieved documents:
  - Naive: Sort the retrieved documents based on the value of cosine similarity, return top K documents:

    => time complexity = ?
  - Better: Build a Max-Heap with retrieved documents, extract top K documents:

    => time complexity = ?

Query Processing: Term-At-A-Time

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

Evaluating an IR System

- Note: **user need** is translated into a **query**.
- Relevance is assessed relative to the **user need,** *not* the **query**.
- Information need: *My swimming pool bottom is becoming black and needs to be cleaned.*
- Query: ***pool cleaner***
- Assess whether the retrieved document addresses the underlying need, not whether it has these words:
  - Binary Assessments: Relevant or Nonrelevant.

Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = P(relevant | retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved | relevant)

	Relevant	Nonrelevant
Retrieved	*tp*	*fp*
Not Retrieved	*fn*	*tn*

$$\text{Precision } P = tp / (tp + fp)$$

$$\text{Recall } \quad R = tp / (tp + fn)$$

Vector Space Model: PROS

- Simple, mathematically based approach:
  - Represent the query & documents as weighted tf-idf vectors.
  - Compute the cosine similarity score for the query vector and each document vector.
  - Rank documents by score and return the top K (e.g. 10) to the user.
- Considers both local (*tf*) and global (*idf*) word frequencies.
- Provides partial matching and ranked results.
- Tends to work quite well in practice despite obvious weaknesses.
- Allows efficient implementation for large corpora:
  - See also IIR 4, 5, and 7.1.

Vector Space Model: CONS

- Missing semantic information (e.g. word sense).
- Missing syntactic information (e.g. phrase structure, word order, proximity information):
  - But see IIR 7.2.2 and 7.2.3.
- Assumption of term independence (e.g. synonymy):
  - But see IIR 9 (query expansion and relevance feedback).
- Lacks the control of a Boolean model (e.g., requiring a term to appear in a document):
  - Given a two-term query "A B", may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.