Retrieval Models 1

CS4422/7263 Information Retrieval
Lecture 04

Jiho Noh

Department of Computer Science
Kennesaw State University

CS4422/7263 Spring 2025

KENNESAW STATE UNIVERSITY
COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING

Retrieval Models

- Retrieval Models are the theories about **relevance**
- Retrieval models provide a mathematical framework for defining the search process
- Relevance is a complex concept based on various assumptions

In document search engine, a document is **relevant** if it is one that the user perceives as containing information of value with respect to their personal information need

Boolean Retrieval Model

- Simplest form of ranking
  - e.g., `apple AND store`
- Query usually in **Boolean Logic expression**.
- Two possible outcomes: *TRUE* or *FALSE*
- The result is the *unordered* set of documents.
- "Exact-match" retrieval

Boolean Operators — Apache Examples

Operator	Description
OR	either terms should exist anywhere in the text
AND	both terms should exist anywhere in the text
+	the term after '+' should exist in the text
NOT	excludes documents that contain the term after 'NOT'
-	equivalent to **NOT**
$\sim$	returns documents that contain all the terms within a specific distance

Query Returns

Query	Returns
Apple	Documents which contain the term 'Apple'
Apple Store	Documents which contain either terms
Apple AND Store	All documents which contain both terms
+Apple Store	Documents which must contain 'Apple' and may contain 'Store'
Apple NOT "Apple Store"	Documents that contain 'Apple' but not 'Apple Store'
(Apple Store)~5	All documents which contain both terms within a specific distance

Boolean Search — Pros and Cons

Advantages

- Intuitive and explainable results
- Other types of feature (other than word existence) can be easily incorporated
- Efficient searching process, why?

Boolean Search — Cons

Disadvantages

- Rather stricter set theory rules are applied
  - a document can be either relevant or irrelevant; there's no 'somewhat' relevant
  - Boolean queries often result in either too few ($= 0$) or too many (1000s) results.
- The importance of a token is neglected
- Less flexibility to express the users' information needs
  - Users should know how to build a query using the Boolean operators

Ranked Retrieval Models (vs. Set-based Models)

Set-theoretic Models	Ranked Retrieval Models
set of “relevant” documents	list of documents ordered by relevance
query as a boolean expression	free-text query
large set of relevant documents	top k (≈ 10) relevant documents

Scoring Documents

- Measure how much information a document contains w.r.t. the user's information need
- Assign a score (e.g., [0-1]) to each document
- This score measures how well document and query "match".
- We need to model the scoring function.

Take 1: Jaccard Coefficient

- Used for gauging the similarity and diversity of sample set
- In *document-query matching* scenario, the elements are the tokens
- Alwasy assigns a score between 0 and 1

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$J(A, A) = 1$$

$$J(A, B) = 0 \text{ if } A \cap B = \emptyset$$

Scoring example using Jaccard Coefficient

- **query**: Connect phone to car
- **document 1**: connect your phone via Android apps to your car display
- **document 2**: Cactus Car Wash Marietta was voted the best car wash in town

Issues with Jaccard for scoring

- *Term Frequency* is not considered
  - (In IR, *frequency* means the count of a term)
  - Rare terms in a collection are more informative than common terms.
  - However, a rare word is contributing the same to the score (*connect* vs. *to*)

How can we weight terms differently and normalize them for length?

Term Frequency, tf

- $tf_{t,d}$ of term t in document d denotes the number of times that t occur in d.
- We want to use tf in computing document-query match scores.
- Assumptions on **term informativeness**:
  1. A document with more occurrences of the term is more relevant to the query, and
  2. Rare terms are more informative.
- Note, Relevance does not increase proportionally with term frequency.

tf Normalization

- We need to normalize the raw occurrence counts of a term for length

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization	$K + (1 - K) \frac{f_{t,d}}{max_{\{t' \in d\}} f_{t'd}}$

Other Frequency Data

Collection vs. Document Frequency

- **Collection frequency** ($cf_{t,d}$) is the number of occurrences of t in the collection
- **Document frequency** ($df_{t,d}$) is the number of documents in which t occurs
- Usually, $df_{t,d} \propto cf_{d,f}$

Inverse Document Frequency, *idf*

- df_t is document frequency of t; the number of documents that contain t
- df_t is an inverse measure of the informativeness of t
- $df_t \leq |D|$ (document count)

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

- We use log to "dampen" the effect of idf

idf example, suppose $|D| = 1M$

$$idf(t, D) = \log \frac{|D|}{df_t} = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

term	**df_t**	**idf_t**
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Note, there is one *idf* value for each term i in a collection.

TF.IDF

- Most fundamental, best known term weighting scheme
- The tf-idf weight of a term is the product of its tf weight and its idf weight.
- **tf** reflects the importance of the term t in a single document d.
- **idf** reflects the relative importance of the term t over the document collection D.

Effect of idf on Ranking

- Does idf have an effect on ranking for one-term queries, like "iPhone"
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
- For the query *capricious person*, idf weighting makes occurrences of *capricious* count for much more in the final document ranking than occurrences of *person*.

Variants of tf and idf

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(\frac{N}{1 + n_t}\right) + 1$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Practical scoring function in Lucene

$$\text{score}(q,d) = \sum_{t \in q} \left(tf_{t,d} \cdot idf_t^2 \cdot \text{t.getBoost()} \cdot \text{norm}(t,d) \right)$$

- $tf_{t,d} = \sqrt{f_{t,d}}$
- $idf(t,D) = 1 + \log\left(\frac{|D|+1}{df_t+1}\right)$
- t.getBoost() is a search time boost of term t in the query q as specified in the query text
- norm(t,d) is an index-time boost factor that solely depends on the number of tokens of this field in the document.

Term Weighting

binary (existence) ⇒ **count (occurrence)** ⇒ term weight

	a	able	about	academic	access	according	account	⋯
D1	0.01	0	0.04	0	0	0	0	
D2	0.02	0	0.04	0	0	0	0.64	
D3	0.01	0.12	0	0	0	0	0	
D4	0.03	0	0	0	2.51	0	0	
D5	0.005	0.01	0.02	6.12	4.72	0.41	0	
⋯								

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

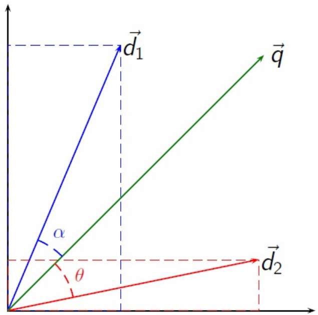
Vector Space Models (VSM)

Documents (also queries) as vectors

- A document can be represented by a point in a $|V|$**-dimensional vector space**, where terms are the axes of the space.
- The vector space is *very* high dimensional.
- The document representations are sparse vectors — Most entries are zeros.

Formalizing Vector Space Proximity

- Distance between two points (the end points of two vectors)
- Why is Euclidean distance bad?
- Use angle instead of distance
- Key idea: rank documents based on their angular relationship with the query.



Length Normalization

A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L2 norm:

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

- Dividing a vector by its L2 norm makes it a unit (length) vector (on surface of unit hypersphere).
- Effect on the two documents d and d′ (d appended to itself): they have identical vectors after length-normalization.
- Long and short documents now have comparable weights.

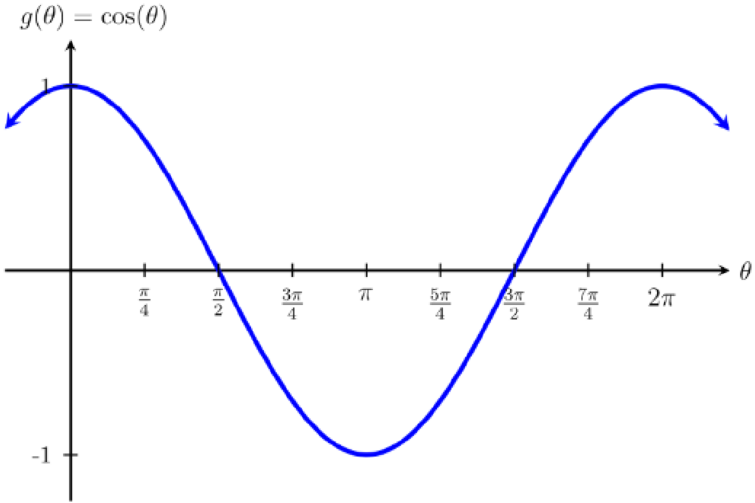
Cosine Similarity

Cosine similarity is a measure to compute the angle between two vectors, which can be used to compare the vector representations for documents/query. Let x and y be two vectors for comparison, such that $x = (x_1, x_2, \ldots, x_t)$ and $y = (y_1, y_2, \ldots, y_t)$ and θ is the angle between the two vectors,

$$\begin{aligned} cos\,\theta &= \frac{x \cdot y}{\|x\| \|y\|} \\ &= \frac{\sum_{j=1}^{t} x_j \cdot y_j}{\sqrt{\sum_{j=1}^{t} x_j^2 \cdot \sum_{j=1}^{t} y_j^2}} \end{aligned}$$

$x \cdot y$ is the dot product of x and y, and $\|x\|$ is the Euclidean norm of x

From Angle to Cosine



General Idea of Cosine Similarity-based Document Search

- Document d is represented as a vector $\vec{d}$ in the vector space
- Query q is represented as a vector in the same vector space
- The cosine similarity between the document vector and the query vector is computed
- High cosine similarity indicates a strong match between the document and the query

Issues:

- It does not consider the order of terms in the document
- It depends heavily on the representation method of the document and query vectors

Summary

- Questions?
- Discussion?