CS4051

Information Retrieval

Week 05

Muhammad Rafi

February 20, 2023

Scoring, Term weighting & Vector Space Model (VSM)

Chapter No. 6

Agenda

- Limitation of Boolean Retrieval Model
- Ranked Retrieval /Scoring
- Scoring between Query (q) and Documents (d_i)
- Zone Indexing & Learning Zone Weights
- Vector space model

Boolean Model

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

Ranked Retrieval

- Rather than a set of documents satisfying a query expression, in 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

Score –ranking

- 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?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match".

Query –Matching Documents

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

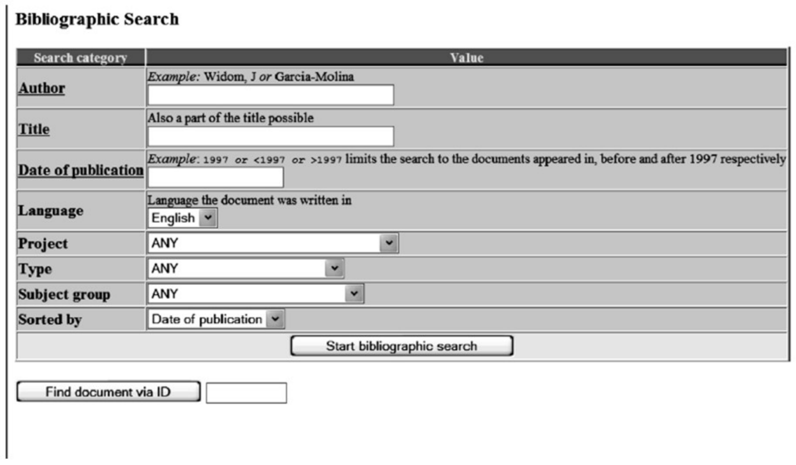
First Attempt- Jaccard Coefficient

- Recall from Lecture 3: A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| / |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$
- *A* and *B* don't have to be the same size.
- Always assigns a number between 0 and 1.

Parametric Search

- Documents
  - Data
  - Metadata
- A parametric search provide IR based on Parameters.
- Example:
  - Find documents which contains “computational linguistics” in title, “Manning” in author and “Parameters” in the body of the text.

Example



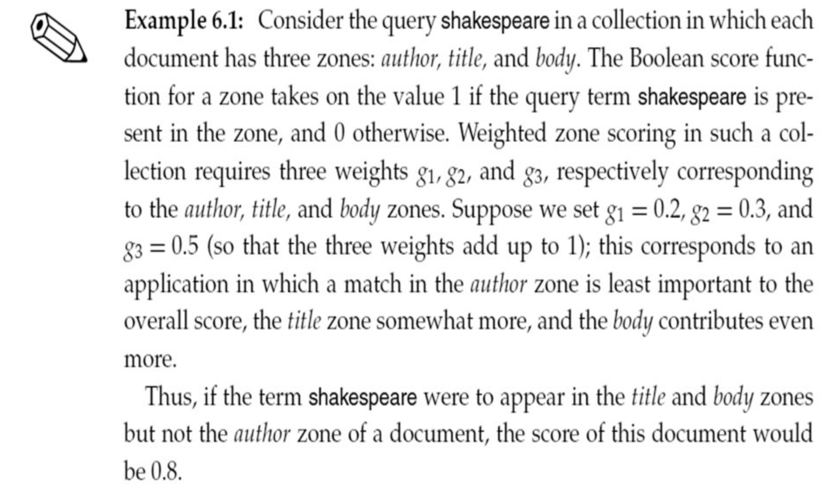
Bibliographic Search

Search category	Value
Author	*Example:* Widom, J *or* Garcia-Molina
Title	Also a part of the title possible
Date of publication	*Example*: `1997` *or* `<1997` *or* `>1997` limits the search to the documents appeared in, before and after 1997 respectively
Language	Language the document was written in English
Project	ANY
Type	ANY
Subject group	ANY
Sorted by	Date of publication

Weighted Zone Scoring

- Different fields/zones may have different importance in evaluating how a document matches a query
- For a query q and a document d, weighted zone scoring assigns to pair (q, d) a score in range [0, 1] by computing a linear combination of zone scores
- Suppose each document has l zones, let $g_1, \ldots, g_l \in [0, 1]$ such that $\Sigma_{i=1}^{l} g_i = 1$
  - Each field/zone of the document contributes a Boolean value – let s_i be the Boolean score denoting a match or absence between q and the i-th zone
  - The weighted zone score is $\Sigma_{i=1}^{l} g_i \times s_i$

Weighted Zone Indexing



✎ **Example 6.1:** Consider the query shakespeare in a collection in which each document has three zones: *author, title,* and *body*. The Boolean score function for a zone takes on the value 1 if the query term shakespeare is present in the zone, and 0 otherwise. Weighted zone scoring in such a collection requires three weights g_1, g_2, and g_3, respectively corresponding to the *author, title,* and *body* zones. Suppose we set $g_1 = 0.2$, $g_2 = 0.3$, and $g_3 = 0.5$ (so that the three weights add up to 1); this corresponds to an application in which a match in the *author* zone is least important to the overall score, the *title* zone somewhat more, and the *body* contributes even more.

Thus, if the term shakespeare were to appear in the *title* and *body* zones but not the *author* zone of a document, the score of this document would be 0.8.

Learning Weights for Zone Scoring

- Using training examples that have been judged editorially
- Each training example is a tuple consisting of a query q and a document d, and a relevance judgment for d on q
  - The judgment can be binary – relevant or not
  - A judgment score can also be used
- Compute the weights such that the learned scores approximate the relevance judgments as much as possible
  - An optimization problem

Example – Learning Weights

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

Figure 6.5 An illustration of training examples.

Learning Weights for Zone Scoring

We now consider a simple case of weighted zone scoring, where each document has a *title* zone and a *body* zone. Given a query q and a document d, we use the given Boolean match function to compute Boolean variables $s_T(d, q)$ and $s_B(d, q)$, depending on whether the title (respectively, body) zone of d matches query q. For instance, the algorithm in Figure 6.4 uses an AND of the query terms for this Boolean function. We will compute a score between 0 and 1 for each (document, query) pair using $s_T(d, q)$ and $s_B(d, q)$ by using a constant $g \in [0, 1]$, as follows:

$$score\,(d, q) = g \cdot s_T(d, q) + (1 - g)s_B(d, q).$$

We now describe how to determine the constant g from a set of *training examples*, each of which is a triple of the form $\Phi_j = (d_j, q_j, r(d_j, q_j))$. In each training example, a given training document d_j and a given training query q_j are assessed by a human editor who delivers a relevance judgment $r(d_j, q_j)$ that is either *relevant* or *nonrelevant*. This is illustrated in Figure 6.5, where seven training examples are shown.

Learning Weights for Zone Scoring

For each training example Φ_j we have Boolean values $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$ that we use to compute a score from (6.2)

$$score\,(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g)s_B(d_j, q_j).$$

We now compare this computed score with the human relevance judgment for the same document–query pair (d_j, q_j); to this end, we quantize each *relevant* judgment as a 1 and each *nonrelevant* judgment as a 0. Suppose that we define the error of the scoring function with weight g as

$$\varepsilon(g, \Phi_j) = (r(d_j, q_j) - score\,(d_j, q_j))^2,$$

where we have quantized the editorial relevance judgment $r(d_j, q_j)$ to 0 or 1. Then, the total error of a set of training examples is given by

$$\sum_j \varepsilon(g, \Phi_j).$$

Learning Weights for Zone Scoring

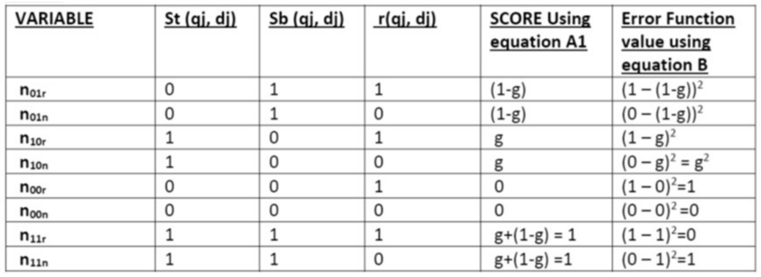
We begin by noting that for any training example Φ_j for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$, the score computed by Equation (6.2) is $1 - g$. In similar fashion, we may write down the score computed by Equation (6.2) for the three other possible combinations of $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$; this is summarized in Figure 6.6.

Let n_{01r} (respectively, n_{01n}) denote the number of training examples for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$ and the editorial judgment is *relevant* (respectively, *nonrelevant*). Then the contribution to the total error in Equation (6.4) from training examples for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$ is

$$[1 - (1 - g)]^2 n_{01r} + [0 - (1 - g)]^2 n_{01n}.$$

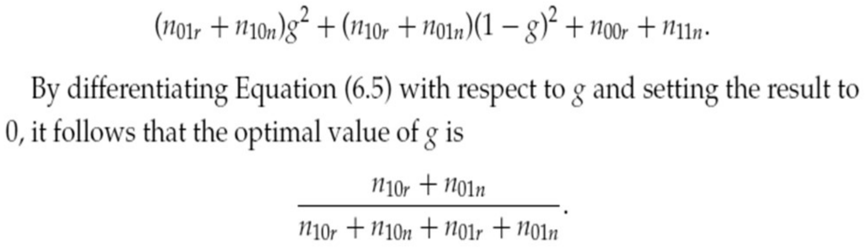
By writing in similar fashion the error contributions from training examples of the other three combinations of values for $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$ (and extending the notation in the obvious manner), the total error corresponding to Equation (6.4) is

Learning Weights for Zone Scoring



VARIABLE	St (qj, dj)	Sb (qj, dj)	r(qj, dj)	SCORE Using equation A1	Error Function value using equation B
n_{01r}	0	1	1	(1-g)	$(1 - (1-g))^2$
n_{01n}	0	1	0	(1-g)	$(0 - (1-g))^2$
n_{10r}	1	0	1	g	$(1 - g)^2$
n_{10n}	1	0	0	g	$(0 - g)^2 = g^2$
n_{00r}	0	0	1	0	$(1 - 0)^2=1$
n_{00n}	0	0	0	0	$(0 - 0)^2 =0$
n_{11r}	1	1	1	g+(1-g) = 1	$(1 - 1)^2=0$
n_{11n}	1	1	0	g+(1-g) =1	$(0 - 1)^2=1$

Learning Weights for Zone Scoring



$$(n_{01r} + n_{10n})g^2 + (n_{10r} + n_{01n})(1 - g)^2 + n_{00r} + n_{11n}.$$

By differentiating Equation (6.5) with respect to g and setting the result to 0, it follows that the optimal value of g is

$$\frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}.$$

Bag of Words

- In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.
- A document is simply the collection of terms/words/lexemes with frequencies.
- Similarly a query is also transformed as BoW representation.

Bag of Words

- Example

  D1: {John likes to watch movies. Mary likes movies too. }

  D2: {John also likes to watch football games.}

  D3: { Mary does not like football games }
- Bow Representation

  D1: {John, like,watch,movie,Mary,like,movie}

  D2: {John,like,watch,football,game}

  D3: { Mary,like,football,game }
- Query Term

  Q: {like football game}

Bag of Words

- Scores

  Common Terms in document and query over total terms

  (D1,Q) = (D1 ∩ Q) / (D1 U Q) = 1/8

  (D2,Q)= 3/5

  (D3,Q)= 3/4
- Issues

  Document size affects the overall similarity.

  Frequency of common terms are ignored.

  Similarity is only based on common terms, which are treated as independent of each other.

  Vector Space Model is the answer to many of these problems.

Vector Space Model

- Vector Space Model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, terms of a document.
- It is used in information filtering, information retrieval, indexing and relevancy rankings.
- Its first used was in the SMART Information Retrieval System.

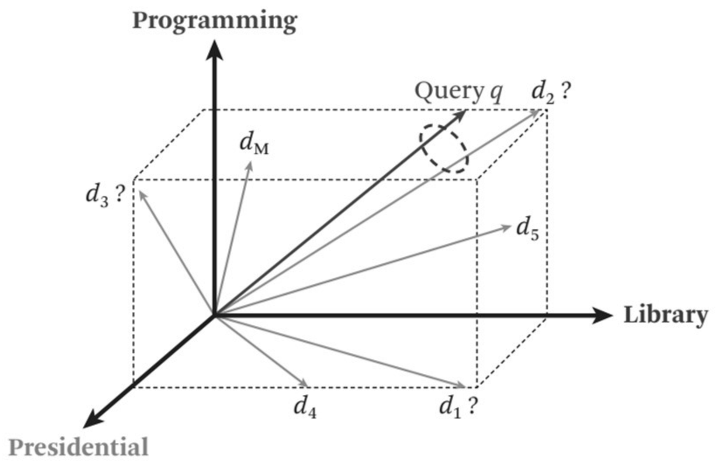
Vector Space Model

- The basic idea of Vector Space Model is to have a high-dimensional hyper space where each term is a dimension and document and query can be represented as vector in this space.
- The vector space (VS) retrieval model is a simple, yet effective method of designing ranking functions for information retrieval.
- It assume relevance is roughly correlated to similarity between a document and a query. The distance or angle between the query vector and document vector can serve this purpose.

Vector Space Model

- Assumptions of Vector Space Model
  - All the terms are features and they are all independent
  - The vocabulary (V) is limited
  - The vocabulary formed a space of |v| dimensions.
  - We can place document vector in this space
  - We can place query vector in this space
  - Relevance can be computed as distance or angle between vectors of query and document
  - It is a generic framework can be easily implemented on computer.

Vector Space Model



Vector Space Model

- **Instantiations of VSM**
- The vector space model(VSM) is really a framework.
- It did not say how we place a document vector or query vector into this space.
- How to instantiate a vector space model. We need to do three things:
  - Define the dimensions (the concept of what a document is);
  - Decide how to place documents and queries as vectors in the vector space;
  - Define the similarity between two vectors

Vector Space Model

- **Instantiations of VSM**
- The simplest instantiation of the vector space model is using a bit vector for each term.
- Each dimension is represents as 1 or 0, if the corresponding word is present in the document or query it is 1 otherwise it is 0.
- Similarity can be dot product between vectors.

$q = (x_1, \ldots, x_N)$ $\quad x_i, y_i \in \{0, 1\}$

1: word W_i is **present**

$d = (y_1, \ldots, y_N)$ $\quad$ **0:** word W_i is **absent**

$$Sim(q, d) = q.d = x_1 y_1 + \cdots + x_N y_N = \sum_{i=1}^{N} x_i y_i$$

Vector Space Model

- **Instantiations of VSM**
  - **Problem :** We saw the bit vector representation essentially counts how many unique query terms match the document.

Vector Space Model

- **Term frequency**
  - Term frequency – represent how many times a term appears in a document (TF $_{(t,d)}$)
  - It represents a local feature of each document
  - It can be used as a weight to VSM
- **Cumulative frequency**
  - How many time a term appears in entire collection? $CF_{(t)}$
- **Document Frequency**
  - How many documents contains a term t.? $DF_{(t)}$
- **Inverse Document Frequency**
  - Inverse document frequency is a normalized global feature and it is represents as IDF = log(DF/N)

Vector Space Model

- **Improved Instantiation**
  - A natural thought is to consider multiple occurrences of a term in a document as opposed to binary representation;
  - we should consider the TF instead of just the absence or presence. T F(w, d) = count(w, d).
  - Document Frequency is the number of document contains term t.
  - DF(w,D)= count(d) that contains w.
  - This particular idea is called the inverse document frequency (IDF). It is a very important signal used in modern retrieval functions. The document frequency is the count of documents that contain a particular term.

Vector Space Model

- **Improved Instantiation**
  - Heuristic - One of the popular and effective weighting scheme is term frequency * inverse document frequency

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t.$$

    - Highest when t occurs many times within a small number of documents (thus lending high discriminating power to those documents);
    - Lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
    - Lowest when the term occurs in virtually all documents.

Vector Space Model

- **Efficient Calculation / Implementation**
  - The similarity between the query **q** and document **d** can be computed as

    Cos(q,d) = (q . d) / (|q| * |d|)
  - How to compute it efficiently?
  - Unit vector treatment
  - Cos(Θ) ~ Θ treatment

Vector Space Model

- Calculating Scores

```
CosineScore($q$)
float $Scores[N] = 0$
Initialize $Length[N]$
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 $Scores[d] += wf_{t,d} \times w_{t,q}$
Read the array $Length[d]$
for each $d$
do $Scores[d] = Scores[d] / Length[d]$
return Top $K$ components of $Scores[]$
```

Vector Space Model

- Calculating Scores

```
FASTCOSINESCORE($q$)
 1  float $Scores[N] = 0$
 2  for each $d$
 3  do Initialize $Length[d]$ to the length of doc $d$
 4  for each query term t
 5  do calculate $w_{t,q}$ and fetch postings list for $t$
 6      for each pair$(d, \mathrm{tf}_{t,d})$ in postings list
 7      do add $\mathrm{wf}_{t,d}$ to $Scores[d]$
 8  Read the array $Length[d]$
 9  for each $d$
10  do Divide $Scores[d]$ by $Length[d]$
11  return Top $K$ components of $Scores[]$
```

Vector Space Model

- Advantages
  - Simple model based on linear algebra
  - Term weights not binary
  - Allows computing a continuous degree of similarity between queries and documents
  - Allows ranking documents according to their possible relevance
  - Allows partial matching
  - Last 40 years of working knowledge

Vector Space Model

- Disadvantages
  - The order in which the terms appear in the document is lost in the vector space representation.
  - Theoretically assumes terms are statistically independent.
  - Search keywords must precisely match document terms; word substrings might result in a "false positive match“
  - Semantic sensitivity; documents with similar context but different term vocabulary won't be associated, resulting in a "false negative match".

Sub linear “ tf ” Scaling

- Term Frequency can be scaled in different ways (tf)
- Log Linear $1 + \log(\mathrm{tf}_{t,d})$
- Augmented $0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$
- Maximum Frequency normalization

Document Frequency

- Document Frequency – How many documents contains a given term? (df)?
- Scaling the df $\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}.$
- Augmented $\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$

Weighting Schemes

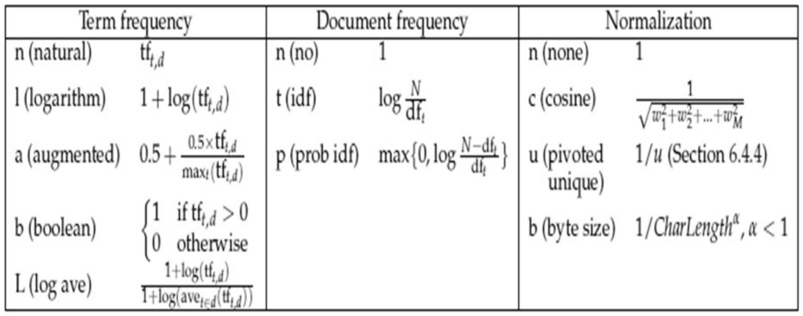
- One of the popular and effective weighting scheme is
  - term frequency * inverse document frequency

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t.$$

Tf*Idf Scheme

- The tf-idf weighting scheme assigns term t a weight in document d that is
  - Highest when t occurs many times within a small number of documents (thus lending high discriminating power to those documents);
  - Lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
  - Lowest when the term occurs in virtually all documents.

Different Variants



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)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

▶ **Figure 6.15** SMART notation for tf-idf variants. Here *CharLength* is the number of characters in the document.

SMART System

- The SMART system did a rigor research on different schemes, for document and query they have used mnemonic ddd.qqq
- The first letter in each triplet specifies the term frequency component of the weighting, the second the document frequency component, and the third the form of normalization used.
- For example, a very standard weighting scheme is lnc.ltc, where the document vector has log-weighted term frequency, no idf (for both effectiveness and efficiency reasons), and cosine normalization, while the query vector uses log-weighted term frequency, idf weighting, and cosine normalization.