IR Modeling

Modern Information Retrieval

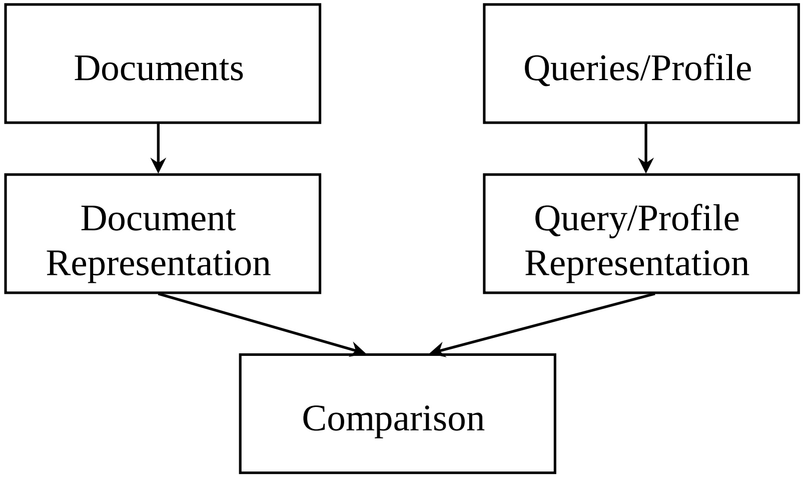
by **R. Baeza-Yates and B. Ribeiro-Neto**

(Chapter 2)

Introduction

Remember: Judging Relevance between document and query(or profile) is important!

» The central problem regarding IR systems is the issue of predicting which documents are relevant and which are not to a query (or a profile) by comparison.



Taxonomy of IR Models

» Boolean: set theoretic

» Vector: algebraic

Revisited: Retrievals in two ways

Document Search (Ad hoc Retrieval)

- » the documents in the collection remain relatively static while new information needs are submitted to the system
- » construction of information needs as user queries

Document Filtering (Routing)

- » information need remains relatively static while new documents come into the system
- » construction of the static information need as user profile

Basic Concepts

In the classic models

- » each document is described as a document vector for a set of representative keywords called *index terms*
- » index terms are mainly nouns and sometimes other part-of-speeches
- » distinct index terms have varying relevance weights
- » index term weights are usually assumed to be mutually independent

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

weight of index term “caesar” for “Hamlet” play

Boolean Model (1/2)

Data retrieval model based on binary decision criterion

A query is a Boolean expression which can be represented as a disjunction of conjunctive vectors

- example query: (intelligent $\wedge$ system) $\vee$ (information $\wedge$ retrieval)

Advantage

- clean formalism, simplicity : explained in the next slide

Disadvantage

- exact matching may lead to retrieval of too few or too many documents

Boolean Model (2/2)

Example

doc$_1$:

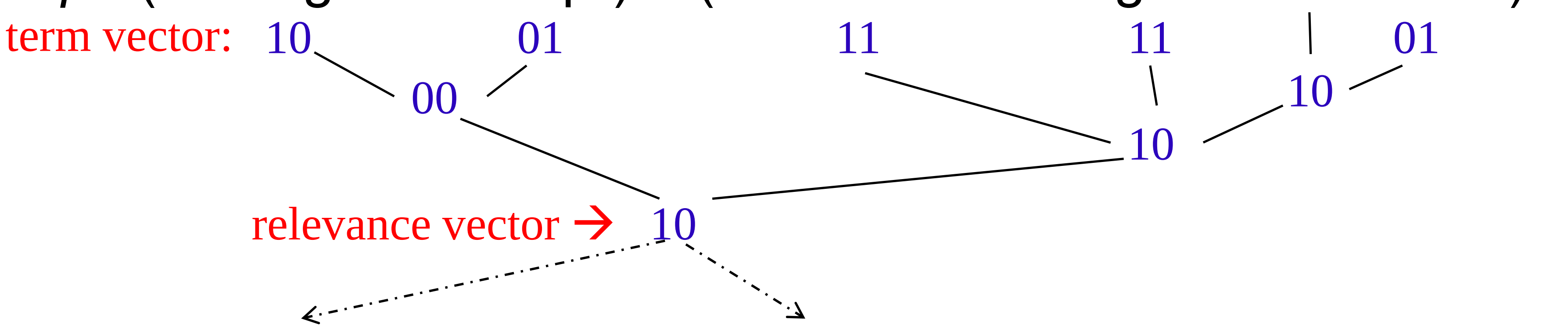
d_1= {“intelligent”,“information”,“retrieval”,“learning”,“agent”}

doc$_2$:

d_2= {“information”,“management”,“travel”,“agent”,“map”}

query :

q = (“intelligent”$\wedge$“map”) $\vee$ (“information”$\wedge$“agent”$\wedge$ $\neg$“travel”)



Therefore, d_1 is relevant to q but d_2 is not

Vector Model (1/6)

Index terms are assigned non-binary weights

Index term weights are used to compute the degree of similarity between documents and the user query

Then, retrieved documents are sorted in decreasing order.

Definition

For the vector model, the weight $w_{i,j}$ is associated with term k_i and document d_j (or query *q*)

$$\vec{d}_j = (w_{1,j}, w_{2,j}, \cdots, w_{t,j})$$

$$\vec{q} = (w_{1,q}, w_{2,q}, \cdots, w_{t,q})$$

Vector Model (2/6)

Degree of similarity

$$sim(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \times |\vec{q}|}$$

$$= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} = \cos\theta$$

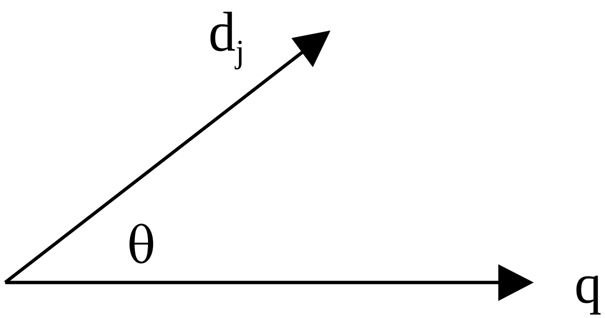


Figure 2.4 The cosine of θ is adopted as sim(d_j,q)

Vector Model (3/6)

Salton

- IR can be defined as clustering a document collection into a relevant subcollection and a irrelevant one
- Intra-cluster similarity: *tf* factor (term frequency)
- inter-cluster dissimilarity: *idf* factor (inverse document frequency)

Definition (1/2)

- Term frequency is based on a simple assumption "Frequent terms are more informative than rare terms **within a document**"

  $freq_{i,j}$ = number of appearances of term k_i in document d_j

- Normalized term frequency and Log term frequency

$$f_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}}$$

$$w_{i,j} = \begin{cases} 1 + \log_{10} freq_{i,j}, & \text{if } freq_{i,j} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Vector Model (4/6)

Definition

- Inverse document frequency is based on a simple assumption "Rare terms are more informative than frequent terms **across documents**"

$$idf_i = \log \frac{N}{n_i}$$

N = total number of documents
n_i = number of documents containing a term k_i

- document term-weighting scheme

$$w_{i,j} = freq_{i,j} \times idf_i$$ often called tf×idf scheme

- query-term weighting scheme

$$w_{i,q} = (0.5 + \frac{0.5\, freq_{i,q}}{\max_l freq_{l,q}}) \times \log \frac{N}{n_i}$$

Vector Model (5/6)

Advantages

- its term-weighting scheme improves retrieval performance
- its partial matching strategy allows retrieval of documents that approximate the query conditions
- its cosine ranking formula sorts the documents according to their degree of similarity to the query

Disadvantage

- The assumption of mutual independence between index terms may be unrealistic in practice

Vector Model (6/6)

Example: when using tf×idf scheme

doc$_1$: d_1= {"intelligent"^{2},"information","agent"^{2}}

doc$_2$: d_2= {"information"^{2},"travel"^{3},"agent"}

doc$_3$: d_3= {"intelligent","mobile" ^{3},"robot"^{3}}

query : q = {"mobile","agent"}

a sequence of all terms = <intelligent,information,agent,travel,mobile,robot>

$$\vec{d}_1 = (w_{1,1}, \cdots, w_{6,1}) = (2\times\log\frac{3}{2}, 1\times\log\frac{3}{2}, 2\times\log\frac{3}{2}, 0\times\log\frac{3}{1}, 0\times\log\frac{3}{1}, 0\times\log\frac{3}{1})$$

$$\vec{d}_2 = (w_{1,2}, \cdots, w_{6,2}) = (0\times\log\frac{3}{2}, 2\times\log\frac{3}{2}, 1\times\log\frac{3}{2}, 3\times\log\frac{3}{1}, 0\times\log\frac{3}{1}, 0\times\log\frac{3}{1})$$

$$\vec{d}_3 = (w_{1,3}, \cdots, w_{6,3}) = (1\times\log\frac{3}{2}, 0\times\log\frac{3}{2}, 0\times\log\frac{3}{2}, 0\times\log\frac{3}{1}, 3\times\log\frac{3}{1}, 3\times\log\frac{3}{1})$$

$$\vec{q} = (w_{1,q}, \cdots, w_{6,q}) = (\,(0.5+\frac{0.5\times 0}{1})\times\log\frac{3}{2}, (0.5+\frac{0.5\times 0}{1})\times\log\frac{3}{2}, (0.5+\frac{0.5\times 1}{1})\times\log\frac{3}{2},$$
$$(0.5+\frac{0.5\times 0}{1})\times\log\frac{3}{1}, (0.5+\frac{0.5\times 1}{1})\times\log\frac{3}{1}, (0.5+\frac{0.5\times 0}{1})\times\log\frac{3}{1}))$$

Vector Model (6/6)

Example (Continued)

$$\vec{d}_1 = (0.35,\ 0.18,\ 0.35,\ 0,\ 0,\ 0)$$

$$\vec{d}_2 = (0,\ 0.35,\ 0.18,\ 1.43,\ 0,\ 0)$$

$$\vec{d}_3 = (0.18,\ 0,\ 0,\ 0,\ 1.43,\ 1.43)$$

$$\vec{q} = (0.09,\ 0.09,\ 0.18,\ 0.24,\ 0.48,\ 0.24)$$

$$sim(d_1, q) = \frac{\vec{d}_1 \cdot \vec{q}}{|\vec{d}_1| \times |\vec{q}|} = \cdots$$

$$sim(d_2, q) = \frac{\vec{d}_2 \cdot \vec{q}}{|\vec{d}_2| \times |\vec{q}|} = \cdots$$

$$sim(d_3, q) > sim(d_1, q) > sim(d_2, q)$$

$$sim(d_3, q) = \frac{\vec{d}_3 \cdot \vec{q}}{|\vec{d}_3| \times |\vec{q}|} = \cdots$$

$\therefore \quad sim(d_3, q)$ is largest, so d_3 is most relevant to q !

For more detailed explanation of term frequency, document frequency, and cosine similarity…

Term frequency *tf*

The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.

We want to use *tf* when computing query-document match scores. But how?

Raw term frequency is not what we want:

» A document with 10 occurrences of the term is more relevant than a document with one occurrence of the term.

» But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.

Log term frequency weighting

Sometimes we can use the log frequency weight of term t in d

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

The score is 0 if none of the query terms is present in the document.

0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

Simply, we can caculate the score for a document-query pair by just summing over terms *t* in both *q* and *d*:

Simple similarity score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

Document frequency

Rare terms are more informative than frequent terms

Consider a term in the query that is rare in the document collection (e.g., *arachnocentric*)

A document containing this term is very likely to be relevant to the query *arachnocentric*

→ We want a high weight for rare terms like *arachnocentric*.

Document frequency, continued

Consider a query term that is frequent in the document collection (e.g., *high, increase, line*)

A document containing such a term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.

→ For frequent terms, we want positive weights for words like *high, increase, and line*, but lower weights than for rare terms like *arachnocentric*.

We will use document frequency (df) to capture this in the score.

df ($\leq N$) is the number of documents that contain the term

idf weight

- df_t is the document frequency of t that is defined as the number of documents that contain t
  - » df is a measure of the non-informativeness of t
- We define the idf (inverse document frequency) of t by

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t$$

  - » idf is a measure of the informativeness of t
  - » We use log N/df_t instead of N/df_t to "damping" the effect of idf because informativeness does not increase proportionally with inverse document frequency(N/df_t)

Will turn out the base of the log is immaterial.

idf example, suppose N= 1 million

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

There is one *idf* value for each term *t* in a Shakespeare collection.

Collection vs. Document frequency

The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences within a document.

Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

Which word is a better search term (and should get a higher weight)? Ans: Insurance Why? $\text{Idf}_{\text{insurance}} > \text{idf}_{\text{try}}$

tf-idf weighting

The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\mathrm{w}_{t,d} = \mathrm{tf}_{t,d} \times \log_{10} N / \mathrm{df}_t \quad \textit{or}$$
$$\mathrm{w}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10} N / \mathrm{df}_t$$

Best known weighting scheme in information retrieval

Note: the “-” in tf-idf is a hyphen, not a minus sign!

Alternative names: tf.idf, tfxidf

Increases with the number of occurrences within a document

Increases with the rarity of the term across documents in the collection

Binary(Boolean) → count(tf) → weight matrix(tf×idf scheme)

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

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: hundreds of millions of dimensions when you apply this to a web search engine

This is a very sparse vector - most entries are zero.

Queries as vectors and Ranking documents

Key idea 1: Do the similar to document for queries: represent queries as vectors in the vector space using query-term weighting scheme :

$$w_{i,q} = (0.5 + \frac{0.5\, tf_{i,q}}{\max_l tf_{l,q}}) \times \log \frac{N}{df_i} \quad \textit{used for only query vector}$$

$$\mathrm{w}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10} N / \mathrm{df}_t \quad \textit{used for document and also query vectors}$$

Key idea 2: Rank documents according to their proximity to the query in this space

proximity = similarity between vectors

proximity ≈ inverse of distance

Generally, rank documents in decreasing order of proximities

Formalizing vector space proximity

First cut of proximity: Inverse of Euclidean distance between two vector points

- (= Inverse of Euclidean distance between the end points of the two vectors)

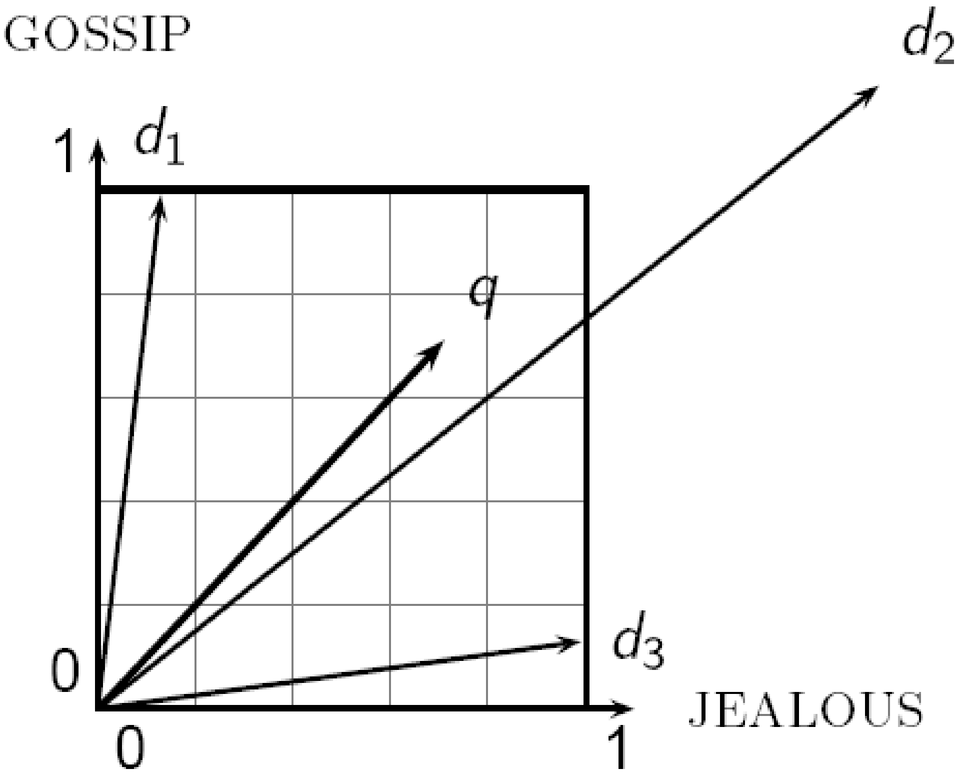
Inverse of Euclidean distance is good for proximity ?

Inverse of Euclidean distance may be a bad idea. Why?

- Because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is largest (that is, their proximity is lowest) even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of Euclidean distance

Suppose that we take a document d and append it to itself. Call this document d′ (= d|d = two same documents).

Then, “Semantically” d and d′ have the same content

The Euclidean distance between the two documents(d and d′) can be quite large, which means their proximity is quite low when we use “inverse of distance” for proximity measure

We can use the angle between the two document vectors

- The angle is 0 when having maximal similarity(proximity)
- The angle is 90 when having minimum similarity(proximity)

Key idea: Rank documents according to their angle with query in increasing order.

From angles to cosines

The following two notions are equivalent.

- Rank documents in increasing order of the angle between query vector and document vector
- Rank documents in decreasing order of cosine(query vector,document vector)

It is because *cosine* is a monotonically decreasing function for the interval [0^o, 90^o]

Before explaining cosine similarity, check Length normalization of Vector

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

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

Dividing a vector by its norm makes it a unit (length) vector

Effect on the two documents d and d′(d appended to itself) from earlier slide: they(d and d') have identical vectors after length-normalization. (→ trivial!)

cosine(query vector, document vector)

Dot product

Unit vectors after normalization

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

Length independent proximity measure

q_i is the tf-idf weight of term i in the query vector
d_i is the tf-idf weight of term i in the document vector
$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.