Retrieval Models

CiTIUS Centro Singular de Investigación en Tecnoloxías Intelixentes

Tecnologías de Gestión de Información No Estructurada

Prof. Dr. David E. Losada

USC Universidade de Santiago de Compostela

CAMPUS VIDA Campus de Excelencia Internacional

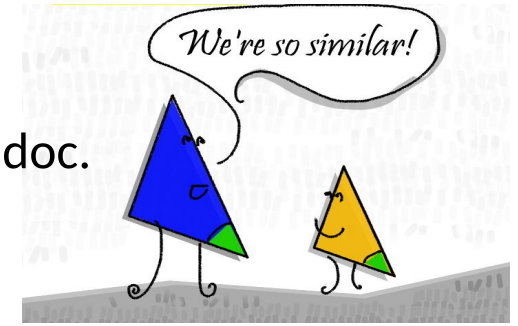
Máster Interuniversitario en Tecnologías de Análisis de Datos Masivos: Big Data

retrieval models

similarity-based

ranking function based on the similarity between query and doc.

vector space model [Salton et al. 1975]



probabilistic

queries and docs: observations from random variables

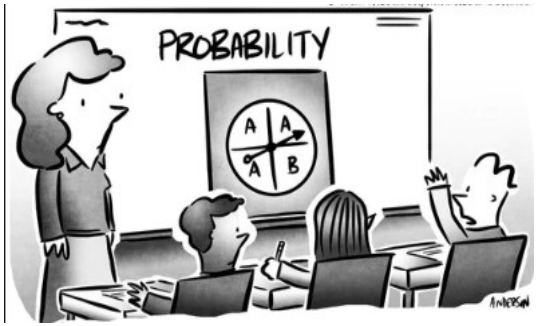
binary random variable R (relevance, indicates whether a doc is relevant to a query)

score of a doc with respect to a query: probability that R is equal to 1 given a doc and query

classic probabilistic model (60s & 70s)

language modeling (late 90s)

divergence from randomness (early 2000s)



retrieval models

probabilistic inference

associates uncertainty to inference rules.

quantifies the probability that we can show that the query follows from the document

axiomatic

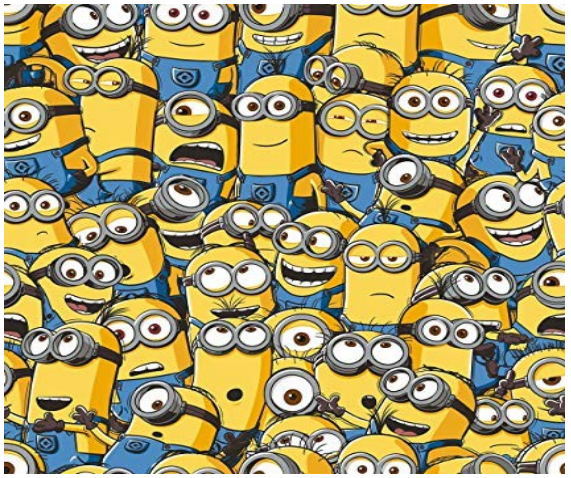
define a set of constraints that a good retrieval function should satisfy

find a good ranking function that can satisfy all the desired constraints

retrieval models

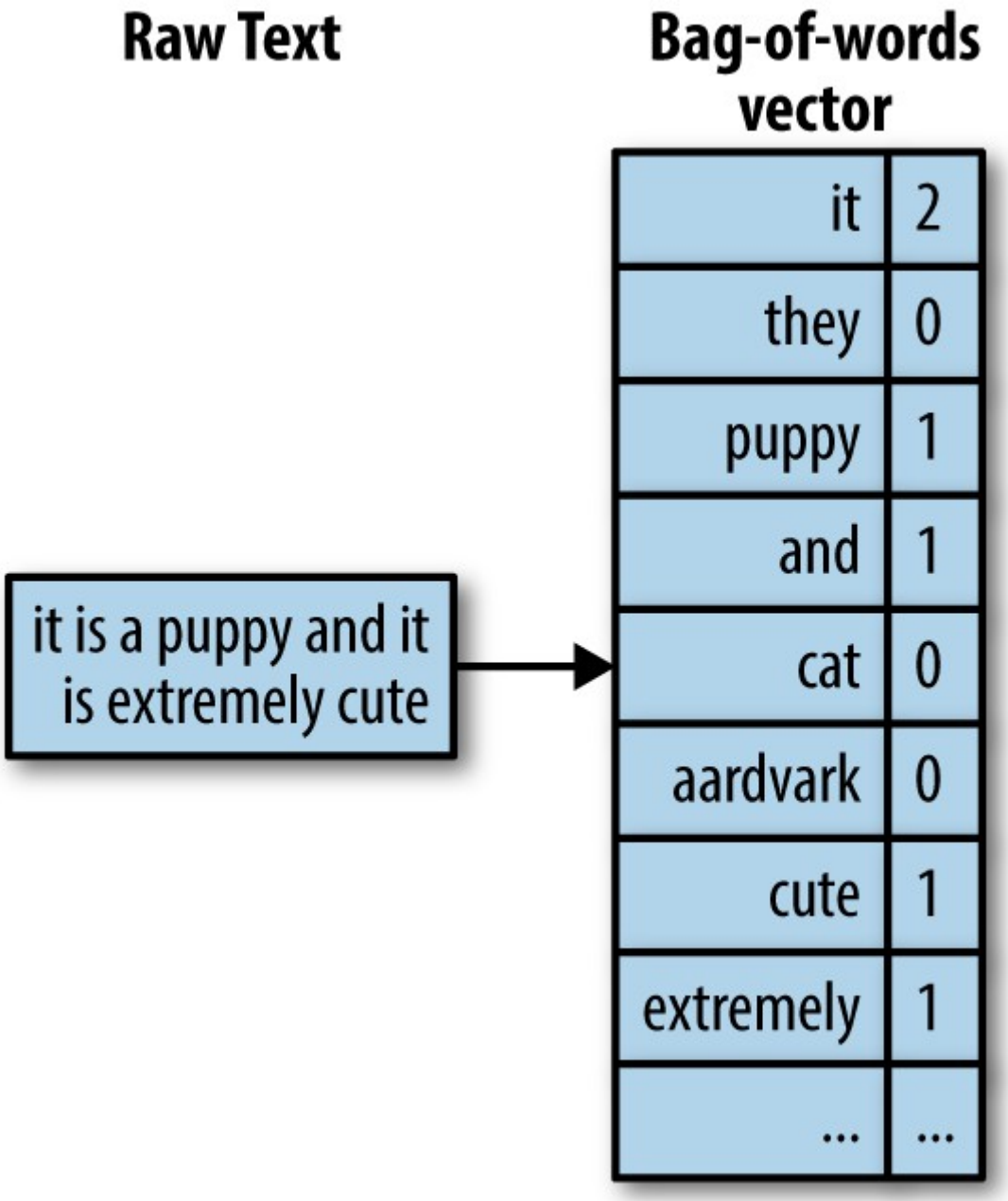
all these models are based on different thinking

but in the end the retrieval functions tend to be very similar and involve similar variables

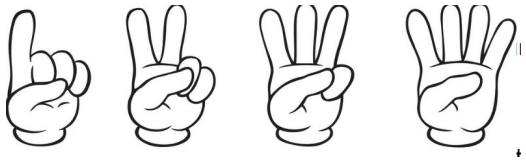


bag of words

common form of a state-of-the-art retrieval model <u>assumption</u>



term frequency (tf): how many times the query word occurs in each doc

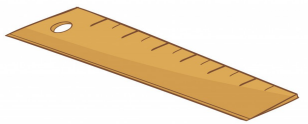


doc frequency (df): how often the word occurs in docs of the collection

attempts to characterize the popularity of the term in the collection.

matching a rare term contributes more to the overall score than matching a common term.

doc length (dl): if a term occurs in a long document many times, it is not as significant as if it occurred the same number of times in a short document



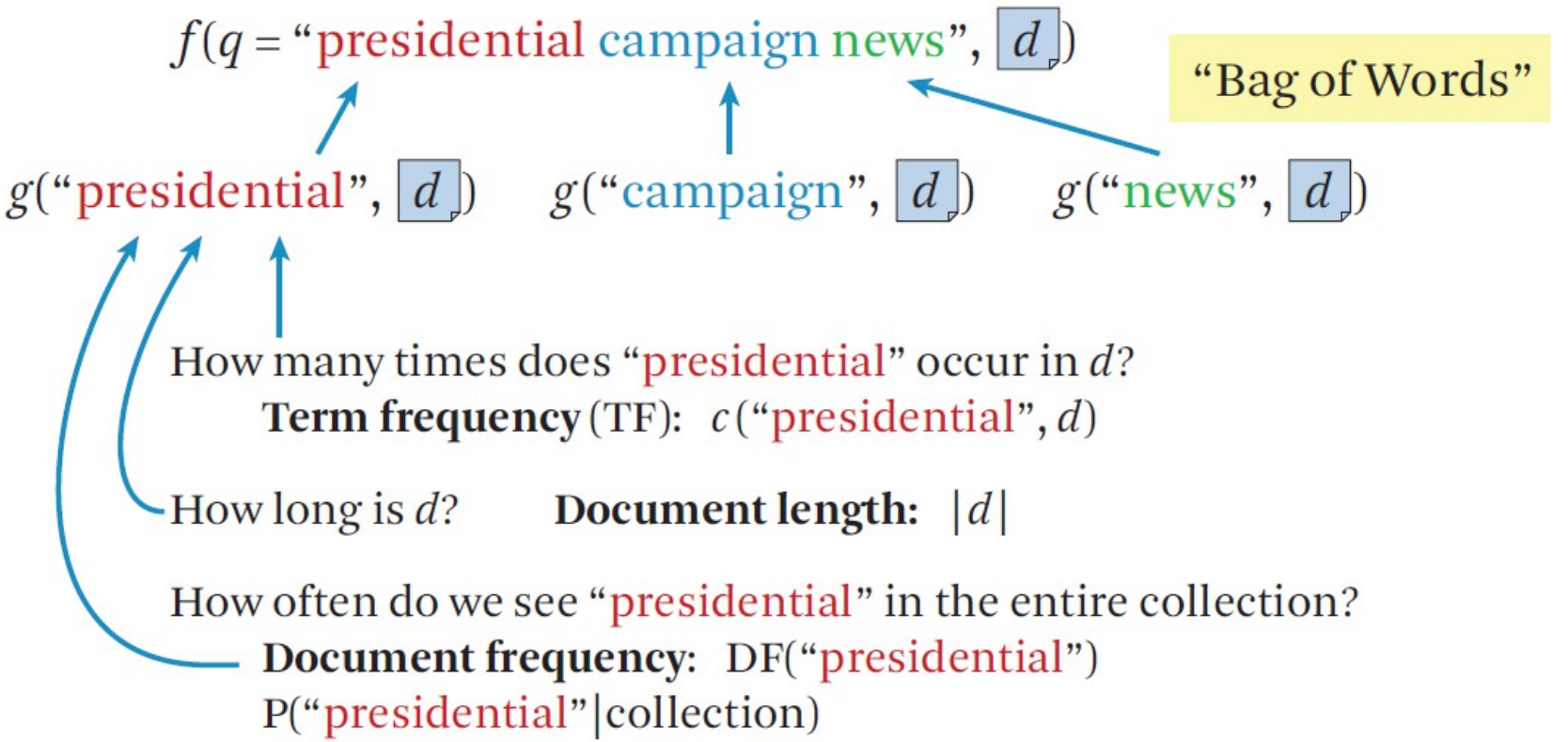


Figure 6.1 Illustration of common ideas for scoring with a bag-of-words representation.

four major state-of-the-art models

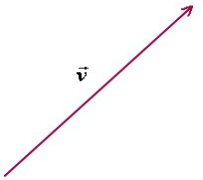
pivoted length normalization [Singhal et al. 1996]

Okapi BM25 [Robertson and Zaragoza 2009]

query likelihood [Ponte and Croft 1998]

PL2 [Amati and Van Rijsbergen 2002]

vector space retrieval model



if a doc is more similar to a query than another doc, then the first document would be assumed to be more relevant

high dimensional space

each dimension corresponds to a term

plot our docs and query in this space

representations as vectors of term magnitudes

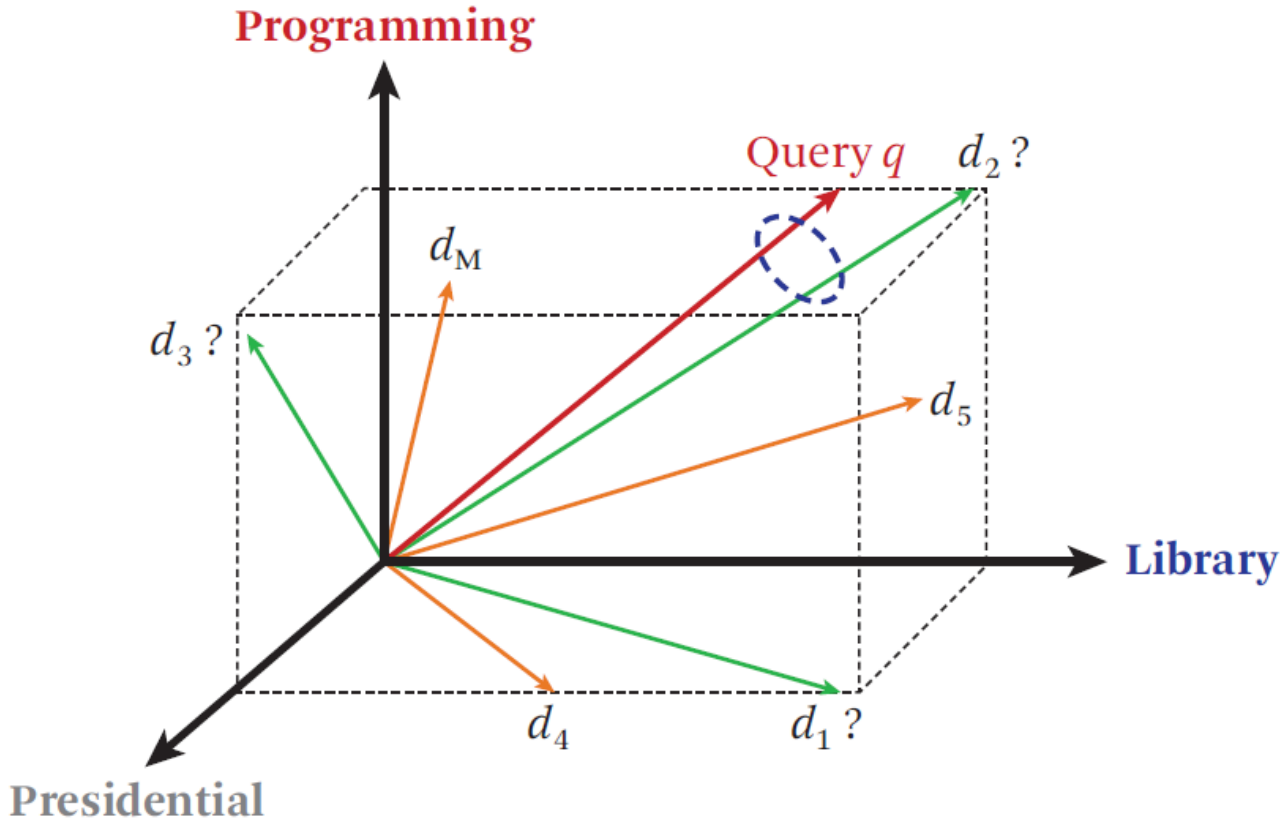
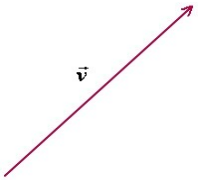


Figure 6.2 **Illustration of documents plotted in vector space.** (Courtesy of Marti Hearst)

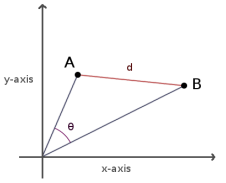
vector space retrieval model



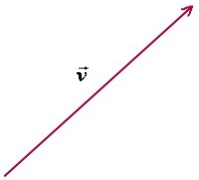
ignores everything else, including, e.g., the order of the words

not an optimal representation, but often suffices for many retrieval problems

measure the similarity between the query vector and every document vector



vector space retrieval model

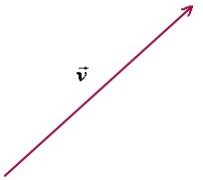


a term can be any basic concept such as a word or a phrase, n-grams of characters or any other feature representation

but typically each word is assumed to define one dimension: |V|-dimensional space (V is the vocabulary)

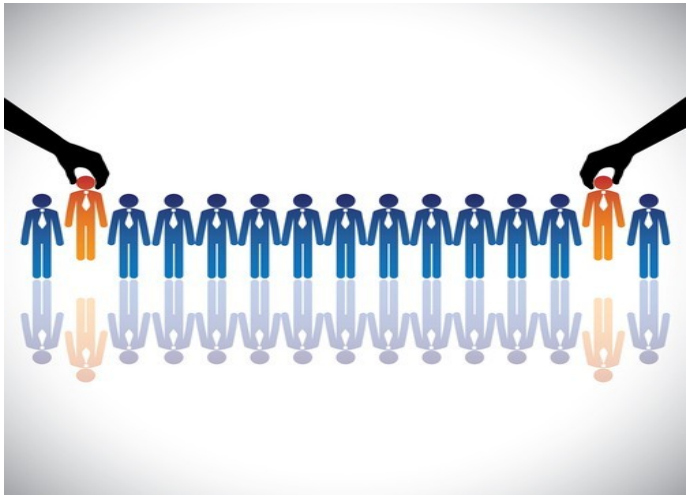
each value of each element: weight of the term

vector space retrieval model

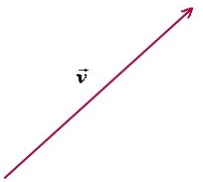


assume the terms are **orthogonal**

redundancy: two synonyms would be represented in two different dimensions



simplest VSP



BoW: each word defines a dimension

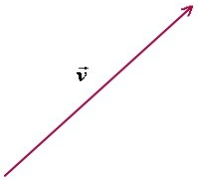
(|V| dimensions)

binary weights

similarity function: **dot product**

(counts the number of matching terms, how many unique query terms are matched in the doc)

simplest VSP



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

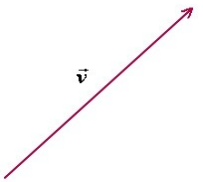
1: word W_i is **present**

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

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

Figure 6.3 Computing the similarity between a query and document vector using a bit vector representation and dot product similarity.

simplest VSP



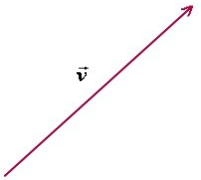
Query = "news about presidential campaign"

Ideal ranking?

d_1	... news about ...	d_4 +
d_2	... news about organic food campaign ...	d_3 +
d_3	... news of presidential campaign ...	
d_4	... news of presidential campaign ... ... presidential candidate ...	d_1 −
d_5	... news of organic food campaign ... campaign ... campaign ... campaign ...	d_2 − d_5 −

Figure 6.4 Application of the bit vector VS model in a simple example.

simplest VSP



Query = "news about presidential campaign"

d_1 | ... news about ...

d_3 | ... news of presidential campaign ...

V =	{news,	about,	presidential,	campaign,	food	...}
q =	(1,	1,	1,	1,	0,	...)
d_1 =	(1,	1,	0,	0,	0,	...)

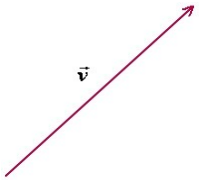
$f(q, d_1) = 1 * 1 + 1 * 1 + 1 * 0 + 1 * 0 + 0 * 0 + \ldots = 2$

d_3 =	(1,	0,	1,	1,	0,	...)

$f(q, d_3) = 1 * 1 + 1 * 0 + 1 * 1 + 1 * 1 + 0 * 0 + \ldots = 3$

Figure 6.5 Computation of the bit vector retrieval model on a sample query and corpus.

simplest VSP

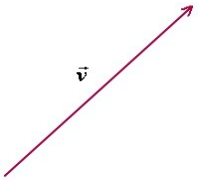


Query = "news about presidential campaign"

d_1	... news about ...	$f(q, d_1) = 2$
d_2	... news about organic food campaign ...	$f(q, d_2) = 3$
d_3	... news of presidential campaign ...	$f(q, d_3) = 3$
d_4	... news of presidential campaign ... ... presidential candidate ...	$f(q, d_4) = 3$
d_5	... news of organic food campaign ... campaign ... campaign ... campaign ...	$f(q, d_5) = 2$

Figure 6.6 Ranking of example documents using the simple vector space model.

improved VSP



$$TF(w, d) = \text{count}(w, d).$$

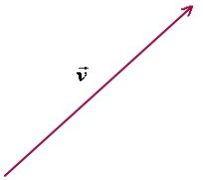
$q = (x_1, \ldots, x_N)$ $\boxed{x_i = \text{count of word } W_i \text{ in query}}$

$d = (y_1, \ldots, y_N)$ $\boxed{y_i = \text{count of word } W_i \text{ in doc}}$

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

Figure 6.7 **Frequency vector representation and dot product similarity.**

improved VSP



d_2 | ... news about organic food campaign ... | $f(q, d_2) = 3$

$q = (1, 1, 1, 1, 0, \ldots)$

$d_2 = (1, 1, 0, 1, 1, \ldots)$

d_3 | ... news of presidential campaign ... | $f(q, d_3) = 3$

$q = (1, 1, 1, 1, 0, \ldots)$

$d_3 = (1, 0, 1, 1, 0, \ldots)$

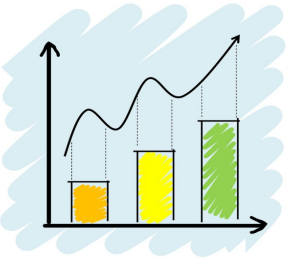
d_4 | ... news of presidential campaign ... ... presidential candidate ... | $f(q, d_4) = 4!$

$q = (1, 1, 1, 1, 0, \ldots)$

$d_4 = (1, 0, 2, 1, 0, \ldots)$

Figure 6.8 Frequency vector representation rewards multiple occurrences of a query term.

more improvements...



global statistics of terms or some other information to try to decrease the weight of common words

inverse document frequency (IDF)

document frequency is the count of documents that contain a particular term

idf

$$\mathrm{IDF}(w) = \left(\frac{M+1}{\mathrm{df}(w)}\right), \quad \text{M: \# docs in the collection}$$

$$\mathrm{IDF}(\text{about}) = \log\left(\frac{10,001}{\mathrm{df}(\text{about})}\right) = \log\left(\frac{10,001}{5000}\right) \approx 1.0$$

$$\mathrm{IDF}(\text{campaign}) = \log\left(\frac{10,001}{\mathrm{df}(\text{campaign})}\right) = \log\left(\frac{10,001}{1166}\right) \approx 3.1.$$

idf

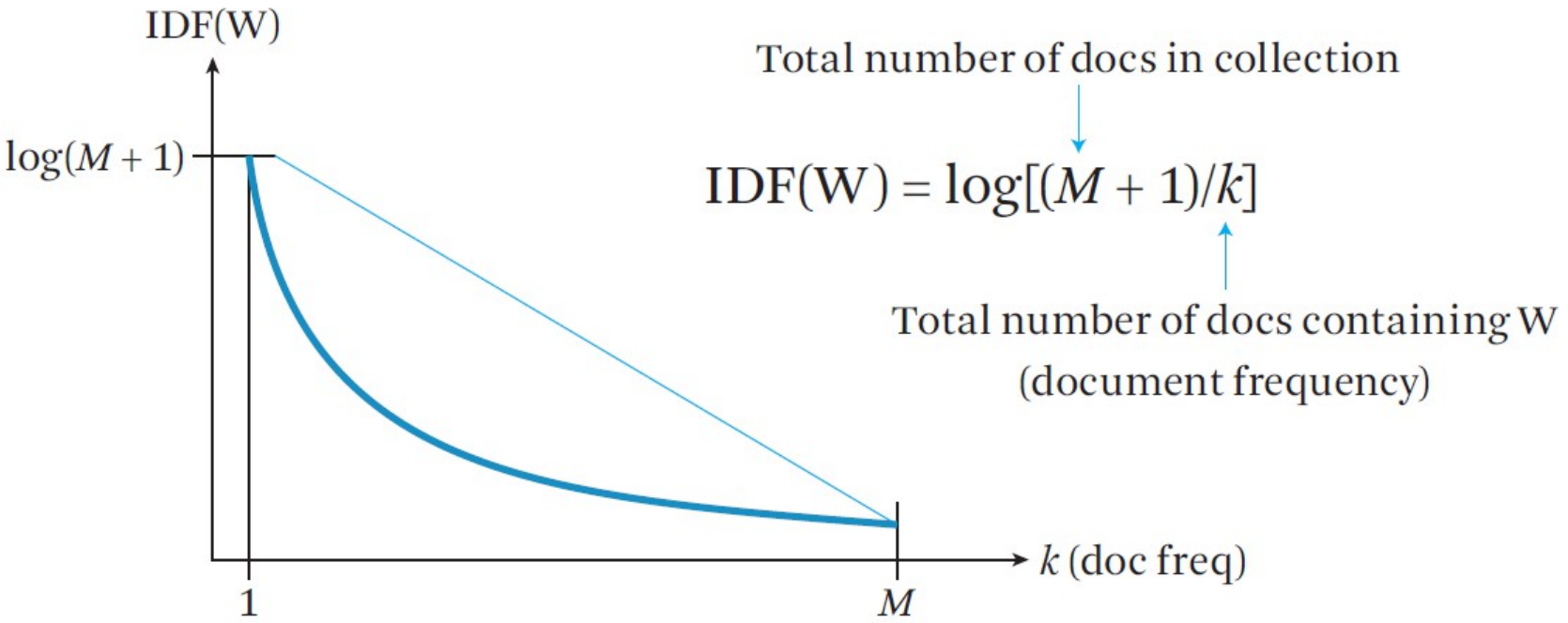


Figure 6.10 Illustration of the IDF function as the document frequency varies.

a linear function would not focus on the **discrimination** of low df terms

tf*idf

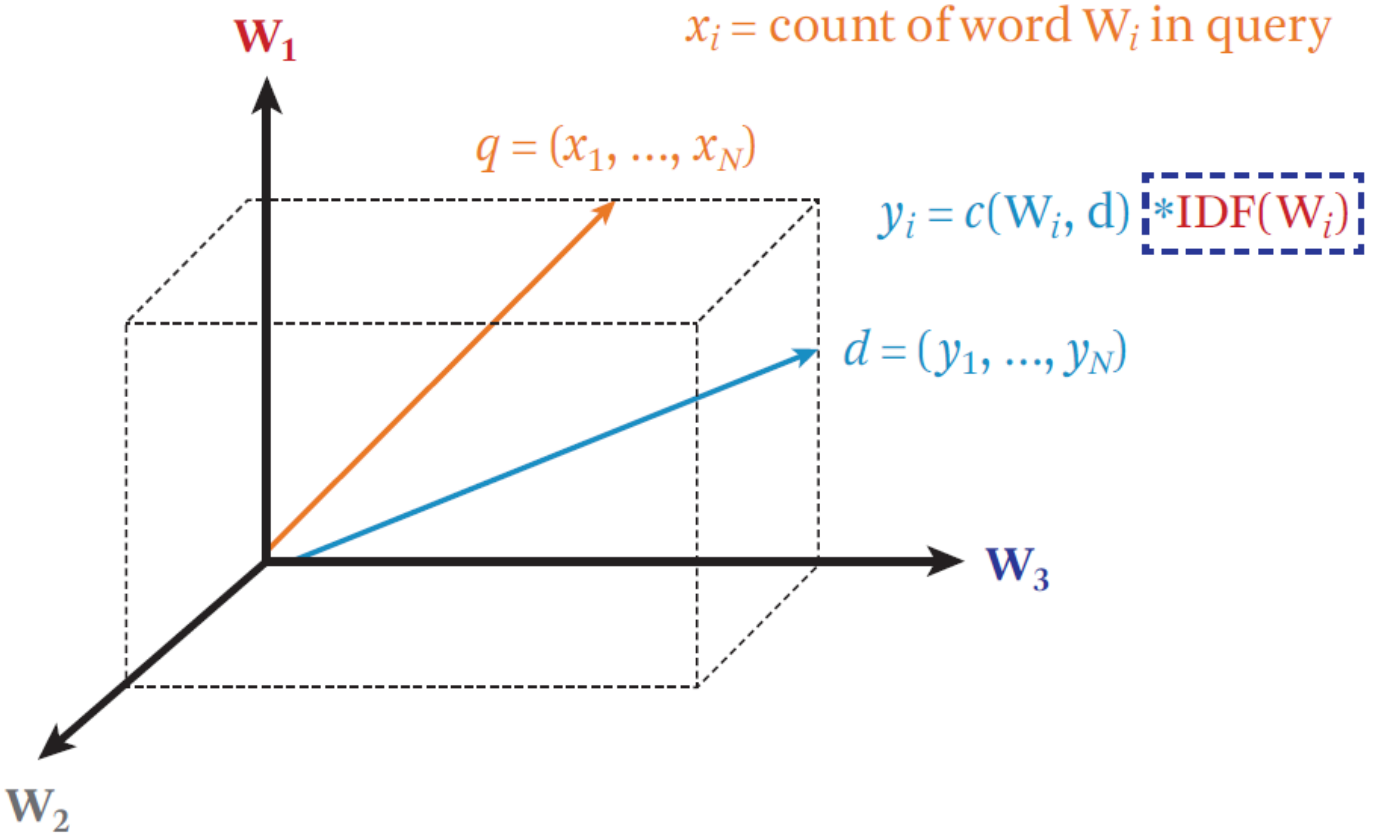


Figure 6.9 Representation of a blue document vector and red query vector with TF-IDF weighting.

tf*idf

d_2 ... news about organic food campaign ...

d_3 ... news of presidential campaign ...

$V =$	{news,	about,	presidential,	campaign,	food	...}
IDF(W) =	**1.5**	**1.0**	**2.5**	**3.1**	**1.8**	
$q =$	(1,	1,	1,	1,	0,	...)
$d_2 =$	(1 * 1.5,	1 * 1.0,	0,	1 * 3.1,	0,	...)
$q =$	(1,	1,	1,	1,	0,	...)
$d_3 =$	(1 * 1.5,	0,	1 * 2.5,	1 * 3.1,	0,	...)

$$f(q, d_2) = 5.6 \quad < \quad f(q, d_3) = 7.1$$

Figure 6.11 The impact of IDF weighting on document ranking.

tf*idf

Query = "news about presidential campaign"

d_1	... news about ...	$f(q, d_1) = 2.5$
d_2	... news about organic food campaign ...	$f(q, d_2) = 5.6$
d_3	... news of presidential campaign ...	$f(q, d_3) = 7.1$
d_4	... news of presidential campaign ... ... presidential candidate ...	$f(q, d_4) = 9.6$
d_5	... news of organic food campaign ... campaign ... campaign ... campaign ...	$f(q, d_5) = 13.9!$

Figure 6.12 **Scores of all five documents using TF-IDF weighting.**

tf*idf

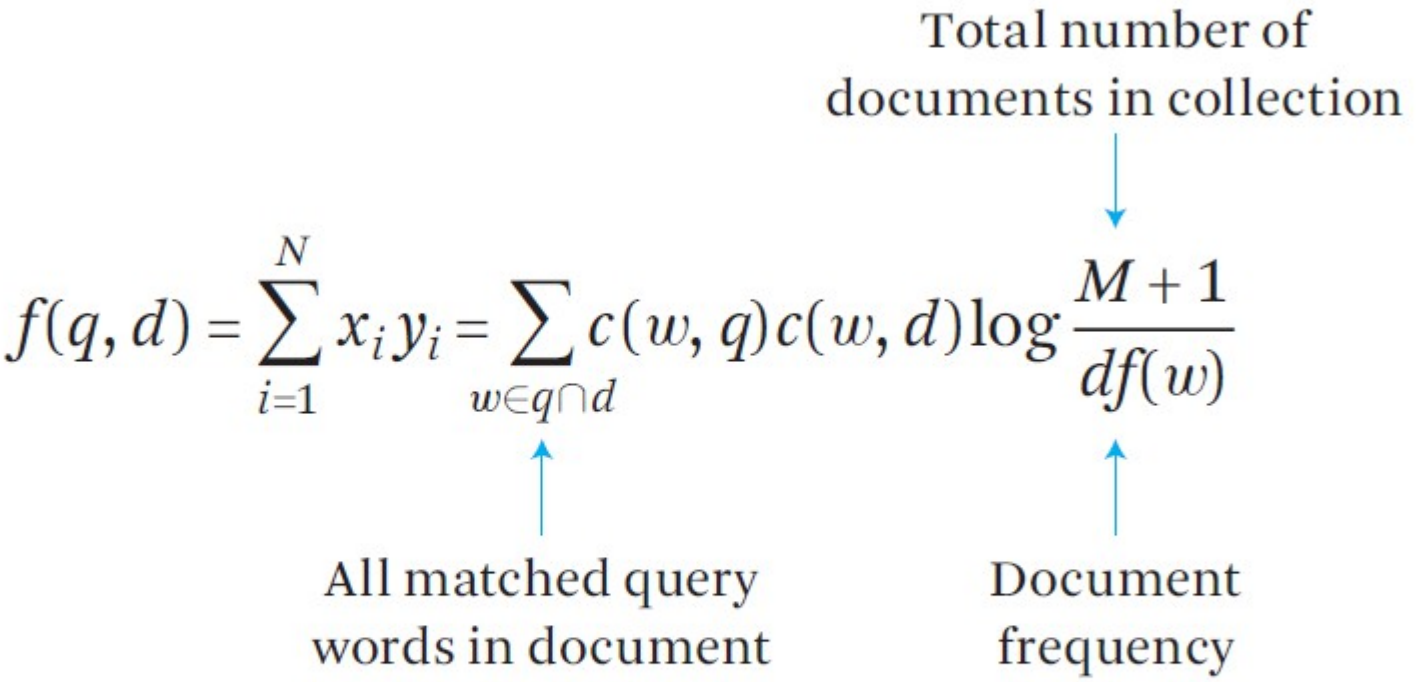


Figure 6.13 **A ranking function using a TF-IDF weighting scheme.**

tf transformation (sublineal)

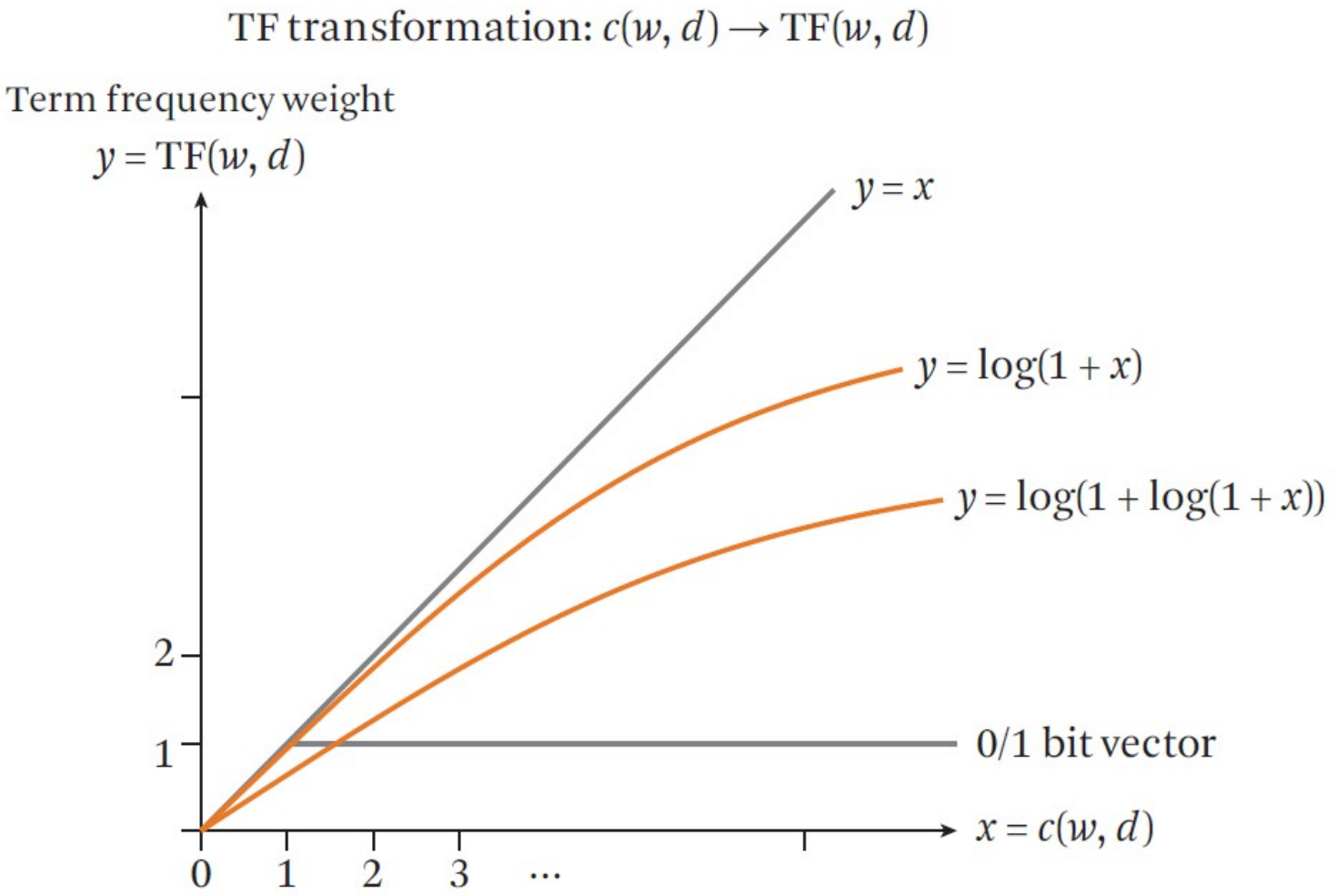


Figure 6.14 Illustration of different ways to transform TF.

state-of-the art transformation (bm25)

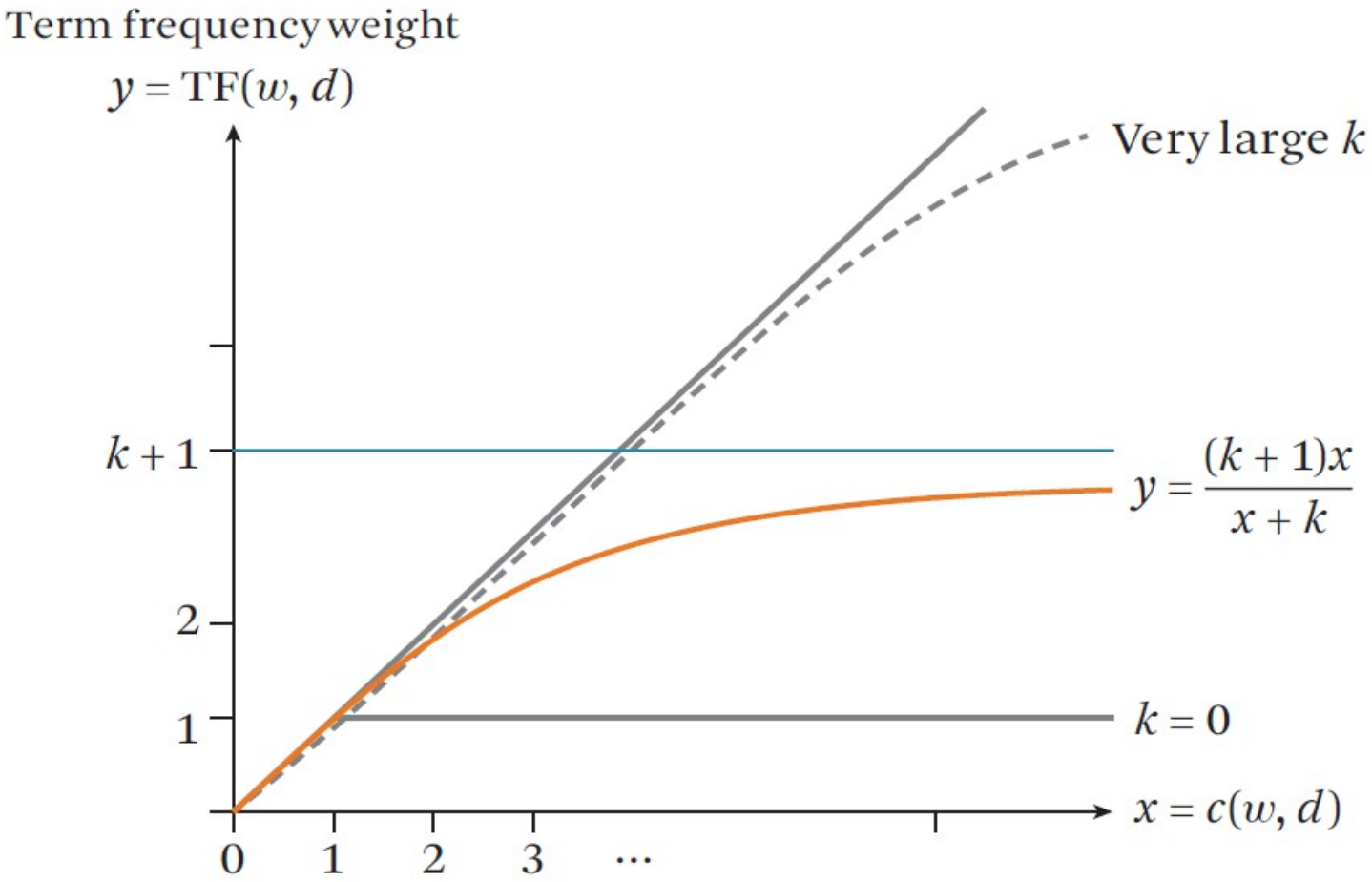


Figure 6.15 Illustration of BM25 TF transformation.

bm25 tf weight

unlike the log transformation,

BM25 TF is **upper bounded** (by k+1)

k=0 => binary weight

k large => linear TF

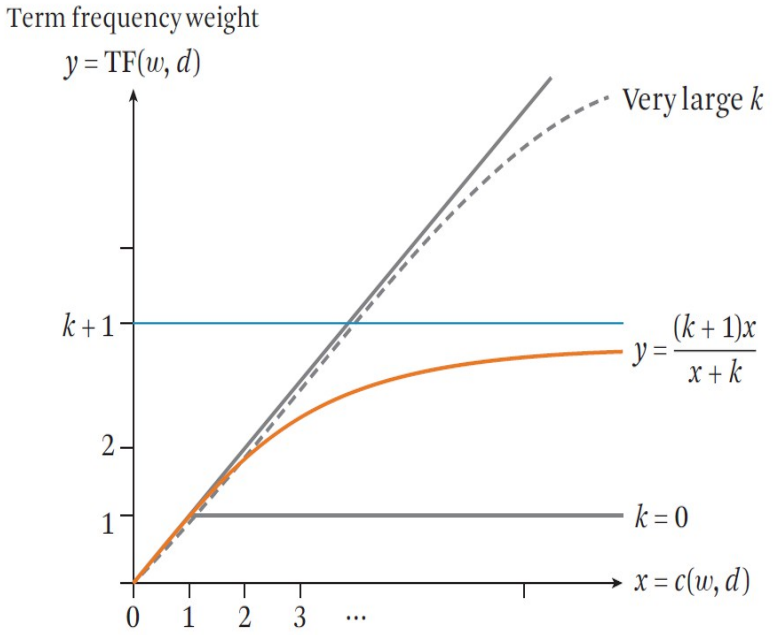


Figure 6.15 Illustration of BM25 TF transformation.

the upper bound is useful to control the influence of a particular term.

For example, we can prevent a spammer from just increasing the count of one term to spam all queries that might match this term

avoids a dominance by one single term over all others

document length normalization

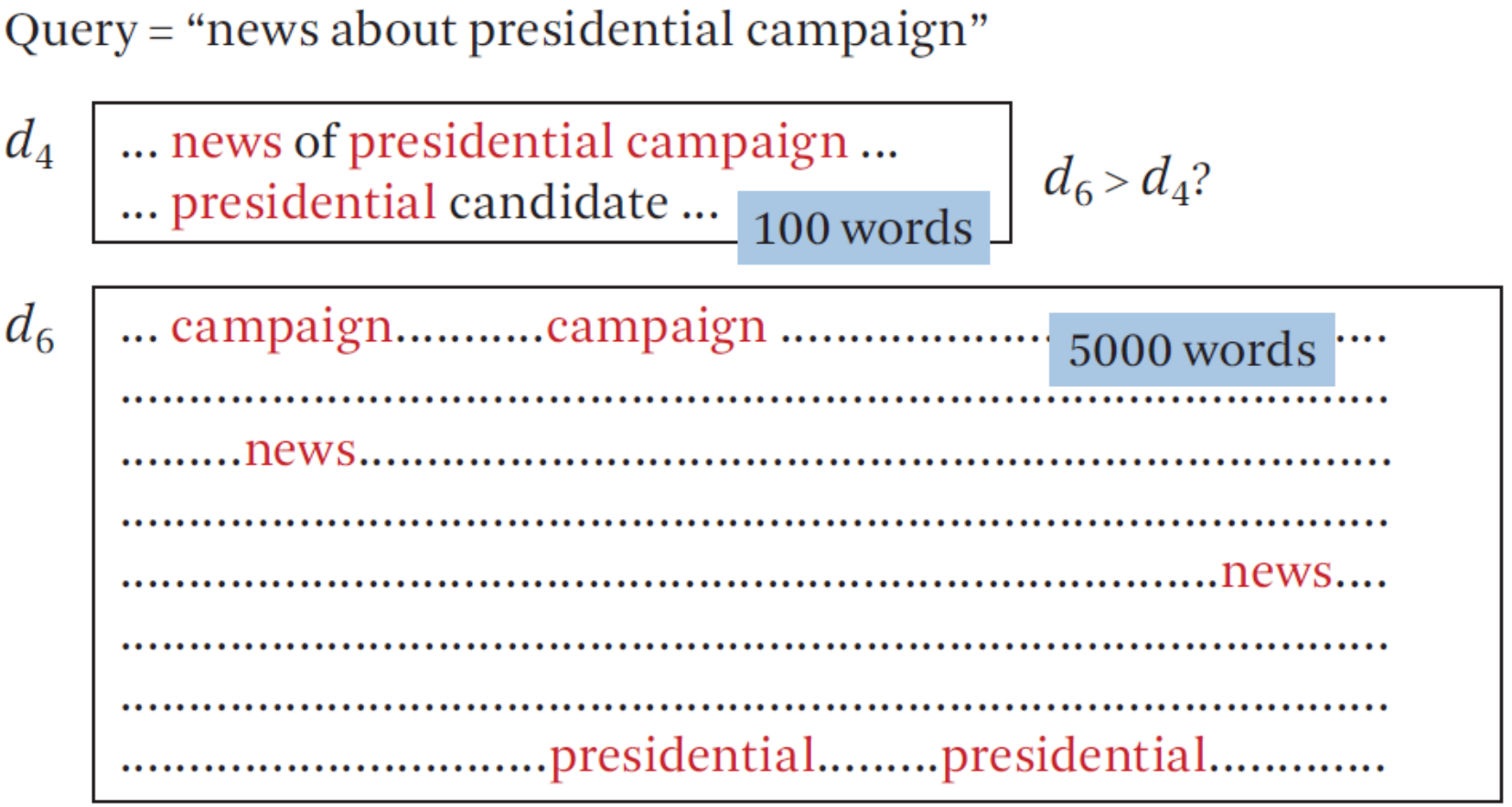


Figure 6.16 Two documents with very different document lengths.

document length normalization

long docs: higher chance to **match any query** (since they contain more words)

long docs: **higher TFs**

tradeoff: penalize long docs vs overpenalize long docs

verbosity vs scope

an understanding of the **discourse** structure of documents is needed for optimal document length normalization

pivoted length normalization Singhal et al. [1996]

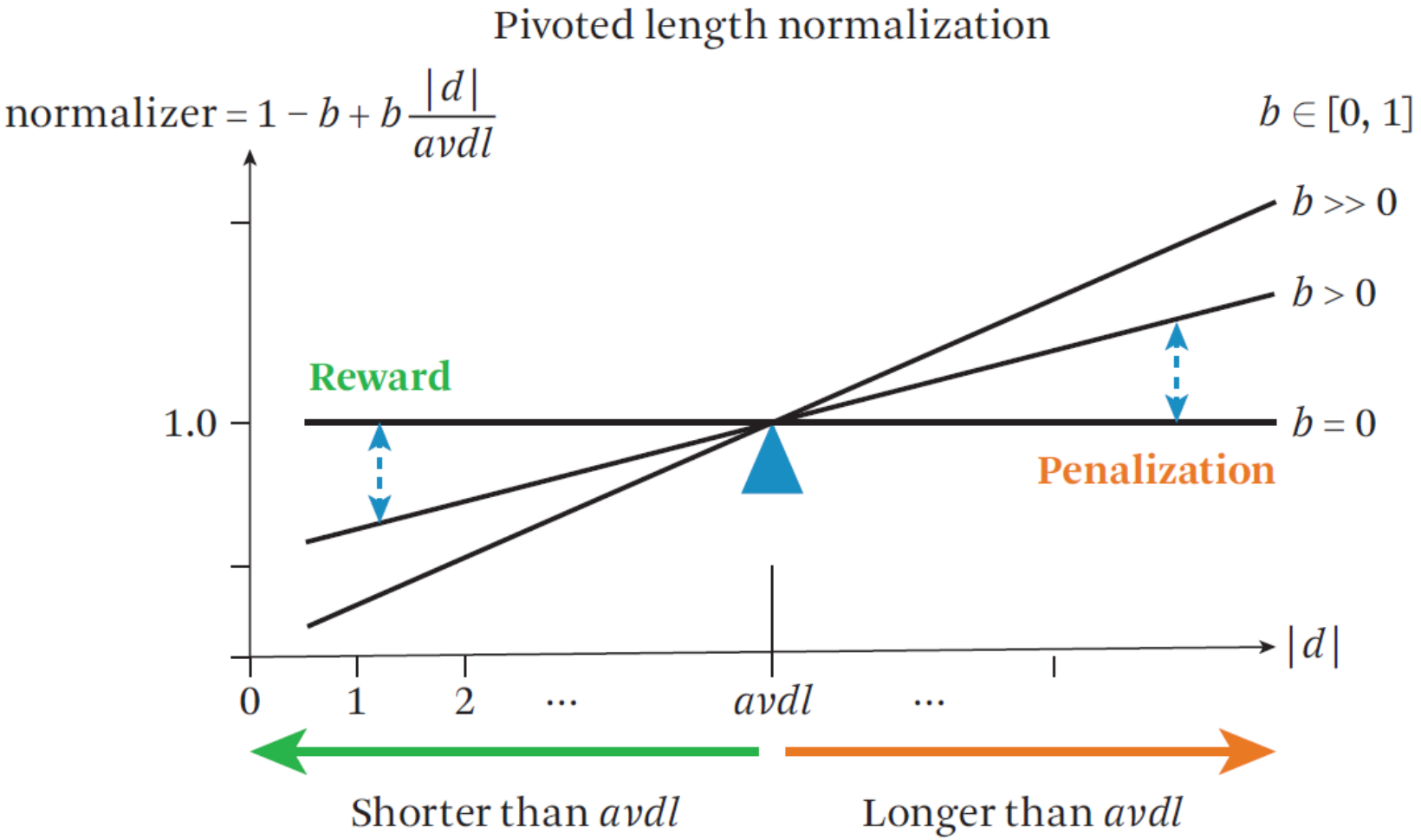


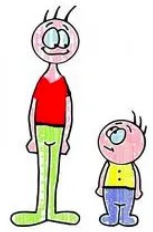
Figure 6.17 Illustration of pivoted document length normalization.

pivoted length normalization Singhal et al. [1996]

average document length as a **pivot**, or reference point

average length documents, the score is about right (a normalizer would be one)

a document is **longer** than the average document length, then there will be some **penalization**



If it's **shorter** than the average document length, there's even some **reward**

Pivoted length normalization VSM

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{\ln(1+\ln(1+c(w,d)))}{1-b+b\frac{|d|}{avdl}} \log \frac{M+1}{\mathrm{df}(w)} \quad b \in [0,1]$$

BM25/Okapi

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})} \log \frac{M+1}{\mathrm{df}(w)} \quad b \in [0,1], \ \ k \in [0,+\infty)$$

Figure 6.18 **State-of-the-art vector space models: pivoted length normalization and Okapi BM25.**

further improvements

stemmed words (computing => comput)

stop word removal (e.g, removing the, a, of, ...)

use **phrases** or even **latent semantic analysis**

character n-grams

in practice, researchers have found that the BoW representation with phrases (or “bag-of-phrases”) is the most effective representation

further improvements

in some other languages we may need to do some natural language processing to determine **word boundaries**

other similarity measures: cosine of the angle between two vectors, or an Euclidean distance measure

field-based BM25 (e.g. title field, the abstract field, the body of the research article, or even anchor text on web pages). applies BM25 on each field, and then combines the scores, but keeps global (i.e., across all fields) frequency counts

probabilistic retrieval models

p(R = 1|d, q): probability that a given doc d is relevant to a query q

R ∈ {0, 1} is a **binary random variable** denoting **relevance**

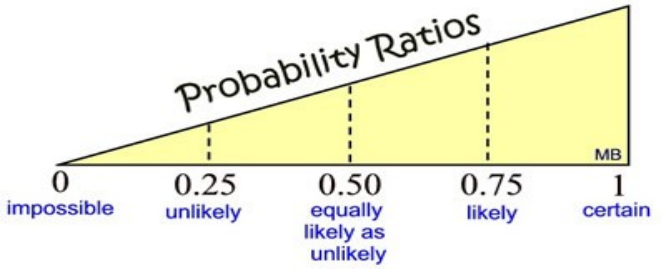
query & docs: **observations** from random variables

the classic probabilistic model has led to the BM25 retrieval function (its form is quite similar to these types of models)

probabilistic retrieval models

language modeling: **query likelihood** retrieval model

probability of relevance approximated by the probability of a query given a document and relevance, **p(q|d, R = 1)**



<u>if a user likes document d, how likely would the user enter query q in order to retrieve document d?</u>

query likelihood. estimation of p(q|d, R = 1)

if we look at all the entries where we see a particular d and a particular q, we can calculate how likely we will see a 1 in the 3rd column

Query q	Document d	Relevant? R
q1	d1	1
q1	d2	1
q1	d3	0
d1	d4	0
q1	d5	1
⋮		
q1	d1	0
q1	d2	1
q1	d3	0
q2	d3	1
q3	d1	1
q4	d2	1
q4	d3	0

$$f(q, d) = p(R = 1 \mid d, q) = \frac{\text{count}(q, d, R = 1)}{\text{count}(q, d)}$$

$$P(R = 1 \mid \text{q1}, \text{d1}) = 1/2$$

$$P(R = 1 \mid \text{q1}, \text{d2}) = 2/2$$

$$P(R = 1 \mid \text{q1}, \text{d3}) = 0/2$$

Figure 6.19 Basic idea of probabilistic models for information retrieval.

query likelihood. estimation of p(q|d, R = 1)

doable with **clickthrough** data but..

we don't observe all the queries, all of the docs and all the relevance values

many **unseen documents** (we can only collect data from the docs that we have shown to the users)

even more **unseen queries** because you cannot predict what queries will be typed in by users

Query q	Document d	Relevant? R
q1	d1	1
q1	d2	1
q1	d3	0
d1	d4	0
q1	d5	1
⋮		
q1	d1	0
q1	d2	1
q1	d3	0
q2	d3	1
q3	d1	1
q4	d2	1
q4	d3	0

$$f(q, d) = p(R = 1 \mid d, q) = \frac{\text{count}(q, d, R = 1)}{\text{count}(q, d)}$$

$$P(R = 1 \mid \text{q1}, \text{d1}) = 1/2$$

$$P(R = 1 \mid \text{q1}, \text{d2}) = 2/2$$

$$P(R = 1 \mid \text{q1}, \text{d3}) = 0/2$$

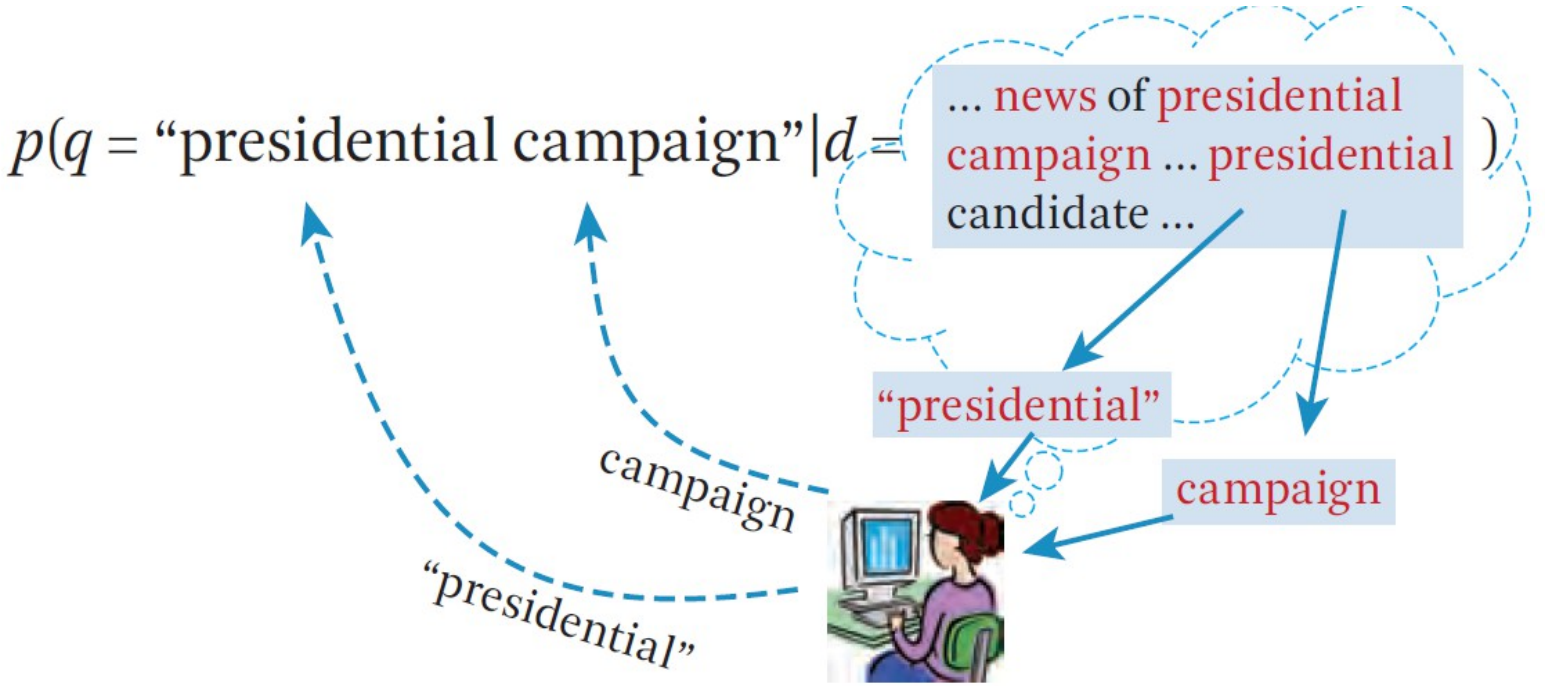
Figure 6.19 Basic idea of probabilistic models for information retrieval.

query likelihood. estimation of $p(q|d, R = 1)$

we assume that a user formulates the query based on an **imaginary relevant document**

which doc is most likely the imaginary relevant doc in the user's mind when the user formulates this query?

query likelihood. estimation of p(q|d, R = 1)

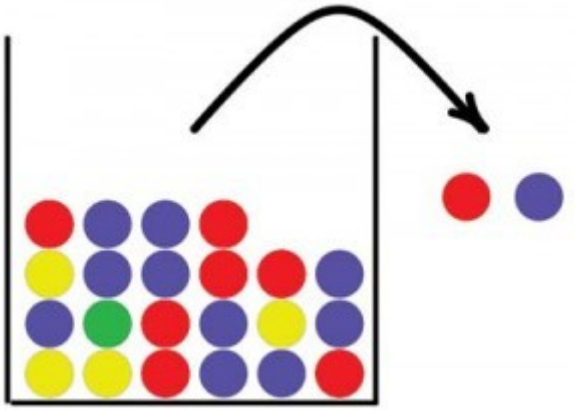


If the user is **thinking of this doc**,
how likely would she **pose this query**?

Figure 6.20 Generating a query by sampling words from a document.

query likelihood. estimation of $p(q|d, R = 1)$

assume that the **query is generated by sampling words from the document**



each query word is **independent** and that each word is obtained from the imagined ideal document

the probability of the query is just a product of the probability of each query word, where the probability of each word is just the relative frequency of the word in the document

query likelihood. estimation of p(q|d, R = 1)

$$p(q = \text{“presidential campaign”}|d) = \frac{c(\text{“presidential”}, d)}{|d|} * \frac{c(\text{“campaign”}, d)}{|d|}$$

$$p(q|d_4 = \text{... news of presidential campaign ... presidential candidate ...}) = \frac{2}{|d_4|} * \frac{1}{|d_4|}$$

$$p(q|d_3 = \text{... news of presidential campaign ...}) = \frac{1}{|d_3|} * \frac{1}{|d_3|}$$

$$p(q|d_2 = \text{... news about organic food campaign ...}) = \frac{0}{|d_2|} * \frac{1}{|d_2|} = 0$$

Figure 6.21 **Computing the probability of a query given a document using the query likelihood formulation.**

captures TF

but if the query is “presidential campaign update”… all docs get 0 probability

query likelihood. estimation of p(q|d, R = 1)

we have to assume that the user could have drawn a word **not necessarily from the document**...

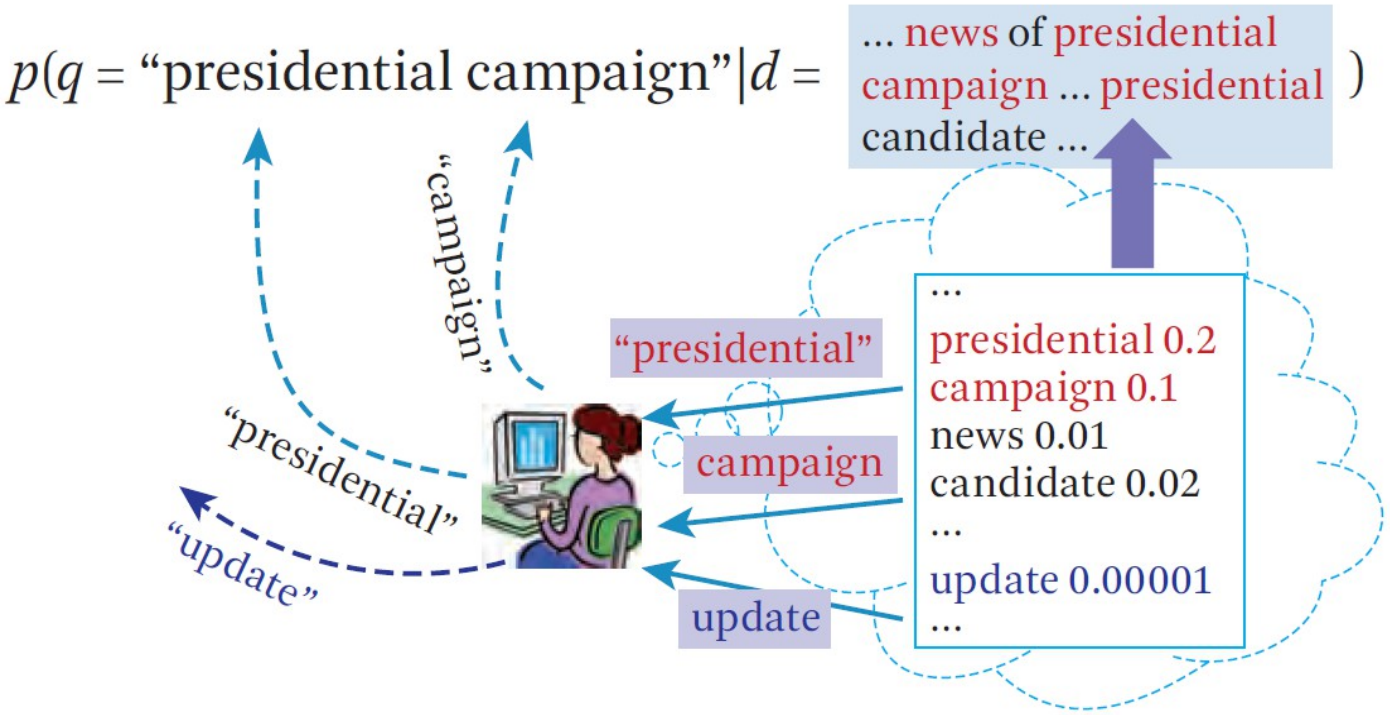


Figure 6.22 Computing the probability of a query given a document using a document language model.

query likelihood

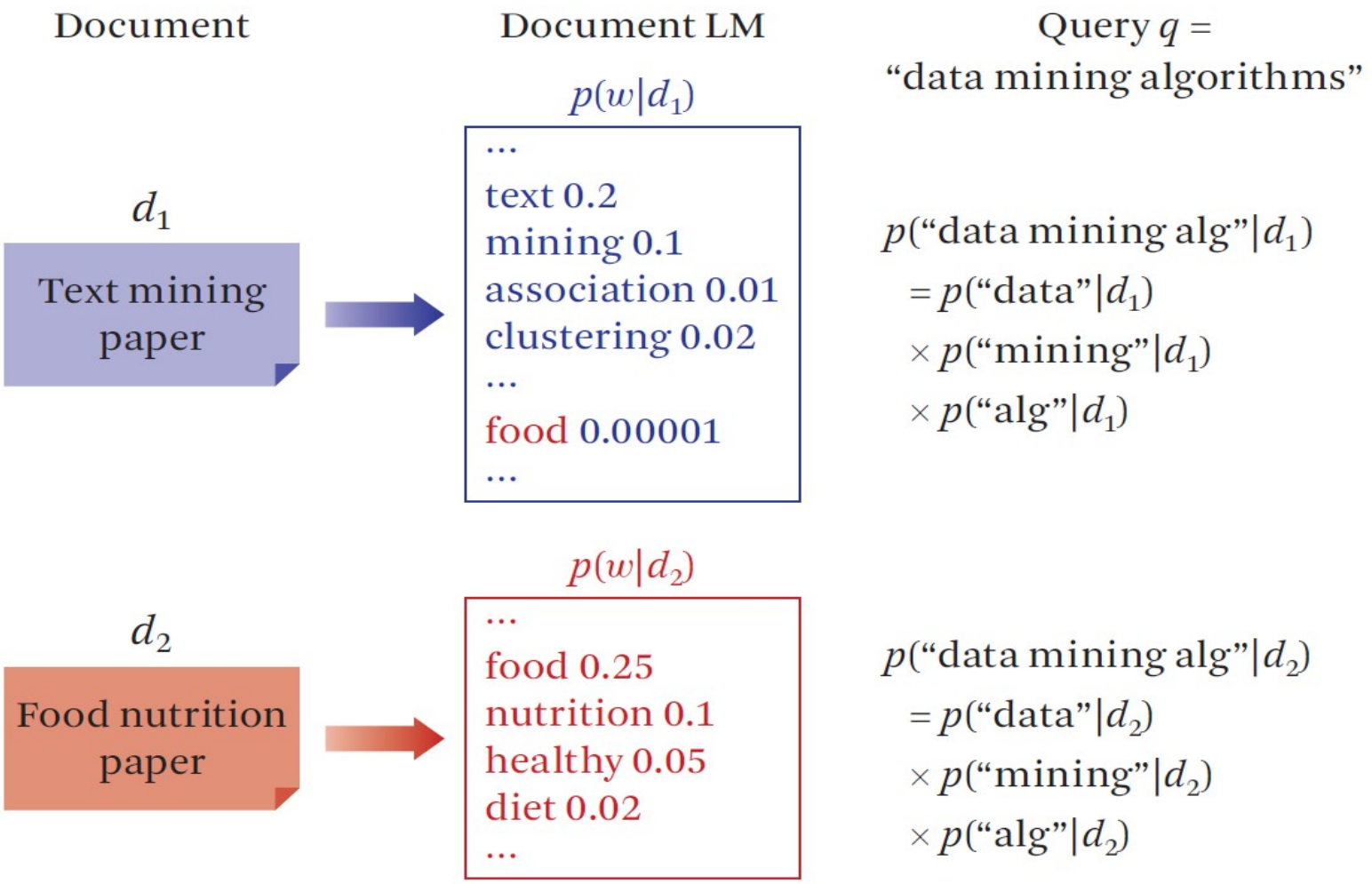


Figure 6.23 Scoring a query on two documents based on their language models.

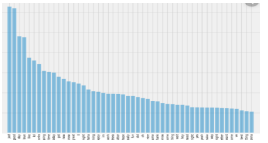
query likelihood

$$q = w_1, w_2, \ldots, w_n$$

$$p(q \mid d) = p(w_1 \mid d) \times p(w_2 \mid d) \times \ldots \times p(w_n \mid d).$$

$$\text{score}(q, d) = \log p(q \mid d) = \sum_{i=1}^{n} \log p(w_i \mid d) = \sum_{w \in V} c(w, q) \log p(w \mid d).$$

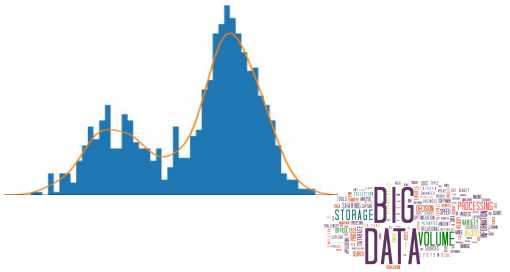
the need of smoothing



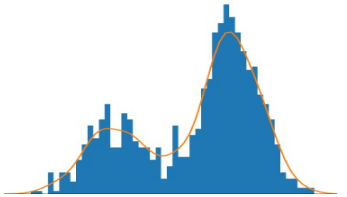
maximum likelihood estimation (MLE): we normalize the word frequencies in the document by the document length

BUT words that have not occurred in the document will have zero probability! 🚨

in order to assign a non-zero probability to words that have not been observed in the document, we need to **take away some probability mass** from seen words (we need some extra probability mass for the unseen words)



smoothing

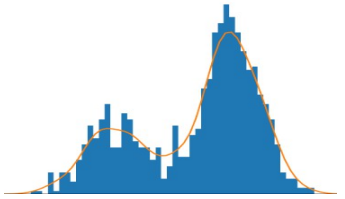


probability of an **unseen word**: proportional to its probability as given by a **reference language model:**

$$p(w \mid d) = \begin{cases} p_{\text{seen}}(w \mid d) & \text{if } w \text{ seen in } d \\ \alpha_d \cdot p(w \mid C) & \text{otherwise.} \end{cases}$$

tells us which unseen words have a higher probability than other unseen words

smoothing



$$\log p(q|d) = \sum_{w \in V} c(w,q) \log p(w|d)$$

$$= \sum_{w \in V, c(w,d)>0} c(w,q) \log p_{\text{Seen}}(w|d) + \sum_{w \in V, c(w,d)=0} c(w,q) \log \alpha_d p(w|C)$$

Query words **matched** in d

Query words **not matched** in d

$$\sum_{w \in V} c(w,q) \log \alpha_d p(w|C) - \sum_{w \in V, c(w,d)>0} c(w,q) \log \alpha_d p(w|C)$$

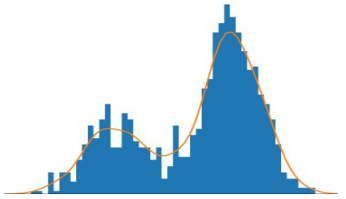
All query words

Query words **matched** in d

$$= \sum_{w \in V, c(w,d)>0} c(w,q) \log \frac{p_{\text{Seen}}(w|d)}{\alpha_d p(w|C)} + |q| \log \alpha_d + \sum_{w \in V} c(w,q) \log p(w|C)$$

Figure 6.24 **Substituting smoothed probabilities into the query likelihood retrieval formula.**

smoothing



$$\log p(q|d) = \sum_{\substack{w \in d \\ w \in q}} c(w,q)[\log \frac{p_{\text{Seen}}(w|d)}{\alpha_d p(w|C)}] + n \log \alpha_d + \boxed{\sum_{i=1}^{N} \log p(w_i|C)}$$

TF weighting — Doc length normalization — Matched query terms — IDF weighting — Ignore for ranking

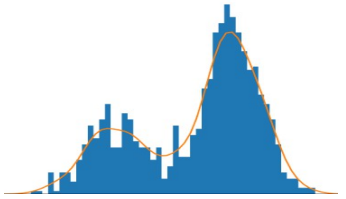
Figure 6.25 **The query likelihood retrieval formula captures the three heuristics from the vector space models.**

αd encodes how much probability mass we want to give to unseen words

long document: less smoothing (we can assume that it is large enough that we have probably observed all of the words that the author could have written)

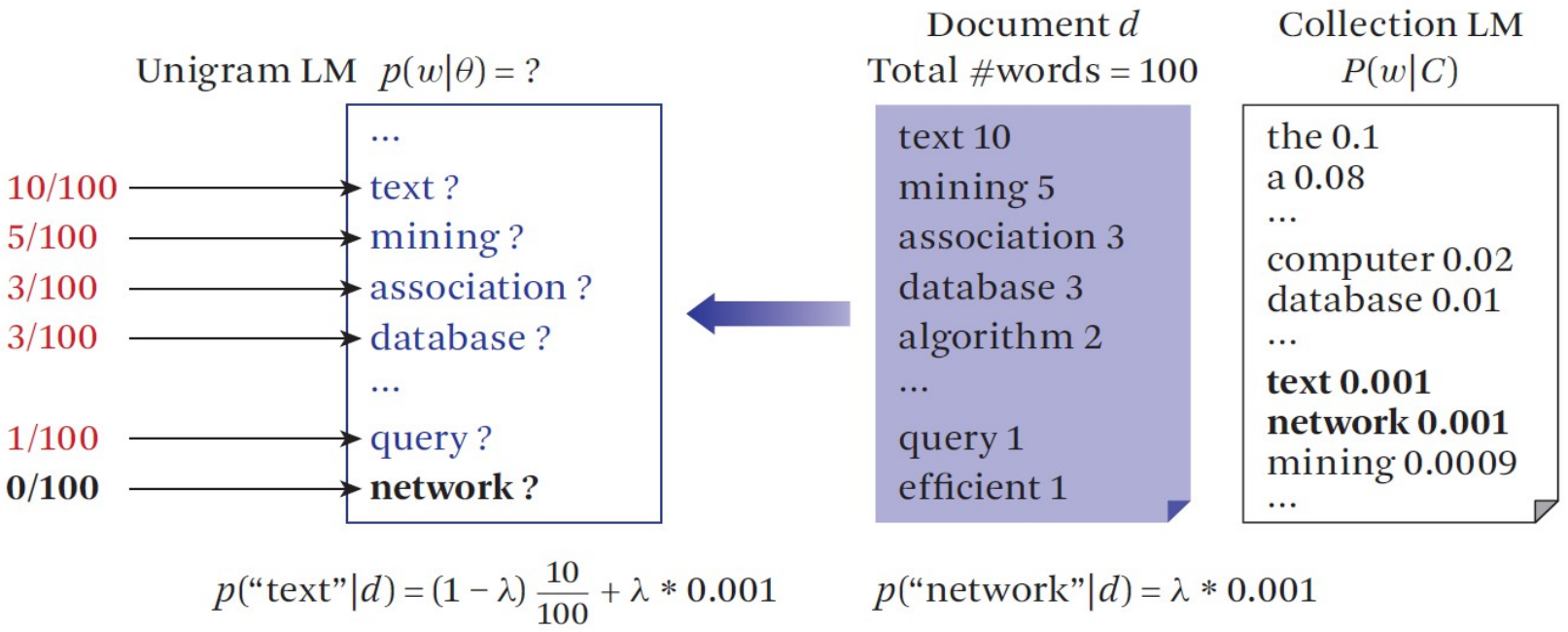
short document: more smoothing (the number of unseen words is large)

Jelinek-Mercer smoothing



α_d is a fixed λ. linear interpolation with a fixed mixing coefficient

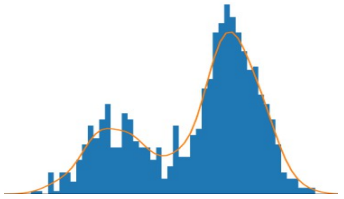
$$p(w|d) = (1-\lambda)\frac{c(w,d)}{|d|} + \lambda p(w|C) \qquad \lambda \in [0, 1]$$



$$p(\text{“text”}|d) = (1-\lambda)\frac{10}{100} + \lambda * 0.001 \qquad p(\text{“network”}|d) = \lambda * 0.001$$

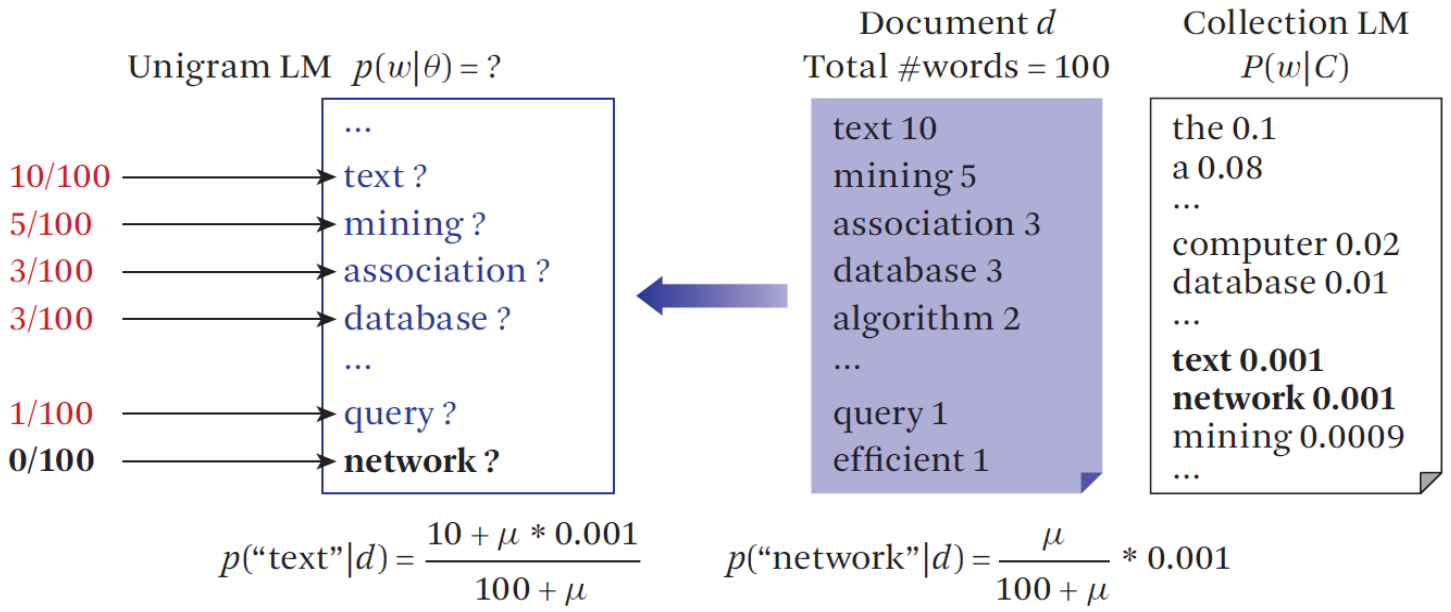
Figure 6.26 **Smoothing the query likelihood retrieval function with linear interpolation: Jelinek-Mercer smoothing.**

Dirichlet or Bayesian smoothing



- α_d doc-dependent and grows with μ
- a long document gets less smoothing

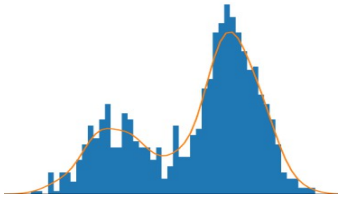
$$p(w|d) = \frac{c(w,d) + \mu p(w|C)}{|d| + \mu} = \frac{|d|}{|d| + \mu}\frac{c(w,d)}{|d|} + \frac{\mu}{|d| + \mu}p(w|C) \qquad \mu \in [0, +\infty)$$



$$p(\text{“text”}|d) = \frac{10 + \mu * 0.001}{100 + \mu} \qquad p(\text{“network”}|d) = \frac{\mu}{100 + \mu} * 0.001$$

Figure 6.27 Smoothing the query likelihood retrieval function with linear interpolation: Dirichlet prior smoothing.

final scoring



$$\text{score}_{JM}(q,d) = \sum_{w \in q,d} c(w,q) \log \left(1 + \frac{1-\lambda}{\lambda} \cdot \frac{c(w,d)}{|d| \cdot p(w \mid C)}\right).$$

$$\text{score}_{DIR}(q,d) = \sum_{w \in q,d} c(w,q) \log \left(1 + \frac{c(w,d)}{\mu \cdot p(w \mid C)}\right) + |q| \log \frac{\mu}{\mu + |d|}.$$