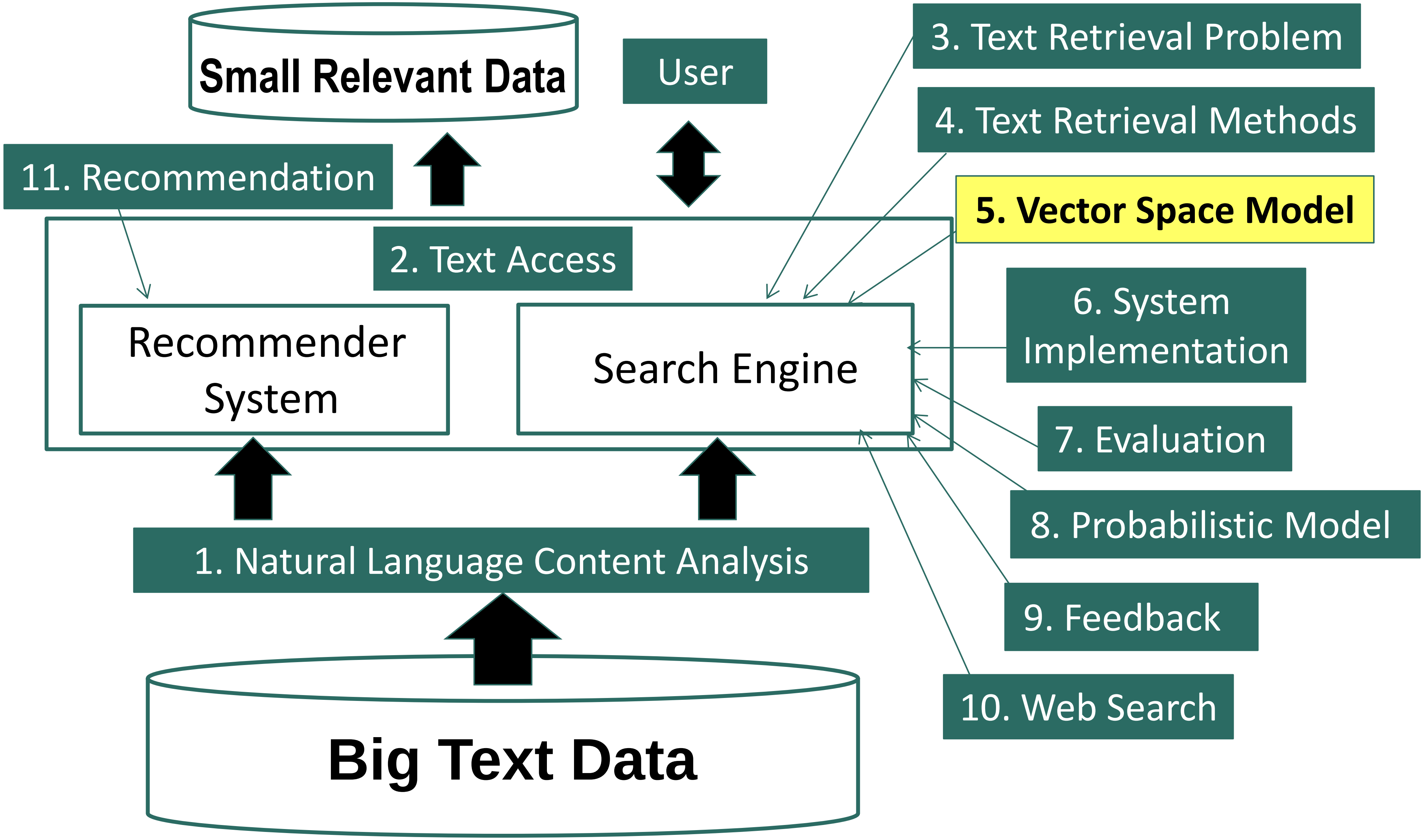
Vector Space Model: Improved Instantiation

Course Schedule



Slide adapted from ChengXiang Zhai's presentation

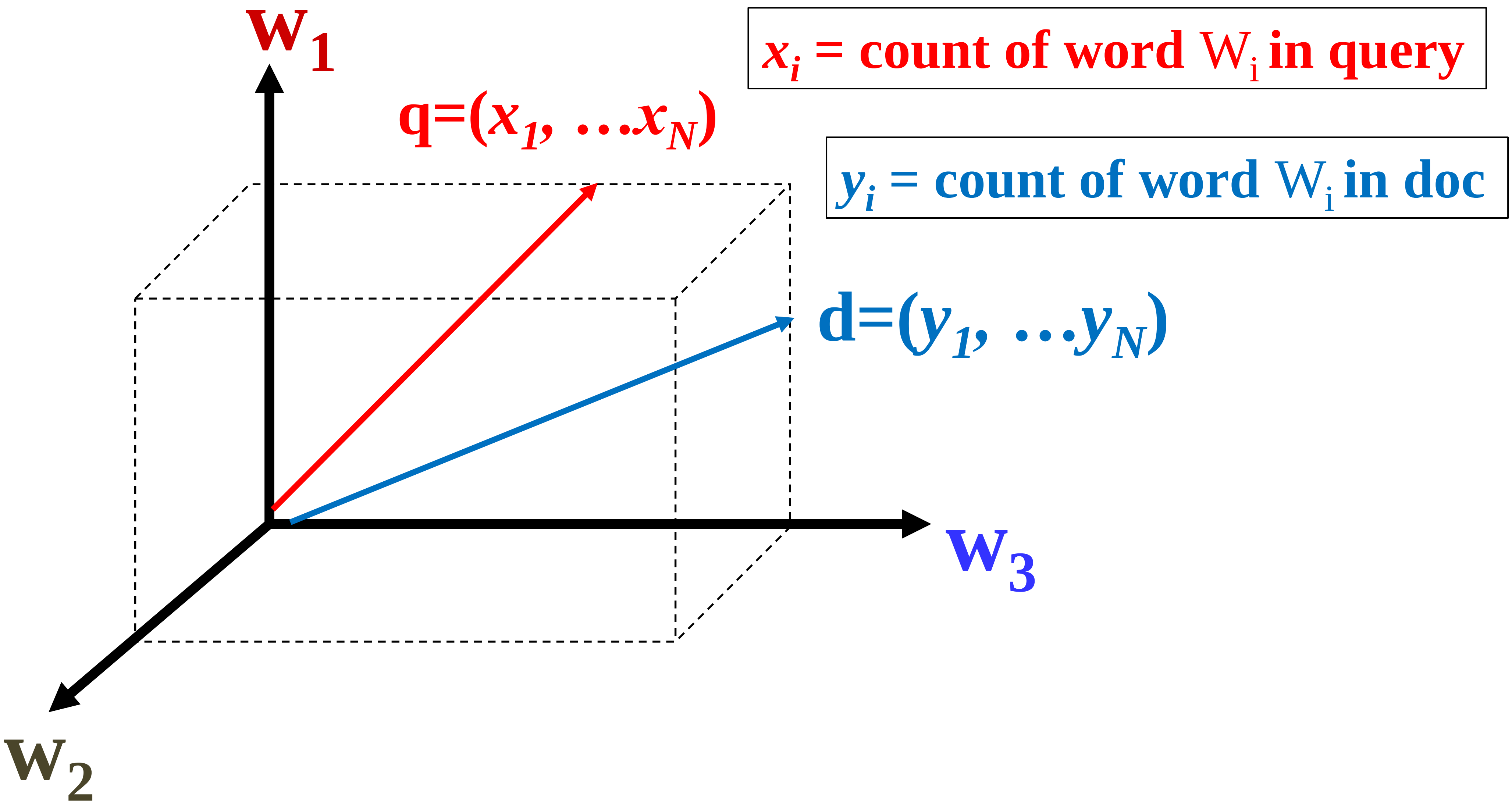
Two Problems of the Simplest VSM

Query = "news about presidential campaign"

d2	… **news about** organic food **campaign**…	**f(q,d2)=3**
d3	… **news** of **presidential campaign** …	**f(q,d3)=3**
d4	… **news** of **presidential campaign** … … **presidential** candidate …	**f(q,d4)=3**

1. Matching "presidential" more times deserves more credit
2. Matching "presidential" is more important than matching "about"

Improved Vector Placement: Term Frequency Vector



Slide adapted from ChengXiang Zhai's presentation

Improved VSM with Term Frequency Weighting

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

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

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

What does this ranking function intuitively capture?

Does it fix the problems of the simplest VSM?

Slide adapted from ChengXiang Zhai's presentation

Ranking Using Term Frequency (TF) Weighting

d2 | … **news about** organic food **campaign**… | f(q,d2)=3

q= (1, 1, 1, 1, 0, …)
d2= (1, 1, 0, 1, 1, …)

d3 | … **news** of **presidential campaign** … | f(q,d3)=3

q= (1, 1, 1, 1, 0, …)
d3= (1, 0, 1, 1, 0, …)

d4 | … **news** of **presidential campaign** … … **presidential** candidate … | **f(q,d4)=4!**

q= (1, 1, 1, 1, 0, …)
d4= (1, 0, 2, 1, 0, …)

How to Fix Problem 2 (“presidential” vs. “about”)

d2 … **news about** organic food **campaign**…

d3 … **news** of **presidential campaign** …

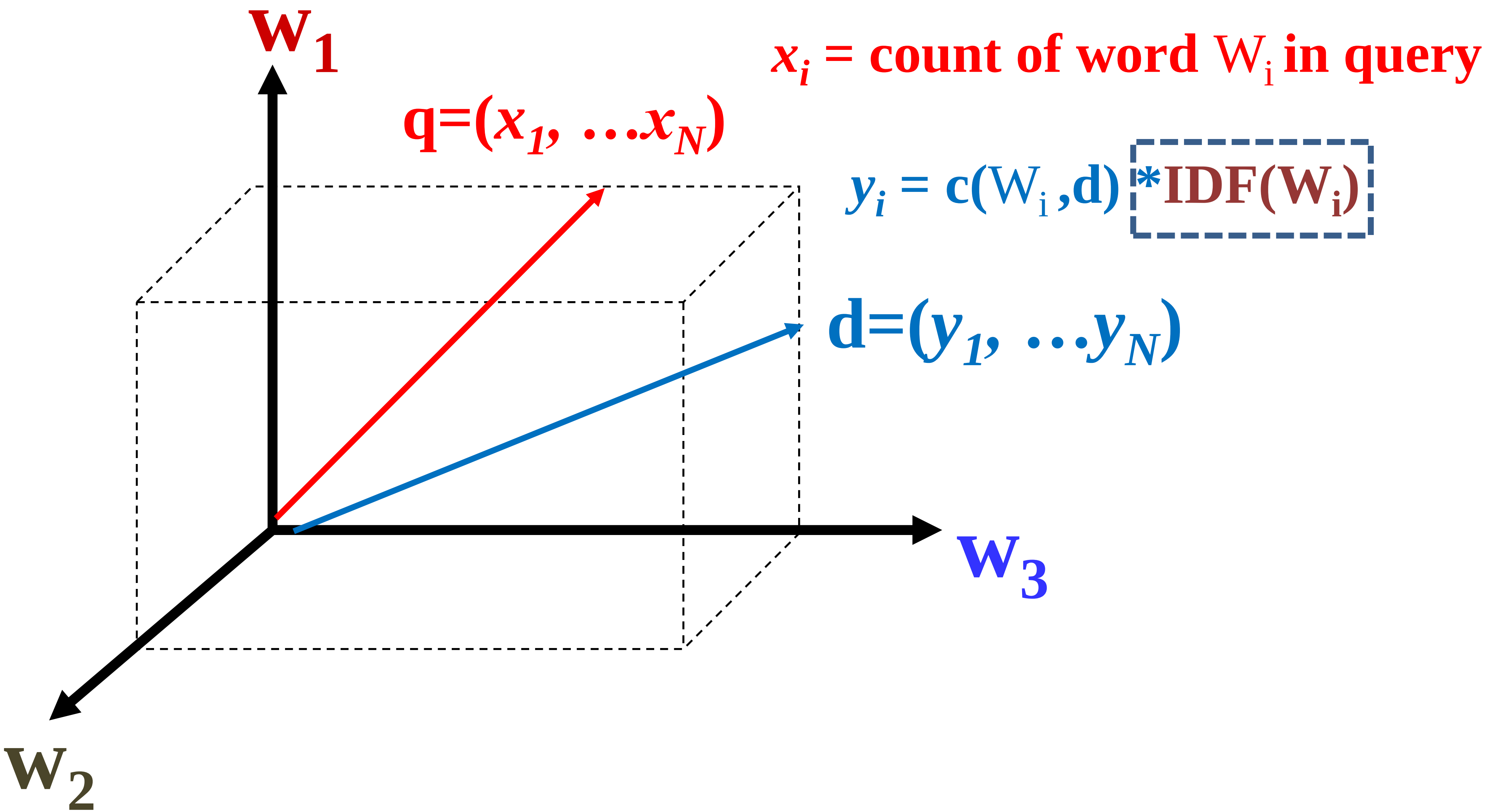
V= {news, about, presidential, campaign, food …. }

q=	**(1,**	**1,**	**1,**	**1,**	**0,**	**…)**
d2=	**(1,**	**1,**	**0,**	**1,**	**1,**	**…)**
q=	**(1,**	**1,**	**1,**	**1,**	**0,**	**…)**
d3=	**(1,**	**0,**	**1,**	**1,**	**0,**	**…)**

f(q,d2)<3

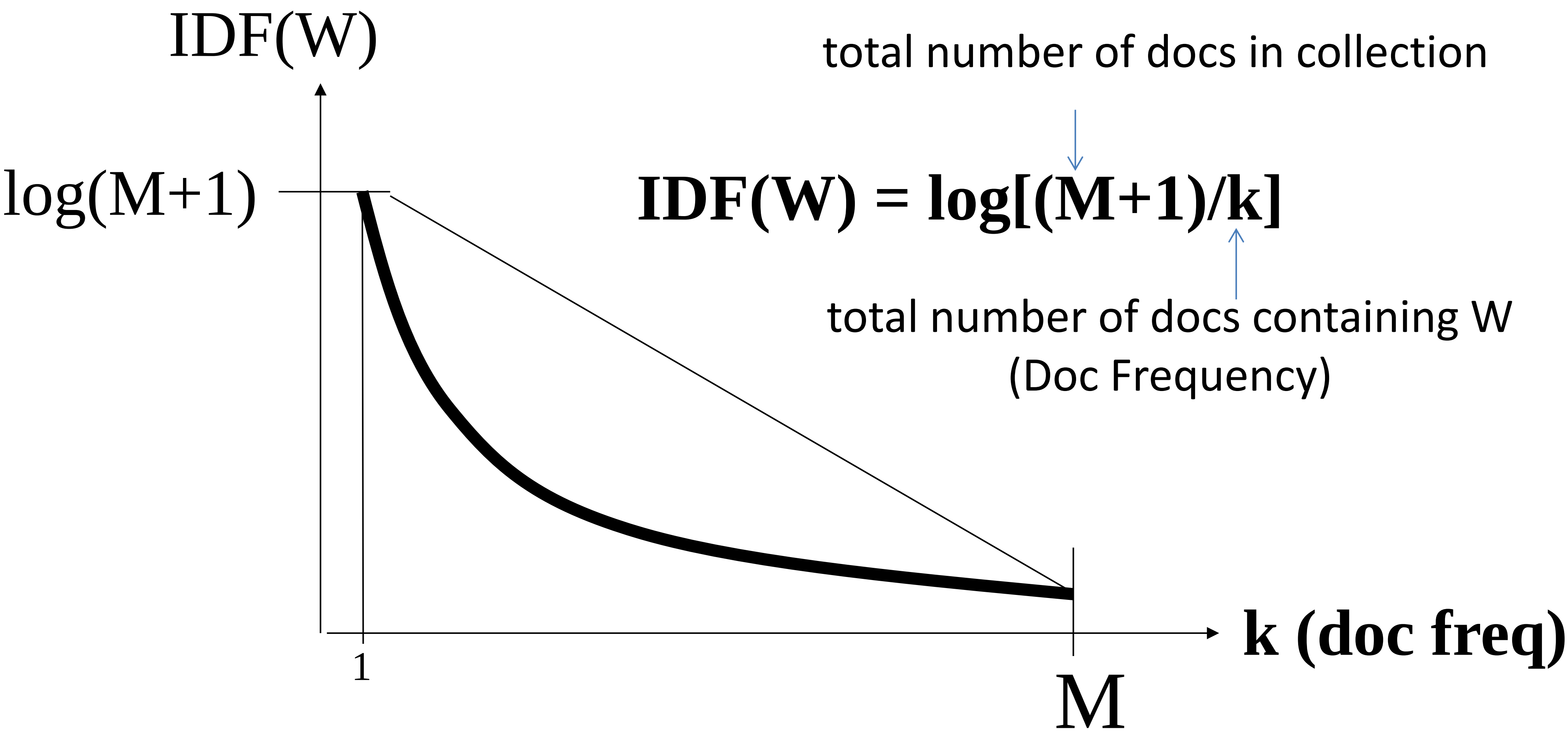
f(q,d3)>3

Further Improvement of Vector Placement: Adding Inverse Document Frequency (IDF)



Slide adapted from ChengXiang Zhai's presentation

IDF Weighting: Penalizing Popular Terms



total number of docs in collection

$$\mathbf{IDF(W) = \log[(M+1)/k]}$$

total number of docs containing W
(Doc Frequency)

Solving Problem 2 (“Presidential” vs “About”)

d2 … **news about** organic food **campaign**…

d3 … **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,	…)
d2=	(1*1.5,	1*1.0	0,	1*3.1,	0,	…)
q=	(1,	1,	1,	1,	0,	…)
d3=	(1*1.5,	0,	1*2.5	1*3.1,	0,	…)

f(q,d2) = 5.6 < f(q,d3)=7.1

Slide adapted from ChengXiang Zhai's presentation

How Effective Is VSM with TF-IDF Weighting?

Query = "news about presidential campaign"

d1	… **news about** …	f(q,d1)=2.5
d2	… **news about** organic food **campaign**…	f(q,d2)=5.6
d3	… **news** of **presidential campaign** …	f(q,d3)=7.1
d4	… **news** of **presidential campaign** … … **presidential** candidate …	f(q,d4)=9.6
d5	… **news** of organic food **campaign**… **campaign**…**campaign**…**campaign**…	**f(q,d5)=13.9!**

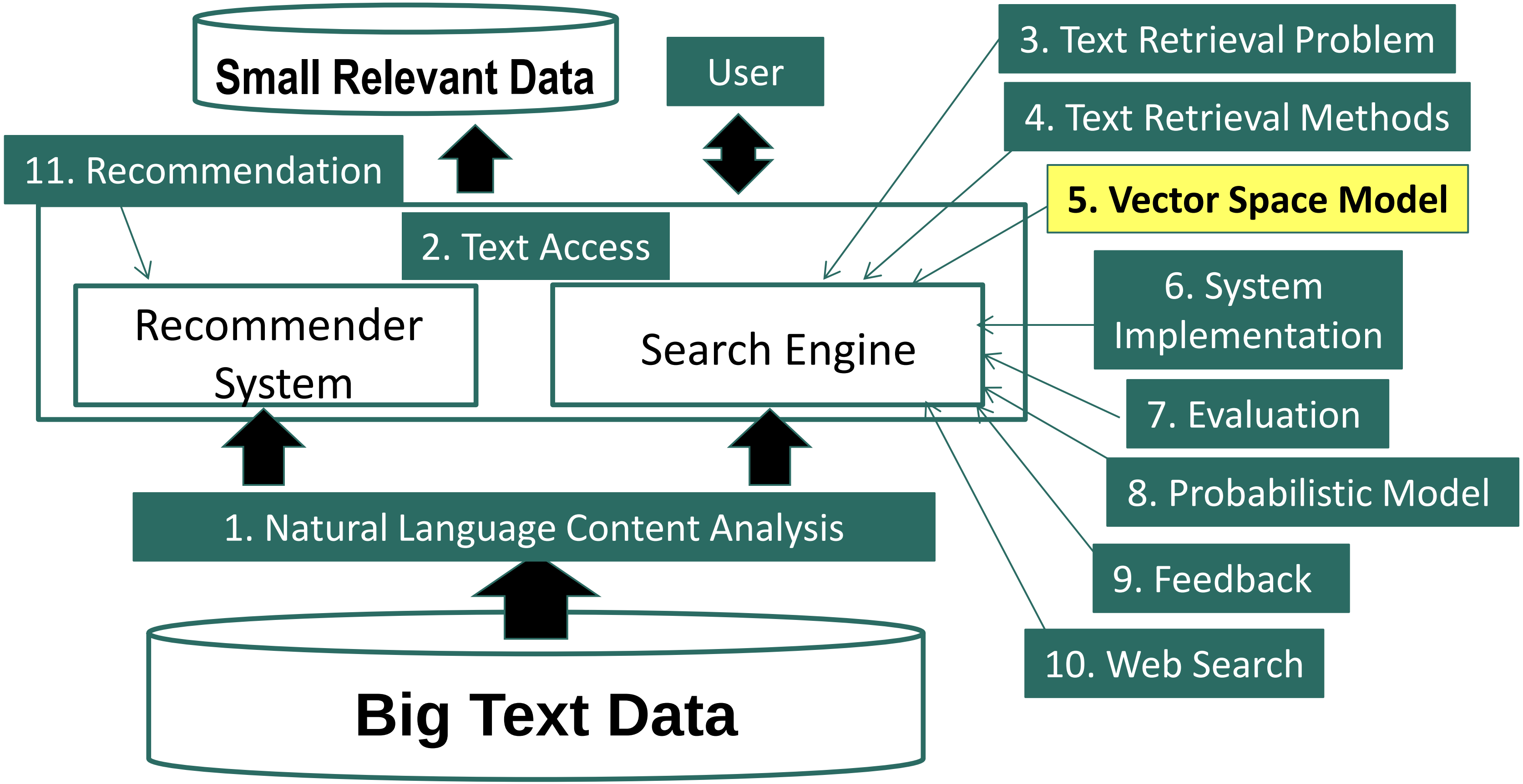
Summary

- Improved VSM
  - Dimension = word
  - Vector = TF-IDF weight vector
  - Similarity = dot product
  - Working better than the simplest VSM
  - Still having problems

Slide adapted from ChengXiang Zhai's presentation

Vector Space Model: TF Transformation

Course Schedule



Slide adapted from ChengXiang Zhai's presentation

VSM with TF-IDF Weighting Still Has a Problem!

Query = "news about presidential campaign"

d1	… **news about** …	f(q,d1)=2.5
d2	… **news about** organic food **campaign**…	f(q,d2)=5.6
d3	… **news** of **presidential campaign** …	f(q,d3)=7.1
d4	… **news** of **presidential campaign** … … **presidential** candidate …	f(q,d4)=9.6
d5	… **news** of organic food **campaign**… **campaign**…**campaign**…**campaign**…	**f(q,d5)=13.9?**

Ranking Function with TF-IDF Weighting

Total # of docs in collection

$$f(q,d)=\sum_{i=1}^{N}x_i y_i=\sum_{w\in q\cap d}c(w,q)\underline{c(w,d)}\log\frac{M+1}{df(w)}$$

All matched query words in d

Doc Frequency

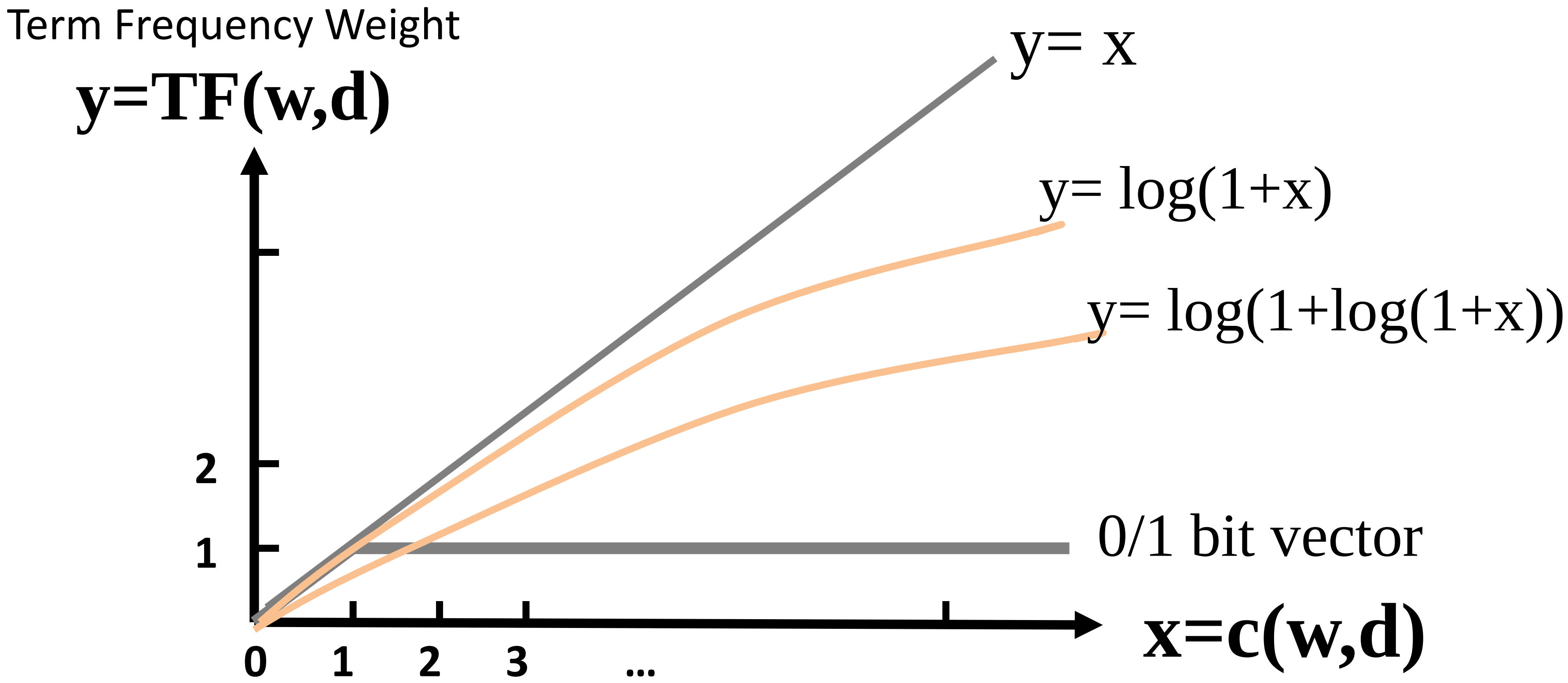
d5 … **news** of organic food **campaign**… **campaign**…**campaign**…**campaign**…

c("campaign",d5)=4

➔f(q,d5)=13.9?

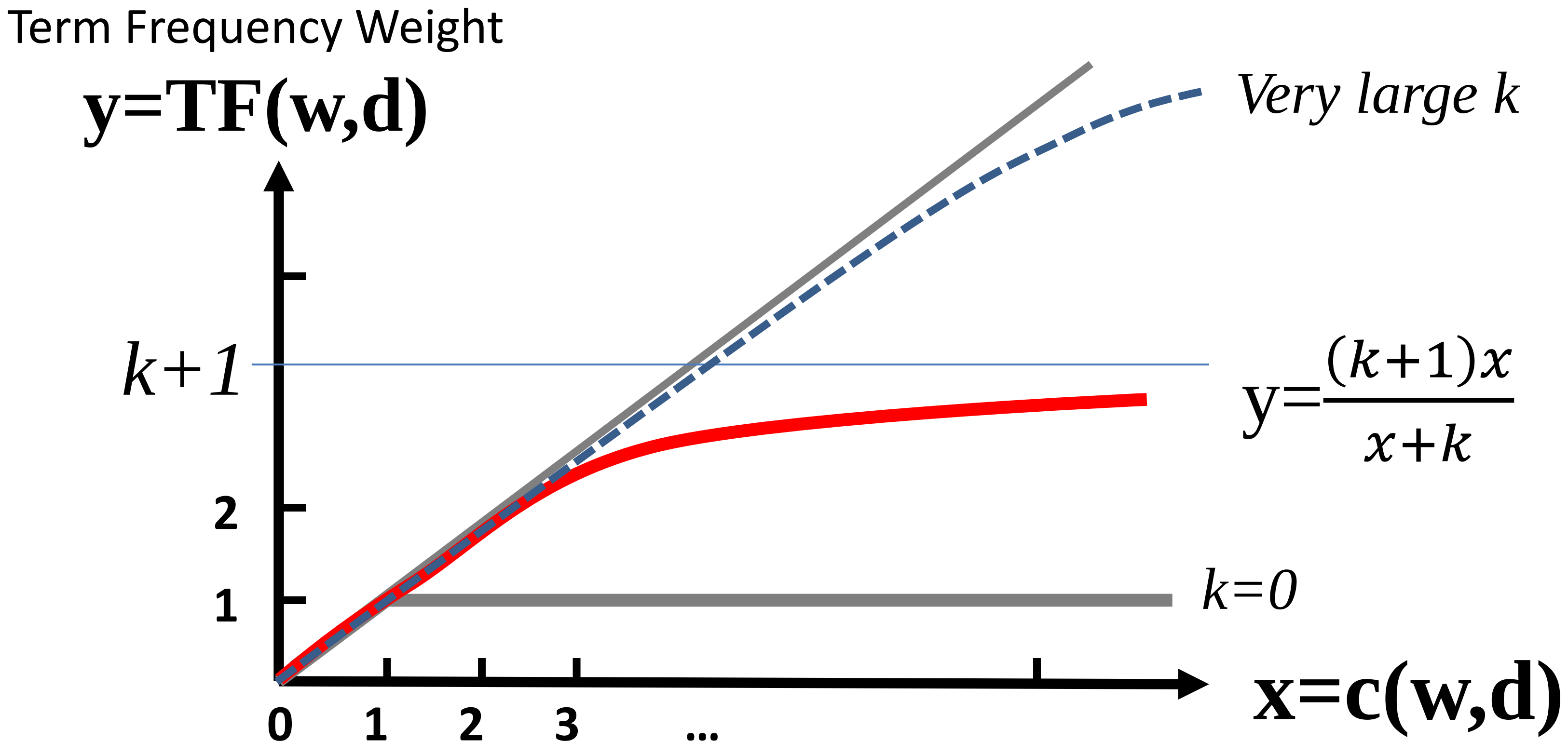
Slide adapted from ChengXiang Zhai's presentation

TF Transformation: c(w,d)➔TF(w,d)



Slide adapted from ChengXiang Zhai's presentation

TF Transformation: BM25 Transformation



Slide adapted from ChengXiang Zhai's presentation

Summary

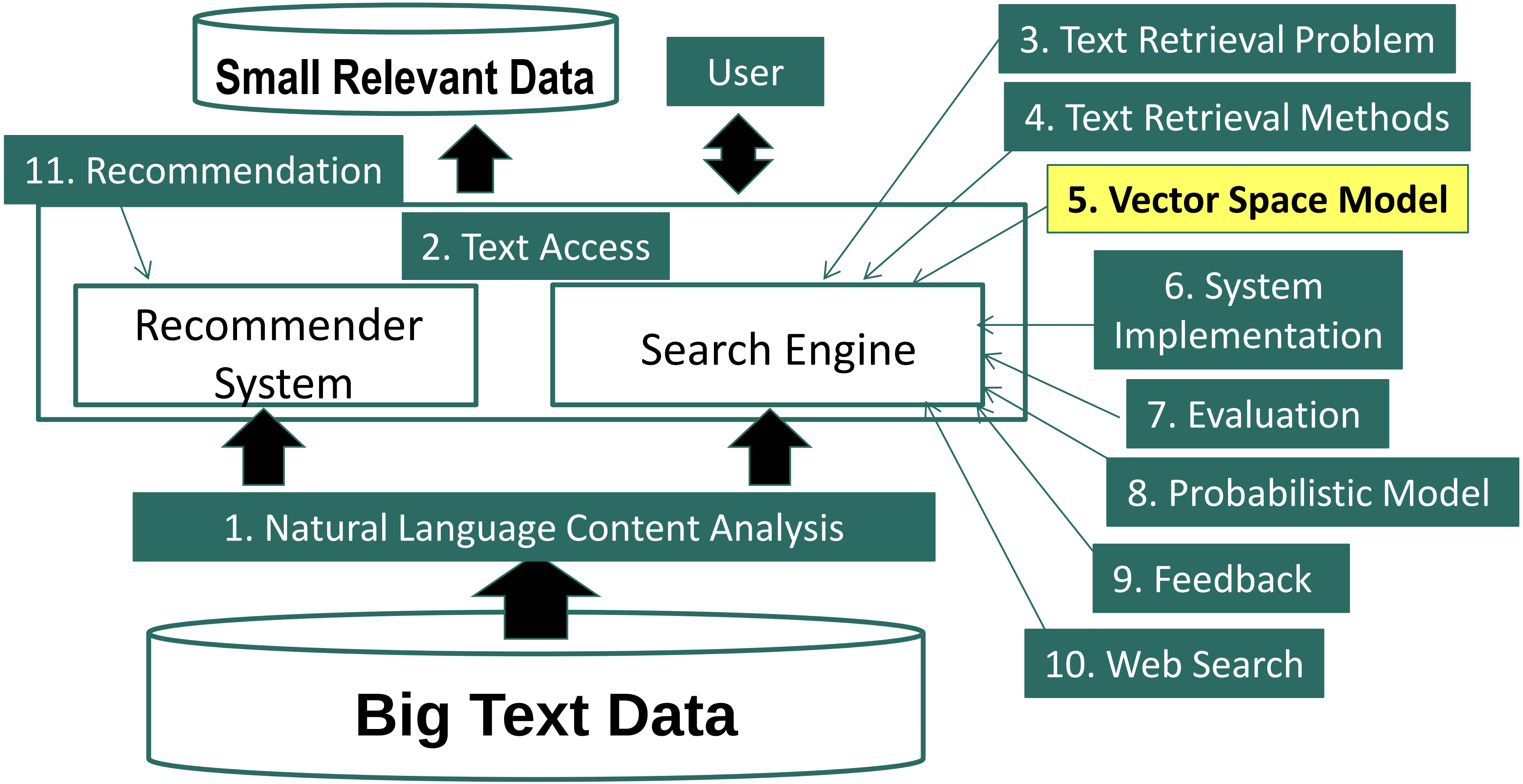
- Sublinear TF Transformation is needed to
  - capture the intuition of “diminishing return” from higher TF
  - avoid dominance by one single term over all others
- BM25 Transformation
  - has an upper bound
  - is robust and effective
- Ranking function with BM25 TF (k >=0)

$$f(q,d) = \sum_{i=1}^{N} x_i y_i = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k} \log \frac{M+1}{df(w)}$$

Slide adapted from ChengXiang Zhai's presentation

Vector Space Model: Doc Length Normalization

Course Schedule



Slide adapted from ChengXiang Zhai's presentation

What about Document Length?

Query = “news about presidential campaign”

d4

… news of presidential campaign …
… presidential candidate …

100 words

d6 > d4?

d6

… campaign……… campaign…………………… ……

5000 words

…………………………………………………………………………

……….news……………………………………………………………

…………………………………………………………………………

………………………………………………………… news…..

…………………………………………………………………………

……………………………………… ……………………………

……………………… presidential ……. presidential……

Slide adapted from ChengXiang Zhai's presentation

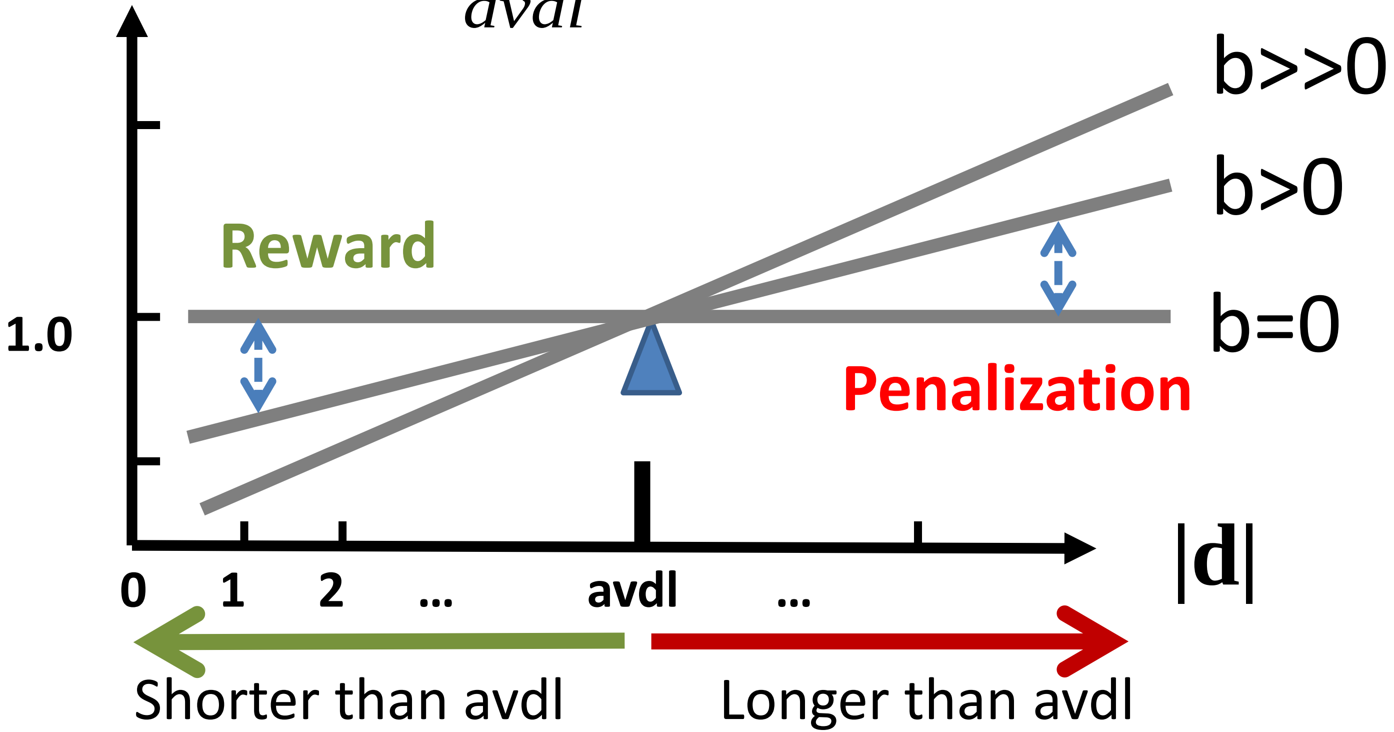
Document Length Normalization

- Penalize a long doc with a doc length normalizer
  - Long doc has a better chance to match any query
  - Need to avoid over-penalization
- A document is long because
  - it uses more words ➔ more penalization
  - it has more contents ➔ less penalization
- Pivoted length normalizer: average doc length as "pivot"
  - Normalizer = 1 if |d| =average doc length (avdl)

Pivoted Length Normalization

$$normalizer = 1 - b + b\frac{|d|}{avdl}$$

$$b \in [0,1]$$



Slide adapted from ChengXiang Zhai's presentation

State of the Art VSM Ranking Functions

- Pivoted Length Normalization VSM [Singhal et al 96]

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

- BM25/Okapi [Robertson & Walker 94]

$$b\in[0,1]$$
$$k_1,k_3\in[0,+\infty)$$

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

Slide adapted from ChengXiang Zhai's presentation

Further Improvement of VSM?

- Improved instantiation of **dimension?**
  - stemmed words, stop word removal, phrases, latent semantic indexing (word clusters), character n-grams, …
  - bag-of-words with phrases is often sufficient in practice
  - Language-specific and domain-specific tokenization is important to ensure “normalization of terms”
- Improved instantiation of **similarity function?**
  - cosine of angle between two vectors?
  - Euclidean?
  - dot product seems still the best (sufficiently general especially with appropriate term weighting)

Further Improvement of BM25

- BM25F [Robertson & Zaragoza 09]
  - Use BM25 for documents with structures (“F”=fields)
  - Key idea: combine the frequency counts of terms in all fields and then apply BM25 (instead of the other way)
- BM25+ [Lv & Zhai 11]
  - Address the problem of over penalization of long documents by BM25 by adding a small constant to TF
  - Empirically and **analytically** shown to be better than BM25

Summary of Vector Space Model

- Relevance(q,d) = similarity(q,d)
- Query and documents are represented as vectors
- Heuristic design of ranking function
- Major term weighting heuristics
  - TF weighting and transformation
  - IDF weighting
  - Document length normalization
- BM25 and Pivoted normalization seem to be most effective

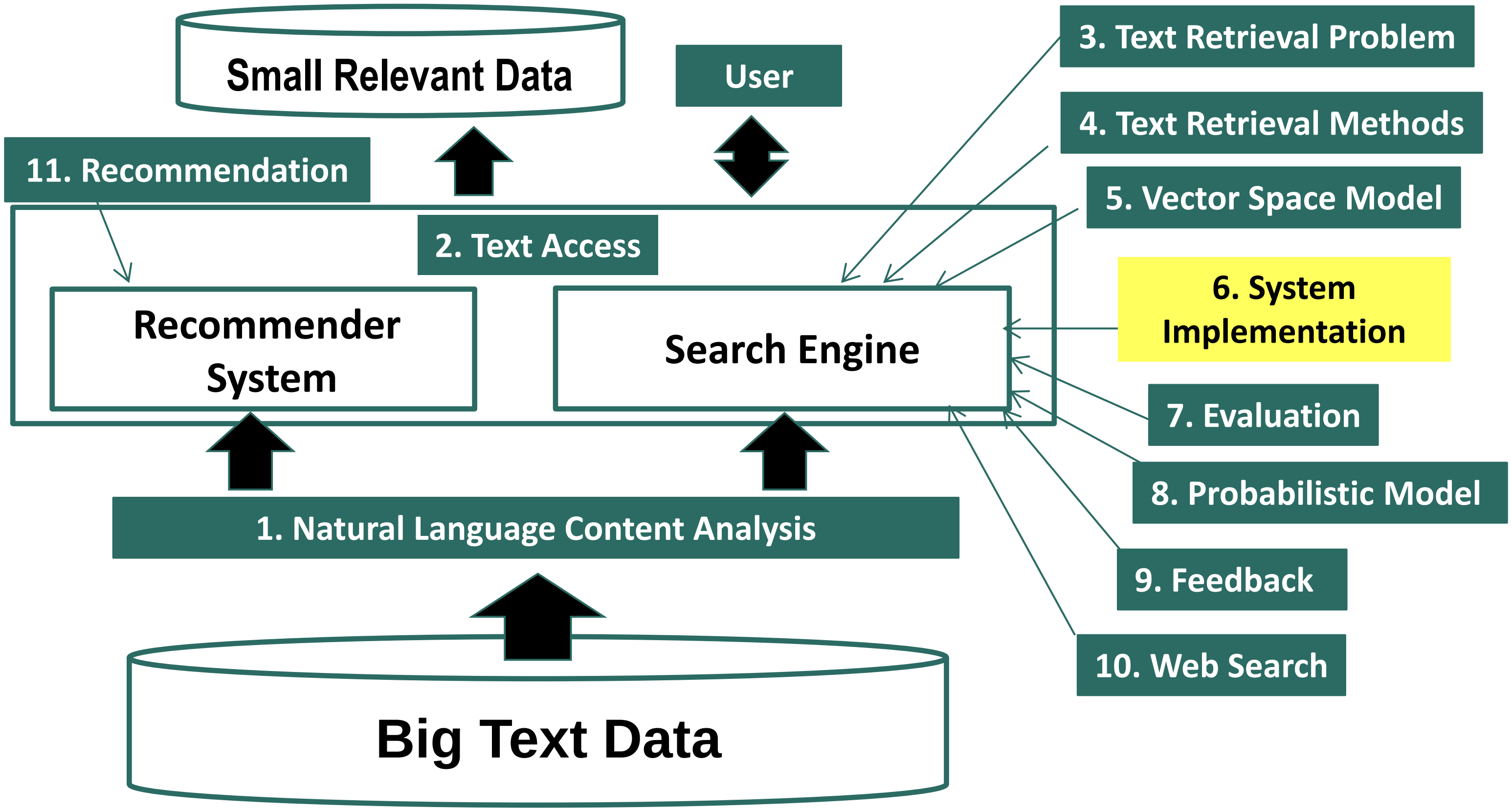
Additional Readings

- A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of ACM SIGIR 1996.*
- S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval, *Proceedings of ACM SIGIR 1994*.
- S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond, *Found. Trends Inf. Retr.* 3, 4 (April 2009).
- Y. Lv, C. Zhai, Lower-bounding term frequency normalization. In *Proceedings of ACM CIKM 2011.*

Slide adapted from ChengXiang Zhai's presentation

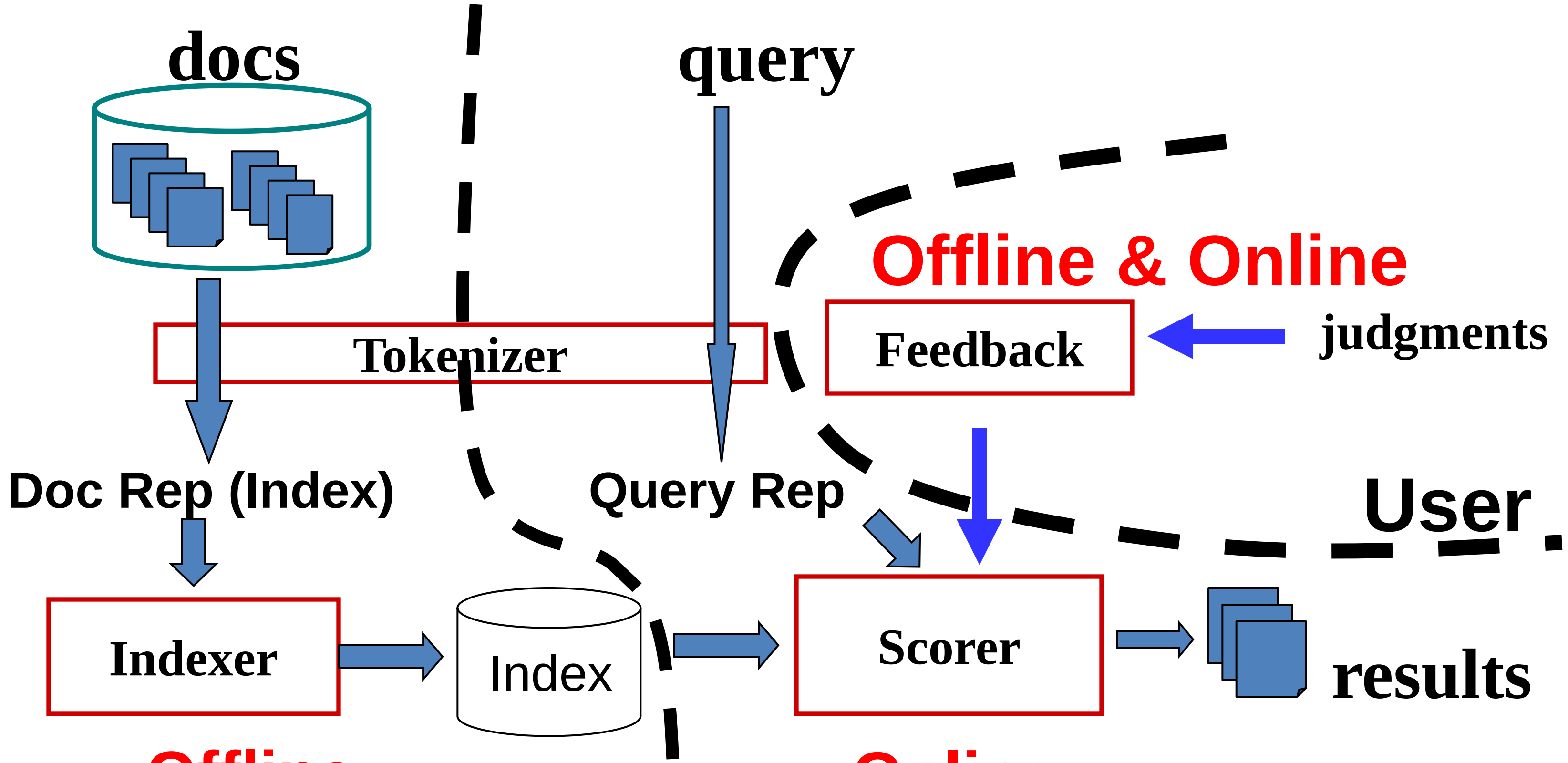
Implementation of Text Retrieval Systems

Implementation of Text Retrieval Systems



Slide adapted from ChengXiang Zhai's presentation

Typical TR System Architecture



Slide adapted from ChengXiang Zhai's presentation

Tokenization

- Normalize lexical units: Words with similar meanings should be mapped to the same indexing term
- Stemming: Mapping all inflectional forms of words to the same root form, e.g.
  - computer -> compute
  - computation -> compute
  - computing -> compute
- Some languages (e.g., Chinese) pose challenges in word segmentation

Indexing

- Indexing = Convert documents to data structures that enable fast search (precomputing as much as we can)
- Inverted index is the dominating indexing method for supporting basic search algorithms
- Other indices (e.g., document index) may be needed for feedback

Inverted Index Example

doc 1

… **news about**

doc 2

… **news about** organic food **campaign**…

doc 3

… **news** of **presidential campaign** …
… **presidential** candidate …

Dictionary (or lexicon)

Term	# docs	Total freq
news	3	3
campaign	2	2
presidential	1	2
food	1	1
…	…	…

Postings

Doc id	Freq	Position
1	1	p1
2	1	p2
3	1	p3
2	1	p4
3	1	p5
3	2	p6,p7
2	1	p8
…	…	
…	…	

Slide adapted from ChengXiang Zhai's presentation

Inverted Index for Fast Search

- Single-term query?
- Multi-term Boolean query?
  - Must match term “A” AND term “B”
  - Must match term “A” OR term “B”
- Multi-term keyword query
  - Similar to disjunctive Boolean query (“A” OR “B”)
  - Aggregate term weights
- More efficient than sequentially scanning docs (why?)

Slide adapted from ChengXiang Zhai's presentation

Empirical Distribution of Words

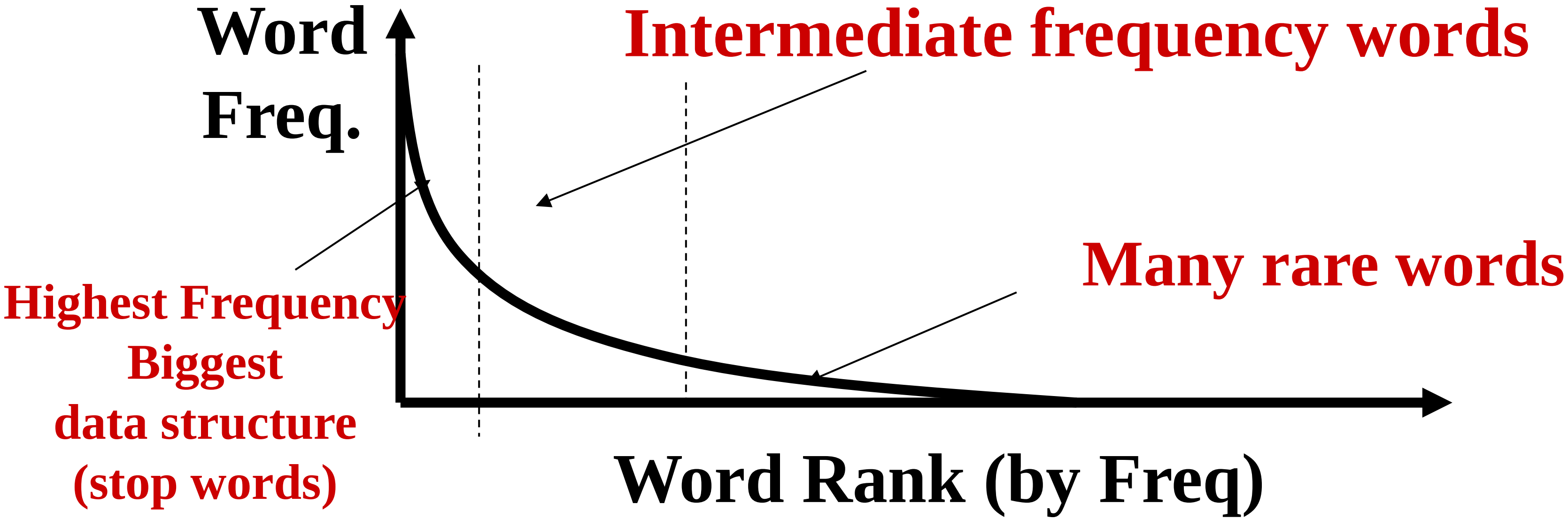
- There are stable language-independent patterns in how people use natural languages
- A few words occur very frequently; most occur rarely. E.g., in news articles,
  - Top 4 words: 10~15% word occurrences
  - Top 50 words: 35~40% word occurrences
- The most frequent word in one corpus may be rare in another

Slide adapted from ChengXiang Zhai's presentation

Zipf's Law

- rank * frequency ≈ constant

$$F(w) = \frac{C}{r(w)^{\alpha}} \qquad \alpha \approx 1, C \approx 0.1$$

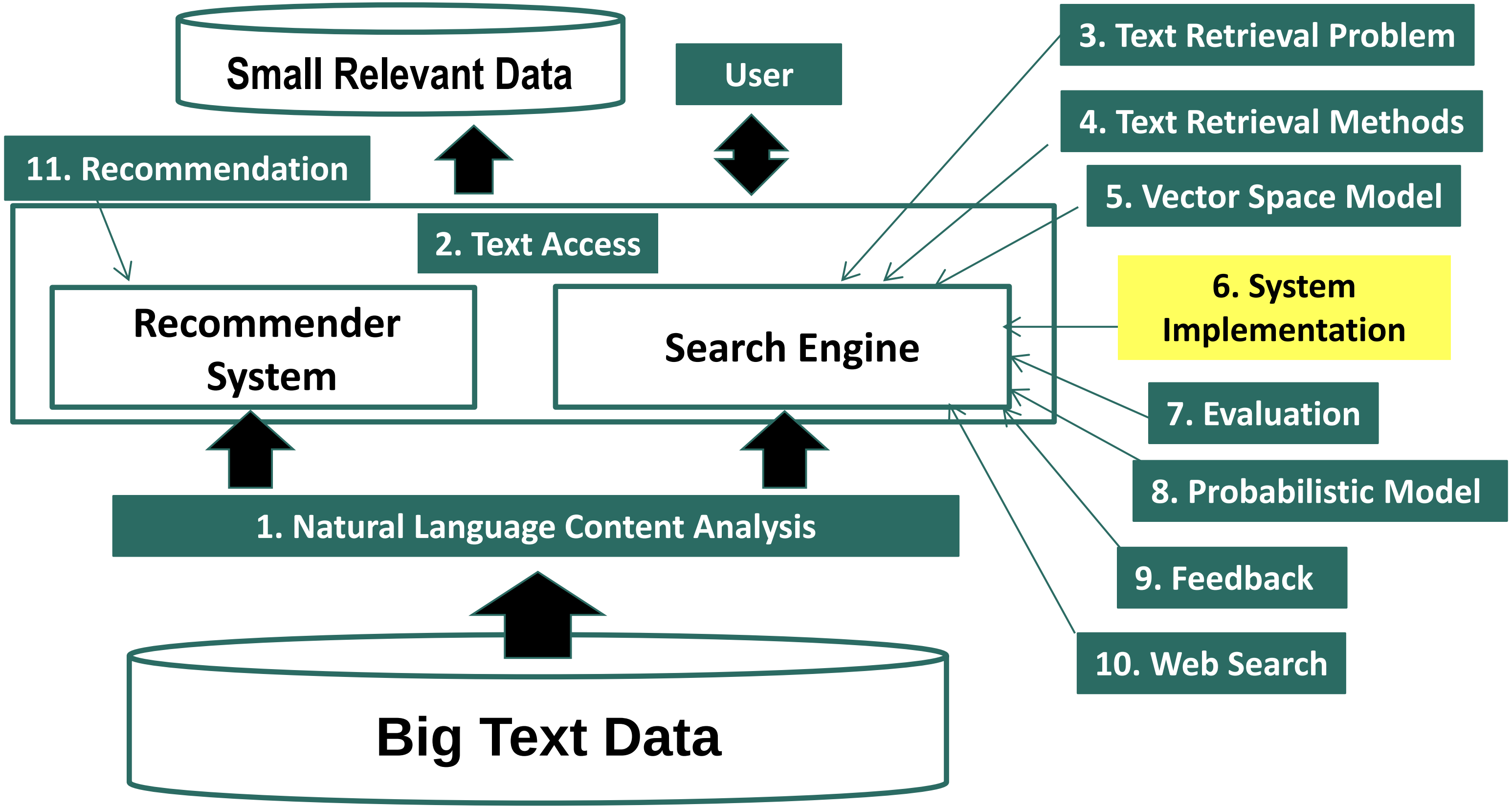


Data Structures for Inverted Index

- Dictionary: modest size
  - Needs fast random access
  - Preferred to be in memory
  - Hash table, B-tree, trie, …
- Postings: huge
  - Sequential access is expected
  - Can stay on disk
  - May contain docID, term freq., term pos, etc
  - Compression is desirable

System Implementation: Inverted Index Construction

System Implementation: Inverted Index Construction

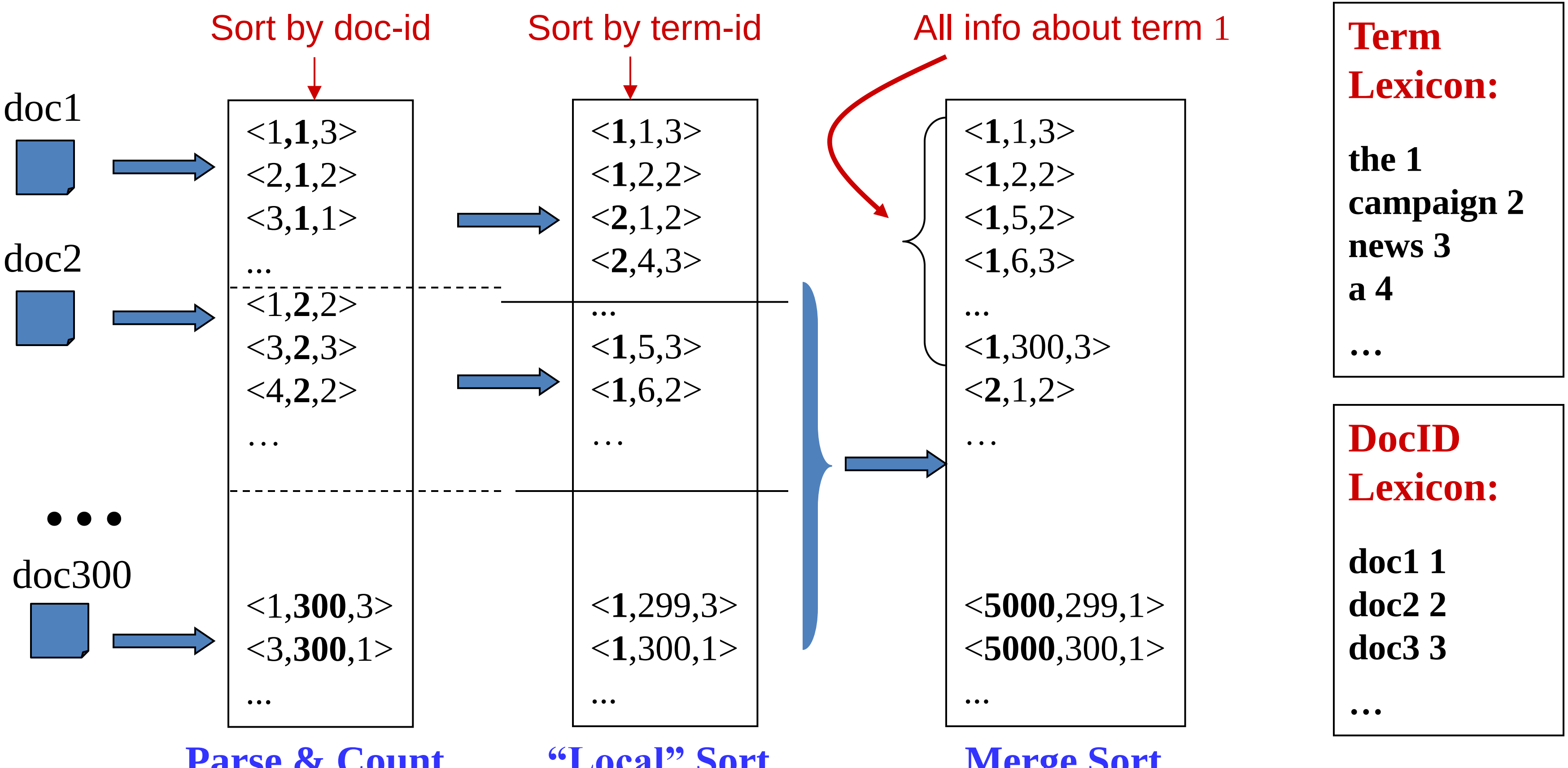


Slide adapted from ChengXiang Zhai's presentation

Constructing Inverted Index

- The main difficulty is to build a huge index with limited memory
- Memory-based methods: not usable for large collections
- Sort-based methods:
  – Step 1: Collect local (termID, docID, freq) tuples
  – Step 2: Sort local tuples (to make “runs”)
  – Step 3: Pair-wise merge runs
  – Step 4: Output inverted file

Sort-based Inversion



Slide adapted from ChengXiang Zhai's presentation

Inverted Index Compression

- In general, leverage skewed distribution of values and use variable-length encoding
- TF compression
  - Small numbers tend to occur far more frequently than large numbers (why?)
  - Fewer bits for small (high frequency) integers at the cost of more bits for large integers
- Doc ID compression
  - "d-gap" (store difference): d1, d2-d1, d3-d2,…
  - Feasible due to sequential access
- Methods: Binary code, unary code, γ-code, δ-code, …

Integer Compression Methods

- Binary: equal-length coding
- Unary: $x \geq 1$ is coded as x-1 one bits followed by 0, e.g., 3=> 110; 5=>11110
- γ-code: x=> unary code for $1+\lfloor \log x \rfloor$ followed by uniform code for $x-2^{\lfloor \log x \rfloor}$ in $\lfloor \log x \rfloor$ bits, e.g., 3=>101, 5=>11001
- δ-code: same as γ-code ,but replace the unary prefix with γ-code. E.g., 3=>1001, 5=>10101

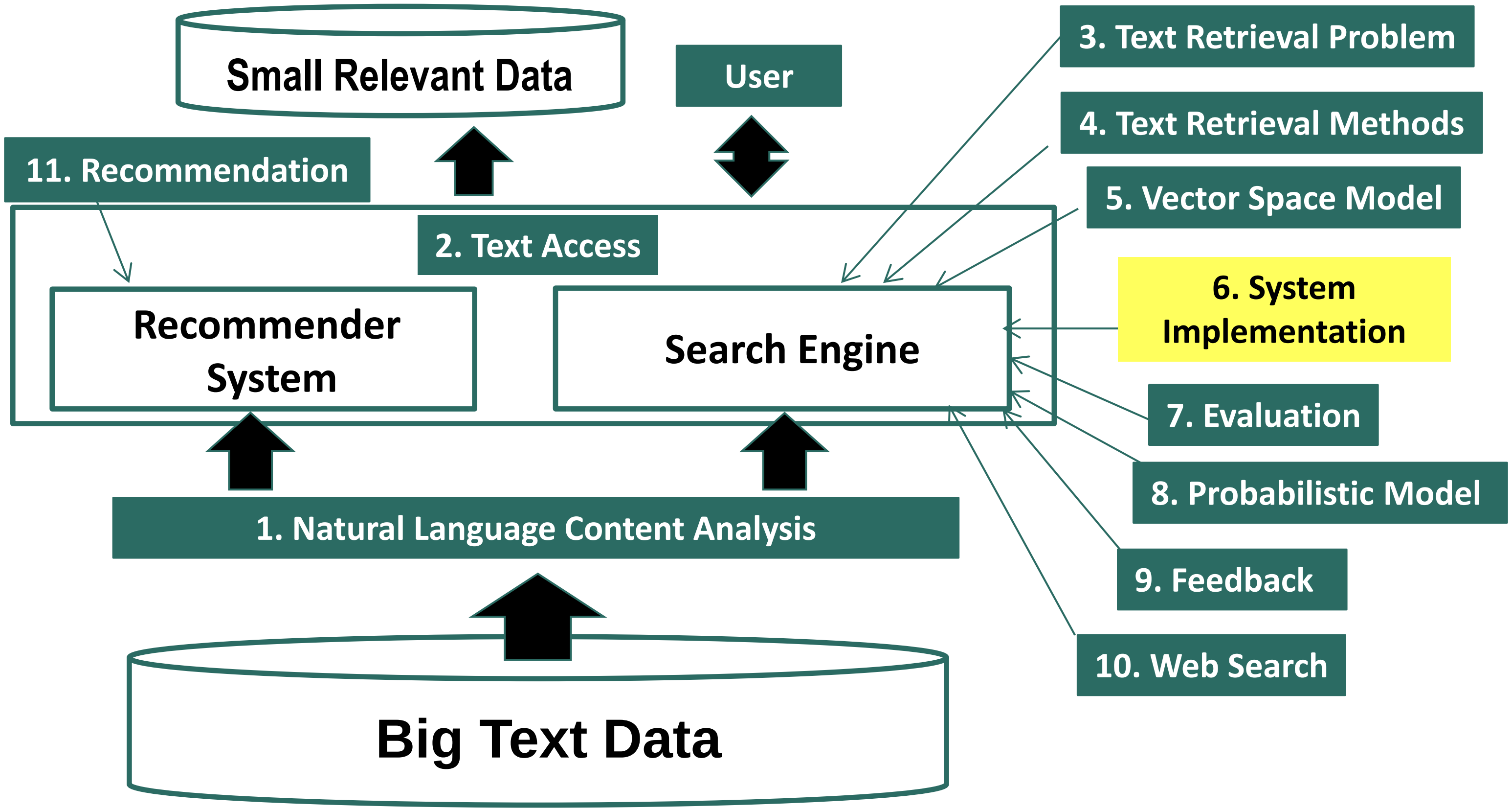
Slide adapted from ChengXiang Zhai's presentation

Uncompress Inverted Index

- Decoding of encoded integers
  – Unary decoding: count 1's until seeing a zero
  – γ-decoding
    - first decode the unary part; let value be k+1
    - read k more bits decode them as binary code; let value be r
    - the value of the encoded number is $\mathbf{2^k+r}$
- Decode doc IDs encoded using d-gap
  – Let the encoded ID list be x1, x2, x3, ….
  – Decode x1 to obtain doc ID1; then decode x2 and add the recovered value to the doc ID1 just obtained
  – Repeatedly decode x3, x4, …., and the recovered value to the previous doc ID.

System Implementation: Fast Search

System Implementation: Fast Search



Slide adapted from ChengXiang Zhai's presentation

How to Score Documents Quickly

General Form of Scoring Function

Final score **adjustment**

$$f(q,d) = f_a(\boldsymbol{h}(\underbrace{\boldsymbol{g}(\boldsymbol{t_1},\boldsymbol{d},\boldsymbol{q})},\ldots,\underbrace{\boldsymbol{g}(\boldsymbol{t_k},\boldsymbol{d},\boldsymbol{q})}),f_d(d),f_q(q))$$

Weight **aggregation**

Weight a **matched** query term in d

A General Algorithm for Ranking Documents

$$\mathrm{f}(\mathrm{q},\mathrm{d}) = \mathrm{f}_{\mathrm{a}}(\boldsymbol{h}(\,\boldsymbol{g}(\boldsymbol{t}_1, \boldsymbol{d}, \boldsymbol{q}), \ldots, \boldsymbol{g}(\boldsymbol{t}_k, \boldsymbol{d}, \boldsymbol{q})\,), f_d(d), f_q(q))$$

- $f_d(d)$ and $f_q(q)$ are pre-computed
- Maintain a score accumulator for each **d** to compute ***h***
- For each query term $\boldsymbol{t_i}$
  - Fetch the inverted list $\{(d_1,f_1),\ldots,(d_n,f_n)\}$
  - For each entry (d_j,f_j), compute **$g(t_i,d_j,q)$**, and update score accumulator for doc d_i to incrementally compute ***h***
- Adjust the score to compute **f_a**, and sort

Slide adapted from ChengXiang Zhai's presentation

An Example: Ranking Based on TF Sum

$f(d,q)=g(t_1,d,q)+\ldots+g(t_k,d,q)$ where $g(t_i,d,q) = c(t_i,d)$

Query = **"info security"**

Info: (d1, 3), (d2, 4), (d3, 1), (d4, 5)
Security: (d2, 3), (d4,1), (d5, 3)

		Accumulators:	d1	d2	d3	d4	d5
			0	0	0	0	0
info	(d1,3)	=>	**3**	0	0	0	0
info	(d2,4)	=>	3	**4**	0	0	0
info	(d3,1)	=>	3	4	**1**	0	0
info	(d4,5)	=>	3	4	1	**5**	0
security	(d2,3)	=>	3	**7**	1	5	0
security	(d4,1)	=>	3	7	1	**6**	0
security	(d5,3)	=>	3	7	1	6	**3**

Slide adapted from ChengXiang Zhai's presentation

Further Improving Efficiency

- Caching (e.g., query results, list of inverted index)

- Keep only the most promising accumulators

- Scaling up to the Web-scale? (need parallel processing)

Some Text Retrieval Toolkits

- Lucene: http://lucene.apache.org/
- Lemur/Indri: http://www.lemurproject.org/
- Terrier: http://terrier.org/
- MeTA: http://meta-toolkit.github.io/meta/
- More can be found at http://timan.cs.uiuc.edu/resources

Summary of System Implementation

- Inverted index and its construction
  - Preprocess data as much as we can
  - Compression when appropriate
- Fast search using inverted index
  - Exploit inverted index to accumulate scores for documents matching a query term
  - Exploit Zipf’s law to avoid touching many documents not matching any query term
  - Can support a wide range of ranking algorithms
- Great potential for further scaling up using distributed file system, parallel processing, and caching

Additional Readings

- Ian H. Witten, Alistair Moffat, Timothy C. Bell: Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition. Morgan Kaufmann, 1999.
- Stefan Büttcher, Charles L. A. Clarke, Gordon V. Cormack: Information Retrieval - Implementing and Evaluating Search Engines. MIT Press, 2010.