TEXT RETRIEVAL
AND
SEARCH ENGINES

The basic concepts, principles, and the major techniques in text retrieval, which is the underlying science of search engines.

Course author:

ChengXiang Zhai

University of Illinois at Urbana-Champaign
&
Coursera

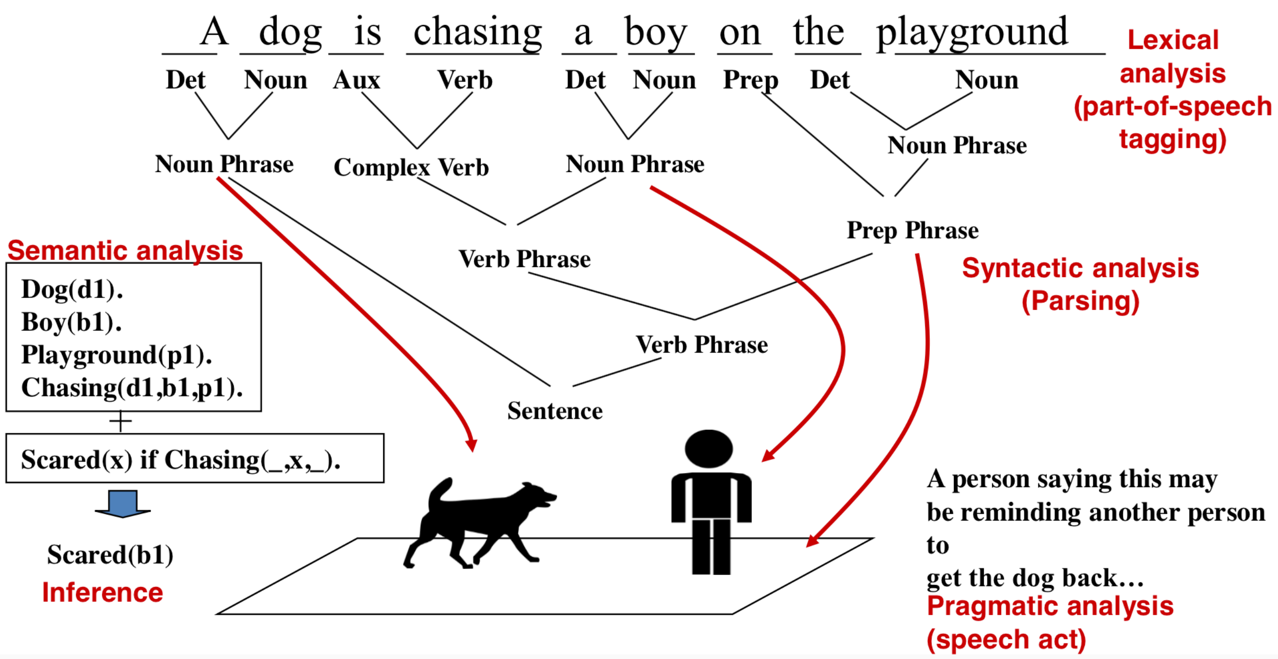
2015

Contents

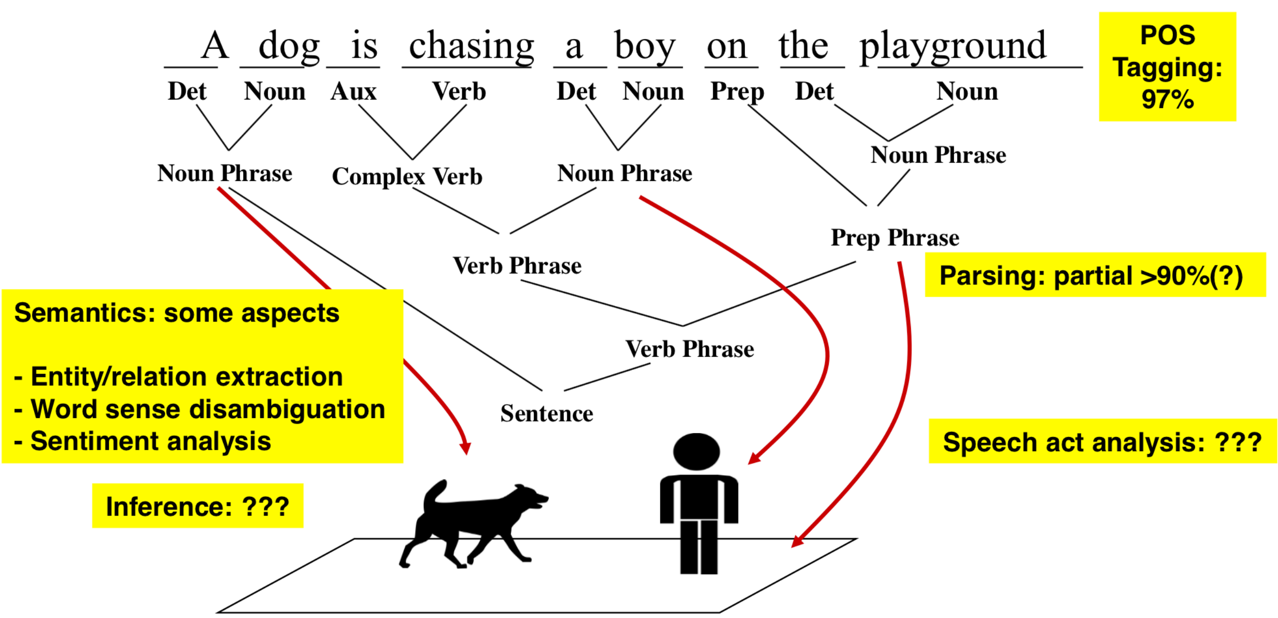
1 Natural Language Content Analysis

NLP = Natural Language Processing

1.1 An Example of NLP



1.2 The State of the Art



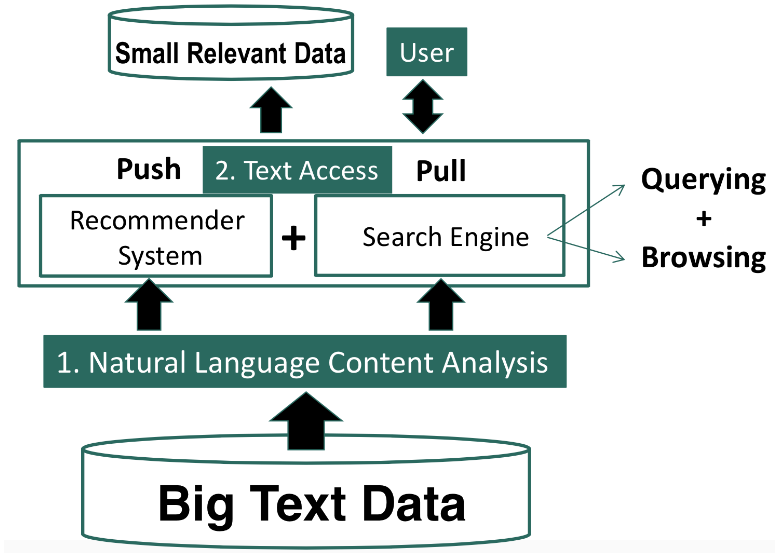
1.3 Recommended reading

- Chris Manning and Hinrich Schütze, «Foundations of Statistical Natural Language Processing», MIT Press. Cambridge, MA: May 1999.

2 Text Access

2.1 Two Modes of Text Access: Pull vs. Push

- Pull Mode (search engines) – Users take initiative
  - Ad hoc information need
- Push Mode (recommender systems)
  - Systems take initiative
  - Stable information need or system has good knowledge about a user's need



2.2 Pull Mode: Querying vs. Browsing

- Querying
  - User enters a (keyword) query
  - System returns relevant documents
  - Works well when the user knows what keywords to use
- Browsing
  - User navigates into relevant information by following a path enabled by the structures on the documents
  - Works well when the user wants to explore information, doesn't know what keywords to use, or can't conveniently enter a query

2.3 Recommended reading

- N. J. Belkin and W. B. Croft. 1992. «Information filtering and information retrieval: two sides of the same coin?» Commun. ACM 35, 12 (Dec. 1992), 29-38.

3 Text Retrieval Problem

3.1 What Is Text Retrieval?

TR = Text Retrieval[1]

- Collection of text documents exists
- User gives a query to express the information need
- Search engine system returns relevant documents to users
- Often called "information retrieval" (IR), but IR is actually much broader
- Known as «search technology» in industry

TR is an empirically defined problem:

- Can't mathematically prove one method is better than another
- Must rely on empirical evaluation involving users!

3.2 Formal Formulation of TR

- **Vocabulary**: $V = \{w_1, w_2, \dots, w_N\}$ of language
- **Query**: $q = q_1, \dots, q_m$, where $q_i \in V$
- **Document**: $d_i = d_{i1}, \dots, d_{im_i}$, where $d_{ij} \in V$
- **Collection**: $C = \{d_1, \dots, d_M\}$
- **Set of relevant documents**: $R(q) \subseteq C$
  - Generally unknown and user-dependent
  - Query is a «hint» on which doc is in $R(q)$
- **Task**: compute $R'(q)$, an approximation of $R(q)$

3.3 How to Compute $R'(q)$

- Strategy 1: Document selection
  - $R'(q) = \{d \in C \mid f(d,q) = 1\}$, where $f(d,q) \in \{0,1\}$ is an indicator function or binary classifier
  - System must decide if a doc is relevant or not (absolute relevance)
- Strategy 2 (generally preferred): Document ranking
  - $R'(q) = \{d \in C \mid f(d,q) > \theta\}$, where $f(d,q) \in \Re$ is a relevance measure function; θ is a cutoff determined by the user
  - System only needs to decide if one doc is more likely relevant than another (relative relevance)

[1]Retrieval - поиск

3.4 Theoretical Justification for Ranking

Probability Ranking Principle [Robertson 77]: Returning a ranked list of documents in descending order of probability that a document is relevant to the query is the optimal strategy under the following two assumptions:

- The utility of a document (to a user) is independent of the utility of any other document
- A user would browse the results sequentially

3.5 Recommended reading

- S.E. Robertson, «The probability ranking principle in IR». Journal of Documentation 33, 294-304, 1977
- **C. J. van Rijsbergen, «Information Retrieval», 2nd Edition**, Butterworth-Heinemann, Newton, MA, USA, 1979

4 Overview of Text Retrieval Methods

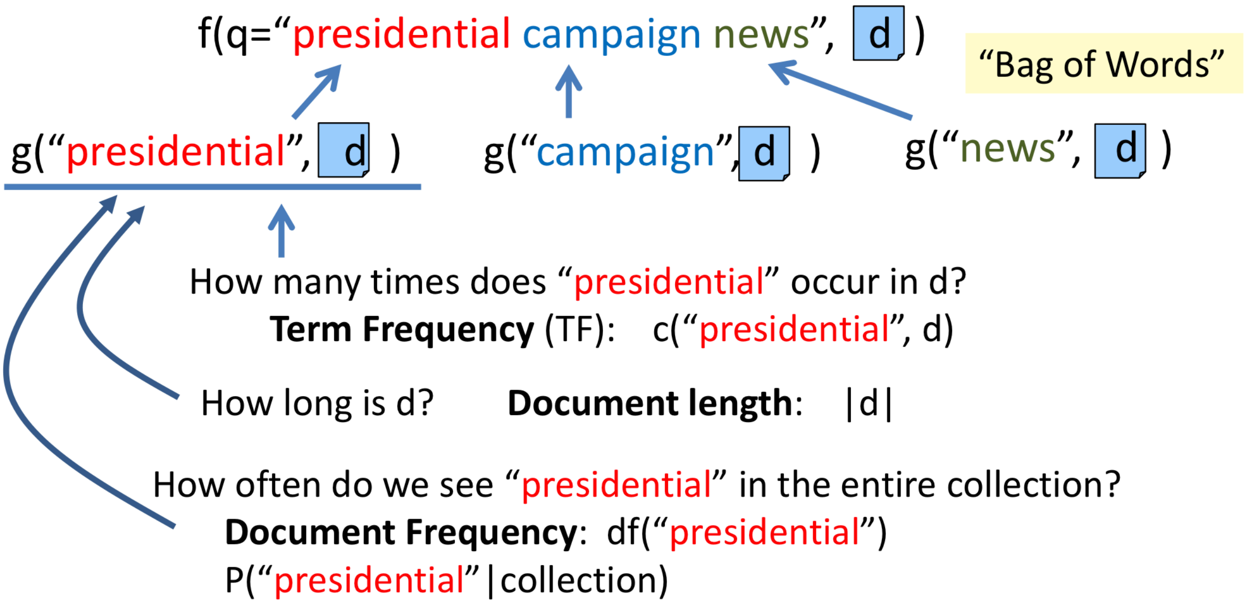
4.1 How to Design a Ranking Function

- **Query**: $q = q_1, \dots, q_m$, where $q_i \in V$
- **Document**: $d = d_1, \dots, d_n$, where $d_i \in V$
- **Ranking function**: $f(q, d) \in \Re$
- **Key challenge**: how to measure the likelihood that document d is relevant to query q
- **Retrieval model**: formalization of relevance (give a computational definition of relevance)

4.2 Retrieval Models

- **Similarity-based models**: $f(q, d) = similarity(q, d)$
  - Vector space model
- **Probabilistic models**: $f(d, q) = p(R = 1 \mid d, q)$, where $R \in 0, 1$
  - Classic probabilistic model
  - Language model
  - Divergence-from-randomness model
- **Probabilistic inference model**: $f(q, d) = p(d \to q)$
- **Axiomatic model**: $f(q, d)$ must satisfy a set of constraints

4.3 Common Ideas in State of the Art Retrieval Models



State of the art ranking functions tend to rely on:

- Bag of words representation
- Term Frequency (TF) and Document Frequency (DF) of words
- Document length

4.4 Which Model Works the Best?

When optimized, the following models tend to perform equally well [Fang et al. 11]:

- **Pivoted length normalization – BM25**
- Query likelihood
- PL2

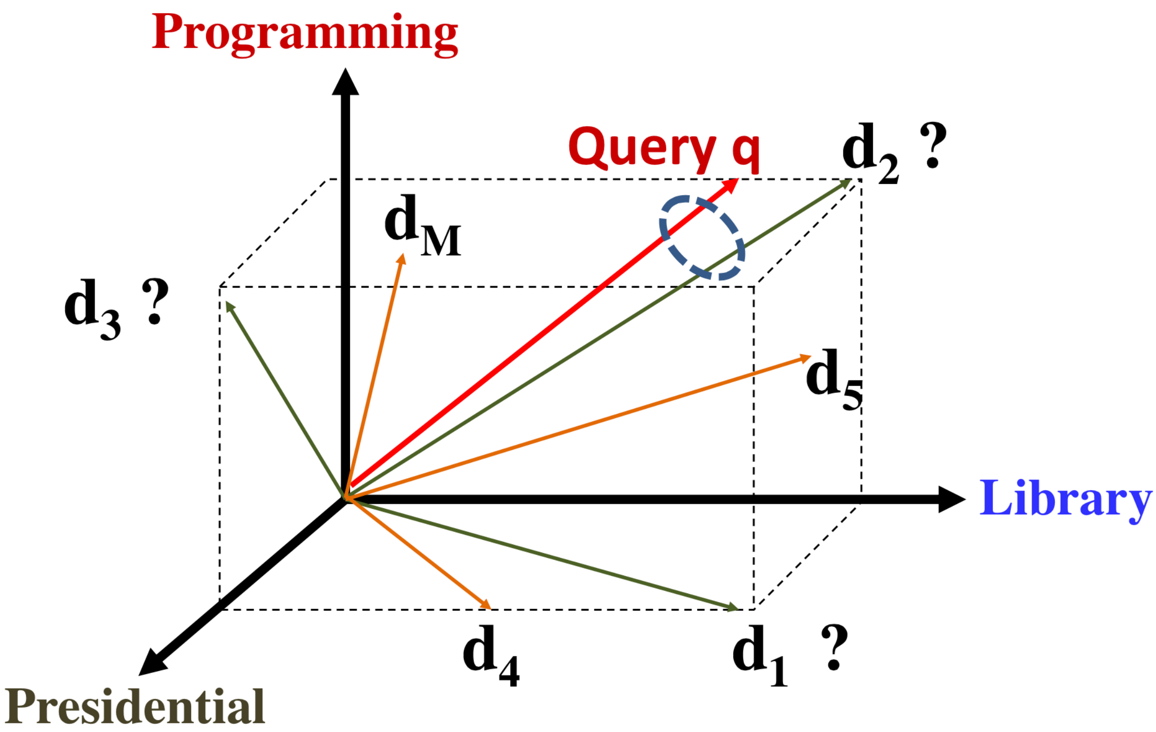
4.5 Recommended reading

- Hui Fang, Tao Tao, and Chengxiang Zhai. 2011. «Diagnostic Evaluation of Information Retrieval Models». ACM Trans. Inf. Syst. 29, 2, Article 7 (April 2011)
- ChengXiang Zhai, «Statistical Language Models for Information Retrieval», Morgan & Claypool Publishers, 2008. (Chapter 2)

5 Vector Space Retrieval Model

VSM - Vector Space Model

5.1 Vector Space Model (VSM): Illustration

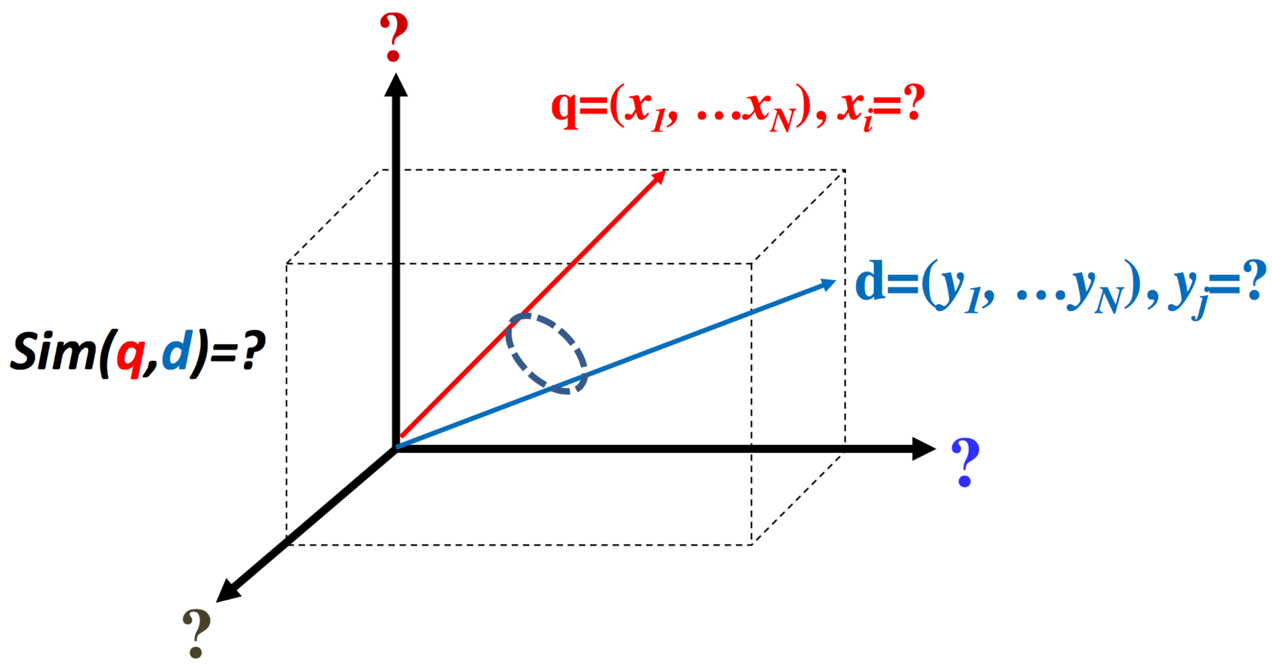


5.2 VSM Is a Framework

- Represent a doc/query by a term vector
  - **Term**: basic concept, e.g., word or phrase
  - Each term defines one dimension
  - N terms define an **N-dimensional space**
  - **Query vector**: $q = (x_1, \dots x_N), x_i \in \Re$ is query term weight
  - **Doc** vector: $d = (y_1, \dots y_N), y_j \in \Re$ is doc term weight
- $relevance(q, d) \propto similarity(q, d) = f(q, d)$

5.3 What VSM Doesn't Say

- How to define/select the "basic concept" – Concepts are assumed to be orthogonal
- How to place docs and query in the space (= how to assign term weights)
  - Term weight in query indicates importance of term
  - Term weight in doc indicates how well the term characterizes the doc
- How to define the similarity measure



5.4 Simplest VSM = Bit-Vector + Dot-Product + BOW

$q=(x_1, \dots x_N)$

$d=(y_1, \dots y_N)$

$x_i, y_i \in \{0,1\}$

1: word W_i is **present**

0: word W_i is **absent**

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

Simplest VSM:

- Dimension = word
- Vector = 0-1 bit vector (word presence/absence)
- Similarity = dot product
- f(q,d) = number of distinct query words matched in d

5.5 Improved Instantiation

Improved VSM:

- Dimension = word
- Vector = TF-IDF weight vector
- Similarity = dot product

5.6 Improved VSM with Term Frequency (TF) Weighting

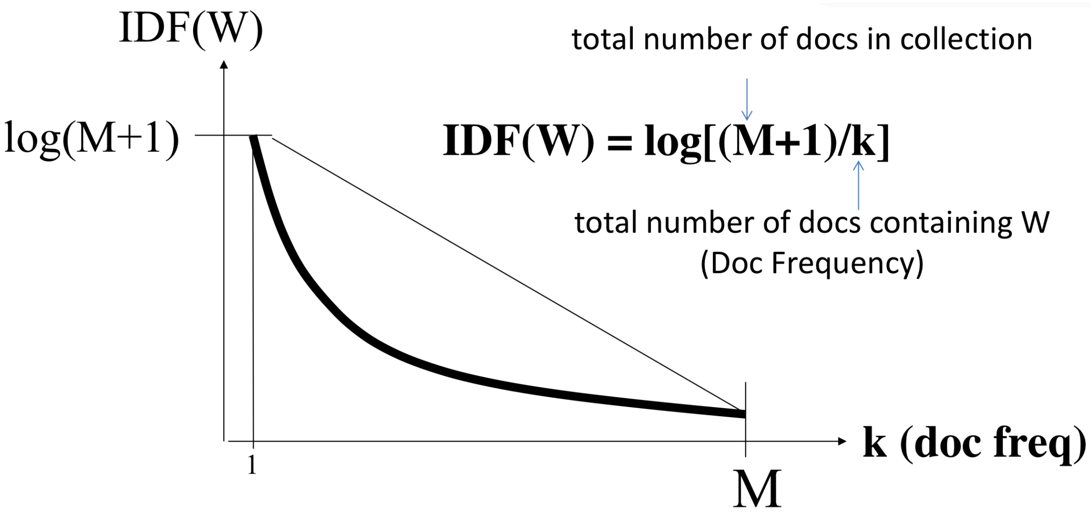
$q=(x_1, \ldots x_N)$ $\boxed{x_i = \text{count of word } W_i \text{ in query}}$

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

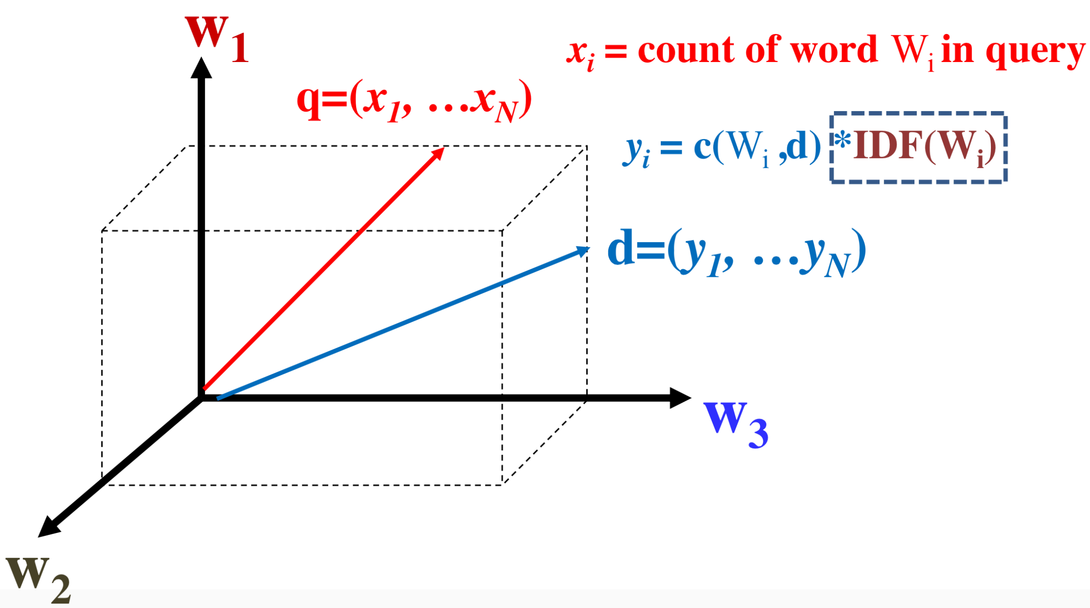
$$Sim(q,d)=q.d= x_1 y_1 + \ldots + x_N y_N = \sum_{i=1}^{N} x_i y_i$$

5.7 IDF Weighting: Penalizing Popular Terms

IDF — inverse document frequency



5.8 Adding Inverse Document Frequency (IDF)



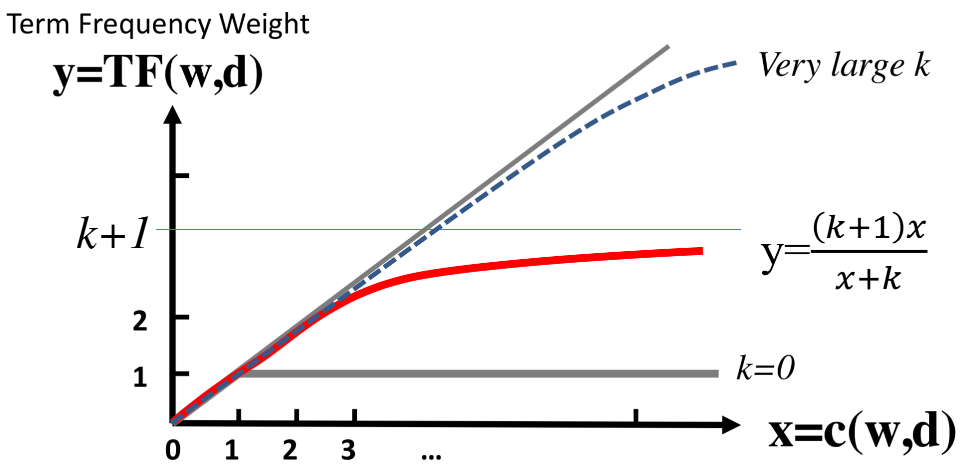
5.9 Ranking Function with TF-IDF Weighting

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

- $w \in q \cap d$ - all matched query (q) words in document (d)
- $c(w,q)$ - count of word w in document d
- M - total number of documents in collection
- $df(w)$ - Doc Frequency (total number of documents containing word w)

5.10 TF Transformation: BM25 Transformation

BM = Best Matching



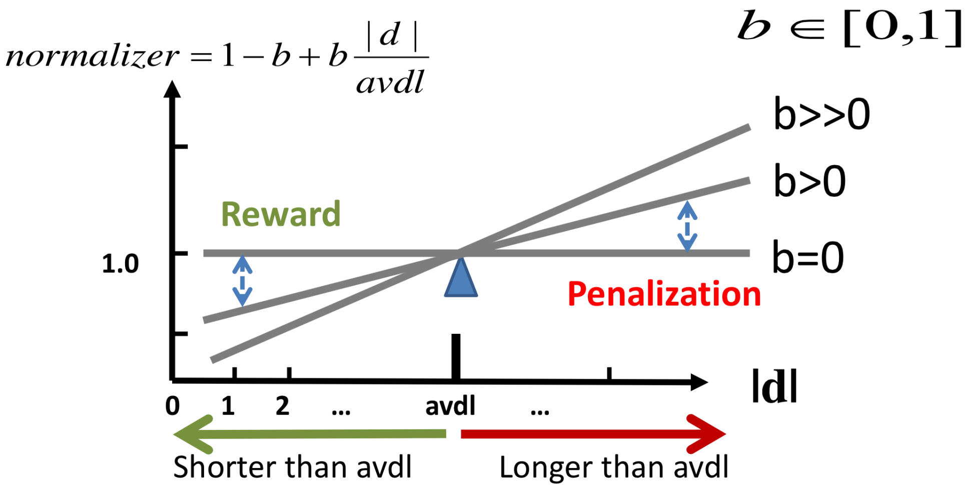
5.11 Summary

- 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 ($k >= 0$):

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

5.12 Pivoted Length Normalization

Pivoted length normalizer: use average doc length as «pivot»[2]. Normalizer = 1 if $|d|$ = average doc length (avdl).



5.13 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]:

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

5.14 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)

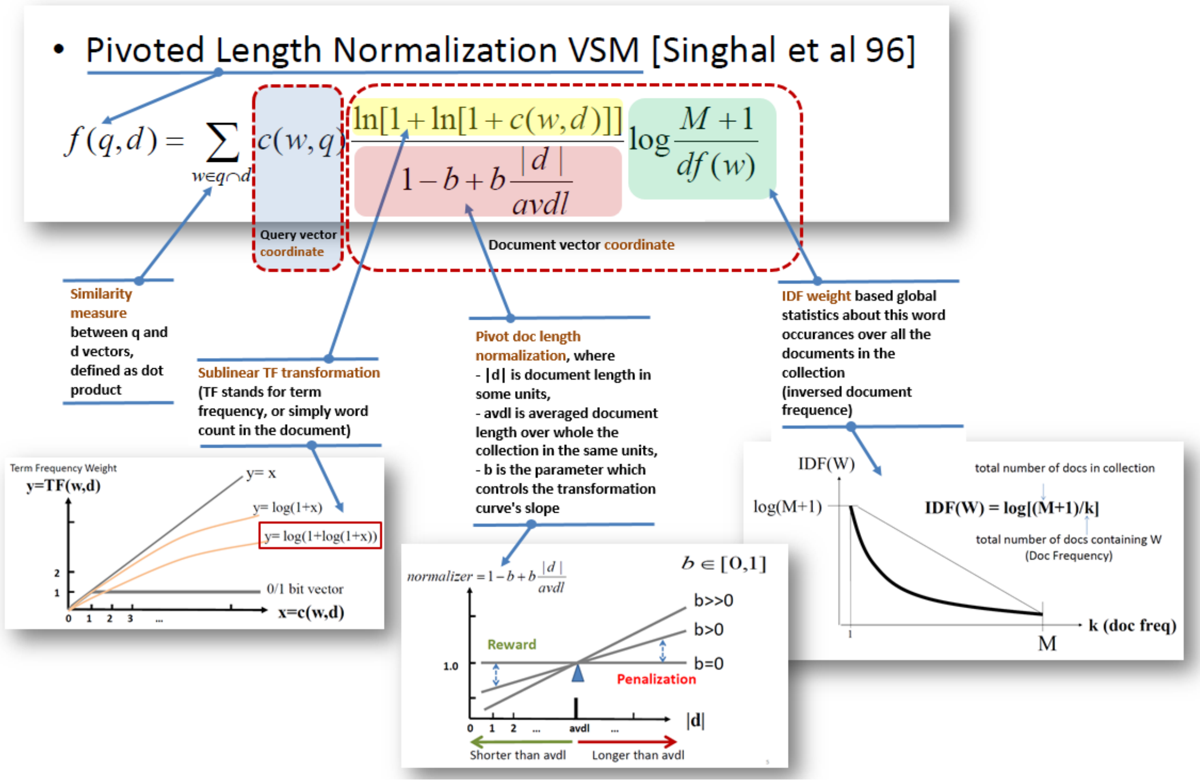
[2] Pivot - стержень; точка опоры, вращения

5.15 Further Improvement of BM25

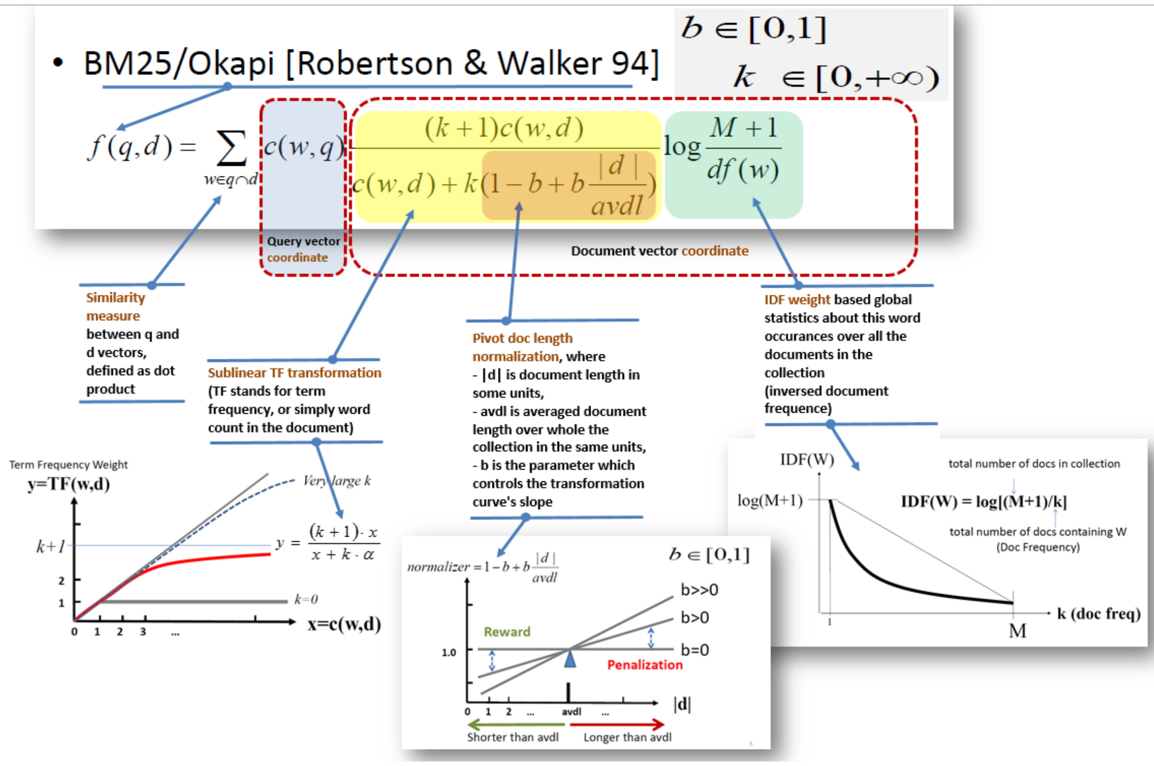
- BM25F [Robertson & Zaragoza 09]
  - Use BM25 for documents with structures («F»=fields)
  - Key idea: combine the frequency counts of terms in all fields and then apply BM25 (instead of the other way)
- BM25+ [Lv & Zhai 11]
  - Address the problem of over penalization of long documents by BM25 by adding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

5.16 Summary of Vector Space Model

- Relevance(q,d) = similarity(q,d)
- Query and documents are represented as vectors
- Heuristic[3] 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



[3]Heuristic - эвристический

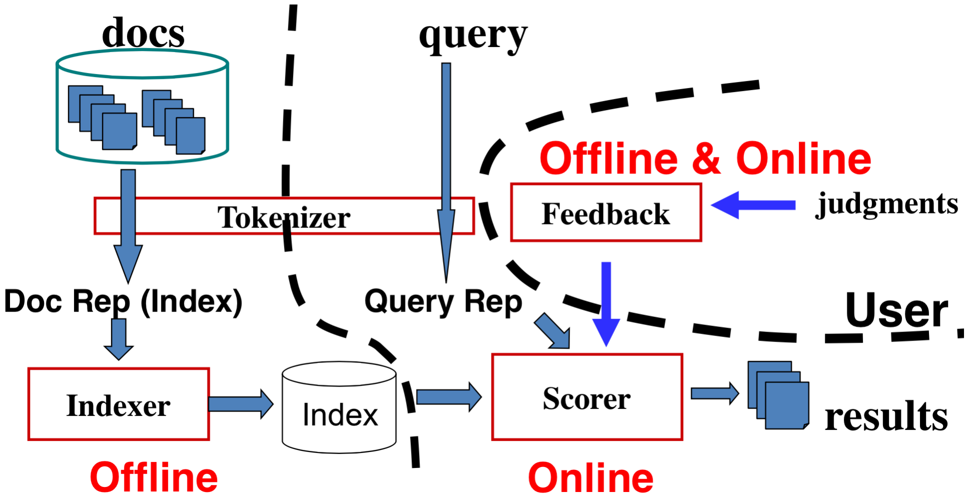


5.17 Recommended reading

- 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.

6 Implementation of TR Systems

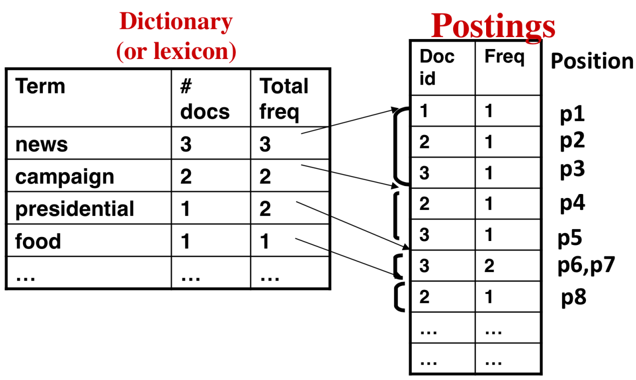
6.1 Typical TR System Architecture



6.2 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
- Some languages (e.g., Chinese) pose challenges in word segmentation

6.3 Inverted Index

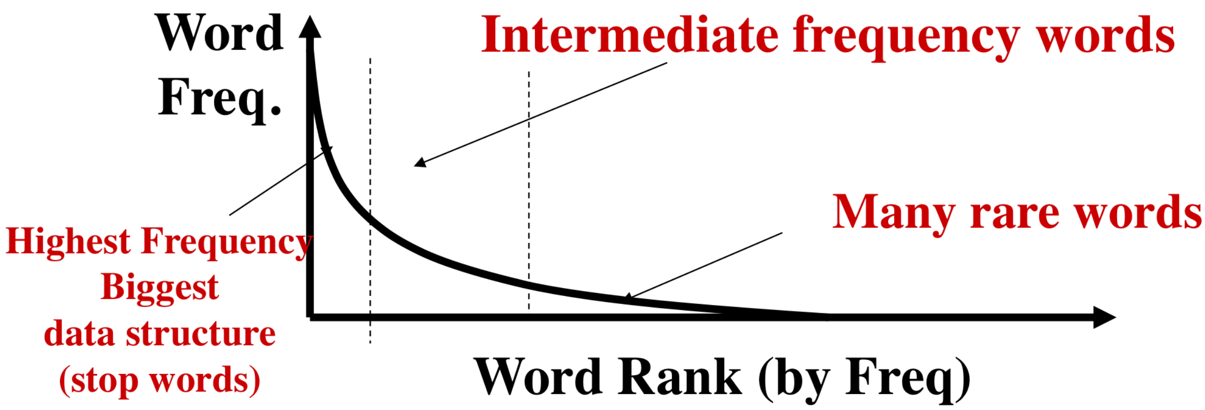


6.4 Empirical Distribution of Words

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

6.5 Zipf's Law



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

rank $\times$ frequency $\approx$ constant:

- $F(w)$ - word frequency
- $r(w)$ - word rank

6.6 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

6.7 Constructing Inverted Index

Sort-based method:

- Step 1: Collect local (termID, docID, freq) tuples from documents
- Step 2: Sort local tuples by termID (to make «runs») and save to files
- Step 3: Pair-wise merge runs
- Step 4: Output inverted file

6.8 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 (Zipf's law)
  - Fewer bits for small (high frequency) integers at the cost of more bits for large integers
- Doc ID compression:
  - «d-gap» (store difference): $d_1, d_2 - d_1, d_3 - d_2, \dots$
  - Feasible due to sequential access

6.9 Integer Compression Methods

- **Binary**: equal-length coding
- **Unary**: $x \geqslant 1$ 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

6.10 General Form of Scoring Function

$$f(q, d) = f_a\left(h\left(g(t_1, d, q), \dots, g(t_k, d, q)\right), f_d(d), f_q(q)\right)$$

- $f_d(d), f_q(q)$ - adjustment factors of document and query
- $g(t_i, d, q)$ - weight of a **matched** query term t_i in d
- $h()$ - weights aggregation function
- $f_a()$ - final score adjustment function

6.11 A General Algorithm for Ranking Documents

- $f_d(d)$ - can be precomputed at index time, $f_q(q)$ - at query time
- Maintain a score accumulator for each d to compute h
- For each query term t_i
  - Fetch the inverted list $\{(d_1, f_1), \dots, (d_n, f_n)\}$
  - 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

6.12 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)

6.13 Some Text Retrieval Toolkits

- Lucene
- Lemur/Indri
- Terrier
- MeTA
- More can be found here

6.14 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
- Further scaling up using distributed file system, parallel processing, and caching

6.15 Recommended reading

- 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.

7 Evaluation of Text Retrieval Systems

7.1 The Cranfield Evaluation Methodology

A methodology for laboratory testing of system components developed in 1960s. General idea is to build reusable test collections and define measures. A test collection can then be reused many times to compare different systems.

- A sample collection of documents (simulate real document collection)
- A sample set of queries/topics (simulate user queries)
- Relevance judgments (ideally made by users who formulated the queries) => Ideal ranked list
- Measures to quantify how well a system's result matches the ideal ranked list

7.2 Evaluating a Set of Retrieved Docs

	Retrieved	**Not Retrieved**
Relevant	a	b
Not Relevant	c	d

- Precision: are the retrieved results all relevant?

$$Precision = \frac{a}{a+c}$$

- Recall: have all the relevant documents been retrieved?

$$Recall = \frac{a}{a+b}$$

- In reality, high recall tends to be associated with low precision

7.3 Combine Precision and Recall: F-Measure

$$F_\beta = \frac{1}{\frac{\beta^2}{\beta^2+1}\frac{1}{R} + \frac{1}{\beta^2+1}\frac{1}{P}} = \frac{(\beta^2+1)\cdot P \cdot R}{\beta^2 \cdot P + R}$$

- P - precision
- R - recall
- β - parameter, often set to 1: $F_1 = \frac{2 \cdot P \cdot R}{P + R}$