# Search Engines

## Information Retrieval in Practice

# Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of the ranking algorithms
- Progress in retrieval models has corresponded with improvements in effectiveness
- Theories about relevance

# Relevance

- Complex concept that has been studied for some time
  - Many factors to consider
  - People often disagree when making relevance judgments
- Retrieval models make various assumptions about relevance to simplify problem
  - e.g., *topical* vs. *user* relevance
  - e.g., *binary* vs. *multi-valued* relevance

# Topical vs. User Relevance

- Topical Relevance
  - Document and query are on the same topic
  - Query: “U.S. Presidents”
  - Document:  Wikipedia article on Abraham Lincoln
- User Relevance
  - Incorporate factors beside document topic
    - Document freshness
    - Style
    - Novelty
    - Content presentation

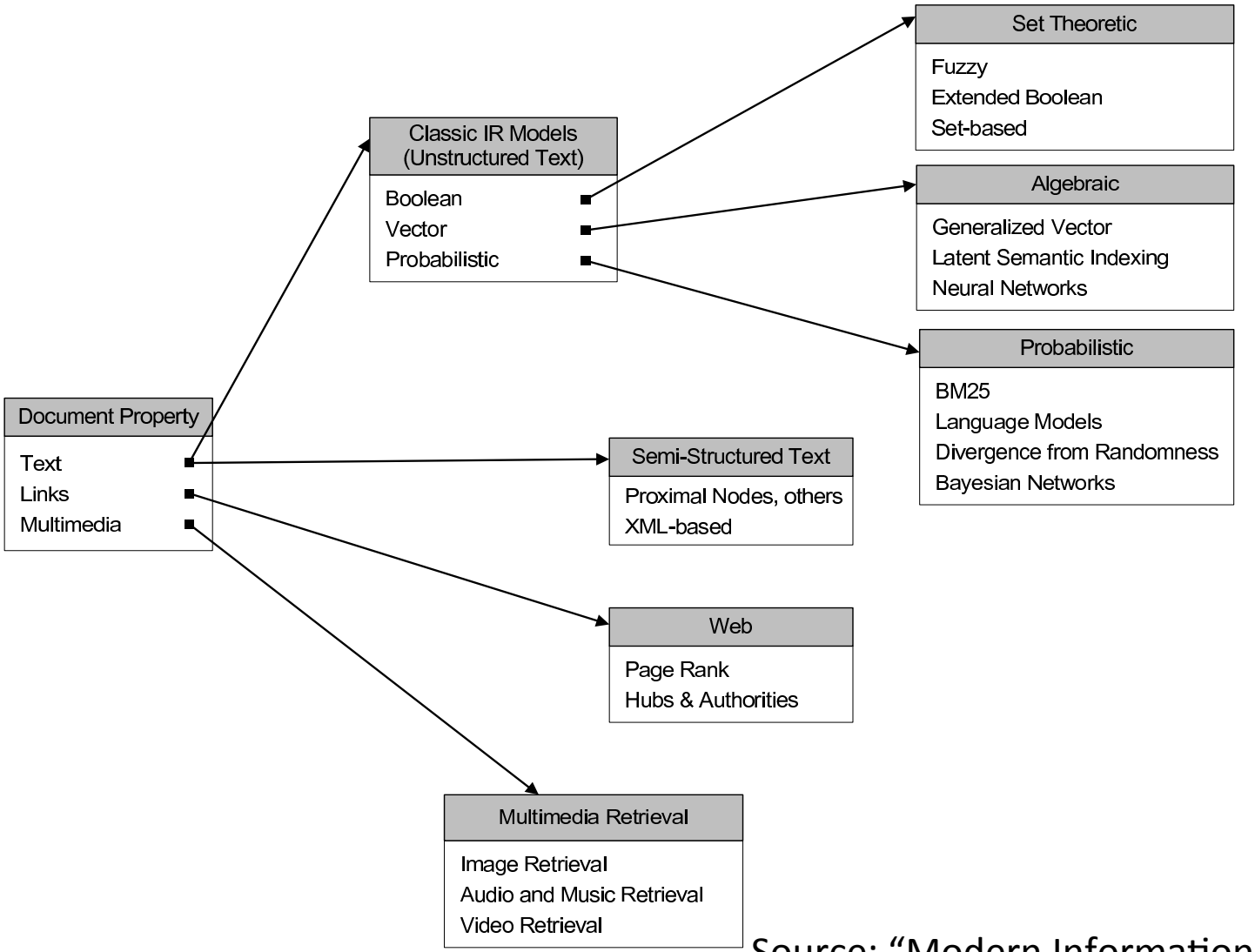
# Binary vs. Multi-Valued Relevance

- Binary Relevance
  - The document is either relevant or not
- Multi-Valued Relevance
  - Makes the evaluation task easier for the judges
  - Not as important for retrieval models
  - Many retrieval models calculate the *probability of relevance*

# Retrieval Model Overview

- Older models
  - Boolean retrieval
  - Vector Space model
- Probabilistic Models
  - BM25
  - Language models
- Combining evidence
  - Inference networks
  - Learning to Rank

# Retrieval Model Overview




Source: "Modern Information Retrieval", Yates et al.

# Boolean Retrieval

- Two possible outcomes for query processing
  - TRUE and FALSE
  - "exact-match" retrieval
  - simplest form of ranking
- Query usually specified using Boolean operators
  - AND, OR, NOT
  - proximity operators and wildcards also used
  - Extra features like date or type

# Boolean Retrieval

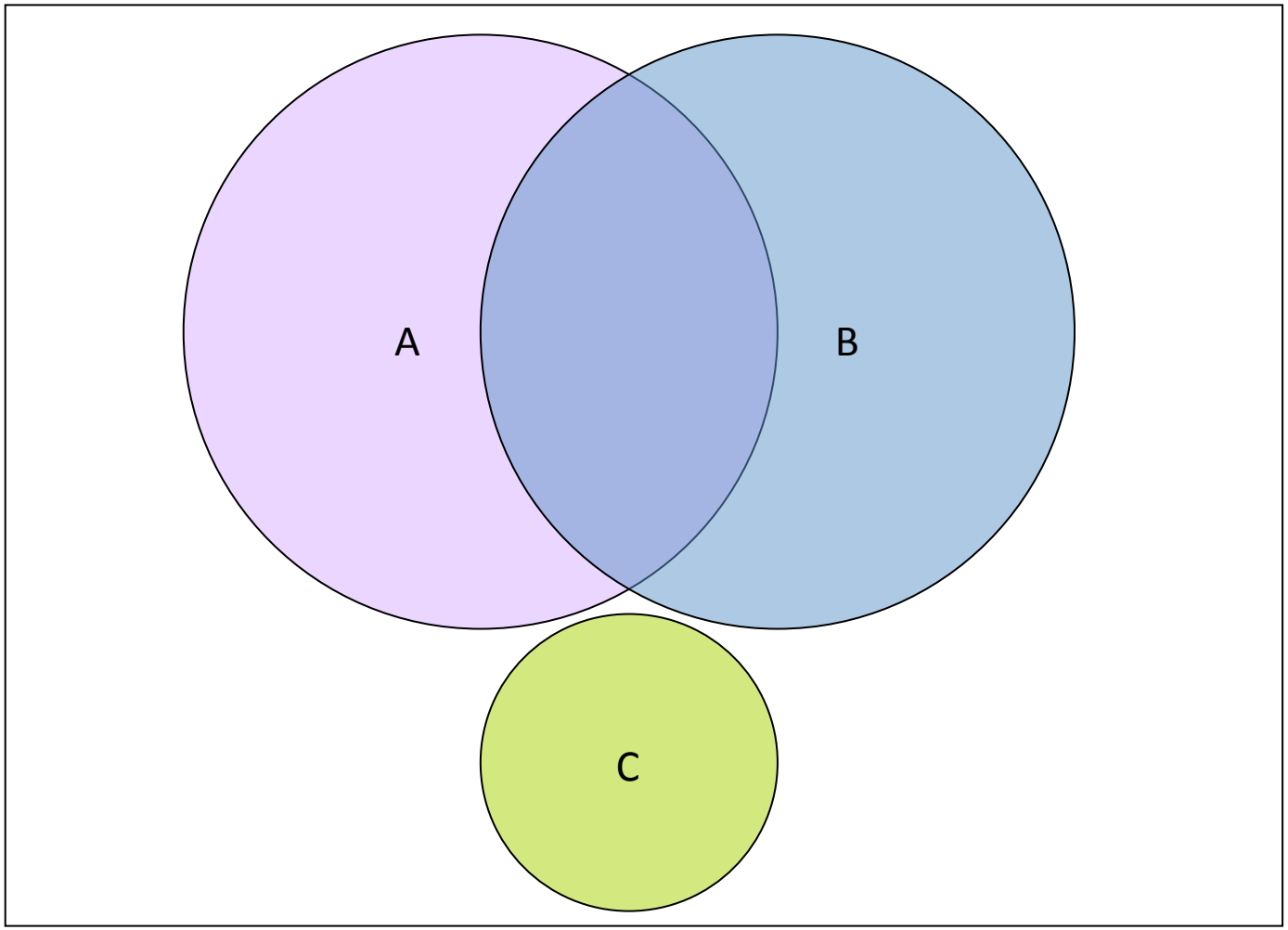
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight
- Very effective in some specific domains
  - e.g., legal search
  - e.g., patent search
  - Expert users

# Example: WestLaw
# www.westlaw.com/

- Largest commercial (paying subscribers) legal search service
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
    - ! = wildcard, /3 = within 3 words, /S = in same sentence

# AND/OR/NOT

All documents



# Boolean View of a Collection

| Term | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 | Doc 6 | Doc 7 | Doc 8 |
|---|---|---|---|---|---|---|---|---|
| aid | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| all | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| back | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| brown | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| come | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| dog | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| fox | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| good | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| jump | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| lazy | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| men | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| now | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| over | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| party | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| quick | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| their | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| time | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

Each column represents the view of a particular document: What terms are contained in this document?

Each row represents the view of a particular term: What documents contain this term?

To execute a query, pick out rows corresponding to query terms and then apply logic table of corresponding Boolean operator

# Sample Queries

| Term | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 | Doc 6 | Doc 7 | Doc 8 | |
|---|---|---|---|---|---|---|---|---|---|
| dog | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | |
| fox | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| dog ∧ fox | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | dog AND fox → Doc 3, Doc 5 |
| dog ∨ fox | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | dog OR fox → Doc 3, Doc 5, Doc 7 |
| dog ¬ fox | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | dog NOT fox → empty |
| fox ¬ dog | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | fox NOT dog → Doc 7 |

| Term | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 | Doc 6 | Doc 7 | Doc 8 | |
|---|---|---|---|---|---|---|---|---|---|
| good | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| party | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | |
| g ∧ p | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | good AND party → Doc 6, Doc 8 |
| over | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | |
| g ∧ p ¬ o | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | good AND party NOT over → Doc 6 |

# Searching by Numbers

- Sequence of queries driven by number of retrieved documents
  1. lincoln
  2. president AND lincoln
  3. president AND lincoln AND NOT (automobile OR car)
  4. president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
  5. president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

# Boolean Retrieval

- Advantages
  - Results are relatively easy to explain
  - Many different features can be incorporated
  - Efficient processing since many documents can be eliminated from search
  - We do not miss any relevant document
- Disadvantages
  - Effectiveness depends entirely on user
    - Simple queries usually don't work well
    - Complex queries are difficult
  - No ranking
  - No control over size of result set: either too many documents or none
  - What about partial matches? Documents that "don't quite match" the query may be useful also

# Vector Space Model

- Provides a framework
  - Term weighting
    - Which terms are more important?
  - Ranking
    - Which documents are more relevant?
  - Relevance Feedback
    - How to employ feedbacks?

# Vector Space Model

- Documents and query represented by a vector of term weights
- Collection represented by a matrix of term weights

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

$$\begin{array}{lcccc} & Term_1 & Term_2 & \ldots & Term_t \\ Doc_1 & d_{11} & d_{12} & \ldots & d_{1t} \\ Doc_2 & d_{21} & d_{22} & \ldots & d_{2t} \\ \vdots & \vdots & & & \\ Doc_n & d_{n1} & d_{n2} & \ldots & d_{nt} \end{array}$$

# Vector Space Model

$D_1$ Tropical Freshwater Aquarium Fish.
$D_2$ Tropical Fish, Aquarium Care, Tank Setup.
$D_3$ Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
$D_4$ The Tropical Tank Homepage - Tropical Fish and Aquariums.

| **Terms** | **Documents** | | | |
|---|---|---|---|---|
| | **$D_1$** | **$D_2$** | **$D_3$** | **$D_4$** |
| aquarium | 1 | 1 | 1 | 1 |
| bowl | 0 | 0 | 1 | 0 |
| care | 0 | 1 | 0 | 0 |
| fish | 1 | 1 | 2 | 1 |
| freshwater | 1 | 0 | 0 | 0 |
| goldfish | 0 | 0 | 1 | 0 |
| homepage | 0 | 0 | 0 | 1 |
| keep | 0 | 0 | 1 | 0 |
| setup | 0 | 1 | 0 | 0 |
| tank | 0 | 1 | 0 | 1 |
| tropical | 1 | 1 | 1 | 2 |

# *Bag of words* model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model
- The positional index is able to distinguish these two documents
  - We will look at positional information later in this course

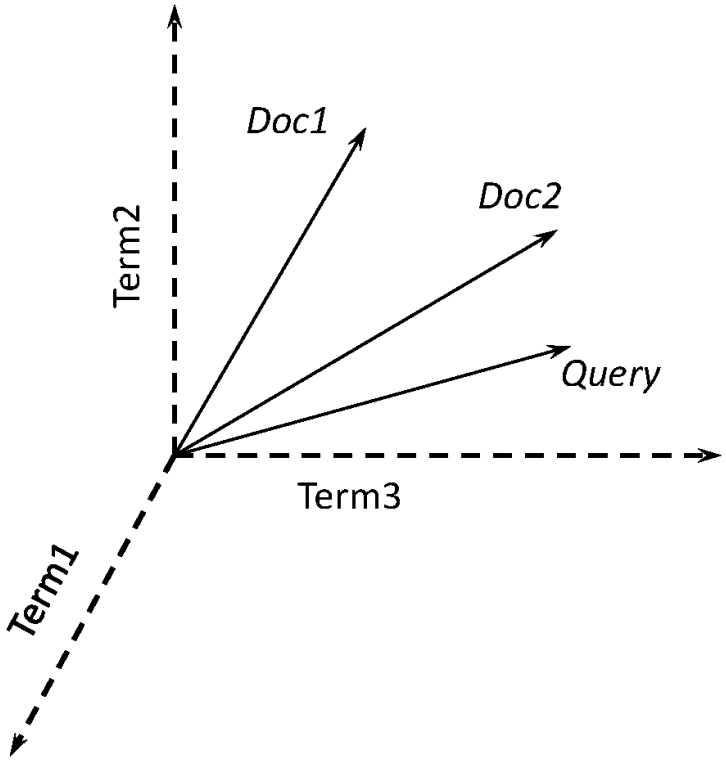
# Vector Space Model

- Query: “tropical fish”

| Term | Query |
|---|---|
| aquarium | 0 |
| bowl | 0 |
| care | 0 |
| **fish** | **1** |
| freshwater | 0 |
| goldfish | 0 |
| homepage | 0 |
| keep | 0 |
| setup | 0 |
| tank | 0 |
| **tropical** | **1** |

# Vector Space Model

- 3-d pictures useful, but can be misleading for high-dimensional space



# How do we weight doc terms?

- Here's the intuition:
  - Terms that appear often in a document should get high weights

    The more often a document contains the term "dog", the more likely that the document is "about" dogs.

  - Terms that appear in many documents should get low weights

    Words like "the", "a", "of" appear in (nearly) all documents.

- How do we capture this mathematically?
  - Term frequency
  - Inverse document frequency

# 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 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

# Log-frequency weighting

- The log frequency weight of term t in d is

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

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score

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

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

# Document frequency

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the 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 high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

# idf weight

- $n_k$ is the document frequency of *k*: the number of documents that contain *k*
  - $n_k$ is an inverse measure of the informativeness of *k*
  - $n_k \leq N$
- We define the idf (inverse document frequency) of *k* by: $\text{idf}_k = \log_{10}(N/n_k)$
  - We use log ($N/n_k$) instead of $N/n_k$ to "dampen" the effect of idf.

# Term Weights

- *tf.idf* weight
  - Term frequency weight measures importance in document
  - Inverse document frequency measures importance in collection
  - Heuristic combination

$$d_{ik} = \frac{(\log(f_{ik})+1)\cdot\log(N/n_k)}{\sqrt{\sum_{k=1}^{t}[(\log(f_{ik})+1.0)\cdot\log(N/n_k)]^2}}$$

# Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
  - iPhone
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

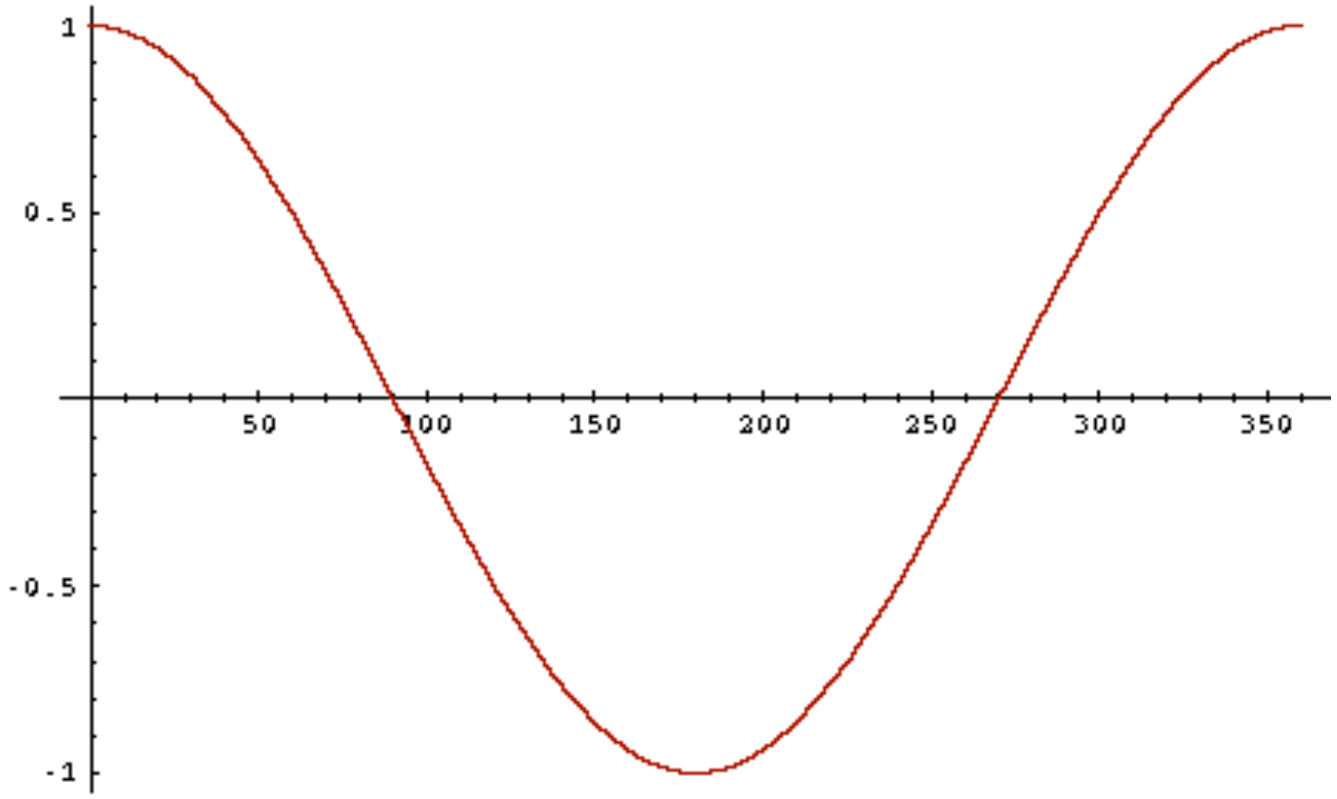
# Queries as vectors

- Do the same for queries: represent them as vectors in the space
- Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance

# From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [$0^o$, $180^o$]

# From angles to cosines




- But how should we be computing cosines?

# cosine(query,document)

Dot product

Unit vectors

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

$q_i$ is the tf-idf weight of term $i$ in the query
$d_i$ is the tf-idf weight of term $i$ in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

# Components of Similarity

- The "inner product" (aka dot product) is the key to the similarity function

$$\vec{d}_i \cdot \vec{q} = \sum_{j=1}^{t} d_{ij} q_j$$

Example:

$$\begin{bmatrix}1 & 2 & 3 & 0 & 2\end{bmatrix} \cdot \begin{bmatrix}2 & 0 & 1 & 0 & 2\end{bmatrix}$$
$$= 1\times 2 + 2\times 0 + 3\times 1 + 0\times 0 + 2\times 2 = 9$$

- The denominator handles document length normalization

$$\left|\vec{d}_i\right| = \sqrt{\sum_{j=1}^{t} d_{ij}^2}$$

Example:

$$\left|\begin{bmatrix}1 & 2 & 3 & 0 & 2\end{bmatrix}\right|$$
$$= \sqrt{1+4+9+0+4} = \sqrt{18} \approx 4.24$$

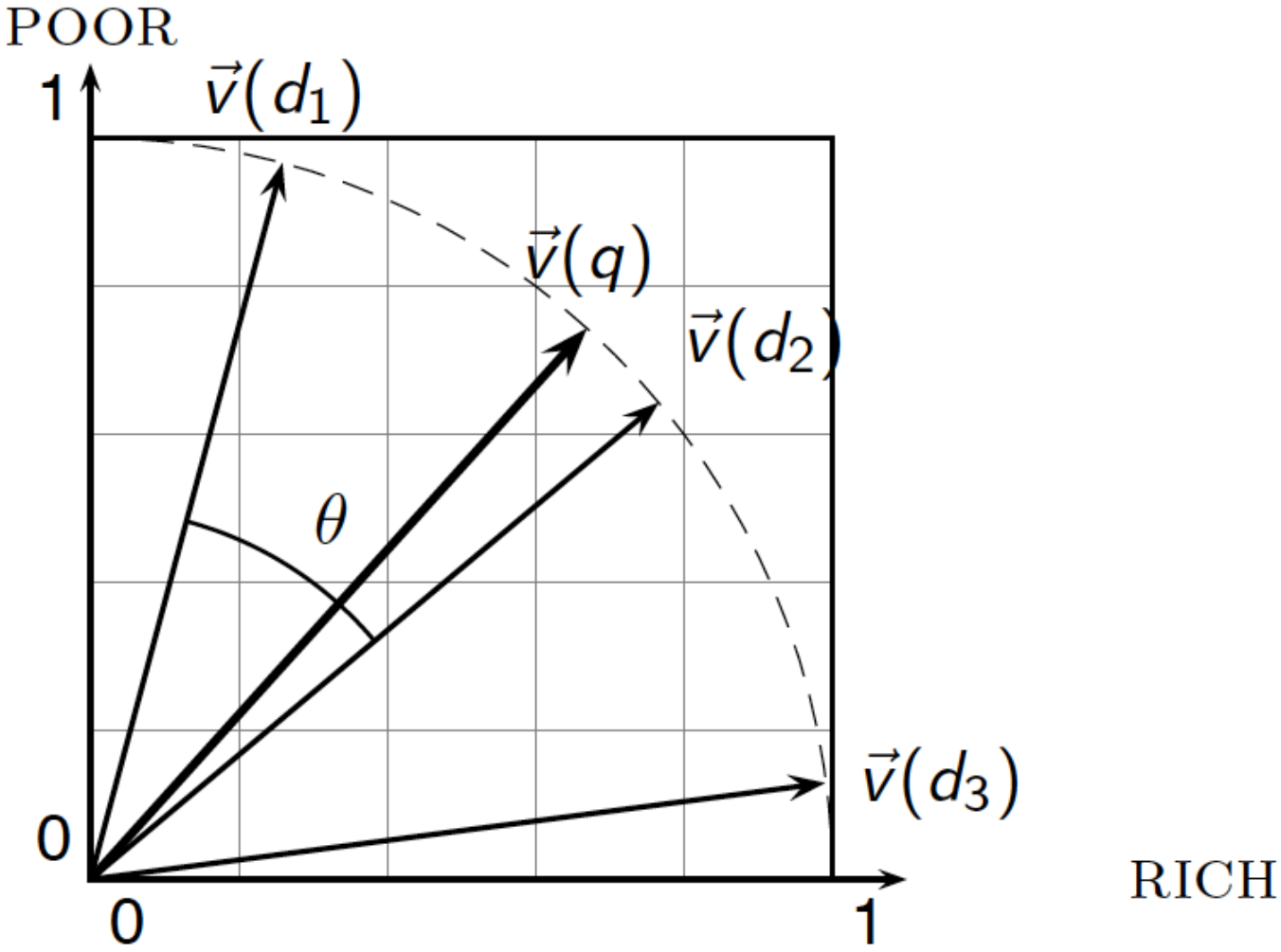
# Similarity Calculation

- Consider two documents $D_1$, $D_2$ and a query $Q$
  - $D_1$ = (0.5, 0.8, 0.3), $D_2$ = (0.9, 0.4, 0.2), $Q$ = (1.5, 1.0, 0)

$$Cosine(D_1, Q) = \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}}$$
$$= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87$$

$$Cosine(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}}$$
$$= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97$$

# Cosine similarity illustrated



# Computing cosine scores

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

# Variations of TF-IDF

| weighting scheme | document term weight | query term weight |
|---|---|---|
| 1 | $f_{i,j} * \log \frac{N}{n_i}$ | $(0.5 + 0.5 \frac{f_{i,q}}{max_i \ f_{i,q}}) * \log \frac{N}{n_i}$ |
| 2 | $1 + \log f_{i,j}$ | $\log(1 + \frac{N}{n_i})$ |
| 3 | $(1 + \log f_{i,j}) * \log \frac{N}{n_i}$ | $(1 + \log f_{i,q}) * \log \frac{N}{n_i}$ |

# Difference from Boolean Retrieval

- Similarity calculation has two factors that distinguish it from Boolean retrieval
  - Number of matching terms affects similarity
  - Weight of matching terms affects similarity
- Documents can be *ranked* by their similarity scores

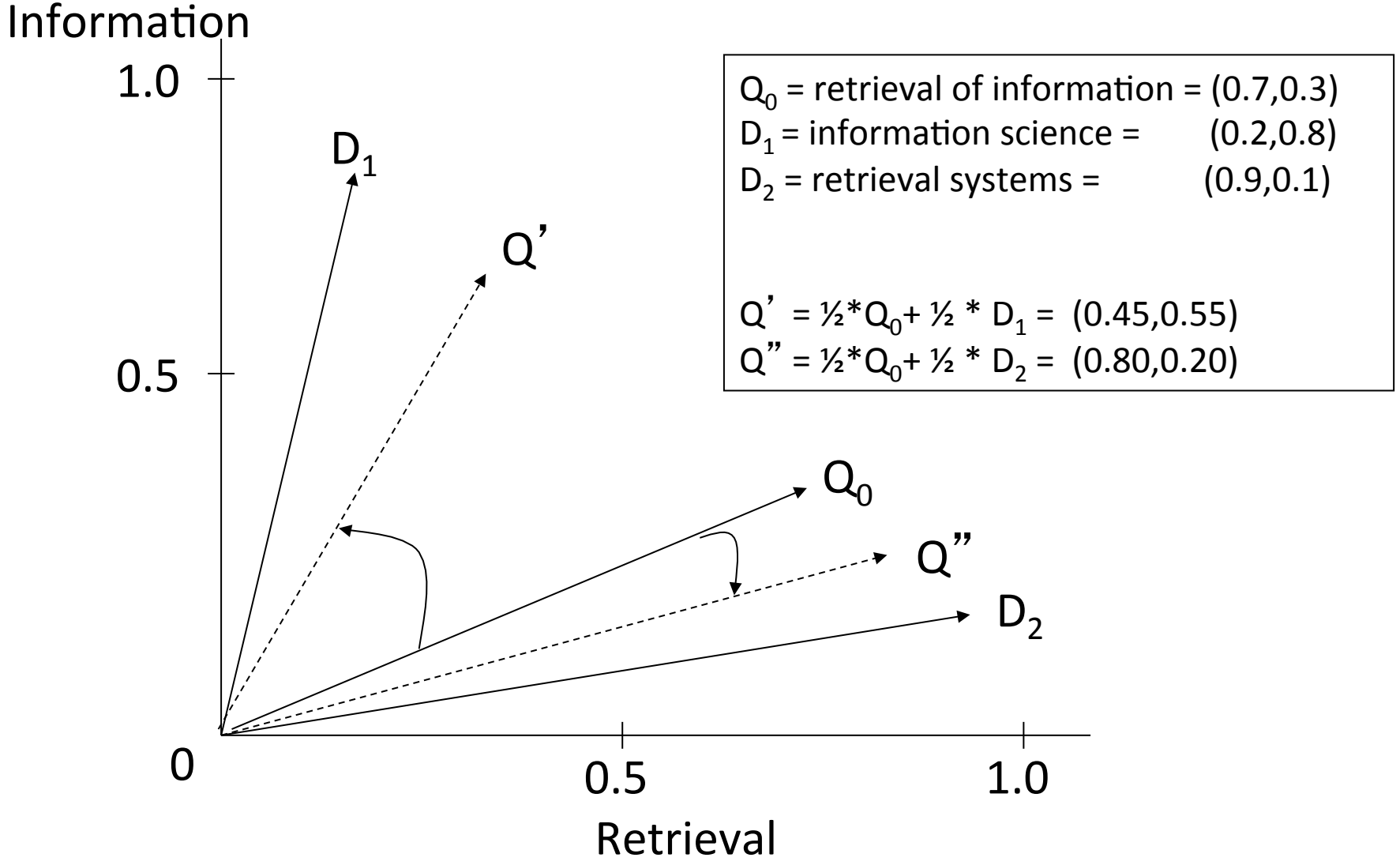
# Relevance Feedback

- Rocchio algorithm
- *Optimal query*
  - Maximizes the difference between the average vector representing the relevant documents and the average vector representing the non-relevant documents
- Modifies query according to

$$q'_j = \alpha.q_j + \beta.\frac{1}{|Rel|}\sum_{D_i \in Rel} d_{ij} - \gamma.\frac{1}{|Nonrel|}\sum_{D_i \in Nonrel} d_{ij}$$

  - $\alpha$, $\beta$, and $\gamma$ are parameters
    - Typical values 8, 16, 4

# Rocchio Illustration




$Q_0$ = retrieval of information = (0.7,0.3)
$D_1$ = information science = (0.2,0.8)
$D_2$ = retrieval systems = (0.9,0.1)

$Q' = ½*Q_0 + ½ * D_1 = (0.45,0.55)$
$Q'' = ½*Q_0 + ½ * D_2 = (0.80,0.20)$

# Example Rocchio Calculation

$R_1 = (.030, 0.00, 0.20, .025, .050, .050, 0.00, 0.00, .120)$

$R_2 = (.020, .009, .000, .002, .025, .025, .100, .100, .120)$

Relevant docs

$S_1 = (.030, .010, .020, 0.00, .005, .025, 0.00, .020, 0.00)$

Non-rel doc

$Q = (0.00, 0.00, 0.00, 0.00, .500, 0.00, .450, 0.00, .950)$

Original Query

$\alpha = 1$

$\beta = 0.75$

$\gamma = 0.25$

Constants

$$Q_{new} = \alpha \times Q + \left(\frac{\beta}{2} \times (R_1 + R_2)\right) - \left(\frac{\gamma}{1} \times S_1\right)$$

Resulting feedback query

$$Q_{new} = (0.011, 0.000875, 0.002, 0.01, 0.527, 0.022, 0.488, 0.033, 1.04)$$

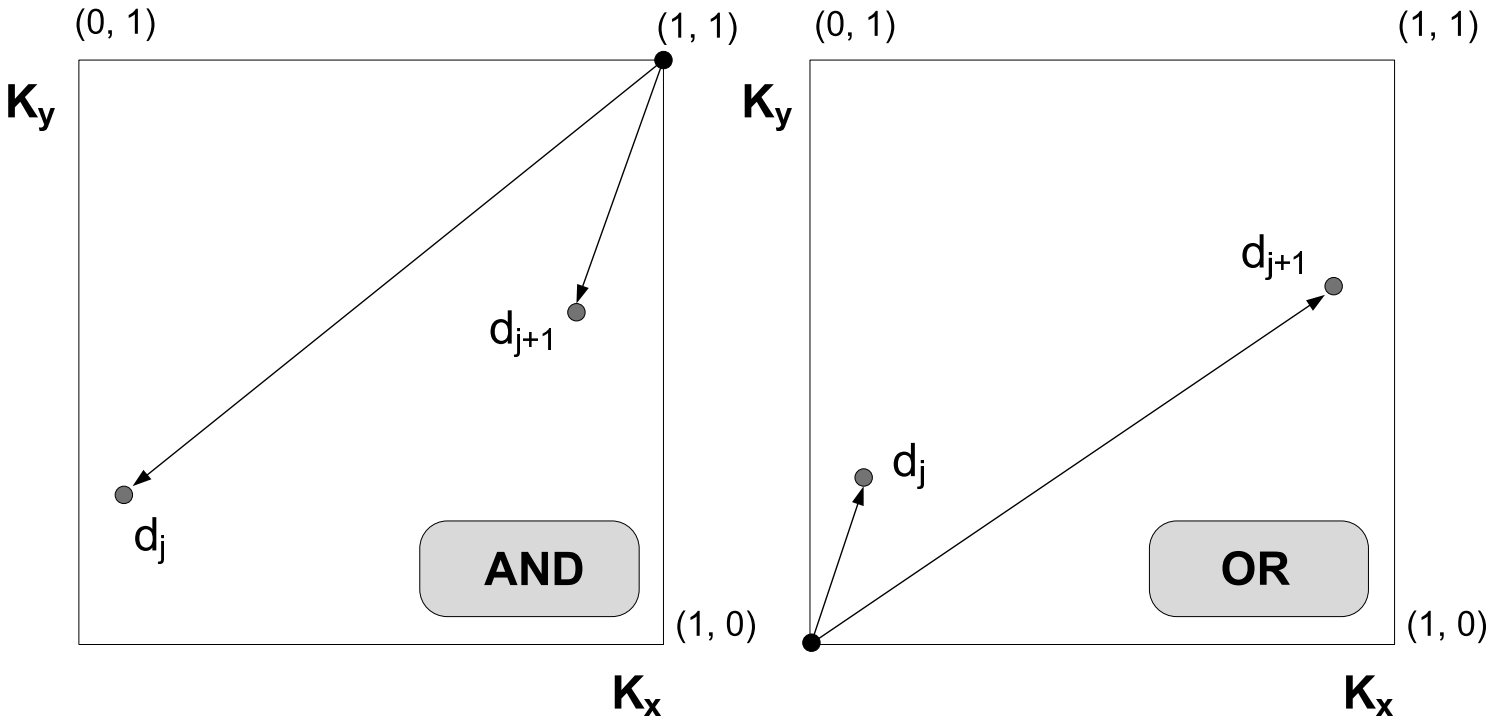
# Vector Space Model

- Advantages
  - Simple computational framework for ranking
  - Any similarity measure or term weighting scheme could be used
- Disadvantages
  - Assumption of term independence
  - No *predictions* about techniques for effective ranking

# Extensions

- Generalized Vector Space Model
  - Capturing dependencies between terms
  - Introducing new dimensions in the space
  - Each new dimension is a combination of the terms
- Extended Boolean Model
  - Partial matching and term weighting
  - Distance from the most interesting or least interesting answers

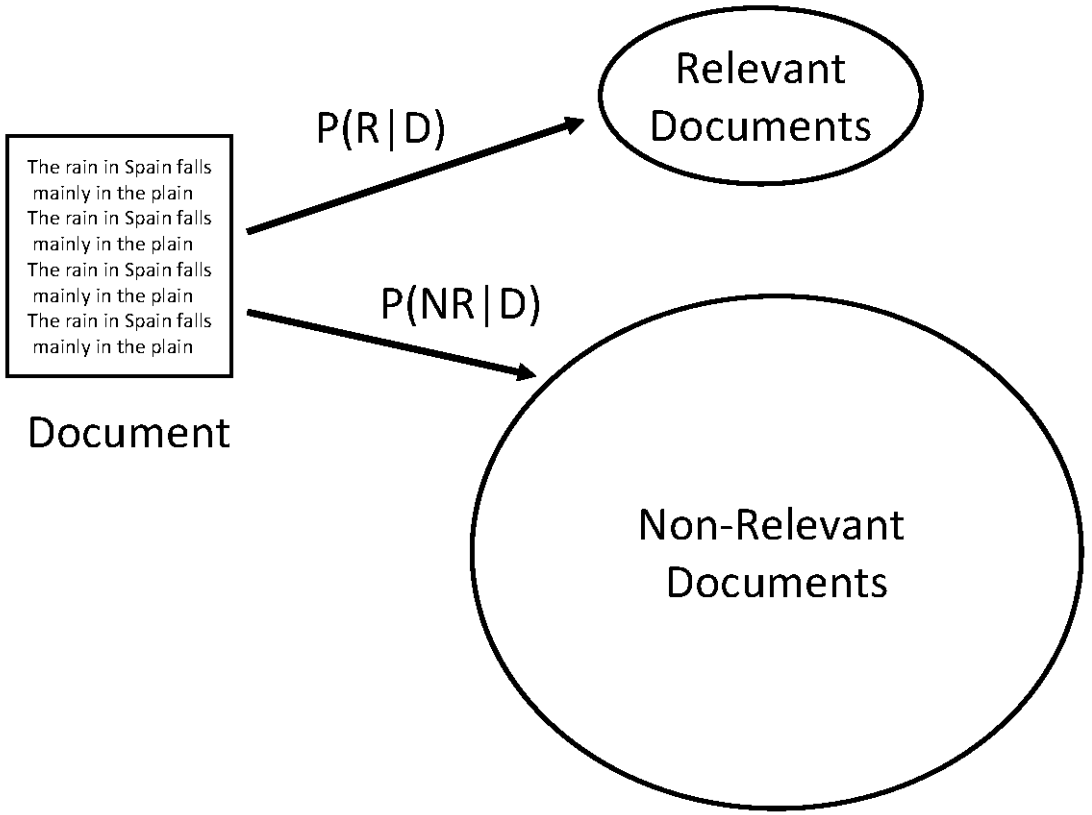
# Extended Boolean Model



# Probability Ranking Principle

- Robertson (1977)
  - "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request,
  - where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose,
  - the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

# IR as Classification



# Bayes Classifier

- Bayes Decision Rule
  - A document *D* is relevant if $P(R|D) > P(NR|D)$
- Estimating probabilities
  - use Bayes Rule

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)}$$

  - classify a document as relevant if

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)}$$

    - This is *likelihood ratio*

# Estimating P(D|R)

- Assume independence

$$P(D|R) = \prod_{i=1}^{t} P(d_i|R)$$

- *Binary independence model*
  - document represented by a vector of binary features indicating term occurrence (or non-occurrence)
  - $p_i$ is probability that term i occurs (i.e., has value 1) in relevant document, $s_i$ is probability of occurrence in non-relevant document

# Binary Independence Model

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left(\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i}\right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_{i} \frac{1-p_i}{1-s_i}$$

# Binary Independence Model

- Scoring function is

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

- Query provides information about relevant documents
- If we assume $p_i$ constant, $s_i$ approximated by entire collection, get *idf*-like weight

$$\log \frac{0.5(1-\frac{n_i}{N})}{\frac{n_i}{N}(1-0.5)} = \log \frac{N-n_i}{n_i}$$

# Contingency Table

| | Relevant | Non-relevant | Total |
|---|---|---|---|
| $d_i = 1$ | $r_i$ | $n_i - r_i$ | $n_i$ |
| $d_i = 0$ | $R - r_i$ | $N - n_i - R + r_i$ | $N - r_i$ |
| Total | $R$ | $N - R$ | $N$ |

$$p_i = (r_i + 0.5)/(R + 1)$$

$$s_i = (n_i - r_i + 0.5)/(N - R + 1)$$

Gives scoring function:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

# BM25

- BM25 was created as the result of a series of experiments on variations of the probabilistic model
- A good term weighting is based on three principles
  - inverse document frequency
  - term frequency
  - document length normalization
- The classic probabilistic model covers only the first of these principles
- This reasoning led to a series of experiments which led to the BM25 ranking formula

# BM25

- Popular and effective ranking algorithm based on binary independence model
  - adds document and query term weights

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1) f_i}{K + f_i} \cdot \frac{(k_2 + 1) q f_i}{k_2 + q f_i}$$

  - $k_1$, $k_2$ and $K$ are parameters whose values are set empirically
  - $K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$ *dl* is doc length
  - Typical TREC value for $k_1$ is 1.2, $k_2$ varies from 0 to 1000, b = 0.75

# BM25 Example

- Query with two terms, “president lincoln”, (*qf = 1)*
- No relevance information (*r and R are* zero)
- $N$ = 500,000 documents
- *“president”* occurs in 40,000 documents ($n_1$ = 40, 000)
- *“lincoln”* occurs in 300 documents ($n_2$ = 300)
- “president” occurs 15 times in doc ($f_1$ = 15)
- *“lincoln”* occurs 25 times ($f_2$ = 25)
- document length is 90% of the average length (*dl/avdl* = .9)
- $k_1$ = 1.2, $b$ = 0.75, and $k_2$ = 100
- $K = 1.2 \cdot (0.25 + 0.75 \cdot 0.9) = 1.11$

# BM25 Example

$$
\begin{aligned}
BM25(Q,D) \quad = \quad & \log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\
& \times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\
& + \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\
& \times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1} \\
\\
= \quad & \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\
& + \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\
= \quad & 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\
= \quad & 5.00 + 15.66 = 20.66
\end{aligned}
$$

# BM25 Example

- Effect of term frequencies

| Frequency of "president" | Frequency of "lincoln" | BM25 score |
|---|---|---|
| 15 | 25 | 20.66 |
| 15 | 1 | 12.74 |
| 15 | 0 | 5.00 |
| 1 | 25 | 18.2 |
| 0 | 25 | 15.66 |

# Language models

- Based on the notion of probabilities and processes for generating text
- Documents are ranked based on the probability that they generated the query
- Best/partial match

# Uses of Language Models

- Speech recognition
  - "I ate a cherry" is a more likely sentence than "Eye eight uh Jerry"
- OCR & Handwriting recognition
  - More probable sentences are more likely correct readings.
- Machine translation
  - More likely sentences are probably better translations.
- Context sensitive spelling correction
  - "Their are problems wit this sentence."

# Completion Prediction

- A language model also supports predicting the completion of a sentence.
  - Please turn off your cell _____
  - Your program does not _____
- *Predictive text input* systems can guess what you are typing and give choices on how to complete it.

# What is a Language Model?

- Probability distribution over strings of text
  - How likely is a string in a given "language"?

$p_1$ = P("the dog barks")

$p_2$ = P("barks the dog")

$p_3$ = P("быстрая brown dog")

$p_4$ = P("быстрая собака")

In a language model for English: $p_1 > p_2 > p_3 > p_4$

- Probabilities depend on what language we're modeling

In a language model for Russian: $p_1 < p_2 < p_3 < p_4$

# Language Model

- *Unigram language model*
  - probability distribution over the words in a language
  - generation of text consists of pulling words out of a "bucket" according to the probability distribution and replacing them
- N-gram language model
  - some applications use bigram and trigram language models where probabilities depend on previous words

# Statistical Foundation

- Let *S* be a sequence of *r* consecutive terms that occur in a document of the collection:

$$S = k_1, k_2, \ldots, k_r$$

- An *n*-gram language model uses a Markov process to assign a probability of occurrence to *S*:

$$P_n(S) = \prod_{i=1}^{r} P(k_i | k_{i-1}, k_{i-2}, \ldots, k_{i-(n-1)})$$

where n is the order of the Markov process. The occurrence of a term depends on observing *n − 1* terms that precede it in the text

# N-gram Language Model

- *Unigram language model* (*n = 1*): the estimations are based on the occurrence of individual words
- *Bigram language model* (*n = 2*): the estimations are based on the co-occurrence of pairs of words
- Higher order models such as *Trigram language models* (*n = 3*) are usually adopted for speech recognition
- *Term independence assumption*: in the case of IR, the impact of word order is less clear
  - As a result, Unigram models have been used extensively in IR

# Unigram Language Model

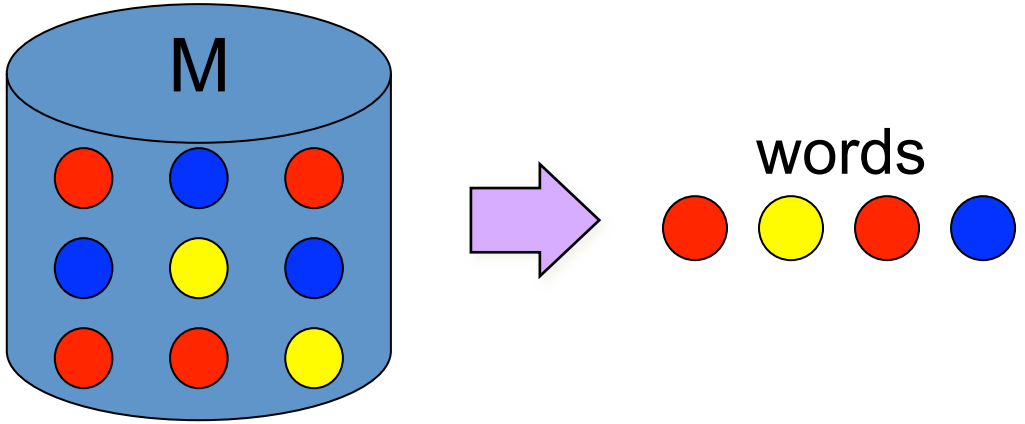
- Assume each word is generated independently
  - Obviously, this is not true…
  - But it seems to work well in practice!
- The probability of a string, given a model:

$$P(q_1 \ldots q_k \mid M) = \prod_{i=1}^{k} P(q_i \mid M)$$

The probability of a sequence of words decomposes into a product of the probabilities of individual words

# A Physical Metaphor

- Colored balls are randomly drawn from an urn (with replacement)




$$P(\bullet\ \bullet\ \bullet\ \bullet) = P(\bullet) \times P(\bullet) \times P(\bullet) \times P(\bullet)$$

$$= (4/9) \times (2/9) \times (4/9) \times (3/9)$$

# An Example

| Model M | |
|---|---|
| P(w) | w |
| 0.2 | the |
| 0.1 | a |
| 0.01 | man |
| 0.01 | woman |
| 0.03 | said |
| 0.02 | likes |
| … | |

| the | man | likes | the | woman |
|---|---|---|---|---|
| 0.2 | 0.01 | 0.02 | 0.2 | 0.01 |

multiply

P(s | M) = 0.00000008

P(“the man likes the woman”|M)
= P(the|M) × P(man|M) × P(likes|M) × P(the|M) × P(man|M)
= 0.00000008

# Comparing Language Models

| Model $M_1$ | |
|---|---|
| P(w) | w |
| 0.2 | the |
| 0.0001 | yon |
| 0.01 | class |
| 0.0005 | maiden |
| 0.0003 | sayst |
| 0.0001 | pleaseth |
| … | |

| Model $M_2$ | |
|---|---|
| P(w) | w |
| 0.2 | the |
| 0.1 | yon |
| 0.001 | class |
| 0.01 | maiden |
| 0.03 | sayst |
| 0.02 | pleaseth |
| … | |

| the | class | pleaseth | yon | maiden |
|---|---|---|---|---|
| 0.2 | 0.01 | 0.0001 | 0.0001 | 0.0005 |
| 0.2 | 0.001 | 0.02 | 0.1 | 0.01 |

$$P(s|M_2) > P(s|M_1)$$

What exactly does this mean?

# LMs for Retrieval

- 3 possibilities:
  - probability of generating the query text from a document language model
  - probability of generating the document text from a query language model
  - comparing the language models representing the query and document topics
- Models of topical relevance

# Language Model in IR

- A **language model** for IR is composed of the following components
  - A set of document language models, one per document $d_j$ of the collection
  - A probability distribution function that allows estimating the likelihood that a document language model $M_j$ *generates* each of the query terms
  - A ranking function that combines these generating probabilities for the query terms into a rank of document $d_j$ with regard to the query

# Language Model

- A *topic* in a document or query can be represented as a language model
  - i.e., words that tend to occur often when discussing a topic will have high probabilities in the corresponding language model
- *Multinomial* distribution over words
  - text is modeled as a finite sequence of words, where there are *t* possible words at each point in the sequence
  - commonly used, but not only possibility
  - doesn't model *burstiness*

# Query-Likelihood Model

- Rank documents by the probability that the query could be generated by the document model (i.e. same topic)
- Given a query, start with P(D|Q)
- Using Bayes' Rule

$$p(D|Q) \overset{rank}{=} P(Q|D)P(D)$$

- Assuming prior is uniform, unigram model

$$P(Q|D) = \prod_{i=1}^{n} P(q_i|D)$$

# Estimating Probabilities

- Obvious estimate for unigram probabilities is

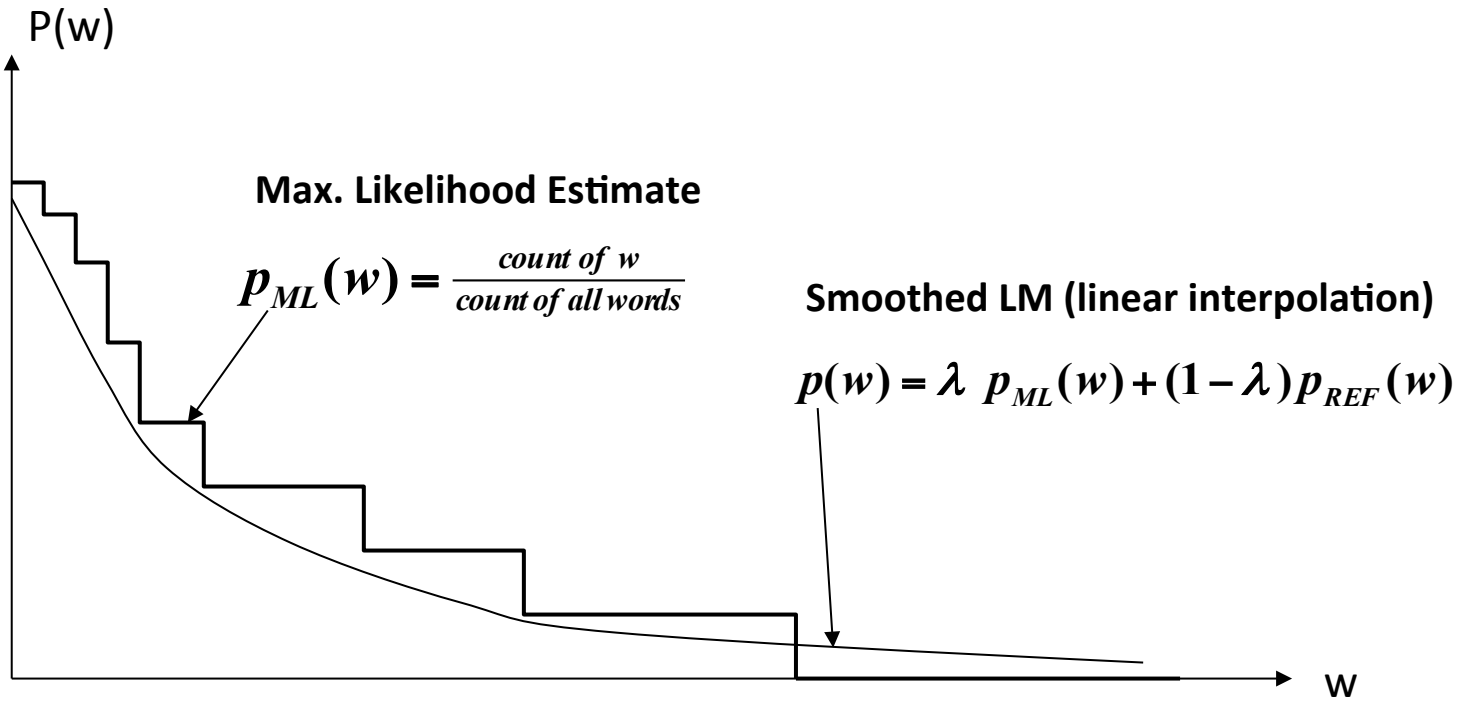
$$P(q_i|D) = \frac{f_{q_i,D}}{|D|}$$

- *Maximum likelihood estimate*
  - makes the observed value of $f_{q;D}$ most likely
- If query words are missing from document, score will be zero
  - Missing 1 out of 4 query words same as missing 3 out of 4

# Smoothing

- Document texts are a *sample* from the language model
  - Missing words should not have zero probability of occurring
  - A document is a very small sample of words, and the maximum likelihood estimate will be inaccurate.
- *Smoothing* is a technique for estimating probabilities for missing (or unseen) words
  - lower (or *discount*) the probability estimates for words that are seen in the document text
  - assign that "left-over" probability to the estimates for the words that are not seen in the text

# Language Model Smoothing



# Estimating Probabilities

- Estimate for unseen words is $\alpha_D P(q_i|C)$
  - $P(q_i|C)$ is the probability for query word *i* in the *collection* language model for collection *C* (background probability)
  - $\alpha_D$ is a parameter
- Estimate for words that occur is

$$(1 - \alpha_D)\, P(q_i|D) + \alpha_D\, P(q_i|C)$$

- Different forms of estimation come from different $\alpha_D$

# Mixture model

$$(1 - \alpha_D)\, P(q_i|D) + \alpha_D\, P(q_i|C)$$

- Mixes the probability from the document with the general collection frequency of the word.
- Low value of $\alpha_D$: "conjunctive-like" search – tends to retrieve documents containing all query words.
- High value of $\alpha_D$: more disjunctive, suitable for long queries
- Correctly setting $\alpha_D$ is very important for good performance.

# Jelinek-Mercer Smoothing

- $\alpha_D$ is a constant, λ
- Gives estimate of

$$p(q_i|D) = (1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

- Ranking score

$$P(Q|D) = \prod_{i=1}^{n}((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

- Use logs for convenience
  - accuracy problems multiplying small numbers

$$\log P(Q|D) = \sum_{i=1}^{n}\log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

# Where is *tf.idf* Weight?

$$\log P(Q|D) = \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

$$= \sum_{i:f_{q_i,D}>0} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) + \sum_{i:f_{q_i,D}=0} \log(\lambda\frac{c_{q_i}}{|C|})$$

$$= \sum_{i:f_{q_i,D}>0} \log \frac{((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})}{\lambda\frac{c_{q_i}}{|C|}} + \sum_{i=1}^{n} \log(\lambda\frac{c_{q_i}}{|C|})$$

$$\overset{rank}{=} \sum_{i:f_{q_i,D}>0} \log\left(\frac{((1-\lambda)\frac{f_{q_i,D}}{|D|}}{\lambda\frac{c_{q_i}}{|C|}} + 1\right)$$

- proportional to the term frequency, inversely proportional to the collection frequency

# Dirichlet Smoothing

- $\alpha_D$ depends on document length

$$\alpha_D = \frac{\mu}{|D|+\mu}$$

- Gives probability estimation of

$$p(q_i|D) = \frac{f_{q_i,D}+\mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

- and document score

$$\log P(Q|D) = \sum_{i=1}^{n} \log \frac{f_{q_i,D}+\mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

- Jelinek-Mercer with document-dependent parameter
- Pseudo-count as a small version of collection

# Absolute discounting

- Subtract a constant $\delta$ from all frequencies in the document
- Divide the subtracted mass between all the terms

$$p(w|d) = \frac{\max(c(w;d)-\delta,0)+\delta|d|_u\, p(w|C)}{|d|}$$

# Query Likelihood Example

- For the term “president”
  - $f_{qi,D}$ = 15, $c_{qi}$ = 160,000
- For the term “lincoln”
  - $f_{qi,D}$ = 25, $c_{qi}$ = 2,400
- number of word occurrences in the document |d| is assumed to be 1,800
- number of word occurrences in the collection is $10^9$
  - 500,000 documents times an average of 2,000 words
- Score using JM smoothing with λ=0.1 ?
- Score using Dirichlet smoothing with μ = 2,000?

# Query Likelihood Example

$$
\begin{aligned}
QL(Q,D) &= \log \frac{15 + 2000 \times (1.6 \times 10^5 / 10^9)}{1800 + 2000} \\
&\quad + \log \frac{25 + 2000 \times (2400/10^9)}{1800 + 2000} \\
&= \log(15.32/3800) + \log(25.005/3800) \\
&= -5.51 + -5.02 = -10.53
\end{aligned}
$$

- Negative number because summing logs of small numbers

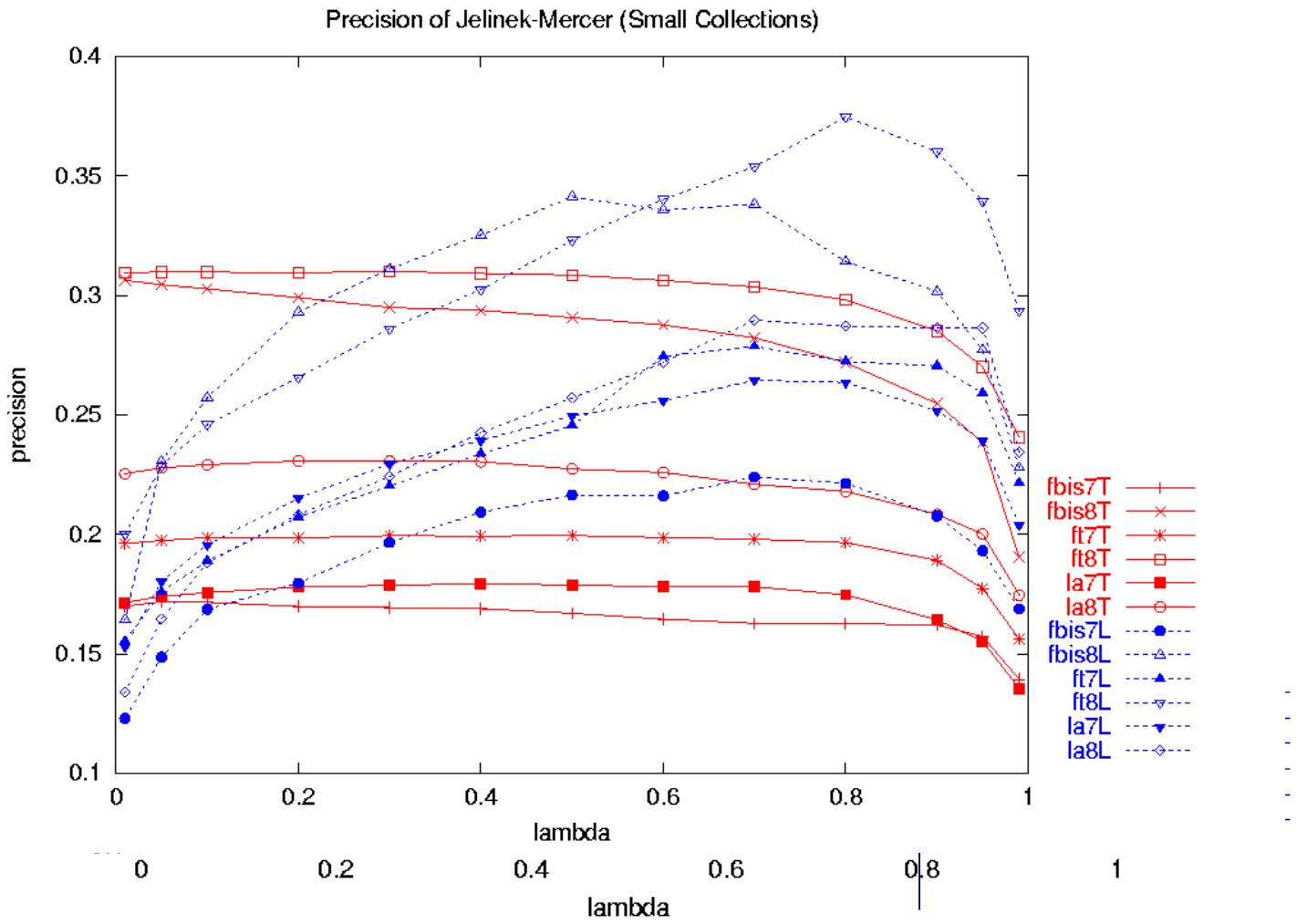
# Query Likelihood Example

| Frequency of "president" | Frequency of "lincoln" | QL score |
|---|---|---|
| 15 | 25 | -10.53 |
| 15 | 1 | -13.75 |
| 15 | 0 | -19.05 |
| 1 | 25 | -12.99 |
| 0 | 25 | -14.40 |

# Title queries vs. Long queries

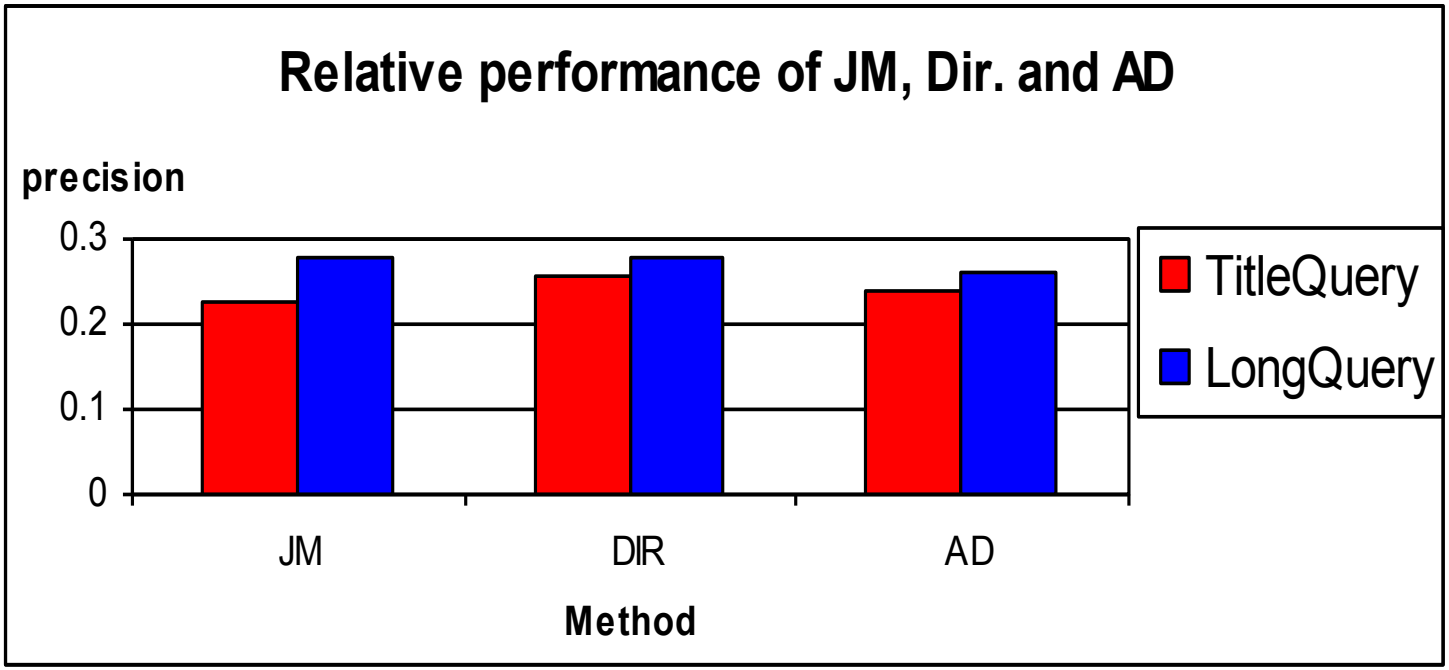
(Jelinek-Mercer on FBIS, FT, and LA)

**Optim**



# Comparison of Three Methods

| Query Type | JM | Dir | AD |
|---|---|---|---|
| Title | 0.228 | **0.256** | 0.237 |
| Long | **0.278** | 0.276 | 0.260 |



# Vector space (tf-idf) vs. LM

| | precision | | | significant? |
|---|---|---|---|---|
| Rec. | tf-idf | LM | %chg | |
| 0.0 | 0.7439 | 0.7590 | +2.0 | |
| 0.1 | 0.4521 | 0.4910 | +8.6 | |
| 0.2 | 0.3514 | 0.4045 | +15.1 | * |
| 0.4 | 0.2093 | 0.2572 | +22.9 | * |
| 0.6 | 0.1024 | 0.1405 | +37.1 | * |
| 0.8 | 0.0160 | 0.0432 | +169.6 | * |
| 1.0 | 0.0028 | 0.0050 | +76.9 | |
| 11-point average | 0.1868 | 0.2233 | +19.6 | * |

The language modeling approach always does better in these experiments . . .

# Relevance Models

- *Relevance model* – language model representing information need
  - query and relevant documents are samples from this model
- *P(D|R)* - probability of generating the text in a document given a relevance model
  - *document likelihood* model
  - less effective than query likelihood due to difficulties comparing across documents of different lengths
  - Alternative: Comparing language models

# Pseudo-Relevance Feedback

- Estimate relevance model from query and top-ranked documents
- Rank documents by similarity of document model to relevance model
- *Kullback-Leibler divergence* (KL-divergence) is a well-known measure of the difference between two probability distributions

# KL-Divergence

- Given the *true* probability distribution *P* and another distribution *Q* that is an *approximation* to *P*,

$$KL(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

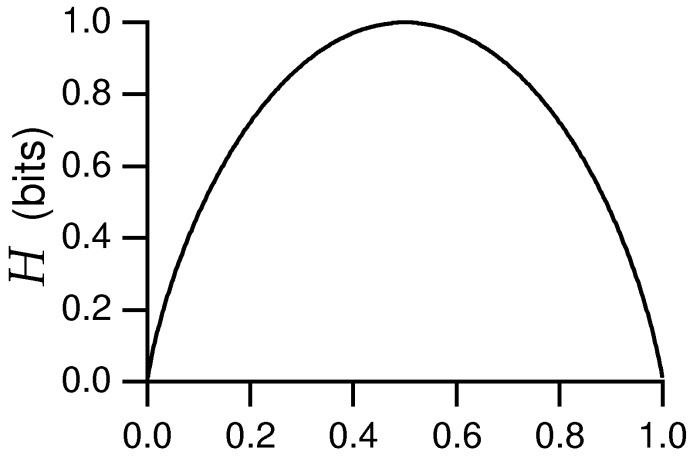
- KL-Divergence is the relative entropy between two distributions

# Entropy

- The entropy is the average uncertainty of a single random variable.
- Let $p(x)=P(X=x)$ where $x \in X$
- $H(p)= H(X)= - \Sigma_{x \in X} p(x) log_2 p(x)$

- In other words, entropy measures the amount of information in a random variable. It is normally measured in bits.

# Entropy




- If X has two values
- Inversely related to predictability

# Relative Entropy or Kullback-Leibler Divergence

$$KL(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

- The relative entropy (also known as the *Kullback-Leibler divergence*) is a measure of how different two probability distributions are.
- The KL divergence between p and q can also be seen as the average number of bits that are wasted by encoding events from a distribution p with a code based on a not-quite-right distribution q.

- *KL(P||Q)>=0*

- *KL(P,Q) ≠ KL(Q,P)*

# KL-Divergence for Ranking

- Use negative KL-divergence for ranking, and assume relevance model *R* is the true distribution (not symmetric),

$$\sum_{w \in V} P(w|R) \log P(w|D) - \sum_{w \in V} P(w|R) \log P(w|R)$$

# KL-Divergence

- Given a simple maximum likelihood estimate for *P(w|R),* based on the frequency in the query text, ranking score is

$$\sum_{w \in V} \frac{f_{w,Q}}{|Q|} \log P(w|D)$$

  – rank-equivalent to query likelihood score

- Query likelihood model is a special case of retrieval based on relevance model

# Estimating the Relevance Model

- Probability of pulling a word *w* out of the "bucket" representing the relevance model depends on the *n* query words we have just pulled out

$$P(w|R) \approx P(w|q_1 \ldots q_n)$$

- By definition

$$P(w|R) \approx \frac{P(w,q_1 \ldots q_n)}{P(q_1 \ldots q_n)}$$

# Estimating the Relevance Model

- Joint probability is

$$P(w, q_1 \dots q_n) = \sum_{D \in \mathcal{C}} p(D) P(w, q_1 \dots q_n | D)$$

- Assume

$$P(w, q_1 \dots q_n | D) = P(w|D) \prod_{i=1}^{n} P(q_i|D)$$

- Gives

$$P(w, q_1 \dots q_n) = \sum_{D \in \mathcal{C}} P(D) P(w|D) \prod_{i=1}^{n} P(q_i|D)$$

# Estimating the Relevance Model

- *P(D)* usually assumed to be uniform
- *P(w, q1 . . . qn)* is simply a weighted average of the language model probabilities for *w* in a set of documents, where the weights are the query likelihood scores for those documents
- Formal model for pseudo-relevance feedback
  - query expansion technique

# Pseudo-Feedback Algorithm

1. Rank documents using the query likelihood score for query $Q$.
2. Select some number of the top-ranked documents to be the set $\mathcal{C}$.
3. Calculate the relevance model probabilities $P(w|R)$. $P(q_1 \dots q_n)$ is used as a normalizing constant and is calculated as

$$P(q_1 \dots q_n) = \sum_{w \in V} P(w, q_1 \dots q_n)$$

4. Rank documents again using the KL-divergence score

$$\sum_{w} P(w|R) \log P(w|D)$$

# Example from Top 10 Docs

| *president lincoln* | *abraham lincoln* | *fishing* | *tropical fish* |
|---|---|---|---|
| lincoln | lincoln | fish | fish |
| president | america | farm | tropic |
| room | president | salmon | japan |
| bedroom | faith | new | aquarium |
| house | guest | wild | water |
| white | abraham | water | species |
| america | new | caught | aquatic |
| guest | room | catch | fair |
| serve | christian | tag | china |
| bed | history | time | coral |
| washington | public | eat | source |
| old | bedroom | raise | tank |
| office | war | city | reef |
| war | politics | people | animal |
| long | old | fishermen | tarpon |
| abraham | national | boat | fishery |

# Example from Top 50 Docs

| *president lincoln* | *abraham lincoln* | *fishing* | *tropical fish* |
|---|---|---|---|
| lincoln | lincoln | fish | fish |
| president | president | water | tropic |
| america | america | catch | water |
| new | abraham | reef | storm |
| national | war | fishermen | species |
| great | man | river | boat |
| white | civil | new | sea |
| war | new | year | river |
| washington | history | time | country |
| clinton | two | bass | tuna |
| house | room | boat | world |
| history | booth | world | million |
| time | time | farm | state |
| center | politics | angle | time |
| kennedy | public | fly | japan |
| room | guest | trout | mile |

# Query Language

- A query language is a way to give the engine more information about the relationships between terms, importance of terms, etc
  - Phrases
  - Date
  - Document Structure

# Query Languages and Indexes

- The query language is only as good as the index
  - Your query language can support phrases,
  - It can support document structure,
  - It can support date ranges,
  - But none of that matters if there's no support in the index!

# Query Languages and Retrieval Models

- Query language is only as good as the retrieval model behind it
  - If your retrieval model does not model phrases,
  - If it does not model document structure,
  - If it does not model date ranges,
  - Then a query language supporting those is no use.
- Most of the models we've discussed so far only model terms independently

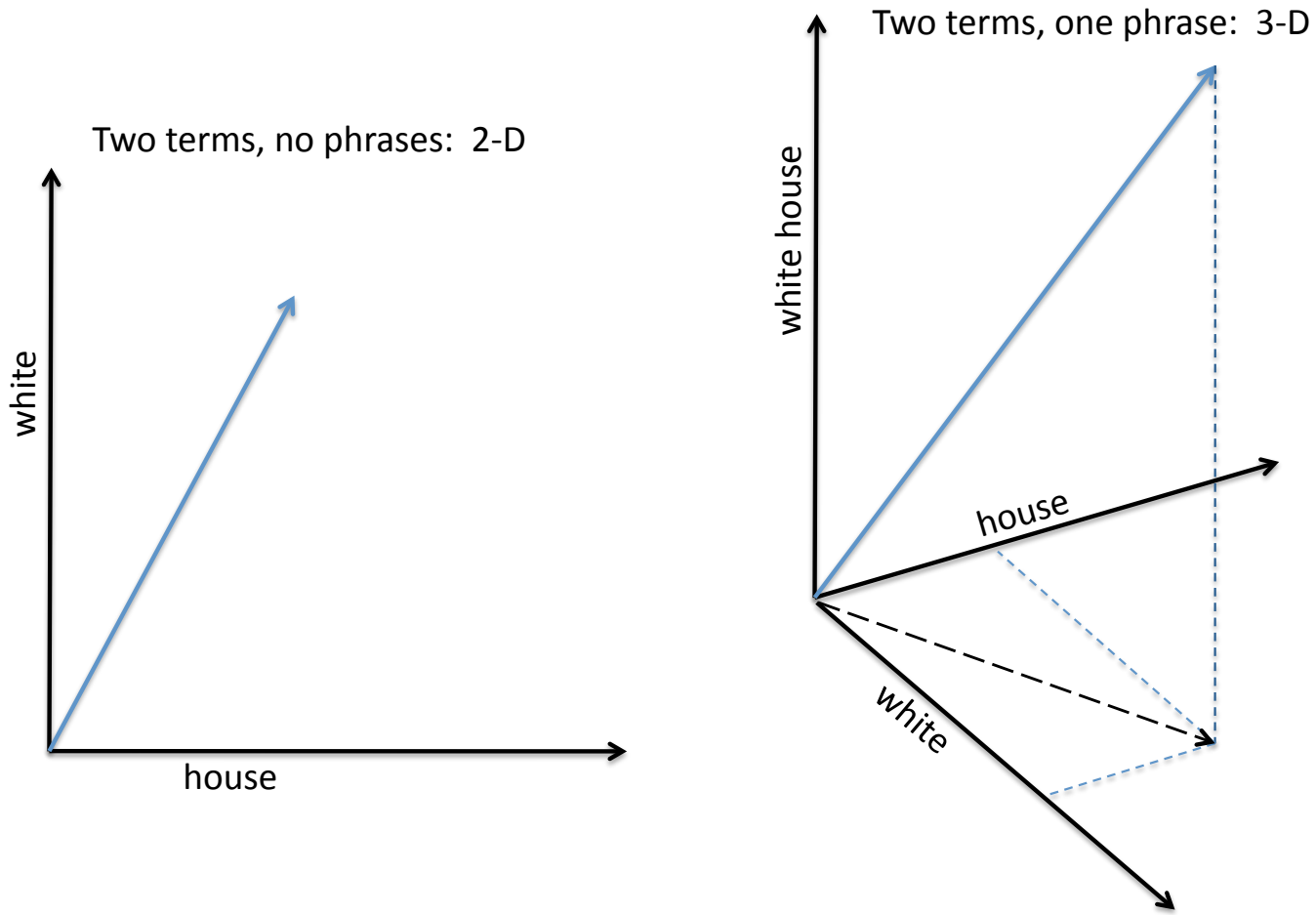
# A Simple Example

- Suppose we have an inverted index that can support phrases
- We have an engine that uses the vector space retrieval model and cosine similarity scoring
- We want to add support for phrases to the query language
  - Vector space model does not support phrases by default
  - We need to add phrase support to the VSM

# Phrases in the VSM

- Add phrase support to the vector space model
  - Document represented as vector of feature weights
  - Features are terms and phrases
  - Therefore each phrase becomes a new dimension in the vector space
  - Term weight: $$w_{ik} = \frac{tf_{ik}}{|D_k|} \log \frac{N}{n_i}$$
  - Phrase weight: $$\bar{p}w_{ik} = \frac{pf_{ik}}{|D_k|} \log \frac{N}{np_i}$$

# Phrases in the VSM




Some of the slides in this section are taken from Ben Carterette's course in University of Delaware

# A More Complex Example

- Suppose we have an inverted index that supports field information
- We have an engine that uses the vector space retrieval model and cosine similarity scoring
- We want to add support for field information to the query language, e.g. intitle: white house
  - Vector space model does not support this by default
  - We need to add field support to the VSM

# Fields in the VSM

- One possibility: separate term weights for each field the term appears in

$$w_{ik}^{title} = \frac{tf_{ik}^{title}}{|title_k|} \log \frac{N}{n_i^{title}}$$

$$w_{ik}^{body} = \frac{tf_{ik}^{body}}{|body_k|} \log \frac{N}{n_i^{body}}$$

$$\ldots$$

# Fields in the VSM

- Another possibility: weigh terms according to the fields they occur in

$$w_{ik} = \frac{tf_{ik}}{|D_k|} \log \frac{N}{n_i} \prod_{f:i \in f} \frac{fw_f}{|f_k|}$$

a field weight factor

length of field in k

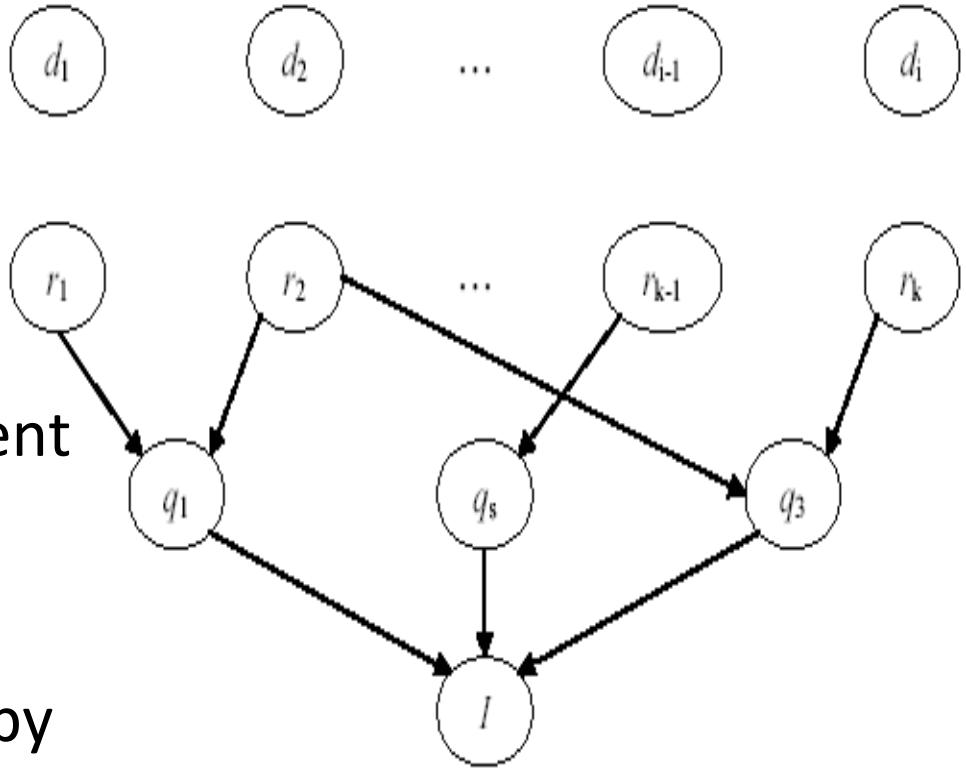
- If <title> weight is high, terms that appear in title will count for more in cosine similarity
- If <script> weight is low, terms that appear in script will count for less in cosine similarity

# Combining Evidence

- Effective retrieval requires the combination of many pieces of evidence about a document's potential relevance
  - Many types of evidence
    - structure, PageRank, metadata, even scores from different models
- *Inference network* model is one approach to combining evidence
  - uses Bayesian network formalism

# Inference Network Framework

- Node types
  - document ($d_i$)
  - concept ($r_i$)
  - query ($q_i$)
  - information need ($I$)
- Set evidence at document nodes
- Run belief propagation
- Documents are scored by P($I$ = true | $d_i$ = true)



# Network Semantics

- All events in network are binary
- Events associated with each node:

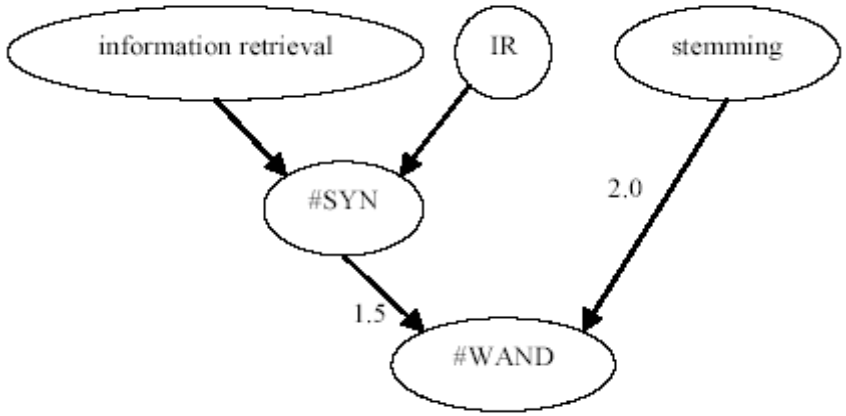
  $d_i$ – document *i* is observed

  $r_i$ – representation concept *i* is observed

  $q_i$ – query representation *i* is observed

  $I$ – information need is satisfied

# Example Query



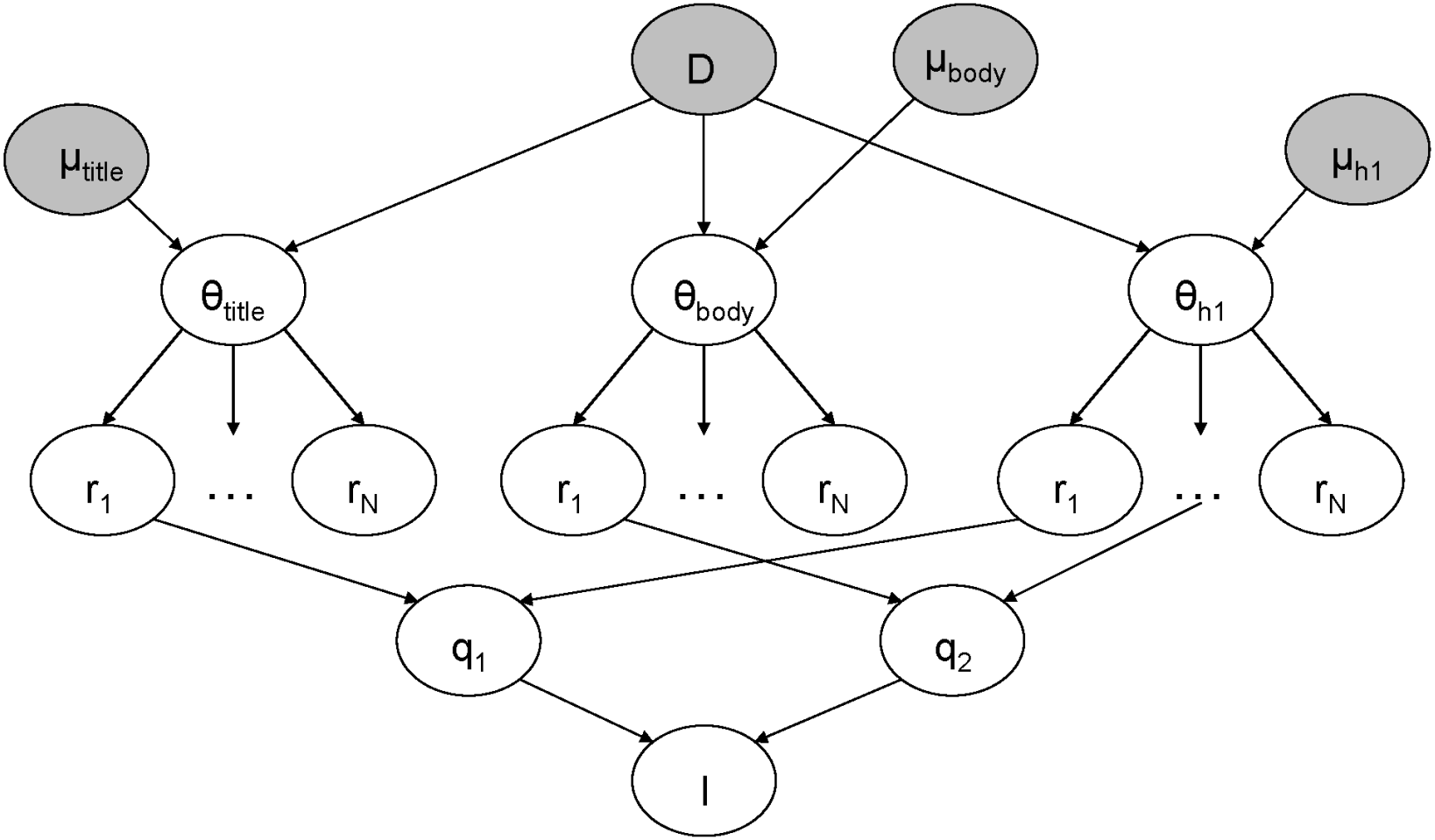

## Unstructured:

```
stemming information retrieval
```

## Structured:

```
#wand(1.5 #syn(#phrase(information retrieval) IR)
      2.0 stemming)
```

# Inference Network



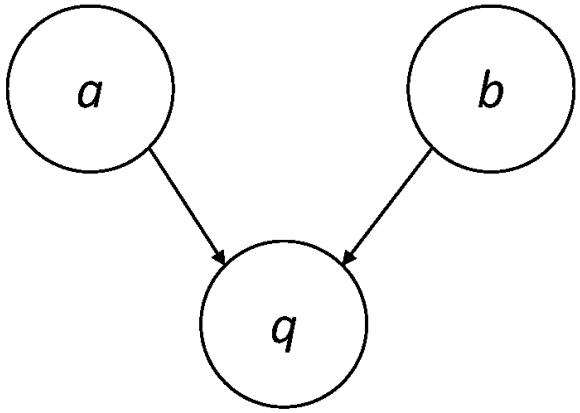
# Inference Network

- *Document node* (D) corresponds to the event that a document is observed
- *Representation nodes* ($r_i$) are document features (evidence)
  - Probabilities associated with those features are based on language models θ estimated using the parameters μ
  - one language model for each significant document structure
  - $r_i$ nodes can represent proximity features, or other types of evidence (e.g. date)

# Inference Network

- *Query nodes* ($q_i$) are used to combine evidence from representation nodes and other query nodes
  - represent the occurrence of more complex evidence and document features
  - a number of combination operators are available
- *Information need node* (I) is a special query node that combines all of the evidence from the other query nodes
  - network computes $P(I|D, \mu)$

# Example: AND Combination




*a* and *b* are *parent* nodes for q

| $P(q = \text{TRUE}\|a, b)$ | $a$ | $b$ |
|---|---|---|
| $0$ | FALSE | FALSE |
| $0$ | FALSE | TRUE |
| $0$ | TRUE | FALSE |
| $1$ | TRUE | TRUE |

# Example: AND Combination

- Combination must consider all possible states of parents
- Some combinations can be computed efficiently

$$
\begin{aligned}
bel_{and}(q) &= p_{00}P(a = \text{FALSE})P(b = \text{FALSE}) \\
&\quad + p_{01}P(a = \text{FALSE})P(b = \text{TRUE}) \\
&\quad + p_{10}P(a = \text{TRUE})P(b = \text{FALSE}) \\
&\quad + p_{11}P(a = \text{TRUE})P(b = \text{TRUE}) \\
&= 0 \cdot (1 - p_a)(1 - p_b) + 0 \cdot (1 - p_a)p_b + 0 \cdot p_a(1 - p_b) + 1 \cdot p_a p_b \\
&= p_a p_b
\end{aligned}
$$

# Inference Network Operators

$$
\begin{aligned}
bel_{not}(q) &= 1 - p_1 \\
bel_{or}(q) &= 1 - \prod_{i}^{n}(1 - p_i) \\
bel_{and}(q) &= \prod_{i}^{n} p_i \\
bel_{wand}(q) &= \prod_{i}^{n} p_i^{wt_i} \\
bel_{max}(q) &= max\{p_1, p_2, \ldots, p_n\} \\
bel_{sum}(q) &= \frac{\sum_i^n p_i}{n} \\
bel_{wsum}(q) &= \frac{\sum_i^n wt_i p_i}{\sum_i^n wt_i}
\end{aligned}
$$

# Galago Query Language

- A document is viewed as a sequence of text that may contain arbitrary tags
- A single *context* is generated for each unique tag name
- An *extent* is a sequence of text that appears within a single begin/end tag pair of the same type as the context

# Galago Query Language

```
<html>
<head>
<title>Department Descriptions</title>
</head>
<body>
The following list describes ...
<h1>Agriculture</h1> ...
<h1>Chemistry</h1> ...
<h1>Computer Science</h1> ...
<h1>Electrical Engineering</h1> ...
</body>
</html>
```

title context:

```
<title>Department Descriptions</title>
```

h1 context:

```
<h1>Agriculture</h1>
<h1>Chemistry</h1> ...
<h1>Computer Science</h1> ...
<h1>Electrical Engineering</h1> ...
```

body context:

```
<body> The following list describes ...
<h1>Agriculture</h1> ...
<h1>Chemistry</h1> ...
<h1>Computer Science</h1> ...
<h1>Electrical Engineering</h1> ...
</body>
```

# Galago Query Language

*Simple terms*:

term – term that will be normalized and stemmed.
"term" – term is not normalized or stemmed.
*Examples*:
presidents
"NASA"

# Galago Query Language

*Proximity terms*:

#od:N( ... ) – ordered window – terms must appear ordered, with at most N-1 terms between each.
#od( ... ) – unlimited ordered window – all terms must appear ordered anywhere within current context.
#uw:N( ... ) – unordered window – all terms must appear within a window of length N in any order.
#uw( ... ) – unlimited unordered window – all terms must appear within current context in any order.
*Examples*:
#od:1(white house) – matches "white house" as an exact phrase.
#od:2(white house) – matches "white * house" (where * is any word or null).
#uw:2(white house) – matches "white house" and "house white".

# Galago Query Language

*Synonyms*:

#syn( ... )
#wsyn( ... )
*Examples*:
#syn(dog canine) – simple synonym based on two terms.
#syn( #od:1(united states) #od:1(united states of america) ) – creates a synonym from two proximity terms.
#wsyn( 1.0 donald 0.8 don 0.5 donnie ) – weighted synonym indicating relative importance of terms.

# Galago Query Language

*Anonymous terms*:

#any:.() – used to match extent types
*Examples*:
#any:person() – matches any occurrence of a person extent.
#od:1(lincoln died in #any:date()) – matches exact phrases of the form:"lincoln died in <date>...</date>".

# Galago Query Language

*Context restriction and evaluation*:

expression.C1,,...,CN – matches when the expression appears in all contexts C1 through CN.
expression.(C1,...,CN) – evaluates the expression using the language model defined by the concatenation of contexts C1...CN within the document.
*Examples*:
dog.title – matches the term "dog" appearing in a title extent.
#uw(smith jones).author – matches when the two names "smith" and "jones" appear in an author extent.
dog.(title) – evaluates the term based on the title language model for the document.
#od:1(abraham lincoln).person.(header) – builds a language model from all of the "header" text in the document and evaluates #od:1(abraham lincoln).person in that context (i.e., matches only the exact phrase appearing within a person extent within the header context).

# Galago Query Language

*Belief operators*:

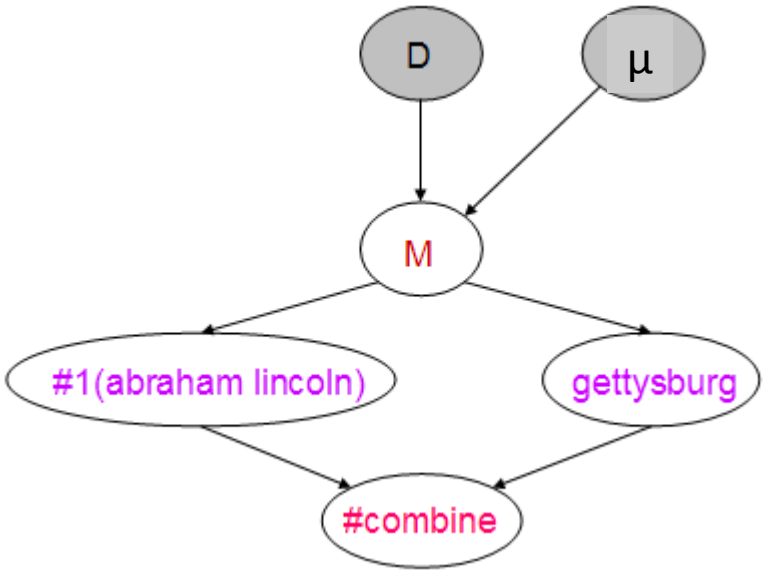
#combine(...) – this operator is a normalized version of the $bel_{and}(q)$ operator in the inference network model. See the discussion below for more details.
#weight(...) – this is a normalized version of the $bel_{wand}(q)$ operator.
#filter(...) – this operator is similar to #combine, but with the difference that the document must contain at least one instance of all terms (simple, proximity, synonym, etc.). The evaluation of nested belief operators is not changed.

# Galago Query Language

*Examples*:
#combine( #syn(dog canine) training ) – rank by two terms, one of which is a synonym.
#combine( biography #syn(#od:1(president lincoln) #od:1(abraham lincoln)) ) – rank using two terms, one of which is a synonym of "president lincoln" and "abraham lincoln".
#weight( 1.0 #od:1(civil war) 3.0 lincoln 2.0 speech ) – rank using three terms, and weight the term "lincoln" as most important, followed by "speech", then "civil war".
#filter( aquarium #combine(tropical fish) ) – consider only those documents containing the word "aquarium" and "tropical" or "fish", and rank them according to the query #combine(aquarium #combine(tropical fish)).
#filter( #od:1(john smith).author) #weight( 2.0 europe 1.0 travel ) – rank documents about "europe" or "travel" that have "John Smith" in the author context.

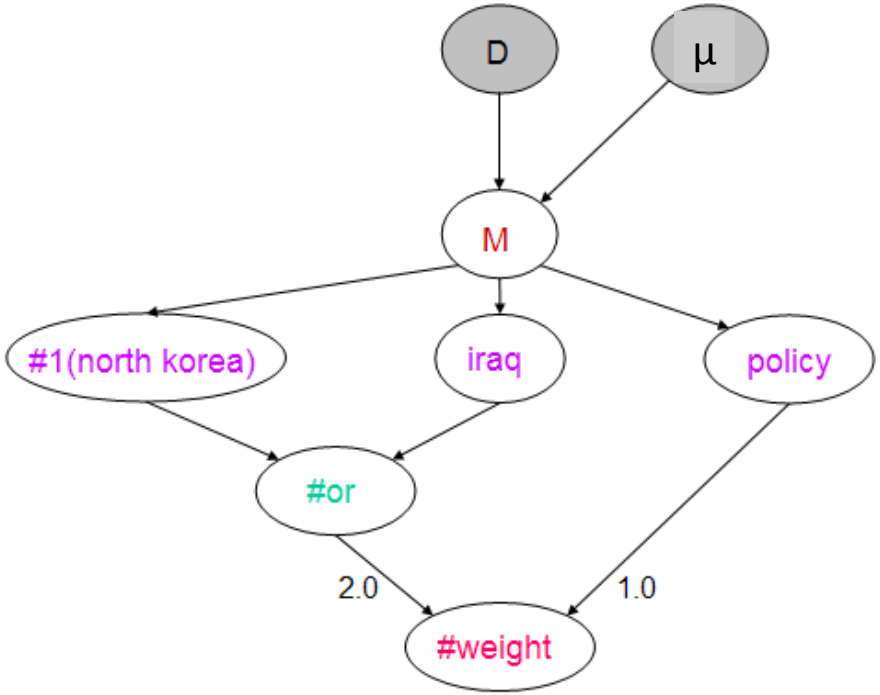
# Inference Network Example

Query: #combine( #1( abraham lincoln ) gettysburg )



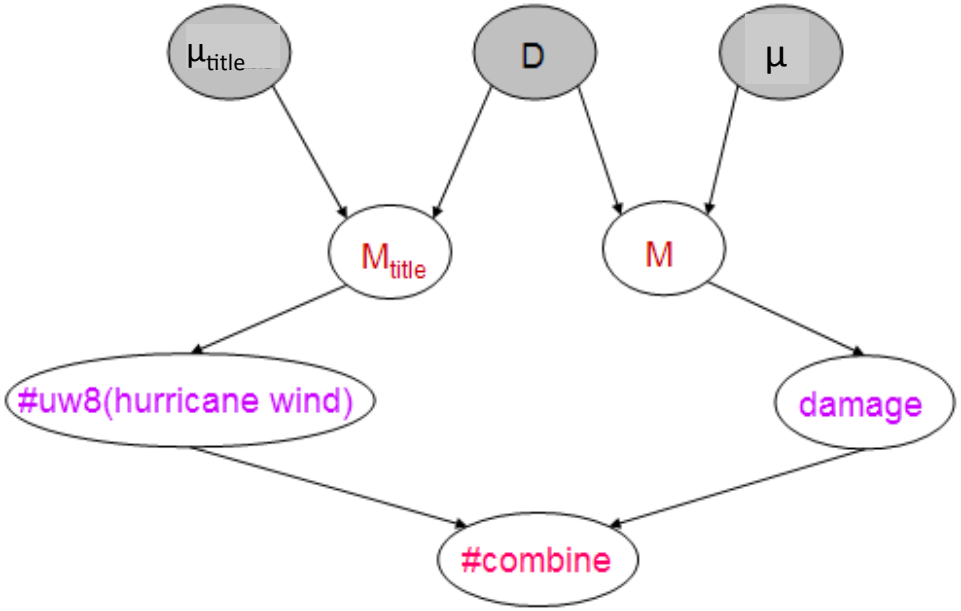
# Inference Network Example

Query:  #weight( 2.0 #or( #1(north korea) iraq ) 1.0 policy )



# Inference Network Example

Query: #combine( #uw8( hurricane wind ).(title) damage )



# Web Search

- Most important, but not only, search application
- Major differences to TREC news
  - Size of collection
  - Connections between documents
  - Range of document types
  - Importance of spam
  - Volume of queries
  - Range of query types

# Search Taxonomy

- *Informational*
  - Finding information about some topic which may be on one or more web pages
  - Topical search
- *Navigational*
  - finding a particular web page that the user has either seen before or is assumed to exist
- *Transactional*
  - finding a site where a task such as shopping or downloading music can be performed

# Web Search

- For effective navigational and transactional search, need to combine features that reflect *user relevance*
- Commercial web search engines combine evidence from *hundreds* of features to generate a ranking score for a web page
  - page content, page metadata, anchor text, links (e.g., PageRank), and user behavior (click logs)
  - page metadata – e.g., "age", how often it is updated, the URL of the page, the domain name of its site, and the amount of text content

# Search Engine Optimization

- *SEO*: understanding the relative importance of features used in search and how they can be manipulated to obtain better search rankings for a web page
  - e.g., improve the text used in the title tag, improve the text in heading tags, make sure that the domain name and URL contain important keywords, and try to improve the anchor text and link structure
  - Some of these techniques are regarded as not appropriate by search engine companies

# Web Search

- In TREC evaluations, most effective features for navigational search are:
  - text in the title, body, and heading (h1, h2, h3, and h4) parts of the document, the anchor text of all links pointing to the document, the PageRank number, and the inlink count
- Given size of Web, many pages will contain all query terms
  - Ranking algorithm focuses on discriminating between these pages
  - Word proximity is important

# Term Proximity

- Many models have been developed
- N-grams are commonly used in commercial web search
- *Dependence model* based on inference net has been effective in TREC - e.g.

```
#weight(
        0.8 #combine(embryonic stem cells)
        0.1 #combine( #od:1(stem cells) #od:1(embryonic stem)
                      #od:1(embryonic stem cells))
        0.1 #combine( #uw:8(stem cells) #uw:8(embryonic cells)
                      #uw:8(embryonic stem) #uw:12(embryonic stem cells)))
```

# Example Web Query

```
#weight(
        0.1 #weight( 0.6 #prior(pagerank) 0.4 #prior(inlinks))
        1.0 #weight(
            0.9 #combine(
                #weight( 1.0 pet.(anchor) 1.0 pet.(title)
                         3.0 pet.(body) 1.0 pet.(heading))
                #weight( 1.0 therapy.(anchor) 1.0 therapy.(title)
                         3.0 therapy.(body) 1.0 therapy.(heading)))
            0.1 #weight(
                1.0 #od:1(pet therapy).(anchor) 1.0 #od:1(pet therapy).(title)
                3.0 #od:1(pet therapy).(body) 1.0 #od:1(pet therapy).(heading))
            0.1 #weight(
                1.0 #uw:8(pet therapy).(anchor) 1.0 #uw:8(pet therapy).(title)
                3.0 #uw:8(pet therapy).(body) 1.0 #uw:8(pet therapy).(heading)))
        )
```

# Machine learning for IR ranking

- This "good idea" has been actively researched – and actively deployed by major web search engines – in the last years
- Why didn't it happen earlier?
  - Modern supervised ML has been around for about 20 years…
  - Naïve Bayes has been around for about 50 years…

# Machine learning for IR ranking

- There were a whole bunch of precursors:
  - Wong, S.K. et al. 1988. Linear structure in information retrieval. SIGIR 1988.
  - Fuhr, N. 1992. Probabilistic methods in information retrieval. Computer Journal.
  - Gey, F. C. 1994. Inferring probability of relevance using the method of logistic regression. SIGIR 1994.
  - Herbrich, R. et al. 2000. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers.

# Why weren't early attempts very successful/influential?

- **Limited training data**
  - Especially for real world use it was very hard to gather test collection queries and relevance judgments that are representative of real user needs and judgments on documents returned
- Poor machine learning techniques
- Insufficient customization to IR problem
- Not enough features for ML to show value

# Why wasn't ML much needed?

- Traditional ranking functions in IR used a very small number of features, e.g.,
  - Term frequency
  - Inverse document frequency
  - Document length
- It was easy to tune weighting coefficients by hand

# Why is ML needed now?

- Modern systems – especially on the Web – use a great number of features:
    - Arbitrary useful features – not a single unified model
  - Log frequency of query word in anchor text?
  - Query word in color on page?
  - # of images on page?
  - # of (out) links on page?
  - PageRank of page?
  - URL length?
  - URL contains "~"?
  - Page edit recency?
  - Page length?
- The New York Times (2008-06-03) quoted Amit Singhal as saying Google was using over 200 such features.

# Generative vs. Discriminative

- Most of the probabilistic retrieval models presented so far fall into the category of *generative models*
  - A generative model assumes that documents were generated from some underlying model (in this case, usually a multinomial distribution) and uses training data to estimate the parameters of the model
  - probability of belonging to a class (i.e. the relevant documents for a query) is then estimated using Bayes' Rule and the document model

# Generative vs. Discriminative

- A *discriminative* model estimates the probability of belonging to a class directly from the observed features of the document based on the training data
- Generative models perform well with low numbers of training examples
- Discriminative models usually have the advantage given enough training data
  - Can also easily incorporate many features

# Discriminative Models for IR

- Discriminative models can be trained using explicit relevance judgments or click data in query logs
- Large class of algorithms called *learning to rank*
  - Learns weights on a linear (or non-linear) combination of features that is used to rank documents
  - Best weights optimize a performance metric

# Simple example: Using classification for ad hoc IR

- Collect a training corpus of (q, d, r) triples
  - Relevance r is here binary (but may be multiclass, with 3–7 values)
  - Document is represented by a feature vector
    - **x** = (α, ω) α is cosine similarity, ω is minimum query window size
      - ω is the the shortest text span that includes all query words
      - Query term proximity is a **very important** new weighting factor

| example | docID | query | cosine score | $\omega$ | judgment |
|---|---|---|---|---|---|
| $\Phi_1$ | 37 | linux operating system | 0.032 | 3 | *relevant* |
| $\Phi_2$ | 37 | penguin logo | 0.02 | 4 | *nonrelevant* |
| $\Phi_3$ | 238 | operating system | 0.043 | 2 | *relevant* |
| $\Phi_4$ | 238 | runtime environment | 0.004 | 2 | *nonrelevant* |
| $\Phi_5$ | 1741 | kernel layer | 0.022 | 3 | *relevant* |
| $\Phi_6$ | 2094 | device driver | 0.03 | 2 | *relevant* |
| $\Phi_7$ | 3191 | device driver | 0.027 | 5 | *nonrelevant* |

# Simple example: Using classification for ad hoc IR

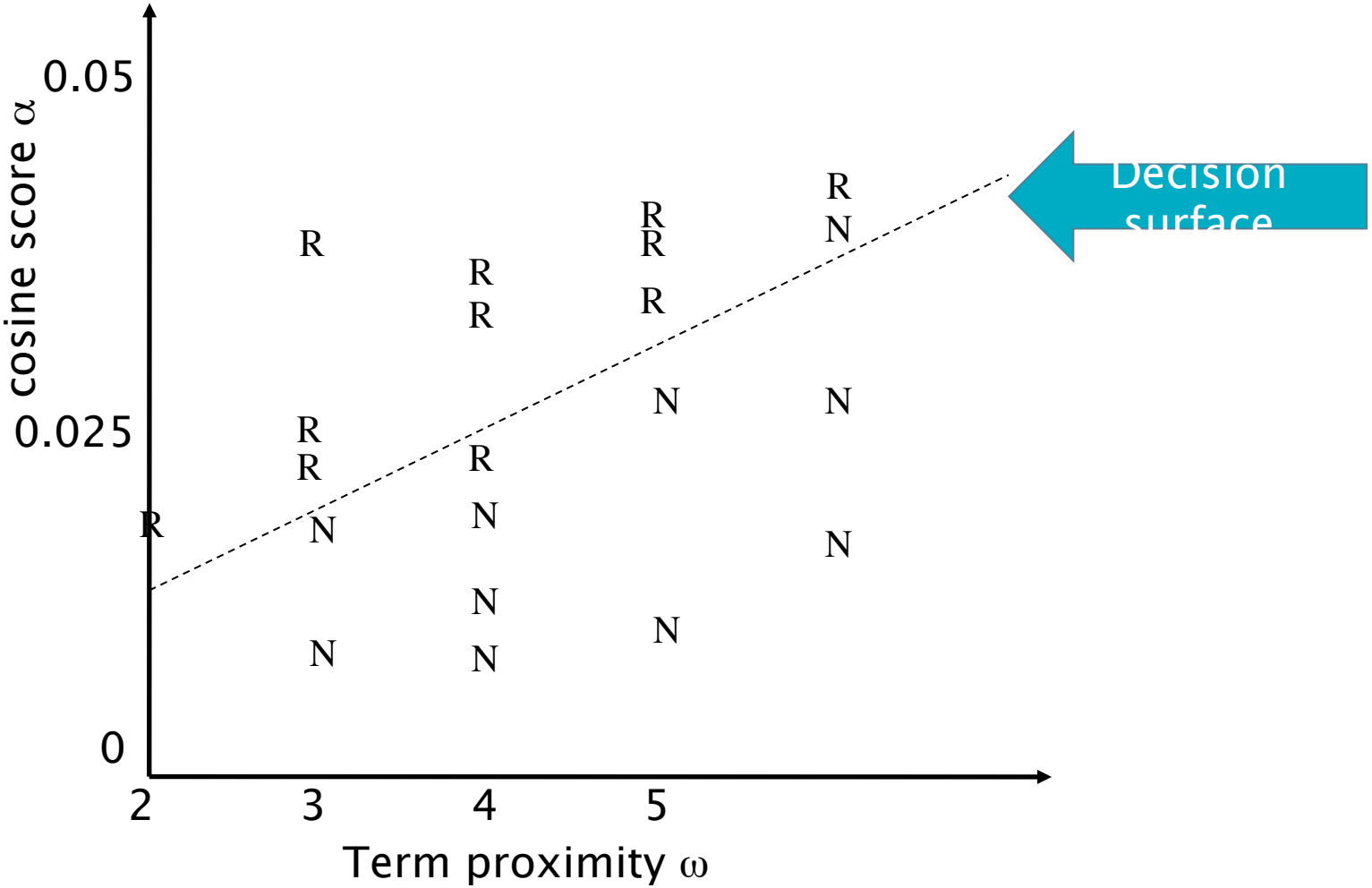
- A linear score function is then

  $$\text{Score}(d, q) = \text{Score}(\alpha, \omega) = a\alpha + b\omega + c$$

- And the linear classifier is

  Decide relevant if $\text{Score}(d, q) > \theta$

# Simple example:
# Using classification for ad hoc IR



# More complex example of using classification for search ranking [Nallapati 2004]

- We can generalize this to classifier functions over more features
- We can use more advanced methods for learning the linear classifier weights

# "Learning to rank"

- Classification probably isn't the right way to think about approaching ad hoc IR:
  - Classification problems: Map to an unordered set of classes
  - Regression problems: Map to a real value
  - Ordinal regression problems: Map to an ordered set of classes
- This formulation gives extra power:
  - Relations between relevance levels are modeled
  - Documents are good versus other documents for query given collection; not an absolute scale of goodness

# Machine Learning and IR

- Considerable interaction between these fields
  - Rocchio algorithm (60s) is a simple learning approach
  - 80s, 90s: learning ranking algorithms based on user feedback
  - 2000s: text categorization
- Limited by amount of training data
- Web query logs have generated new wave of research
  - e.g., “Learning to Rank”

# Ranking SVM

- Training data is

$$(q_1, r_1), (q_2, r_2), \ldots, (q_n, r_n)$$

  - r is partial rank information
    - if document $d_a$ should be ranked higher than $d_b$, then $(d_a, d_b) \in r_i$
  - partial rank information comes from relevance judgments (allows multiple levels of relevance) or click data
    - e.g., $d_1$, $d_2$ and $d_3$ are the documents in the first, second and third rank of the search output, only $d_3$ clicked on → ($d_3$, $d_1$) and ($d_3$, $d_2$) will be in desired ranking for this query

# Ranking SVM

- Learning a linear ranking function $\vec{w}.\vec{d}_a$
  - where *w* is a weight vector that is adjusted by learning
  - $d_a$ is the vector representation of the features of document
  - *non-linear* functions also possible
- Weights represent importance of features
  - learned using training data
  - e.g.,

$$\vec{w}.\vec{d} = (2,1,2).(2,4,1) = 2.2 + 1.4 + 2.1 = 10$$

# Ranking SVM

- Learn *w* that satisfies as many of the following conditions as possible:

$$\begin{aligned} \forall (d_i, d_j) \in r_1 \quad &: \quad \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} \\ &\ldots \\ \forall (d_i, d_j) \in r_n \quad &: \quad \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} \end{aligned}$$

- Can be formulated as an *optimization* problem

# Ranking SVM

$$
\begin{aligned}
minimize: \quad & \frac{1}{2}\vec{w}.\vec{w} + C\sum \xi_{i,j,k} \\
subject\ to: & \\
\forall (d_i, d_j) \in r_1 \quad : \quad & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} + 1 - \xi_{i,j,1} \\
& \ldots \\
\forall (d_i, d_j) \in r_n \quad : \quad & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} + 1 - \xi_{i,j,n} \\
& \forall i \forall j \forall k : \xi_i, j, k \geq 0
\end{aligned}
$$

– $\xi$, known as a slack variable, allows for misclassification of difficult or noisy training examples, and *C* is a parameter that is used to prevent overfitting

# Ranking SVM

- Software available to do optimization
- Each pair of documents in our training data can be represented by the vector:

$$(\vec{d}_i - \vec{d}_j)$$

- Score for this pair is:

$$\vec{w}.(\vec{d}_i - \vec{d}_j)$$

- SVM classifier will find a *w* that makes the smallest score as large as possible
  - make the differences in scores as large as possible for the pairs of documents that are hardest to rank

# Note: Linear vs. nonlinear weighting

- Both of the methods that we've seen use a linear weighting of document features that are indicators of relevance, as has most work in this area
- Much of traditional IR weighting involves nonlinear scaling of basic measurements (such as log-weighting of term frequency, or idf)
- At the present time, machine learning is very good at producing optimal weights for features in a linear combination, but it is not good at coming up with good nonlinear scalings of basic measurements
- This area remains the domain of human feature engineering

# Latent semantic indexing

Recall: Term-document matrix

| | Anthony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| anthony | 5.25 | 3.18 | 0.0 | 0.0 | 0.0 | 0.35 |
| brutus | 1.21 | 6.10 | 0.0 | 1.0 | 0.0 | 0.0 |
| caesar | 8.59 | 2.54 | 0.0 | 1.51 | 0.25 | 0.0 |
| calpurnia | 0.0 | 1.54 | 0.0 | 0.0 | 0.0 | 0.0 |
| cleopatra | 2.85 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| mercy | 1.51 | 0.0 | 1.90 | 0.12 | 5.25 | 0.88 |
| worser | 1.37 | 0.0 | 0.11 | 4.15 | 0.25 | 1.95 |

. . .

- This matrix is the basis for computing the similarity between documents and queries. Can we transform this matrix, so that we get a better measure of similarity between documents and queries?

# Latent semantic indexing: Overview

- We will decompose the term-document matrix into a product of matrices.
- The particular decomposition we'll use: singular value decomposition (SVD).
- SVD: $C = U\Sigma V^T$ (where $C$ = term-document matrix)
- We will then use the SVD to compute a new, improved term-document matrix $C'$.
- We'll get better similarity values out of $C'$ (compared to $C$).
- Using SVD for this purpose is called latent semantic indexing or LSI.

# Example of $C = U\Sigma V^T$ : The matrix $C$

| $C$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 1 | 0 | 1 | 0 | 0 | 0 |
| boat | 0 | 1 | 0 | 0 | 0 | 0 |
| ocean | 1 | 1 | 0 | 0 | 0 | 0 |
| wood | 1 | 0 | 0 | 1 | 1 | 0 |
| tree | 0 | 0 | 0 | 1 | 0 | 1 |

- This is a standard term-document matrix. We use a non-weighted matrix here to simplify the example.

# Example of $C = U\Sigma V^T$ : The matrix $U$

| $U$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ship | −0.44 | −0.30 | 0.57 | 0.58 | 0.25 |
| boat | −0.13 | −0.33 | −0.59 | 0.00 | 0.73 |
| ocean | −0.48 | −0.51 | −0.37 | 0.00 | −0.61 |
| wood | −0.70 | 0.35 | 0.15 | −0.58 | 0.16 |
| tree | −0.26 | 0.65 | −0.41 | 0.58 | −0.09 |

- One row per term, one column per min(*M*,*N*) where *M* is the number of terms and *N* is the number of documents. This is an orthonormal matrix: (i) Row vectors have unit length. (ii) Any two distinct row vectors are orthogonal to each other. Think of the dimensions as "semantic" dimensions that capture distinct topics like politics, sports, economics. Each number $u_{ij}$ in the matrix indicates how strongly related term *i* is to the topic represented by semantic dimension *j* .

# Example of $C = U\Sigma V^T$ : The matrix Σ

| Σ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 2.16 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 1.59 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 1.28 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.39 |

- This is a square, diagonal matrix of dimensionality min(*M*,*N*) × min(*M*,*N*). The diagonal consists of the singular values of *C*. The magnitude of the singular value measures the importance of the corresponding semantic dimension. We'll make use of this by omitting unimportant dimensions.

# Example of $C = U\Sigma V^T$ : The matrix $V^T$

| $V^T$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| 1 | −0.75 | −0.28 | −0.20 | −0.45 | −0.33 | −0.12 |
| 2 | −0.29 | −0.53 | −0.19 | 0.63 | 0.22 | 0.41 |
| 3 | 0.28 | −0.75 | 0.45 | −0.20 | 0.12 | −0.33 |
| 4 | 0.00 | 0.00 | 0.58 | 0.00 | −0.58 | 0.58 |
| 5 | −0.53 | 0.29 | 0.63 | 0.19 | 0.41 | −0.22 |

One column per document, one row per min(*M*,*N*) where *M* is the number of terms and *N* is the number of documents. Again: This is an orthonormal matrix: (i) Column vectors have unit length. (ii) Any two distinct column vectors are orthogonal to each other. These are again the semantic dimensions from the term matrix *U* that capture distinct topics like politics, sports, economics. Each number $v_{ij}$ in the matrix indicates how strongly related document *i* is to the topic represented by semantic dimension *j* .

# Example of $C = U\Sigma V^T$ : All four matrices

| $C$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 1 | 0 | 1 | 0 | 0 | 0 |
| boat | 0 | 1 | 0 | 0 | 0 | 0 |
| ocean | 1 | 1 | 0 | 0 | 0 | 0 |
| wood | 1 | 0 | 0 | 1 | 1 | 0 |
| tree | 0 | 0 | 0 | 1 | 0 | 1 |

$=$

| $U$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ship | −0.44 | −0.30 | 0.57 | 0.58 | 0.25 |
| boat | −0.13 | −0.33 | −0.59 | 0.00 | 0.73 |
| ocean | −0.48 | −0.51 | −0.37 | 0.00 | −0.61 |
| wood | −0.70 | 0.35 | 0.15 | −0.58 | 0.16 |
| tree | −0.26 | 0.65 | −0.41 | 0.58 | −0.09 |

$\times$

| $\Sigma$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 2.16 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 1.59 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 1.28 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.39 |

$\times$

| $V^T$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| 1 | −0.75 | −0.28 | −0.20 | −0.45 | −0.33 | −0.12 |
| 2 | −0.29 | −0.53 | −0.19 | 0.63 | 0.22 | 0.41 |
| 3 | 0.28 | −0.75 | 0.45 | −0.20 | 0.12 | −0.33 |
| 4 | 0.00 | 0.00 | 0.58 | 0.00 | −0.58 | 0.58 |
| 5 | −0.53 | 0.29 | 0.63 | 0.19 | 0.41 | −0.22 |

# LSI: Summary

- We've decomposed the term-document matrix C into a product of three matrices.
- The term matrix U – consists of one (row) vector for each term
- The document matrix VT – consists of one (column) vector for each document
- The singular value matrix Σ – diagonal matrix with singular values, reflecting importance of each dimension
- Next: Why are we doing this?

# How we use the SVD in LSI

- Key property: Each singular value tells us how important its dimension is.
- By setting less important dimensions to zero, we keep the important information, but get rid of the "details".
- These details may
  - be noise – in that case, reduced LSI is a better representation because it is less noisy
  - make things dissimilar that should be similar – again reduced LSI is a better representation because it represents similarity better.
- Analogy for "fewer details is better"
  - Image of a bright red flower
  - Image of a black and white flower
  - Omitting color makes is easier to see similarity

# Reducing the dimensionality to 2

| $U$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ship | −0.44 | −0.30 | 0.00 | 0.00 | 0.00 |
| boat | −0.13 | −0.33 | 0.00 | 0.00 | 0.00 |
| ocean | −0.48 | −0.51 | 0.00 | 0.00 | 0.00 |
| wood | −0.70 | 0.35 | 0.00 | 0.00 | 0.00 |
| tree | −0.26 | 0.65 | 0.00 | 0.00 | 0.00 |

| $\Sigma_2$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 2.16 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 1.59 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

| $V^T$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| 1 | −0.75 | −0.28 | −0.20 | −0.45 | −0.33 | −0.12 |
| 2 | −0.29 | −0.53 | −0.19 | 0.63 | 0.22 | 0.41 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Actually, we only zero out singular values in Σ. This has the effect of setting the corresponding dimensions in U and $V^T$ to zero when computing the product $C = U\Sigma V^T$.

# Reducing the dimensionality to 2

| $C_2$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 0.85 | 0.52 | 0.28 | 0.13 | 0.21 | −0.08 |
| boat | 0.36 | 0.36 | 0.16 | −0.20 | −0.02 | −0.18 |
| ocean | 1.01 | 0.72 | 0.36 | −0.04 | 0.16 | −0.21 |
| wood | 0.97 | 0.12 | 0.20 | 1.03 | 0.62 | 0.41 |
| tree | 0.12 | −0.39 | −0.08 | 0.90 | 0.41 | 0.49 |

=

| $U$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ship | −0.44 | −0.30 | 0.57 | 0.58 | 0.25 |
| boat | −0.13 | −0.33 | −0.59 | 0.00 | 0.73 |
| ocean | −0.48 | −0.51 | −0.37 | 0.00 | −0.61 |
| wood | −0.70 | 0.35 | 0.15 | −0.58 | 0.16 |
| tree | −0.26 | 0.65 | −0.41 | 0.58 | −0.09 |

×

| $\Sigma_2$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 2.16 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 1.59 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

×

| $V^T$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| 1 | −0.75 | −0.28 | −0.20 | −0.45 | −0.33 | −0.12 |
| 2 | −0.29 | −0.53 | −0.19 | 0.63 | 0.22 | 0.41 |
| 3 | 0.28 | −0.75 | 0.45 | −0.20 | 0.12 | −0.33 |
| 4 | 0.00 | 0.00 | 0.58 | 0.00 | −0.58 | 0.58 |
| 5 | −0.53 | 0.29 | 0.63 | 0.19 | 0.41 | −0.22 |

# Recall unreduced decomposition $C=U\Sigma V^T$

| $C$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 1 | 0 | 1 | 0 | 0 | 0 |
| boat | 0 | 1 | 0 | 0 | 0 | 0 |
| ocean | 1 | 1 | 0 | 0 | 0 | 0 |
| wood | 1 | 0 | 0 | 1 | 1 | 0 |
| tree | 0 | 0 | 0 | 1 | 0 | 1 |

$=$

| $U$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ship | −0.44 | −0.30 | 0.57 | 0.58 | 0.25 |
| boat | −0.13 | −0.33 | −0.59 | 0.00 | 0.73 |
| ocean | −0.48 | −0.51 | −0.37 | 0.00 | −0.61 |
| wood | −0.70 | 0.35 | 0.15 | −0.58 | 0.16 |
| tree | −0.26 | 0.65 | −0.41 | 0.58 | −0.09 |

$\times$

| $\Sigma$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 2.16 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 1.59 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 1.28 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.39 |

$\times$

| $V^T$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| 1 | −0.75 | −0.28 | −0.20 | −0.45 | −0.33 | −0.12 |
| 2 | −0.29 | −0.53 | −0.19 | 0.63 | 0.22 | 0.41 |
| 3 | 0.28 | −0.75 | 0.45 | −0.20 | 0.12 | −0.33 |
| 4 | 0.00 | 0.00 | 0.58 | 0.00 | −0.58 | 0.58 |
| 5 | −0.53 | 0.29 | 0.63 | 0.19 | 0.41 | −0.22 |

# Original matrix $C$ vs. reduced $C_2 = U\Sigma_2 V^T$

| $C$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 1 | 0 | 1 | 0 | 0 | 0 |
| boat | 0 | 1 | 0 | 0 | 0 | 0 |
| ocean | 1 | 1 | 0 | 0 | 0 | 0 |
| wood | 1 | 0 | 0 | 1 | 1 | 0 |
| tree | 0 | 0 | 0 | 1 | 0 | 1 |

| $C_2$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 0.85 | 0.52 | 0.28 | 0.13 | 0.21 | −0.08 |
| boat | 0.36 | 0.36 | 0.16 | −0.20 | −0.02 | −0.18 |
| ocean | 1.01 | 0.72 | 0.36 | −0.04 | 0.16 | −0.21 |
| wood | 0.97 | 0.12 | 0.20 | 1.03 | 0.62 | 0.41 |
| tree | 0.12 | −0.39 | −0.08 | 0.90 | 0.41 | 0.49 |

# Why the reduced matrix is “better”?

| $C$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 1 | 0 | 1 | 0 | 0 | 0 |
| boat | 0 | 1 | 0 | 0 | 0 | 0 |
| ocean | 1 | 1 | 0 | 0 | 0 | 0 |
| wood | 1 | 0 | 0 | 1 | 1 | 0 |
| tree | 0 | 0 | 0 | 1 | 0 | 1 |

| $C_2$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 0.85 | 0.52 | 0.28 | 0.13 | 0.21 | −0.08 |
| boat | 0.36 | 0.36 | 0.16 | −0.20 | −0.02 | −0.18 |
| ocean | 1.01 | 0.72 | 0.36 | −0.04 | 0.16 | −0.21 |
| wood | 0.97 | 0.12 | 0.20 | 1.03 | 0.62 | 0.41 |
| tree | 0.12 | −0.39 | −0.08 | 0.90 | 0.41 | 0.49 |

Similarity of d2 and d3 in the original space: 0.
Similarity of d2 und d3 in the reduced space:
0.52 * 0.28 + 0.36 * 0.16 + 0.72 * 0.36 + 0.12 * 0.20 + - 0.39 * - 0.08 ≈ 0.52

# Why the reduced matrix is “better”?

| $C$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 1 | 0 | 1 | 0 | 0 | 0 |
| boat | 0 | 1 | 0 | 0 | 0 | 0 |
| ocean | 1 | 1 | 0 | 0 | 0 | 0 |
| wood | 1 | 0 | 0 | 1 | 1 | 0 |
| tree | 0 | 0 | 0 | 1 | 0 | 1 |

| $C_2$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
|---|---|---|---|---|---|---|
| ship | 0.85 | 0.52 | 0.28 | 0.13 | 0.21 | −0.08 |
| boat | 0.36 | 0.36 | 0.16 | −0.20 | −0.02 | −0.18 |
| ocean | 1.01 | 0.72 | 0.36 | −0.04 | 0.16 | −0.21 |
| wood | 0.97 | 0.12 | 0.20 | 1.03 | 0.62 | 0.41 |
| tree | 0.12 | −0.39 | −0.08 | 0.90 | 0.41 | 0.49 |

“boat” and “ship” are semantically similar. The “reduced” similarity measure reflects this.

# Why we use LSI in information retrieval

- LSI takes documents that are semantically similar (= talk about the same topics), . . .
- . . . but are not similar in the vector space (because they use different words) . . .
- . . . and re-represents them in a reduced vector space . . .
- . . . in which they have higher similarity.
- Thus, LSI addresses the problems of synonymy and semantic relatedness.
- Standard vector space: Synonyms contribute nothing to document similarity.
- Desired effect of LSI: Synonyms contribute strongly to document similarity.

# LSI: Comparison to other approaches

- Recap: Relevance feedback and query expansion are used to increase performance in information retrieval – if query and documents have (in the extreme case) no terms in common.
- LSI addresses the same problems as (pseudo) relevance feedback and query expansion . . .
- . . . and it has the same problems.

# Implementation

- Compute SVD of term-document matrix
- Reduce the space and compute reduced document representations
- Map the query into the reduced space $\vec{q}_2^T = \Sigma_2^{-1} U_2^T \vec{q}^T$.
- This follows from: $C_2 = U\Sigma_2 V^T \Rightarrow \Sigma_2^{-1} U^T C = V_2^T$
- Compute similarity of q2 with all reduced documents in V2.
- Output ranked list of documents as usual

# Topic Models

- Improved representations of documents
  - can also be viewed as improved smoothing techniques
  - improve estimates for words that are related to the topic(s) of the document
    - instead of just using background probabilities
- Approaches
  - *Latent* Semantic Indexing (LSI)
  - Probabilistic *Latent* Semantic Indexing (pLSI)
  - *Latent* Dirichlet Allocation (LDA)

# Topic Models

Notation and terminology (text collections)

- *Word*: the basic unit from a vocabulary of size $V$ (includes $V$ distinct words). The $v$th word is represented by

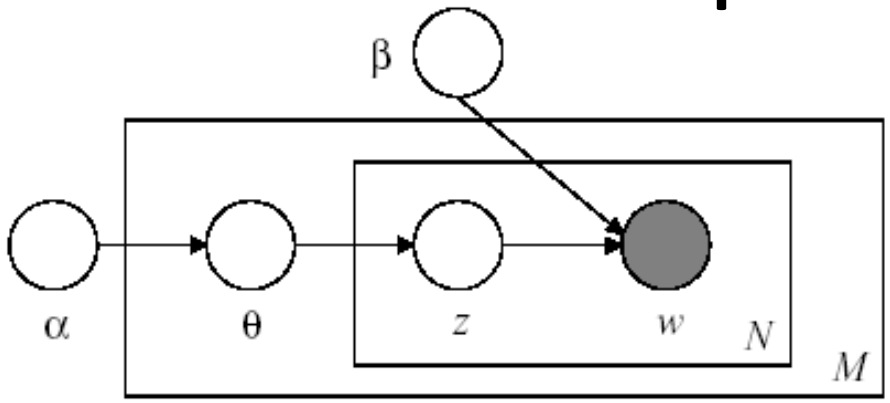
$$w = [\underbrace{0 \cdots 0 \; 1 \; 0 \cdots 0}_{v\text{th}}]^T \quad V\text{-dim}$$

- *Document*: a sequence of $N$ words. $W = [w_1, w_2, \cdots, w_N]$
- *Corpus*: a collection of M documents. $D = \{W_1, W_2, \cdots, W_M\}$

Assumptions:

- The words in a document are exchangeable;
- Documents are also exchangeable.

# Topic Models




$M, N, V, k$ fixed known parameters

$\alpha, \beta$ fixed unknown parameters

$\theta, z, w$ Random variables (*w* are observable)

Generative process for each document *W* in a corpus *D*:

1. Choose $\theta \sim Dirichlet(\alpha), \quad \theta \text{ and } \alpha \text{ are } k-\text{dim}$
2. For each of the $N$ words $w_n$

(a) Choose a topic $z_n \sim Multinomial(\theta), \quad z_n \text{ are } k-\text{dim index}$

(b) Choose a word $w_n \sim Multinomial(\beta_{z_n}), \quad \beta \text{ is a } k \times V \text{ matrix}$

$$\beta_{ij} = p(w^j = 1 \mid z^i = 1)$$

$\theta$ are document-level variables, $z$ and $w$ are word-level variables.

# LDA

- Model document as being generated from a *mixture* of topics

1. For each document $D$, pick a multinomial distribution $\theta_D$ from a Dirichlet distribution with parameter $\alpha$ ,
2. For each word position in document $D$,
   (a) pick a topic $z$ from the multinomial distribution $\theta_D$ ,
   (b) Choose a word $w$ from $P(w|z, \beta)$, a multinomial probability conditioned on the topic $z$ with parameter $\beta$.

# LDA

- Gives language model probabilities

$$P_{lda}(w|D) = P(w|\theta_D, \beta) = \sum_z P(w|z, \beta)P(z|\theta_D)$$

- Used to smooth the document representation by mixing them with the query likelihood probability as follows:

$$P(w|D) = \lambda \left( \frac{f_{w,D} + \mu \frac{c_w}{|C|}}{|D| + \mu} \right) + (1 - \lambda) P_{lda}(w|D)$$

# LDA

- If the LDA probabilities are used directly as the document representation, the effectiveness will be significantly reduced because the features are *too smoothed*
  - e.g., in typical TREC experiment, only 400 topics used for the *entire* collection
  - generating LDA topics is expensive
- When used for smoothing, effectiveness is improved

# LDA Example

– Top words from 4 LDA topics from TREC news

| *Arts* | *Budgets* | *Children* | *Education* |
|---|---|---|---|
| new | million | children | school |
| film | tax | women | students |
| show | program | people | schools |
| music | budget | child | education |
| movie | billion | years | teachers |
| play | federal | families | high |
| musical | year | work | public |
| best | spending | parents | teacher |
| actor | new | says | bennett |
| first | state | family | manigat |
| york | plan | welfare | namphy |
| opera | money | men | state |
| theater | programs | percent | president |
| actress | government | care | elementary |
| love | congress | life | haiti |

# Summary

- Best retrieval model depends on application and data available
- Evaluation corpus (or test collection), training data, and user data are all critical resources
- Open source search engines can be used to find effective ranking algorithms
  - Galago query language makes this particularly easy