Information Retrieval

Models for Information Retrieval 1

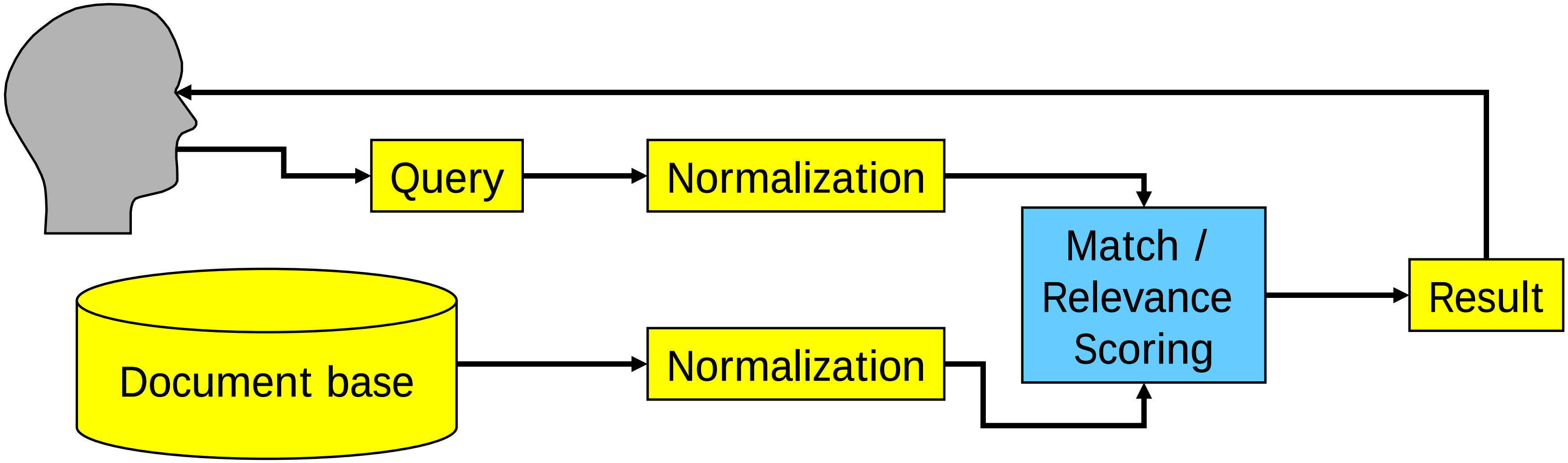
Ulf Leser

Content of this Lecture

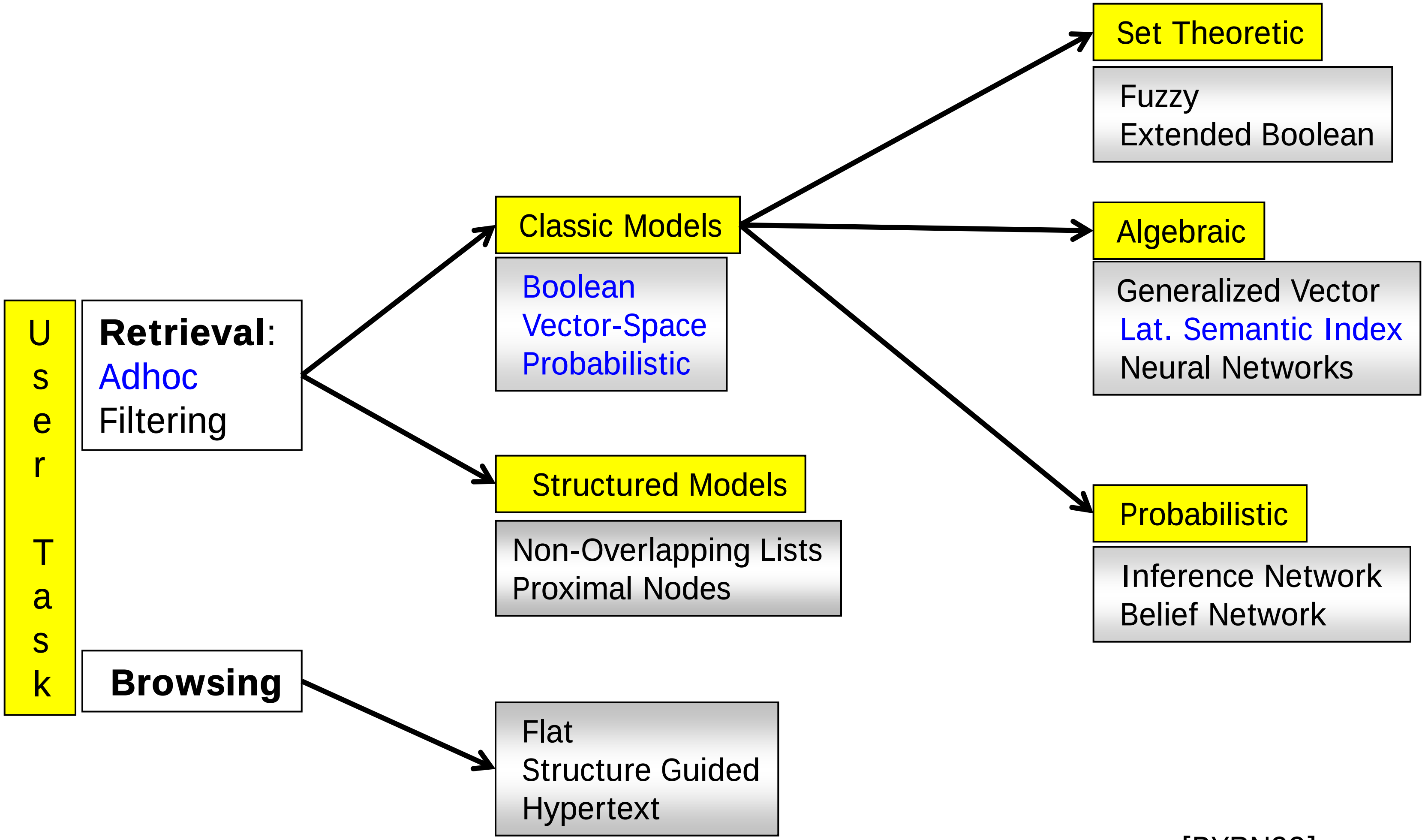
- IR Models
- Boolean Model
- Vector Space Model
- Relevance Feedback in the VSM
- Probabilistic Model
- Latent Semantic Indexing
- Other IR Models

Information Retrieval Core

- The core question in IR:
  Which from a given set of (normalized) documents are relevant for a given query?
- Ranking: How relevant for a given query is each document?



Judging Relevance



[BYRN99]

Notation

- All models we discuss use the "Bag of Words" view
- Definition
  - *Let D be the set of all normalized documents, $d \in D$ is a document*
  - *Let K be the set of all unique tokens in D, $k \in K$ is a token*
    - *Can as well be terms*
  - *Let w be the function that maps a given d to its bag of tokens from K (its bag-of-words)*
  - *Let v_d by a vector of size $|K|$ for d (or a query q) with*
    - $v_d[i]=0$ iff $k_i \notin w(d)$
    - $v_d[i]=1$ iff $k_i \in w(d)$
  - *Often, we use weights instead of a Boolean membership function*
    - *Let $w_{ij} \geq 0$ be the weight of term k_i in document d_j ($w_{ij}=v_j[i]$)*
    - $w_{ij}=0$ if $k_i \notin d_j$

Content of this Lecture

- IR Models
- Boolean Model
- Vector Space Model
- Relevance Feedback in the VSM
- Probabilistic Model
- Latent Semantic Indexing
- Other IR Models

Boolean Model

- Simple model based on set theory
- Queries are specified as Boolean expressions
  - Token are atoms
  - Atoms are connected by AND, OR, NOT, (XOR, ...)
  - Parenthesis are possible (but ignored here)
- Relevance of a document is either 0 or 1
  - Let q contain the atoms <k_1, k_2, ...>
  - An atom k_i evaluates to true for a document d iff $v_d[k_i]=1$
  - Compute values of all atoms for each d
  - Compute value of q for d as logical expression over atom values
- No weights, no ranking

Properties

- Simple, clear semantics, widely used in (early) systems
- Disadvantages
  - No partial matching
    - Suppose query $\mathbf{k_1 \wedge k_2 \wedge \ldots \wedge k_9}$
    - A doc d with $\mathbf{k_1 \wedge k_2 \ldots k_8}$ is as irrelevant as one with none of the terms
  - No ranking
  - Token cannot be weighted
    - But some are more important for a doc than others
  - Average users don’t like (understand) Boolean expressions
- Results: Often unsatisfactory
  - Too many documents (too few restrictions, many OR)
  - Too few documents (too many restrictions, many AND)
- Several extensions exist, but generally not satisfactory

A Note on Implementation

- One should not iterate over D, but use a term index
  - Assume we have an index with fast operation find: $K \rightarrow P^D$
  - Search each atom k_i of the query, resulting in a set $D_i \subseteq D$
  - Evaluate query in the given order using set operations on D_i's
    - $k_i \wedge k_j$: $D_i \cap D_j$
    - $k_i \vee k_j$: $D_i \cup D_j$
    - NOT k_i: $D \setminus D_i$
- Improvements: Cost-based evaluation
  - Evaluate sub-expressions first that result in smaller intermediate results
  - Less memory requirements, faster intersections, ...

Negation in the Boolean Model

- Evaluating "**NOT** k_i" can be very expensive
  - If k_i is not a stop word, result is very large: $|D \backslash D_i| \approx |D|$
    - Most other terms appear in almost no documents
    - Recall Zipf's Law – the tail of the distribution
- Solution 1: Disallow negation
  - This is what many web search engines do
- Solution 2: Allow only in the form "$k_i \wedge$ **NOT** k_j"
  - Should not use implementation scheme as given before
    - $D_{not\text{-}kj}$ would be very large
  - Better: $D := D_i \backslash D_j$

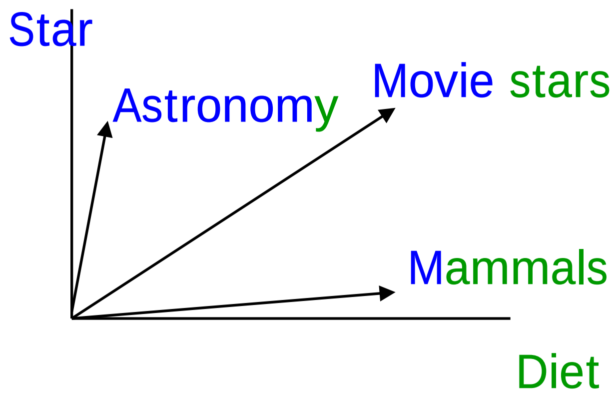
Content of this Lecture

- IR Models
- Boolean Model
- Vector Space Model
- Relevance Feedback in the VSM
- Probabilistic Model
- Latent Semantic Indexing
- Other IR Models

Vector Space Model

- Salton, G., Wong, A. and Yang, C. S. (1975). "A Vector Space Model for Automatic Indexing." *Communications of the ACM*
  - A breakthrough in IR
  - Still most popular model today
- General idea
  - Fix a vocabulary K
  - View each doc and query as a point in a |K|-dimensional space
  - Rank docs according to distance from the query in that space
- Main advantages
  - Natural ranking of docs (according to distance)
  - Naturally supports partial matching (increases distance)

Vector Space



- Each term is one dimension
  - Different suggestions for determining co-ordinates, i.e., term weights
- The closest docs are the most relevant ones
  - Rationale: Vectors correspond to themes which are loosely related to sets of terms
  - Distance between vectors ~ distance between themes
  - Different suggestions for defining distance

The Angle between Two Vectors

- Recall: The scalar product between two vectors v and w of equal dimension is defined as follows

$$v \circ w = |v| * |w| * \cos(v, w)$$

- This gives us the angle

$$\cos(v, w) = \frac{v \circ w}{|v| * |w|}$$

  - With

$$|v| = \sqrt{\sum {v_i}^2} \qquad v \circ w = \sum_{i=1..n} v_i * w_i$$

Distance as Angle

Distance = cosine of the angle between doc d and query q

$$sim(d,q) = \cos(v_d, v_q) = \frac{v_d \circ v_q}{|v_d| * |v_q|} = \frac{\sum \left(v_q[i] * v_d[i] \right)}{\sqrt{\sum v_d[i]^2} * \sqrt{\sum v_q[i]^2}}$$

Length normalization

Can be dropped for ranking

Example

- Assume stop word removal, stemming, and binary weights

	Text	verkauf	haus	italien	gart	miet	blüh	woll
1	Wir verkaufen Häuser in Italien	1	1	1				
2	Häuser mit Gärten zu vermieten		1		1	1		
3	Häuser: In Italien, um Italien, um Italien herum		1	1				
4	Die italienschen Gärtner sind im Garten			1	1			
5	Der Garten in unserem italienschen Haus blüht		1	1	1		1	
Q	Wir wollen ein Haus mit Garten in Italien mieten		1	1	1	1		1

Ranking

1	1	1	1				
2		1		1	1		
3		1	1				
4			1	1			
5		1	1	1		1	
Q		1	1	1	1		1

$$sim(d,q) = \frac{\sum \left(v_q[i] * v_d[i]\right)}{\sqrt{\sum v_d[i]^2}}$$

- $sim(d_1,q) = (1*0+1*1+1*1+0*1+0*1+0*0+0*1) / \sqrt{3}$ ~ 1.15
- $sim(d_2,q) = (1+1+1) / \sqrt{3}$ ~ 1.73
- $sim(d_3,q) = (1+1) / \sqrt{2}$ ~ 1.41
- $sim(d_4,q) = (1+1) / \sqrt{2}$ ~ 1.41
- $sim(d_5,q) = (1+1+1) / \sqrt{4}$ ~ 1.5

Rg	Q: Wir wollen ein **Haus** mit **Garten** in **Italien mieten**
1	d_2: **Häuser** mit **Gärten** zu **vermieten**
2	d_5: Der **Garten** in unserem **italienschen Haus** blüht
3	d_4: Die **italienschen Gärtner** sind im **Garten**
	d_3: **Häuser**: In **Italien**, um **Italien**, um **Italien** herum
5	d_1: Wir verkaufen **Häuser** in **Italien**

Introducing Term Weights

- Definition
  Let D be a document collection, K be the set of all terms in D, $d \in D$ and $k \in K$
  - *The term frequency tf_{dk} is the frequency of k in d*
  - *The document frequency df_k is the frequency of docs in D containing k*
    - *This should rather be called "corpus frequency"*
    - *Sometimes defined as the frequency of occurrences of k in D*
    - *Both definitions are valid and both are used*
  - *The inverse document frequency idf_k is $idf_k = |D| / df_k$*
    - *In practice, one usually uses $idf_k = \log(|D| / df_k)$*

Ranking with TF scoring

$$sim(d,q) = \frac{\sum \left(v_q[i] * v_d[i]\right)}{\sqrt{\sum v_d[i]^2}}$$

1	1	1	1				
2		1		1	1		
3		1	3				
4			1	2			
5		1	1	1		1	
Q		1	1	1	1		1

- $sim(d_1,q) = (1*0+1*1+1*1+0*1+0*1+0*0+0*1) / \sqrt{3}$ ~ 1.15
- $sim(d_2,q) = (1+1+1) / \sqrt{3}$ ~ 1.73
- $sim(d_3,q) = (1+3) / \sqrt{10}$ ~ 1.26
- $sim(d_4,q) = (1+2) / \sqrt{5}$ ~ 1.34
- $sim(d_5,q) = (1+1+1) / \sqrt{4}$ ~ 1.5

Rg	Q: Wir wollen ein **Haus** mit **Garten** in **Italien mieten**
1	d_2: **Häuser** mit **Gärten** zu **vermieten**
2	d_5: Der **Garten** in unserem **italienschen Haus** blüht
3	d_4: Die **italienschen Gärtner** sind im **Garten**
4	d_3: **Häuser**: In **Italien**, um **Italien**, um **Italien** herum
5	d_1: Wir verkaufen **Häuser** in **Italien**

Alternative Scoring: TF*IDF

- 1^{st} problem: The longer a doc, the higher the probability of finding query terms by pure chance
  - Solution: Normalize TF values on document length (yields $0 \leq w_{dk} \leq 1$)

$$tf'_{dk} = \frac{tf_{dk}}{|d|} = \frac{tf_{dk}}{\sum_{j=1..k} tf_{dj}}$$

  - Note: Longer docs also get down-ranked by normalization on doc-length in similarity function. Use only one measure!
- 2^{nd} problem: Terms frequent in D don't help to discriminate and should be scored less
  - Solution: Also use IDF scores

$$w_{dk} = \frac{tf_{dk}}{|d|} * idf_k$$

Example TF*IDF

IDF	5	5/4	5/4	5/3	5	5	DIV-0
1 (tf)	1/3	1/3	1/3				
2 (tf)		1/3		1/3	1/3		
3 (tf)		1/4	3/4				
4 (tf)			1/3	2/3			
5 (tf)		1/4	1/4	1/4		1/4	
Q		1	1	1	1		1

$$w_{dk} = \frac{tf_{dk}}{|d_d|} * idf_k = \frac{tf_{dk}}{|d_d|} * \frac{|D|}{df_k}$$

$$sim(d,q) = \frac{\sum \left(v_q[i] * v_d[i]\right)}{\sqrt{\sum v_d[i]^2}}$$

- $sim(d_1,q) = (5/4*1/3 + 5/4*1/3) / \sqrt{0.3}$ ~ 1.51
- $sim(d_2,q) = (5/4*1/3 + 5/3*1/3 + 5*1/3) / \sqrt{0.3}$ ~ 4,80
- $sim(d_3,q) = (5/4*1/4 + 5/4*3/4) / \sqrt{0.63}$ ~ 1,57
- $sim(d_4,q) = (5/4*1/3 + 5/3*2/3) / \sqrt{0.56}$ ~ 2,08
- $sim(d_5,q) = (5/4*1/4 + 5/4*1/4 + 5/3*1/4) / \sqrt{0.25}$ ~ 2,08

wollen ein **Haus** mit **Garten** in **Italien mieten**	wollen ein **Haus** mit **Garten** in **Italien mieten**
d_2: **Häuser** mit **Gärten** zu **vermieten**	**Häuser** mit **Gärten** zu **vermieten**
d_5: Der **Garten** in unserem **italienschen Haus** blüht	Der **Garten** in unserem **italienschen Haus** blüht Die **italienschen Gärtner** sind im **Garten**
d_4: Die **italienschen Gärtner** sind im **Garten**	
d_3: **Häuser**: In **Italien**, um **Italien**, um **Italien** herum	**Häuser**: In **Italien**, um **Italien**, um **Italien** herum
d_1: Wir verkaufen **Häuser** in **Italien**	Wir verkaufen **Häuser** in **Italien**

TF*IDF in Short

- Give query terms in a doc d high weights which are (1) frequent in d and (2) infrequent in D
- IDF deals with the consequences of Zipf's law
  - The few very frequent (and unspecific) terms get lower scores
  - The many infrequent (and specific) terms get higher scores
- Interferes with stop word removal
  - If stop words are removed, IDF might not be necessary any more
  - If IDF is used, stop word removal might not be necessary any more
- Many variations: log? Smoothing?

A Concrete (and Popular) VSM-Model

- Okapi BM25
  - Okapi: First system which used it (80ties)
  - BM25: Best-Match, version 25 (roughly)
- Good results in several TREC evaluations

$$sim(d,q) = \sum_{k \in q} IDF(k) * \frac{tf_{dk} * (k_1 + 1)}{tf_{dk} + k_1 * \left(1 - b + b * \frac{|d|}{a}\right)}; \quad IDF(k) = \frac{|D| - tf_k + 0.5}{tf_k + 0.5}$$

  - k_1, b constants (often b=0.75, k_1=0.2)
  - a is the average document length in D

Distance Measure

- Why not use Euclidean distance?

- Length of vectors would be much more important
- Since queries usually are very short, very short documents would always win
- Cosine measures normalizes by the length of both vectors

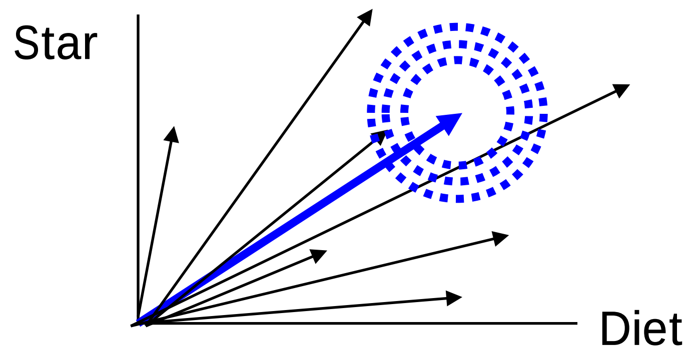
Shortcomings

- We assume that all terms are independent
  - Clearly wrong: some terms are semantically closer than others
    - Their co-appearance doesn't mean more than only one appearance
    - The appearance of "red" in a doc with "wine" doesn't mean much
  - Extension: Topic-based Vector Space Model (LSI - later)
- No treatment of synonyms (query expansion, ...)
- No treatment of homonyms
  - Different senses = different dimensions
  - We would need to disambiguate terms into their senses (later)
- Term-order independent
  - But order carries semantic meaning (object? subject?)

A Note on Implementation

- Assume we want to retrieve the top-r docs
  - Look up all terms k_i of the query in an index
  - Build the union of all documents which contain at least one keyword from query
    - Hold in a list sorted by score (initialize with 0)
  - Walk through terms k_i in order of decreasing IDF-weights
    - Go through docs in order of current score
    - For each document d_j: Add w_{ji}*IDF_i to current score s_j
  - Report top-r documents
- Several tricks to speed up search at the cost of accuracy
  - But we are anyway only computing approximation of relevance

A Different View



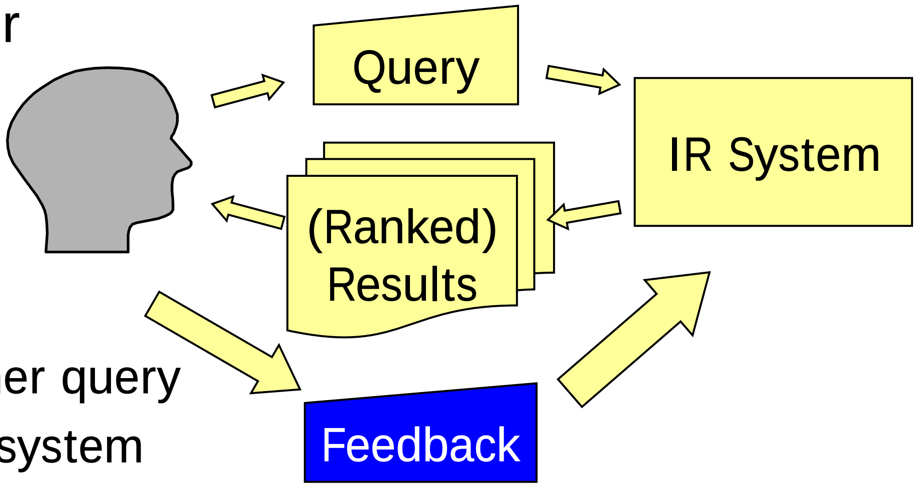
- Query evaluation actually searches for the top-r nearest neighbors (for some similarity measure)
- Can be achieved using multidimensional indexing
  - kDD-Trees, Grid files, etc.
  - No sequential scan of (all, many) docs
- But: keyword search is faster

Content of this Lecture

- IR Models
- Boolean Model
- Vector Space Model
- Relevance Feedback in the VSM
- Probabilistic Model
- Latent Semantic Indexing

Interactive IR

- Recall: IR is a process, not a query
- Relevance feedback
  - User poses query
  - System computes ranked answer
  - User judges the relevance of the (top-k) results
  - System generates new (improved) ranked answers
    - User never needs to pose another query
    - New query is generated by the system
  - Loop until satisfaction



Relevance Feedback

- Basic assumptions
  - Relevant docs are similar to each other – the common theme should be emphasized
  - Irrelevant docs are different from relevant docs – the differences should be de-emphasized
- "Emphasize, de-emphasize" – Modify terms and weights
  - Query expansion: Add new terms to the query
    - From the relevant documents
    - More aggressive: add "NOT" with terms from irrelevant docs
  - Term re-weighting: Assign new weights to terms
    - Up-weight terms from the relevant docs
    - Down-weight terms from the irrelevant docs

Rocchio Algorithm

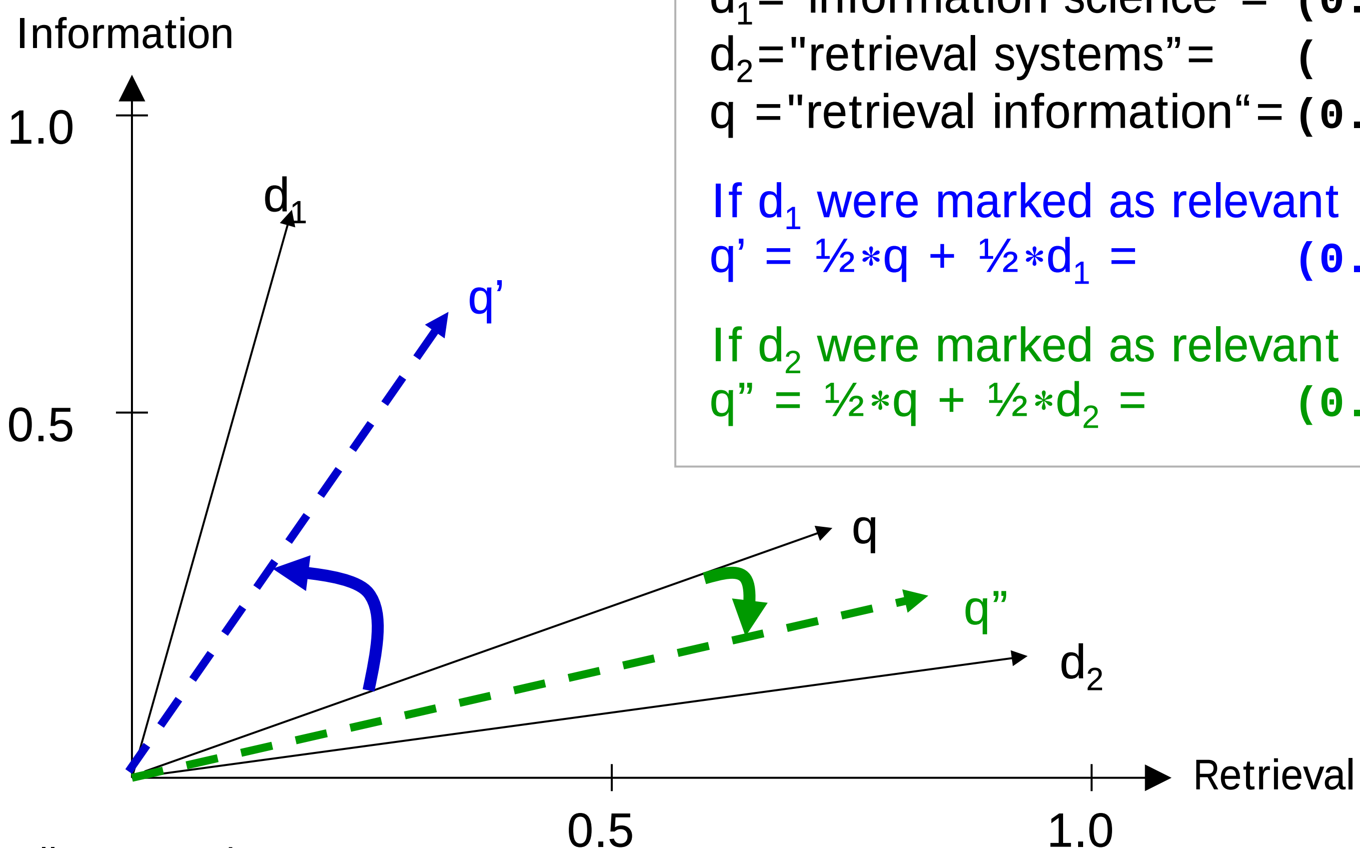
- Let R (N) be the set of docs marked as relevant (irrelevant) by the user
- Do not forget the original query
- Rocchio: Adapt query vector after each feedback

$$v_{q_{new}} = \alpha * v_q \; + \; \beta * \frac{1}{|R|} \sum_{d \in R} v_d \; - \; \gamma * \frac{1}{|N|} \sum_{d \in N} v_d$$

  - Implicitly performs query expansion and term re-weighting
  - Rocchio, J., Relevance Feedback in Information Retrieval,. In J. Rocchio and G. Salton (ed): „The SMART Retrieval System“, Prentice Hall, 1971

Example

Let α=0.5, β=0.5, γ=0, K={information, science, retrieval, system}



d_1="information science"= **(0.8,0.4, 0, 0)**
d_2="retrieval systems"= **(0, 0,0.8,0.2)**
q ="retrieval information"= **(0.4, 0,0.8, 0)**

If d_1 were marked as relevant
q' = ½∗q + ½∗d_1 = **(0.6,0.2,0.4, 0)**

If d_2 were marked as relevant
q" = ½∗q + ½∗d_2 = **(0.2, 0,0.8,0.1)**

Quelle: A. Nürnberger: IR

Choices for N

- How can we determine N?
  - Naïve: N = D\R
    - Infeasible: R not known, N too large and unknown
  - Ask the user for explicit negative feedback
    - More work for the user
  - Implicit: Docs presented for assessment and marked relevant
    - Assumes that user looked at all
- Generally: Large N make things slow
  - Query after first round has ~|K| dimensions with non-null values
- R has a theme, N probably very heterogeneous
  - High likelihood that terms get weights reflecting only the corpus, not the "not in R" property

Variations

- How to choose α, β, γ?
  - Tuning with gold standard sets – difficult
  - Educated guess, user study
- Alternative treatment for N
  - Intuition: Non-relevant docs are heterogeneous and tear in every direction – better to only take the worst instead of all of them

$$v_{q_{new}} = \alpha * v_q \; + \; \beta * \frac{1}{|R|} \sum_{d \in R} v_d \; - \; \gamma * \{ v_d \mid d = \arg\min_{d \in N} (sim(v_q, v_d)) \}$$

  - But: Probably many documents with similarity 0 – which to take?
  - Engines are tuned to find most relevant docs – inefficient
- Probably most popular: Ignore N

Effects of Relevance Feedback

- Advantages
  - Improves results (many studies) compared to single queries
  - Comfortable: Users need not generate new queries themselves
  - Iterative process converging to the best possible answer
  - Especially helpful for increasing recall
    - Due to query expansion – kind-of synonym expansion
- Disadvantages
  - Still requires some work by the user
    - Excite: Only 4% used relevance feedback (“more of this” button)
  - Writing a new query based on returned results might be faster (and easier and more successful) than rating results
  - Assumes that relevant docs are similar
    - What if user searches for all meanings of “jaguar”?
  - Query very long already after one iteration – slow retrieval

Collaborative Filtering

- More inputs for improving IR performance
- Collaborative filtering: Return to the user what other yet similar users liked
  - "Customers who bought this book also bought ..."
  - In IR: Find users posing similar queries and look at what they did with the answers
    - In e-Commerce: Which produces did they buy? (very reliable)
    - In IR, we need to approximate
      - Documents a user clicked on (if known)
      - Did the user look at the second page? (Low credit for first results)
      - Did the user pose a "refinement query" next?
      - ...
    - All these measures are not very reliable; we need many users

Thesaurus-based Query Expansion [M07, CS276]

- Expand query with synonyms and hyponyms of each term
  - feline → feline cat
  - One may weight added terms less than original query terms
- Often used in scientific IR systems (Medline)
- Requires high quality thesaurus
- General observation
  - Increases recall
  - May significantly decrease precision
    - “interest rate” → “interest rate fascinate evaluate”
  - Do synonyms really exist?

Self Assessment

- Explain the vector space model
- How is the size of K (vocabulary) influenced by pre-processing?
- Describe some variations of deducing term weights
- How could we extend the VSM to also consider the order of terms (to a certain degree)?
- How does the Rocchio algorithm determine the next query after feedback?
- How can we determine a useful set of negative documents in relevance feedback?
- How does relevance feedback work in current search engines?