Text Analytics
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 of a **given set of (normalized) documents is relevant** for a given query?
• Ranking: **How relevant** is each document of a given set for a given query?
How can Relevance be Judged?

**User Task**
- Retrieval: Adhoc Filtering
- Browsing

**Retrieval:**
- Adhoc
- Filtering

**Structured Models**
- Non-Overlapping Lists
- Proximal Nodes

**Classic Models**
- Boolean
- Vector-Space
- Probabilistic

**Set Theoretic**
- Fuzzy
- Extended Boolean

**Algebraic**
- Generalized Vector
- Lat. Semantic Index
- Neural Networks

**Probabilistic**
- Inference Network
- Belief Network

**Browsing**
- Flat
- Structure Guided
- Hypertext

[BYRN99]
Notation

• All of the models we discuss use the “Bag of Words” view
• Definition
  - Let $K$ be the set of all terms, $k \in K$ is a term
  - Let $D$ be the set of all documents, $d \in D$ is a document
  - Let $w$ be the function that maps a document to its set of distinct terms in an arbitrary, yet fixed order (bag-of-words)
    • After pre-processing: Stemming, stop-word removal etc.
  - Let $v_d$ by a vector for $d$ (or a query $q$) with
    • $v_d[i] = 0$ if the $i$’th term $k_i$ in $w(D)$ is not contained in $w(d)$
    • $v_d[i] = 1$ if the $i$’th term $k_i$ in $w(D)$ is contained in $w(d)$
  - Often, we shall use weights instead of 0/1 memberships
    • 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**
  – Terms are atoms
  – Terms are connected by AND, OR, NOT, (XOR, ...)
• No weights; all terms contribute equally to relevance
• Relevance of a document is either 0 or 1
  – Compute all atoms (keywords) of the query \( q = \langle k_1, k_2, \ldots \rangle \)
  – An atom \( k_i \) evaluates to true on \( d \) iff \( v_d[k_i] = 1 \)
  – Compute values of all atoms for each \( d \)
  – Compute value of \( q \) as **logical expression** over these values
Properties

• Simple, clear semantics, widely used in (early) systems

• Disadvantages
  - No partial matching
    • Suppose query $k_1 \land k_2 \land \ldots \land k_9$
    • A doc $d$ with $k_1 \land k_2 \ldots k_8$ is as irrelevant as one with none of the terms
  - No ranking
  - Query terms cannot be weighted
  - Average users don’t like (understand) Boolean expressions

• Results: Often unsatisfactory
  - Too many documents (too few restrictions)
  - Too few documents (too many restrictions)

• Several extensions exist, but generally not satisfactory
A Note on Implementation

• Implementation search $k_i$’s in $D$ and do not iterative over $D$

• Searching is supported by an index
  – Assume we have an index with fast operation $\text{find: } K \rightarrow P^D$
  – Search each $k_i$ of the query, resulting in a set $D_i \subseteq D$
  – Evaluate query in the given order using set operations on $D_i$’s
    • $k_i \land k_j : D_i \cap D_j$
    • $k_i \lor k_j : D_i \cup D_j$
    • NOT $k_i : D \setminus D_i$
    • Parenthesis treated in the usual way

• Improvements: Cost-based evaluation
  – Evaluate sub-expressions first (hopefully) resulting in small results
  – Smaller intermediate results, faster intersections, …
Negation in the Boolean Model

• Evaluating "\textbf{NOT} k_i" is expensive
  - Result often is very, very large ($|D \setminus D_i| \approx |D|$)
    • If $k_i$ is not a stop word (which we removed anyway)
    • 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 "$k_i \land \textbf{NOT} k_j$"
  - Cannot use previous implementation scheme ($D_{\text{not-kj}}$ would be very large)
  - Better: $D := D_i \setminus D_j$
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

  - 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**
  - Ability to rank docs (according to distance)
  - Naturally supports partial matching
Vector Space

- Each term is one dimension
  - Different suggestions for determining co-ordinates
    - Weights

- The closest docs are the most relevant ones
  - Rational: vectors correspond to themes which are loosely related to sets of terms
  - Distance between vector ~ distance between themes
  - Different suggestions for defining distance
Computing the Angle between Two Vectors ??? Wie ist das mit dem Einheitsvektor?

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

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

- This gives us the angle

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

- With

$$|v| = \sqrt{\sum v_i^2} \quad v \circ w = \sum_{i=1..n} v_i \times w_i$$
Distance as Angle

Distance = \textit{cosine of the angle} between doc d and query q

\[\text{sim}(d, q) = \cos(v_d, v_q) = \frac{v_d \circ v_q}{|v_d| \times |v_q|} = \frac{\sum (v_q[i] \times v_d[i])}{\sqrt{\sum v_d[i]^2} \times \sqrt{\sum v_q[i]^2}}\]

- Length normalization
- Can be dropped for ranking
Example Data

- Assume stop word removal, stemming, and **binary weights**

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

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

- \( sim(d_1, q) = \frac{1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1}{\sqrt{3}} \approx 1.15 \)
- \( sim(d_2, q) = \frac{1 + 1 + 1}{\sqrt{3}} \approx 1.73 \)
- \( sim(d_3, q) = \frac{1 + 1}{\sqrt{2}} \approx 1.41 \)
- \( sim(d_4, q) = \frac{1 + 1}{\sqrt{2}} \approx 1.41 \)
- \( sim(d_5, q) = \frac{1 + 1 + 1}{\sqrt{4}} \approx 1.5 \)

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 italienischen Haus blüht
3   | \( d_4: \) Die italienischen 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 number of docs in $D$ containing $k$
  
  • This should rather be called "corpus frequency"
  • Often differently defined as the number of occurrences of $k$ in $D$
  • Both definitions are valid and both are used

- The inverse document frequency is defined as $idf_k = |D| / df_k$
  
  • This should rather be called "inverse relative document frequency"
  • In practice, one usually uses $idf_k = \log(|D| / df_k)$
Ranking with TF scoring

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

- \(\sim 1.15\)  
- \(\sim 1.73\)  
- \(\sim 1.26\)  
- \(\sim 1.34\)  
- \(\sim 1.5\)

Q: Wir wollen ein Haus mit Garten in Italien mieten

<table>
<thead>
<tr>
<th>Rg</th>
<th>Q: Wir wollen ein Haus mit Garten in Italien mieten</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(d_2: \text{Häuser mit Gärten zu vermieten})</td>
</tr>
<tr>
<td>2</td>
<td>(d_5: \text{Der Garten in unserem italienschen Haus blüht})</td>
</tr>
<tr>
<td>3</td>
<td>(d_4: \text{Die italienschen Gärtner sind im Garten})</td>
</tr>
<tr>
<td>4</td>
<td>(d_3: \text{Häuser: In Italien, um Italien, um Italien herum})</td>
</tr>
<tr>
<td>5</td>
<td>(d_1: \text{Wir verkaufen Häuser in Italien})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Improved Scoring: TF*IDF

• One obvious problem: The longer a document, the more non-null and the higher are the tf values, and thus the higher the chances of being ranked high
  – Solution: Normalize on document length (also yields 0 ≤ wij ≤ 1)
    \[
    tf'_{dk} = \frac{tf_{dk}}{|d|} = \frac{tf_{dk}}{\sum_{k=1..|d|} tf_{dk}}
    \]

• Second problem: Some terms are everywhere in D, don’t help to discriminate, and should be scored less
  – Solution: Use IDF scores
    \[
    w_{ij} = \frac{tf_{ij}}{|d_i|} \ast idf_j
    \]
### Example TF*IDF

$$w_{ij} = \frac{tf_{ij}}{|d_i|} \cdot idf_j = \frac{tf_{ij}}{|d_i|} \cdot \frac{|D|}{df_k}$$

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

- sim(d1,q) = $\frac{(5/4 \times 1/3 + 5/4 \times 1/3)}{\sqrt{0.3}}$ ~ 1.51
- sim(d2,q) = $\frac{(5/4 \times 1/3 + 5/3 \times 1/3 + 5 \times 1/3)}{\sqrt{0.3}}$ ~ 4.80
- sim(d3,q) = $\frac{(5/4 \times 1/4 + 5/4 \times 3/4)}{\sqrt{0.63}}$ ~ 1.57
- sim(d4,q) = $\frac{(5/4 \times 1/3 + 5/3 \times 2/3)}{\sqrt{0.56}}$ ~ 2.08
- sim(d5,q) = $\frac{(5/4 \times 1/4 + 5/4 \times 1/4 + 5/3 \times 1/4)}{\sqrt{0.25}}$ ~ 2.08

<table>
<thead>
<tr>
<th>IDF</th>
<th>5</th>
<th>5/4</th>
<th>5/4</th>
<th>5/3</th>
<th>5</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (tf)</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 (tf)</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 (tf)</td>
<td>1/4</td>
<td>3/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (tf)</td>
<td>1/3</td>
<td>2/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (tf)</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>wollen ein <strong>Haus</strong> mit <strong>Garten</strong> in <strong>Italien</strong> mieten</th>
<th>wollen ein <strong>Haus</strong> mit <strong>Garten</strong> in <strong>Italien</strong> mieten</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>d2</strong>: <strong>Häuser</strong> mit <strong>Gärten</strong> zu <strong>vermieten</strong></td>
<td><strong>Häuser</strong> mit <strong>Gärten</strong> zu <strong>vermieten</strong></td>
</tr>
<tr>
<td><strong>d5</strong>: Der <strong>Garten</strong> in unserem <strong>italienschen Haus</strong> blüht</td>
<td>Der <strong>Garten</strong> in unserem <strong>italienschen Haus</strong> blüht</td>
</tr>
<tr>
<td><strong>d4</strong>: Die <strong>italienschen Gärtner</strong> sind im <strong>Garten</strong></td>
<td>Die <strong>italienschen Gärtner</strong> sind im <strong>Garten</strong></td>
</tr>
<tr>
<td><strong>d3</strong>: <strong>Häuser</strong>: <strong>In Italien</strong>, um <strong>Italien</strong>, um <strong>Italien</strong> herum</td>
<td><strong>Wir verkaufen Häuser</strong> in <strong>Italien</strong></td>
</tr>
<tr>
<td><strong>d1</strong>: <strong>Wir verkaufen Häuser</strong> in <strong>Italien</strong></td>
<td><strong>Häuser</strong>: <strong>In Italien</strong>, um <strong>Italien</strong>, um <strong>Italien</strong> herum</td>
</tr>
</tbody>
</table>
TF*IDF in Short

- Give terms in a doc $d$ **high weights** which are …
  - frequent in $d$ and
  - 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
Further Improvements

• **Scoring the query** in the same way as the documents
  - Note: Repeating words in a query then makes a difference
• Empirically estimated “best” scoring function [BYRN99]
  - Different length normalization

\[
\text{tf}''_{ij} = \frac{tf_{ij}}{\max_j tf_{ij}}
\]

- Use log of IDF with slightly different measure; rare terms are not totally distorting the score any more

\[
w_{ij} = \frac{tf_{ij}}{\max_j tf_{ij}} \cdot \log \left( \frac{|D|}{df_j} \right)
\]
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**, i.e., that their vectors are orthogonal
  - Clearly wrong: some terms are **semantically closer** than others
    • The appearance of “red” in a doc with “wine” doesn’t mean much
    • But “wine” is an important match for “drink”, “red” not
  - Extension: Topic-based Vector Space Model (LSI)

• No treatment of **synonyms** (query expansion, …)

• No treatment of **homonyms**
  - Different senses = different dimensions
  - We would need to disambiguate words into their senses (later)

• Term-order independent
  - But order carries semantic meaning (object? subject?)
A first 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 $k_i$
    - 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 $w_{ji} \times \text{IDF}_i$ to current score $s_j$
      - Early stop
        - Look at $s_j^r - s_j^{r+1}$: Can doc with rank $r+1$ still reach doc with rank $r$?
        - Can be estimated from distribution of TF and IDF values
  - Early stop might produce the wrong order of top-r docs, but cannot produce wrong docs
    - Why not?
A Different View

- Query evaluation actually searches for the top-r nearest neighbors (for some similarity measure)
- Can be achieved using multidimensional indexing
  - kD-D Trees, Grid files, etc.
  - Hope: No sequential scan of (all, many) docs, but directed search according to distances
- Severe problem: High dimensionality
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**

\[
sim(d, q) = \sum_{k \in q} \text{IDF}(k) \cdot \frac{tf_{dk} \cdot (k_1 + 1)}{tf_{dk} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{a}\right)};
\]

\[
\text{IDF}(k) = \frac{|D| - tf_k + 0.5}{tf_k + 0.5}
\]

- \(k_1, b\) constants (often \(b=0.75, k_1=0.2\))
- \(a\) is the average document length in \(D\)
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 answers
  - User judges the relevance of the top-k returned docs
  - System considers feedback and generates new (improved) ranked answers
    • User never needs to pose another query
    • The new query is generated by the system
  - Loop until satisfaction
Relevance Feedback

• Basic assumptions
  – Relevant docs are somewhat similar to each other – the **common core** should be emphasized
  – Irrelevant docs are different from relevant docs – the differences should be deemphasized

• System may **adapt to feedback** by
  – 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
    • Overweight terms the relevant documents, down-weight other terms
Formal Approach

• Assume we know the set R of all relevant docs, and N=D\R
• How would a good query for R look like?

\[ v_q = \frac{1}{|R|} \sum_{d \in R} v_d - \frac{1}{|N|} \sum_{d \in N} v_d \]

- Weight terms high that are in R
- Weight terms low that are not in R, i.e., that are in N

• Remark
  - It is not clear whether this is the “best” of all queries
    • Relevance feedback is a heuristic
    • Especially when only a sample of R is known or the corpus changes
Rocchio Algorithm

- Usually we do not know the real R
- Best bet: Let R (N) be the set of docs marked as relevant (irrelevant) by the user
- Still: Do not forget the original query
- Rocchio: Adapt query vector after each feedback

\[ v_{q_{new}} = \alpha \cdot v_q + \beta \cdot \frac{1}{|R|} \sum_{d \in R} v_d - \gamma \cdot \frac{1}{|N|} \sum_{d \in N} v_d \]

- Implicitly performs query expansion and term re-weighting
Example

Let $\alpha=0.5$, $\beta=0.5$, $\gamma=0$, $K=\{\text{information, science, retrieval, system}\}$

$\begin{align*}
    d_1 &= \text{"information science"} = (0.2, 0.8, 0, 0) \\
    d_2 &= \text{"retrieval systems"} = (0, 0, 0.8, 0.2) \\
    q &= \text{"retrieval of inf..."} = (0.6, 0, 0.2, 0)
\end{align*}$

If $d_1$ were marked relevant
$q' = \frac{1}{2}q + \frac{1}{2}d_1 = (0.4, 0.4, 0.1, 0)$

If $d_2$ were marked relevant
$q'' = \frac{1}{2}q + \frac{1}{2}d_2 = (0.3, 0, 0.5, 0.1)$
Choices for N

- How can we determine N?
  - Naïve: \( N = D \setminus R \)
    - N very large
  - Let the user explicitly define it by negative feedback (or not)
    - More work for the user; Might by a by-product of looking at promising, yet non relevant docs
  - Chose those presented for assessment and not rated R
    - Implicitly assumes that user looked at all

- Generally: Large N make things slow
  - Query after first round has \( \sim |K| \) dimensions with non-null values
  - Computing the novel score very slow due to third term
Variations

- How to choose $\alpha$, $\beta$, $\gamma$?
  - Tuning with gold standard sets – difficult
  - Educated guess

- 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
  - Also faster due to much shorter query vectors

\[ v_{q_{new}} = \alpha \cdot v_q + \beta \cdot \frac{1}{|R|} \sum_{d \in R} v_d - \gamma \cdot \{v_d \mid d = \arg \min_{d \in N}(\text{sim}(v_q, v_d)) \} \]
Effects of Relevance Feedback

• Advantages
  - Improved 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)

• 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 classifying results
  - Based on the assumption 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**: Give 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 recordable)
      - 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 $\rightarrow$ 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” $\rightarrow$ “interest rate fascinate evaluate”
  - Do synonyms really exist?