Query Languages

This material was prepared by Diana Inkpen, University of Ottawa, 2005, updated 2021. Some of these slides were originally prepared by Raymond Mooney, University of Texas Austin.

Boolean Queries

- Keywords combined with Boolean operators:
  - OR: (e_1 OR e_2)
  - AND: (e_1 AND e_2)
  - BUT: (e_1 BUT e_2) Satisfy e_1 but **not** e_2
- Negation only allowed using BUT to allow efficient use of inverted index by filtering another efficiently retrievable set.
- Naïve users have trouble with Boolean logic.

Boolean Retrieval with Inverted Indices

- Primitive keyword: Retrieve containing documents using the inverted index.
- OR: Recursively retrieve e_1 and e_2 and take union of results.
- AND: Recursively retrieve e_1 and e_2 and take intersection of results.
- BUT: Recursively retrieve e_1 and e_2 and take set difference of results.

“Natural Language” Queries

- Full text queries as arbitrary strings.
- Typically just treated as a bag-of-words for a vector-space model.
- Typically processed using standard vector-space retrieval methods.

Phrasal Queries

- Retrieve documents with a specific phrase (ordered list of contiguous words)
  - "information theory"
- May allow intervening stop words and/or stemming.
  - "buy camera" matches:
    "buy a camera"
    "buying the cameras"
    etc.

Phrasal Retrieval with Inverted Indices

- Must have an inverted index that also stores *positions* of each keyword in a document.
- Retrieve documents and positions for each individual word, intersect documents, and then finally check for ordered contiguity of keyword positions.
- Best to start contiguity check with the least common word in the phrase.

Phrasal Search

Find set of documents D in which all keywords ($k_1 \ldots k_m$) in phrase occur (using AND query processing).

Intitialize empty set, R, of retrieved documents.

For each document, d, in D:

- Get array, P_i ,of positions of occurrences for each k_i in d
- Find shortest array P_s of the P_i's
- For each position p of keyword k_s in P_s
  - For each keyword k_i except k_s
    - Use binary search to find a position ($p - s + i$) in the array P_i
  - If correct position for every keyword found, add d to R

Return R

Proximity Queries

- List of words with specific maximal distance constraints between terms.
- Example: “dogs” and “race” within 4 words match “…dogs will begin the race…”
- May also perform stemming and/or not count stop words.

Proximity Retrieval with Inverted Index

- Use approach similar to phrasal search to find documents in which all keywords are found in a context that satisfies the proximity constraints.
- During binary search for positions of remaining keywords, find closest position of k_i to p and check that it is within maximum allowed distance.

Pattern Matching

- Allow queries that match strings rather than word tokens.
- Requires more sophisticated data structures and algorithms than inverted indices to retrieve efficiently.

Simple Patterns

- Prefixes: Pattern that matches start of word.
  - "anti" matches "antiquity", "antibody", etc.
- Suffixes: Pattern that matches end of word:
  - "ix" matches "fix", "matrix", etc.
- Substrings: Pattern that matches arbitrary subsequence of characters.
  - "rapt" matches "enrapture", "velociraptor" etc.
- Ranges: Pair of strings that matches any word lexicographically (alphabetically) between them.
  - "tin" to "tix" matches "tip", "tire", "title", etc.

Allowing Errors

- What if query or document contains typos or misspellings?
- Judge similarity of words (or arbitrary strings) using:
  - Edit distance (Levenstein distance)
  - Longest Common Subsequence (LCS)
- Allow proximity search with bound on string similarity.

Edit (Levenstein) Distance

- Minimum number of character *deletions*, *additions*, or *replacements* needed to make two strings equivalent.
  - "misspell" to "mispell" is distance 1
  - "misspell" to "mistell" is distance 2
  - "misspell" to "misspelling" is distance 3
- Can be computed efficiently using *dynamic programming* in O(*mn*) time where *m* and *n* are the lengths of the two strings being compared.

Longest Common Subsequence (LCS)

- Length of the longest subsequence of characters shared by two strings.
- A *subsequence* of a string is obtained by deleting zero or more characters.
- Examples:
  - "misspell" to "mispell" is 7
  - "misspelled" to "misinterpretted" is 7 "mis…p…e…ed"

Regular Expressions

- Language for composing complex patterns from simpler ones.
  - An individual character is a regex.
  - Union: If e_1 and e_2 are regexes, then $(e_1 \mid e_2)$ is a regex that matches whatever either e_1 or e_2 matches.
  - Concatenation: If e_1 and e_2 are regexes, then $e_1\ e_2$ is a regex that matches a string that consists of a substring that matches e_1 immediately followed by a substring that matches e_2
  - Repetition (Kleene closure): If e_1 is a regex, then e_1* is a regex that matches a sequence of zero or more strings that match e_1

Regular Expression Examples

- (u|e)nabl(e|ing) matches
  - unable
  - unabling
  - enable
  - enabling
- (un|en)*able matches
  - able
  - unable
  - unenable
  - enununenable

Enhanced Regex's (Perl)

- Special terms for common sets of characters, such as alphabetic or numeric or general "wildcard".
- Special repetition operator (+) for 1 or more occurrences.
- Special optional operator (?) for 0 or 1 occurrences.
- Special repetition operator for specific range of number of occurrences: {min,max}.
  - A{1,5} One to five A's.
  - A{5,} Five or more A's
  - A{5} Exactly five A's

Perl Regex's

- Character classes:
  - \w (word char) Any alpha-numeric (not: \W)
  - \d (digit char) Any digit (not: \D)
  - \s (space char) Any whitespace (not: \S)
  - . (wildcard) Anything
- Anchor points:
  - \b (boundary) Word boundary
  - ^ Beginning of string
  - $ End of string

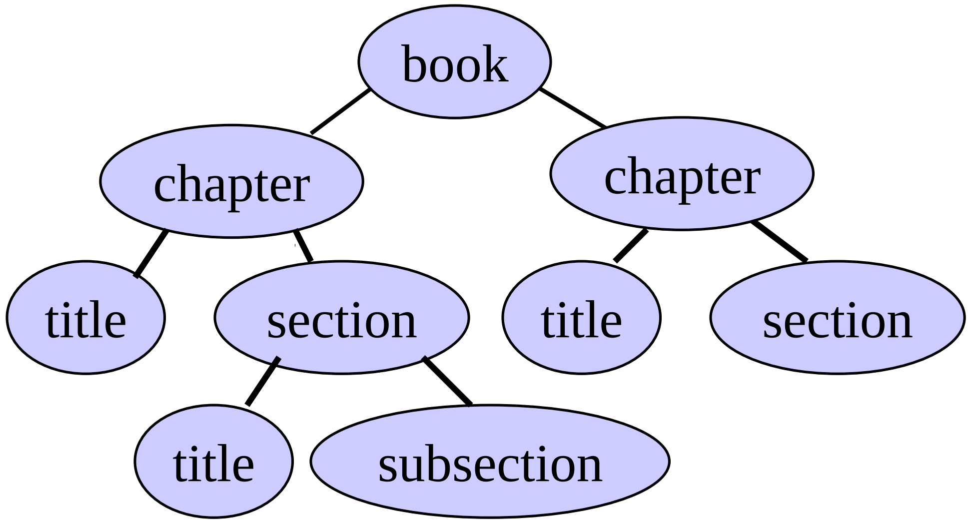
Perl Regex Examples

- U.S. phone number with optional area code:
  - /\b(\(\d{3}\)\s?)?\d{3}-\d{4}\b/
- Email address:
  - /\b\S+@\S+(\.com|\.edu|\.gov|\.org|\.net)\b/

Note: Packages available to support Perl regex's in Java

Structural Queries

- Assumes documents have structure that can be exploited in search.
- Structure could be:
  - Fixed set of fields, e.g. title, author, abstract, etc.
  - Hierarchical (recursive) tree structure:



Queries with Structure

- Allow queries for text appearing in specific fields:
  - “nuclear fusion” appearing in a chapter title
- SFQL: Relational database query language SQL enhanced with “full text” search.
  - Select abstract from journal.papers where
    author contains “Teller” and
    title contains “nuclear fusion” and
    date < 1/1/1950

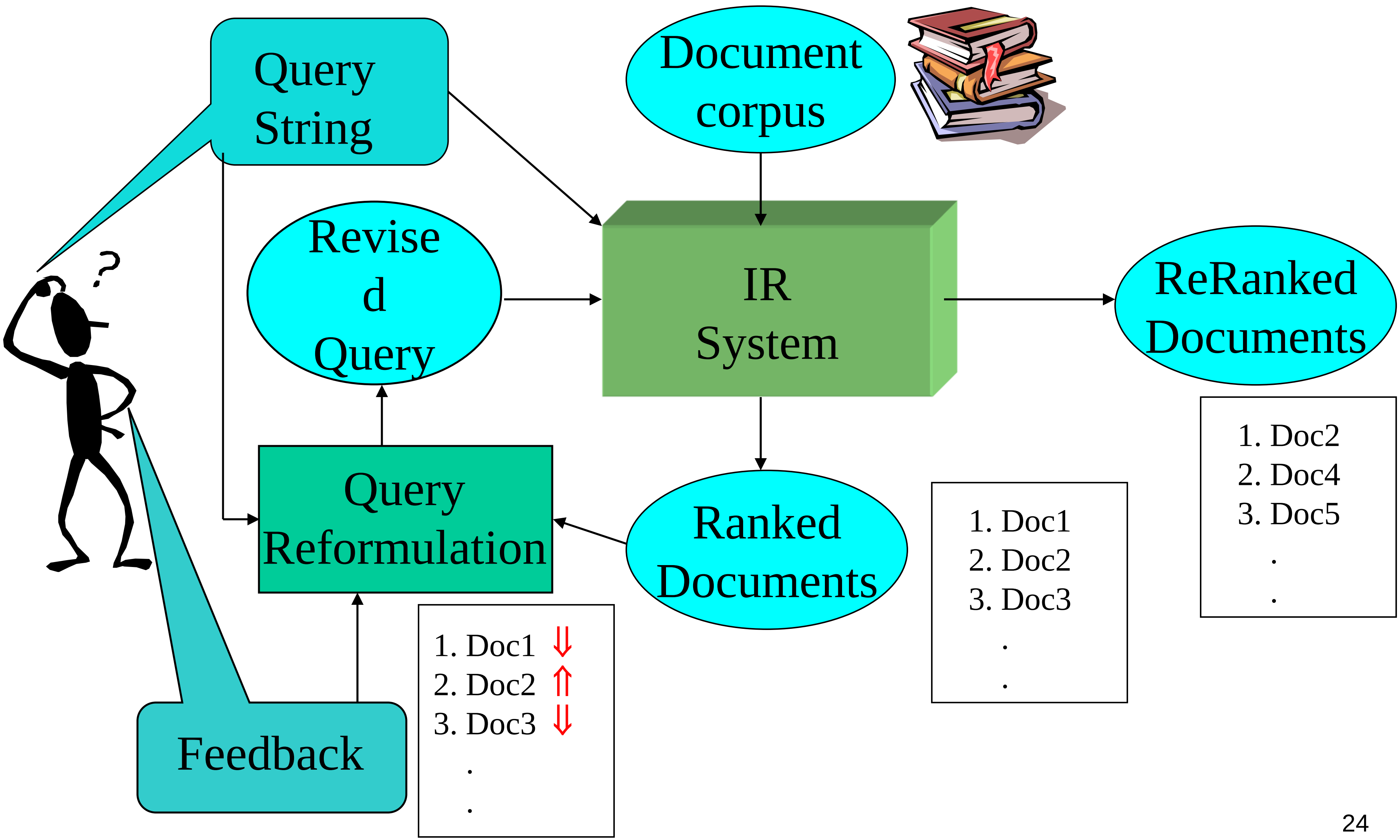
Query Operations

Relevance Feedback &
Query Expansion

Relevance Feedback

- After initial retrieval results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents.
- Use this feedback information to reformulate the query.
- Produce new results based on reformulated query.
- Allows more interactive, multi-pass process.

Relevance Feedback Architecture



Query Reformulation

- Revise query to account for feedback:
  - Query Expansion: Add new terms to query from relevant documents.
  - Term Reweighting: Increase weight of terms in relevant documents and decrease weight of terms in irrelevant documents.
- Several algorithms for query reformulation.

Query Reformulation for VSR

- Change query vector using vector algebra.
- **Add** the vectors for the **relevant** documents to the query vector.
- **Subtract** the vectors for the **irrelevant** docs from the query vector.
- This adds both positive and negative weighted terms to the query, as well as reweighting the initial terms.

Optimal Query

- Assume that the relevant set of documents C_r are known.
- Then the best query that ranks all and only the relevant queries at the top is:

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j$$

Where N is the total number of documents.

Standard Rochio Method

- Since all relevant documents unknown, just use the **known** relevant (D_r) and irrelevant (D_n) sets of documents and include the initial query q.

$$\vec{q}_m = \alpha \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

α: Tunable weight for initial query.

β: Tunable weight for relevant documents.

γ: Tunable weight for irrelevant documents.

Ide Regular Method

- Since more feedback should perhaps increase the degree of reformulation, do not normalize for amount of feedback:

$$\vec{q}_m = \alpha \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

α: Tunable weight for initial query.
β: Tunable weight for relevant documents.
γ: Tunable weight for irrelevant documents.

Ide "Dec Hi" Method

- Bias towards rejecting **just** the highest ranked of the irrelevant documents:

$$\vec{q}_m = \alpha\vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \max_{non-relevant}(\vec{d}_j)$$

α: Tunable weight for initial query.
β: Tunable weight for relevant documents.
γ: Tunable weight for irrelevant document.

Comparison of Methods

- Overall, experimental results indicate no clear preference for any one of the specific methods.
- All methods generally improve retrieval performance (recall & precision) with feedback.
- Generally just let tunable constants equal 1.

Evaluating Relevance Feedback

- By construction, reformulated query will rank explicitly-marked relevant documents higher and explicitly-marked irrelevant documents lower.
- Method should not get credit for improvement on *these* documents, since it was told their relevance.
- In machine learning, this error is called "testing on the training data."
- Evaluation should focus on generalizing to **other** un-rated documents.

Fair Evaluation of Relevance Feedback

- Remove from the corpus any documents for which feedback was provided.
- Measure recall/precision performance on the remaining *residual collection*.
- Compared to complete corpus, specific recall/precision numbers may decrease since relevant documents were removed.
- However, **relative** performance on the residual collection provides fair data on the effectiveness of relevance feedback.

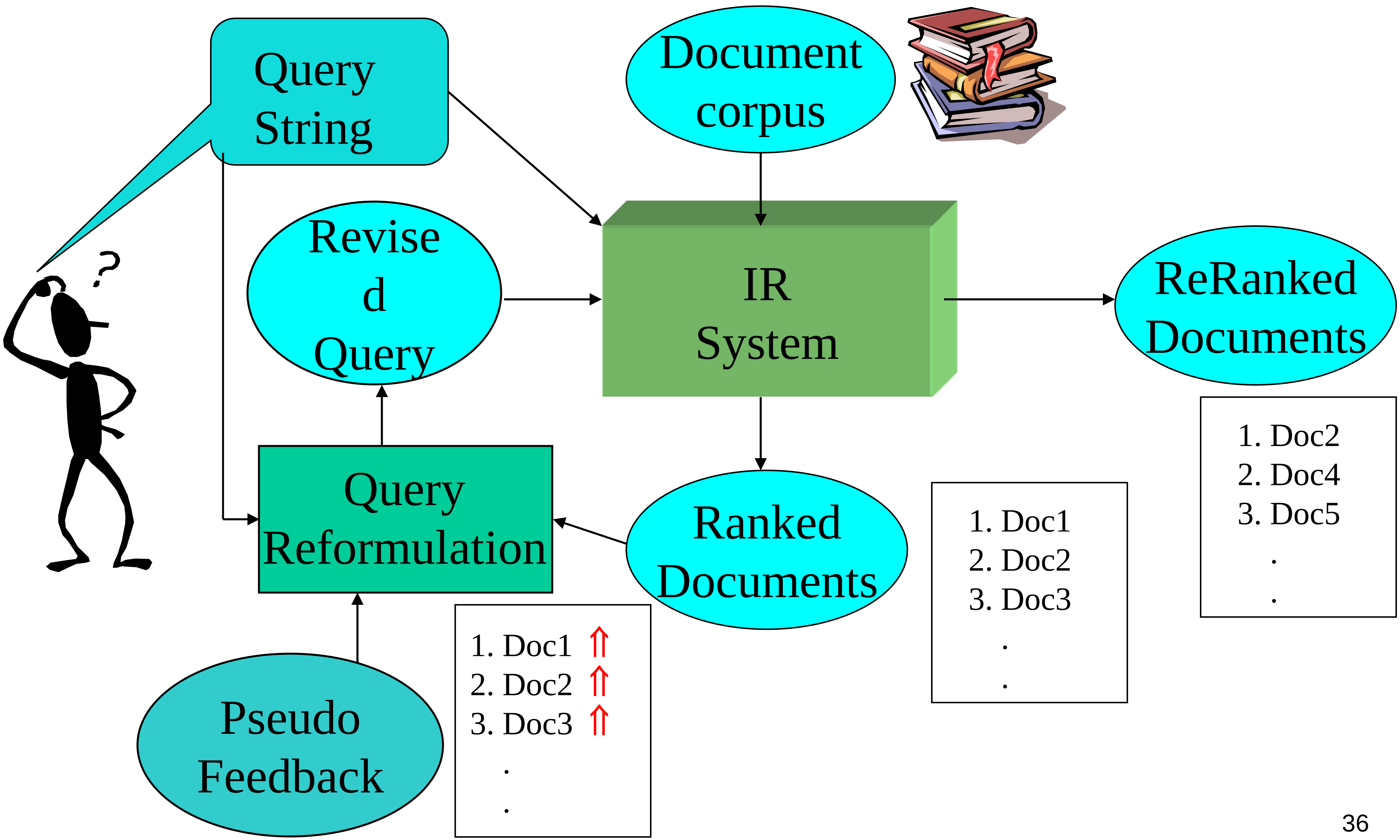
Why is Feedback Not Widely Used

- Users sometimes reluctant to provide explicit feedback.
- Results in long queries that require more computation to retrieve, and search engines process lots of queries and allow little time for each one.
- Makes it harder to understand why a particular document was retrieved.

Pseudo Feedback

- Use relevance feedback methods without explicit user input.
- Just **assume** the top *m* retrieved documents are relevant, and use them to reformulate the query.
- Allows for query expansion that includes terms that are correlated with the query terms.

Pseudo Feedback Architecture



PseudoFeedback Results

- Found to improve performance on TREC competition ad-hoc retrieval task.
- Works even better if top documents must also satisfy additional boolean constraints in order to be used in feedback.

Thesaurus

- A thesaurus provides information on synonyms and semantically related words and phrases.
- Example:

```
physician
   syn: ||croaker, doc, doctor, MD,
medical, mediciner, medico, ||sawbones
   rel: medic, general practitioner,
surgeon
```

Thesaurus-based Query Expansion

- For each term, *t*, in a query, expand the query with synonyms and related words of *t* from the thesaurus.
- May weight added terms less than original query terms.
- Generally increases recall.
- May significantly decrease precision, particularly with ambiguous terms.
  - "interest rate" → "interest rate fascinate evaluate"

WordNet

- A more detailed database of semantic relationships between English words.
- Developed by famous cognitive psychologist George Miller and a team at Princeton University.
- About 152,059 English words.
- Nouns, adjectives, verbs, and adverbs grouped into about 115,424 synonym sets called *synsets*.

WordNet Synset Relationships

- Antonym: front → back
- Attribute: benevolence → good (noun to adjective)
- Pertainym: alphabetical → alphabet (adjective to noun)
- Similar: unquestioning → absolute
- Cause: kill → die
- Entailment: breathe → inhale
- Holonym: chapter → text (part-of)
- Meronym: computer → cpu (whole-of)
- Hyponym: tree → plant (specialization)
- Hypernym: fruit → apple (generalization)

WordNet Query Expansion

- Add synonyms in the same synset.
- Add hyponyms to add specialized terms.
- Add hypernyms to generalize a query.
- Add other related terms to expand query.

Statistical Thesaurus

- Existing human-developed thesauri are not easily available in all languages.
- Human thesauri are limited in the type and range of synonymy and semantic relations they represent.
- Semantically related terms can be discovered from statistical analysis of corpora.

Automatic Global Analysis

- Determine term similarity through a pre-computed statistical analysis of the complete corpus.
- Compute association matrices which quantify term correlations in terms of how frequently they co-occur.
- Expand queries with statistically most similar terms.

Association Matrix

$$
\begin{array}{c|cccc|}
 & w_1 & w_2 & w_3 \ldots\ldots\ldots\ldots & w_n \\
\hline
w_1 & c_{11} & c_{12} & c_{13} \ldots\ldots\ldots\ldots & c_{1n} \\
w_2 & c_{21} & & & \\
w_3 & c_{31} & & & \\
. & . & & & \\
. & . & & & \\
w_n & c_{n1} & & & \\
\hline
\end{array}
$$

c_{ij}: Correlation factor between term *i* and term *j*

$$c_{ij} = \sum_{d_k \in D} f_{ik} \times f_{jk}$$

f_{ik} : Frequency of term *i* in document *k*

Normalized Association Matrix

- Frequency based correlation factor favors more frequent terms.
- Normalize association scores:

$$s_{ij} = \frac{c_{ij}}{c_{ii} + c_{jj} - c_{ij}}$$

- Normalized score is 1 if two terms have the same frequency in all documents.

Metric Correlation Matrix

- Association correlation does not account for the proximity of terms in documents, just co-occurrence frequencies within documents.
- Metric correlations account for term proximity.

$$c_{ij} = \sum_{k_u \in V_i} \sum_{k_v \in V_j} \frac{1}{r(k_u, k_v)}$$

V_i: Set of all occurrences of term i in any document.
$r(k_u,k_v)$: Distance in words between word occurrences k_u and k_v
(∞ if k_u and k_v are occurrences in different documents).

Normalized Metric Correlation Matrix

- Normalize scores to account for term frequencies:

$$s_{ij} = \frac{c_{ij}}{|V_i| \times |V_j|}$$

Query Expansion with Correlation Matrix

- For each term i in query, expand query with the n terms, j, with the highest value of c_{ij} (s_{ij}).
- This adds semantically related terms in the "neighborhood" of the query terms.

Problems with Global Analysis

- Term ambiguity may introduce irrelevant statistically correlated terms.
  - "Apple computer" $\rightarrow$ "Apple red fruit computer"
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

Automatic Local Analysis

- At query time, dynamically determine similar terms based on analysis of top-ranked retrieved documents.
- Base correlation analysis on only the "local" set of retrieved documents for a specific query.
- Avoids ambiguity by determining similar (correlated) terms only within relevant documents.
  - "Apple computer" → "Apple computer Powerbook laptop"

Global vs. Local Analysis

- Global analysis requires intensive term correlation computation only once at system development time.
- Local analysis requires intensive term correlation computation for every query at run time (although number of terms and documents is less than in global analysis).
- But local analysis gives better results.

Global Analysis Refinements

- Only expand query with terms that are similar to *all* terms in the query.

$$sim(k_i, Q) = \sum_{k_j \in Q} c_{ij}$$

  - "fruit" not added to "Apple computer" since it is far from "computer."
  - "fruit" added to "apple pie" since "fruit" close to both "apple" and "pie."
- Use more sophisticated term weights (instead of just frequency) when computing term correlations.

Query Expansion Conclusions

- Expansion of queries with related terms can improve performance, particularly recall.
- However, must select similar terms very carefully to avoid problems, such as loss of precision.