Human Language Technology: Applications to Information Access

Lesson 3
Beyond IR: Finding without searching

October 13, 2016

EPFL Doctoral Course EE-724
Andrei Popescu-Belis
Idiap Research Institute, Martigny

Issues

- Trend in information retrieval: *find more relevant documents or "nuggets" with less information*

- How to improve on the ad-hoc retrieval problem?
  - in Lesson 2: query expansion, relevance feedback
  - today: re-ranking results with ML models

- Can we entirely remove the need for queries?
  - recommender systems
  - just-in-time query-free retrieval

Plan of Lesson 3

- Learning-to-rank for IR
  - typical methods
- Recommender systems
  - content-based / collaborative filtering / hybrid
  - evaluation of recommender systems
- Just-in-time retrieval (= query-free = implicit queries)
  - main concept and some prototypes
  - example: the Idiap Automatic Content Linking Device
- Practical work: use a Java text editing frame to design a query-free just-in-time recommender

Learning to rank
for information retrieval

Learning to rank: principles

- Ranked retrieval (as opposed to Boolean model) seen in Lesson 2
  - ranked models: vector space or probabilistic (e.g. Okapi BM25)
  - no machine learning apart from optimizing the parameters
- **Learning to rank** using machine learning methods
  - train a scoring function over well-ranked retrieval results
    - in order to improve the ranking of future results
  - practical use in IR systems
    1. Get *k*-best results using an IR model, or Web search;
    2. Re-rank them using a scorer.
- Global vs. local scoring functions than can be learned
  - global: generate a sorted list of documents given a query and an unordered set of documents
  - local: generates a score for each document and query, for sorting
  - rank aggregation: merge several rankings (without looking at query)

Learning to rank: data and metrics

- Data = many exemplars of (query, ((doc. features), score))
  - features are obtained using, e.g., a standard IR model
  - scores are from an ordered set
    - either obtained explicitly from human annotators
    - or by using implicit click through data from (many) users
- Metrics (used when training or testing): compare sorted lists
  - mean average precision (MAP)
  - (normalized) discounted cumulative gain (DCG or NDCG)
  - mean reciprocal rank (MRR)
  - winners take all (WTA)
  - Kendall's *Tau*

Example: discounted cumulative gain (DCG)

- For a query q and a document set $D_q = \{d_1, \ldots, d_i, \ldots, d_n\}$
  - π is the permutation proposed by the system
    - $\pi(i)$ is the position of document d_i in the ranking
  - Y is the set of correct relevance grades of documents, $Y = \{y_1, \ldots, y_n\}$ (e.g. binary {0, 1} or on a wider scale)
- DCG: how many "good" documents are highly ranked?
  - $DCG(k,q) = \sum_{\{i \mid d_i \in D_q,\ \pi(i) \le k\}} Gain(i)\ Disc(\pi(i))$
  - higher grades count more, e.g. $Gain(i) = 2^{y_i} - 1$
  - discount lower ranks, e.g. $Disc(\pi(i)) = 1 / \log_2(1 + \pi(i))$
- $DCG(k)$ can be normalized by its maximum value, and can also be computed over the entire list of results rather than k

Features for learning to rank

- Generally not all those used for retrieval (tf-idf vectors)
  - e.g. reuse only the BM25 score + a few other parameters
- BM25 coefficient between document and query
  - but also between the query and the document title, or URL, or anchor texts toward document, etc.
  - instead of BM25 (or in addition to it): term frequency, or full query frequency, or edit distance
- For Web search: PageRank coefficient
  - but also: number of clicks observed in the search log, likelihood of spam, quality score, number of inlinks, etc.

Learning methods

- Pointwise approach: learns scores
  - transforms L2R into a classification problem
    - or regression, or ordinal classification, depending on *Y*
  - put all data together: (queries, documents, scores)
  - e.g., One-Class SVM, Prank
- Pairwise approach: also learns scores, but different loss function
  - consider pairwise training data, i.e. positive examples: $y_m > y_n$ for a pair of documents and a given query
  - use pairwise loss to learn a classification/regression model
  - e.g., Ranking SVM, RankBoost, RankNet, etc.
- Listwise approach: consider the ranked list in its entirety
  - few off-the-shelf machine learning algorithms
  - techniques: estimate permutation probabilities
  - e.g. ListNet, ListMLE, AdaRank

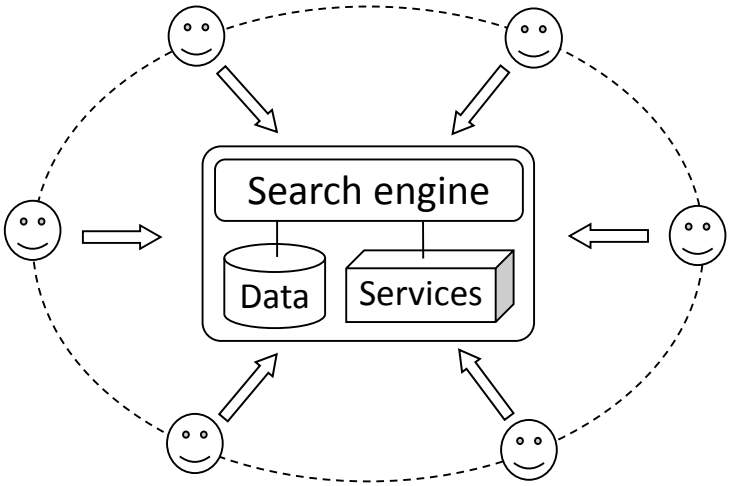
Trends and applications

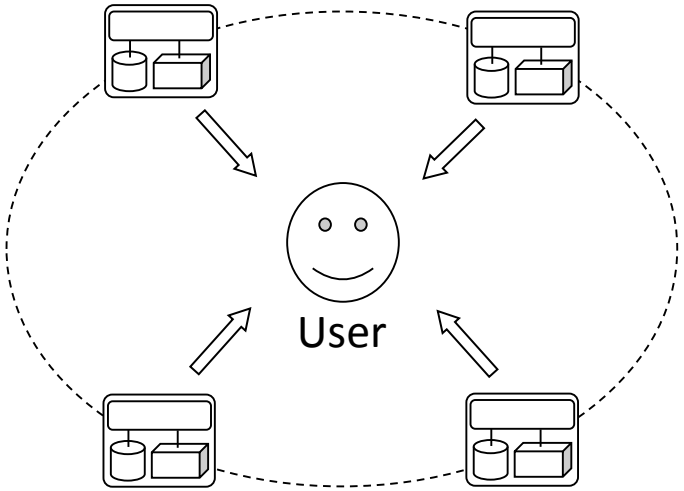
- Research in learning to rank
  - deriving “objective” data from user behavior
  - feature engineering
  - domain adaptation
  - learning: semi-supervised, active, efficient, ensemble
- Applications beyond document retrieval
  - Web search
  - results of collaborative filtering for recommender systems
  - definition or answer search
  - extractive summarization
  - and even machine translation: rank output of decoder+LM

Recommender systems

Thanks to Nikos Pappas for some slides.

From Search to Recommendation





Search *versus* Recommendation

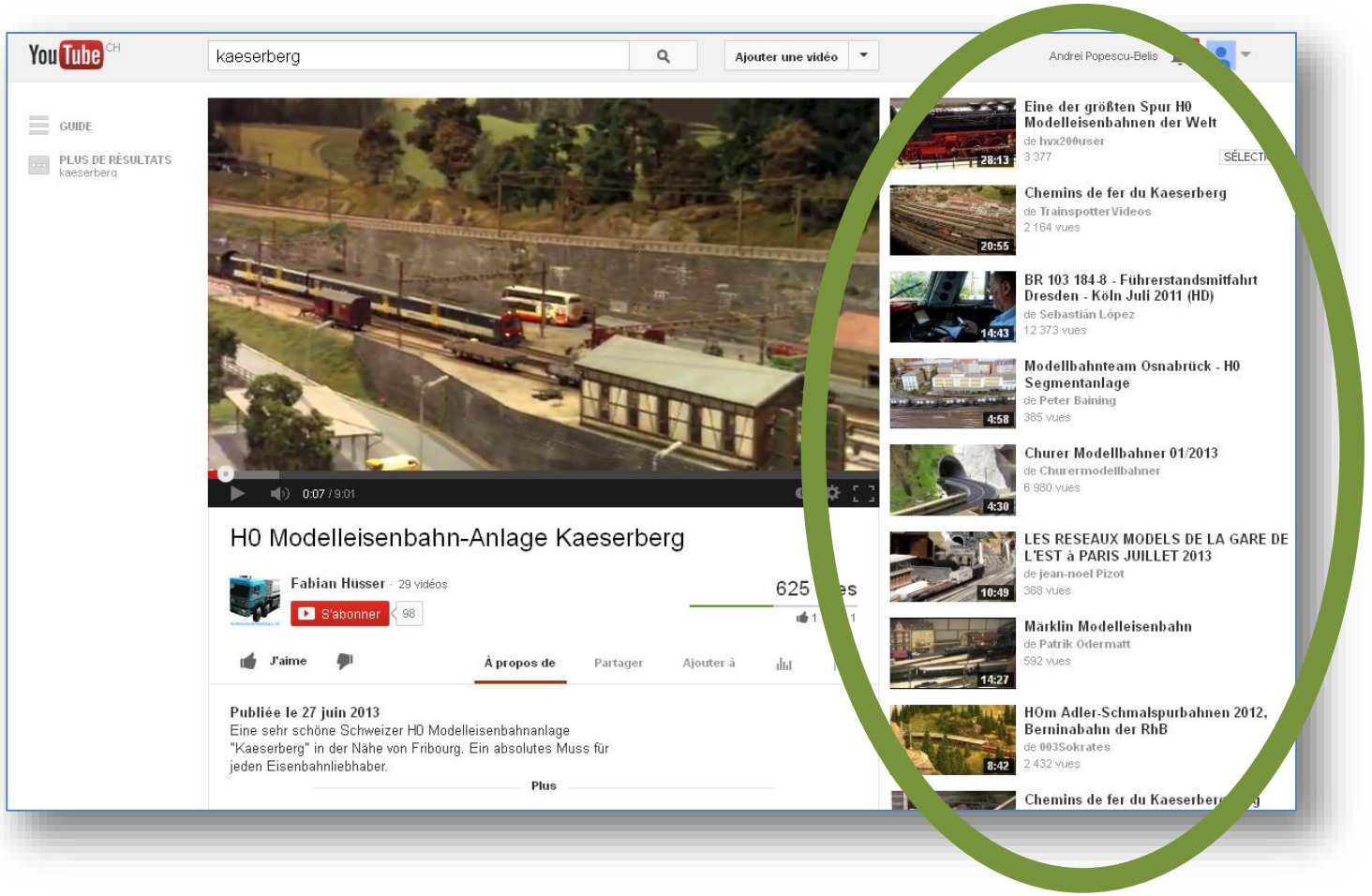
- Search: generally based on keywords or queries
- Recommendation: “finding without searching”
  - based on input from the context and on known preferences

Recommender systems

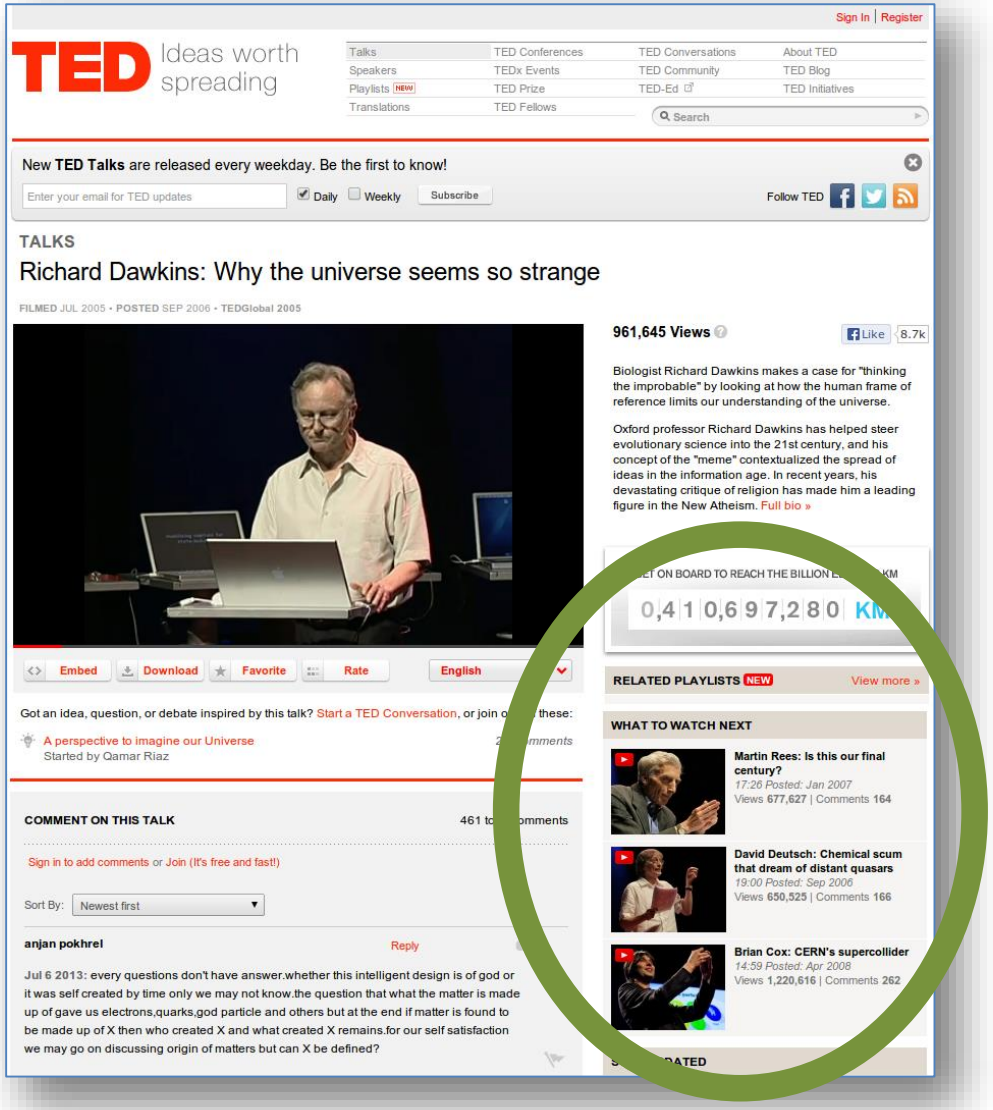
- Idea: retrieval without explicit queries (or almost)
- Goal: suggest what to buy/view/read next
  - based on user's previous behavior (expressed as **ratings**) and/or other users' previous behaviors
- Type of recommended items and examples
  - for products (Amazon), movies (Movielens, Netflix), music (Last.fm), videos (YouTube, Vimeo), friends (Facebook), colleagues (LinkedIn), etc.

→ Solution for dealing with information overload

Example: YouTube



Example: TED lectures



Metadata for recommendation is available for:

- 1,149 talks
- 69k users
- 100k favorites
- 200k comments

idiap.ch/dataset/

Definition of the task

- Given the ratings of a user for some items
  - predict the user's ratings of unseen items
  - select the top-N unseen items

	K-PAX	Life of Brian	Memento	Notorious
Alice	4	3	2	4
Bob	∅	4	5	5
Cindy	2	2	4	∅
David	3	∅	5	2

- Ratings
  - explicit: grades, likes, favorites, settings, etc.
  - implicit: views, viewing time, click history, comments, etc.

Methods for recommendation

- Human-made recommendations: costly, static
  - need for automatic and dynamic ones
- Two main automatic approaches
  - content-based (CB): use features/descriptors of items and recommend based on similarity (like in IR), i.e. recommend items similar to the ones "liked" in the past
    - what features work for multimedia events?
    - can NLP techniques help to extract them?
  - collaborative filtering (CF): recommend what other users with similar preferences have liked
    - wisdom of the crowds… when available

Content-based recommendation

- Construct item profiles by extracting features
  - attribute-based, keyword-based, etc.
  - Vector Space Models (TF-IDF, etc.)
- Construct user profiles from item history
  - in the same space
- Compute “similarity” between users and items
  - e.g. using: Rocchio algorithm (averaging user content vectors), or k-Nearest Neighbors, or Bayesian classifiers, etc.

Collaborative filtering (1/2)

- Common prediction function (user-based)
  - exploit ratings *r* of users *c'* similar to *c*

$$r_{c,s} = \underset{c' \in \hat{C}}{\text{aggr}}\, r_{c',s},$$

$$\text{(a) } r_{c,s} = \frac{1}{N} \sum_{c' \in \hat{C}} r_{c',s},$$

Similarity between users: Pearson correlation, cosine

$$\text{(b) } r_{c,s} = k \sum_{c' \in \hat{C}} sim(c, c') \times r_{c',s},$$

$$\text{(c) } r_{c,s} = \bar{r}_c + k \sum_{c' \in \hat{C}} sim(c, c') \times (r_{c',s} - \bar{r}_{c'}),$$

Normalization factor

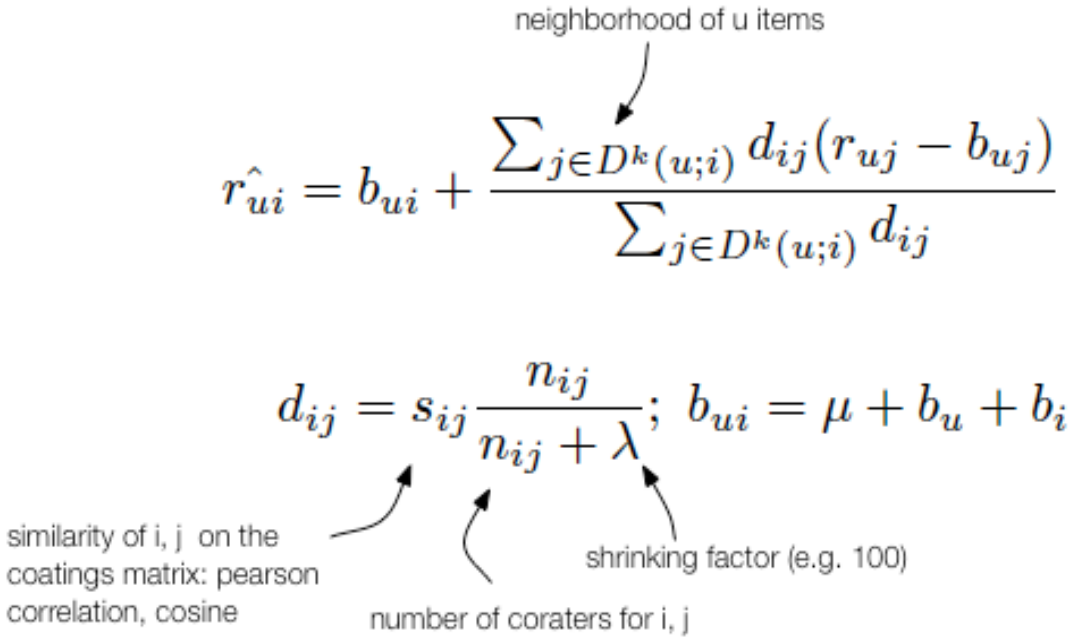
$$k = 1 / \sum_{c' \in \hat{C}} |sim(c, c')|$$

$$\bar{r}_c = (1/|S_c|) \sum_{s \in S_c} r_{c,s}, \text{ where } S_c = \{s \in S | r_{c,s} \neq \varnothing\}.$$

Average rating of user c

Collaborative filtering (2/2)

- User-item matrix (item-based approach)



- Latent factor models (matrix factorization with SVD), regression, probabilistic models, etc.

Hybrid methods and evaluation

- Hybrid methods: CB plus CF
  - combine the predictions | incorporate characteristics from one to the other | build a unifying general model with both characteristics
- Evaluation metrics for results
  - error metrics: RMSE, MSE etc.
  - classification accuracy metrics: precision-at-*n*, recall-at-*n*, F-measure-at-*n*
  - coverage and other metrics

Challenges to recommender systems

- New user problem (= cold-start), no feedback
- New item problem (sparse or no ratings) [CF]
- Sparsity of data [CF]
- Overspecialization [CB]
- Limited content analysis
  - difficulty of extracting rich metadata from items (descriptions) or ratings (reviews, comments)
  - example of research topic: analyzing content (of lectures → CB) or sentiment (of comments → CF) to improve recommendations

Just-in-time query-free retrieval (using implicit queries)

“Finding without Searching”

Review of systems, including an Idiap one

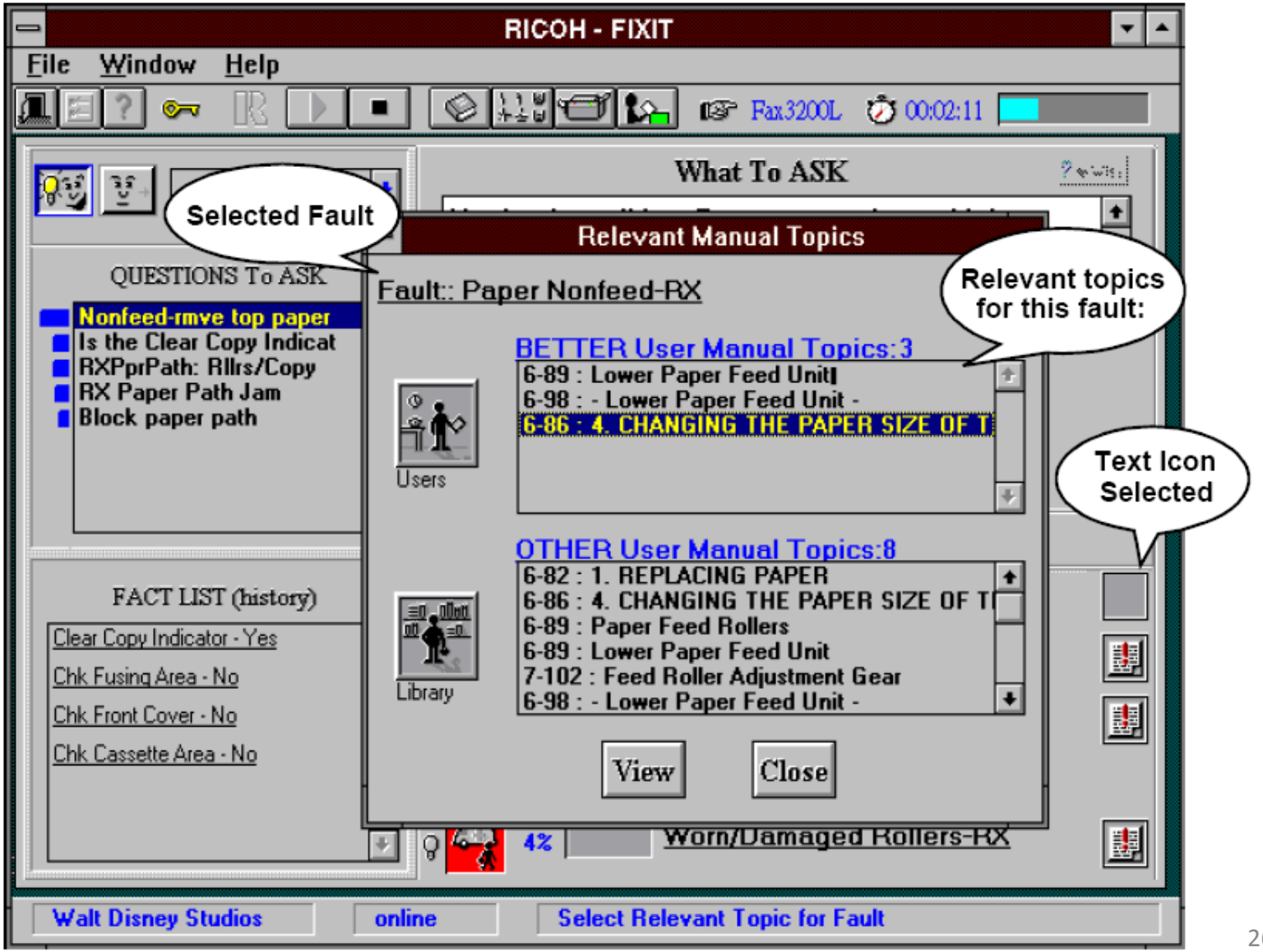
Query-free retrieval: the idea

- User has information needs
- User performs some actions in a context
- System infers needs from actions and context
  - builds a query-like representation
- System matches the two representations and returns a document set as “suggestions”
- System builds document representations
- Collection of documents contains information

Fixit (Hart and Graham 1997)

- Monitors an experts interaction with a diagnostic system for (faulty) photocopiers
  - expert provides assistance through questions/answers
  - state of dialogue expressed as positions in a belief network
- Fixit searches in a database of maintenance manuals
  - goal: provide additional support information
  - NB: search results are in fact pre-computed for each node of the belief network.
- Results: additional “topics” found in maintenance manuals, depending on the interaction state

Fixit sample screenshot



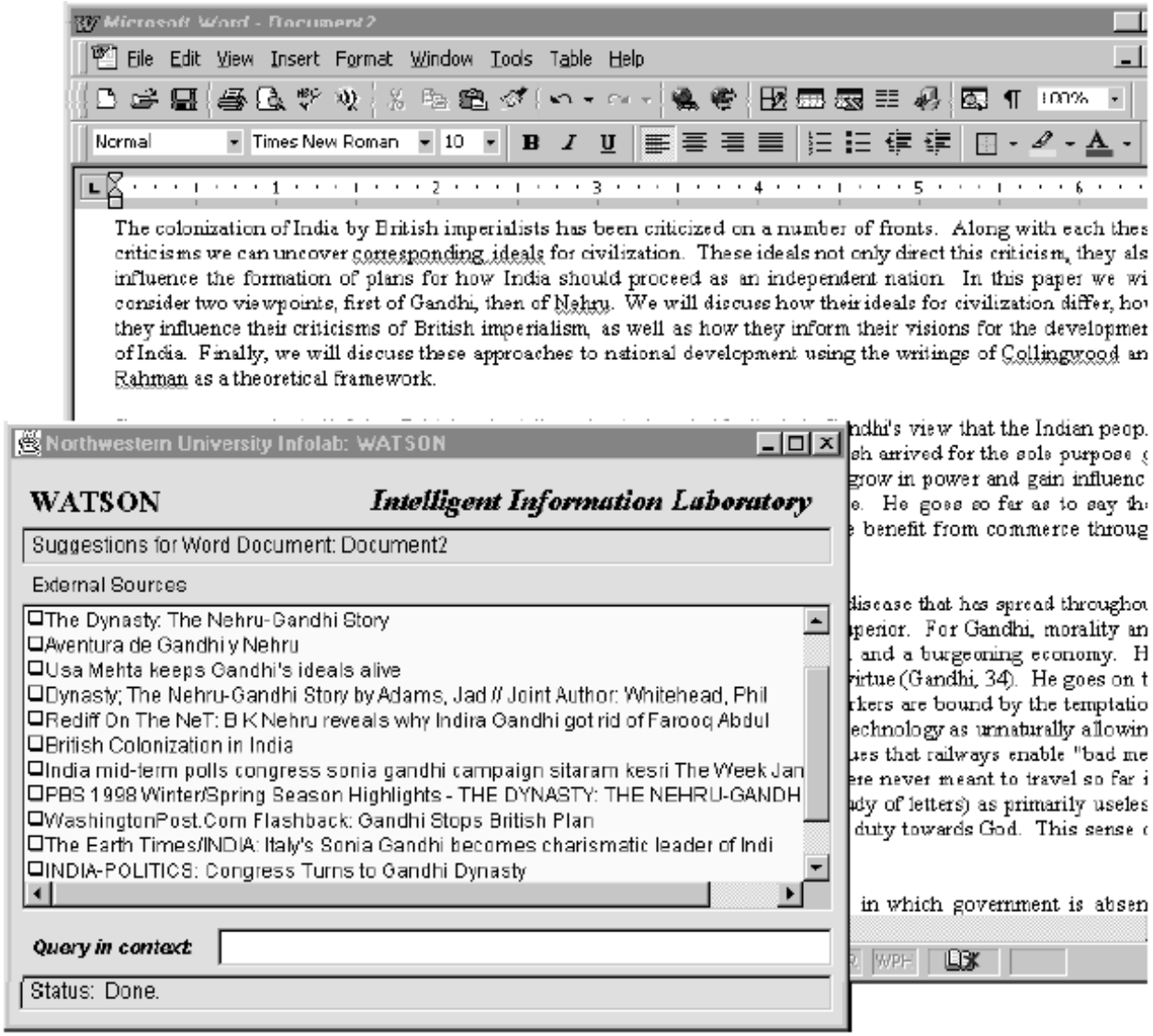
Remembrance Agent and Jimminy

- Remembrance Agent (Rhodes and Maes, 1997, 2000)
  - software integrated to the Emacs text editor
  - runs searches over emails or text notes, every few seconds
  - uses as query the latest 20-500 words typed by the user
  - results displayed in separate frame, can open with Emacs
- Jimminy: wearable assistant for taking notes and accessing information (same team)
  - contextual capture devices
    - identify the room, identify user's interlocutor (using badge)
    - ASR not available, but simulated with topic entered as note
    - testing: mostly the note-taking function
  - NB: PDAs did not exist at that time
- Screenshots are not very appealing…

Watson

- An “Information Management Assistant”
  - Budzik and Hammond 2000
- Monitors user's operations in a text editor
  - complex set of heuristics to select and weigh terms for queries → Web search engine
  - automatic queries: use context representation
    - in theory, an “anticipator” decides when to trigger them
    - also some filters for frequently-used actions
  - user-generated queries: appended to the currently built automatic one → sort of query expansion
- Additional mechanism for clustering results (Web pages) to reduce information overload

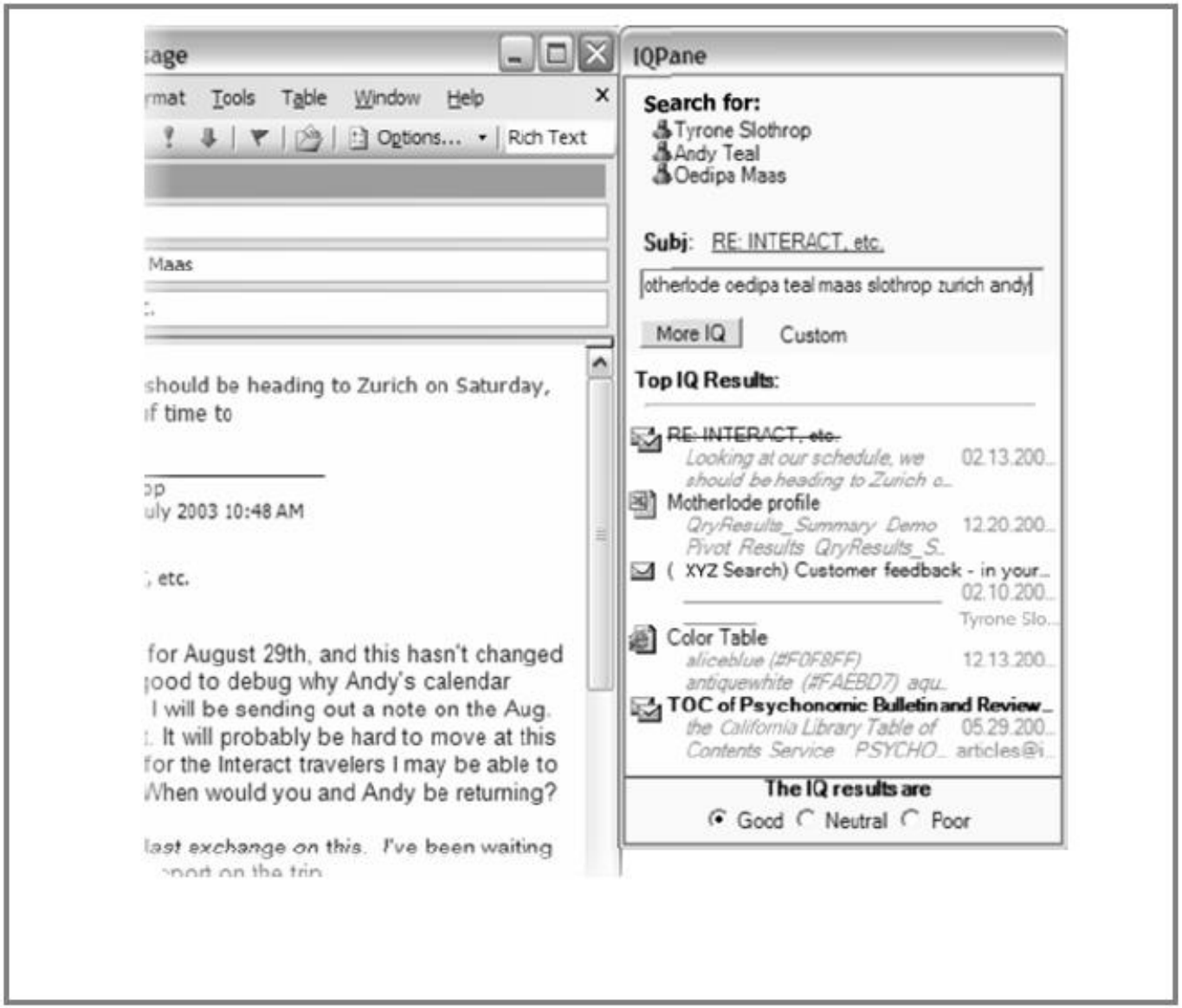
Watson (Budzik 2000)



Microsoft's Implicit Queries (IQ)
(Dumais et al. 2004; Cutrell et al. 2006)

- Generates context-sensitive searches
  - based on user's activities on their computer: reading or writing emails
  - automatically identifies words to use in a query (ranked by tf-idf)
  - runs searches in a local index: email, calendar, IM messages, files, copies of web pages
- Demoed, but never turned into a product

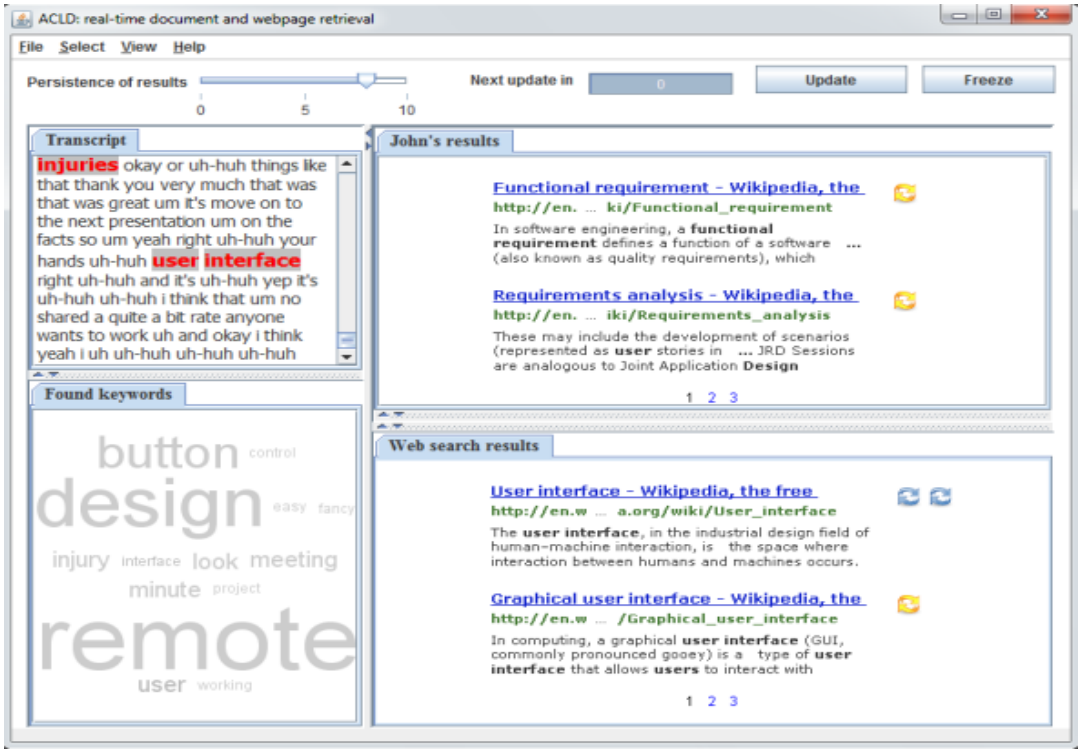
Implicit Queries



More systems

- Pro-active information systems (Billsus et al. 2005)
- Query-free system for enriching television news with articles from the Web (Henziker et al. 2003)
- A-Propos assistant for authoring environments (Puerta-Melguizo et al. 2008)
- Speech-based search: not query-free, but keyboard-free
  - Google Now, Microsoft Cortana and Apple's Siri: commercial products
  - MindMeld app.
  - FAME Interactive Space (Metze et al. 2005): use of voice commands to access past meeting recordings
  - Speech Spotter (Goto et al. 2004) and (Lyons 2004)
    - dual-purpose speech: spoken commands from normal speech flow
    - e.g. put an appointment in an agenda while conversating

Real-time recommendation during a meeting: “Automatic Content Linking”



Automatic Content Linking Device (ACLD)

- Performs real-time search in a repository of documents, based on the words pronounced during a meeting discussion
  - searches are run all the time, in the background
- Documents include reports, emails, slides, snippets of past recordings, websites
- Search results are **suggested** to participants

  → behaves like a “Virtual Secretary”

CB recommendation for ACLD

- Idea: document retrieval in meetings using implicit queries from conversations
- Stages: at regular time intervals (… or not)
  1. Extract the best keywords that cover all the main topics of the conversation with high probability
  2. Cluster keywords by topics into implicit queries
  3. Retrieve several sets of results from IR system
  4. Merge result sets by relevance and diversity

Functionalities

- Two main visual modes
  - Informative full-screen UI: all widgets shown side by side
  - Unobtrusive use: one widget displayed at the time + tabs
- Widgets can be arranged at will, and turned off/on
  - Results of automatic speech recognition (ASR)
  - Tag-cloud of recognized keywords
  - Names of suggested documents
  - Names of suggested web pages
- Behavior of links to suggested documents
  - hover over → metadata plus matching context
  - click → open with appropriate viewing program

Automatic search

- Queries
  - from words that were automatically recognized
  - emphasizing keywords, if any were detected
    - pre-defined list for a setting, updatable anytime
  - use Lucene query parser (emphasis: ^5)

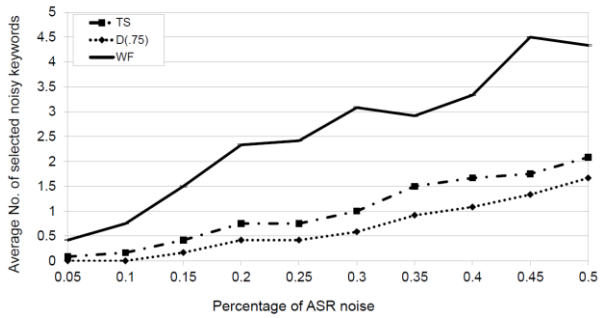
- Results
  - shown every N seconds ($10 \leq N \leq 30$) or on demand
  - results from a local repository using Lucene
  - results from the Web using Google API
  - “persistence of results” tunable by users

Diverse keyword extraction for building implicit queries (Maryam Habibi)

- For a short dialogue fragment (e.g. 30-90 s)
- Algorithm that rewards
  - representativeness of keywords
  - diversity of the keyword set
- Evaluation by human subjects (Mechanical Turk)

  New algorithm preferred over others

Corpus	Compared methods (m_1 vs. m_2)	Relevance (%)	
		m_1	m_2
Fisher	D(.75) vs. TS	68	32
	TS vs. WF	82	18
	WF vs. D(.5)	95	5
AMI	D(.75) vs. TS	78	22
	TS vs. WF	60	40
	WF vs. D(.5)	78	22



Robustness to ASR noise

Methods for evaluating the ACLD

- Feedback from audience and focus groups
  - positive, but difficult to quantify / suggestions
- Comparison of retrieval algorithms in terms of relevance assessed (offline) by users
- Evaluation in use on a task-based scenario
- Usability of the interface

Conclusions

- Several ideas to give more information to the user while asking for less input
  - cognitive overload should be considered
  - some justification of suggestions is appreciated
  - enrich IR algorithms with “contextual” information
- Many ideas, some feasibility assessment, but have yet to be turned into successful products

References

- Learning to rank
  - Hang Li, *Learning to Rank for Information Retrieval and Natural Language Processing*, Morgan & Claypool, 2011
- Recommender systems
  - G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems", *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734-749, 2005
  - A. Gunawardana and G. Shani, "A Survey of Accuracy Evaluation Metrics of Recommendation Tasks", *Journal of Machine Learning Research*, 2009
  - F. Ricci et al., *Recommender Systems Handbook*, Springer-Verlag, 2011
- Just-in-time query-free retrieval
  - Cutrell, E., Dumais, S.T., and Teevan, J., "Searching to eliminate personal information management", *Communications of the ACM,* 49(1):58–64, 2006.
  - Popescu-Belis A. et al., A Speech-based Just-in-Time Retrieval System using Semantic Search. *Proc. of ACL-HLT 2011*, *System Demonstrations,* p.80-86.
  - Habibi, M., *Modeling Users' Information Needs in a Document Recommender for Meetings*, EPFL PhD thesis n. 6760, 2015.

Practical work

A just-in-time retrieval system for text editing: coupling a text editor with an information retrieval system

Instructions: overall plan

- Look at the Lucene demo source code from last time (Lucene/contrib/demo), find out how to run searches from a Java program, using a given index, and get results
- Get Document Listener code, compile it with 'javac'
  http://docs.oracle.com/javase/tutorial/uiswing/events/documentlistener.html
- Modify the Document Listener to run spontaneous Lucene searches using the words that are typed, in a "just-in-time" manner
  - **(1) prepare good queries | (2) make results clickable | (3) estimate the relevance of the results**
- Options
  - modify and compile the demo indexer for Simple English Wikipedia pages (see Course 2 / Instructions 2), using ExtractWikipedia and WikipediaTokenizer
  - try to use Google API or another Web search engine
- **Demo + 1-page report for Fri Oct 28 → graded work (20%)**

Detailed instructions (1)

- Compiling DocumentEventDemo.java
  - create a working folder
  - inside it, create a folder called ‘events’
  - put DocumentEventDemo.java into ‘events’
  - from the working folder, run

    `javac ./events/DocumentEventDemo.java` [to compile it]

    `java events.DocumentEventDemo` [to execute it]
  - change something in the java source file, then recompile & run

- Transforming words into queries
  - in method `MyDocumentListener.updateLog(…)`
  - construct queries e.g. by checking if the last character of the update is a space, counting words, etc. – if space and already accumulated more than N words, then fire a query
    - suggestion: use `textArea.getText().substring(…, …)`

Detailed instructions (2)

- Insert code into DocumentEventDemo.java to submit a query to an existing index (e.g. the index from lesson 2)
- Look at docs/demo/index.html for the code of SearchFile.java (click on name)
  - or search the source files (if installed) for lucene-6.2.1-src/contrib/demo/src/java/org/apache/lucene/demo/SearchFiles.java
  - get inspiration from SearchFile.java to modify DocumentEventDemo.java, as follows

1. Insert the `import org.apache.lucene.*` and `import java.*` declarations
2. To avoid ambiguity on the class Document, change the line

   ```
   Document doc = (Document)e.getDocument();
   ```

   - into this line (on one line):

   ```
   javax.swing.text.Document doc =
   (javax.swing.text.Document)e.getDocument();
   ```

3. Copy the lucene-core-6.2.1.jar file into the working folder (from 'core'), and lucene-analyzers-common-6.2.1.jar (from 'analysis/common') and lucene-queryparser-6.2.1.jar (from 'queryparser')
4. Compile with (one line):

   ```
   javac -cp "lucene-core-6.2.1.jar:lucene-analyzers-common-6.2.1.jar:lucene-queryparser-6.2.1.jar" ./events/DocumentEventDemo.java
   ```

Detailed instructions (3)

5. Insert code for reading the index and running queries, copying relevant lines from SearchFiles.java

   – insert it within the code making provision for exceptions:
   ```
   try { INSTRUCTIONS } catch  (Exception ex) { WHAT TO DO };
   ```

6. Run it with (indicating all libraries plus the local folder):

   ```
   java -cp "lucene-core-6.2.1.jar:lucene-analyzers-common-
   6.2.1.jar;lucene-queryparser-6.2.1.jar:." events.DocumentEventDemo
   ```

- Consider converting the filenames of the results into something readable, or catch mouse clicks on them so that they can be consulted, or display part of their contents (need to add them to the index upon indexing)