Information Retrieval
Quiz (26 April)

Name:
Roll Number:

Instructions:

1. It is a close book examination.
2. Please write name, roll number, and project group number in the answer sheet.
3. Exam duration is 30 minutes.
4. All questions are mandatory.
5. Total marks are 20.

Question 1: Explain with an example how you retrieve documents for a given query based on boolean search. Briefly mention all steps to retrieve the documents. Highlight the problems with this approach. How can you fix these problems? *[6 marks]*

Answer: Around the following slides.

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query

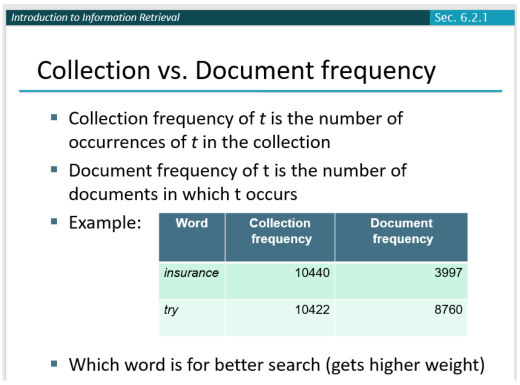
- Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa
- When a system produces a ranked result set, large result sets are not an issue
    - Indeed, the size of the result set is not an issue
    - We just show the top *k* (≈ 10) results
    - We don't overwhelm the user
    - Premise: the ranking algorithm works

All steps of IR include pre-processing, indexing, document representation, matching, and relevance feedback as discussed in earlier slides in course.

Question 2: For information retrieval discuss the importance of the following: (i) term frequency and (ii) document frequency. Assume N is the total number of documents and df$_t$ is the document frequency of token *t* (i.e., the number of documents that contain *t*). Discuss the relationship between df$_t$ with the informativeness of *t.* Assume IDF is defined by the following formula: $\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$. Discuss with an example the effect of TF and IDF on Information Retrieval and how they can be used together for the best ranked result. How to handle the smoothing in the combined TF and IDF formula (write the updated formula)? *[8 marks]*

Answer:
- Rare terms are more informative than frequent terms
    - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Recall the example:



Introduction to Information Retrieval Sec. 6.2.1

Collection vs. Document frequency

- Collection frequency of *t* is the number of occurrences of *t* in the collection
- Document frequency of t is the number of documents in which t occurs
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is for better search (gets higher weight)

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $\text{df}_t \leq N$
- We define the idf (inverse document frequency) of t by

$$\text{idf}_t = \log_{10}\,(N/\text{df}_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$\text{idf}_t = \log_{10}\,(N/\text{df}_t)$$

There is one idf value for each term t in a collection.

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.

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\text{w}_{t,d} = \log(1+\text{tf}_{t,d}) \times \log_{10}(N/\text{df}_t)$$

- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Question 3: Discuss the importance of the user's search intent in Information Retrieval with an example. Discuss the approaches to capture user's intent in the context of IR. Discuss the pros and cons of these approaches for the quality of the IR results and user experience (discuss with respect to convenience and privacy). *[6 marks]*

Answer: See the slides

How can we more robustly match a user's search intent?

Query expansion:

- **Relevance feedback** could allow us to capture this if we get near enough to matching documents with these words
- We can also use information on **word similarities**:
  - A manual **thesaurus** of synonyms for query expansion
  - A **measure of word similarity**
    - Calculated from a big document collection
    - Calculated by query log mining (common on the web)

Document expansion:

- Use of **anchor text** may solve this by providing human authored synonyms, but not for new or less popular web pages, or non-hyperlinked collections

Search log query expansion

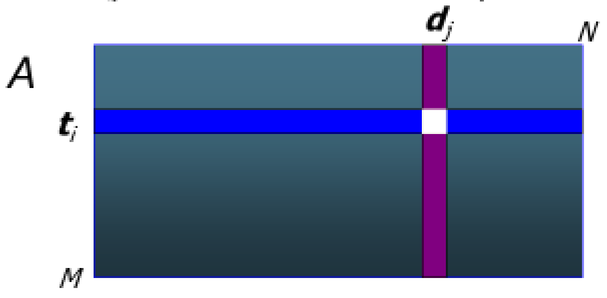
- Context-free query expansion ends up problematic
  - [wet ground] ≈ [wet earth]
  - So expand [ground] ⇒ [ground earth]
  - But [ground coffee] ≠ [earth coffee]
- You can learn query context-specific rewritings from search logs by attempting to identify the same user making a second attempt at the same user need
  - [Hinton word vector]
  - [Hinton word embedding]
- In this context, [vector] ≈ [embedding]

Automatic Thesaurus Generation

- Attempt to generate a thesaurus automatically by analyzing a collection of documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
- You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate. Why?

Simple Co-occurrence Thesaurus

- Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is term-document matrix.
- $w_{i,j}$ = (normalized) weight for (t_i,$\mathbf{d}_j$)



- For each t_i, pick terms with high values in C

Automatic thesaurus generation example … sort of works

Word	Nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, cease, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasites
senses	grasp, psyche, truly, clumsy, naïve, innate

Too little data (10s of millions of words) treated by **too sparse method**.
100,000 words = 10^{10} entries in *C*.

How can we more robustly match a user's search intent?

We want to **understand** a query, not just do String equals()

- If user searches for [Dell notebook battery size], we would like to match documents discussing "Dell laptop battery capacity"
- If user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"

A pure keyword-matching IR system does nothing to help….

Simple facilities that we have already discussed do a bit to help

- Spelling correction
- Stemming / case folding

But we'd like to better **understand** when query/document match