Query Languages

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.

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"

Searching for Similar Words

- When spell-correcting a word, it is inefficient to serially search every word in the dictionary, compute the edit distance or LCS for each, and then take the most similar word.
- Use indexing to find most similar dictionary word without doing a linear search.

k-gram Index

- An inverted index for sequences of *k* characters contained in a word.
  - 3-grams for "index": \$in, ind, nde, dex, ex\$ (where \$ is a special char denoting start or end of a word)
- For each *k*-gram encountered in the dictionary, the *k*-gram index has a pointer to all words that contain that *k*-gram.
  - dex → {index, dexterity, ambidextrous}

Using a *k*-gram Index

- Given a word, generate its "bag of *k*-grams" and use the *k*-gram index like a normal inverted index to find a word that contains many of the same *k*-grams.
- Like normal document retrieval except:
  - words → *k*-grams
  - documents → words
- Example:
  - Query: endex →{en, end, nde, dex, ex}
  - Retrieval Result: 1) index, 2) ended, 3) endear….
  - Compute detailed score just for top retrievals and take final top-scoring candidate.

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

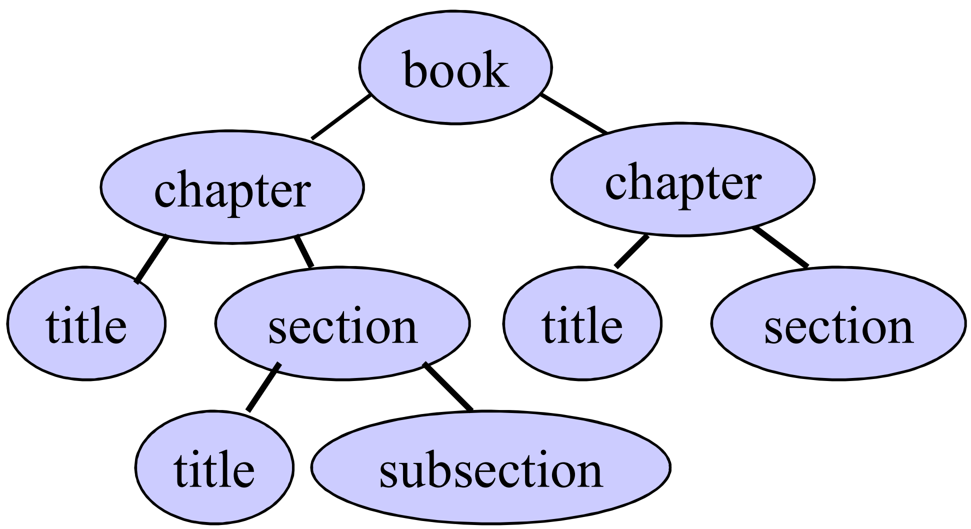
Perl Regex Examples

- 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: Perl regex's supported in java.util.regex package

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