1 Introduction

1.1 Text

Searching

• A breathtaking number of information resources are available: books, databases, the web, newspapers, ...

• To locate relevant information, we need to be able to search these resources, which often are written texts:
  • Searching in a library catalog (e.g., using OSCAR)
  • Searching the web (e.g., using Google)
  • Advanced searching in text corpora (using regular expressions) (e.g., using Opus)

1.2 Speech

Searching in speech

• One might also want to search for speech, e.g., to find a particular sentence spoken in an interview one only has a recording (audio file) of.

• With current technology, this is only possible if the interview is transcribed, using the IPA or another writing system.

• It is, however, already possible to
  • detect the language of a spoken conversation, e.g., when listening in to a telephone conversation
  • detect a new topic being started in a conversation

• In the following, we focus on searching in text.

2 Searching in a Library Catalog

Searching in a library catalog

• To find articles, books, and other library holdings, a library generally provides a database containing information on its holdings.

• OSCAR is the database frontend providing access to the library database at OSU.

• OSCAR makes it possible to search for the occurrence of literal strings occurring in the author, title, call number, etc. associated with an item held by the library.
Basic searching in OSCAR

- Literal strings are composed of characters which naturally must be in the same character encoding system (e.g. ASCII, ISO8859-1, UTF-8) as the strings encoded in the database.
- For literal strings, OSCAR does not distinguish between upper and lower-case letters (i.e. they aren’t so literal after all ;-) )
- Adjacent words are searched as a phrase.
  - art therapy
  - vitamin c

Keyword searching in OSCAR

- In addition to querying literal strings, the keyword search query language of OSCAR also supports the use of
  - special characters to abbreviate multiple options
  - special operators for combining two query strings (boolean operators) or modifying the meaning of a single string (unary operators)

2.1 Special characters

OSCAR: Special characters

- Use * for 1–5 characters at end or within a word.
  - art* finds arts, artists, artistic
  - gentle*n
- Use ** for any number of characters at end of word.
  - art** finds artificial, artillery
- Use ? for a single character at end or within a word.
  - gentle?m
- The special * and ? characters must have at least 2 characters to their left. (→ for efficiency reasons)

2.2 Operators

OSCAR: Literal Strings and Operators (I)

- Use and or to specify multiple words in any field, any order.
  - art and therapy
  - art or therapy
  - c+ or c++
- Use and not to exclude words.
  - art and not therapy

OSCAR: Operators (II)

- Use parentheses to group words together when using more than one operator.
  - art therapy and not ((music or dance) therapy)
- Use near to specify words within 10 words of each other, in any order.
  - art near therapy
- Use within n to specify words within n words of each other. The value of n has no limit.
  - art within 12 therapy

3 Searching the web

Searching the web

A computer user

- wants to find something on “the web”, i.e., in files accessible via the hypertext transfer protocol (http) protocol on the internet
- goes to a search engine = program that matches documents to a user’s search requests
- enters a query = request for information
- gets a list of websites that might be relevant to the query
- evaluates the results: either picks a website with the information looked for or reformulates the query

The nature of the web

- Web pages are generally less structured than a record in a library database (with title, author, subject, and other fields).
- One generally searches for words found anywhere in the document.
- It is, however, possible to include metadata in a web page.
- Metadata is additional, structured information that is not shown in the web page itself; e.g., the language a web page is in, its character encoding, author, keywords, etc.
- Example for a meta tag: <META name="keywords" lang="en-us" content="vacation, Greece">
Search engines

• Search engines (e.g., Google)
  – store a copy of all web pages
  – create an index to provide efficient access to this large number of pages (e.g., Google currently searches over 4 billion pages)
  – compute a rank for each web page to be able to rank the query results

• Search engines differ in various ways:
  – stemming: treat bird and birds as the same or not
  – capitalization: treat trip and Trip the same or not
  – use of operators
  – special interface for advanced searching
  – how search results are ranked
  – clustering: group similar results or not

3.1 Operators

Google: Operators (I)

• +: Require a word to occur in the result
  e.g., To find a restaurant that serves both tofu and BBQ one could try +tofu +BBQ.

• -: Disallow a word from occurring in the result
  e.g., As a potatoes purist, I search for potatoes -potatoes.

• ˜: Include synonyms of the word

Quotation Marks (phrases)
  e.g., “What Cheer” when looking for sites on What Cheer, Iowa

Google: Operators (II)

• intitle: Find words used in a title
  e.g., intitle:Buckeye finds only web pages which has this word in the title

• inurl: Find words used in the url
  e.g., inurl:ling returns more linguistics webpages than ling does

• link: Find pages that link to a certain page
  e.g., link:www.osu.edu to show pages linking to the main osu web page

• site: Find pages that are part of a single domain
  e.g., I want to find strange attractions involving fish. Knowing one site which has such stuff, one can try fish site:www.roadsideamerica.com.

3.2 Improving searching

Improving searching (I)

How can I make my searches better?

• Be on the watch for ambiguity = one word has multiple meanings
  e.g., bed: flower bed, sleeping bed, truck bed

• Use synonyms and other related words
  e.g., plant: building, complex, works, power (distinguish from flora)

• Be aware of stop words = words that search engines ignore because they are “uninformative,” such as the, of, and so on
  e.g., The Police won’t help you find the rock band any more than Police will

Improving searches (II)

• Exclude problematic words
  e.g., “jefferson airplane -starship” (if you don’t want info on the Starship years)

• Be aware of parts of speech and what other guises they come in.
  e.g., plant: planting, planter, planted (distinguish from power plant)

• Continually narrow your focus (using the feedback)
  e.g., Want to find information on the game Hearts
  1. hearts: too vague, too many non-card game sites → add a related word
  2. hearts cards: better, but still greeting cards listed → I see trick listed on one site’s description and realize this makes for a good keyword
  3. hearts cards trick: good, but now we get card tricks → time for boolean expressions

3.3 Ranking of results

Ranking of results

• Ideally, the webpages matching a query are returned as an ordered list based on a page’s relevance.

• How can a search engine, which does not understand language, determine the relevance of a particular page?
Information used to rank results

- Counting the number of links to and from a page, to determine how popular a page is. (As a result, unpopular or new pages require a more specific query to be found.)
- Keeping track of the nature of links to a page; linked pages might be thematically related.
  e.g., Even if I never mention Sinclair Lewis on a page describing his book *Babbit*, it can be identified if many Sinclair Lewis sites link to my page.
- Bonuses/penalties for sites known to be of high/low quality
- Looking for keywords in metadata
- Counting how often a web result was clicked on by a user (click-through measurement)
- Various secret ingredients

3.4 Evaluating search results

Evaluating search results

What measures can one use to evaluate how successful a query is?

- Precision: How many of the pages returned are the ones we want?
  e.g., Google gives me 400 hits for a query, 200 of which are related to the topic I want; precision = 50%.
- Recall: How many pages on the topic we wanted were actually given? (Hard to calculate for web searching)
  e.g., Google gave me 200 pages I wanted, but there were actually 1000 pages on that topic out there somewhere on the internet; recall = 20%.

We saw earlier how to use our initial results to refine our query and improve precision

4 Advanced searches with regular expressions

Motivating regular expressions

If one wants to be able to describe more complex patterns of words and text, sometimes boolean expressions aren’t enough:

- In a large document I want to find addresses with a zip code starting with 911 (around Pasadena, CA); but clearly we would not want to report back all occurrences of emergency phone numbers in the document.
- I want to find all osu email addresses which occur in a long text.
- I’m writing an online fill-in-the-blank quiz, and I ask you to name the Jackson 5: for Jermaine, I want to accept Germaine, Germane, Jermain, and so on.
  ⇒ It would be nice to have a compact way of representing all of these options.
- Anything where you have to match a complex pattern so-called regular expressions are useful.

Regular expressions: What they are

- A regular expression is a compact description of a set of strings, i.e., a language (in formal language theory).
- They can be used to search for occurrences of these strings
- Regular expressions can only describe so-called regular languages.
- This means that some patterns cannot be specified using regular expressions, e.g., finding a string containing any number of a’s followed by exactly the same number of b’s.
- Note that just like any other formalism, regular expressions as such have no linguistic contents, but they can be used to refer to strings encoding a natural language text.

Regular expressions: Tools that use them

- A variety of unix tools (grep, sed, . . . ), editors (emacs, . . . ), and programming languages (perl, python, . . . ) incorporate regular expressions.
- Implementations are very efficient so that large text files can be searched quickly; but not efficient enough for web searching → no web search engine offers them (yet).
- The various tools and languages differ w.r.t. the exact syntax of the regular expressions they allow.

4.1 Syntax of regular expressions

The syntax of regular expressions (I)

Regular expressions consist of

- Strings of literal characters: c, A100, natural language, 30 years!
- Disjunction:
  - Ordinary disjunction: devoured|ate, familial(y|ies)
  - Character classes: [Tt]he, bec[a|a]me
  - Ranges: [A-Z] (Any capital letter)
- Negation: [^[a]] (Any symbol but a)
  [^[A-Z]-9] (Not an uppercase letter or number)

The syntax of regular expressions (II)

- Counters
  - Optionality: ? colou?r
  - Any number of occurrences: * (Kleene star) [*-9]* years
  - At least one occurrence: + [*-9]+ dollars
- Wildcard for any character: . beg.n for any character in between beg and n
The syntax of regular expressions (III)

- Escaped characters: to specify a character with a special meaning (*, +, .., ?, (, ), |, [, ])) it is preceded by a backslash (\).
- E.g., a period is expressed as \.

- Operator precedence, from highest to lowest:
  - Parentheses (): counters * + ?
  - Character sequences disjunction |

4.2 Grep: An example for using regular expressions

Grep

- Grep is a powerful and efficient program for searching in text files using regular expressions.
- It is standard on Unix, Linux, and Mac OSX, and there also are various ports to Windows (e.g., http://gnuwin32.sourceforge.net/packages/grep.htm, http://www.interlog.com/~tcharron/grep.html, or http://www.wingrep.com/).
- The version of grep that supports the full set of operators mentioned above is generally called egrep (for extended grep).

Grep: Examples for using regular expressions (I)

In the following, we assume a text file f.txt containing, among others, the strings that we mention as matching.

- Strings of literal characters:
  - `egrep 'and' f.txt` matches and, Ayn Rand, Candy and so on
- Character classes:
  - `egrep 'the year [0-9][0-9][0-9][0-9]' f.txt` matches the year 1776, the year 1812, the year 2001, and so on
- Escaped characters:
  - `egrep 'why\?' f.txt` matches why?, whereas `egrep 'why?' f.txt` matches why and wh

Grep: Examples for using regular expressions (II)

- Disjunction (|):
  - `egrep '[G]g' f.txt` matches G or g, so `egrep '[G]ouda' f.txt` matches gouda or Gouda. Note that [G]ouda has the same effect.
- Grouping with parentheses:
  - `egrep 'um(interest|excit)ing' f.txt` matches uninteresting or unexciting.
- Any character (\.):
  - `egrep 'o.e' f.txt` matches ore, one, ole

4.3 Text corpora and searching them

Corpora

- A corpus is a collection of text.
- Corpora with the works of various writers, newspaper texts, etc. have been collected and electronically encoded.
- Corpora can be quite large
- The British National Corpus is a 100 million word collection representing a wide cross-section of current written and spoken British English.

How corpora can be searched

- Both the BNC and the European Parliament corpus can be searched using on-line web-forms.
- Both of the web forms allow regular expressions for advanced searching.
- To provide efficient searching in large corpora, in these search engines regular expressions over characters are limited to single tokens (i.e. generally words).
- BNC:
  - Web form: http://sara.natcorp.ox.ac.uk/lookup.html
  - Regular expressions are enclosed in { }
- European Parliament Corpus:
  - Web form: http://logos.uio.no/cgi-bin/opus/opuscqp.pl?corpus=EUROPARL;lang=en
  - In the simplest case, regular expressions are enclosed in " "

Grep: Examples for using regular expressions (III)

- Kleene star (*):
  - `egrep 'a*rgh' f.txt` matches argh, aargh, aarargh
  - `egrep 'sha(la)*' f.txt` matches sha, shala, shalala, or if you're Van Morrison shalalalalalalala
- One or more (+):
  - `egrep 'john+y' f.txt` matches johny, johnny, ..., but not johy
- Optionality (?):
  - `egrep 'joh?n' f.txt` matches jon and john
