Information Retrieval and Extraction

11.8.2020

Course Logistics

- 3 major parts:
  - 4 sessions: introduction from multiple aspects
  - 9 " : IR fundamentals:
    - models
    - scoring functions
    - crawling, evaluation, etc.

    NLP required specifically for IR
    ML for IR
  - : IE fundamentals
    - majorly: NER
  - : Information Access / Applications of IR
    - moving specifics: social media, sentiments,
    - computational advertising
    - sentiment analysis.

- Tutorials: Thu / Tue evening

- Recommended Books
  - Stanford IR book
  - Search Engines in Practice

- Grading:

Quizzes / In-class:		10%
Assignments (best 3/4)		15%
Project		60%
├ mini	20	
└ major	40	
Term Paper (Final Exam, can be done anytime)		15%

_____ × _____ × _____

- Project Details
  - MINI
    - Individual
    - 4 weeks, starting today
    - Deliverables:
      1.
      2.

Lang:
Py / C++ / Java

DEADLINES
1. 24th August: offline
2. 7th Sept: online + offline

Design and develop a scalable and efficient search engine on wikipedia.

REQUIREMENTS
- Query (1s) → result
- Support "Field Queries"[1]
- Total index size: < 1/4th the size of the doc repository
- build your own indexing scheme

EVAL
- online:
  - search time
  - search efficiency
- offline:
  - Indexing time
  - Indexing size

26th: Dummy Queries

1. Field Query: Searching within subset.

— MAJOR

- Team of 4 constrained choice
- 10 weeks.

Advanced topics
Scope well defined (by us)

- 5 touchpoints
  Report to mentor every 2 weeks
- 3 Evaluations
  - first deliverable: (26th Sept) — scope doc
  - MVP deliverable: (25th Oct) — full system, V1
  - Complete System: (14th Nov) — Demo, presentation report code, etc.

Basic Overview

- Searching is important
- Computational advertising

---

13. 8. 2020.

RECAP

- Information Retrieval: Science of finding documents of unstructured information
- Economic viability of IR
- Standard Search Engines and the new players (Fb, Amazon, etc)

_____ × _____

History:

- 1993:
  - new — WWW
  - no search engines
  - new URLs added manually to CERN "whats new"

  [Aside] — 1990:
  - Archie. (Archive_v). First "search engine"
  - U.Minnisotta: → Veronica → Jughead.

  - First .robo search program www Automated (developed at MIT)
  - Nov. 1993. 2nd websearch engine. (Aliweb)

- 1994:
  - First proper webcrawler (also looks at content).
  - Another: Lycos, in CMU
  - Many others.

- 1995:
  - Two prominent technologies:
    - crawling
    - indexing.

1. CRAWLING:

- problems:
  1a) Need diversity in seed URLs
  b) Need diversity in crawling strategy.
  2. Pages have a lifecycle.
     - TOI, Twitter change every minute
     - Something like IIIT changes occasionally.
     - Some websites never change.

     → Strat: estimate lifecycles.
     1. After I crawl, crawl again.
        - if page has changed, increase freq. for it.
        - if page has not changed, decrease freq. for it.

2. INDEXING

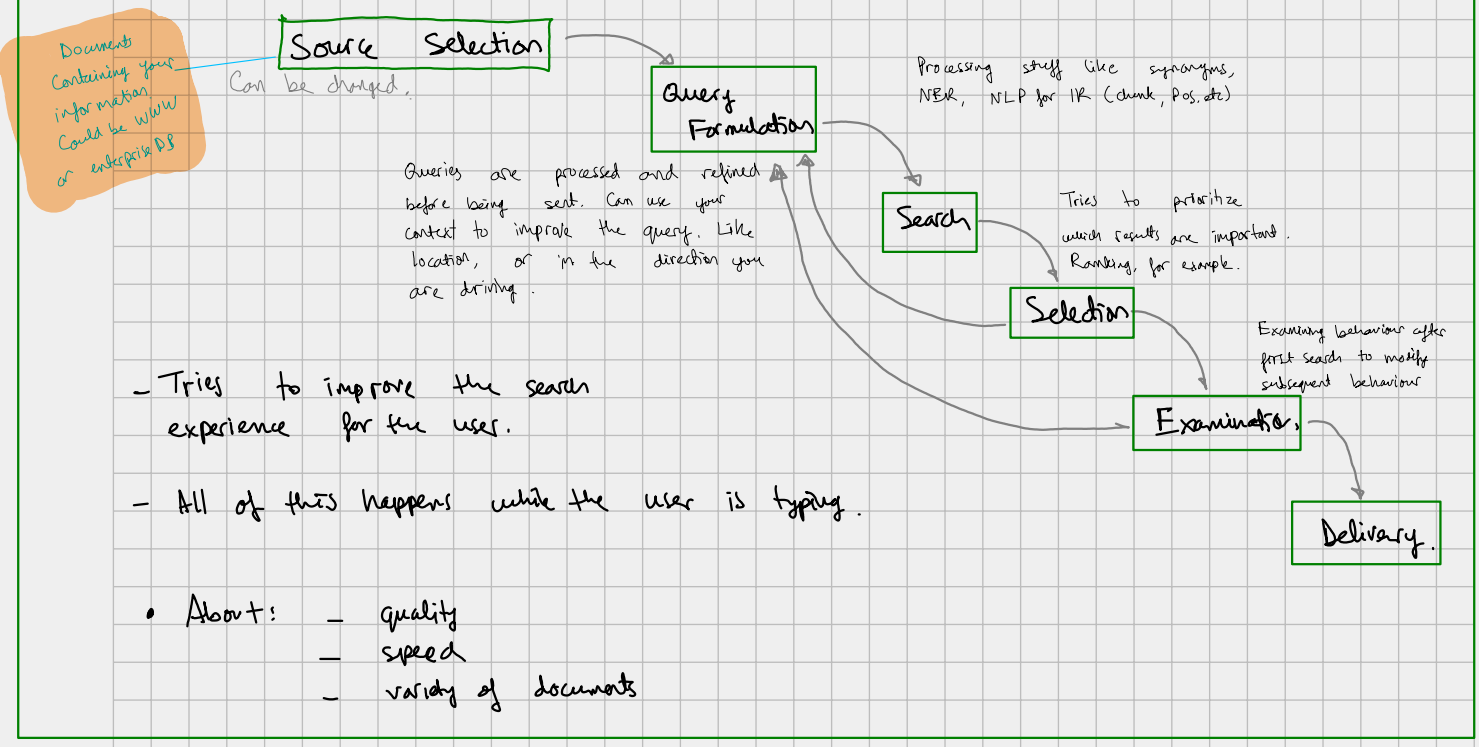
- helps find URL and content.
- Many classes of indexing (schemes) exist.

Improvements to be made: coverage, scale

- Yahoo starts from Stanford in 1994.
  - Introduces the idea of "browse", searching enabled within a "topic directory".
- the web started growing faster than human editors could keep up
  - introduction of classification and clustering programs
- 1998: Google → PageRank
  - Even if you know content of each page, that is not enough.
  - Value assigned by reputation of a given page.
  - Need to rank pages for query by reputation.
  - think of the web as a massive graph, where each URL has
    inlinks, and
    outlinks.
  - Rank: the more inlinks there are, recursively.
- domain specific search engines: let you incorporate more domain knowledge than generic search can do.

18.8.2020

IR Cycle



- Tries to improve the search experience for the user.
- All of this happens while the user is typing.

- About:
  - quality
  - speed
  - variety of documents

Central problem with search

- As a searcher, I have information need in my head, expressed as a few terms. } Diff b/w concepts query terms
- As a document writer, I also have a concept in my head expressed as a few document terms. } Another diff.

Both are suboptimal

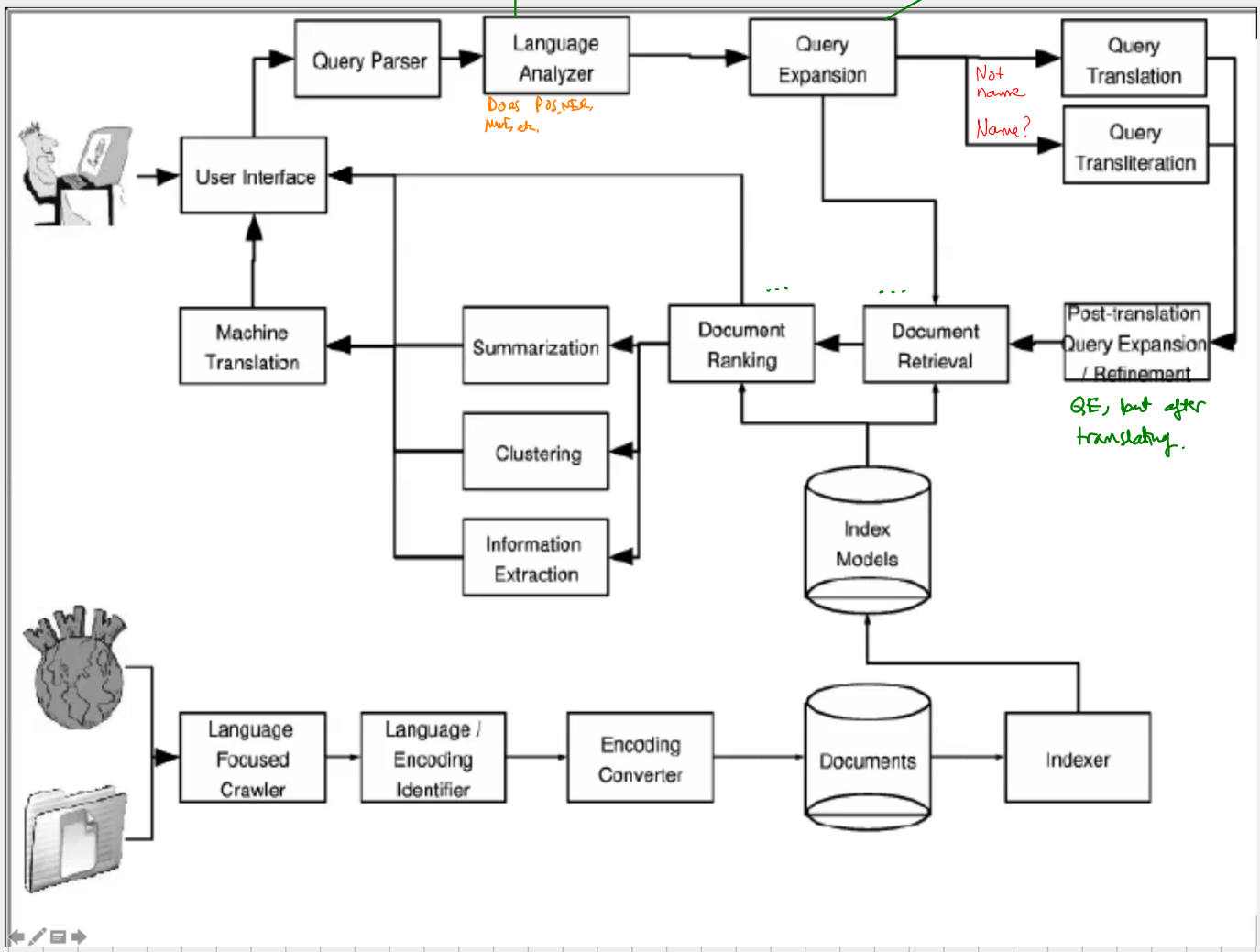
- Job of search engine: Match the (suboptimal) Document Terms and Query Terms

Map the meaning, not the words
hence, search is complex.

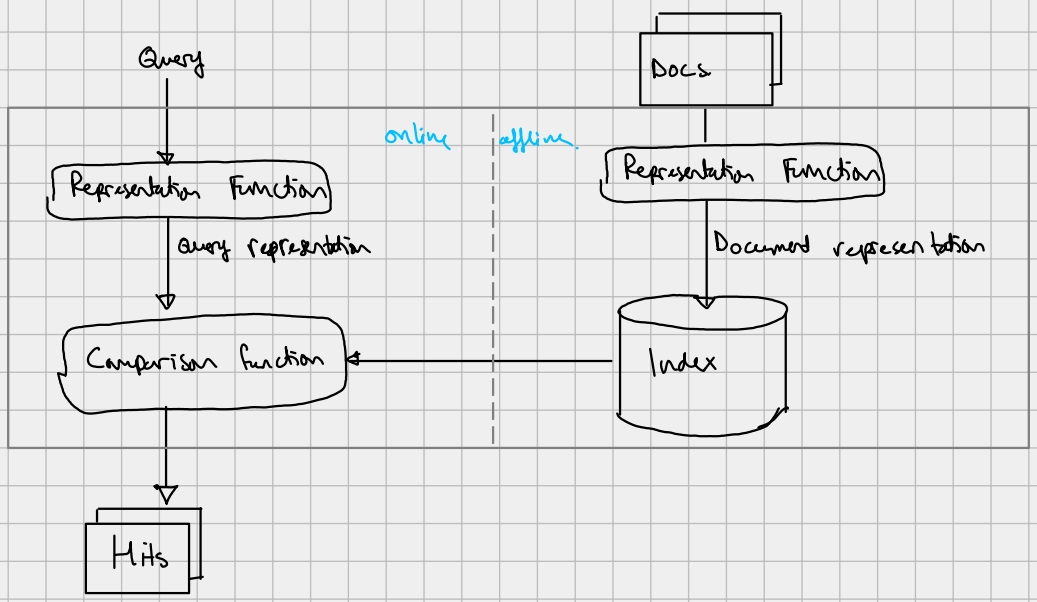
Cross-Language IR system by IRE

- Auto identification of lang (when not unicode) hard
- language ≠ script

Get: synonyms, spellcheck, etc.



Abstract IR arch



20.8.20

- How do we approach retrieving data
  1. Looking at the content
  2. Looking at the meta data

Labeling by processing metadata } But this is not a replacement
- No IR without content.
- But metadata helps.

Document-query similarities

Much of IR depends on one idea: - Documents are similar to each other if they are relevant to the same query.
- likewise

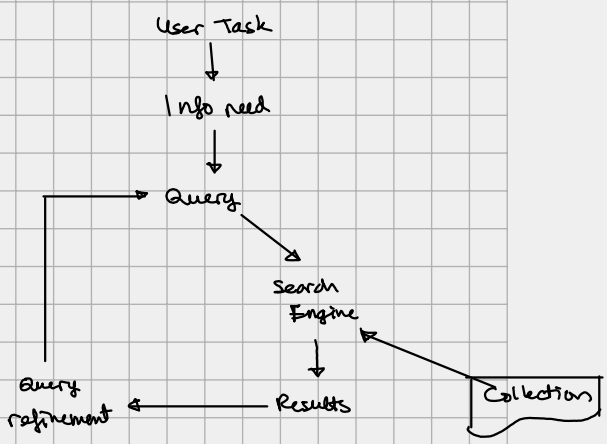
- How is similarity measured?
  - {query vocab} ⊂ {doc. vocab}
  - prob model based on term occurences
  - Bag of words.
  - How to compare search engines? hard.
    a given query will determine quality.
- Getting unstructured data hard.

25.8.20

Text Indexing

Classic Search Model

- getting query from info need is search engine's job.
- getting collection also hard but lets take as black box for now.



Query on Unstructured Data

- boolean query for AND + NOT
- could grep
  - slow
  - NOT is non-trivial
  - no ranking
  - no advanced search: like proximity

⇒ BUILD INDEXES

Reqs:
1. Indexing time smol
2. Indexing size "
3. query processing fast

1. **Term-Document Incidence Matrix:**
   - rows: vocabulary
   - cols: documents

Can bitwise AND/OR

Coming to bigger Collections

- say: N = 1 mill. documents, ~1000 words each
- ~ 6 GB data in the documents
- say: M = 500 K distinct terms.

> Standard TDI Matrix: 500 GB.
> but matrix is sparse (~ 1:499)

⇓ **Inverted Index**

- do not use fixed-size arrays
- Constructing
  - take documents
    ↓ tokenizer
    ↓ Token stream
    ↓ Ling. modules (stemming, etc)
    ↓ ??.
  - ... Linguistics
  - ... Sort
    - sort by terms, then docID:

Search

Map-Reduce
Hadoop / spark

Saltorn

Index Compression

27.08.20

Why compress the dictionary? (slides)

- want to keep it in memory.

- Could by fixed width? but mostly wasted.
- Store as long list of chracters.
- Blocking: store term lengths (1 extra byte) int seperators.
- Front-coding
  - sorted words have long common prefixes
  - store differences.

Postings compression:

1. Store the gap
2. Variable-length encoding

3. Gamma codes.