Information Retrieval and Evaluation

Knowledge Discovery and Data Mining 2 (VU) (707.004)

Heimo Gursch, Roman Kern

ISDS, TU Graz

2019-04-04

Outline

Basics

Inverted Index & Text-Preprocessing

Foundations

Definition

Information Retrieval (IR) is **finding material** (usually documents) of an **unstructured** nature (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).

The term “term” is very common in Information Retrieval and typically refers to single words (also common: bi-grams, phrases, ...)

Basic assumptions of Information Retrieval

Collection Fixed set of documents

Goal Retrieve documents with information that is **relevant** to user’s **information need** and help her complete a **task**.

Structured, Semi-Structured Data

Structured data

- Format and type of information is known
- IR task: normalize representations between different sources
- Example: SQL databases

Semi-structured data

- Combination of structured and unstructured data
- IR task: extract information from the unstructured part and combine it with the structured information
- Example: email, document with meta-data

Unstructured data

Unstructured data

- Format of information is not know, information is hidden in (human-readable) text
- IR task: extract information at all
- Example: blog-posts, Wikipedia articles

This lecture focuses on unstructured data.

Overview Indexed Searching

Split process

Indexing Convert all input documents into a data-structure suitable for fast searching

Searching Take an input query and map it onto the pre-processed data-structure to retrieve the matching documents

Inverted index

- Reverses the relationship between documents and contained terms (words)
- Central data-structure of a search engine

Information Stored in an Inverted Index

Dictionary

Sorted List of all terms found in all documents

Postings

Information about the occurrence of a term, consisting of:

Document (`DocID`) Document identifier, i. e. , in which document the term was found

Position(s)(`Pos`) Location(s) of the term in den original document

Term Frequency (`TF`) How often the term occurs in the document[a]

[a] The term frequencies could also be calculated by counting the position entries. As computing time is more precious than memory, the term frequencies are stored as well.

Example

Document 1

Sepp was stopping off in Hawaii. Sepp is a co-worker of Heidi.

Document 2

Heidi stopped going to Hawaii. Heidi is a co-worker of Sepp.

Dictionary	$\mapsto$	Postings
co-worker	$\mapsto$	(Doc:1; TF:1; Pos:10);(Doc:2; TF:1; Pos:9)
going	$\mapsto$	(Doc:2; TF:1; Pos:3)
Hawaii	$\mapsto$	(Doc:1; TF:1; Pos:6);(Doc:2; TF:1; Pos:5)
Heidi	$\mapsto$	(Doc:1; TF:1; Pos:12);(Doc:2; TF:2; Pos:1,6)
Sepp	$\mapsto$	(Doc:1; TF:2; Pos:1,7);(Doc:2; TF:1; Pos:11)
stopped	$\mapsto$	(Doc:2; TF:1; Pos:2)
stopping off	$\mapsto$	(Doc1; TF:1; Pos 3)

Properties

Properties

- Search for terms
- Formulate boolean queries
- Search for phrases (i. e. , terms in a particular order)
- Information where the term occurred in the document
- Possible creation of snippets
- Documents can be ranked by the number of terms they contain

Keep in mind that an inverted index

- ... takes up additional space, not only storing the original documents, but also the inverted index
- ... must be kept up to date

Text Preprocessing

To make a text indexable, a couple of steps are necessary:

1. Detect the language of the text
2. Detect sentence boundaries
3. Detect term (word) boundaries
4. Stemming and normalization
5. Remove stop words

Language Detection

Letter Frequencies

Letter	Percentage of occurrence English	German
A	8.17	6.51
E	12.70	17.40
I	6.97	7.55
O	7.51	2.51
U	2.76	4.35

N-Gram Frequencies

Better–more elaborate–methods use statistics over more than one letter, e. g. statistics over two, three or even more consecutive letters.

Sentence Detection

First approach

Every period marks the end of a sentence

Problem

Periods also mark abbreviations, decimal points, email-addresses, etc.

Utilize other features

- Is the next letter capitalized?
- Is the term in front of the period a known abbreviation?
- Is the period surround by digits?
- ...

Tokenization – Problem

Goal

- Split sentences into tokens
- Throw away punctuations

Possible Pit Falls

- White spaces are no safe delimiter for word boundaries(e. g. *New York*)
- Non-alphanumerical characters can separate words, but need not (e. g. *co-worker*)
- Different forms (e. g. *white space* vs. *whitespace*)
- German compound nouns (e. g. *Donaudampfschiffahrtsgesellschaft*, meaning *Danube Steamboat Shipping Company*)

Tokenization – Solutions

Supervised Learning

- Train a model with annotated training data, use the trained model to tokenize unknown text
- Hidden-Markov-Models and conditional random fields are commonly used

Dictionary Approach

- Build a dictionary (i. e. list) of tokens
- Go over the sentence and always take the longest fitting token (greedy algorithm!)
- Remark: Works well for Asian languages without white spaces and short words. Problematic for European languages

Normalization

Some definitions

Token Sequence of characters representing a useful semantic unit, instance of a type

Type Common concept for tokens, element of the vocabulary

Term Type that is stored in the dictionary of an index

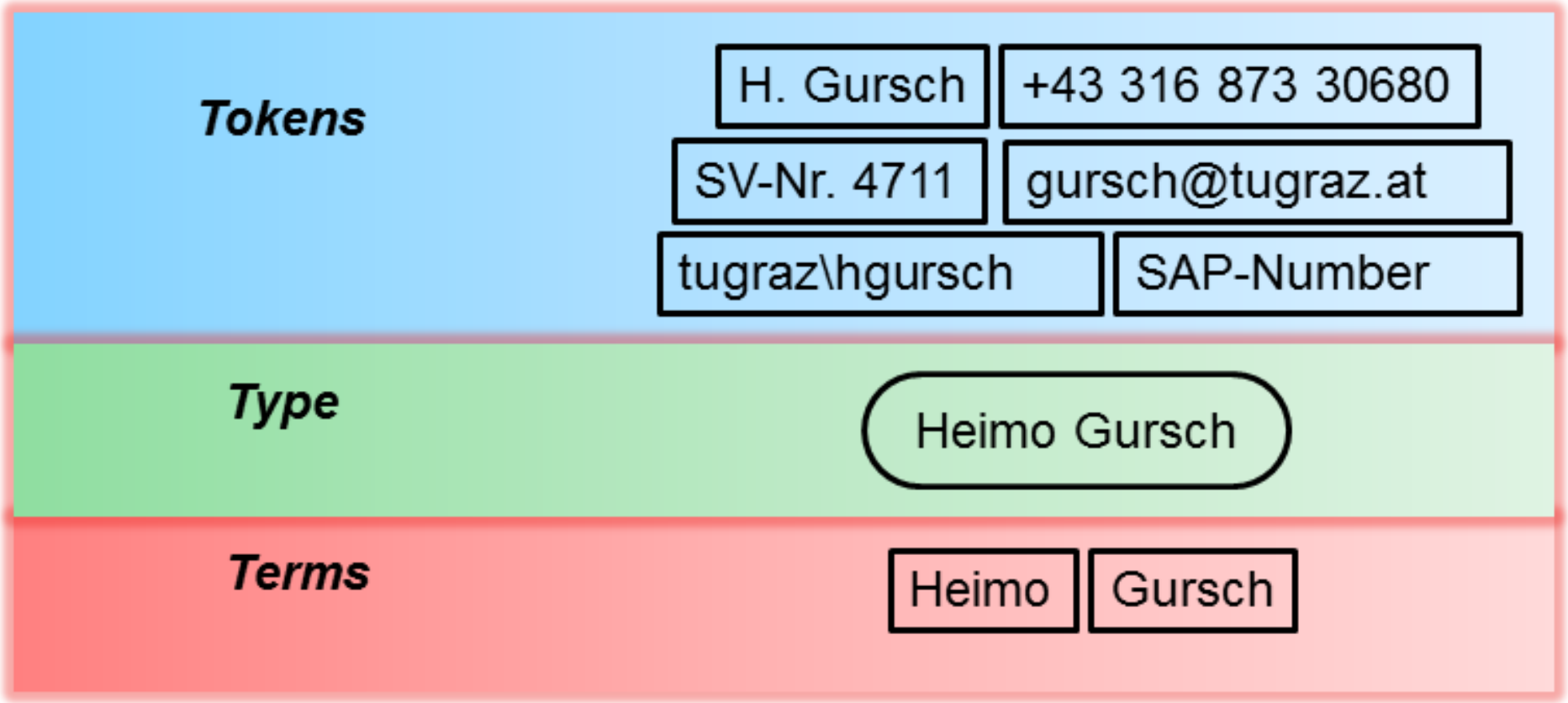
Task

1. Map all possible tokens to the corresponding type
2. Store all representing terms of one type in the dictionary

Solutions

- Provide list of different representations
- Provide rules for mapping representations to main form
- Map different spelling version by using a phonetic algorithm

Example


Tokens
H. Gursch
+43 316 873 30680
SV-Nr. 4711
gursch@tugraz.at
tugraz\hgursch
SAP-Number
Type
Heimo Gursch
Terms
Heimo
Gursch

Stemming & Lemmatization

Definition

Goal of both Reduce different grammatical forms of a word to their common infinitive[a]

Stemming Usage of heuristics to chop off / replace last part of words (Example: Porter's algorithm).

Lemmatization Usage of proper grammatical rules to recreate the infinitive.

[a] The common infinitive is the form of a word how it is written in a dictionary. This form is called lemma, hence the name *Lemmatization*.

Examples

- Map *do*, *doing*, *done*, to common infinitive *do*
- Map *going*, *went*, *gone* to *go*

Stop Words

Definition Extremely common words that appear in nearly every text

Problem As stop words are so common, their occurrence does not characterise a text

Solution Just drop them, i. e. , do not put them in the index directory

Common stop words

- Write a list with stop words (List might be topic specific!)
- Usual suspects: articles (*a*, *an*, *the*), conjunction (*and*, *or*, *but*, ...), pre- and postposition (*in*, *for*, *from*), etc.

Solution

Stop word list Ignore word that are given on a list (black list)

Problem Special names and phrases (*The Who*, *Let It Be*, ...)

Solution Make another list... (white list)

Advanced Concepts

Relevance Feedback, Query Processing, ...

Relevance Feedback

Integrate feedback from the user

- Given a search result for a user's query
- ... allow the user to rate the retrieved results (good vs. not-so-good match)
- This information can then be used to re-rank the results to match the user's preference
- Often automatically inferred from the user's behaviour using the search query logs
- ... and ultimately improve the search experience

The query logs (optimally) contain the queries plus the items the user has clicked on (the user's session)

Relevance Feedback

Pseudo relevance feedback

- No user interaction is needed for the pseudo relevance feedback
- ... the first n results are simply assumed to be a good match
- As if the users has clicked on the first results and marked them as good match
- $\rightarrow$ often improves the quality of the search results

Pseudo relevance feedback is sometimes also called blind relevance feedback

Query Reformulation

Modify the query, during of after the user interaction

- Query suggestion & completion
- (Automatic) query correction
- (Automatic) query expansion

Query Suggestion

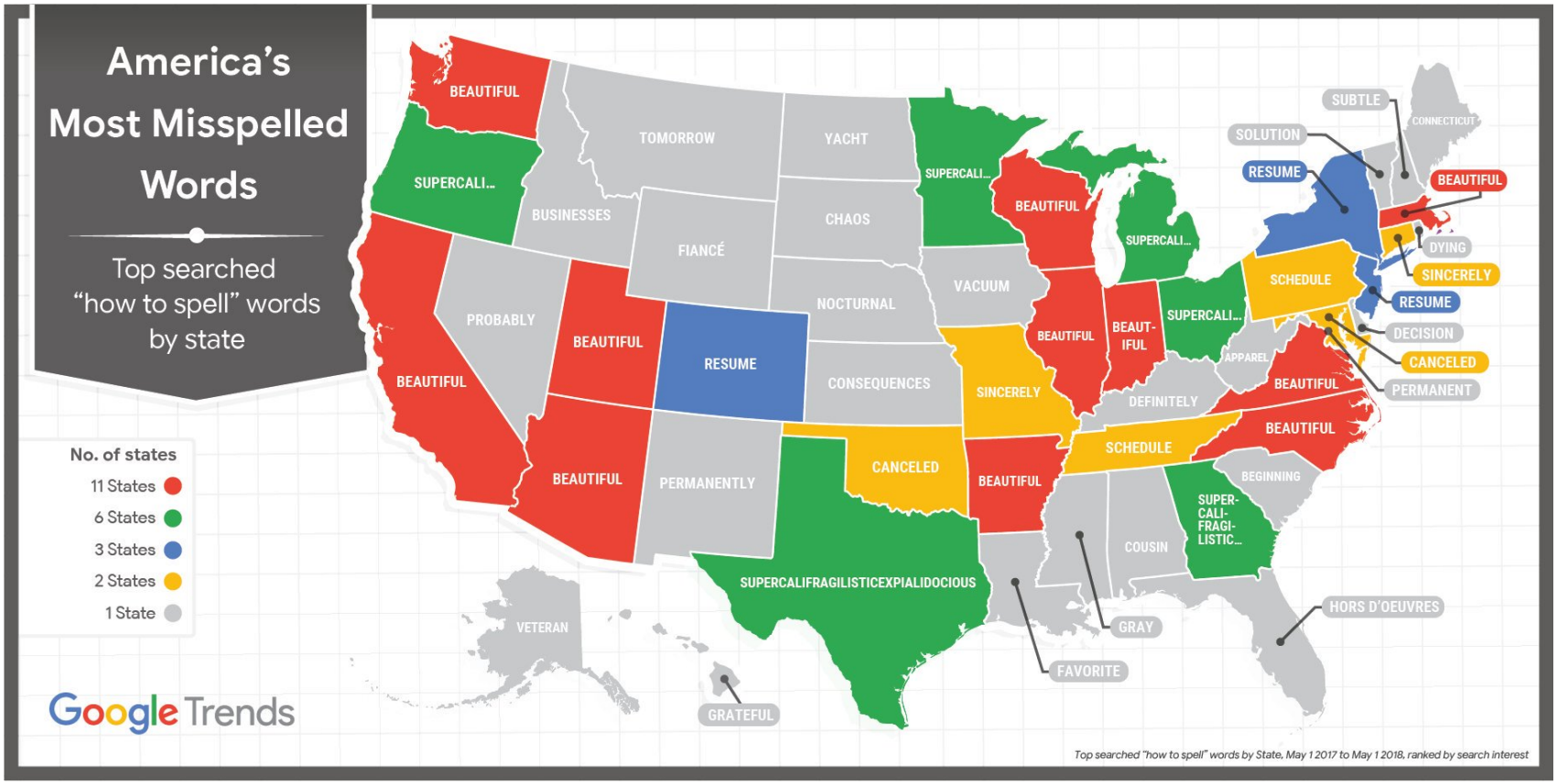
- The user start typing a query
- ... automatically gets suggestions
- $\rightarrow$ to complete the currently typed word
- $\rightarrow$ to complete a whole phrase (e.g. the next word)
- $\rightarrow$ suggest related queries

Query Correction

- The user entered a query, which may contain spelling errors
- ... automatically correct or suggest possible rectified queries
- ... also known as Query Reformulation
- $\rightarrow$ use other users behaviour (harvest query logs)
- $\rightarrow$ use the frequency of terms found in the indexed documents (and the similarity with the entered words)

Notebook with various distances (Levenshtein being the best known):
`http://nbviewer.ipython.org/gist/MajorGressingham/7691723`

Using Google as Spell Checker



Query Expansion

- The user has entered a query
- ... automatically add (related) words to the query
- $\rightarrow$ Global query expansion - only use the query plus corpus or background knowledge
- $\rightarrow$ Local query expansion - conduct the search and analyse the search results

Global Query Expansion

- Uses only the query and background knowledge
- Example: Use of a thesaurus
  - Query: `[buy car]`
  - Use synonyms from the thesaurus
  - Expanded query: `[(buy OR purchase) (car OR auto)]`

Local query expansion

- Uses the query plus the search results
- ... conceptually similar to the pseudo relevance feedback
- Look for terms in the first n search results and add them to the query
- Re-run the query with the added terms

More Like This

A common feature of search engines is the more like this search

- Given a search results
- ... the user wants items similar to one of the presented results
- This can be seen as: taking a document as query
- $\rightarrow$ which is a common implementation (but restricting the query to the most discriminative terms)

Learning-To-Rank

Motivation from Web-Search

- Relevance might be due to a lot of different reasons
  - Keywords in the text or the meta-data
  - Anchor texts
  - PageRank
  - ... many other
- Basic idea:
  - Use evidence (from users)
  - ... which source of information
  - ... is the most beneficial
  - ... by the use of Machine Learning

Learning-To-Rank

Supervised Learning - Setting

- Number of queries - $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$
- Set of documents - $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$
  - Each queries has labelled relevant documents
  - With labels $\mathcal{Y} = \{1, 2, \ldots, l\}$
  - being ranked, i. e. $l \succ l-1 \succ \cdots \succ 1$
  - Relevant documents for query q_i: $D_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,n_i}\}$
  - Labels for the query q_i: $\boldsymbol{y}_i = \{y_{i,1}, y_{i,2}, \ldots, y_{i,n_i}\}$
- Training data set: $\mathcal{S} = \{(q_i, D_i), \boldsymbol{y}_i\}_{i=1}^{m}$
  - By replacing (q_i, D_i) by representative features

H. Li, “A Short Introduction to Learning to Rank,” IEICE Trans. Inf. Syst., vol. E94-D, no. 1, pp. 1–2, 2011.

Supervised Learning - Data Labelling

- Where to take the labelling information from?
  - Directly from human judgements
    - ★ e. g. , perfect, excellent, good, fair, bad
  - Indirectly via user behaviour
    - ★ e. g. , via Web logs analysis

Note: Ordinal classification (ordinal regression) vs. ranking, i. e. , ranking is about recreating the expected ordering

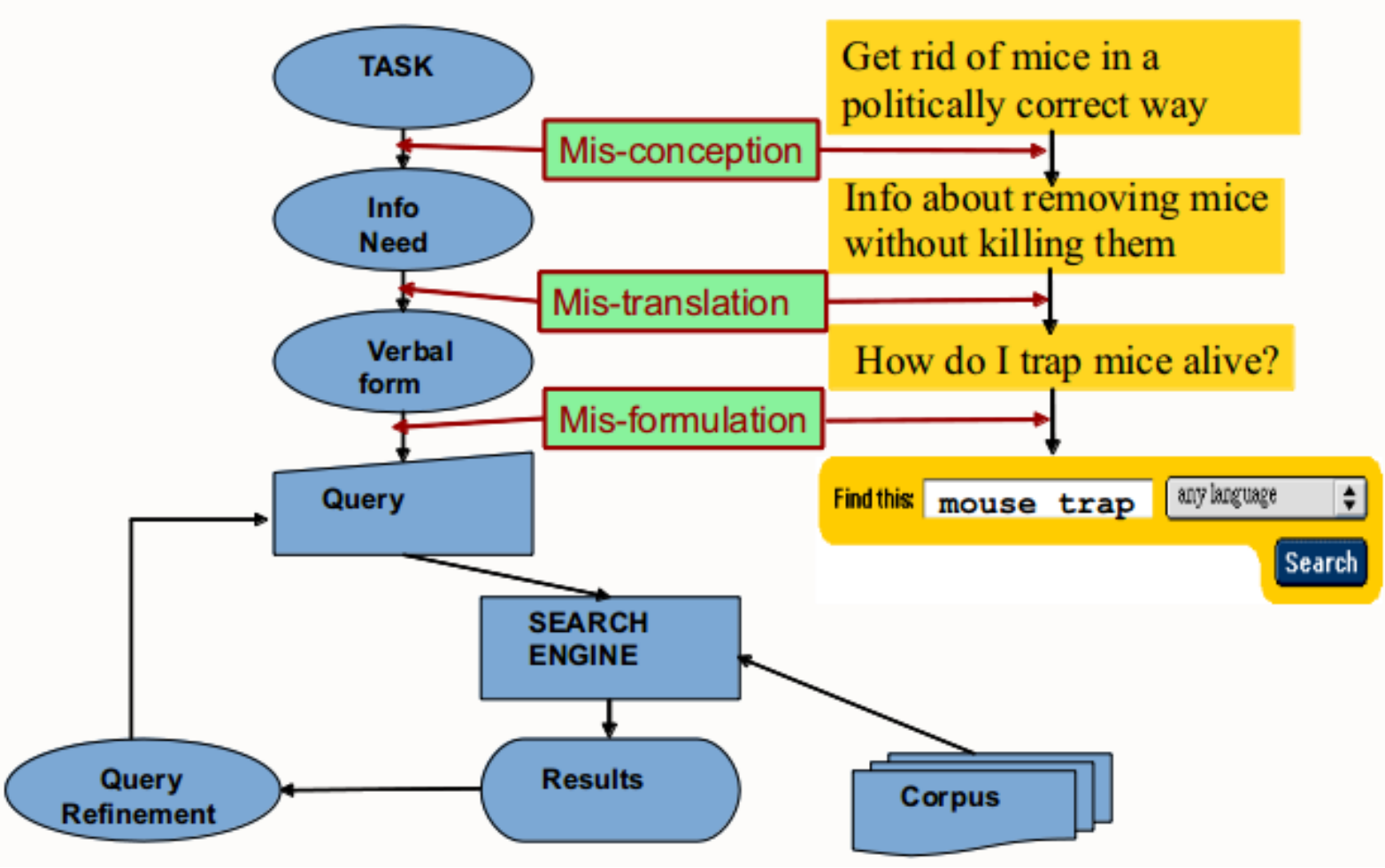
Learning-To-Rank

Supervised Learning - Main Approaches

- Pointwise approach
  - Transform the problem in a classification, regression or ordinal classification
  - e. g. , SVM for ordinal classification, i. e. , find parallel hyperplanes
- Pairwise approach
  - Transform the problem in a pairwise classification, pairwise regression
  - e. g. , Ranking SVM, i. e. , to classify the order of pairs
- Listwise approach
  - Directly learn/optimise the ranking
  - e.g. SVM MAP, i.e. to optimise on a scoring function (goodness of ranking)

Information flow

The classic search model



Probabilistic Information Retrial

Every transformation can hold a possible information loss

1. Task to query
2. Query to document

- Information loss leads to uncertainty
- Probability theory can deal with uncertainty

Basic idea behind probabilistic IR

- Model the uncertainty, which originates in the imperfect transformations
- Use this uncertainty as a measurement how relevant a document is, given a certain query

Probabilistic Information Retrial

Basic model

- Given a query - q
  - ... and an (unknown/uncertain) set of relevant document - $\mathcal{D}$
  - ... and irrelevant documents - $\neg\mathcal{D}$
- Representation of a document - d_i
- $P(\mathcal{D}|d_i)$ is the probability that d_i is relevant and $P(\neg\mathcal{D}|d_i)$ that it is irrelevant
- Rank documents in decreasing order of probability of relevance - $P(\mathcal{D}|q, d_i)$

Probabilistic Information Retrial Models

Binary Independence Model

- **Binary:** documents and queries represented as binary term incidence vectors
- **Independence:** terms are assumed to be independent of each other

Note: Computing the “true” probabilities would not be feasible

Language Models

Basic Idea

- Train a probabilistic model M_d for document d, i. e. , as many trained models as documents
- M_d ranks documents on how possible it is that the user's query q was created by the document d
- Results are document models (or documents, respectively) which have a high probability to generate the user's query

Language Models

Alternative point of view

- Intuitively, each document defines a “language”
- What is the probability to generate the given query, given a language model
- What is the probability to generate the given document, given a language model

Examples for Language Models

Successive Probability of query terms

$$P(t_1 t_2 \dots t_n) = P(t_1)P(t_2|t_1)P(t_3|t_1 t_2) \dots P(t_n|t_1 \dots t_{n-1})$$

Unigram Languge Model

$$P_{uni}(t_1 t_2 \dots t_n) = P(t_1)P(t_2) \dots P(t_n)$$

Bigram Languge Model

$$P(t_1 t_2 \dots t_n) = P(t_1)P(t_2|t_1)P(t_3|t_2) \dots P(t_n|t_{n-1})$$

Probability $P(t_n)$ Probabilities are generated from the term occurrence in the document in question

And many more... A lot more, and more complex models are available

Note: Smoothing plays an important role in language models

Index Construction

Going beyond simple indexing

- Term-document matrices are very large
- But the number of **topics** that people talk about is small (in some sense)
  - Clothes, movies, politics, ...
- Can we represent the term-document space by a lower dimensional latent space?

Index Construction

Latent Semantic Indexing (LSI)

- is based on Latent Semantic Analysis (see KDDM1), which
- is based on Singular Value Decomposition (SVD), which
- creates a new space (new orthonormal base), which
- allows to be mapped to lower dimensions, which
- might improve the search performance

Singular Value Decomposition

SVD overview

- For an $m \times n$ matrix A of rank r there exists a factorization (Singular Value Decomposition = SVD) as follows:
  - $M = U\Sigma V^T$
  - ... U is an $m \times m$ unitary matrix
  - ... Σ is an $m \times n$ diagonal matrix
  - ... V^T is an $n \times n$ unitary matrix
- The columns of U are orthogonal eigenvectors of AA^T
- The columns of V are orthogonal eigenvectors of A^TA
- Eigenvalues $\lambda_1 \ldots \lambda_r$ of AA^T are the eigenvalues of A^TA.

Singular Value Decomposition

Matrix Decomposition

- SVD enables lossy compression of a term-document matrix
  - Reduces the dimensionality or the rank
  - Arbitrarily reduce the dimensionality by putting zeros in the bottom right of sigma
  - This is a mathematically optimal way of reducing dimensions

Singular Value Decomposition

Reduced SVD

- If we retain only k singular values, and set the rest to 0
- Then Σ is $k \times k$, U is $m \times k$, V^T is $k \times n$, and A_k is $m \times n$
- This is referred to as the **reduced SVD**
- It is the convenient (space-saving) and usual form for computational applications

Approximation Error

- How good (bad) is this approximation?
- It's the best possible, measured by the Frobenius norm of the error

Latent Semantic Indexing

Application of SVD - LSI

- From term-document matrix A, we compute the approximation A_k.
- There is a row for each term and a column for each document in A_k
- Thus documents live in a space of $k << r$ dimensions (these dimensions are not the original axes)
- Each row and column of A gets mapped into the k-dimensional LSI space, by the SVD.
- Claim – this is not only the mapping with the "best" approximation to A, but in fact improves retrieval.

Latent Semantic Indexing

Application of SVD - LSI

- A query q is also mapped into this space
- ... within this space the query q is compared to all the documents
- ... the document closest to the query are returned as search result

LSI - Results

- Similar terms map to similar location in low dimensional space
- Noise reduction by dimension reduction

Evaluation

Assess the quality of search

Overview: Evaluation

Two basic questions / goals

- Are retrieved documents relevant for the user's information need?
- Have all relevant documents been retrieved?

In practice these two goal contradict each other.

Precision & Recall

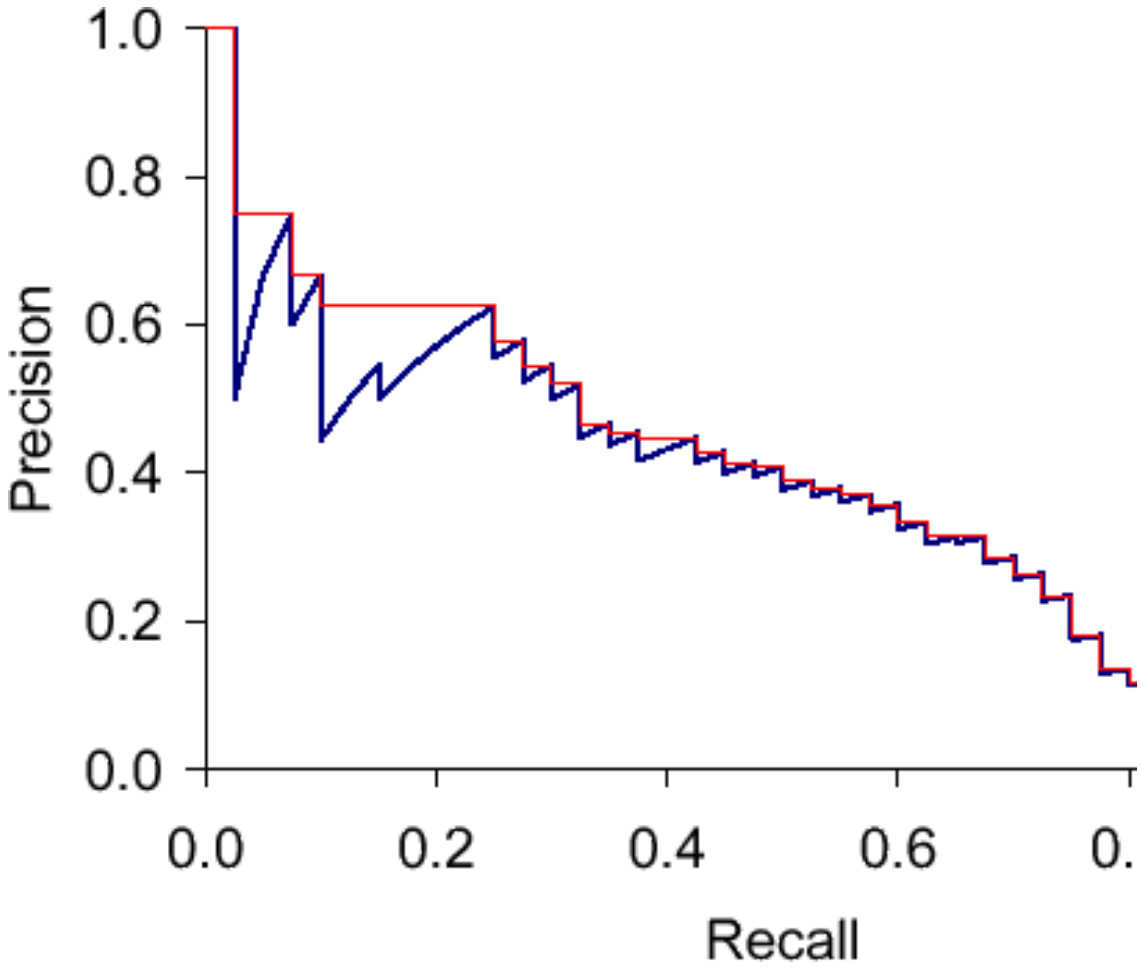
Evaluation Scenario

- Fixed document collection
- Number of queries
- Set of documents marked as relevant for each query

Main Evaluation Metrics

- $Precision = \frac{\#(relevant\ items\ retrieved)}{\#(retrieved\ items)}$
- $Recall = \frac{\#(relevant\ items\ retrieved)}{\#(relevant\ items)}$
- $F_1 = \frac{2PrecisionRecall}{Precision+Recall}$

Precision & Recall



Advanced Evaluation Measures

Mean Average Precision - MAP

- ... mean of all average precision of multiple queries
- $Map = \frac{1}{|Q|} \sum_{q \in Q} AP_q$
- $AP_q = \frac{1}{n} \sum_{k=1}^{n} Precision(k)$

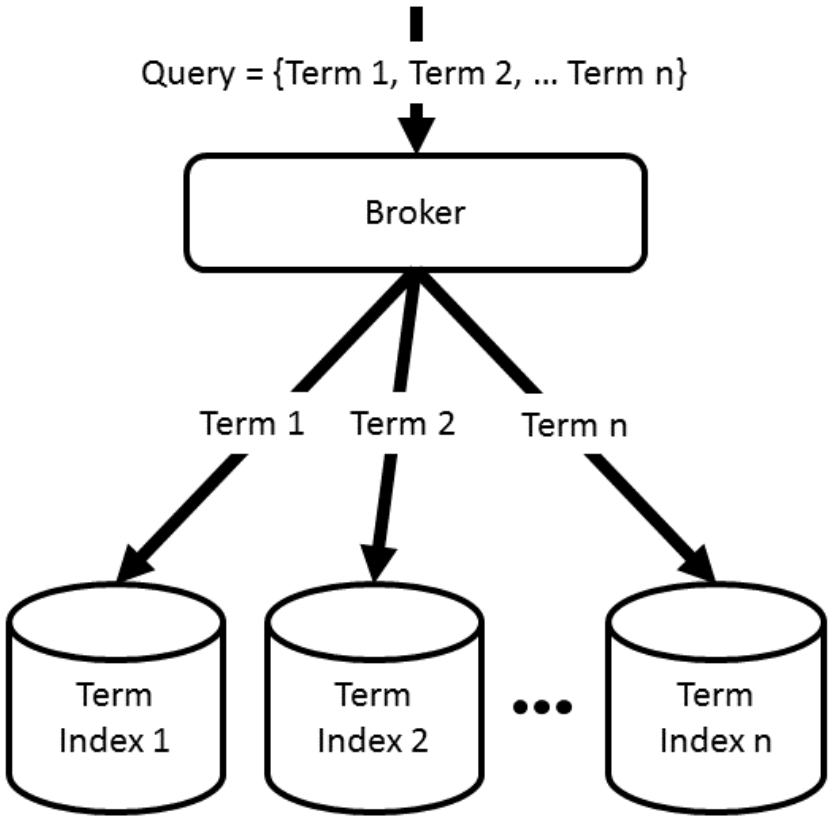
(Normalised) discounted cumulative gain (nDCG)

- ... where there is a reference ranking the relevant results

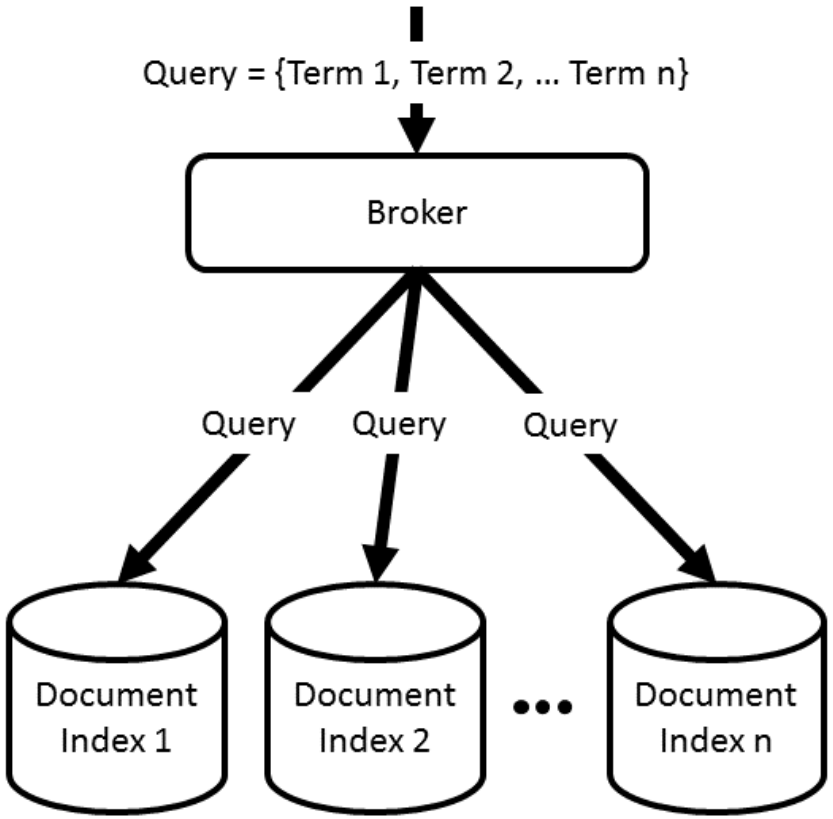
Application & Solutions

Federated Search, Recommender Systems, Open-source, & commercial IR solutions

Distributed Search - Architectures



1 – Term Partitioning



2 – Document Partitioning

Term Based Index Split

Index Creation

- The index is split based on dictionary terms
- Called *Global Inverted Files* or *Global Index Organization*
- Example:
  1. Terms beginning with A to B
  2. Terms beginning with C to G
  3. ...

Properties

- Single index is split over multiple machines
- Each index returns part of its postings to broker
- Broker merges postings to generate result
- $\Rightarrow$ High network traffic, high load at broker

Document Based Index Split

Index Creation

- The index is split based on document categories
- Called *local inverted file*
- Example:
  1. Documents from computer science
  2. Documents from physics
  3. ...

Properties

- Each partition is a full index
- Broker unifies the result lists
- $\Rightarrow$ re-ranking problem at broker

Federated Search

Definition

- Simultaneous search in multiple search engines
- Single user interface or API to access multiple search engine
- Examples:
  1. Google Scholar
  2. Expedia
  3. ...

Properties

- Unified access to multiple search engines
- Each search engines is a full search on its own, not only an index
- $\Rightarrow$ Re-ranking problem at broker
- $\Rightarrow$ Access control

Recommender systems - Overview

- Recommender systems should provide usable suggestion to users
- Recommender systems should show new, unknown–but similar–documents
- Recommender systems can utilize different information sources

  Content-based Recommender
  : Finds similar documents on the basis of document content

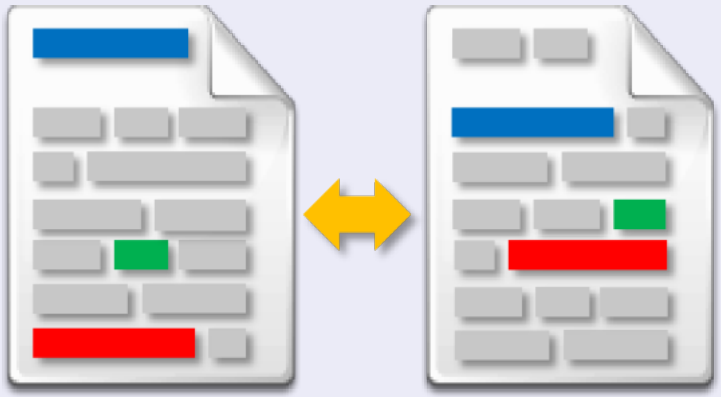
  Collaborative Filering
  : Employs user rating to find new and useful documents

  Knowledge-based Recommender
  : Makes decisions based on pre-programmed rules

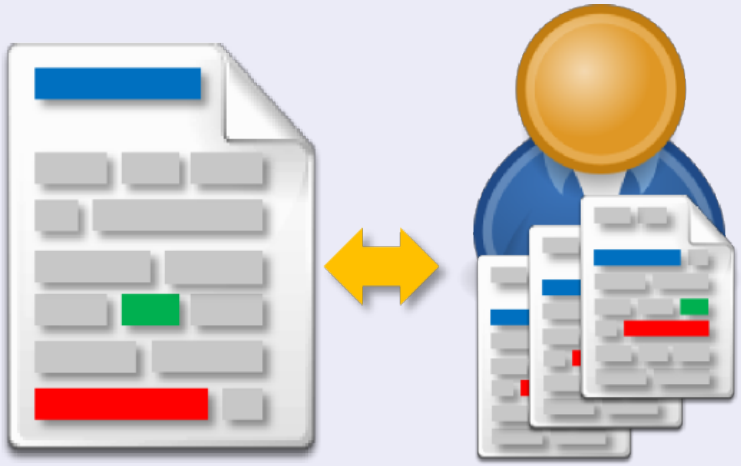
  Hybrid Recommender
  : Combination of two or three other approaches

Types of Content-based Recommender

Document-to-Document Recommender



Document-to-User Recommender



Building a Recommender by Using an Index

Document-to-Document Recommender

1. Take the terms from a document
2. Construct a (weighted) query with them
3. Query the index
4. Results are possible candidates for recommending

Document-to-User Recommender

1. Create a user model from
   - Terms typed by the user
   - Documents viewed by the user
   - $\Rightarrow$ Simplest user model is just a set of terms
2. Construct a (weighted) query with them
3. Query the index
4. Results are possible candidates for recommending

Open Source Search Engine

Apache Lucene

- Very popular, high quality
- Backbone of many other search engines, e.g. Apache Solr (fully), Elasticsearch
  - Lucene is the core search engine library
  - Solr provides a search solution (service & configuration infrastructure, distribution, indexing, etc.)
  - Elasticsearch provides a search solution and document database focusses on distributed
- Implemented in Java, bindings for many other programming languages

Other Open-Source Search Engines

Xapian

- Written in C++, Support for many (programming) languages
- Used by many open-source projects

Terrier

- Many algorithms
- Academic background

Sphinx

- Written in C++
- Offers also features comparable to database

Whoosh

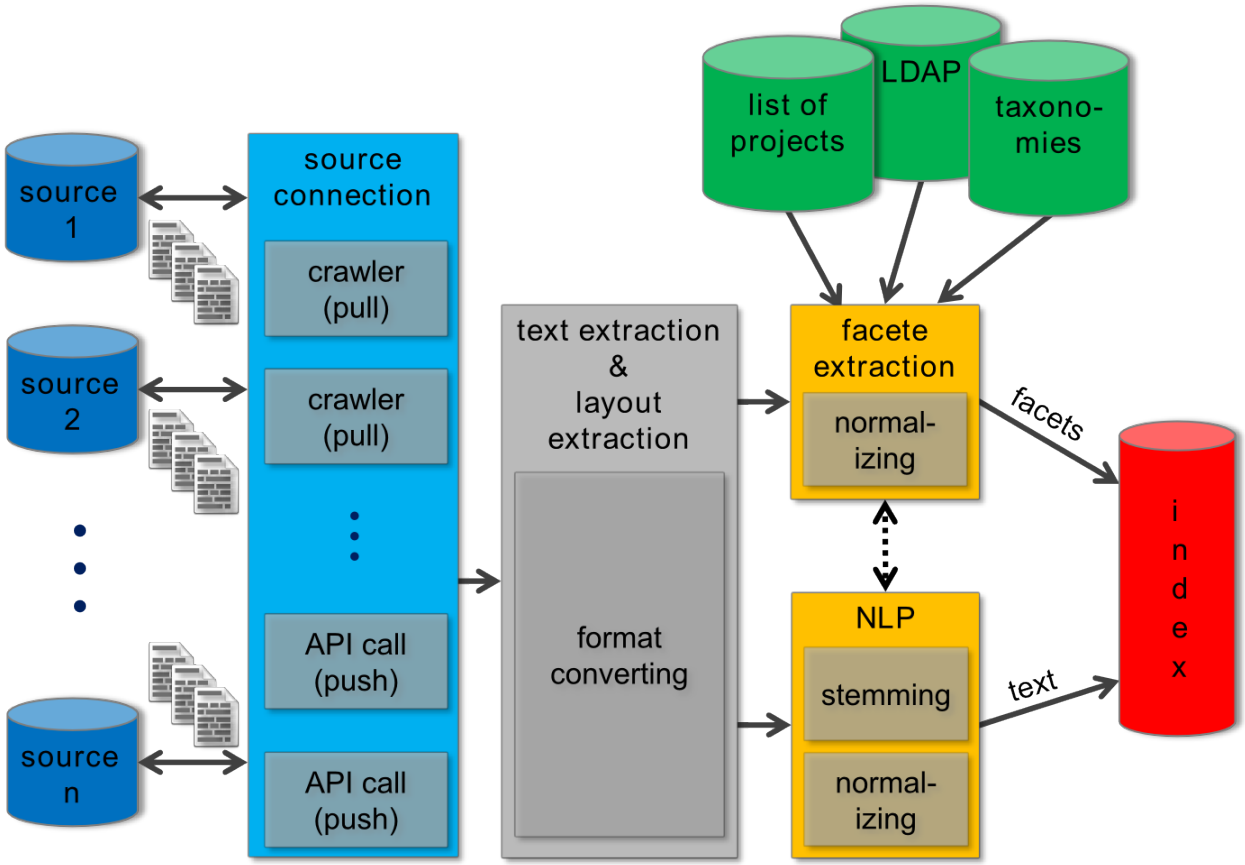
- for Pythonistas

Lemur & Indri

- Language models
- Academic background

What customers want

Indexing pipeline



And also: user interface, access control mechanisms, logging, etc.

Commercial Search Engines

Dassault Exalead

- Dassault acquired the Exalead project
- Offers specialised features for CAD/CAE/CAM files

Sinequa

- Expert Search
- Creates new views, i. e. do not only show result list, but combine results to new unit

Attivio

- Use SQL statements in queries

IntraFind

- Elasticearch-based commercial search solution

And many, many more...

The End

Next: Pattern Mining