Applications of Artificial Intelligence

– Winter Term 21/22 –

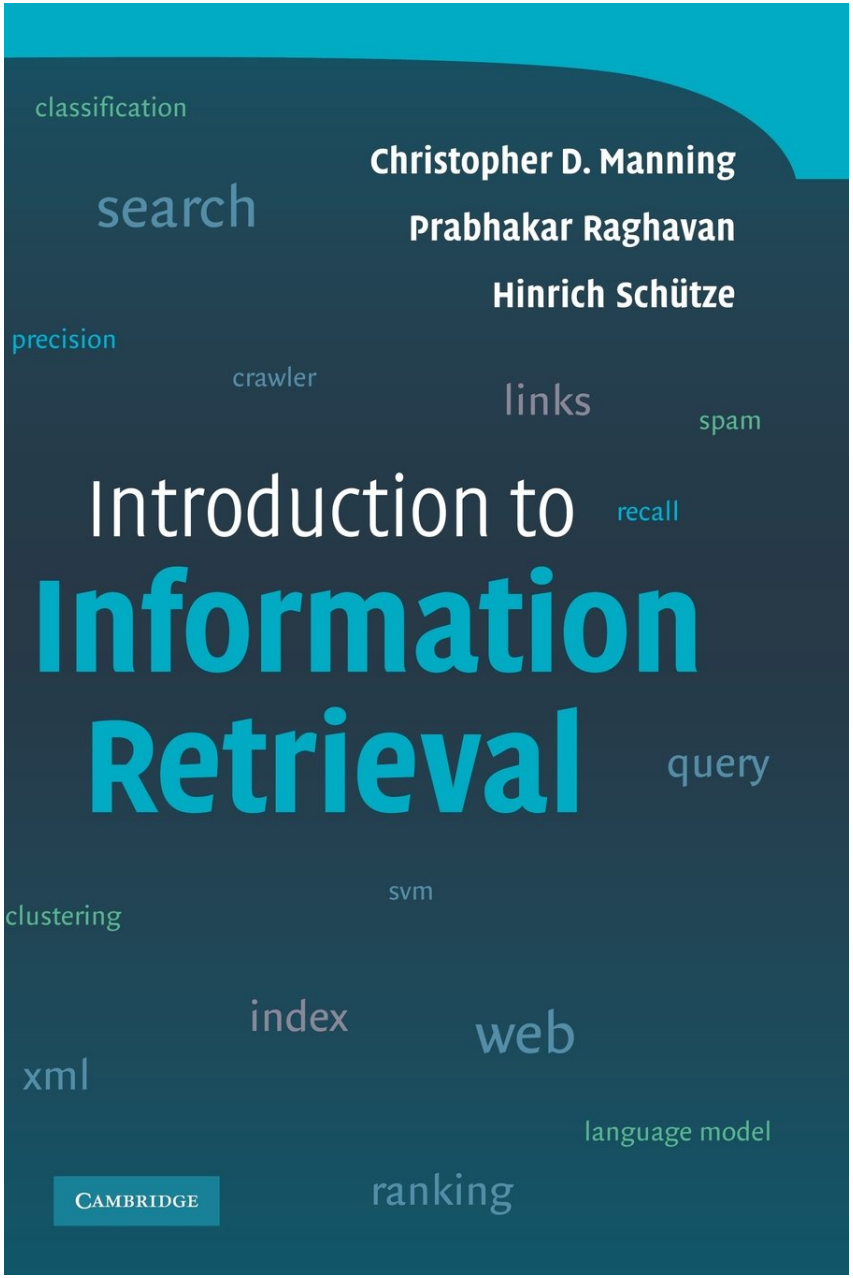
Chapter 02

Information Retrieval

Prof. Dr. Adrian Ulges

RheinMain University of Applied Sciences

Recommended Read [5]

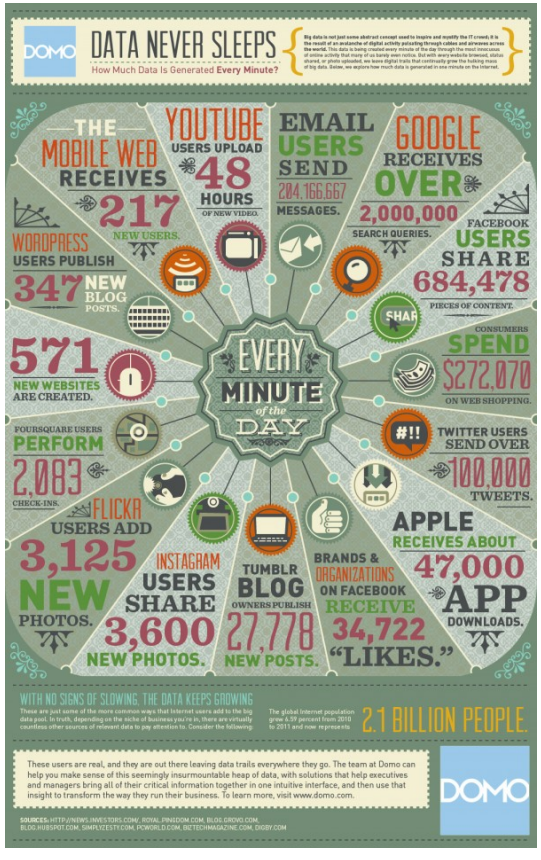


Outline

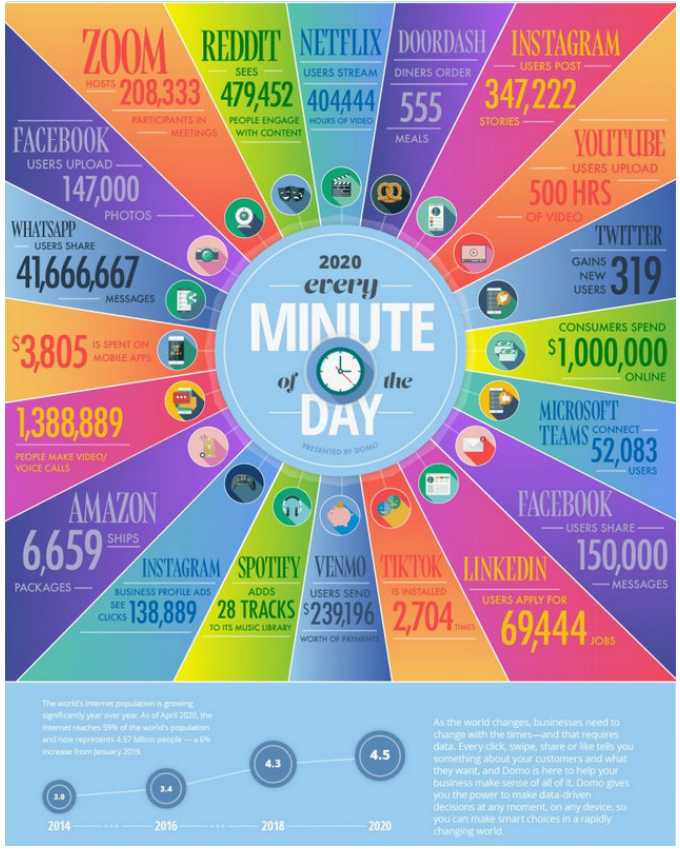
Information Retrieval (IR): Definition images: [4]

"Finding material (typically documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored in computers)"

(Manning, Raghavan, Schütze [5])

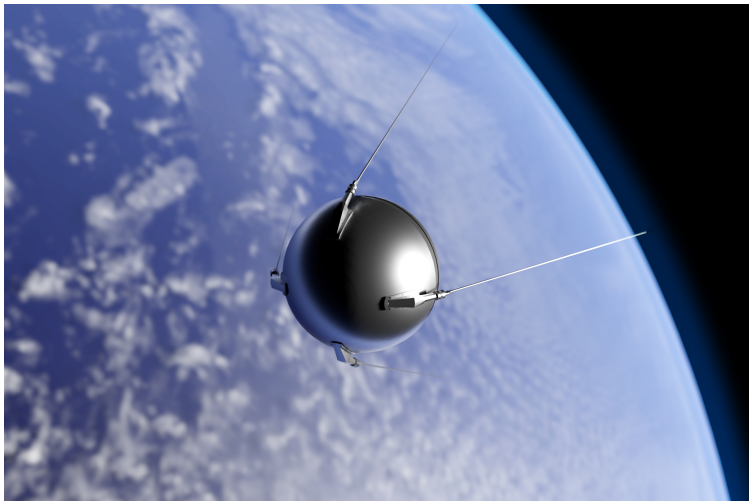


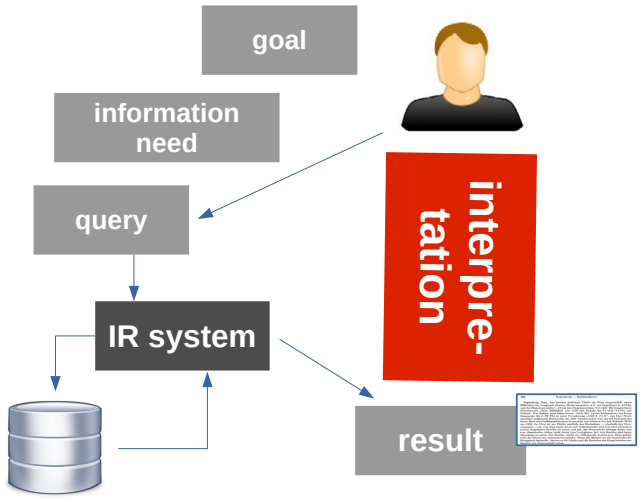
(Jun 2012)



(Sep 2020)

Information Retrieval: Definition

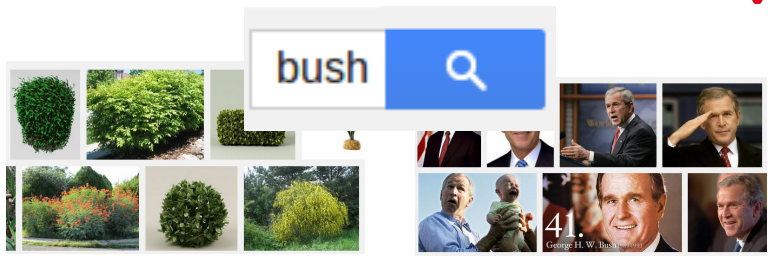




Remarks

- IR as an academic discipline: since 1950s
  (lawyers, librarians, journalists, doctors...)
- since 1990s (www): **key technology** of modern life
- no automated **understanding** of text,
  but **support** of user with search
- heavy use of **text statistics**
- Question to you: is IR a **solved** problem?

IR: Challenges



(A) Unstructured Target Data

Natural language is ambiguous!

	databases	**IR**
query language	formal (e.g., SQL)	natural language
data matching	exact	partial
model	deterministic	probabilistic
data errors	sensitive	insensitive

(B) The Semantic Gap

> *"the lack of coincidence between the information that one can extract from the [...] data and the interpretation that the same data have for a user in a given situation."*
>
> (Smeulders et al. [7])

(A) Unstructured Target Data

Natural language is not well suited for machine processing!

1. Irregular Flexion

> *"I walk / walked / have walked" vs.*
> *"I go / went / have gone"*

- *lemmatization/stemming* groups words by their base form (`fishing`, `fisher`, `fishy` → `fish`)
- **Approach 1**: rule-based *('*ing' → '*')*
- **Approach 2**: Lookup-Tables *(stem['ipads']=ipad)*

2. Synonyms and partial synonyms

> *'car' ↔ 'automobile'*
> *'car' ↔ 'vehicle'*
> *'dog' ↔ 'mutt' (dt. 'Köter')*

- synonyms = different words with identical meaning
- **approach**: *thesauri*

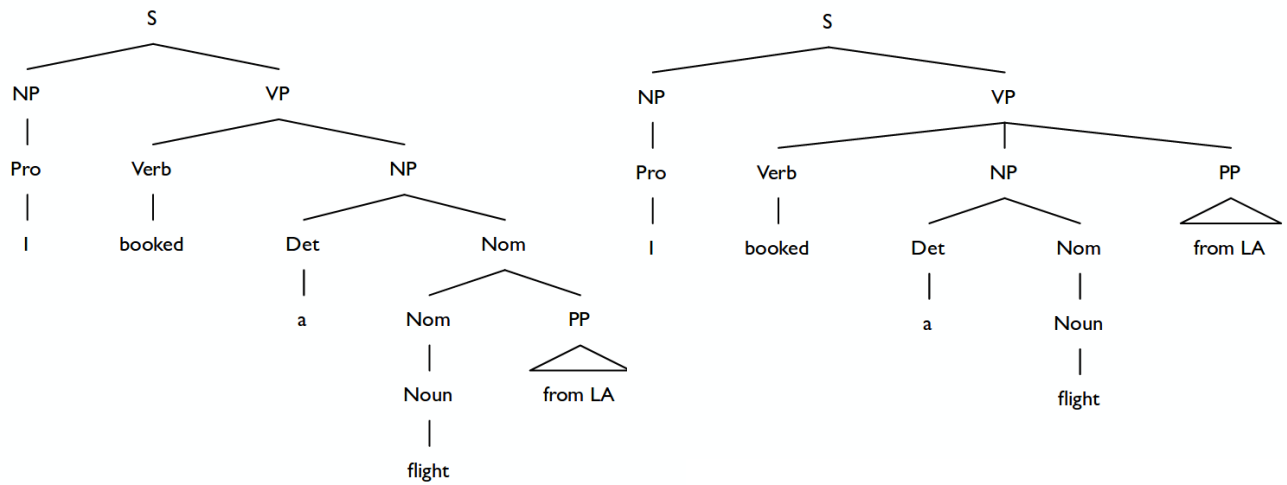
(A) Unstructured Target Data

3. Homonyms

"The spirit is willing but the flesh is weak" → Russian → "The wodka is good but the steak is lousy".

- homonym = same word, different meanings
- **approaches**: thesauri, context disambiguation *(machine learning, later)*.

4. Ambiguous Syntax



- **approach**: probabilistic parsing *(machine learning, later)*.

(A) Unstructured Target Data

5. Coreferences

Peter used to love his dog Bono.
He got lost recently, though. vs.
He used to take him for walks in the park.

- **Approach**: Coreference Resolution *(machine learning, later)*.

6. ...

(B) The Semantic Gap: Example

(B) The Semantic Gap: Consequences

The **interpretation** *(and thus the usefulness)* of a search result is **user-** and **context dependent**!

The IR system does not know...

- ... the user's intent *('things to do in Paris')*
- ... the user's knowledge level *('SVMs')*

The user may not know ...

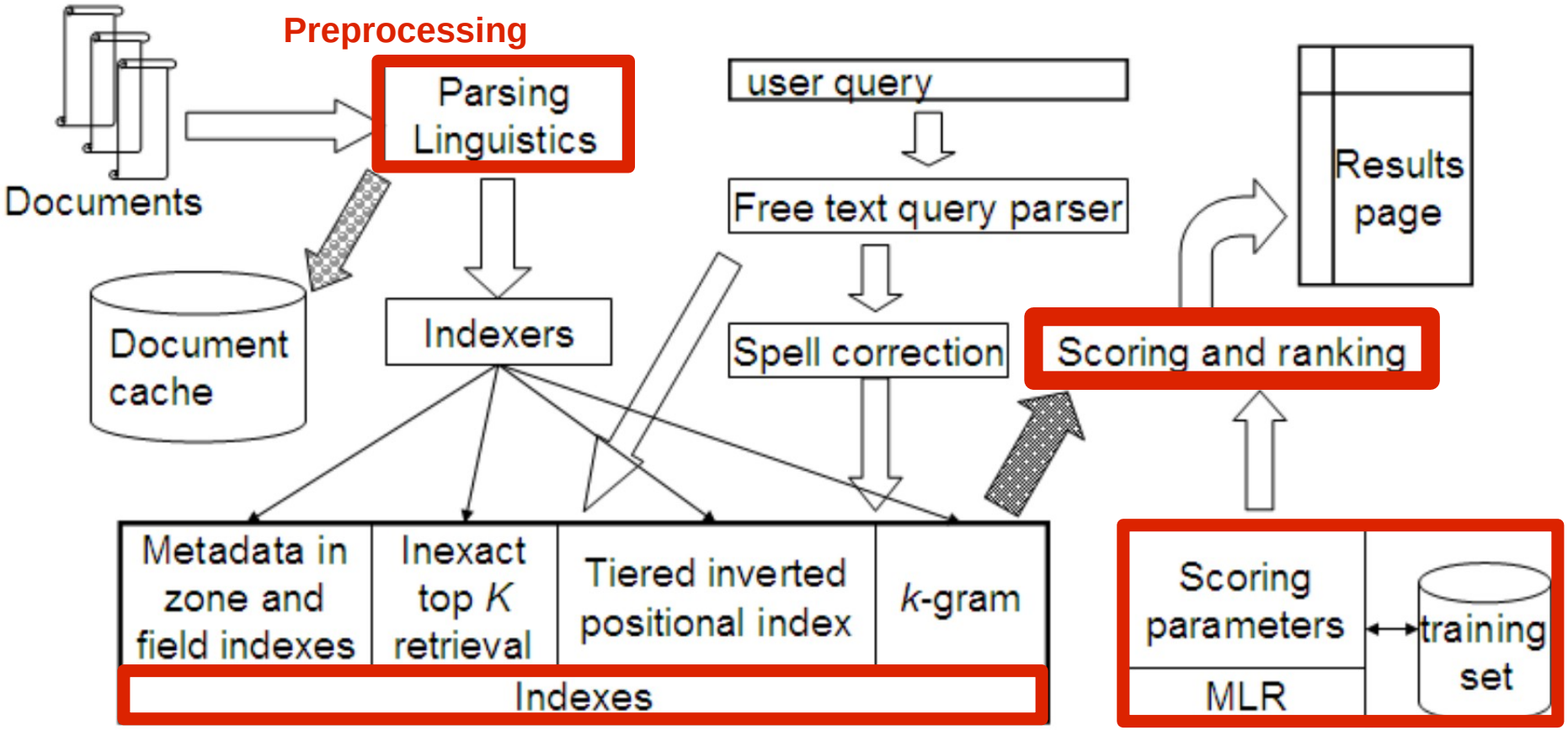
- the domain vocabulary *('classifier combination?')*
- the corpus *('headache treatment' → 0 hits? 13428 hits?)*

Consequence

- Choosing **the "right" query** can be difficult.

Outline

Information Retrieval: Setup image: [5]



Preprocessing

- IR usually operates on the basis of words *(terms, tokens)*.
- We segment the input strings into sequences of tokens.
- Sometimes this is **non-trivial** *(does a dot mark the end of a sentence, or a date or abbreviation?)*
- **Approach**: use **regular expressions** *(later)*.

Operator	Behavior
`.`	Wildcard, matches any character
`^abc`	Matches some pattern *abc* at the start of a string
`abc$`	Matches some pattern *abc* at the end of a string
`[abc]`	Matches one of a set of characters
`[A-Z0-9]`	Matches one of a range of characters
`ed|ing|s`	Matches one of the specified strings (disjunction)
`*`	Zero or more of previous item, e.g., `a*`, `[a-z]*` (also known as *Kleene Closure*)
`+`	One or more of previous item, e.g., `a+`, `[a-z]+`
`?`	Zero or one of the previous item (i.e., optional), e.g., `a?`, `[a-z]?`
`{n}`	Exactly *n* repeats where *n* is a non-negative integer
`{n,}`	At least *n* repeats
`{,n}`	No more than *n* repeats
`{m,n}`	At least *m* and no more than *n* repeats
`a(b|c)+`	Parentheses that indicate the scope of the operators

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Preprocessing (cont'd)

Stemming

- We also reduce words to their stem:

```
> import nltk
> tokens = ['women', 'swords', 'is',
            'lying', 'candle']
> # rule-based standard stemming
> [nltk.PorterStemmer().stem(t) for t in tokens]

  ['women', 'sword', 'is', 'lie', 'candl']

```

Retrieval Models

- Mathematical **retrieval models** estimate the **relevance** of a **document** d, given a **query** q.
- The model computes a **score** $s(q, d) \in \mathbb{R}$.
- We use the score to **rank** documents: The higher the score, the earlier a document appears in the hitlist.
- We will look at three particular scoring models:
  1. boolean retrieval
  2. the vector space model
  3. probabilistic retrieval

q = „soccer world cup“

Lorem ipsum dolor sit amet, consetetur sadipscing elitr,

$s(\mathbf{q},\mathbf{d}_1) = 1.6 \rightarrow \text{rank}=2$

Stet clita kasd guber-gren, no sea takimata sanctus est Lorem.

$s(\mathbf{q},\mathbf{d}_2) = 3.9 \rightarrow \text{rank}=1$

At vero eos et accu-sam et jo duo dolores et ea rebum.

$s(\mathbf{q},\mathbf{d}_3) = -0.5 \rightarrow \text{rank}=3$

Boolean Retrieval

- simplest and oldest model *(1960s)*
- **documents** are **sets of tokens** over a vocabulary V.
- the set of all **queries** Q is defined by induction:
  1. $\forall t \in V : t \in Q$ (single tokens t are queries)
  2. $\forall q \in Q : \neg q \in Q$ (queries can be negated)
  3. $\forall q_1, q_2 \in Q : q_1 \wedge q_2 \in Q$ and $q_1 \vee q_2 \in Q$ (AND/OR connections)
- **scoring** works like in propositional logic *(dt. Aussagenlogik)*:
  1. $s(t, d) = 1_{t \in d}$
  2. $s(\neg t, d) = 1 - s(t, D)$
  3. $s(q_1 \wedge q_2, d) = min\Big(s(q_1, d), s(q_2, d)\Big)$
  4. $s(q_1 \vee q_2, d) = max\Big(s(q_1, d), s(q_2, d)\Big)$
- remark: $\forall q \in Q : s(q, d) \in \{0, 1\}$
- remark: The **order** of terms in the document does not matter.

Boolesches Retrieval: Beispiel

Query: "sand $\wedge\neg$ prince"

They struck out north by northwest, across drylands and parched plains and pale **sands** toward Ghost Hill, the stronghold of House Toland, where the ship that would take them across the Sea of Dorne awaited them. "Send a raven whenever you have news," Prince Doran told her, "but report only what you know to be true. We are lost in fog here, besieged by rumors, falsehoods, and traveler's tales. I dare not act until I know for a certainty what is happening." War is happening, though Arianne, and this time Dorne will not be spared. "Doom and death are coming," Ellaria **Sand** had warned them, before she took her own leave from Doran. "It is time for my little snakes to scatter, the better to survive the carnage." Ellaria was returning to her father's seat at Hellholt. With her went her daughter Loreza, who had just turned ...

The king's voice was choked with anger. "You are a worse pirate than Salladhor Saan." Prince Theon Greyjoy opened his eyes. His shoulders were on fire and he could not move his hands. For half a heartbeat he feared he was back in his old cell under the Dreadfort, that the jumble of memories inside his head was no more than the residue of some fever dream. I was asleep, he realized. That, or passed out from the pain. When he tried to move, he swung from side to side, his back scraping against stone. The prince was hanging from a wall, his wrists chained to a pair of rusted iron rings. The air reeked of burning peat. The floor was hard-packed dirt. Wooden steps spiraled up inside the walls to the roof. He saw no windows. The tower was dank, dark, and comfortless, its only furnishings a high-backed chair and a scarred table resting on three trestles. No privy was in evidence, though Theon saw a champerpot in one shadowed alcove. The only light came from the candles on the table. His feet dangled six feet off the floor. ...

$s(\text{sand} \wedge\neg \text{prince}, Doc_1) = 0$ $s(\text{sand} \wedge\neg \text{prince}, Doc_2) = 0$

Discussion

- boolean models tend to yield either too few or too many hits (*"feast or famine"*). ☹
- **Example**:
  1. *drugs (19248 hits)*
  2. *drugs* $\wedge$ *medical (2412 hits)*
  3. *drugs* $\wedge\neg$ *aging (19119 hits)*
  4. *#2* $\wedge$ *#3 (2349 hits)*
  5. *#4* $\wedge$ *memory (6 hits)*
- No prioritization of key terms. ☹
- Key advantage: **transparency** ("user in control") ☺
  → still used in practice *(e.g., WestLaw, USA, 700,000 users)*.

The Vector Space Model

History

- the most common IR model *(1970s - today)*.
- many implementations available *(Lucene, Elasticsearch, ...)*

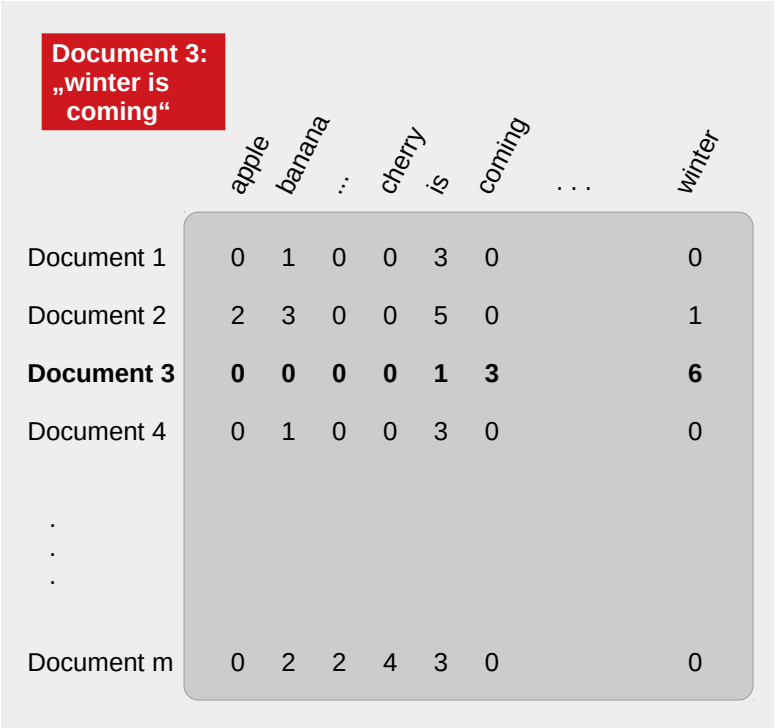
Formalization

- given: a **vocabulary** $V = \{t_1, ..., t_n\}$ of known tokens.
- we represent queries as vectors q $= (q_1, ..., q_n) \in \mathbb{R}^n$ and documents as vectors d $= (d_1, ..., d_n) \in \mathbb{R}^n$.
- the vectors contain **weights** q_i/d_i indicating how **important** token t_i is for the query/document.

The Vector Space Model

The Term Document Matrix

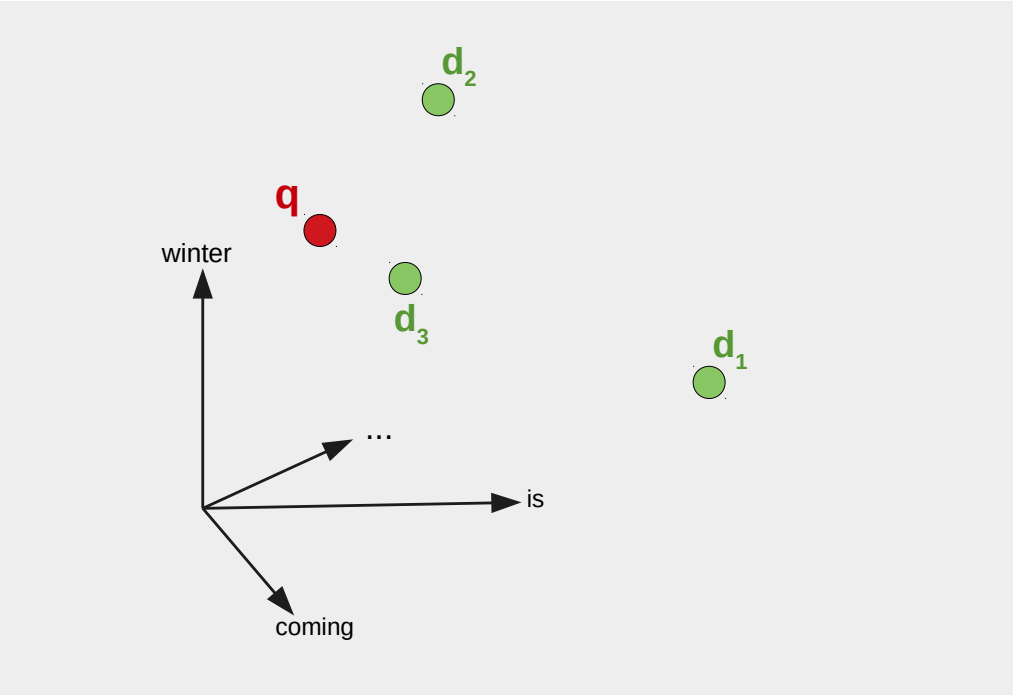
- For each document in the collection $D = \{d_1, ..., d_m\}$, we obtain a vector.
- By stacking all documents' vectors, we obtain the **term document matrix** $M \in \mathbb{R}^{m \times n}$.
- Every **row** represents a document.
- $M_{ij} \neq 0$ holds exactly if t_j **appears** in Document d_i.
- M ist **huge**, but very **sparse** (= most entries are zeros).



The Vector Space Model

Geometric Interpretation

- We can view documents and queries as **points** in a **high-dimensional (term) space**.
- Scoring means to compute a **similarity measure** between q and d.
- What would be a "good" similarity measure?

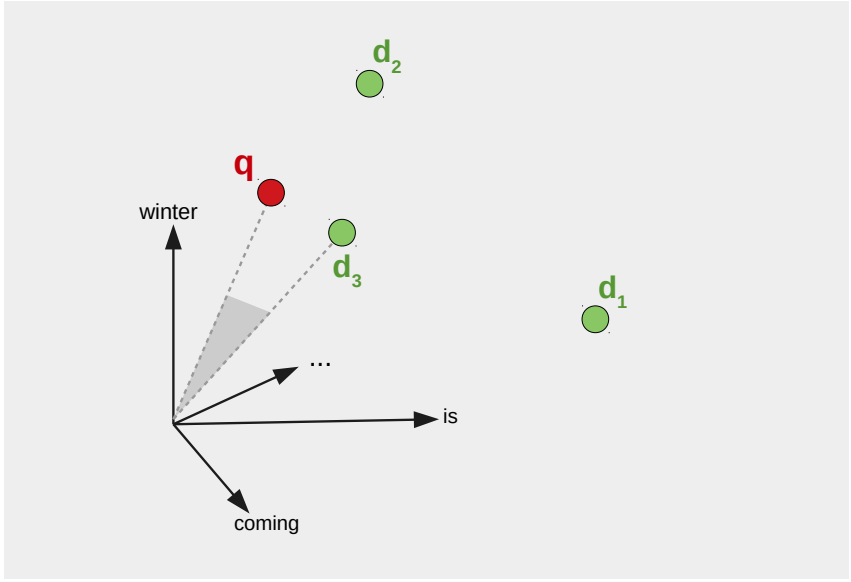


The Vector Space Model: Scoring

- why not use Euclidean distance?
  $\rightarrow$ to avoid a bias towards *shorter documents*.
- instead, we compute the cosine of the **angle** between q and d *(using the scalar product)*:

$$s(q,d) := cos\Big(\sphericalangle(q,d)\Big) = \frac{< q,d >}{||q||_2 \cdot ||d||_2}$$

- **remark**: $s(q,d)$ is **normalized** w.r.t document length!



The Vector Space Model: Term Weights

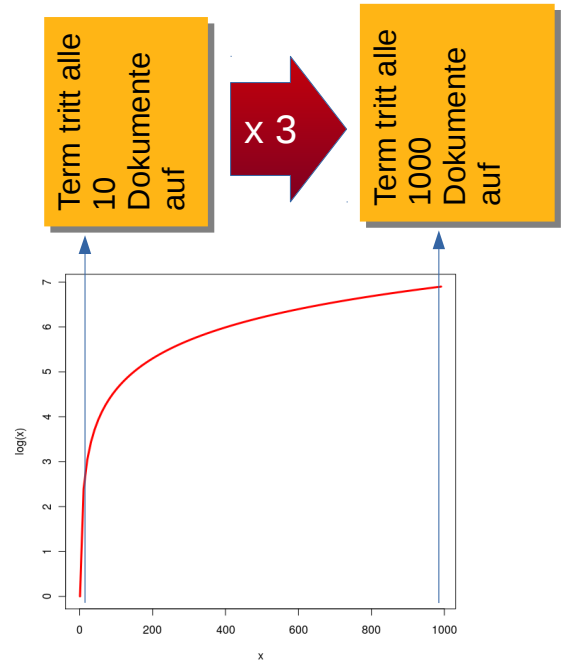
Key Question: How do we choose **the vectors' entries** q_i, d_i?
(they're supposed to model the importance of terms!)

1. **boolean**: $d_i = 1_{\,t_i \text{ appears in } d}$
2. **term frequency (TF)**: $d_i = tf(t_i, d)$
   $= \#\text{mentions of } t_i \text{ in } d$
3. **term frequency (TF), smoothed**:

$$d_i = \begin{cases} 1 + log_{10}\Big(tf(t_i, D)\Big) & \text{if } t_i \text{ appears in } d \\ 0 & \text{else.} \end{cases}$$

(this would mean: a word that appears $10\times$ *is twice as "important" as a term that appears once).*

The Vector Space Model: Term Weights



4. TF-IDF

- **idea**: A word is important if it is rare in the overall corpus, but frequent in the document.
- **example**: Search for "jesus resurrection" in the Bible's new testament.
- we define a term's **inverse document frequency (IDF)** *(the rarer the term, the higher!)* as ...

$$idf(t) := log\left(\frac{\#D}{\#\{d \in D \mid d \text{ contains } t\}}\right)$$

- ... and compute the term weight as the **product of tf and idf**:

$$d_i \;:=\; tf(t_i, d) \;\times\; idf(t_i)$$

The Vector Space Model: Term Weights

Remarks (TF-IDF)

- $\forall t : idf(t) \geq 0$ (idf is non-negative).
- tf-idf weighting can also be applied on the query, but is mostly used on documents.

The Vector Space Model: Example

Query: "sand prince"

They struck out north by northwest, across drylands and parched plains and pale sands toward Ghost Hill, the stronghold of House Toland, where the ship that would take them across the Sea of Dorne awaited them. "Send a raven whenever you have news," Prince Doran told her, "but report only what you know to be true. We are lost in fog here, besieged by rumors, falsehoods, and traveler's tales. I dare not act until I know for a certainty what is happening." War is happening, though Arianne, and this time Dorne will not be spared. "Doom and death are coming," Ellaria Sand had warned them, before she took her own leave from Doran. "It is time for my little snakes to scatter, the better to survive the carnage." Ellaria was returning to her father's seat at Hellholt. With her went her daughter Loreza, who had just turned ...

The king's voice was choked with anger. "You are a worse pirate than Salladhor Saan." Prince Theon Greyjoy opened his eyes. His shoulders were on fire and he could not move his hands. For half a heartbeat he feared he was back in his old cell under the Dreadfort, that the jumble of memories inside his head was no more than the residue of some fever dream. I was asleep, he realized. That, or passed out from the pain. When he tried to move, he swung from side to side, his back scraping against stone. The prince was hanging from a wall, his wrists chained to a pair of rusted iron rings. The air reeked of burning peat. The floor was hard-packed dirt. Wooden steps spiraled up inside the walls to the roof. He saw no windows. The tower was dank, dark, and comfortless, its only furnishings a high-backed chair and a scarred table resting on three trestles. No privy was in evidence, though Theon saw a champerpot in one shadowed alcove. The only light came from the candles on the table. His feet dangled six feet off the floor. ...

- Let Document 1 contain *prince* 1× and *sand* 10×.
  Let Document 2 contain *prince* 10× and *sand* 0×.
- Let *prince* appear in every 10th document.
  Let *sand* appear in every 100th document.

$$q = \left(\ \ldots \quad 1/\sqrt{2} \quad \ldots \quad 1/\sqrt{2} \quad \ldots \ \right) \ // \text{ tf, normalized}$$

$$d_1 = \left(\ \ldots \quad 1 \cdot log_{10}(10) \quad \ldots \quad 10 \cdot log_{10}(100) \quad \ldots \ \right) \ // \text{ tf-idf } (log_{10})$$

$$d_2 = \left(\ \ldots \quad 10 \cdot log_{10}(10) \quad \ldots \quad 0 \cdot log_{10}(100) \quad \ldots \ \right) \ // \text{ tf-idf } (log_{10})$$

$$s(q, d_1) =< q, d_1 >= \frac{21}{\sqrt{2}} = 14.9 \qquad s(q, d_2) =< q, d_2 >= \frac{10}{\sqrt{2}} = 7.1$$

The Vector Space Model: Term Weights

State-of-the-Art Scoring Model: Okapi-BM25

$$s(\mathrm{q},\mathrm{d}) := \sum_{t \in \mathrm{q}} idf(t) \cdot \frac{(k_1+1) \cdot tf(t,D)}{k_1((1-b)+b \cdot L_d/L_{avg})) + tf(t,D)}$$

k_1 defines the **importance of term frequency**:

- for $k_1 \to \infty$ (and B=0), the model appoximates tf-idf.
- for $k_1 \to 0$, the model appoximates pure idf (tf is boolean).

b defines the **bias towards short documents**:

- L_{avg} denotes the corpus' average document length.
- for B>0, long documents are "punished".
- the higher B, the stronger this punishment.

Default values are k_1=1.2 and b=0.75,
typical ranges are $k_1 \in [0,3]$ and $b \in [0,1]$ [3].

Vector Space Model: Discussion

- **good quality** of retrieval results
  (according to benchmarking)
- choice of term weights is **intuitive**
  (but based on heuristics)
- loss of **word order** *(solutions: later)*
- no **"semantic similarity"**: Documents with the same topic but different wording are not found.

Outline

The Cost of Searching

Remember our scoring function?

$$s(\mathrm{q},\mathrm{d}) := cos\Big(\sphericalangle(\mathrm{q},\mathrm{d})\Big) = \frac{<\mathrm{q},\mathrm{d}>}{||\mathrm{q}||_2 \cdot ||\mathrm{d}||_2}$$

Naive Approach I: Compute scalar products

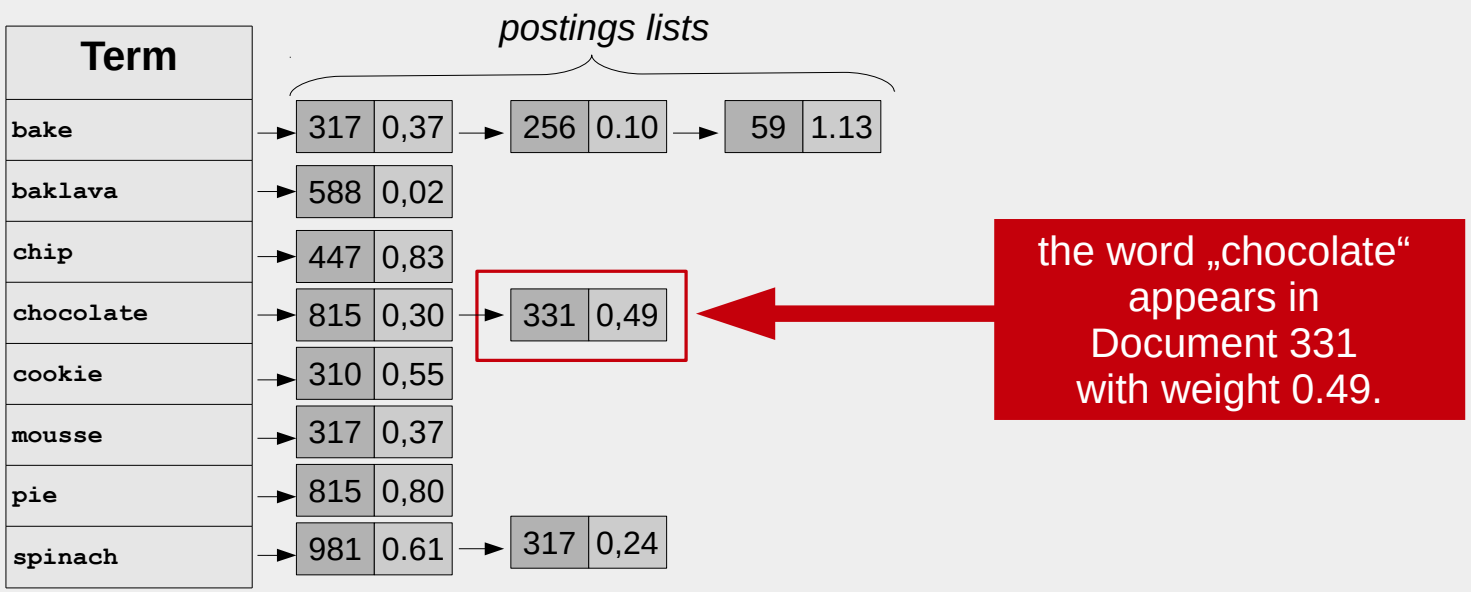
- Computing $s(q,d)$ once? $\rightarrow O(n)$ (where $n = \#$vocabulary)
- Computing $s(q,d_1), s(q,d_2), ..., s(q,d_m)$? $\rightarrow O(n \cdot m)$ ☹
- sorting documents by relevance and selecting the top k documents? $\rightarrow O(m + k \cdot log(m))$ *(using Heapsort, negligible)*

Doing better?

- Key observation: q and d are very sparse!
- <q, d> can be computed only over the words appearing in **both** q **and** d!
- Data Structure: **inverted file** *(aka. inverted index)*.

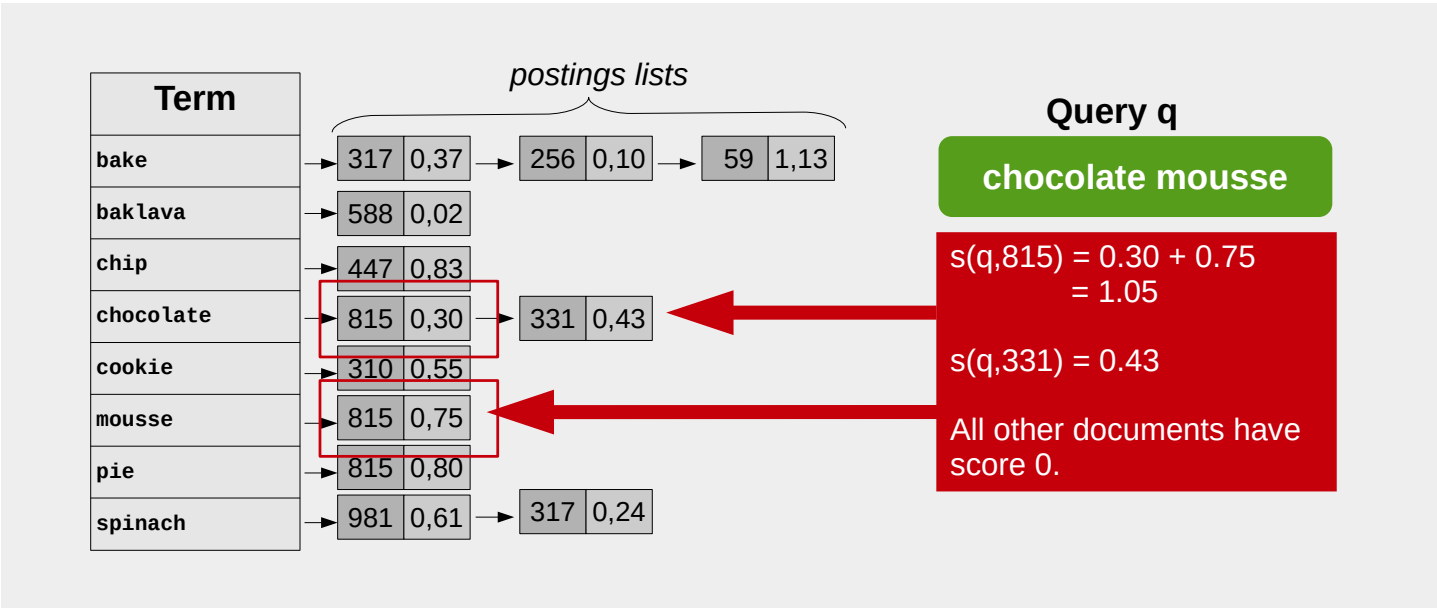
Inverted Files

- ... exploit the **sparseness** of query and document vectors
  (most documents contain no query term → score = 0).
- ... store a **list (aka. postings list)** for each term t, containing
  (1) the documents d containing t and
  (2) the corresponding weights d_i.



Inverted Files: Scoring

1. for each **document** d: set $s(q,d) := 0$
2. for every **term** t_i in the **query**: traverse the list $L(t_i)$
   - for every **entry** (d, d_i) in $L(t_i)$: $s(q,d) \mathrel{+}= q_i \cdot d_i$
4. **rank** the documents by their score $s(q,.)$.



Remarks

- This scoring can be adapted for **any** common **scoring model** (boolean, vectorspace, probabilistic, n-grams, ...)

Tricks of the Trade: Speed-up

Impact Ordering [1]

- sort the **postings lists** by descending weight d_i
- sort the query's **terms** by descending $idf(t_i)$
- abort scoring early (with a "good" **approximation** to the true scores).

Stop Word Filtering

- stop words ("and", "is", ...) would use a vast part of search time and index storage → filter them.

[1] see also ElasticSearch's *common terms query*.

Tricks of the Trade: Phrase Queries

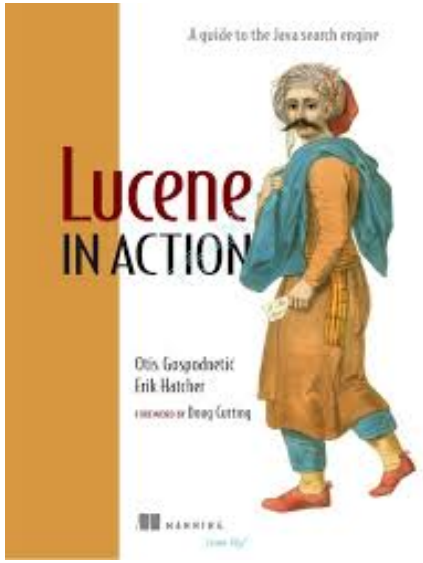
Goal: if the user queries with a phrase, take **word order** into account *("cat lady" vs "lady cat")*.

1. N-Gram Index (cmp. ElasticSearch's *shingle token filter*)
   - An **n-gram** *(or "shingle")* is a sequence of *n* subsequent words
   - n-grams are usually treated **like terms** and obtain **separate postings lists**
   - often: filtering of non-nouns beforehand

     "cost overruns on a power plant"
     → *(cost overruns), (overruns, power), (power,plant)*
2. Positional Index
   - for every word occurrence, store **word and sentence number** *(→ multiple entries per term+document)*
   - enables **proximity queries** (`'bush' NEAR(3) garden'`)
   - often: **combination** of both approaches *(bigram index for very frequent phrases)*.

Outline

Lucene is ...



- ... a **library** for text search
- ... **open-source** (Apache since 2005)
- ... established in practical applications *(linkedin, twitter, ...)* [2]
- ... written in Java, with interfaces to C/C++, C#, Python, PHP, ...
- ... no search engine, but a **library** to build your **own search applications**
- ... the basis for Enterprise Search Engines such as **ElasticSearch**[3]

[2] http://wiki.apache.org/lucene-java/PoweredBy
[3] http://solr-vs-elasticsearch.com

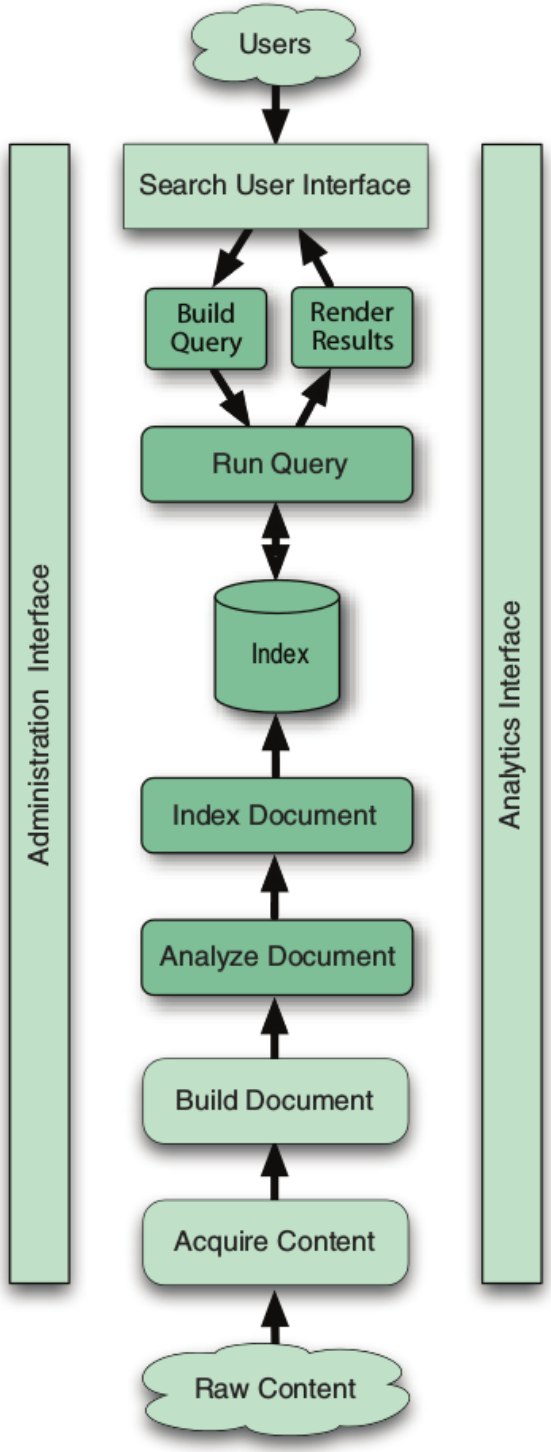
Lucene: Scope image: [6]

Lucene implements IR core steps

- scoring (→ vector space model)
- preprocessing (→ tokenization, stemming)
- indexing (→ inverted files)
- serialization
- query parsing

Lucene is configurable!

- Your own scoring function?
  → override *Similarity* class.
- Your own preprocessing?
  → override *Analyzer* class.



Lucene: Scoring (conceptually)[4]

$$score(q,d) = \underbrace{\left(\frac{< q, d >}{||q||_2} \cdot norm(d)\right)}_{\approx cos\left(\sphericalangle(q,d)\right)} \quad \ldots$$

$$\ldots \quad \times docBoost(d) \times queryBoost(q) \times coord(q,d)$$

- We do not divide by $||d||_2$, but use a separate function $norm(d)$ *(which is configurable)*.
- $docBoost(d)$: can be used to assign an importance to certain documents, e.g. their PageRank *(later)*.
- $queryBoost(q)$: can be used to weigh the query's tokens.
- $coord(q,d)$: the percentage of query tokens found in the document.

[4] further information in the ElasticSearch-Doku

Lucene: Scoring (in practice)

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \ldots$$
$$\ldots \times \sum_{t \in q} tf(t,d) \cdot (idf(t))^2 \cdot queryBoost(t) \cdot norm(t,d)$$

- like above, but scalar product replaced with sum over query terms.
- Basic model: tf-idf *(idf squared)*
- coord, queryBoost, queryNorm: see above
- The factor $norm(t,d)$ contains:
  1. docBoost
     (see above)
  2. The length normalization $1/||\mathrm{d}||_2$
     (with respect to the document's current `(Field)` *)*
  3. fieldBoost
     (fields can be assigned a weight!)

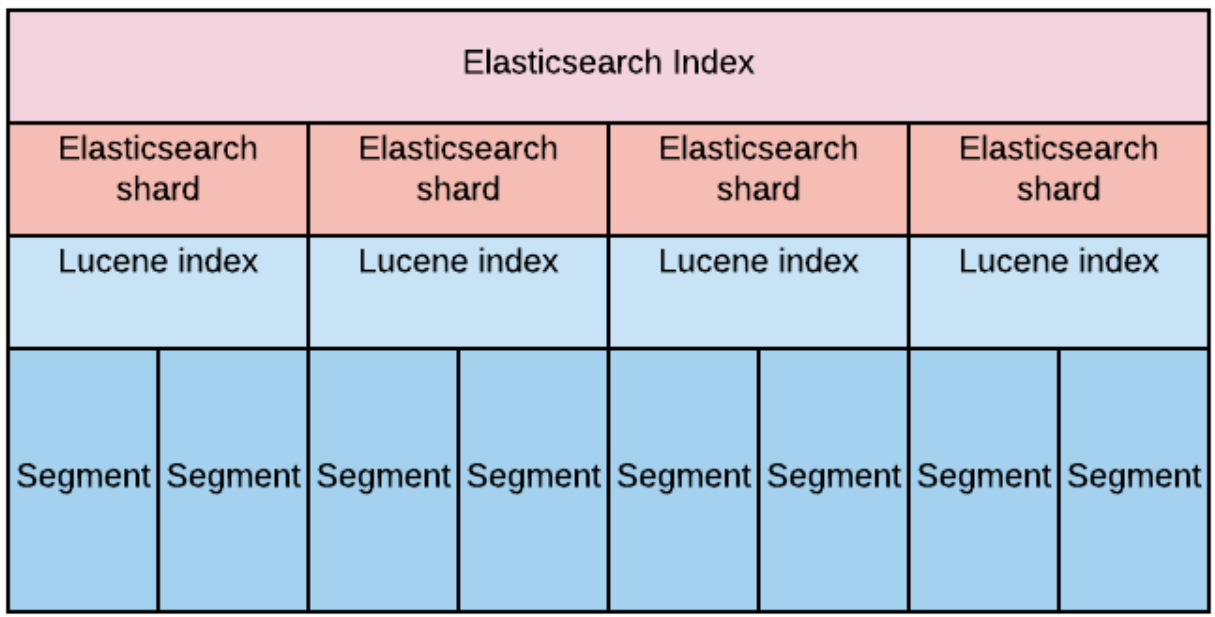
ElasticSearch

- a distributed web service (“enterprise search engine”) for **scalable text search**
- Development by company **ElasticSearch BV**
- open-source *(Apache license)*
- clients in many languages *(incl. Python)*
- operates on basis of Lucene
- most popular enterprise search solution world-wide [1]

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
2.	2.	2.	Splunk	Search engine	87.90	-2.01	+0.89
3.	3.	3.	Solr	Search engine	51.62	-0.08	-7.35
4.	4.	4.	MarkLogic	Multi-model	11.94	-0.28	-1.48
5.	↑ 7.	↑ 7.	Algolia	Search engine	6.77	+0.63	+2.24
6.	↓ 5.	6.	Microsoft Azure Search	Search engine	6.70	0.00	+0.75

ElasticSearch image: [2]

- adds **scalability aspects** on top of Lucene:
  - data distribution to multiple shards
  - shards are replicable (fail safety)
  - **routing** of requests
  - monitoring of cluster state
- add-on **Kibana**: analytics and visualization
- add-on **LogStash**: log file parsing



ElasticSearch image: [2]

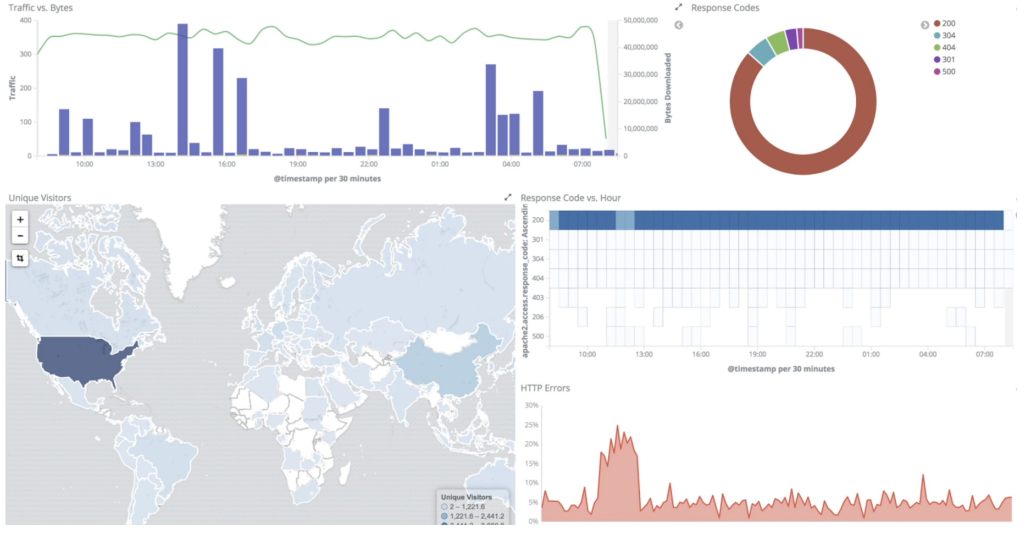
Usage via REST

... is offered via a JSON-based REST API (default port = 9200)

```
1 curl localhost:9200/my_index/my_doctype/_search?q=trump
```

Kibana (`http://localhost:5601`)

- statistics+visualizations on documents and performance
- you can test queries, store queries, filter documents



ES in Python: Creating an index image: [2]

When creating the index, we define the **shard structure** and the **document types**.

```
1  import elasticsearch
2  es = elasticsearch.Elasticsearch()
3
4  settings = {
5      "number_of_shards" : 1,     # sharding
6      "number_of_replicas" : 2
7  }
8
9  mappings = {
10     "properties" : {
11         "house" : { "type":"text", "index":"false" },
12         "words" : { "type":"text", "index":"true" }
13     }
14 }
15
16 es.indices.create(index="my_index",
17                    settings=settings,
18                    mappings=mappings)
```

ES in Python: Adding documents image: [2]

Documents are **dictionaries containing fields** (key-value pairs).

```
1  doc1 = {
2    "house" : "Stark",
3    "words" : "Winter is coming"
4  }
5  doc2 = {
6    "house" : "Greyjoy",
7    "words" : "We do not sow"
8  }
9  doc3 = {
10   "house" : "Baratheon",
11   "words" : "Ours is the fury"
12 }
13
14 # add documents to index
15 es.index(index="my_index", document=doc1)
16 es.index(index="my_index", document=doc2)
17 es.index(index="my_index", document=doc3)
18
19 es.indices.flush()     # persist changes (memory -> disk)
20 es.indices.refresh()   # makes changes available for search
```

ES in Python: Searching image: [2]

Queries can e.g. be defined using Elastic's JSON DSL *(domain-specific language)*

```
query = {
    "match" : {
        "words" : {
            "query" : "winter is", # query terms
            "operator" : "or",     # match >= 1 terms
            "fuzziness" : 0,       # tolerance: 1 char
        }
    }
}
```

We run the query and print the result list:

```
result = es.search(index="my_index", size=10, body=my_body)

for hit in result["hits"]["hits"]:
        score,doc = hit["_score"],hit["_source"]
        print(score, doc["house"], doc["words"])

> 1.57  Stark     Winter is coming
> 0.45  Baratheon Ours is the fury
```

ElasticSearch offers a lot more... image: [2]

- A **boosting score** for fields and term queries *(important for your project!)*.
- Explaining the detailed calculation of a score:

```
{ 'description': 'sum of:',
  'details': [ { 'description': 'weight(words:winter in 0) '
                                '[PerFieldSimilarity], result of:',
                 'details': [ { 'description': 'score(doc=0,freq=1.0 = '
                                               'termFreq=1.0\n'
                                               '), product of:',
                                'details': [ { 'description': 'idf, computed '
                                                              'as log(1 + '
                                                              '(docCount - '
                                                              'docFreq + 0.5) '
                                                              '/ (docFreq + '
                                                              '0.5)) from:',
```

- adapting the **scoring** *(Similarity) (see formula above)*.
- adapting the **preprocessing** *(Analyzer)*.
- matching **phrases** of multiple terms.
- **synonyms**, **stopword filtering**, **fuzzyness**, ...
- ...

References I

[1] DB-Engines: Search Engine Ranking (March 2018).
`https://db-engines.com/en/ranking/search+engine` (retrieved: Sep 2020).

[2] Fred de Villami, Designing the Perfect Elasticsearch Cluster: the (almost) Definitive Guide.
`https://thoughts.t37.net/designing-the-perfect-elasticsearch-cluster-the-almost-definitive-guide-e614eabc1a87` (retrieved: Mar 2018).

[3] Shane Connelly.
https://www.elastic.co/de/blog/practical-bm25-part-3-considerations-for-picking-b-and-k1-in-elasticsearch.
`https://www.elastic.co/de/blog/practical-bm25-part-3-considerations-for-picking-b-and-k1-in-elasticsearch` (retrieved: Sep 2020).

[4] Data never Sleeps X.0.
`www.domo.com/blog/` (retrieved: Sep 2020).

[5] C. Manning, P. Raghavan, and H. Schütze.
Introduction to Information Retrieval.
Cambridge University Press, 2008.

[6] Michael McCandless, Erik Hatcher, and Otis Gospodnetic.
Lucene in Action, Second Edition: Covers Apache Lucene 3.0.
Manning Publications Co., Greenwich, CT, USA, 2010.

[7] A. Smeulders, M. Worring, S. Santini, and A. Gupta R. Jain.
Content-Based Image Retrieval at the End of the Early Years.
IEEE Trans. Pattern Analysis and Machine Intelligence, 22(12):1349–1380, 2000.