Anwendungen der KI

– Sommersemester 2018 –

Kapitel 02: Information Retrieval

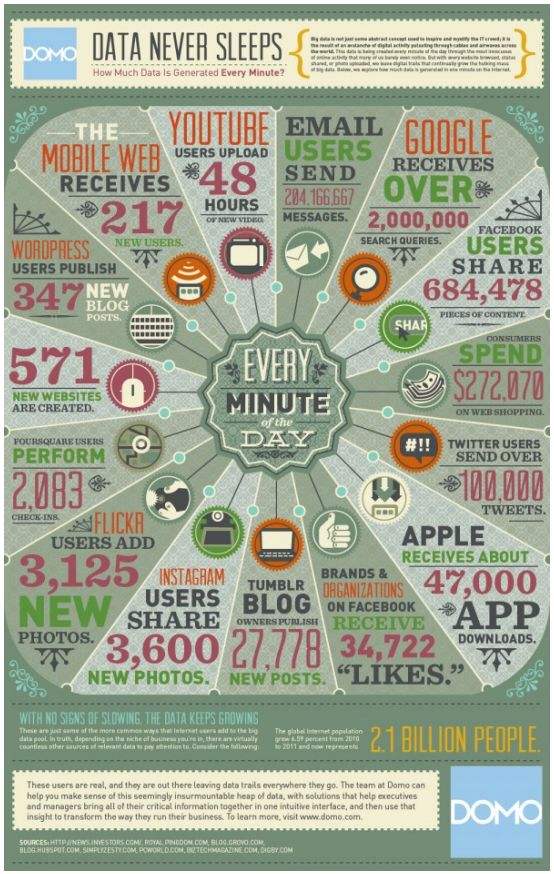
Prof. Dr. Adrian Ulges
B.Sc. Informatik (AI, ITS, MI, WI)
Fachbereich DCSM
Hochschule RheinMain

Outline

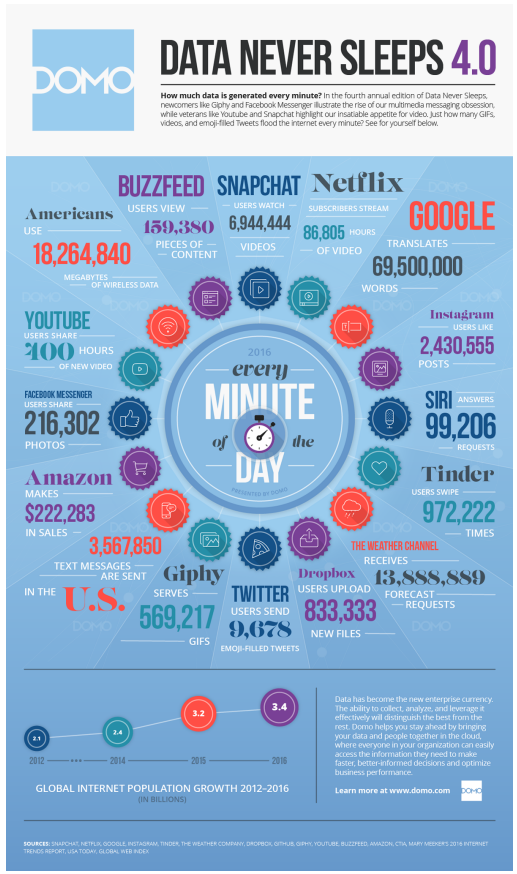
Information Retrieval: Definition und Bedeutung Bilder: [3]

"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 [4])

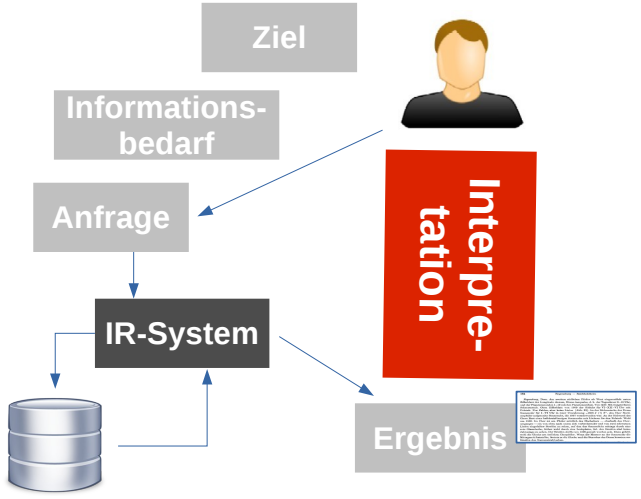


(Juni 2012)



(Juni 2016)

Information Retrieval: Definition und Bedeutung



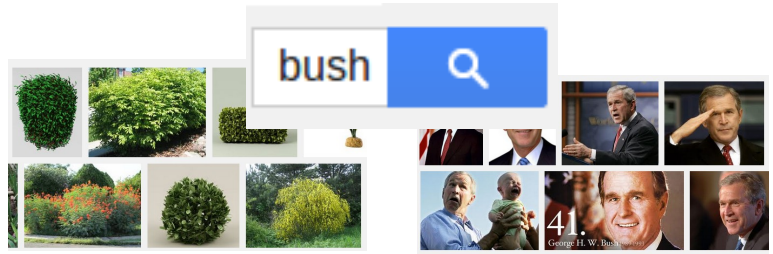
Anmerkungen

- Als akademische Disziplin seit den 1950ern
  (Juristen, Ärzte, Journalisten, ...)
- seit 1990ern (www) **Schlüsseltechnologie** modernen Lebens
- kein automatisiertes **Verstehen** von Text,
  sondern **Unterstützung** des Benutzers bei der Suche
- basierend auf **Text-Statistiken**
- Frage: Ist das **Problem gelöst**?

Information Retrieval: Herausforderungen

1. Unstrukturiertheit der Zieldaten

- Natürliche Sprache ist mehrdeutig!



	Datenbanken	**Information Retrieval**
Anfragesprache	formal (z.B. SQL)	natürlichsprachlich
Datenabgleich	exakt	partiell
Modell	deterministisch	probabilistisch
Datenfehler	sensitiv	insensitiv

2. Die semantische Lücke

> *"'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. [6])

Fallstricke natürlicher Sprache

1. (Irreguläre) Flexion

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

- Lösungsansatz: *Stemming*
  (`fishing`, `fisher` → `fish`)
- **Ansatz 1**: regelbasiert ('*ing' → '*')
- **Ansatz 2**: Lookup-Tables (stem['ipads']=ipad)

2. Synonyme und (partielle) Synonyme

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

- Synonyme = Unterschiedliche Wörter mit gleicher Bedeutung
- **Lösungsansatz**: *Thesauri*

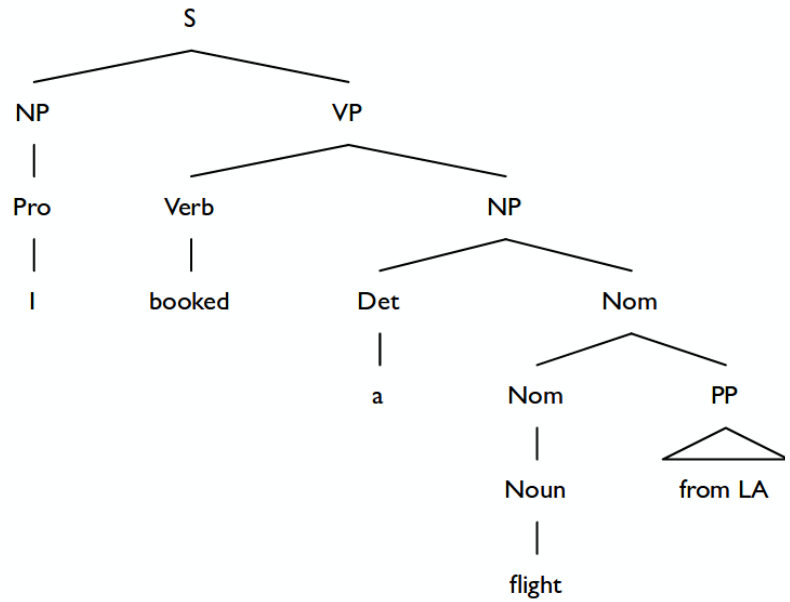
Fallstricke natürlicher Sprache

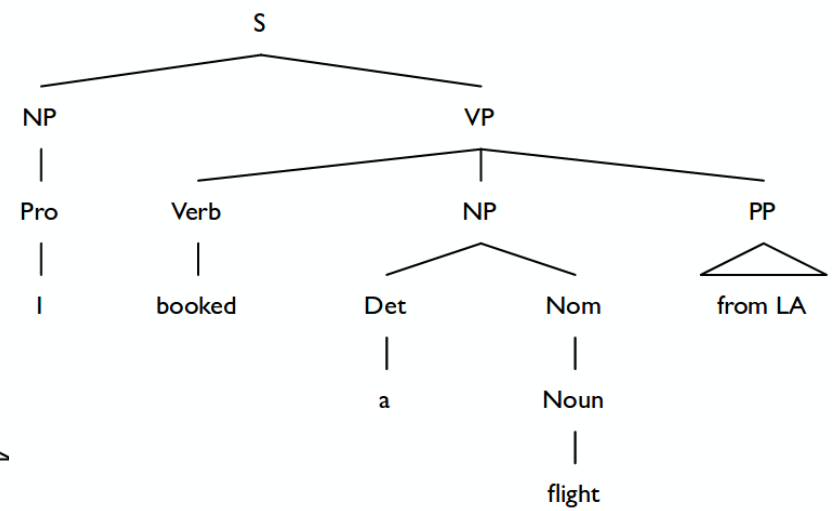
3. Homonyme

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

- Homonym = gleiches Wort, verschiedene Bedeutungen
- **Lösungsansätze**: Thesauri, maschinelles Lernen des Kontexts

4. Mehrdeutige Syntax





Fallstricke natürlicher Sprache

5. Relative Anschlüsse

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

- Lösungsansätze: Maschinelles Lernen *(später)*

6. ...

Die semantische Lücke: Beispiel

Die semantische Lücke: Konsequenzen

Die **Interpretation** *(und somit die Nützlichkeit)* der Retrieval-Ergebnisse ist **benutzer-** und **kontextabhängig**!

Unklar für das IR-System sind ...

- die Zielsetzung des Benutzers → `'things to do in Paris'`
- der Wissensstand des Benutzers → `'machine learning'`

Unklar für den Benutzer sind ...

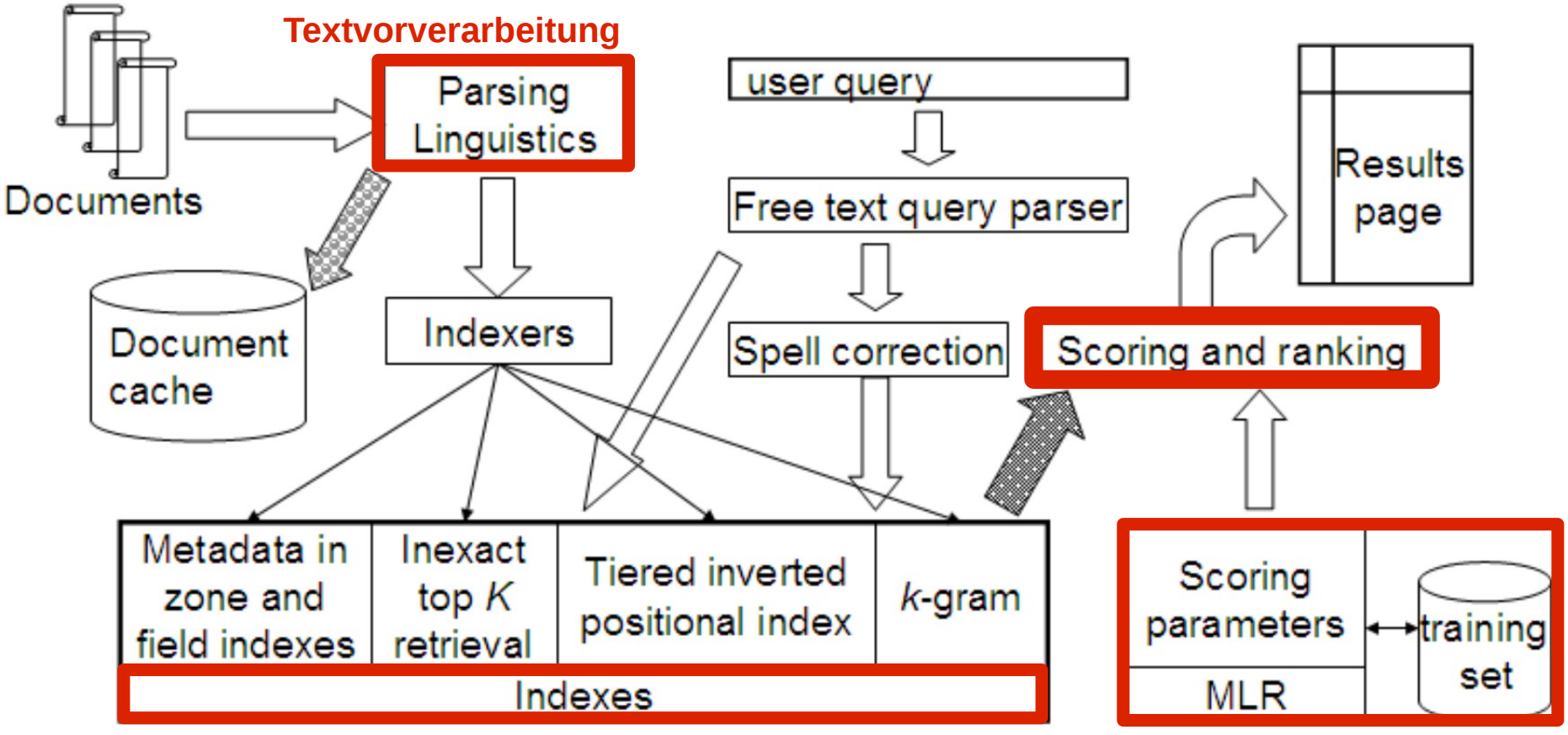
- Fachvokabular → `'classifier combination?'`
- Der Korpus selbst
  → `'headache treatment'` → `0 hits? 13428 hits?`

Konsequenz

- Die **Wahl des "richtigen" Queries** ist oft schwierig.

Outline

Information Retrieval: Setup Bild: [4]



Vorverarbeitung

- Information Retrieval operiert üblicher Weise auf der Basis von **Tokens / Termen**
- Üblicher Weise **segmentieren** wir deshalb Eingabedokumente (= lange Strings) in einzelne Tokens.

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', '...']
```

- Manchmal ist dies **nicht trivial** *(markiert z.B. ein* **Punkt** *ein* **Satzende***?)*: **Erkenner** für Datumsangaben, Telefonnummern, Adressen, ... basierend auf **Regulären Ausdrücken**.

Vorverarbeitung (cont'd)

Stemming

- Wir reduzieren außerdem Worte auf ihren Wortstamm.

```
> 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-Mdelle

- Mathematische Modelle zur Beurteilung der **Relevanz von Dokumenten**.
- Das Modell berechnet einen **Score** $s(\mathbf{q}, \mathbf{d})$, der schätzt wie **relevant** ein Dokument $\mathbf{d}$ für einen Query $\mathbf{q}$ ist.
- Gegeben einen Query, ranken wir alle Dokumente des Korpus nach ihrem **absteigenden Score**.
- Wir behandeln:
  1. Boolesches Retrieval
  2. Das **Vektorraum-Modell**
  3. Probabilistisches Retrieval

q = „soccer world cup“

Boolesches Retrieval

- Einfachstes und ältestes Modell *(1960er)*
- **Dokumente** werden als **Mengen von Termen** aufgefasst
- Die Menge aller **Queries Q** wird *(wie bei aussagenlogischen Formeln)* induktiv definiert:
  1. Einzelne Terme t sind Queries: $t \in Q$
  2. Negation ist zulässig: $t \in Q \rightarrow \neg t \in Q$
  3. Und/Oder-Verknüpfungen ergeben Queries:
     $q_1, q_2 \in Q \quad \rightarrow \quad q_1 \wedge q_2 \in Q, \;\; q_1 \vee q_2 \in Q$
- Anmerkung: Die **Reihenfolge** der Terme spielt keine Rolle!
- Der **Score** wird wie in der Aussagenlogik berechnet:
  1. $s(t, D) = \mathbf{1}_{t \in D}$
  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. $s(q_1 \vee q_2, D) = max\Big(s(q_1, D), s(q_2, D)\Big)$

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. ...

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

Bewertung

- Kein wirkliches Ranking (*"feast or famine"*). **Beispiel**:
  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)*
- Keine **Gewichtung** von Termen möglich
- Großer Vorteil: **Transparenz** ("user in control") $\rightarrow$ Noch häufig angewandt. Bsp. WestLaw *(Rechtswissenschaften, USA, 700,000 Nutzer)*

Das Vektorraum-Modell

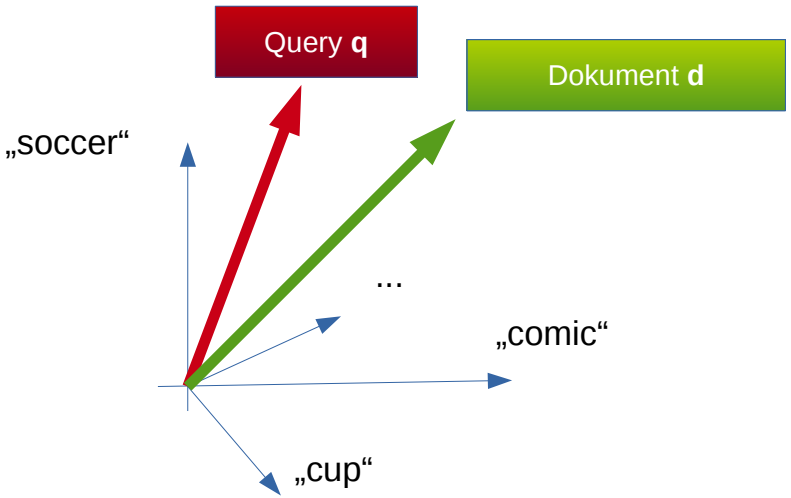
Historie

- Das Standard-Modell im Information Retrieval *(seit den 1970ern, auch heute noch in zahlreichen Varianten im Einsatz)*
- Verschiedene Implementierungen verfügbar, u.a. **Lucene**

Formalisierung

- Gegeben sei ein **Grundvokabular** $t_1, ..., t_n$ bekannter Terme
- Ein **Query** bestehe aus einer Menge von Worten, z.B. `'soccer world cup'` *(die Reihenfolge spielt keine Rolle!)*
- Wir repräsentieren den Query als reellwertigen Vektor $\mathbf{q} = (q_1, ..., q_n)$
- Analog repräsentieren wir jedes Dokument D als reellwertigen Vektor $\mathbf{d} = (d_1, ..., d_n)$
- Die **Einträge** q_i (bzw. d_i) bilden die **Wichtigkeit** des Terms t_i für den Query (bzw. das Dokument) ab.

Das Vektorraum-Modell: Illustration



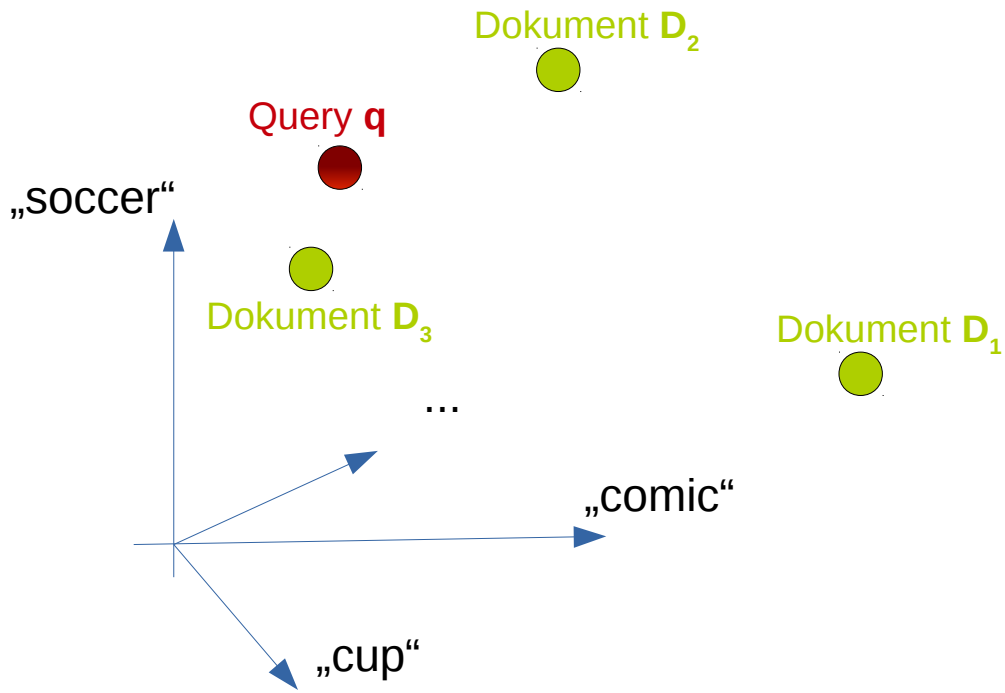
Interpretationen

- Dokumente und Queries werden als **Punkte** in einem **hochdimensionalen Raum** betrachtet.
- Wir erhalten die **Term-Dokument-Matrix** M (rechts).
- Jede **Zeile** steht für ein Dokument.
- $M_{ij} \neq 0$ gilt genau dann, wenn t_j in Dokument D_i **vorkommt**.
- M ist (sehr) **groß**, aber (sehr) **dünn besetzt** (“sparse”)!

Das Vektorraum-Modell

Geometrische Interpretation des Scorings

- Was wäre ein "gutes" Ähnlichkeitsmaß $s(\mathbf{q}, \mathbf{d})$?

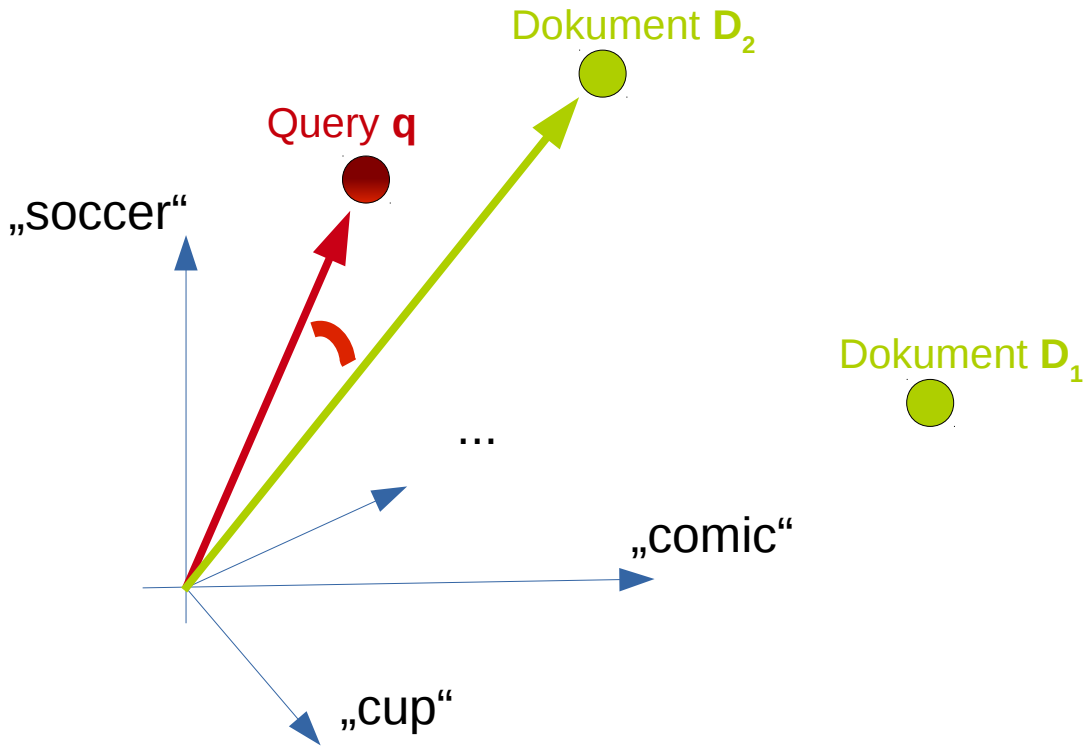


Das Vektorraum-Modell: Scoring

- Wir wählen als Ähnlichkeitsmaß den Kosinus des Winkels zwischen **q** und **d** *(berechnet mit dem Skalarprodukt)*:

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

- **Effekt**: **Normierung** bzgl. der Dokument-/Querylänge



Das Vektorraum-Modell: Term-Gewichte

Schlüsselfrage: Was tragen wir in die Vektoren $\mathbf{q}, \mathbf{d}$ ein?
(bzw. wie drücken wir die "Wichtigkeit" eines Terms aus?)

1. **Boolesche Gewichte**: $d_i = \mathbf{1}_{t_i \text{ kommt in } D \text{ vor}}$
2. **Term Frequency (TF)**: $d_i = tf(t_i, D)$
   $= \Big(\#\text{ Vorkommen von } t_i \text{ in } D \Big)$
3. **TF, gedämpft**:

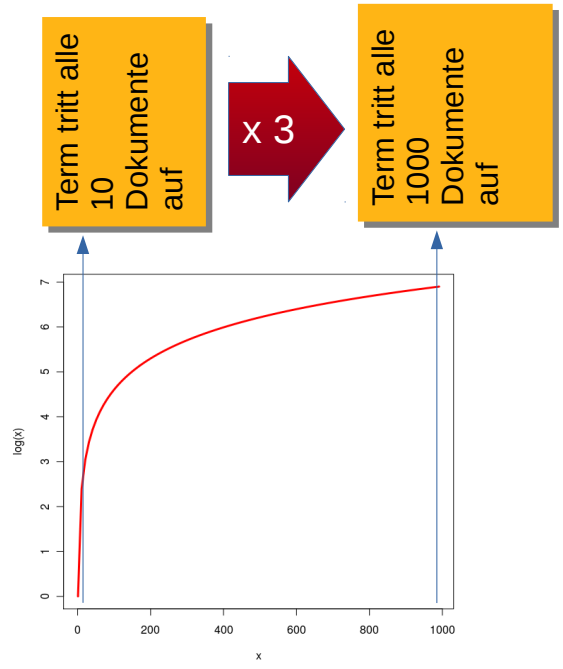
$$d_i = \begin{cases} 1 + log\Big(tf(t_i, D)\Big) & \text{falls } t_i \text{ in } D \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

(Ein Term der 10 mal vorkommt ist genau doppelt so "wichtig" wie ein Term der einmal vorkommt).

Term-Gewichte (cont'd)

4. TF-IDF

- **Idee**: Ein Term ist wichtiger je seltener er im gesamten Textkorpus vorkommt.
- **Beispiel**: Suche nach "Jesus Bergpredikt" im neuen Testament der Bibel.
- **Normalisierung** der Termgewichte (TF) mit der **Inverse Document Frequency (IDF)**



$$d_i := \underbrace{tf(t_i, D)}_{tf(t_i,D)} \cdot \underbrace{log\left(\frac{\#\text{ Dokumente des Korpus}}{\#\text{ Dokumente die } t_i \text{ enthalten}}\right)}_{idf(t_i)}$$

Anmerkungen

- Häufig filtern wir außerdem sehr häufige, uninformative Terme *(engl. "***Stop Words***")*
- Für die **Query-Einträge** q_i bestehen dieselben Optionen.

Das Vektorraum-Modell: Beispiel

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. ...

- Dokument 1 enthält 'prince' 1× und 'sand' 10×,
  Dokument 2 enthält 'prince' 10× und 'sand' 0×,
- 'prince' kommt in 100 von 1000 Dokumenten vor.
  'sand' kommt in 10 von 1000 Dokumenten vor.

$$\mathbf{q} = \left(\quad ... \quad 1/\sqrt{2} \quad ... \quad 1/\sqrt{2} \quad ... \quad \right) \quad // \text{ tf, normalisiert}$$

$$\mathbf{d_1} = \left(\quad ... \quad 1 \cdot log_{10} \frac{1000}{100} \quad ... \quad 10 \cdot log_{10} \frac{1000}{10} \quad ... \quad \right) \quad // \text{ tf-idf } (log_{10})$$

$$\mathbf{d_2} = \left(\quad ... \quad 10 \cdot log_{10} \frac{1000}{100} \quad ... \quad 0 \cdot log_{10} \frac{1000}{10} \quad ... \quad \right) \quad // \text{ tf-idf } (log_{10})$$

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

Vektorraum-Modell: Bewertung

Diskussion

- **Gute Qualität** der Suchergebnisse
  (empirisch ermittelt mittels Benchmarking)
- Wahl der Normalisierungen/Gewichte ist **intuitiv verständlich** *(aber heuristisch)*. Viele Auswahlmöglichkeiten.
- Verlust der **Wort-Reihenfolge** *(Lösungen: später)*
- Keine **"semantische Ähnlichkeit"**: Dokumente zum gleichen Thema aber mit anderen Termen werden nicht gefunden.

Komplexeres Beispiel: Okapi-BM25

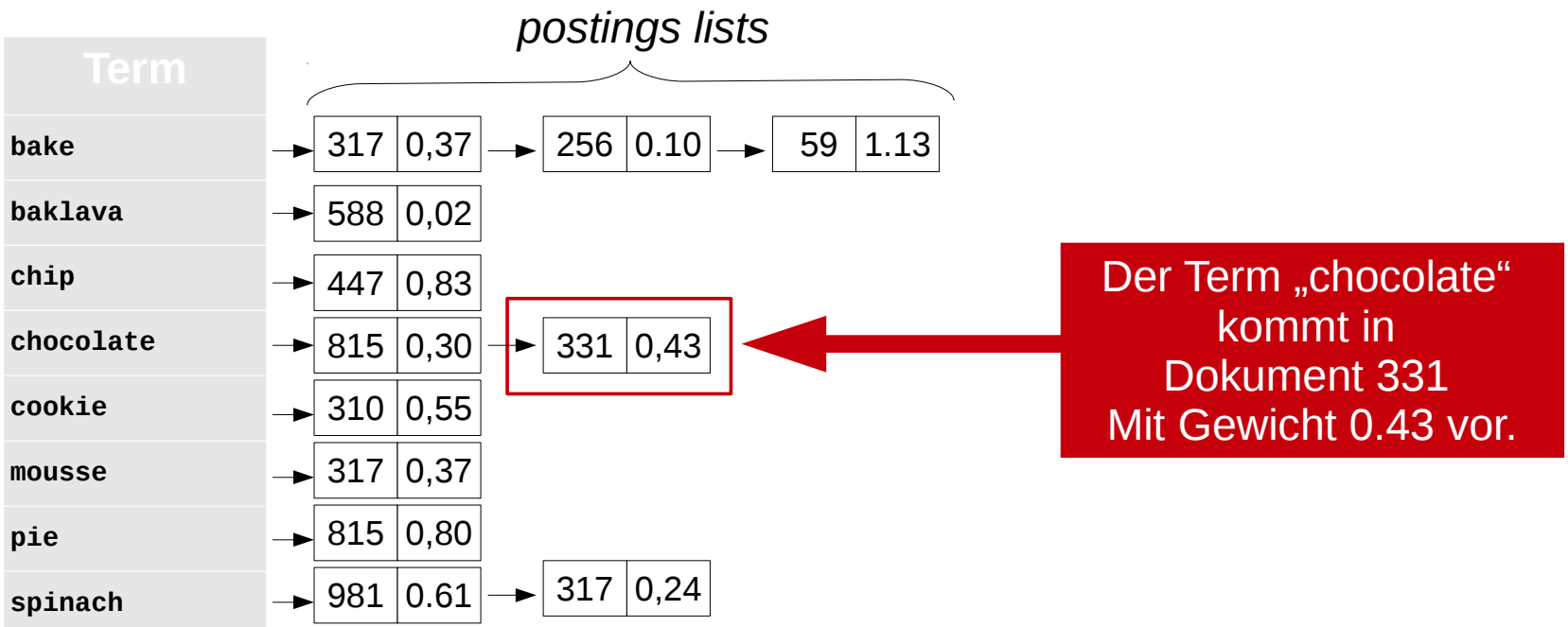
$$s(\mathbf{q},\mathbf{d}) := \sum_{t\in\mathbf{q}} idf(t) \cdot \frac{(k_1+1)\cdot tf(t,D)}{k_1((1-b)+b\cdot L_d/L_{avg}))+tf(t,D)}$$

- Modell variiert zwischen reiner idf ($k_1 \to 0$) und normaler tf-idf ($k_1 \to \infty$)
- Falls $b > 0$, werden lange Dokumente "bestraft".

Outline

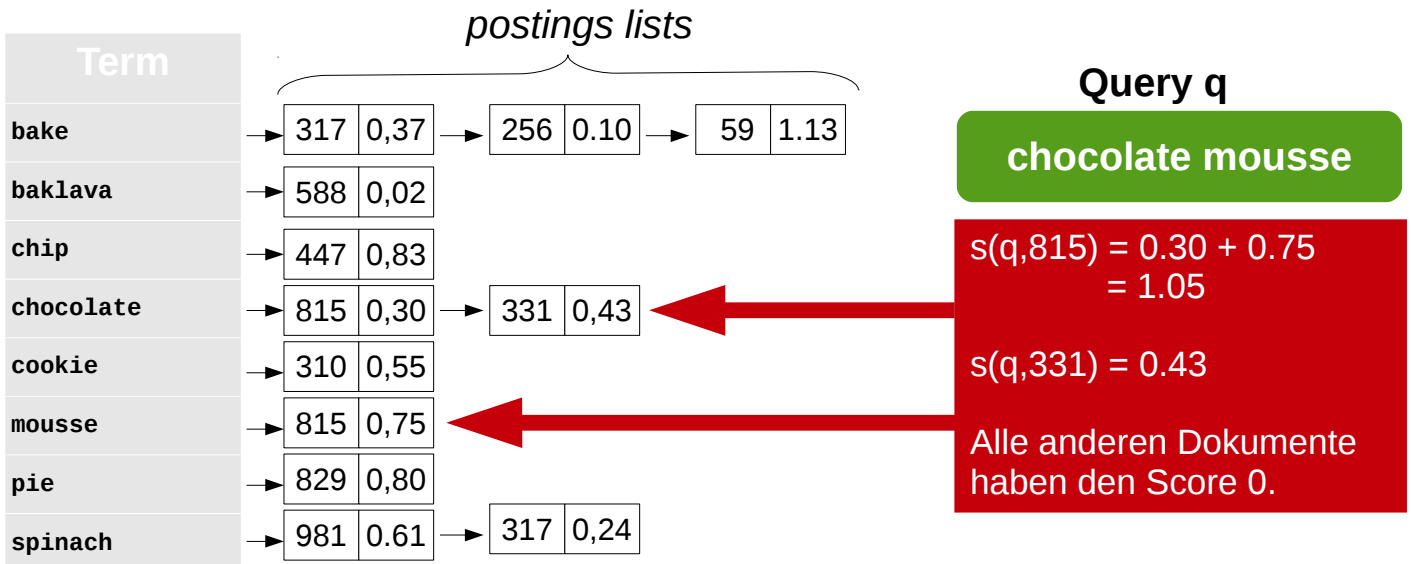
Inverted Files

- Die Standard-Indexstruktur in IR-Systemen!
- **Idee**: >99% der Einträge in der Term-Dokumentmatrix sind 0.
- Die meisten Dokumente enthalten somit keinen der Query-Terme. Ihr Score ist 0.
- Wir speichern für jeden Term *t* eine **Liste** (sog. "**postings list**") mit (1) den IDs der Dokumente, die *t* enthalten, und (2) den zugehörigen Gewichten d_i.



Inverted Files: Scoring

1. Für jedes **Dokument** D: Setze $s(q, D) := 0$
2. Für jeden **Term** t_i im **Query**: Durchlaufe die Liste $L(t_i)$
3. Für jeden **Eintrag** (D, d_i) in $L(t_i)$: $s(q, D) += q_i \cdot d_i$
4. **Ranke** die Dokumente nach s(q,D)



Anmerkungen

- Dieses Scoring kann für **jedes** gängige **Modell** durchgeführt werden (Boolesch, Vektorraum, probabilistisch, n-Gramme, ...)

Weitere Praxis-Tricks: Beschleunigung

Impact Ordering [1]

- Sortiere die **Postings-Lists absteigend** nach Gewicht d_i
- Sortiere die **Terme** des Queries absteigend nach $idf(t_i)$
- Breche das Scoring (mit einer "guten" **Approximation** der echten Scores) vorzeitig ab.

Stop Word Filtering

- Stop words ("und", "ist", ...) beanspruchen einen großen Teil der Suchzeit und einen signifikanten Teil des Index-Speichers *(bis zu 100% der Originaldaten)*
- Ansatz: Filtere häufige Terme!

[1] siehe auch ElasticSearch's `common terms query`

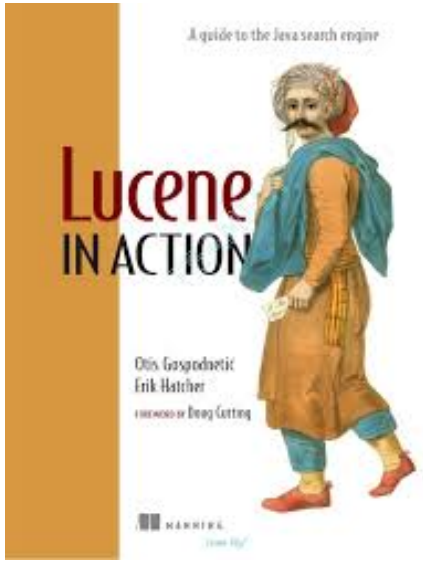
Weitere Praxis-Tricks: Phrasen-Queries

Ziel: Beantwortung von Phrasenqueries (“cat lady” vs “lady cat”)
→ Berücksichtigung der **Wort-Reihenfolge**

1. n-Gram-Index (vgl. ElasticSearch's `shingle token filter`)
   - Definition **n-Gramm** *(oder “shingle”)*:
     Subsequenz von *n* aufeinanderfolgenden Termen
   - n-Gramme werden **wie Terme** behandelt und erhalten **separate Einträge** im Inverted Index.
   - Oft: Vorheriges Filtern von Nicht-Substantiven

     “cost overruns on a power plant”
     → (cost overruns), (overruns, power), (power,plant)
2. Positional Index
   - Speichere zu jedem Wortvorkommen außerdem **Wort- und Satznummer** *(→ mehrere Einträge pro Term und Dokument)*
   - Ermöglicht **proximity queries** (`'bush' NEAR(3) garden'`)
   - Häufig: **Kombination** beider Ansätze *(Bigramm-Index für besonders häufig angefragte Phrasen)*

Lucene ist ...



- ... eine **Bibliothek** zur Textsuche
- ... **open-source** (seit 2005 Apache)
- ... verbreitet im praktischen Einsatz *(linkedin, twitter, wikipedia, ...)* [2]
- ... keine Suchmaschine, bietet aber eine **API** um **eigene Suchapplikationen** zu entwickeln
- ... in Java geschrieben, mit Interfaces zu C/C++, C#, Python, PHP, ...
- ... Grundlage für Enterprise Search Engines wie **ElasticSearch**[3]

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

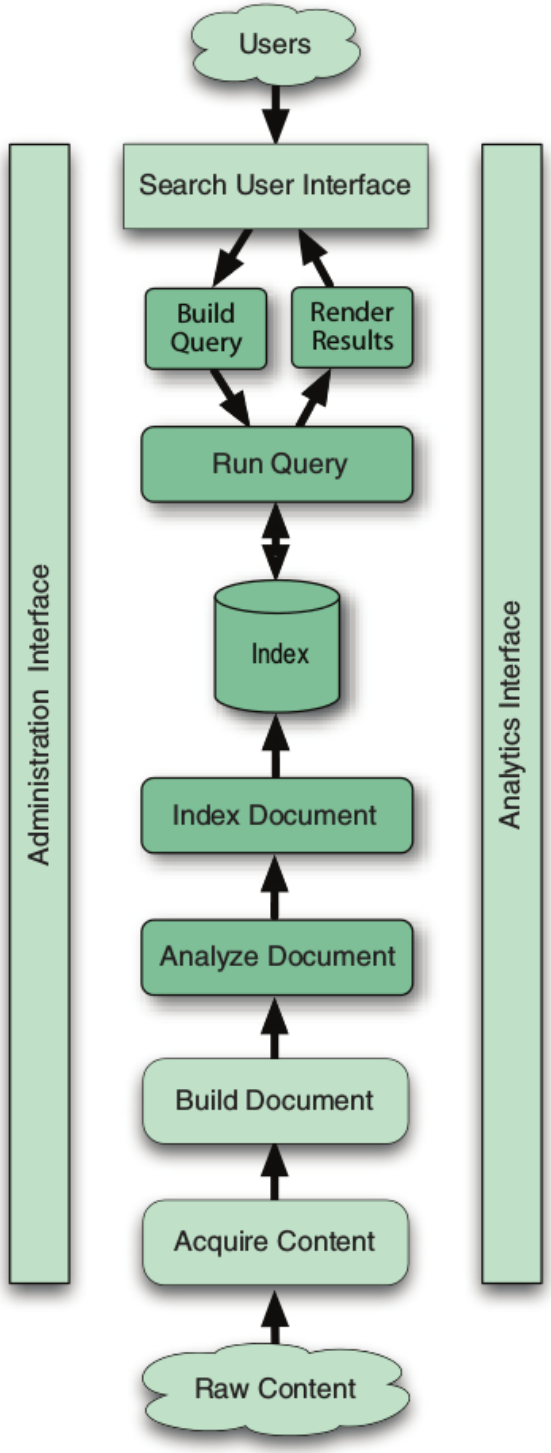
Lucene: Scope Bild: [5]

Lucene übernimmt IR-Kernschritte

- Scoring (→ Vektorraum-Modell)
- Vorverarbeitung (→ Tokenization, Stemming)
- Indexierung (→ Inverted File)
- Persistierung
- Parsing von Queries

Lucenes Suchverhalten ist adaptierbar!

- Scoring-Funktion → Similarity-Klasse ableiten
- Vorverarbeitung → Analyzer-Klasse ableiten



Lucene: Scoring (konzeptuell)[4]

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

$$\ldots \quad \times docBoost(\mathbf{d}) \times QueryBoost(\mathbf{q}) \times coord(\mathbf{q}, \mathbf{d})$$

- Anstatt durch $||\mathbf{d}||_2$ zu teilen, verwenden wir eine eigene Funktion $norm(\mathbf{d})$ *(die wir anpassen können)*.
- Mit $docBoost(\mathbf{d})$ können wir Dokumenten unterschiedliche Gewichte zuweisen, z.B. den PageRank *(später)*.
- Mit $queryBoost(\mathbf{q})$ können wir Termen des Queries unterschiedliche Gewichte zuweisen.
- $coord(\mathbf{q}, \mathbf{d})$ ist der Prozentsatz an Query-Termen, die im Dokument gefunden wurden.

[4] Weitere Infos in der ElasticSearch-Doku

Lucene: Scoring (praktisch)

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

- Wie oben, nur für die einzelnen Query-Terme t aufgeschlüsselt
- Grundmodell: tf-idf *(idf quadriert)*
- coord, queryBoost, queryNorm: siehe oben
- Der Faktor $norm(t, D)$ enthält:
  1. docBoost (siehe oben)
  2. Die Längennormalisierung $1/||\mathbf{d}||_2$
     (bezogen auf das jeweilige Feld `(Field)` *des Dokuments)*
  3. fieldBoost
     (auch Feldern kann ein Gewicht zugewiesen werden!)

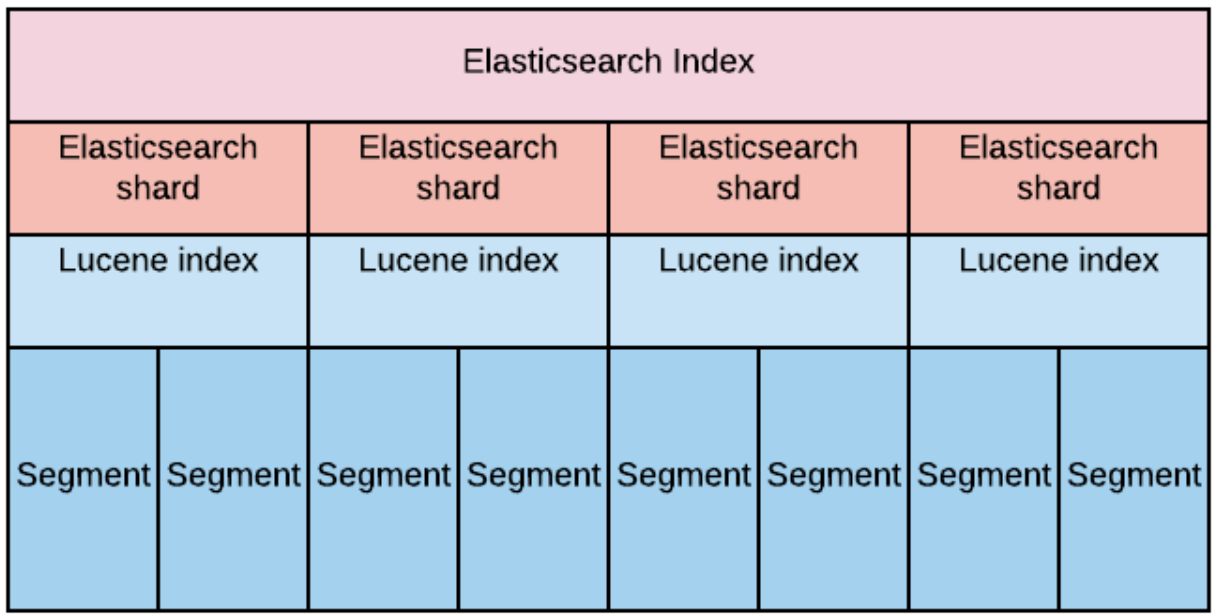
ElasticSearch

- ein verteilter Webserver ("Enterprise Search Engine") für **skalierbare Textsuche**
- Entwicklung durch Firma **ElasticSearch BV**
- open-source *(Apache-Lizenz)*
- Clients in vielen Sprachen *(inkl. Python)*
- Populärste Lösung für Enterprise Search weltweit [1]

Rank			DBMS	Database Model	Score		
Mar 2018	Feb 2018	Mar 2017			Mar 2018	Feb 2018	Mar 2017
1.	1.	1.	Elasticsearch	Search engine	128.54	+3.23	+22.32
2.	2.	↑ 3.	Splunk	Search engine	65.67	-1.60	+11.58
3.	3.	↓ 2.	Solr	Search engine	64.81	+0.94	+0.82
4.	4.	4.	MarkLogic	Multi-model	10.97	-0.05	-0.13
5.	5.	5.	Sphinx	Search engine	5.95	-0.11	-0.43
6.	6.	↑ 7.	Microsoft Azure Search	Search engine	4.60	+0.70	+2.31

ElasticSearch Bild: [2]

- operiert auf Basis von Lucene
- fügt **Skalierbarkeitsaspekte** hinzu:
  - Verteilbarkeit auf mehrere **Shards**
  - Shards sind replizierbar
  - **Routing** von Anfragen
  - Monitoring des Cluster-Zustands
- Zusatzkomponente **Kibana**: Analytics und Visualisierung
- Zusatzkomponente **LogStash**: Log file Parsing



ElasticSearch Bild: [2]

Nutzung via REST

- bietet eine JSON-basierte REST-API (Default-Port 9200)

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

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

- Inspiektion von Statistiken zu Dokumenten und Performance
- Queries testen, speichern, Dokumente filtern
- Visualisierungen

ES in Python: Index anlegen Bild: [2]

Wir definieren beim Anlegen des Index die **Shard-Struktur** und die **Dokument-Typen**.

```
1  import elasticsearch
2  es = elasticsearch.Elasticsearch()
3
4  body = {
5    "settings" : {
6       "number_of_shards" : 1,    # sharding
7       "number_of_replicas" : 2
8    },
9    "mappings" : {                  # document type
10      "my_doctype" : {
11         "properties" : {
12            "house" : { "type" : "text",    # field 1
13                        "index": "false" },
14            "words" : { "type" : "text",    # field 2
15                        "index": "true" },
16         }
17      }
18   }
19 }
20
21 es.indices.create(index="my_index", body=body)
```

ES in Python: Dokumente indexieren Bild: [2]

Dokumente sind nun einfache Attribut-Wert-Paare (Dictionaries).

```
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", doc_type="my_doctype", body=doc1)
16 es.index(index="my_index", doc_type="my_doctype", body=doc2)
17 es.index(index="my_index", doc_type="my_doctype", body=doc3)
18
19 es.indices.flush()    # persist changes (memory -> disk)
```

ES in Python: Suche Bild: [2]

Queries können (u.a.) mit Elastics DSL *(domain-specific language)* per JSON definiert werden:

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

Wir führen den Query durch und erhalten eine Ergebnisliste:

```
result = es.search(index="my_index", size=10, body=body)
hits = result["hits"]["hits"]

for hit in 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: Ausblick Bild: [2]

Wir können noch (viel) mehr

- Einen **Boosting-Score** für einzelne Felder oder Term-Queries festlegen *(Praktikum)*.
- Das Zustandekommen des Scores **erklären** lassen.

```
{ '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:',
```

- Das **Scoring** *(Similarity)* anpassen *(siehe Scoring-Formel oben)*, z.B. um Faktoren wie den PageRank zu ergänzen.
- Das **Preprocessing** *(Analyzer)* anpassen.
- **Phrasen** aus mehreren Termen matchen.
- **Synonyme**, **Stopwords**, **Fuzzyness** definieren.
- ...

References I

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

[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] Data never Sleeps X.0.
`www.domo.com/blog/` (retrieved: Aug 2016).

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

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

[6] 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.