Data Engineering

Elasticsearch #3

Suche

Search is so much more than algorithms. The complexity of good search is vastly underestimated

The case of Amazon

Amazon's success story is mostly associated with cheap products, tons of venture capital, fast delivery, good customer support and a wide range of products. But one of the most important factors is overlooked... Search

- Amazon‘s success is closely linked to the quality of their search engine
- Their user generated product reviews play a vital role in the quality of the search results
- In any category: The top 5 search results from amazon will give you:
    - A good and recent product
    - Decent Price/Value
    - Good Pricing
    - A fast & risk free checkout

The reason people love Amazon is much more related to the fast ordering process than to the fast shipping

What to learn from Amazon

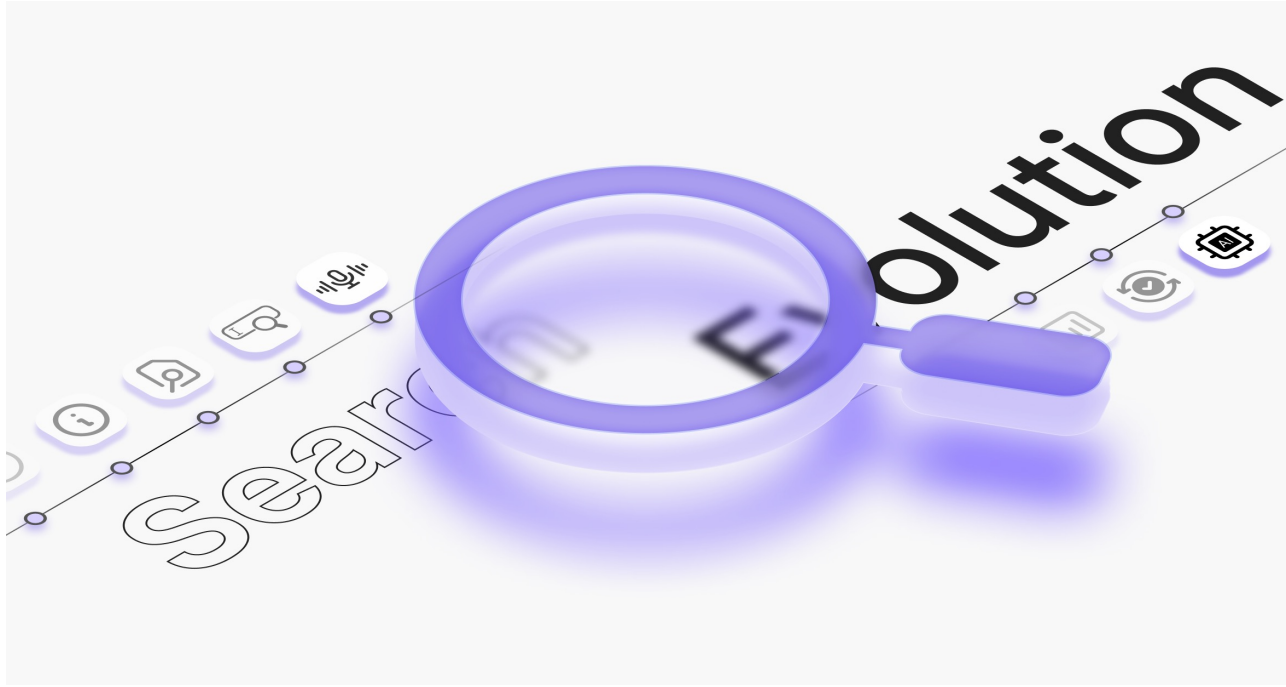
- The Searchbar is your Salesman
- Having the best search is THE competitive advantage in many industries
- Building a brilliant search is hard
- To have the best search, you have to have the best data (eg. Rating System @Amazon)

- In the end everything boils down to „Search Relevance“

Relevance is the art of ranking content based on how much that content satisfies the needs of the user and the business

History of web search

- Google understood that search relies on trust: The more often a site is linked to, the more trustworthy it is
- Google calls this „page rank“
- This is actually something that has been done in academia for decades to determine the trustworthiness of studies



Important Factors

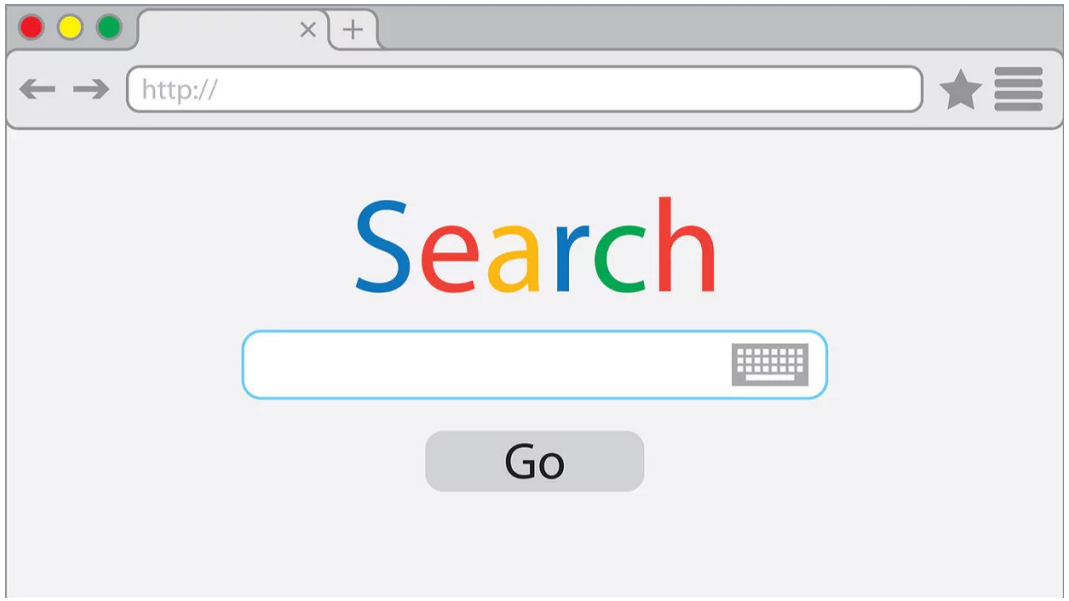
- Content: Tweets, Products, Videos, Files, trusted Content, untrusted Content
- Users (rich shoppers, poor shoppers, researchers,…)
- Language
- Geolocation
- User specific Data
- Typos
- Intent

→ A search engine should be a system that understands the needs of users and businesses

Search Relevance 101

What is a search engine?

- Tasks of a search engine: Store, Find and retrive data
- Documents is what search is all about: It‘s the item being stored, searched and returned
- A document contains a set of fields, also called attributes



Let‘s build our own Netflix search

- The „TMDB“ Dataset holds 5000 movies
- We want to offer a search over these movies using „title“ and „overview“
- Users should be able to find movies based on a fulltext search

original_title	overview
Avatar	In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but becomes torn between following orders and protecting an alien civilization.
Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, has come back to life and is headed to the edge of the Earth with Will Turner and Elizabeth Swann. But nothing is quite as it seems.
Spectre	A cryptic message from Bond’s past sends him on a trail to uncover a sinister organization. While M battles political forces to keep the secret service alive, Bond peels back the layers of deceit
The Dark Knight Rises	Following the death of District Attorney Harvey Dent, Batman assumes responsibility for Dent's crimes to protect the late attorney's reputation and is subsequently hunted by the Gotham City P
John Carter	John Carter is a war-weary, former military captain who's inexplicably transported to the mysterious and exotic planet of Barsoom (Mars) and reluctantly becomes embroiled in an epic conflict.
Spider-Man 3	The seemingly invincible Spider-Man goes up against an all-new crop of villain – including the shape-shifting Sandman. While Spider-Man’s superpowers are altered by an alien organism, his alt
Tangled	When the kingdom’s most wanted-and most charming-bandit Flynn Rider hides out in a mysterious tower, he’s taken hostage by Rapunzel, a beautiful and feisty tower-bound teen with 70 feet
Avengers: Age of Ultron	When Tony Stark tries to jumpstart a dormant peacekeeping program, things go awry and Earth’s Mightiest Heroes are put to the ultimate test as the fate of the planet hangs in the balance. As t
Harry Potter and the Half-Blood Prince	As Harry begins his sixth year at Hogwarts, he discovers an old book marked as 'Property of the Half-Blood Prince', and begins to learn more about Lord Voldemort's dark past.
Batman v Superman: Dawn of Justice	Fearing the actions of a god-like Super Hero left unchecked, Gotham City’s own formidable, forceful vigilante takes on Metropolis’s most revered, modern-day savior, while the world wrestles w
Superman Returns	Superman returns to discover his 5-year absence has allowed Lex Luthor to walk free, and that those he was closest too felt abandoned and have moved on. Luthor plots his ultimate revenge t
Quantum of Solace	Quantum of Solace continues the adventures of James Bond after Casino Royale. Betrayed by Vesper, the woman he loved, 007 fights the urge to make his latest mission personal. Pursuing his
Pirates of the Caribbean: Dead Man's Chest	Captain Jack Sparrow works his way out of a blood debt with the ghostly Davey Jones, he also attempts to avoid eternal damnation.

First things first

After uploading every row of the dataset as a document in an index into the search engine the following steps are done by the search engine

1. Every single document is *analyzed* by a so called *Analizer*
2. The documets are indexed
3. Documents are ready for search

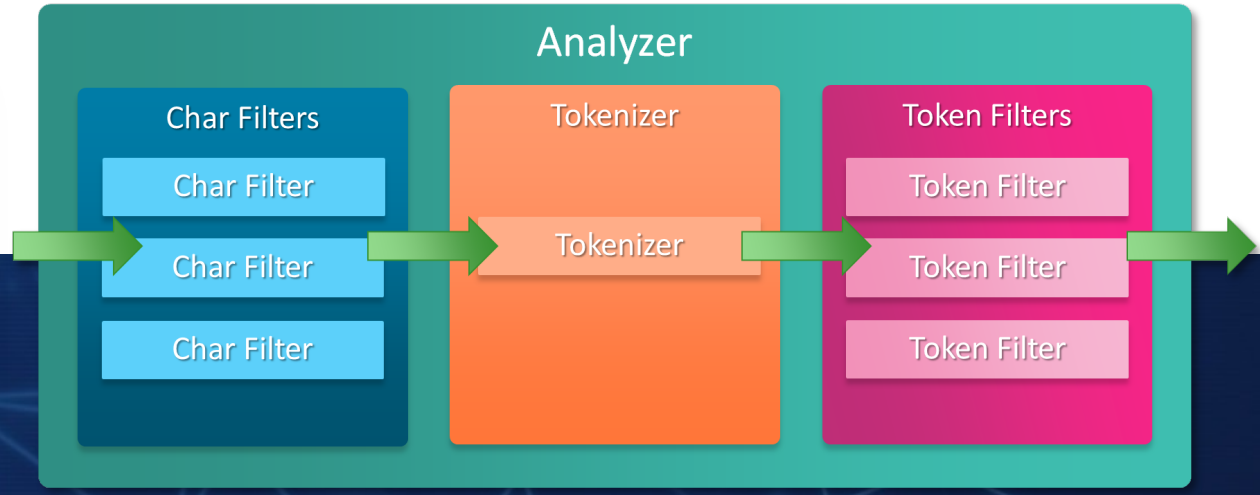
Analyzer

An analyzer is used on every field of a document and does the following three steps

Anatomy of an Analyzer

1. Character Filter to filter out specific characters
2. Tokenizer to split sentences in single words
3. Tokenfilters like converting to lowercase, filter stop words

Search only happens at the token level. Tokenization is one of the major impact factors affecting search engine performance.



Indexing

Search is done on what is called the index.

It‘s comparable to the index of a book

Inverted Index

- Most search Engines have two indices they are searching on

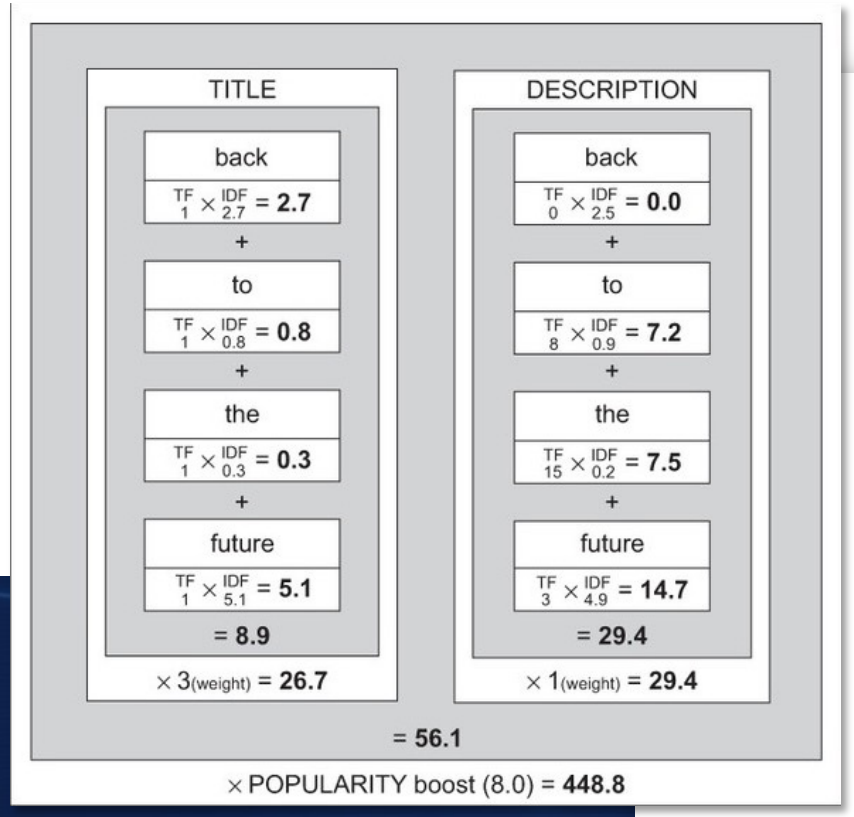
1. Forward Index
2. Inverted Index

Inverted Index

Term	Documents
Batman	Batman, Batman Begins, Batman & Robin, Batman Forever, Batman Returns
Love	Endless Love, Love Happens, Love & Basketball, Love Stinks
Fight	Fight Valley, Fight Club, Fight to the Finish
Galaxy	Galaxy Quest, Guardians of the Galaxy, The Hitchhiker's Guide to the Galaxy

Forward Index

Documents	Terms
Batman & Robin	batman, robin, gotham city, dynamic duo, ...
Love & Basketball	love, basketball, couple, drama, highschool, ...
Fight Club	fight, club, aggression, therapy, underground, ...
Guardians of the Galaxy	guardians, galaxy, science fiction, action, comic, ...



Scoring

- When issuing a search query the search engine analyzes the query
- After that all documents that matches at least one token in at least one of their fields will be returned
- To display the most relevant documents first, a score is attached to each document → the higher the better
- The score is affected by the number of matching tokens and a heuristic
- It‘s also possible to boost certain fields

Let‘s query

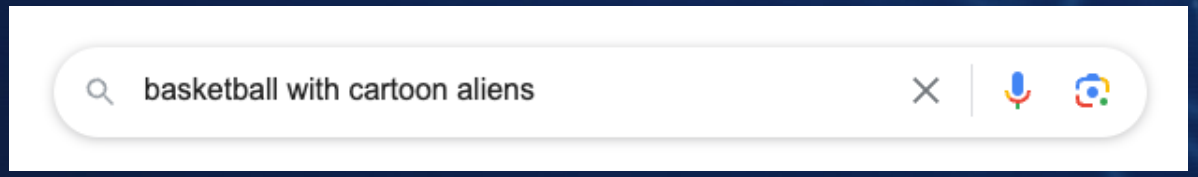
- Our first user wants to watch the movie „Space Jam“
- He doesn‘t know the name but remembers the three most important facts
- So what does our unoptimized query deliver?

basketball with cartoon aliens

evoila

The perfomance

- Our desired movie only ends up in place 43
- Movies featuring Basketball or Aliens alone, significantly outperform „Space Jam“
- To understand why we need to understand Scoring a little better



Num	Relevance Score	Movie Title
1	0.8424165	Aliens
2	0.5603433	The Basketball Diaries
3	0.52651036	Cowboys & Aliens
4	0.42120826	Aliens vs Predator: Requiem
5	0.42120826	Aliens in the Attic
6	0.42120826	Monsters vs Aliens
7	0.262869	Dances with Wolves
8	0.262869	Interview with the Vampire
9	0.262869	From Russia with Love
10	0.262869	Gone with the Wind
11	0.262869	Fire with Fire“
...	...	...
43	0.016977157	Space Jam

evoila

How scoring works

Scoring can be decribed by a simple formular: TF x IDF x FieldNorm

Term Frequency

- How often does a term appear in a field
- The more frequent, the higher the score

Inverse Document Frequency (IDF)

- The Document Frequency counts the number of Documets where a certain term is present
- Inverse DF is defined as 1/DF
- The rarer a certain term is, the higher the IDF gets scored

Field Norm

- A Term-Match in a Field with 2 Terms is less likely and therefore worth more then a match on a field with 1000 Terms
- Will significantly boost shorter fields

Alien vs. Space Jam

- The overview of space jam does not include the keyword „cartoon“
- The overview is written to attract a cinema audience
- It is not optimized to be found

Space Jam	Michael Jordan agrees to help the Looney Toons play a basketball game vs. alien slavers to determine their freedom.
Alien	During its return to the earth, commercial spaceship Nostromo intercepts a distress signal from a distant planet. When a three-member team of the crew discovers a chamber containing thousands of eggs on the planet, a creature inside one of the eggs attacks an explorer. The entire crew is unaware of the impending nightmare set to descend upon them when the alien parasite planted inside its unfortunate host is birthed.

Alien vs. Space Jam

```
0.03975798, weight(overview:alien in 1289) [PerFieldSimilarity], result of:
  0.03975798, score(doc=1289,freq=1.0 = termFreq=1.0), product of:
    0.03382846, queryWeight, product of:
      4.701128, idf(docFreq=70, maxDocs=2875)
      0.0071958173, queryNorm
    1.175282, fieldWeight in 1289, product of:
      1.0, tf(freq=1.0), with freq of:
        1.0, termFreq=1.0
      4.701128, idf(docFreq=70, maxDocs=2875)
      0.25, fieldNorm(doc=1289)
```

```
0.15913194, weight(overview:alien in 223) [PerFieldSimilarity], res
  0.15913194, score(doc=223,freq=1.0 = termFreq=1.0), product of:
    0.033834733, queryWeight, product of:
      4.7032127, idf(docFreq=70, maxDocs=2881)
      0.0071939616, queryNorm
    4.7032127, fieldWeight in 223, product of:
      1.0, tf(freq=1.0), with freq of:
        1.0, termFreq=1.0
      4.7032127, idf(docFreq=70, maxDocs=2881)
      1.0, fieldNorm(doc=223)
```

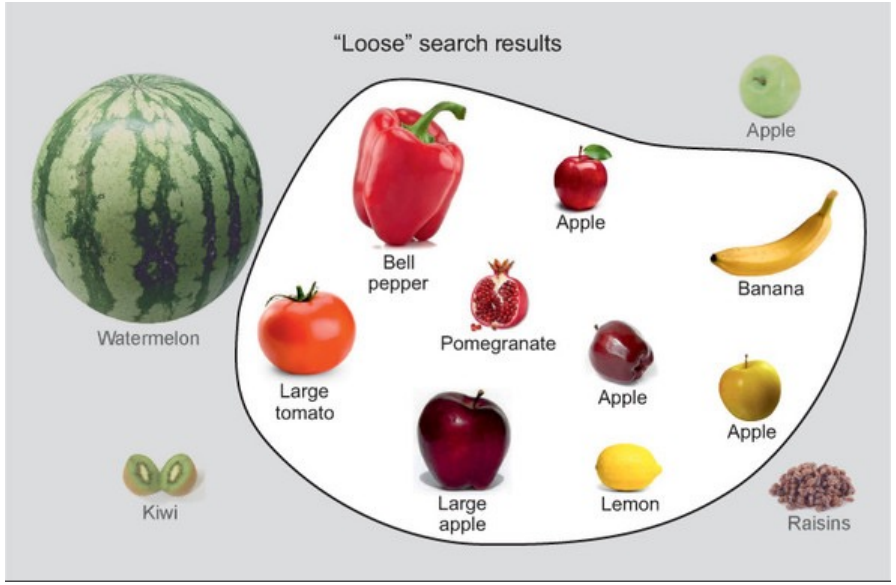
How to solve the problem

- Boosting the title field to be less significant
- Use more complex analyzers, so that terms like „toons“ can be matched with „cartoons“
- Create Synonyms
- Bring in more Data
- Increase Data Quality

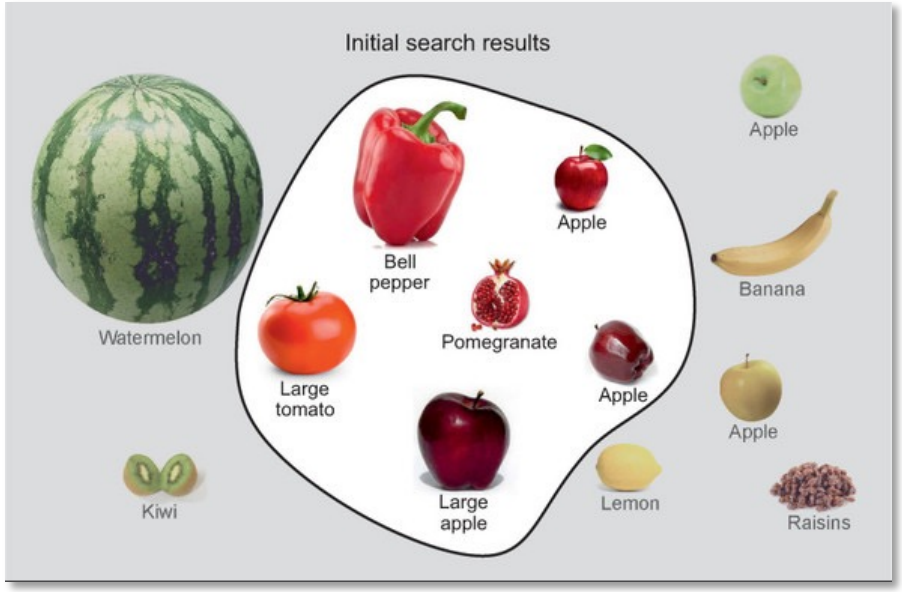
```
PUT /my_search_index
{
  "settings": {
    "index": {
      "analysis": {
        "analyzer": {
          "synonym": {
            "tokenizer": "standard",
            "filter": [ "my_synonyms" ]
          }
        },
        "filter": {
          "my_synonyms": {
            "type": "synonym",
            "lenient": true,
            "synonyms": [ "toons, cartoons", "computer, pc"
          }
        }
      }
    }
  }
}
```

Precision & Recall and why it matters

Precision: The % of documents in the resultset that are relevant
Recall: The % of relevant Documents in a resultset



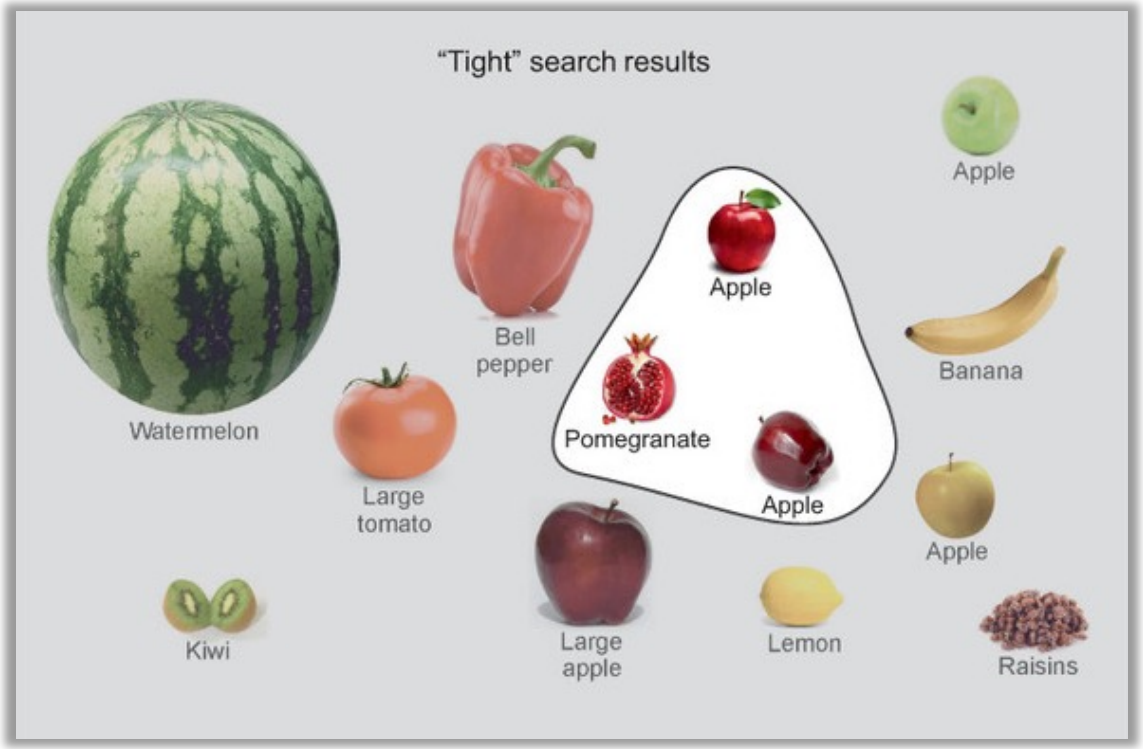
For the search „red & yellow“ when intend is an apple
Precision: 44%
Recall: 80%



For the search „red“ when intend is an apple
Precision: 50%
Recall: 60%

How to fix the dilemma

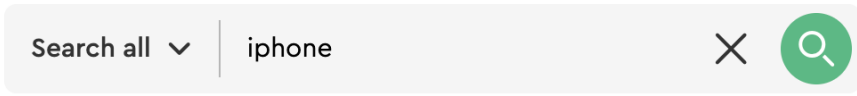
- Precision and recall are proven to be at odds
- Improving one will make the other worse
- The only way out is more Features
- Let‘s add a new feature “size“ to the equation

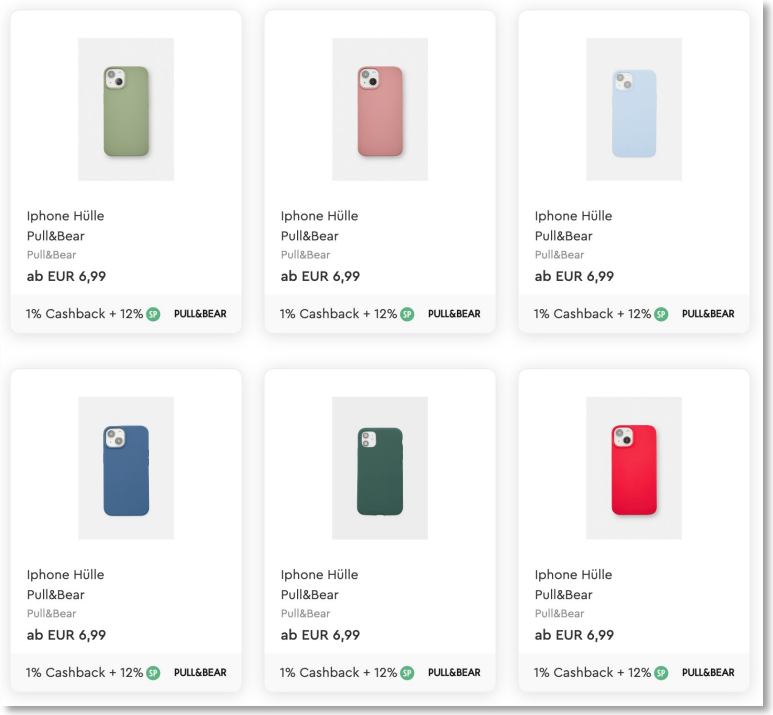


For the search „red + small size“ when intend is an apple
Precision: 66%
Recall: 40%

Precision & Recall: Real World Examples

Precision: The % of documents in the resultset that are relevant
Recall: The % of relevant documents in a resultset



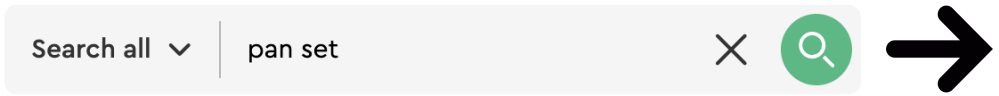


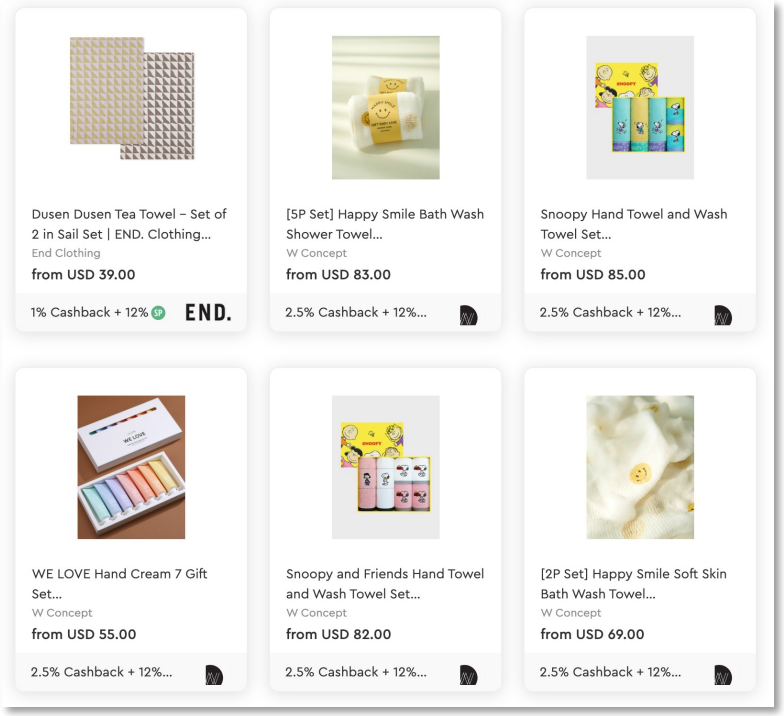
For the search „iphone“ we expect iPhones to be the first results followed by accessories

But with too little precision the first items that come up are iphone cases as they feature the keywords „iPhone“, „iPhone13“ multiple times in their description
Precision: low
Recall: too high

Precision & Recall: Real World Examples

Precision: The % of documents in the resultset that are relevant
Recall: The % of relevant documents in a resultset



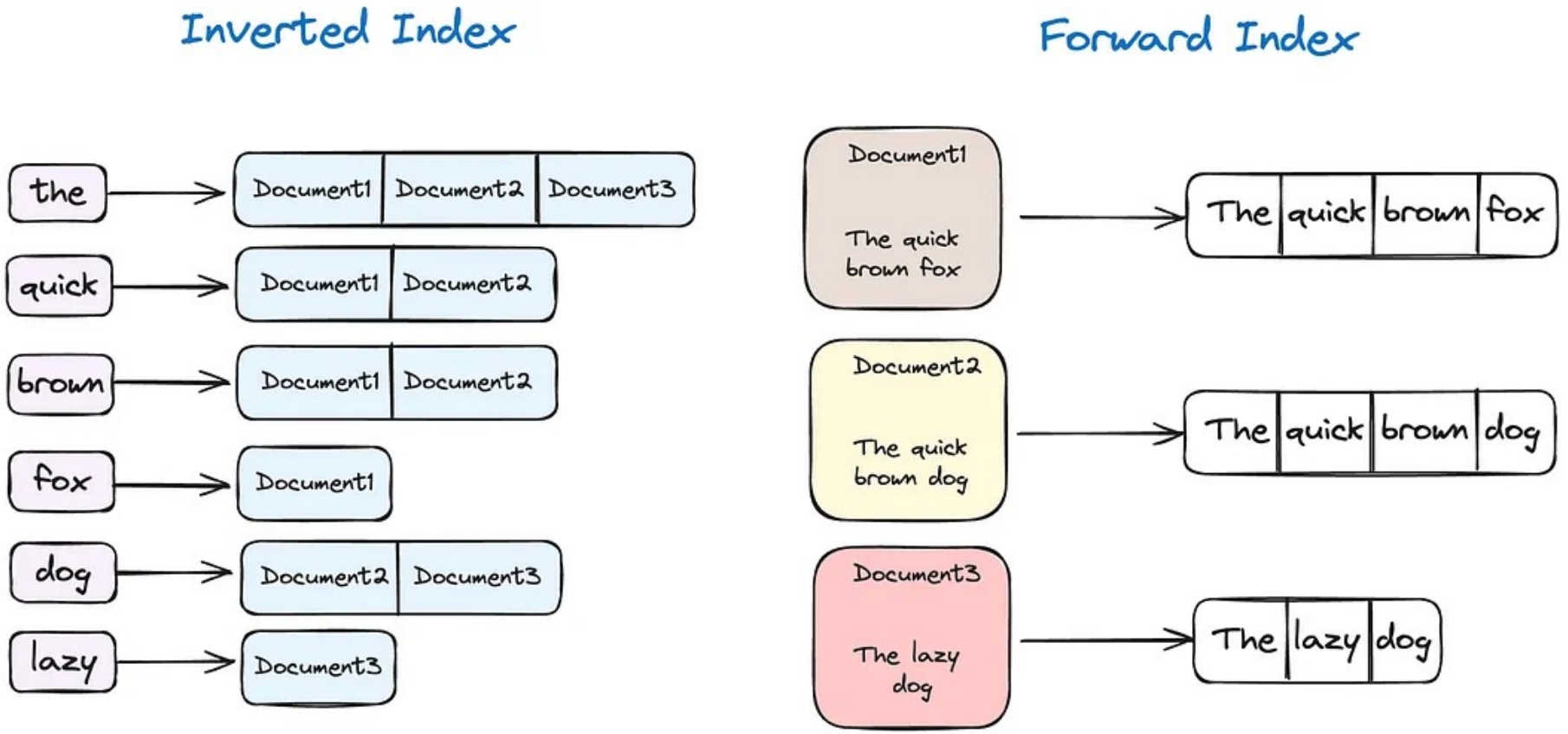


For the search „pan set“ we expect pan set results

But with too little precision the first items that come up are other kind of sets as they feature the keywords „set“ multiple times in their description and title
Precision: low
Recall: too high

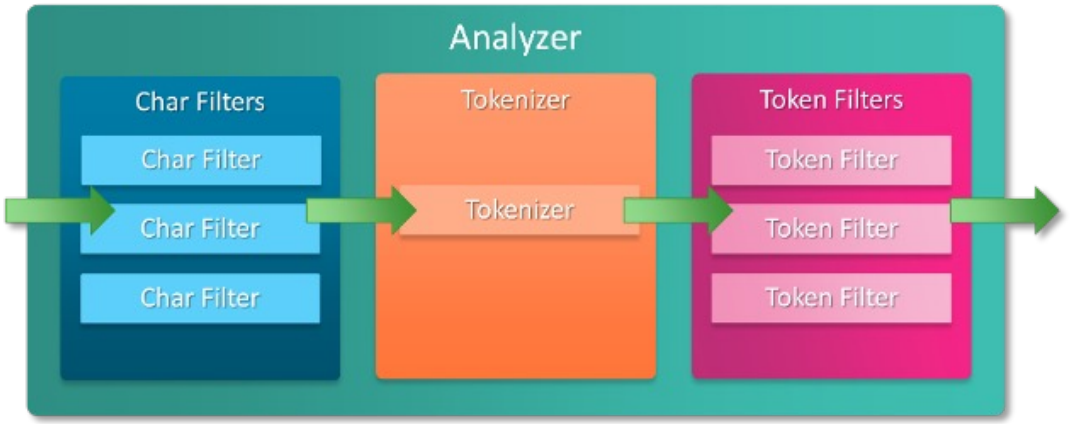
Search in Elasticsearch

Inverted Index



Analyser

- Analyser besteht auch in Elasticsearch aus den drei Komponenten “Character Filters”, “Tokenizer”, “Token Filter”
- Elasticsearch bringt Analyser für gängige Sprachen mit, sollte in Such-Usecases aber unbedingt angepasst werden



Analyzer konfigurieren

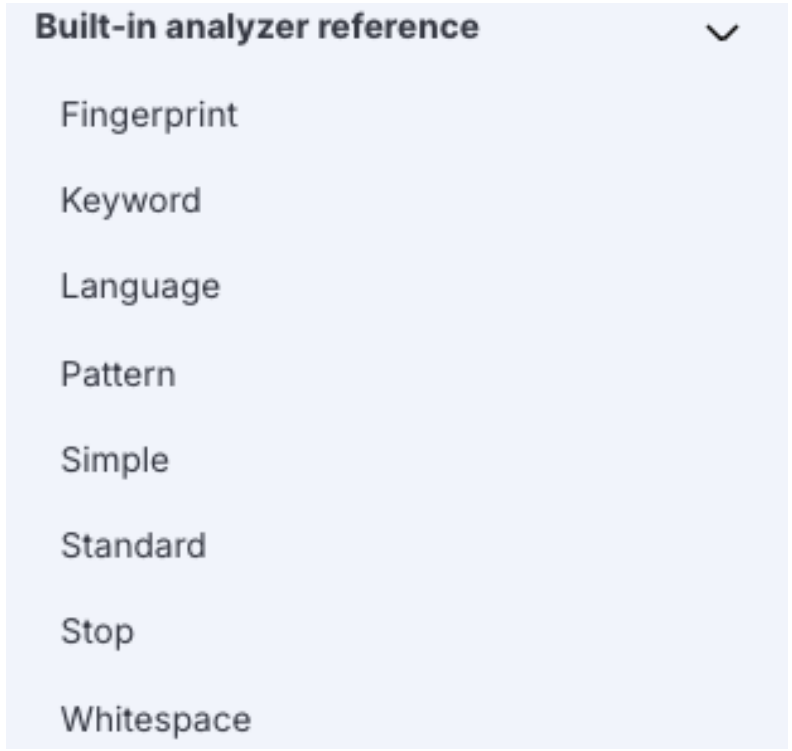
- Elasticsearch liefert eine Reihe an Standard Analyzer
- Manche Analyzer besitzen Optionen wie bspw. Stopwords, die zusätzlich definiert werden können
- Der Analyzer wird in den Index Settings definiert und in den Mappings auf ein spezifisches Feld angewendet
- Verschiedene Felder können unterschiedliche Analyzer verwenden

```
PUT my-index-000001
{
  "settings": {
    "analysis": {
      "analyzer": {
        "std_english": {
          "type":      "standard",
          "stopwords": "_english_"
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "my_text": {
        "type":     "text",
        "analyzer": "standard",
        "fields": {
          "english": {
            "type":     "text",
            "analyzer": "std_english"
          }
        }
      }
    }
  }
}
```

```
PUT my-index-000001
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_custom_analyzer": {
          "char_filter": [
            "emoticons"
          ],
          "tokenizer": "punctuation",
          "filter": [
            "lowercase",
            "english_stop"
          ]
        }
      },
      "tokenizer": {
        "punctuation": {
          "type": "pattern",
          "pattern": "[ .,!?]"
        }
      },
      "char_filter": {
        "emoticons": {
          "type": "mapping",
          "mappings": [
            ":) => _happy_",
            ":( => _sad_"
          ]
        }
      },
      "filter": {
        "english_stop": {
          "type": "stop",
          "stopwords": "_english_"
        }
      }
    }
  }
}
```

Custom Analyzer

- Elasticsearch erlaubt das Erstellen eigener Analyzer
- Der Analyzer benötigt die vorher spezifizierten Komponenten, welche Ebenfalls angepasst werden können



Eingebaute Analyzer

- Elastic verfügt über eine Vielzahl an vordefinierter Analyzer
- Der Language Analyzer verfügt über vorkonfigurierte Analyzer für über 35 Sprachen
- Der Keyword Analyzer baut aus einen beliebigen Input **ein** Keyword
- Der Fingerprint Analyzer normalisiert und dedupliziert Keywords
- Der Pattern Analyzer fordert eine Regex die für die Tokenization verwendet wird

Analyzer – Ausführungszeitpunkt

- **Index Time**
  - Analyzer werden zur Indexierung verwendet um den Inverted Index aufzubauen
- **Search Time**
  - Damit die Suche funktionieren kann, muss die Suchanfrage selbst durch den Analyzer laufen. Ansonsten wären Filter wie „Lowercase“ oder „Stemming“ sinnlos
- In beiden Fällen werden die gleichen Analyzer verwendet

Tokenizer

- Elasticsearch bietet eine Reihe an eingebauten Tokenizer
- Grundsätzlich Unterscheidet man zwischen:
  - Word Oriented Tokenizer
    - Wird genutzt um Text in individuelle Wörter zu tokenizen
  - Partial Word Tokenizer
    - Text wird in Wortfragmente Umgewandelt – wichtig für Type-Ahead Search
  - Structured Text Tokenizer
    - Für strukturierten Text wie E-Mails, Postleitzahlen, IDs, etc.

N-gram Tokenizer

- Tokenisiert Wörter
- Bricht diese Wörter dann auf N-grams runter
- N-grams sind eine Art „*Sliding Window*“ auf einem Text, dass alle Potenziellen Substrings eines Wortes erstellt

```
POST _analyze
{
  "tokenizer": "ngram",
  "text": "Quick Fox"
}
```

```
[ Q, Qu, u, ui, i, ic, c, ck, k, "k ", " ", " F", F, Fo, o, ox, x ]
```

Token-Filter

- Tokenizer bekommen eine Liste an Tokens vom Tokenizer
- Diese können verarbeitet werden:
  - Veränderung des Tokens (lowercase, Stemming, etc.)
  - Löschen von Tokens (Stopwords)
  - Hinzufügen von Tokens (Synonyme)
- Elasticsearch bietet knapp 50 vordefinierte Token-Filter

Hyphenation decompounder token filter

- Häufiges Problem in Suchen: Zusammengesetzte Worte (compound words)
- Beispiel: „Pfannenset“ – Suche nach „Pfanne“ gibt keine relevanten Ergebnisse zurück
- Lösung: Decompounder – Baut deutsche, zusammengesetzte Worte in ihren Wortursprung zurück

```
GET _analyze
{
  "tokenizer": "standard",
  "filter": [
    {
      "type": "hyphenation_decompounder",
      "hyphenation_patterns_path": "analysis/hyphenation_patterns.xml",
      "word_list": ["Kaffee", "zucker", "tasse"]
    }
  ],
  "text": "Kaffeetasse"
}
```

```
[ Kaffeetasse, Kaffee, tasse ]
```

Synonym token Filter

- Synonyme sind ein häufiges Problem in Suchen
- Beispiel: iPhone, i-Phone, i Phone, Apple Phone

```
"filter": {
  "synonyms_filter": {
    "type": "synonym",
    "synonyms_set": "my-synonym-set",
    "updateable": true
  }
}
```

```
ipod, i-pod, i pod
computer, pc, laptop
```

Stemmer

- Konjugationen und der Plural stellen in der Suche ein Problem dar:
- Suchanfrage „Trainingspläne“ matcht nicht auf „ultimativer Trainingsplan“, „rennen“ nicht auf „rannte“
- Lösung: Reduzierung auf den Wortstamm → Stemming
- Stemming Tokenfilter in elasticsearch bietet Stemmer für verschiedene Sprachen

Phonetic token filter

- Problem: Vertipper und fehlerhafte Rechtschreibung
- Lösung: Phonetic token filter tokenisiert Worte nach ihrer Aussprache
- In elastic nur durch zusätzliches Plugin nutzbar
- Achtung: Precision vs. Recall → Erhöht Recall enorm aber hat den gegenteiligen Effekt auf die Precision
- Mit Vorsicht einzusetzen: Nur für spezielle Anwendungsfälle. Für Vertipper gibt es fuzzy search

Normalization

- Terms wie “Quick“ und matchen nicht „quick“
- Elasticsearch bietet sogenannte Normalizer um diese Probleme zu adressieren
- Normalizer arbeiten auf Zeichen (Char) Basis und sind deswegen weniger mächtig als Analyser
- Der Lower-Case Normalizer ist per default aktiviert
- Es lassen sich eigene Normalizer implementieren

Queries

Boolean Queries

- Die Bool Query kombiniert andere Queries mit booleschen Klauseln:
- **must**:
  - Die Query muss in den Treffern vorkommen und beeinflusst den Score.
- **filter**:
  - Die Query muss in den Treffern vorkommen, beeinflusst den Score jedoch nicht.
- **should**:
  - Die Query sollte in den Treffern vorkommen.
- **must_not**:
  - Die Query darf nicht in den Treffern vorkommen. Wird im Filter-Kontext ausgeführt (kein Score, Caching möglich).

Boolean Queries

```
POST _search
{
  "query": {
    "bool" : {
      "must" : {
        "term" : { "user.id" : "kimchy" }
      },
      "filter": {
        "term" : { "tags" : "production" }
      },
      "must_not" : {
        "range" : {
          "age" : { "gte" : 10, "lte" : 20 }
        }
      },
      "should" : [
        { "term" : { "tags" : "env1" } },
        { "term" : { "tags" : "deployed" } }
      ],
      "minimum_should_match" : 1,
      "boost" : 1.0
    }
  }
}
```

Boosting

- **Funktion:**
  - Liefert Treffer für eine **positive Query**, verringert jedoch die Relevanz von Treffern, die auch zur **negativen Query**passen.
- **Anwendung:**
  - Dokumente werden nicht ausgeschlossen, sondern in der Relevanz herabgestuft.
  - Nützlich, um bestimmte Ergebnisse zu **entwerten**, ohne sie komplett zu entfernen.

Boosting

```
GET /_search
{
  "query": {
    "boosting": {
      "positive": {
        "term": {
          "text": "apple"
        }
      },
      "negative": {
        "term": {
          "text": "pie tart fruit crumble tree"
        }
      },
      "negative_boost": 0.5
    }
  }
}
```

Match Query

- **Funktion:**
  - Liefert Dokumente, die mit einem angegebenen Text, einer Zahl, einem Datum oder einem Booleschen Wert übereinstimmen.
- **Besonderheiten:**
  - Der Text wird vor der Suche mittels Analyzer verarbeitet
  - Standard-Query für **Volltextsuche**.
  - Unterstützt Optionen für **unscharfe Suche (fuzzy matching)**.
  - Match Query ist im Standard sehr simple bietet aber viele Optionen

Match Query – boolean

- **Funktionsweise:**
  - Operator gibt an ob alle Terms aus der Query matchen müssen (and) oder mindestens eines
- **Operator-Parameter:**
  - Kann auf **or** (Standard) oder **and** gesetzt werden
- **Optional: Minimum Should Match:**
  - Mit dem Parameter **minimum_should_match** kann die minimale Anzahl der "should"-Klauseln festgelegt werden, die übereinstimmen müssen.

```
GET /_search
{
  "query": {
    "match": {
      "message": {
        "query": "this is a test",
        "operator": "and"
      }
    }
  }
}
```

Match Query – fuzziness

- **Funktion:**
  - Ermöglicht unscharfe Treffer basierend auf der Hamming Distanz
- **Parameter für Steuerung:**
  - **prefix_length:** Minimale Präfixlänge vor der Unscharfe.
  - **max_expansions:** Maximale Anzahl möglicher Varianten.

```
GET /_search
{
  "query": {
    "match": {
      "message": {
        "query": "this is a testt",
        "fuzziness": "AUTO"
      }
    }
  }
}
```

Abschlussprojekt

- **Projektziel:**
  Entwicklung einer End-to-End-Lösung zur Erfassung, Verarbeitung und Speicherung von Daten, inklusive einer Beispielanalyse.
- **Drei Säulen des Projekts:**

1.Scraping von Daten:

1. Daten müssen **periodisch** abgegriffen werden.
2. **Scheduling** idealerweise mit Tools wie **Airflow**.

2.Vorverarbeitung der Daten:

1. Externe und umfangreiche Vorverarbeitung der Daten.

3.Speicherung der Daten:

1. Speicherung in einer für den Usecase sinnvollen **Datenbank**.
2. **Einstellung und Optimierung** der Datenbank.
3. Sinnvolles Deployment mit entsprechenden Einstellungen.

- **Wichtig:** Beispielanalyse ist **kein integraler Bestandteil** des Data Engineerings.

Abschlussprojekt

- **Ablauf:**
- **Projektidee:**
  Einreichung bis zur **letzten Woche im Dezember**.
- **Freigabe:**
  Durch Projektleitung nach Prüfung.
- **Bearbeitungszeit:**
  Bis zum finalen **Präsentationstermin (tbd)**.
- **Abgabe und Bewertung:**

1. **Abschlusspräsentation:**
    1. Darstellung der getroffenen Entscheidungen.
2. **Code-Abgabe:**
    1. Muss in einem zugänglichen **Git Repository** vorliegen.
3. **Einstellungen und Deployment:**
    1. Alle Datenbank- und Deployment-Einstellungen als **Code** verfügbar (z. B. Skripte).
    2. Keine manuellen Änderungen (z. B. via Kibana).