DIS17 – Search Engine Technologies

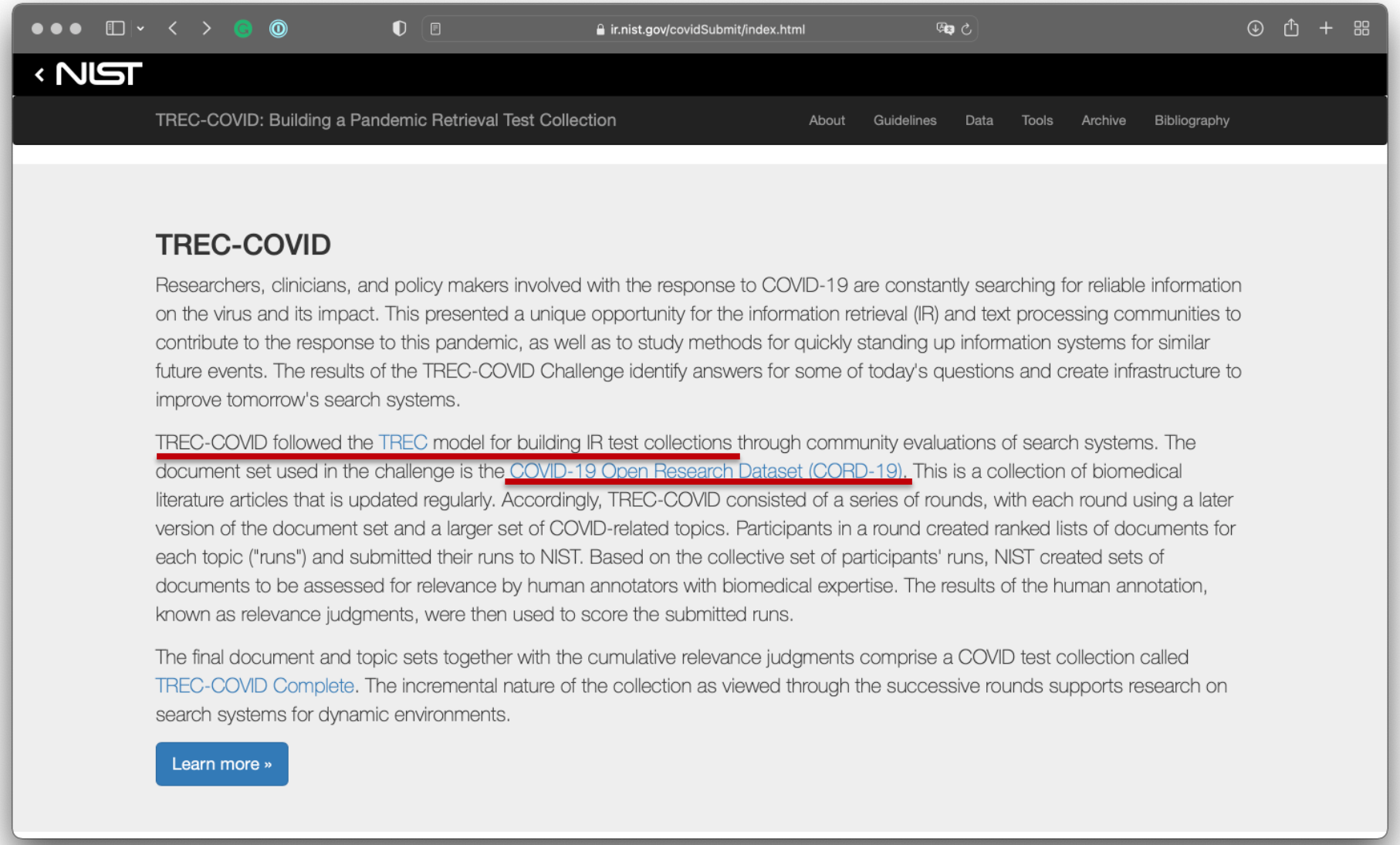
04 – Indexing

Philipp Schaer, Technische Hochschule Köln, Cologne, Germany

Version: WS 2021

Technology
Arts Sciences
TH Köln

A long time ago, in a galaxy…

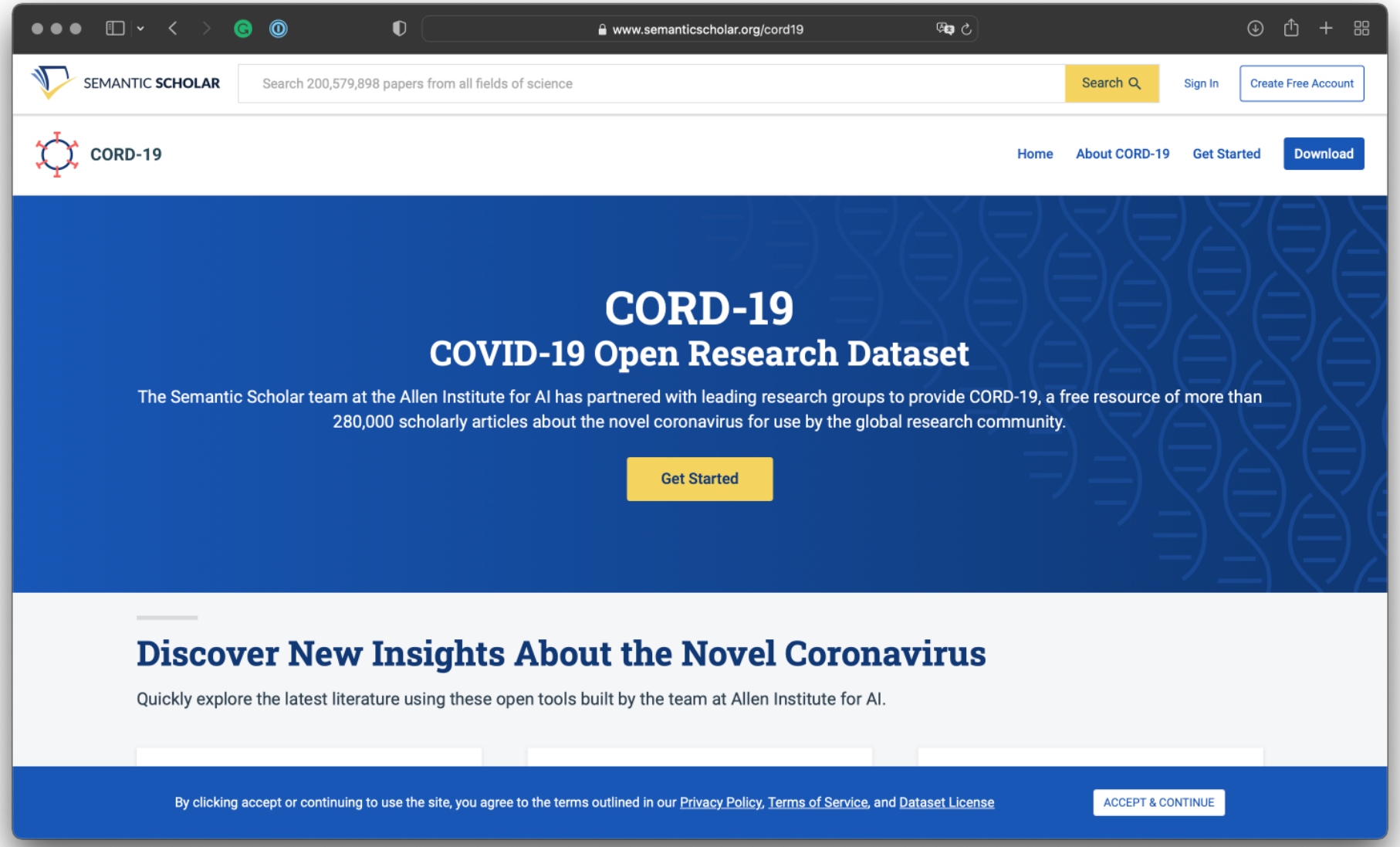


https://ir.nist.gov/covidSubmit/index.html

ir.nist.gov/covidSubmit/data/topics-rnd5.xml

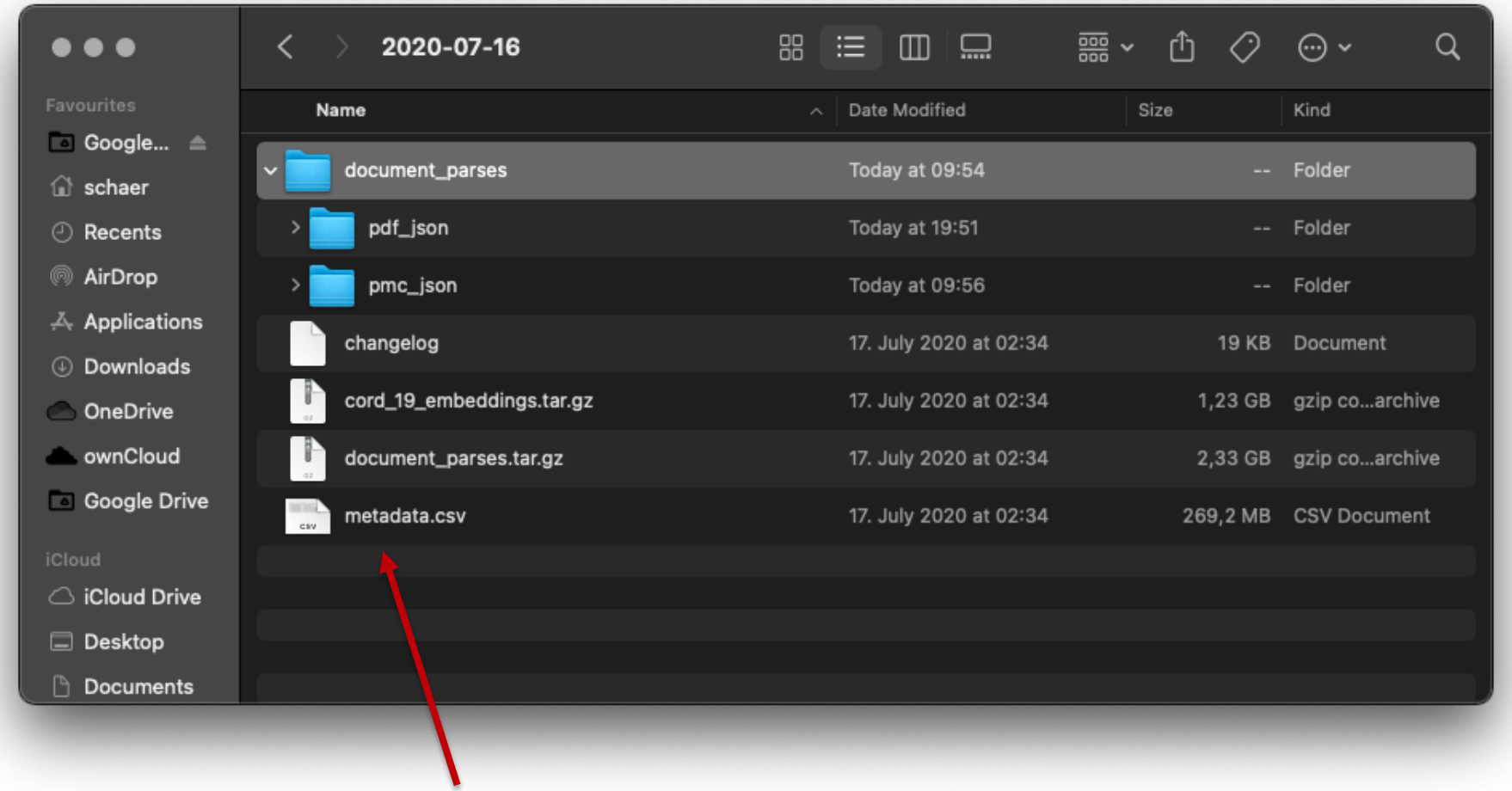
```
<topics task="COVIDSearch 2020" batch="5">
  <topic number="1">
    <query>coronavirus origin</query>
    <question>what is the origin of COVID-19</question>
    <narrative>
      seeking range of information about the SARS-CoV-2 virus's origin,
      including its evolution, animal source, and first transmission
      into humans
    </narrative>
  </topic>
  <topic number="2">
    <query>coronavirus response to weather changes</query>
    <question>
      how does the coronavirus respond to changes in the weather
    </question>
    <narrative>
      seeking range of information about the SARS-CoV-2 virus viability
      in different weather/climate conditions as well as information
      related to transmission of the virus in different climate
      conditions
    </narrative>
```


metadata.csv — 2020-07-16
cord_uid,sha,source_x,title,doi,pmcid,pubmed_id,license,abstract,publish_time,authors,journal,mag_id,who_covidence_id,arxiv_id,pdf_json_files,pmc_json_files,url,s2_id
ug7v899j,d1aafb70c066a2068b02786f8929fd9c900897fb,PMC,"Clinical features of culture-proven Mycoplasma pneumoniae infections at King Abdulaziz University Hospital, Jeddah, Saudi Arabia",10.1186/1471-2334-1-6,PMC35282,11472636,no-cc,"OBJECTIVE: This retrospective chart review describes the epidemiology and clinical features of 40 patients with culture-proven Mycoplasma pneumoniae infections at King Abdulaziz University Hospital, Jeddah, Saudi Arabia. METHODS: Patients with positive M. pneumoniae cultures from respiratory specimens from January 1997 through December 1998 were identified through the Microbiology records. Charts of patients were reviewed. RESULTS: 40 patients were identified, 33 (82.5%) of whom required admission. Most infections (92.5%) were community-acquired. The infection affected all age groups but was most common in infants (32.5%) and pre-school children (22.5%). It occurred year-round but was most common in the fall (35%) and spring (30%). More than three-quarters of patients (77.5%) had comorbidities. Twenty-four isolates (60%) were associated with pneumonia, 14 (35%) with upper respiratory tract infections, and 2 (5%) with bronchiolitis. Cough (82.5%), fever (75%), and malaise (58.8%) were the most common symptoms, and crepitations (60%), and wheezes (40%) were the most common signs. Most patients with pneumonia had crepitations (79.2%) but only 25% had bronchial breathing. Immunocompromised patients were more likely than non-immunocompromised patients to present with pneumonia (8/9 versus 16/31, P = 0.05). Of the 24 patients with pneumonia, 14 (58.3%) had uneventful recovery, 4 (16.7%) recovered following some complications, 3 (12.5%) died because of M pneumoniae infection, and 3 (12.5%) died due to underlying comorbidities. The 3 patients who died of M pneumoniae pneumonia had other comorbidities. CONCLUSION: our results were similar to published data except for the finding that infections were more common in infants and preschool children and that the mortality rate of pneumonia in patients with comorbidities was high.",2001-07-04,"Madani, Tariq A; Al-Ghamdi, Aisha A",BMC Infect Dis,,,,document_parses/pdf_json/d1aafb70c066a2068b02786f8929fd9c900897fb.json,document_parses/pmc_json/PMC35282.xml.json,https://www.ncbi.nlm.nih.gov/pmc/articles/PMC35282/,
02tnwd4m,6b0567729c2143a66d737eb0a2f63f2dce2e5a7d,PMC,Nitric oxide: a pro-inflammatory mediator in lung disease?,10.1186/rr14,PMC59543,11667967,no-cc,"Inflammatory diseases of the respiratory tract are commonly associated with elevated production of nitric oxide (NO•) and increased indices of NO• -dependent oxidative stress. Although NO• is known to have anti-microbial, anti-inflammatory and anti-oxidant properties, various lines of evidence support the contribution of NO• to lung injury in several disease models. On the basis of biochemical evidence, it is often presumed that such NO• -dependent oxidations are due to the formation of the oxidant peroxynitrite, although alternative mechanisms
Line: 1 CSV Tab Size: 4



https://www.semanticscholar.org/cord19

CORD-19 for 2020-07-16



Let's start here…

TREC_EVAL results

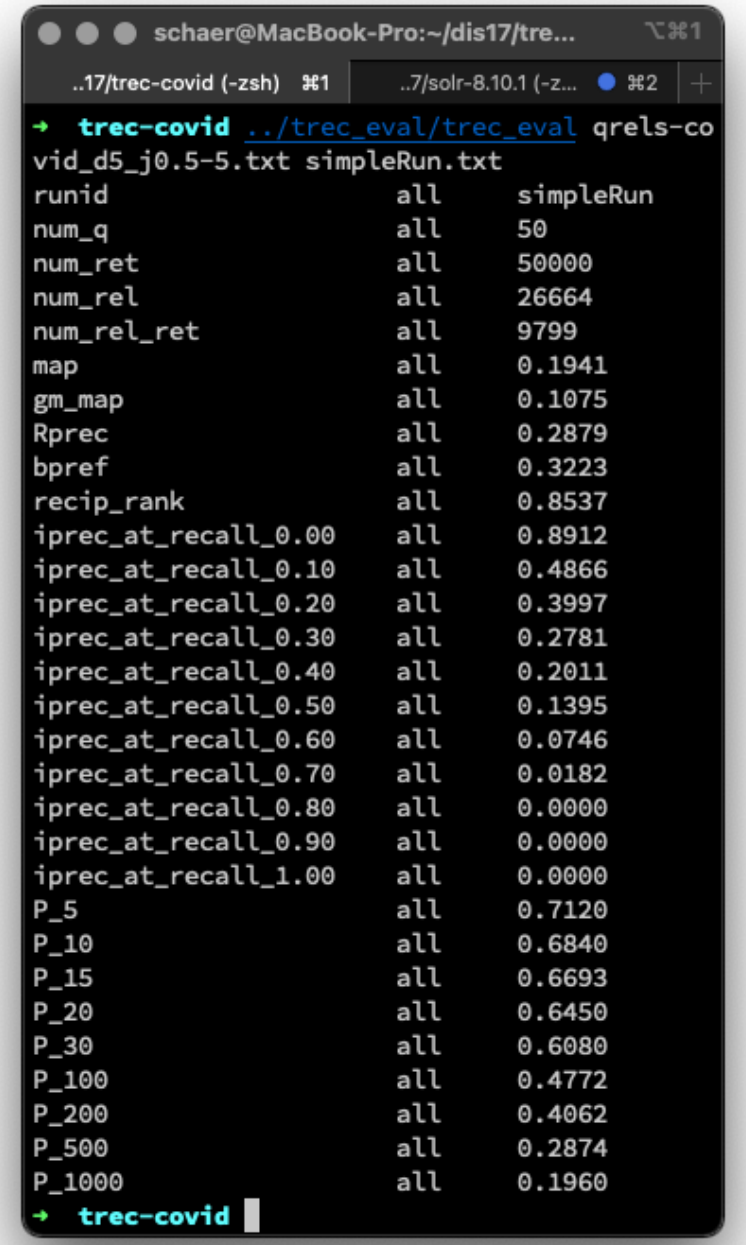
Most simple CORD-19 run ever

- **standard Solr** installation
- indexed **everything** as **text**
- **query** is taken 1:1 from topic file

This leads us to

- precision = 0.196
- recall = 0.367
- map = 0.194

- Not great, not terrible.
- https://castorini.github.io/TREC-COVID/round5/index.html

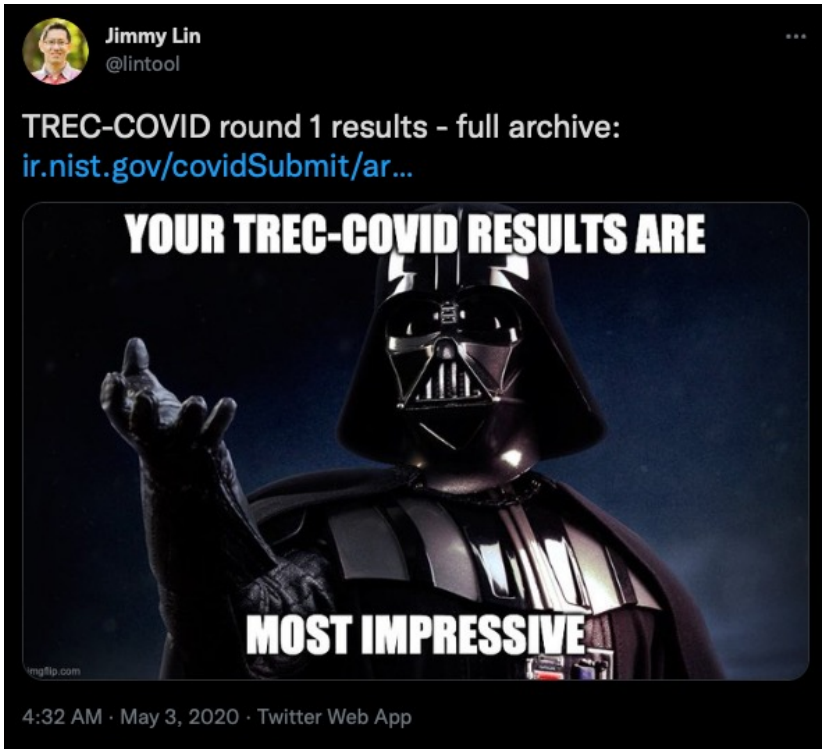


Well…

Best performing systems

- map = 0.4718

"A weighted reciprocal **rank fusion** of (a) **Anserini** & Terrier runs based on **different indexing schemes** and topic variants, including relevance feedback (b) Neural retrieval runs based on synthetic query generation (https://arxiv.org/abs/2004.14503) (c) **TF-Ranking** + multiple **BERT** variants with softmax loss (https://arxiv.org/abs/2004.08476) trained on MS-Marco (d) TF-Ranking + multiple BERT variants and topic variants fine-tuned with softmax loss on the relevance judgments from Rounds 1 - 4. For weighting the runs, we use a simple scheme which doubles the weight of any runs that have access to relevance judgments in the reciprocal rank fusion."



https://castorini.github.io/TREC-COVID/round5/reports/UPrrf102-wt-r5.pdf

What went wrong with our approach?

- data is **not indexed properly**
- we don’t use the full **potential of queries**
- we **don’t use all available data** (embeddings and full-texts)
- we use the **default ranking** algorithm

As a relevance engineer we diagnose the odd results.

- Why do certain documents match query terms?
- Why did less relevant documents rank higher?

Index for organizing data

- DBMS → Databases → Tables → Columns/Rows
- Search Engine → Indices → Types → Documents with Properties

- An search engine (e.g., Solr or ElasticSearch) contains
  - Indices (⇔ databases), that contain
  - Types (⇔ tables), that hold
  - Documents (⇔ rows), that have
  - Properties (⇔ columns)

- Indices can be considered convenient data organization mechanisms, with added performance benefits **depending on how you set up your data.**

Ingesting data into the index

Many options to ingest data into an index

- Upload via a **GUI**
  - native in Solr; Kibana or others for Elastic
- Using **shell command** and “java -jar” calls
  - works if the index (e.g. Lucene or solr) has the specific java classes
- Using **further tools** and services
  - e.g. logstash for Elastic or Postman
- **CURL** commands in shell and command line
  - PUT: ingest single elements to and index
  - POST: ingest documents in a bulk

- They all do the same thing: **put documents into the index**

CURL

```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

This example uses the following variables:

`<VERB>`
The appropriate HTTP method or verb. For example, `GET`, `POST`, `PUT`, `HEAD`, or `DELETE`.

`<PROTOCOL>`
Either `http` or `https`. Use the latter if you have an HTTPS proxy in front of Elasticsearch or you use Elasticsearch security features to encrypt HTTP communications.

`<HOST>`
The hostname of any node in your Elasticsearch cluster. Alternatively, use `localhost` for a node on your local machine.

`<PORT>`
The port running the Elasticsearch HTTP service, which defaults to `9200`.

`<PATH>`
The API endpoint, which can contain multiple components, such as `_cluster/stats` or `_nodes/stats/jvm`.

`<QUERY_STRING>`
Any optional query-string parameters. For example, `?pretty` will *pretty-print* the JSON response to make it easier to read.

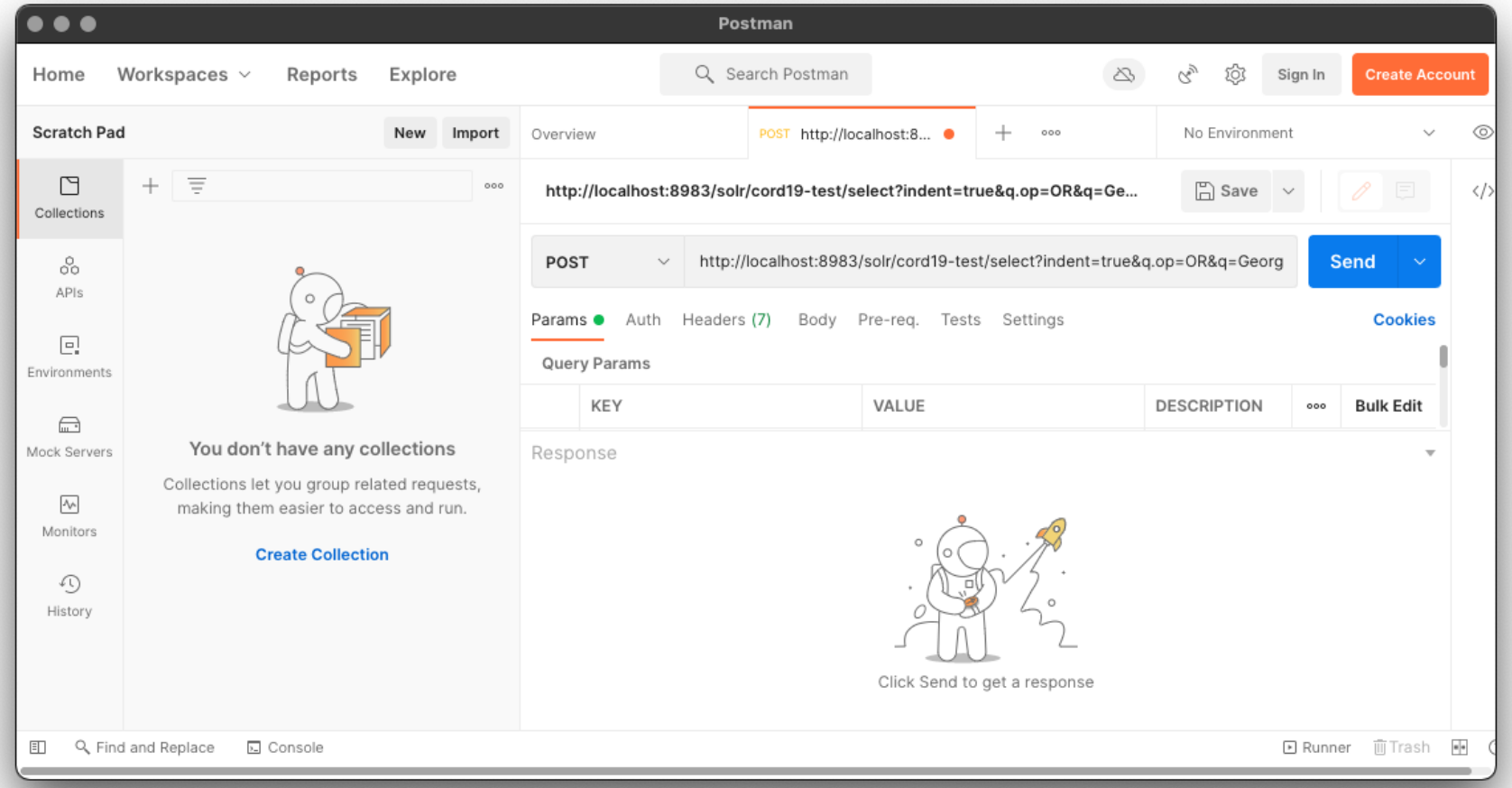
`<BODY>`
A JSON-encoded request body (if necessary).

- **curl** enables us to talk to Web APIs
- response is a HTTP status code like 200 for “OK”
- also returns a JSON-encoded response body (if set as return method (except HEAD requests))
- if security features are enabled, one must also provide a valid username and password

But, do we really want to do everything in the shell / command line?

Using Programming Languages or tools

One can use API programming tools like Postman



Using Programming Languages or tools

Or simply use **Python**

- Import requests library to let Python talk to Web APIs
- we can concentrate on the parameter and the data
- We can use Jupyter Notebooks to better explain what we do
- Automatic ability to parse JSON in Python
- Translate CURL commands to commands in different programming language: https://curl.trillworks.com/

curl command

```
curl "localhost:9200/_cat/indices?v"
```

Python requests

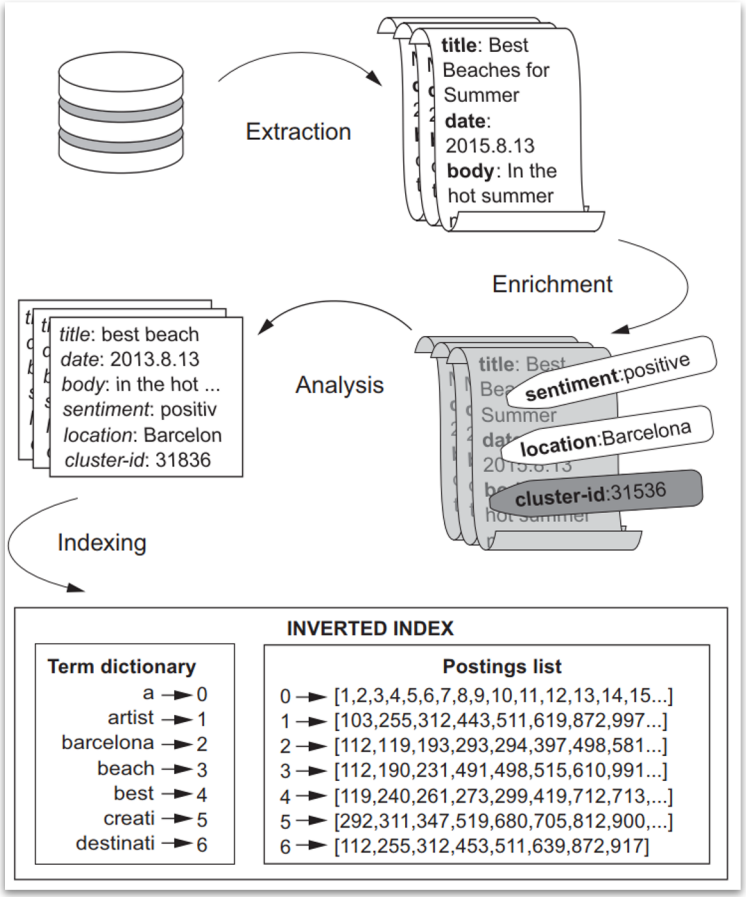
```
import requests

params = (
    ('v', ''),
)

response = requests.get('http://localhost:9200/_cat/indices', params=params)

#NB. Original query string below.
# response = requests.get('http://localhost:9200/_cat/indices?v')
```

Queries are Matching the Data Structure

- Data must be analyzed to reduce “alienness” between documents and queries
  - For example, we need to identify **meaningless terms**. Otherwise they will produce a lot of noise in our result sets
- Only after you understand how the underlying data structures are created and accessed, you can take control of improving results

We need a well-defined ETL-process

ETL - Extract, Transform, Load



Extracted documents from a database, an API or from a file

- each line in a CSV is an extracted document

Enrich documents with further data

- entries are put in structure and are edited
- entries can be enriched with further data

Analyze the documents to transform for better search results

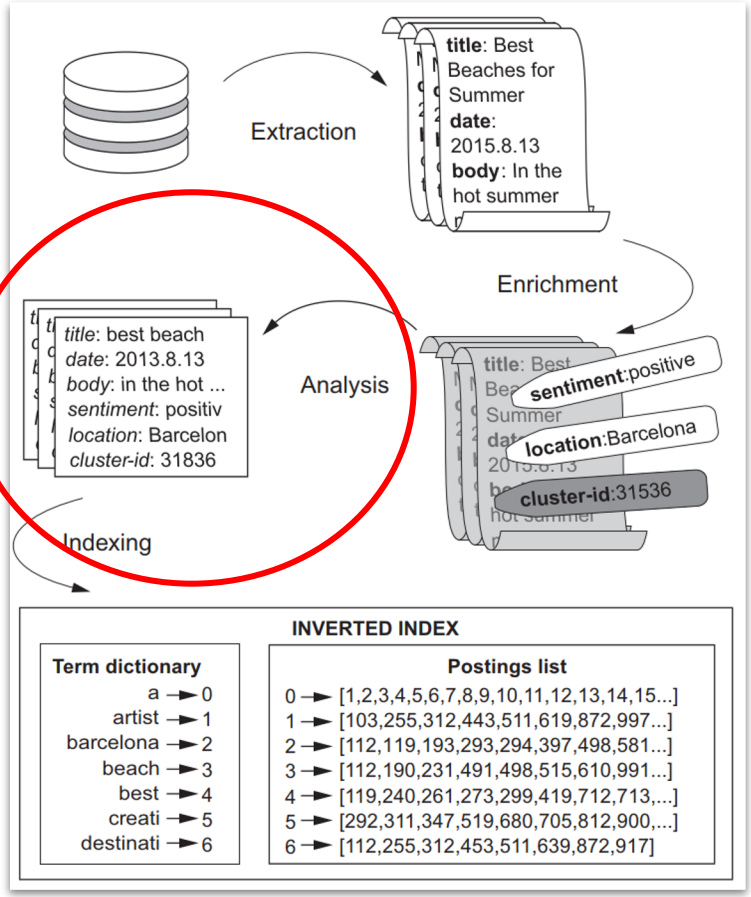
- converts document text into manually specified normalized tokens for matching

The transformed data is **indexed** into the defined data structures

- definition of fields and field types
- definition of copy fields

Picture from: Turnball at al., Relevant Search - With applications in Solr and ElasticSearch, Manning, 2016

Analyzing Data for Indexing



Analysis is very important to search relevance

- Analysis transforms raw text and data from the documents into tokens
- These tokens represent the document's features.
- Engineering these to match features from a user's query is **critical to satisfy the user's information need.**
- Hence, **analysis is one of the most important step when indexing data** into a search engine

Picture from: Turnball at al., Relevant Search - With applications in Solr and ElasticSearch, Manning, 2016

Tokenizing Documents and Data

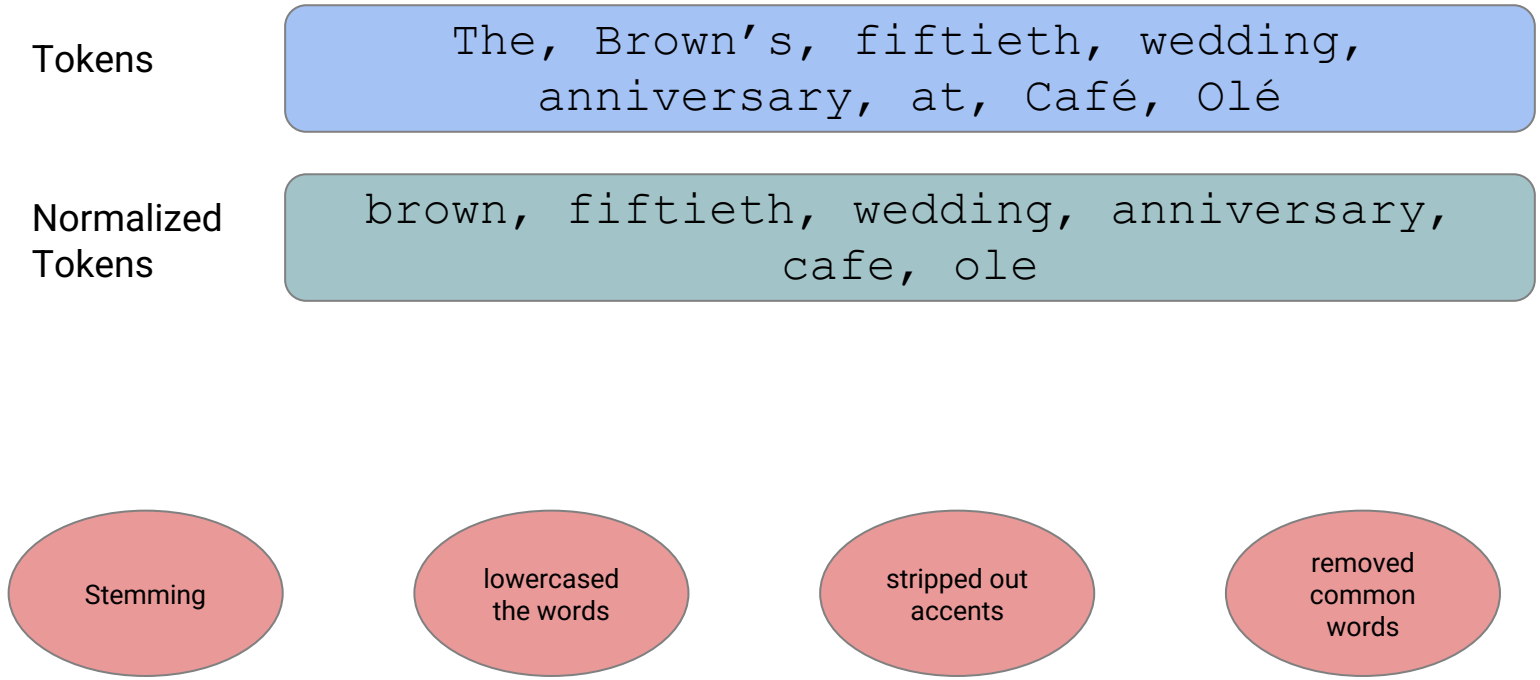
Search engines match tokens between documents and queries

- **Tokenizing** defines basics of the matching process
- We should spend much time fine-tuning this analysis step to control as much as possible when matches occur
- This step is happening in the search engine, i.e., it is a part of Solr and Elastic
  - we send the data to the search engine with some specification. The search engine converts the documents data into tokens according to these specifications. Only then it stores the data within the internal data structures.

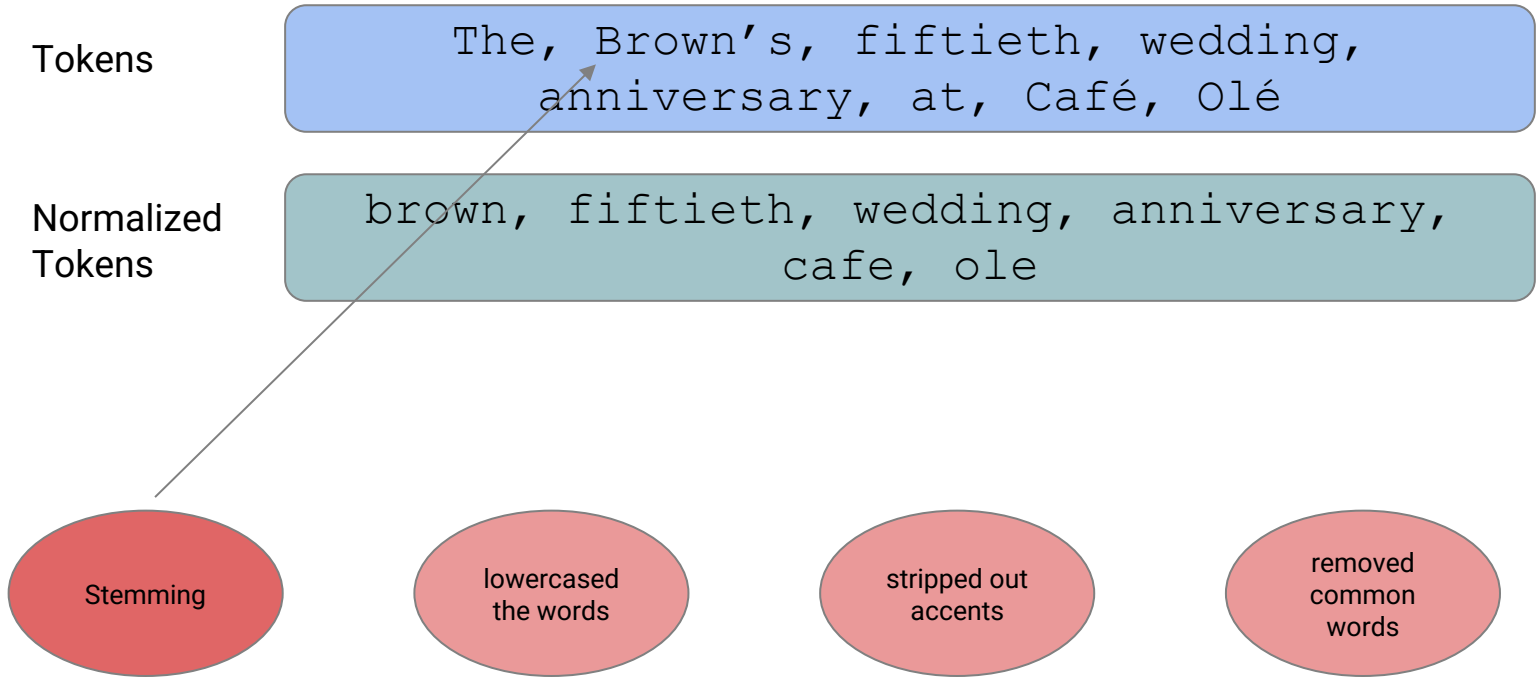
Tokens

Sentence	“The Brown’s fiftieth wedding anniversary, at Café Olé.”
Tokens	`The, Brown’s, fiftieth, wedding, anniversary, at, Café, Olé`
Normalized Tokens	`brown, fiftieth, wedding, anniversary, cafe, ole`

Normalized Tokens



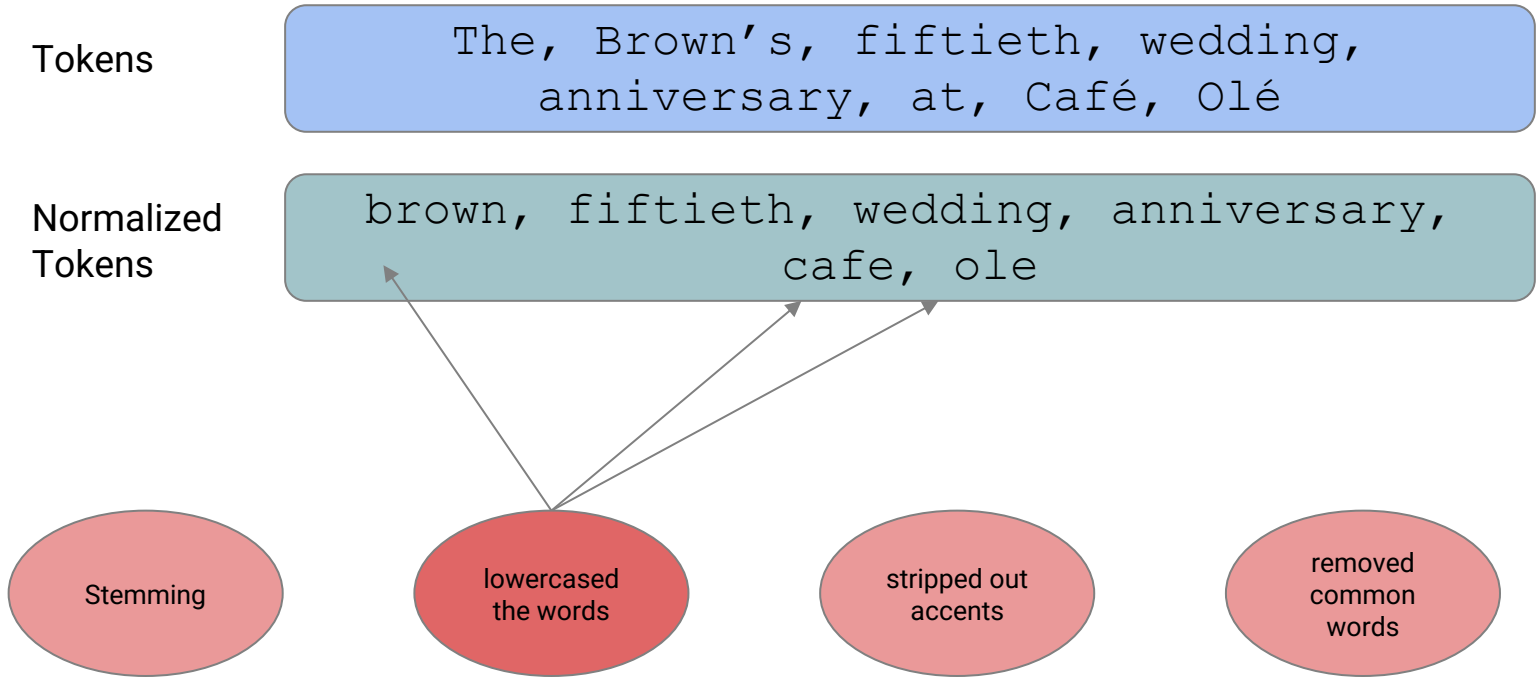
Normalized Tokens



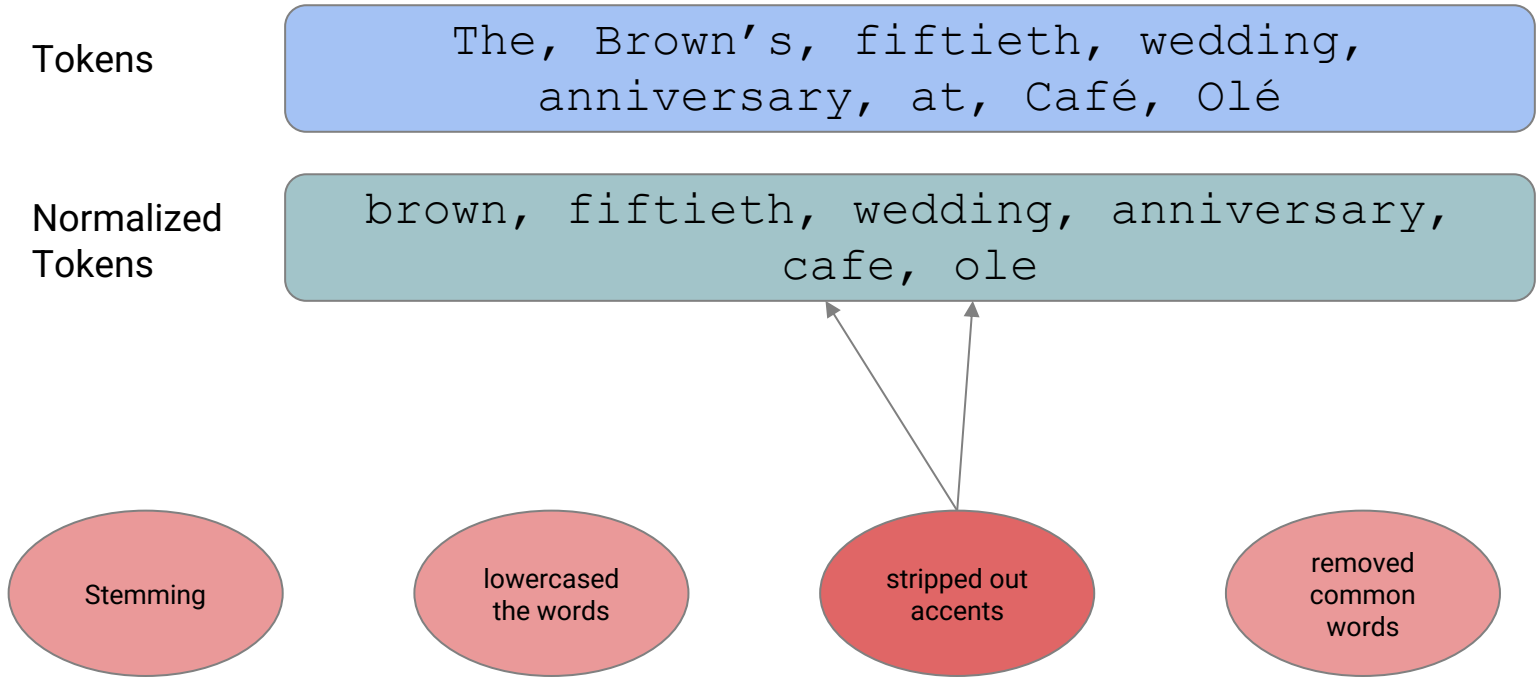
other examples:

going → go

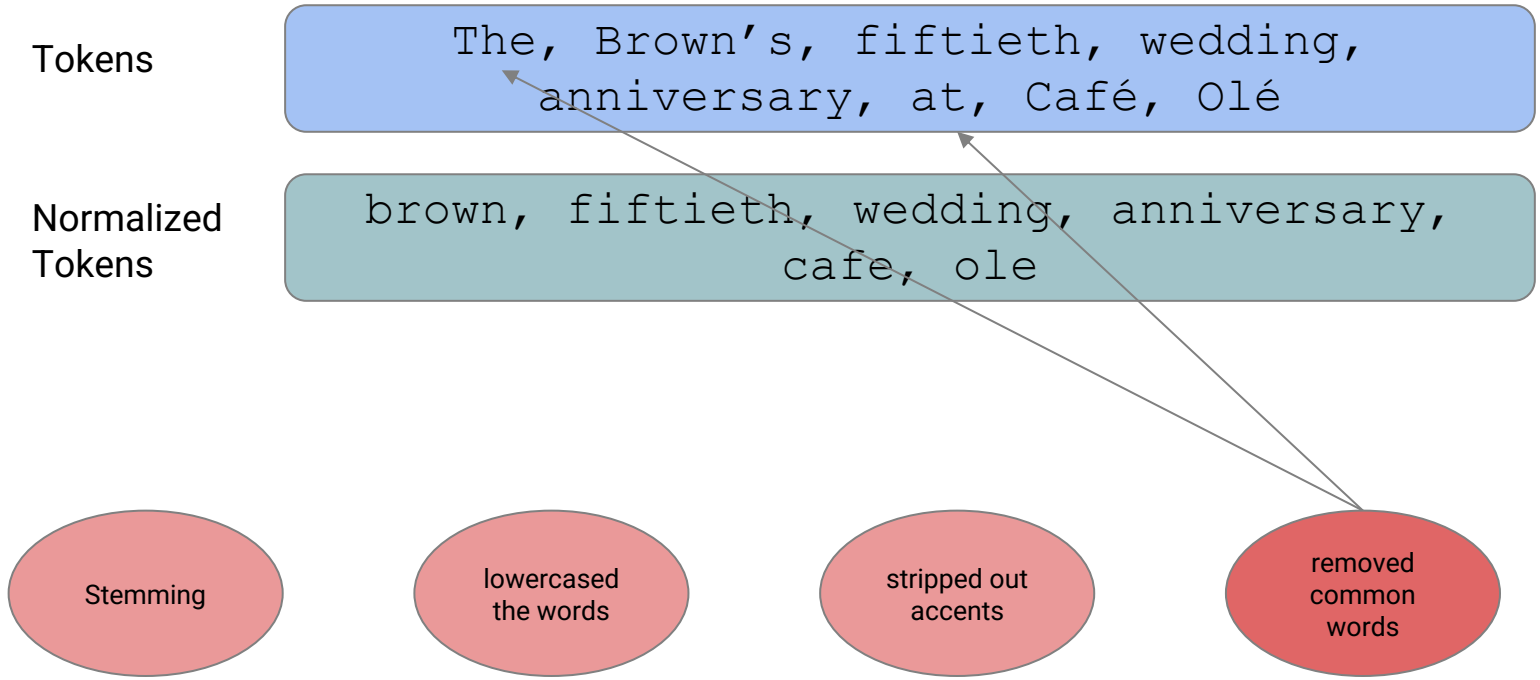
Normalized Tokens



Normalized Tokens



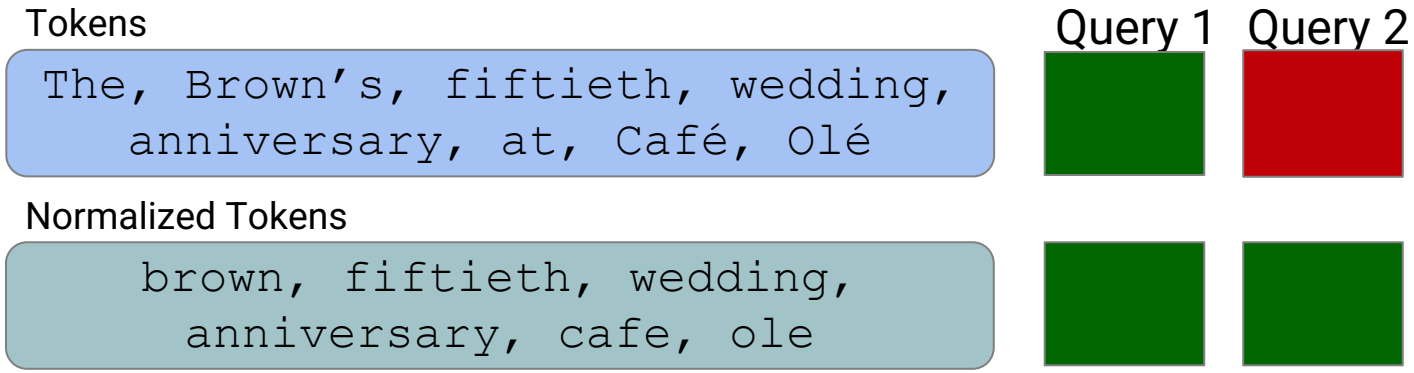
Normalized Tokens



Why Normalized Tokens

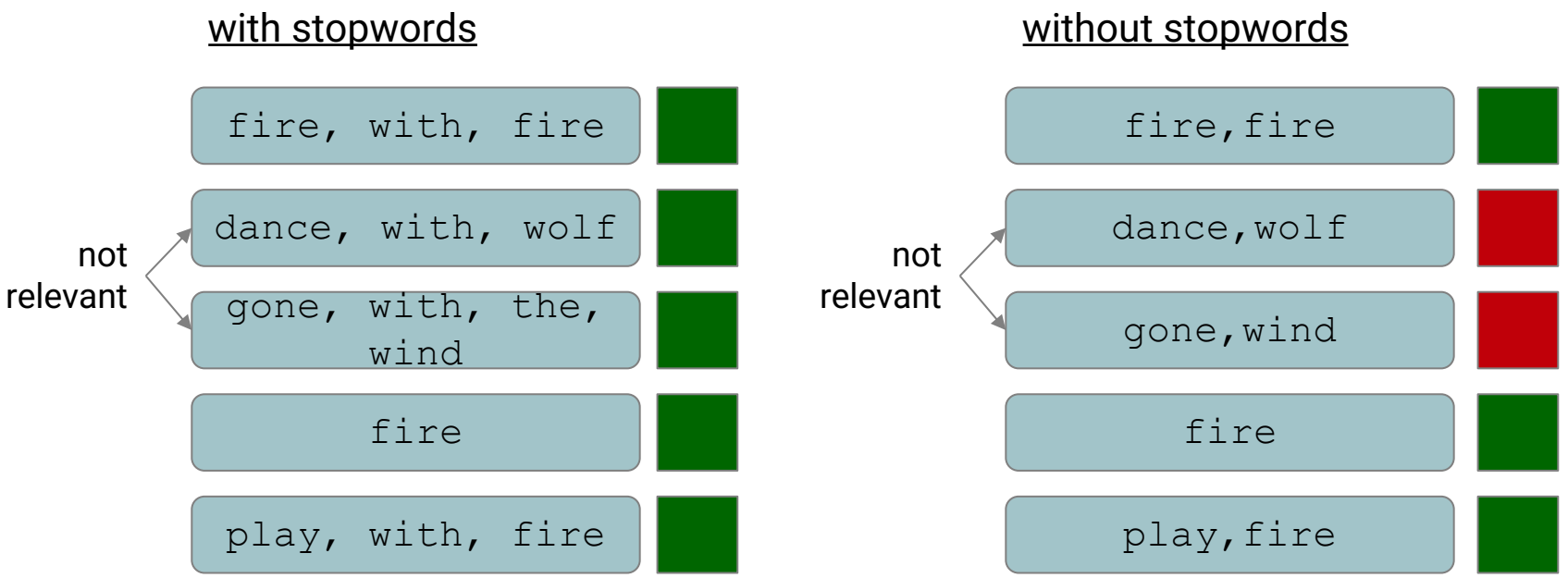
Queries:

- “fiftieth wedding anniversary” → fiftieth, wedding, anniversary
- “Brown Café Olé” → brown, cafe, ole



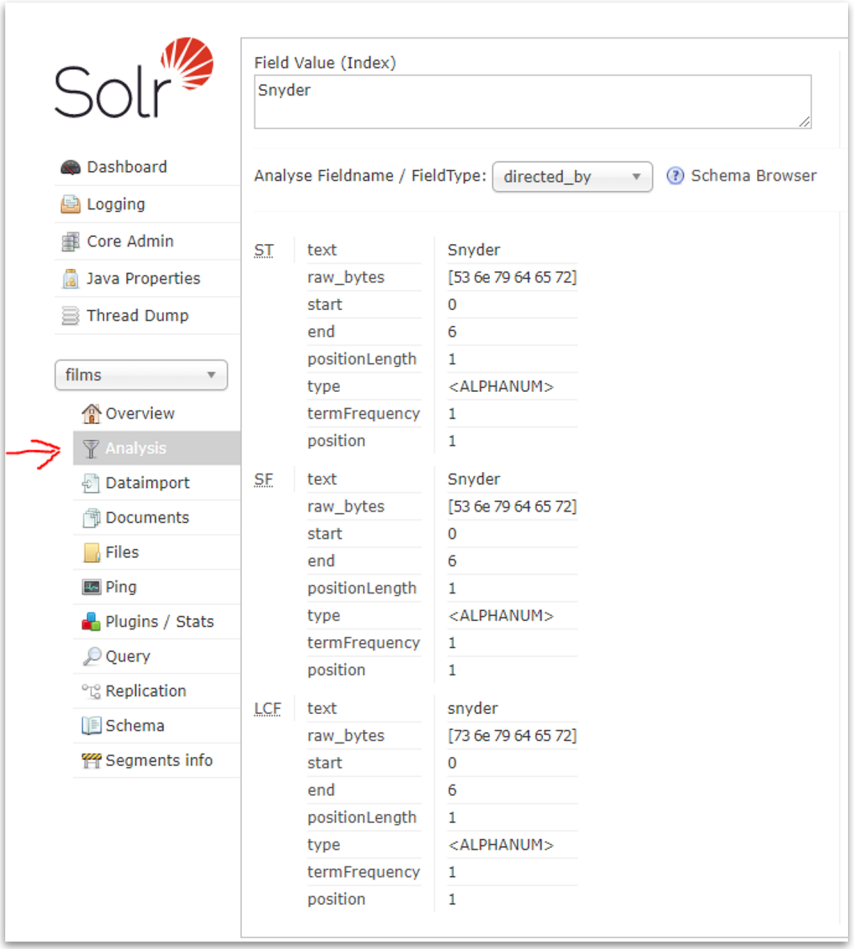
Why Removing Stopwords?

- Query: “Fire with Fire” → fire, with, fire



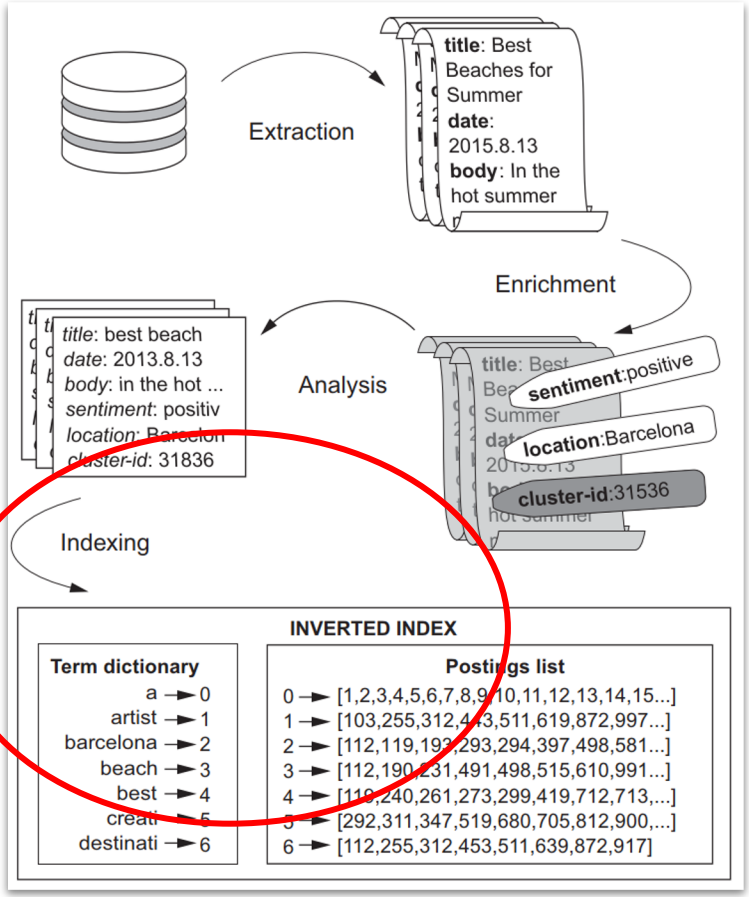
Debugging Analysis

- To understand, why our retrieval results are not as good as we want, we must use our knowledge about document analysis to debug the analyzer
- Solr can use its GUI to analyze documents and queries



https://lucene.apache.org/solr/guide/8_10/schema-api.html

Analyzing Data for Indexing



- Which pieces of data should be indexed, i.e., should we index all the documents' metadata or **maybe just a subset**?
- which data structures should be used to index the data, e.g., is it sufficient to store strings in text fields, or maybe **language specific**?
- Different indexing strategies have an **impact on search** relevance

Picture from: Turnball at al., Relevant Search - With applications in Solr and ElasticSearch, Manning, 2016

Indexing after Analysis

For each data element (e.g. CSV column), we need to define

- **Field**: specifies which concrete data fields are of which type whether this data field is indexed, whether it is stored, whether it is a mandatory field, multi-valued, etc.

In addition, we generally define

- **Field Type**: Indicates which types of data there are and how they are processed, i.e., with or without stop words, which tokenizer, stemmer and much more
- **Dynamic Field**: Is just like a regular field but has a name with a wildcard in it.
- Other entries like **Copy Field**, that is searched if no field is provided for the query

If you analyze your data and queries after you have indexed your data, you must re-index your entire data after the change

Field Type

Define how a specific type of data should be processed

- Specify analysis pipeline here
- Specify for which type of the data the pipeline should be applied to

- Example in Solr's schema.xml: German Text

```
<fieldType name="text_de" class="solr.TextField"  positionIncrementGap="100">
   <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.StopFilterFactory" format="snowball"
                    words="lang/stopwords_de.txt" ignoreCase="true"/>
      <filter class="solr.GermanNormalizationFilterFactory"/>
      <filter class="solr.GermanLightStemFilterFactory"/>
   </analyzer>
</fieldType>
```

Field

Define which data items (e.g., CSV columns) are indexed with which field type

- Align your data to the field types you have defined
- Specify whether the data should be indexed or even stored, multi valued, ...
  - **indexed** = can be used for searching → tokenized representation
  - **stored** = can be used for presentation to user → original content
  - **multi valued** = define whether multiple values per field are possible → list of items (tags)

- Example in Solr's schema.xml:

```
<field name="id" type="string" multiValued="false" indexed="true"
          required="true" stored="true"/>

<field name="title" type="text_de" multiValued="true" indexed="true"
          stored="true"/>
```

Dynamic Field

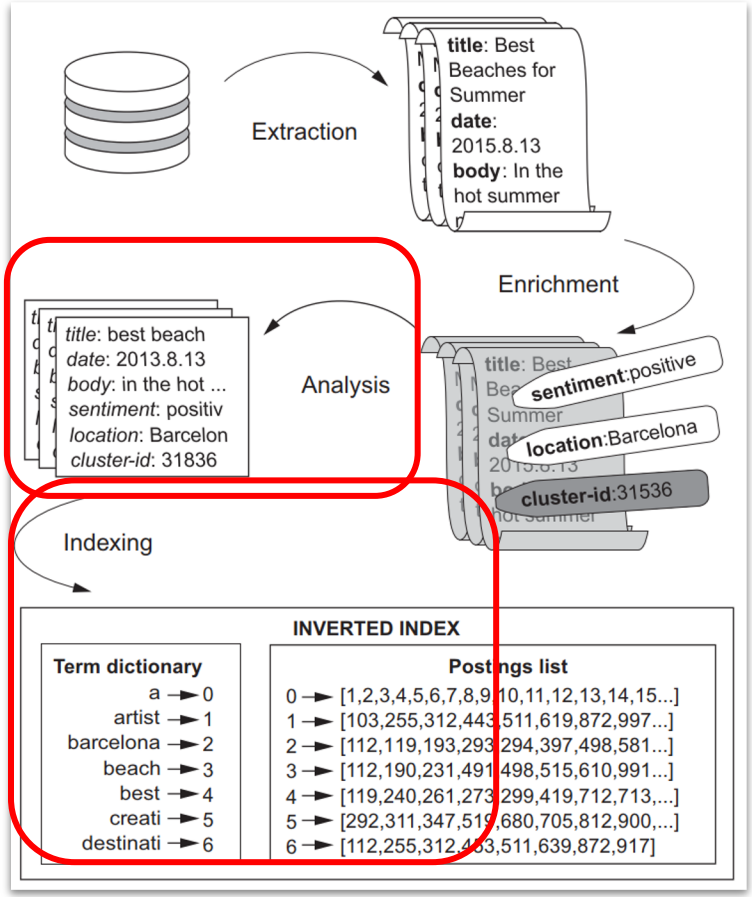
If many data can be put into the index the same way, one can create dynamic fields

- Specify the wildcard-suffix and the type of the dynamic field
- Specify whether the data should be indexed, stored, multi valued
- **Additionally**: you must name your data item (e.g., CSV column) with the same suffix, before indexing
  - e.g. “abstract” → “abstract_txt_en”

- Example in Solr’s schema.xml:

```
<dynamicField name="*_txt_en" type="text_en" indexed="true"
              stored="true"/>

<dynamicField name="*_txt_de" type="text_de" indexed="true"
              stored="true"/>

<dynamicField name="*_t" type="text_general" indexed="true"
              stored="true"/>
```

So, what should we focus on now?



Focus on **document analysis** and indexing

- character filtering, tokenization, token filtering
- Solr: field types, fields, dynamic fields

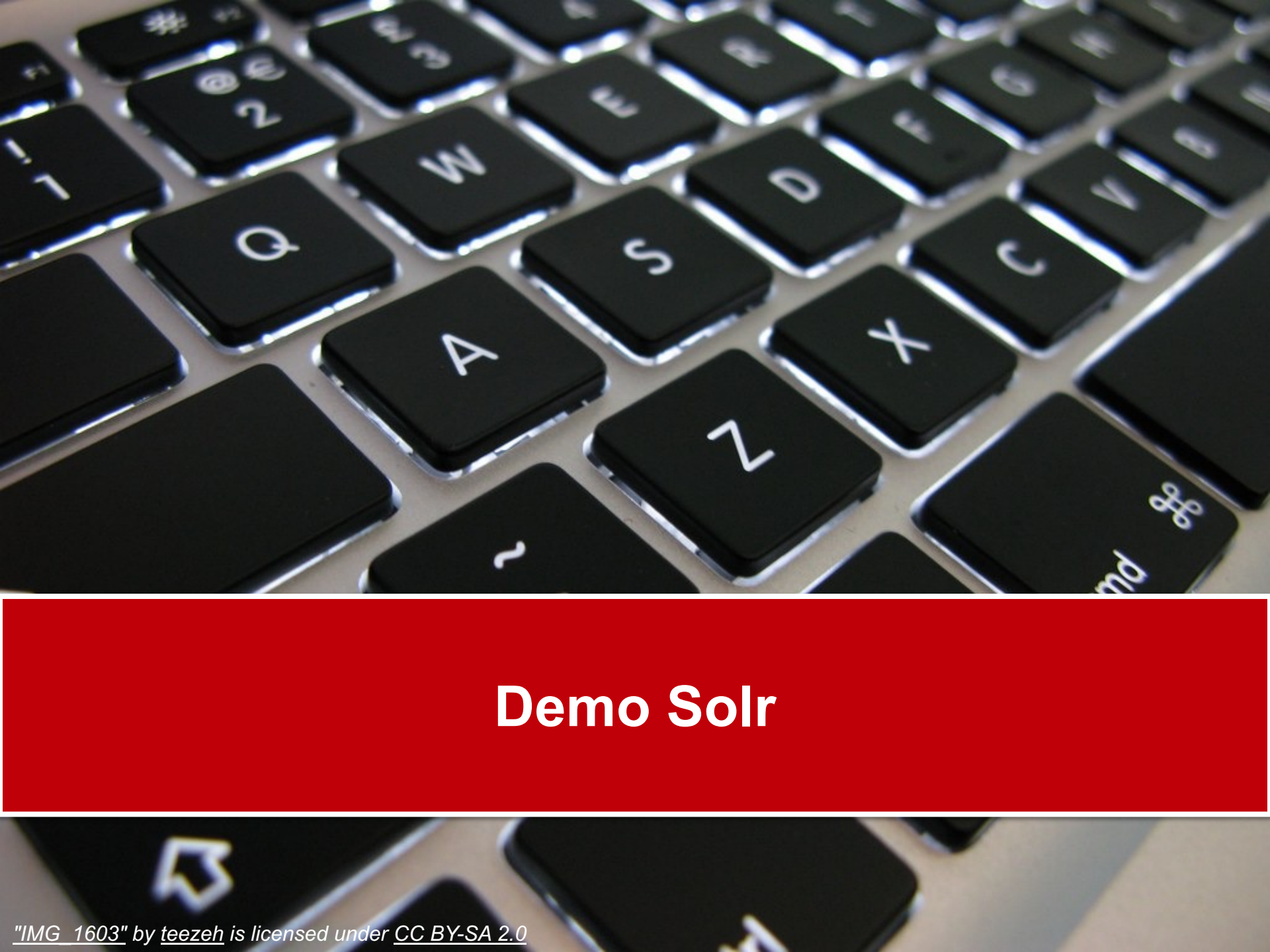
Focusing on **enrichment** is encouraged

- how to use further data (full text, embeddings)
- clean the data, e.g., separate list elements

Focus also on **query analysis**

- How does Solr tokenize your queries
- Maybe extend your queries with further information given in the topics file

Picture from: Turnball at al., Relevant Search - With applications in Solr and ElasticSearch, Manning, 2016



Demo Solr