Re-thinking significant term discovery

Mark Harwood 25/2/26
@elasticmark

Some background

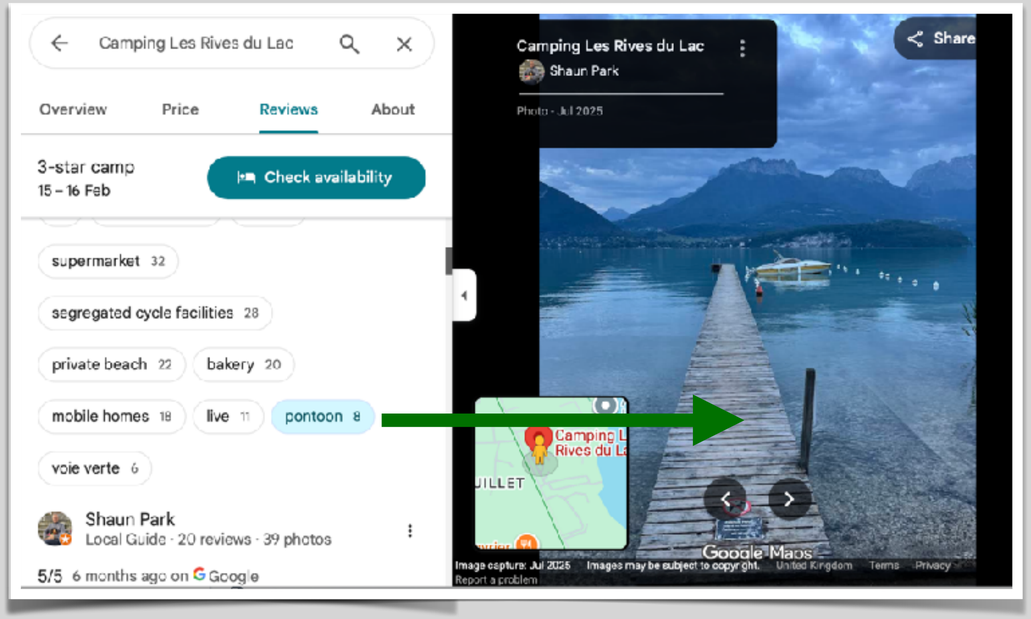
- About me:
  - Early Lucene contributor 
  - Early elastician (employee #30) 
  - Currently tinkering on hybrid search interfaces

Text summarisation with significant terms

Example applications

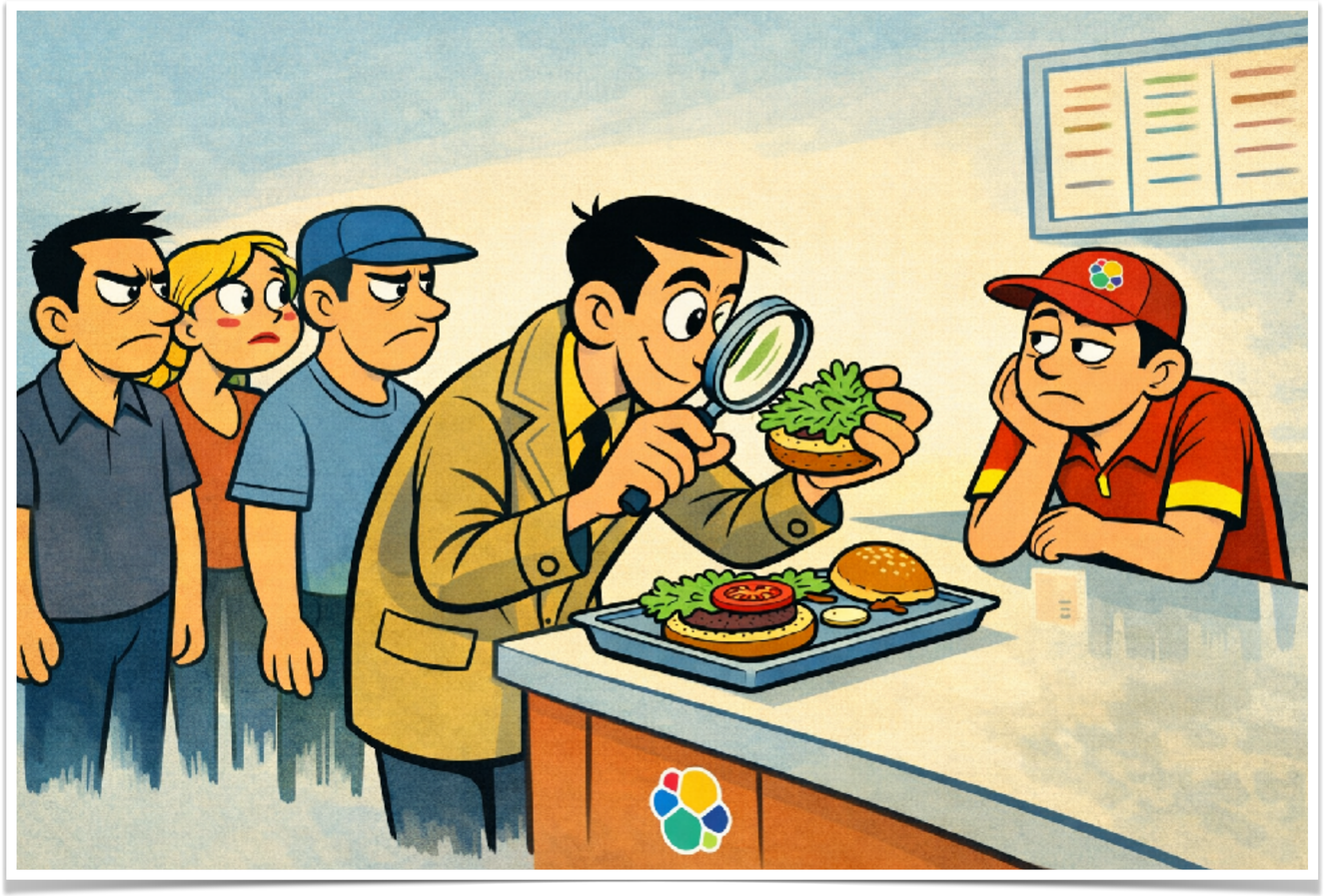
News cluster labels

Google Maps reviews



Elasticsearch’s text analytics

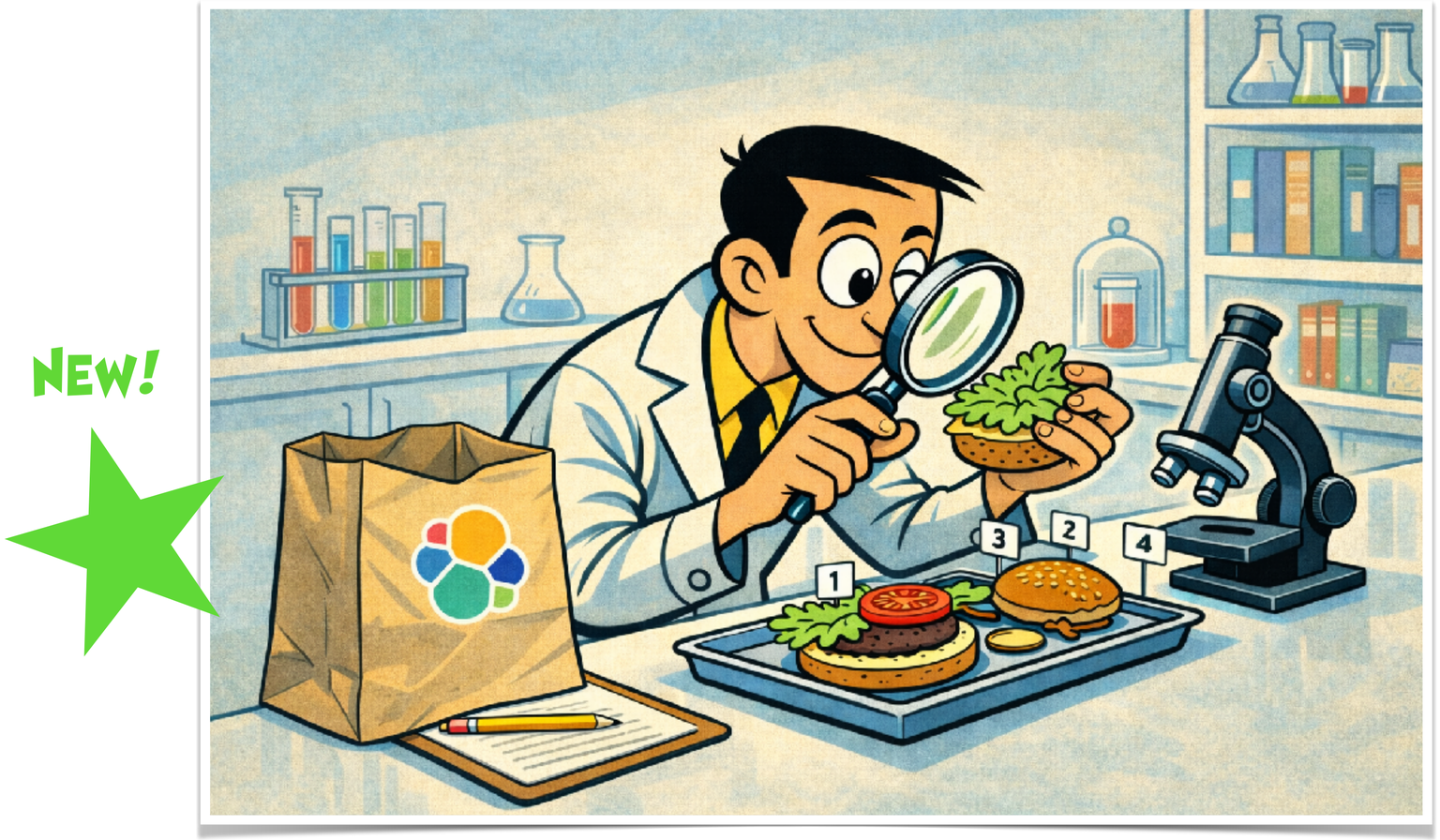
Significant term discovery can be expensive



Expensive aggregations can cause problems

Client-side text analytics

Same discovery algorithms (and more…)



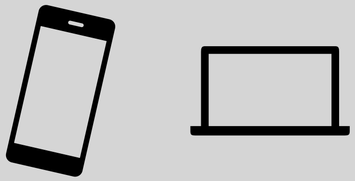
Same analysis, better location

Text summarisation options

Where to compute?

NEW!

On your clients?



FREE!

Significant keywords and **phrases**

In your elasticsearch cluster?

$$

Significant keywords

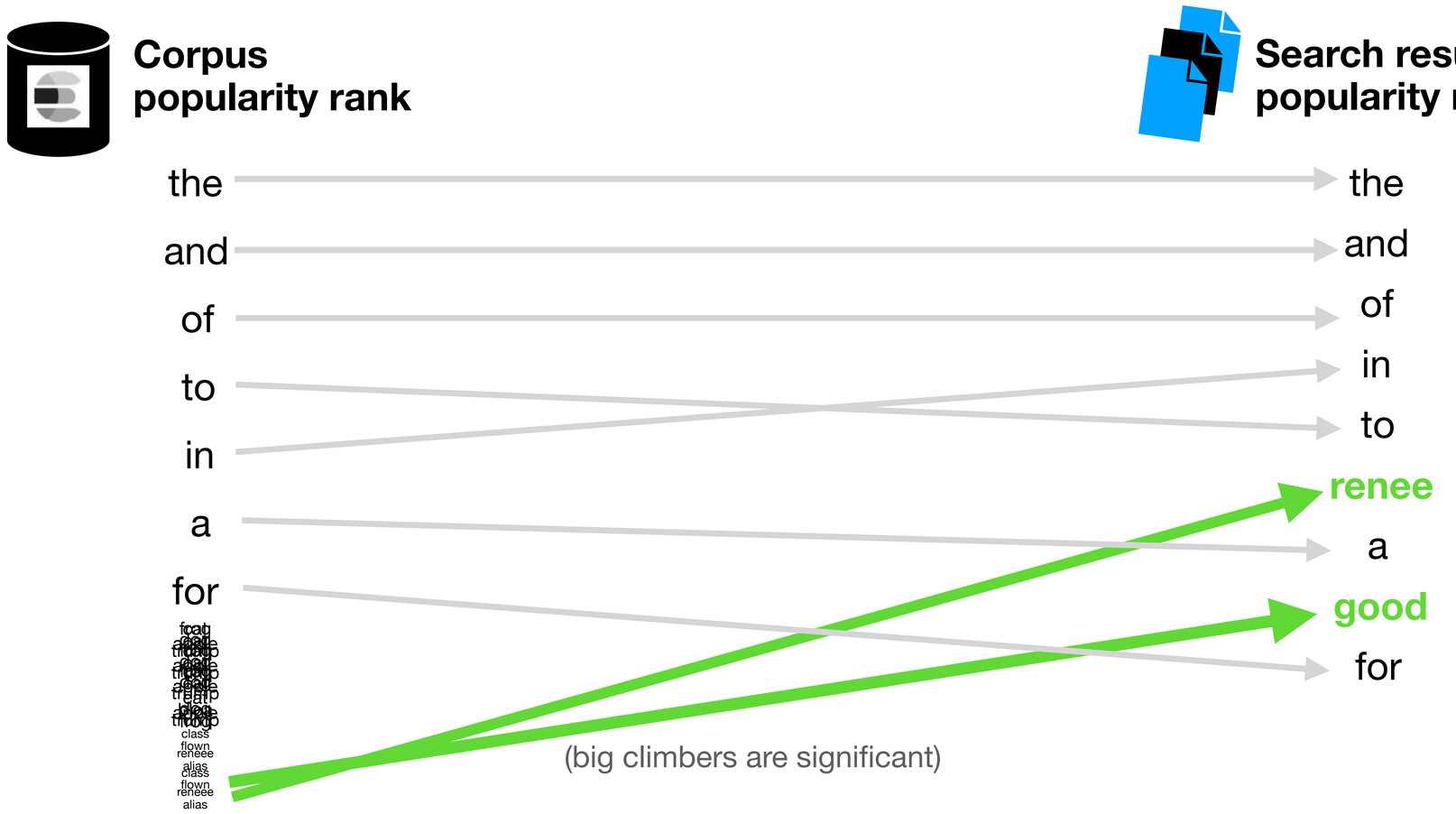
LLM Provider?

$$$

Unstructured and structured summaries

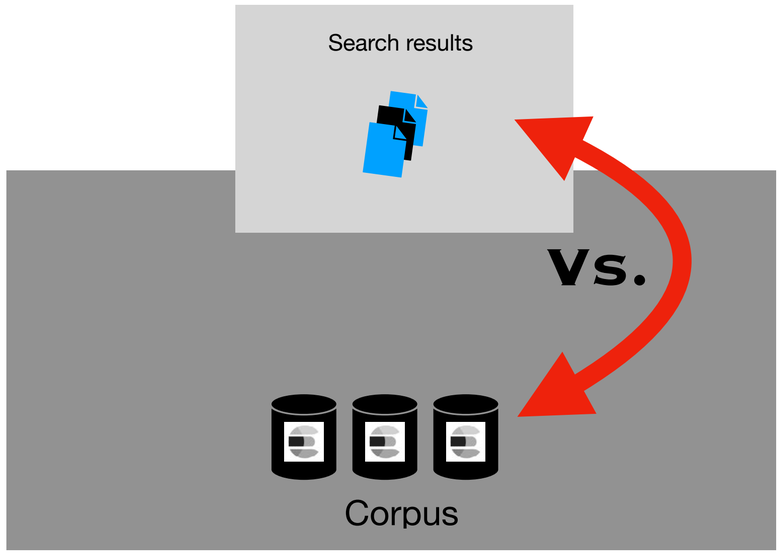
Recap: how elasticsearch determines significance

Foreground vs background word popularity



Comparison issues in elasticsearch

What makes this particular analysis expensive?



- **Tokenisation costs**
  the top search results have to have their text re-analyzed to discover the foreground terms

- **Fragmented background data**
  Each term's background frequency is expensive to derive as it is spread across indexes → shards → on-disk segments. Date-based indices actually make it impossible to answer *"what is trending today (compared to previously)?"*

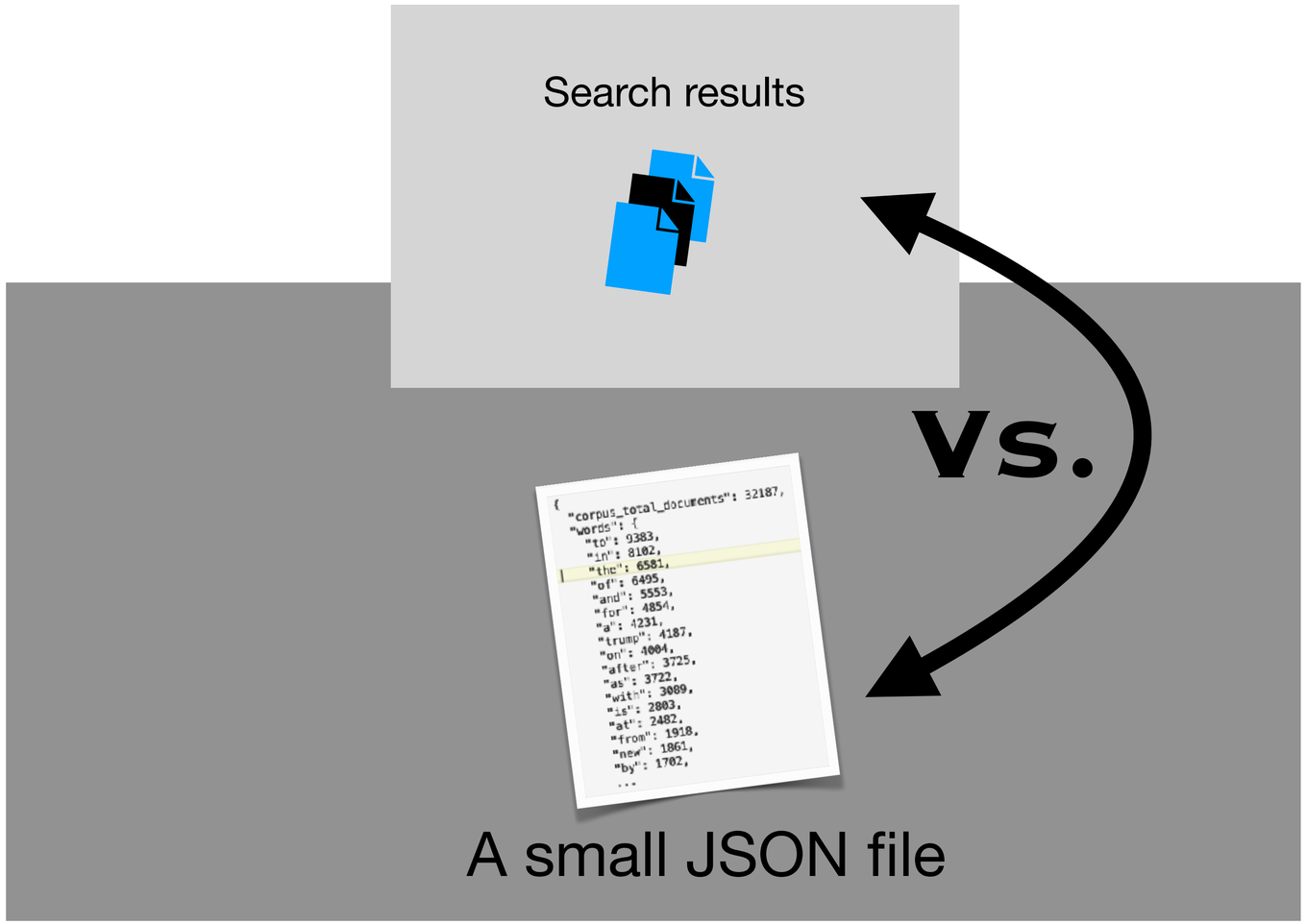
- **Memory pressure**
  large dictionaries / heaps push JVM heap; circuit-breakers and GC churn kick in.

- **IO and network**
  segment seeks, shard fan-out, and cross-node reduces add latency and cost.

- **Shared resources**
  heavy BG lookups run alongside everyone else's queries, competing for CPU/heap/I/O. Risk of hot shards, cache churn, and GC pauses can reduce cluster stability

Clientside alternative

Foreground vs background word popularity…

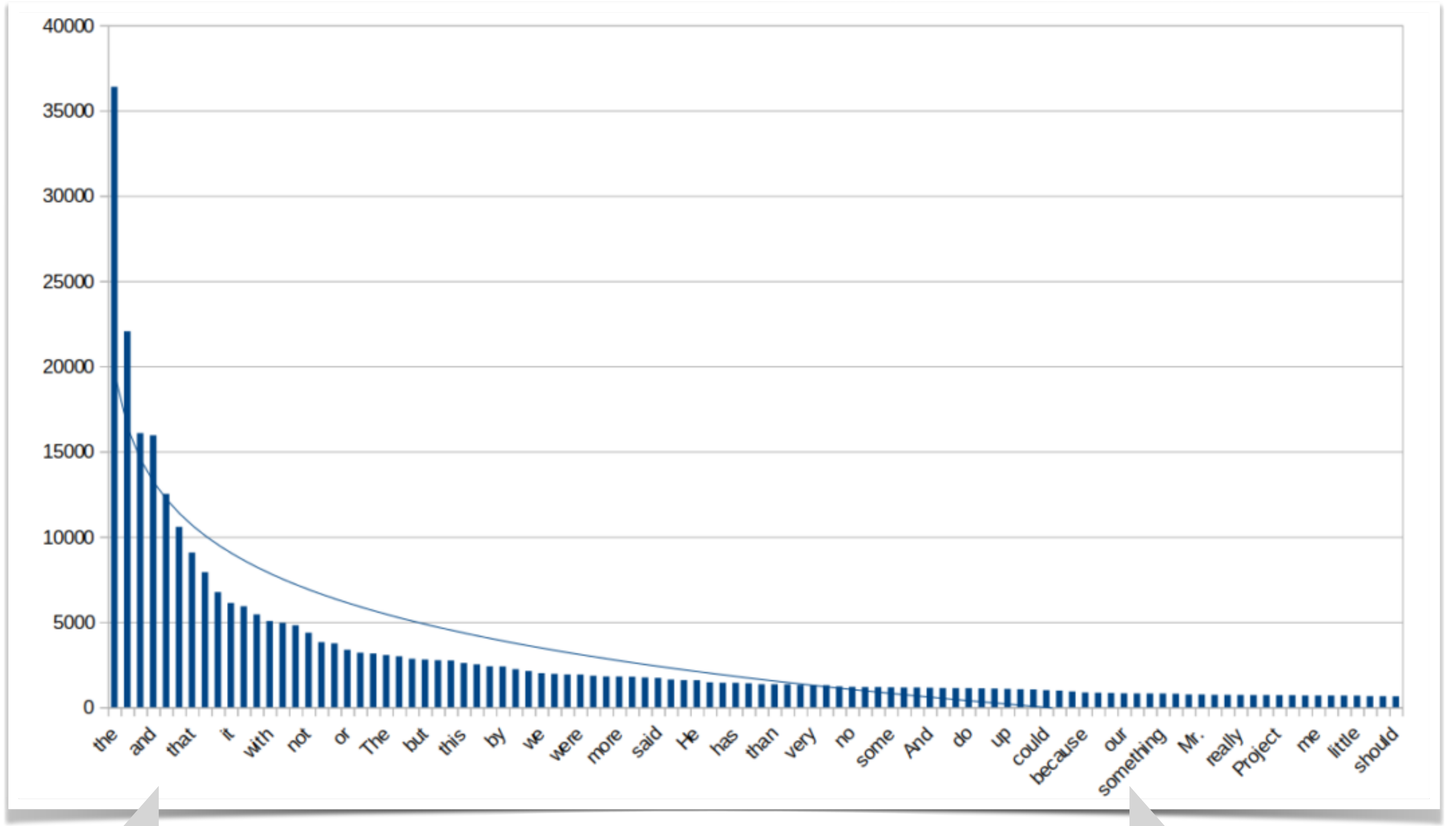


How much data is needed for a background?

Zipf’s law works in our favour…


```
{
  "corpus_total_documents": 32187,
  "words": {
    "to": 9383,
    "in": 8102,
    "the": 6581,
    "of": 6495,
    "and": 5553,
    "for": 4854,
    "a": 4231,
    "trump": 4187,
    "on": 4004,
    "after": 3725,
    "as": 3722,
    "with": 3089,
    "is": 2803,
    "at": 2482,
    "from": 1918,
    "new": 1861,
    "by": 1702,
    ...
```

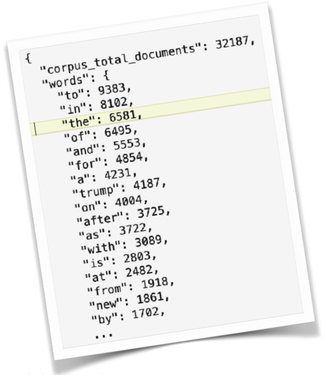

(<100kb zipped)



(~20k top words)

Does it really work?

Can a small background sample produce quality results?



A JSON file of ~20k popular words and frequencies

Sampled background popularity rank

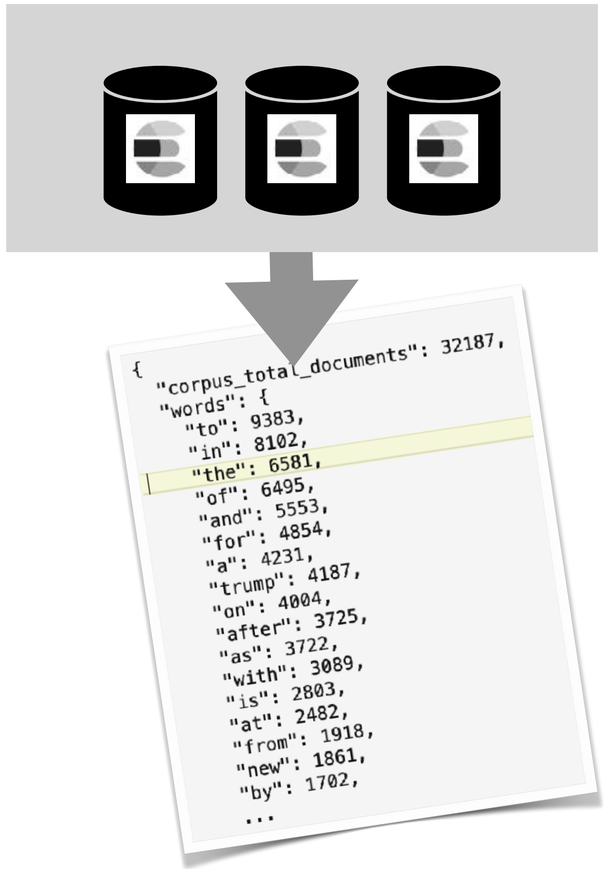
Search results popularity rank

the
and
of
to
in
a
for
class
flown
renee
alias
class

the
and
of
in
to
renee
a
good
for

(missing words are given a low default popularity)

Exporting word popularities from your index



```
{
  "corpus_total_documents": 32187,
  "words": {
    "to": 9383,
    "in": 8102,
    "the": 6581,
    "of": 6495,
    "and": 5553,
    "for": 4854,
    "a": 4231,
    "trump": 4187,
    "on": 4004,
    "after": 3725,
    "as": 3722,
    "with": 3089,
    "is": 2803,
    "at": 2482,
    "from": 1918,
    "new": 1861,
    "by": 1702,
    ...
```

```
GET myindex/_search
{
  "query": {
    "function_score": {
      "query": {
        "match_all": {}
      },
      "random_score": {}
    }
  },
  "size": 0,
  "aggs": {
    "sample": {
      "sampler": {
        "shard_size": 50000
      },
      "aggs": {
        "keywords": {
          "significant_text": {
            "field": "headline",
            "size": 100000,
            "include": "[a-zA-Z]\\S*",
            "script_heuristic": {
              "script": {
                "lang": "painless",
                "source": "params._superset_freq * 1"
              }
            }
```

https://gist.github.com/markharwood/74bb6b8523f6a5746b1b758da2a5372e

Open source JS algorithms for use in browsers

Install

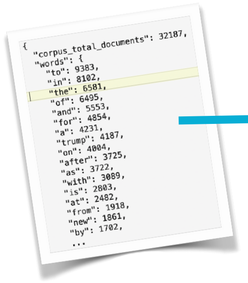
```
> npm i @andorsearch/significant-terms
```

Repository

github.com/markharwood/significant-terms

Significant terms detection example

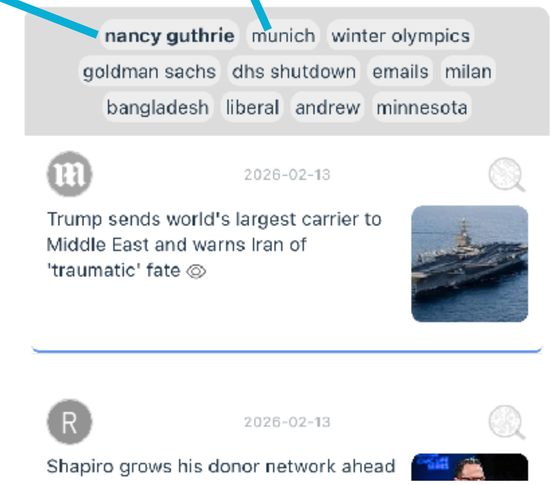
Detect single words *and* phrases e.g. people names




```
function findSignificantWordsOrPhrases(searchResultTexts: string[]) {
    // Tokenise the text found in search results
    let tokenStreams = searchResultTexts.map((textValue) => simpleTokenizer(textValue))

    //Find the significant words compared to the background
    const significantWords = computeSignificantTerms(
        tokenStreams,
        backgroundWordStats,
        backgroundCorpusSize
    );
    // Optionally, examine how the significant words are placed in the text to identify word p
    let significantWordsOrPhrases = detectAndSortSequences(significantWords, tokenStreams)
```

Trending in today's news:



Comparing approaches

Pros and cons

Trade-off	Javascript	Elasticsearch aggregation
Content scope	All raw text under consideration must be sent to the client	Aggregations can summarise content without sending raw original text to clients
Compute cost	Mostly offloaded to client	Can tie-up your cluster
Readability	Can discover readable terms and phrases	Can be unintelligible (e.g. stemmed words used in the search index)
Score correctness	Dictated by choice of background sample	Dictated by physical arrangement of data (number of indices/shards/shared tenants)
Dependencies	Works with any database backend	Works with elasticsearch and indexed fields

Demo