ElasticSearch

Prof. Luca Cagliero
Dipartimento di Automatica e Informatica
Politecnico di Torino

Lecture goal

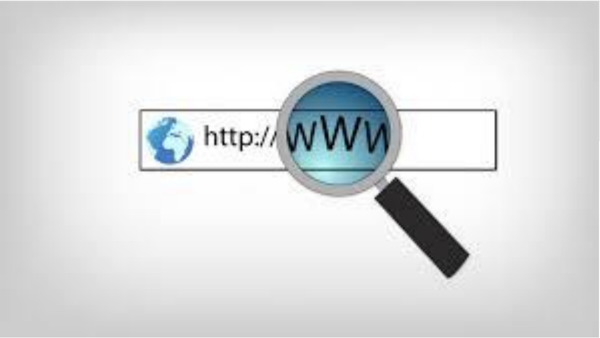
- Information Retrieval
- Web search
- ElasticSearch

Information Retrieval

Find material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.

What does “search” mean?



1. Take a query string
2. Match it against a document collection
   - Perform full-text search
   - handle synonyms
3. Calculate a set of relevant results
4. Score documents by relevance
5. Display a sorted list

Boolean retrieval

- Goal
  - Avoid scanning the text for each query
- Solution
  - Index the document in advance
- Method: Boolean retrieval model
  - pose any query in the form of a Boolean expression of terms
    - terms are combined with operators AND, OR and NOT
      - E.g., "Ceasar AND Brutus"
  - Returns all documents that satisfy the Boolean expression

Term-document incidence matrix

- Binary index representation
  - Avoid scanning the text for each query

Documents

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Term(s) vector(s)

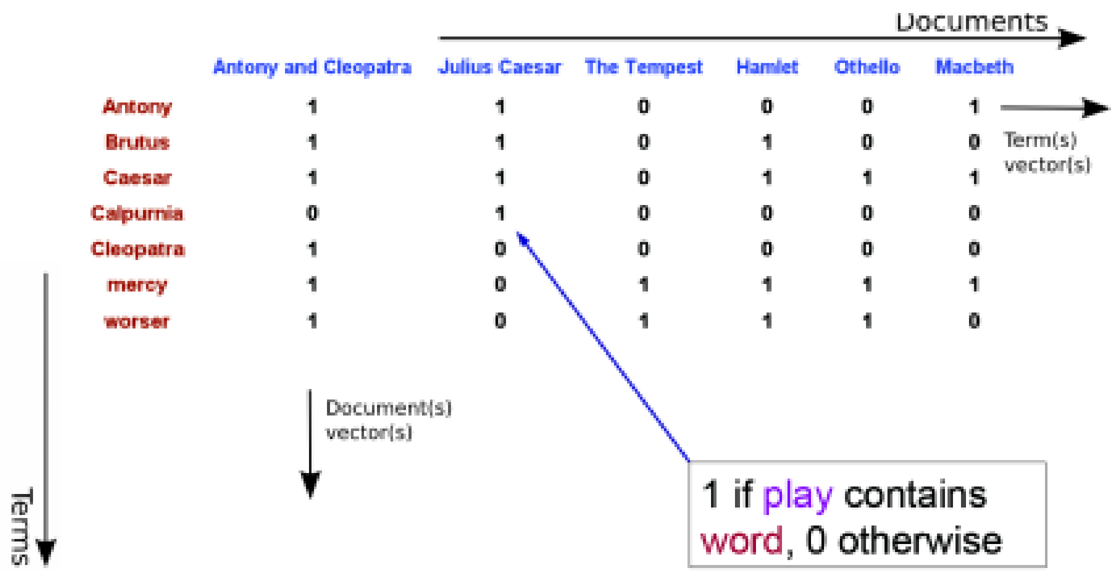
Document(s) vector(s)

Terms

1 if play contains word, 0 otherwise

Term-document incidence matrix

- To answer the query *Brutus AND Ceasar AND NOT Calpurnia*:
  - Take the vectors for *Brutus*, *Ceasar* and the complement of the vector for *Calpurnia:* 110100 AND 110111 AND 101111 = 100100
  - The answer to this query are "Anthony and Cleopatra" and "Hamlet"



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

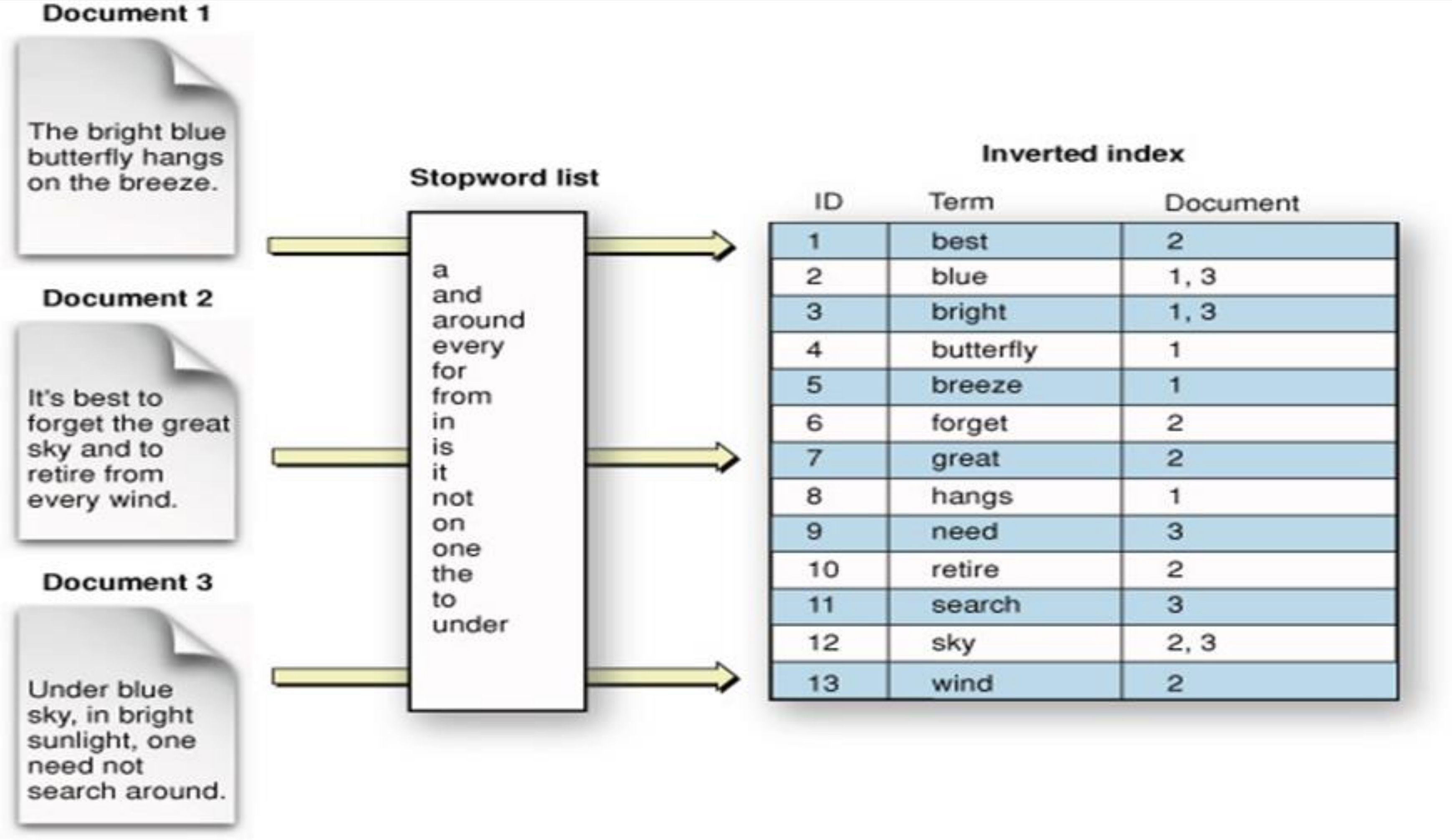
Boolean retrieval

- Drawbacks of the Boolean retrieval model
  - Sparse representation
    - Inefficient
  - All terms weigh equally
  - Capture syntactic text similarities only
- Solution to model sparseness
  - Inverted index
    - Stores only the 1s
- Solution to text similarities
  - Instead of 0s and 1s store the semantic similarities
    - e.g., BERT score

Boolean retrieval

- Solution to model sparseness
  - Inverted index
    - Stores only the 1s

Inverted index example



ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

https://stackoverflow.com/questions/47003336/elasticsearch-index-sharding-explanation (latest access: April 2021)

Boolean retrieval

- Solution to term weighting
  - Use tf-idf weights or similar
- Solution to syntactic-only text similarities
  - Use semantic similarity measure
    - e.g., BERT score

Web search: challenges

- Content publishing on the Web is decentralized
- No central control of authorship
- Web page authors created content in dozens of (natural) languages and thousands of dialects

Web search

- The Web can be modelled as a directed graph in which each web page is a node and each hyperlink a directed edge
- Web search is commonly addressed using graph ranking algorithms
  - E.g., PageRank, HITS

More about graph ranking in the in-class practice

Additional reading on PageRank

- The Anatomy of a Large-Scale Hypertextual Web Search Engine. Sergey Brin, Lawrence Page. Computer Networks, vol. 30 (1998), pp. 107-117
- Download and read the paper: https://snap.stanford.edu/class/cs224w-readings/Brin98Anatomy.pdf

Additional reading on HITS

- Kleinberg, Jon (1999). "Authoritative sources in a hyperlinked environment" (PDF). Journal of the ACM. 46 (5): 604–632. doi:10.1145/324133.324140.
- Download and read the paper: http://www.cs.cornell.edu/home/kleinber/auth.pdf

ElasticSearch

elastic

- Real-time distributed search and analytics engine
- Scalable and efficient data exploration
  - Full-text search
    - Highlighted search snippets
    - search as you type
    - did-you-mean
    - more-like-this
- Structured search
- Analytics
  - Real-time query answers on mixed data types
    - E.g., text, structured data

 |

Elastic Search: popular example

- **GitHub** uses ElasticSearch to query 130+B lines of code.
- **Wikipedia** provides full-text search with highlighted snippets
- **StackOverflow** combines both full-text and geolocation queries for recommending related questions and answers

ElasticSearch

elastic

- Document-oriented (JSON) search engine
    - Complex data structures that may contain dates, geo locations, text, other objects, arrays of values
- Built on Lucene search engine library
    - Documents are indexed and searchable
- Highly available and horizontally scalable

ElasticSearch: data representation

- Data is stored in *named entries* belonging to a variety of data types
- SQL calls such an entry an *attribute* whereas in ElasticSearch it is called *field*
- In ElasticSearch a field can contain multiple values of the same type (list of values)
  - Similar to other NoSQL databases

ElasticSearch: `field` **SQL:** `column`

ElasticSearch: data representation

- Data objects are represented as rows (SQL) or documents (ElasticSearch)
    - Row format in SQL is strict and follows a predefined schema
- Columns and fields are part of a row (SQL) or a document (ElasticSearch)
    - Documents are more flexible and can contain a variety of fields
    - They do not follow a strict schema

ElasticSearch: `document` **SQL:** `row`

ElasticSearch: data representation

- An index is like a table in a relational database
- In Elasticsearch the indices are grouped in a cluster
  - As tables are included into a database

ElasticSearch: `index` **SQL:** `table`

ElasticSearch: `cluster` **SQL:** `database`

ElasticSearch: data representation

ES	cluster	index	document	field
SQL	database	table	row	column

The index term

- The index term has multiple meanings
  - (noun) An index stores a collection of documents
  - (verb) To index a document means to insert a document in an index
    - If the document already exists, it is replaced

Inverted index

- Additional structure that accelerates data retrieval
- Similar to a traditional relational index
- Every field in a document is indexed in ES
  - All inverted indices are used during search
  - Non-indexed fields (if any) are not searchable

The document

- A document is the top-level (root) object serialized into JSON
- It is uniquely identified by the pair
  - Index: where the document (object) is stored
  - Id: the identifier of the document
    - Can be provided or uniquely generated by ElasticSearch

Search in Elastic

Three options:

- Structured query on specific fields
  - eventually sorted
  - similar to SQL query
- Full-text query
  - Finds all documents matching the search keywords
  - Returns them sorted by relevance
- A combination of the above

Key concepts

elastic

- Mapping
  - How the data in each field is interpreted
  - ElasticSearch dinamically generates a mapping by “guessing” data types
    - E.g., it may recognize a date type
- Analysis
  - How full text is processed to make it searchable
- Query DSL (Domain Specific Language)
  - Elastic Search query language

Search for exact values

- Traditional data types (e.g., integer, float, date, but also string)
  - A value must **match exactly** the query
    - Similar to SQL
  - Examples: date, user ID, but also exact strings such as username or email address
- Question answered
  - "Does this document match the query?"

Search for full-text

- Search for textual data
  - usually written in some human language
- Typical search is within the textual field
  - Examples: text of a tweet, body of an email
- It requires the definition of the concept of a document relevance to a query
- Question answered
  - How well does this document match the query?

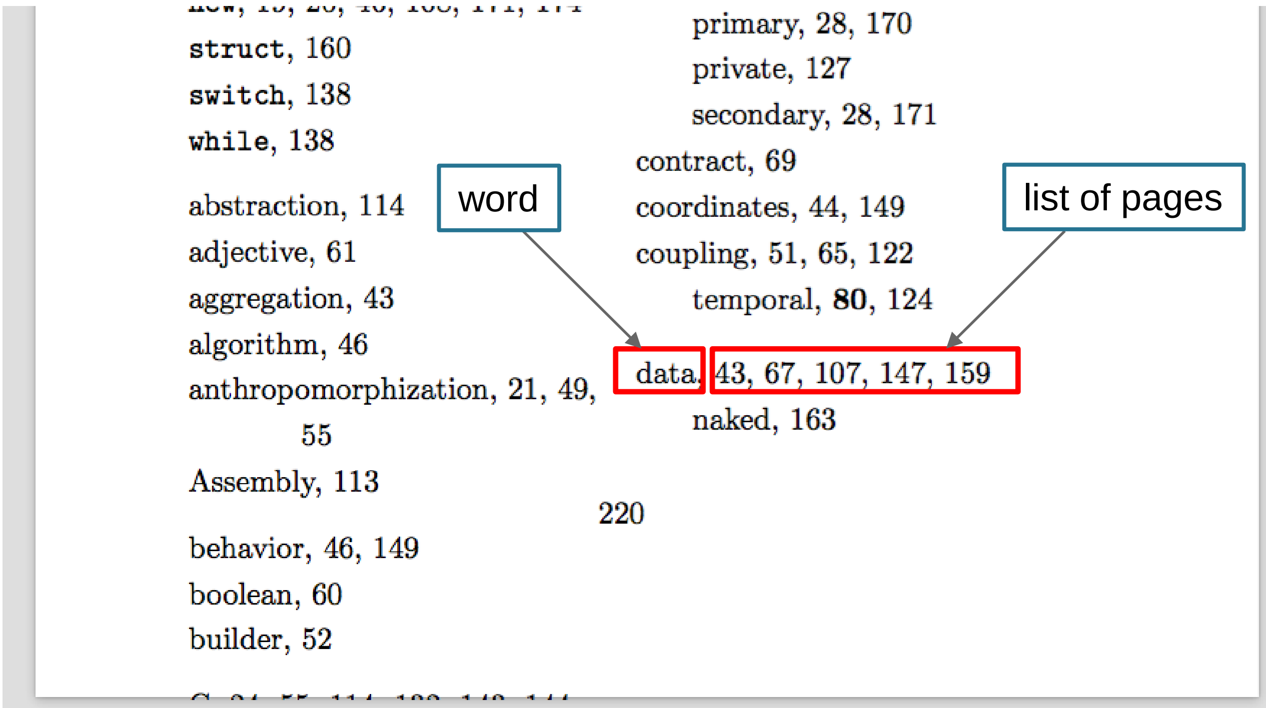
Search for full-text

- For solving full text queries it is very important to understand the underlying intent
  - abbreviations
    - e.g., USA vs United States of America
  - singulars/plurals, verb conjugation
    - e.g., cat vs cats, does vs did vs to do
  - synonyms
    - e.g., game vs competition
  - order of words building a context
    - e.g., fox news hunting vs fox hunting news

Full-text indexing

- ElasticSearch builds an inverted index on every full-text field
  - Designed for fast full-text search
- Inverted index
  - List of all the unique words that appear in any document in the collection
  - For each word it lists of the documents in which it appears

Full-text indexing



220

Analysis

Two main steps:

1. Tokenization of a block of text into individual terms suitable for an inverted index
2. Normalization into a standard form to improve their retrieval (or recall) in queries
   - Terms are not exactly the same, but similar enough to be still relevant
     - Lowercase vs. uppercase
     - Stemming
       - e.g., cats vs cat
     - Synonym management
   - For searching, both indexed text and query string must be analyzed in the same way

Analyzer

Built-in functions provided by ElasticSearch:

- Character filter
  - clean the string before tokenization
- Tokenizer
  - split the string into individual words
  - E.g., by considering white spaces or punctuation as separators
- Token filters: operate on single terms
  - change terms (e.g., to lowercase)
  - remove (e.g., stopwords)
  - add terms (e.g., synonyms)

Filter vs. query

- A filter is used for fields containing exact values
  - It provides a boolean matches/does not match answer for every document
- A query is (typically) used for full-text search
  - It also asks the question: How well does this document match?
    - calculates how relevant each document is to the query
    - assigns it a relevance score, which is later used to sort matching documents by relevance
- The concept of relevance is well suited to full-text search
  - there is seldom a completely "correct" answer

Filter vs. query

- Filter execution is more efficient
- Filters are typically used to reduce the number of documents that have to be examined by a query
- Hint
  - use query clauses for full-text search or for any condition that should affect the relevance score
  - use filter clauses for everything else

 |

ElasticSearch query

- Expressed in Query DSL
- Submitted as formatted JSON in the body of an HTTP request
  - Example: empty query
    - returns all documents in all indices

```
POST /_search
{}
```

- Search on a specific index

```
POST index1/_search
{}
```


Query DSL

- The top level field in an ElasticSearch query is always “query”
    - The query type is specified one level below
    - the query operates on the department index
        - specified in the URI
    - it performs the **search** operation

```
POST departments/_search
{
   “query”: {
       “match” : { “name” : ”John” }
   }
}
```


Query DSL

- Query
  - Find all the documents in the department index that have a field name containing the term John in it.
- Query type
  - Match query

```
POST departments/_search
{
   “query”: {
        “match” : { “name” : ”John” }
   }
}
```


Compound queries

elastic

- Complex queries specifying multiple matching criteria

```
POST departments/_search
{
    "query": {
        "bool": {
            "should": [
                {"match": {"name": "John"}},
                {"match": {"name": "Mark"}}
            ],
                "minimum_should_match":1,
            "must":{
                {"match": {"title": "developer"}}
            }
            "must_not":{
                {"match": {“lastname": “Smith"}}
            }
        }
    }
}
```

bool specifies the compound query

Compound queries

```
POST departments/_search
{
   "query": {
      "bool": {
         "should": [
            {"match": {"name": "John"}},
            {"match": {"name": "Mark"}}
         ],
            "minimum_should_match":1,
         "must":{
            {"match": {"title": "developer"}}
         }
         "must_not":{
            {"match": {“lastname": “Smith"}}
         }
      }
   }
}
```

`should` specifies the OR condition

`must` corresponds to the AND condition

`must_not` specifies the NOT condition

The match query

- Can be used for both full-text and exact queries
- On a full-text field
  - it analyzes the query string with the correct analyzer before executing the search
  - it returns a relevance score _score for the search
- On an exact field or a not analyzed string field
  - it searches the exact value
  - it returns a relevance score _score of 1
- When a bool query is specified on full-text fields
  - It combines the _score from each must or should clause that matches

Query DSL

- It is possible to specify multiple indices to be searched in the query URI

```
POST rooms,students/_search {...}
```

- When a number of documents can be returned as query result, by default the top 10 relevant results are returned
- Earlier versions of ElasticSearch include index types that have been deprecated since version 7.0

Data definition and updating

- Insert of a new single document is performed by means of a POST operation
  - Name of index
  - JSON document to be indexed

- **Index_name**: name of the index in which the document should be inserted
- <**id**>: optional parameter that associate the document with a specific identifier
  - If the ID is not provided, ElasticSearch creates a unique identifier for the document (e.g., W0tpsmIBdwcYyG50zbta)

Data definition

- Documents in ES are immutable
  - To update a document, it is reindexed
- When a document is updated, ES
  - Retrieves the old document
  - Modifies the retrieved copy
  - Deletes the old document
  - Indexes the new document (the copy)
- Internally, the old version of the document is not deleted immediately
  - It is not accessible
  - Deleted documents are cleaned in background

Data updating

- The update of a document is performed using a POST request
  - Name of the index
  - Unique ID of the document
  - the fields to be updated and the associated new values
    - To update a document, it is reindexed

```
PUT index_name/123/_update
{
  "color" : "red",
}
```

This update request modifies the document with ID=123 by setting the value of the "color" field to "red"

Data deletion

elastic

- The deletion of a document is performed using a DELETE request
  - Name of the index
  - Unique ID of the document

```
DELETE index_name/id
```

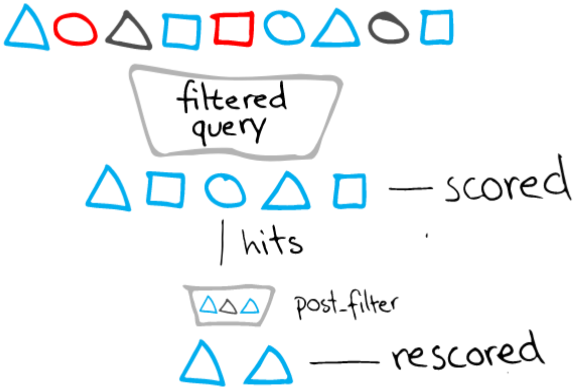
- The operation removes a JSON document from the specified index
  - Document removal is not immediate!

Relevance scoring

- In ElasticSearch the relevance score is a floating-point number
  - computed for each document matching the query
  - stored as _score for each document in the search result
  - higher _score values correspond to more relevant documents
- Sorting by relevance is performed by considering the _score variable
  - by default, documents in a query result are sorted by descending value of the _score field

Relevance scoring

1. Compute matching results for the query
    - Compute relevance score for all documents in the query result
2. Select top relevance documents (hits)
    - Default is 10 hits (documents)
3. (optional) Re-score documents
    - more computationally expensive algorithm



Relevance scoring

- Need to compute the similarity between
  - The query
  - Each document
- Each document may contain a (different) subset of the query terms

Relevance scoring

1. Select documents matching the query
   - Boolean model
   - Fast computation
2. Evaluate the importance (weight) of each term in a document with respect to the query
   - Term importance evaluated using the TF/IDF (Term Frequency/Inverse Document Frequency) score
   - Document and query are represented in vector form
     - Vector Space Model
3. Evaluate the similarity of the vector representation of the query and the document

BM25 score

- Key factors
  1. Term frequency
  2. Inverse document frequency
- They are calculated and stored at index time
- They are used to calculate the weight of a single term in a document
  - Other methods can be used
- Queries usually contain more than one term
  - Need a way to combine multiple terms

Vector Space Model

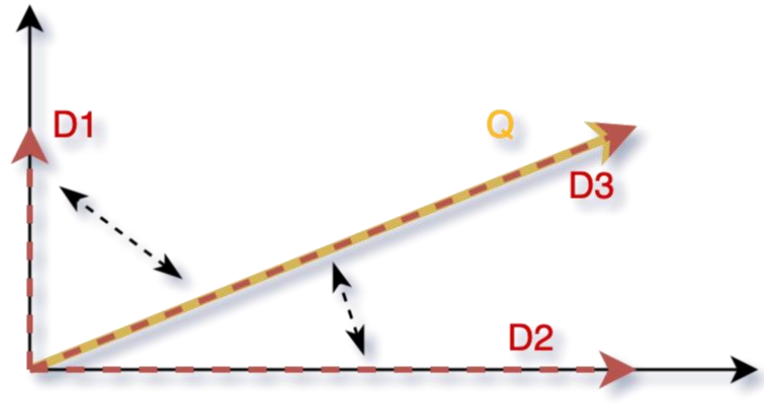
- It represents both query and document as (term) vectors
- It provides a way to compare a multi-term query against a document
- A query (or document) is represented as a vector
  - The vector size is the number of terms in the query
  - Each vector element is the weight of one term, calculated with BM25 scoring
    - Extensions towards semantic similarity (e.g., BERT similarity) are possible!
- Vectors can be compared by measuring the angle between them
  - Cosine similarity

Vector Space Model

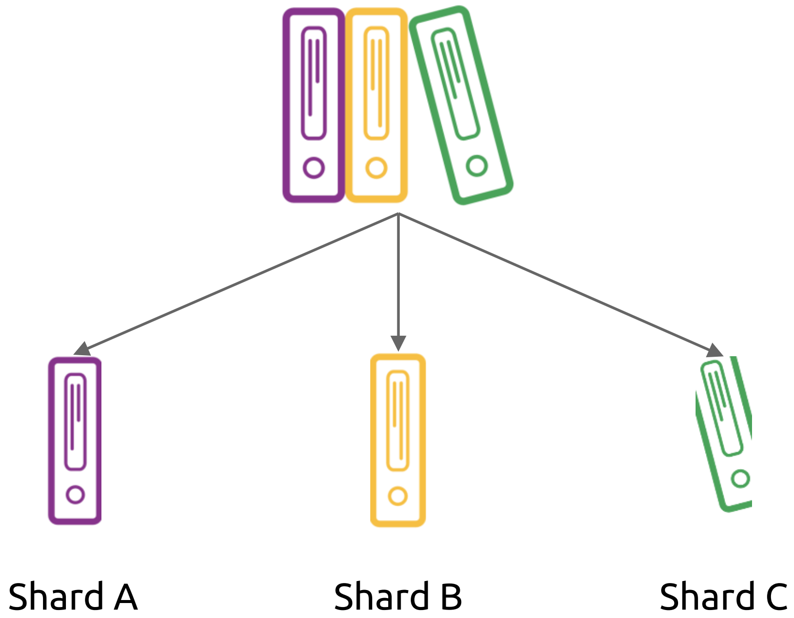
- Vectors can be compared by measuring the angle between them
  - Cosine similarity
- The angle between a document vector and a query vector is used to compute the similarity between a document and a query
  - It assigns to the document its relevance score for the query

Vector Space Model

- Query example
  - *Happy hippopotamus*
- Document examples
  - *I am happy in summer.*
  - *After Christmas I'm a hippopotamus.*
  - *The happy hippopotamus helped Harry.*
- It is possible to create a vector for each document
  - Document 1: (happy, ___________) -> [BM25_{happy},0]
  - Document 2: (___ , hippopotamus) -> [0, $\text{BM25}_{hippopotamus}$]
  - Document 3: (happy, hippopotamus) -> [BM25_{happy}, $\text{BM25}_{hippopotamus}$]



Horizontal scalability

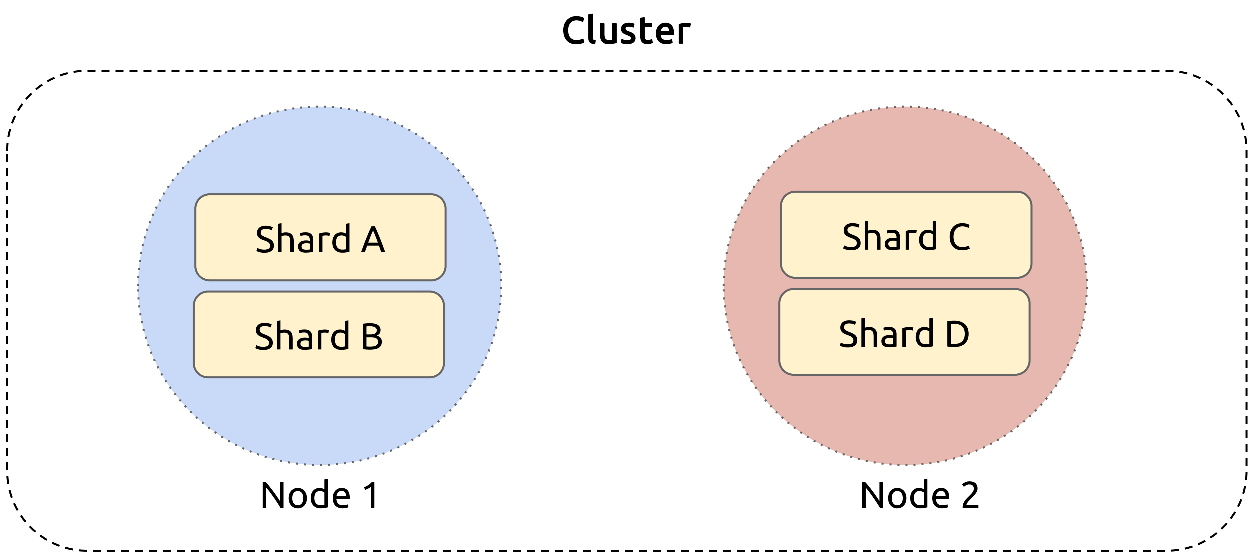


Sharding

- Sharding is a technique to divide an index in smaller partitions
  - Each partition is a shard
- Each document belongs to a single shard
  - Each shard is an instance of a Lucene index
- When data is written to a shard
  - It is periodically (every 1 second) written into a new immutable Lucene segment on disk
  - It becomes available for querying
- Shards are the elementary units in which data is distributed on nodes in a cluster

Clusters

- A cluster is a collection of multiple machines
  - i.e., nodes in the cluster
- Shards can be stored in any node within the cluster



Why sharding?

- It allows splitting data in smaller chunks, and thereby scaling on large volumes of data
  - Data may be distributed across multiple nodes within a cluster
  - Shards can be stored on smaller disks
    - E.g., it is possible to store 1TB of data even without a single node with that disk capacity
- Operations can be distributed across multiple nodes and thereby parallelized
  - Performance is increased, because multiple machines can potentially work on the same query
- Shards may be replicated on different nodes to increase availability

Optimistic concurrency control

- ElasticSearch uses optimistic concurrency control
  - It assumes that conflicts are unlikely to happen
  - However, if the underlying data has been modified between reading and writing, the update will fail
- Different from ACID transactions that need locking
- The process is “simple” for centralized data management

Modification propagation

- ElasticSearch data may be distributed on different nodes in a cluster
    - Shards may be replicated on different nodes (replica shards)
- When documents are created, updated, or deleted, the new version of the document has to be replicated to other nodes in the cluster
    - The primary copy is always written first
    - The replication requests are sent in parallel and may arrive at their destination out of sequence

Document versioning

- Elasticsearch needs a way of ensuring that an older version of a document never overwrites a newer version
    - Every document has a _version number that is incremented whenever a document is changed
- Elasticsearch uses this _version number to ensure that changes are applied in the correct order
    - if an older version of a document arrives after a new version, it can be ignored
    - the _version number is used to ensure that conflicting changes made by applications do not result in data loss
- APIs that update or delete a document accept a version parameter
    - can apply optimistic concurrency control only when needed

Acknowledgements and copyright license

- Acknowledgements
  - I would like to thank Dr. Moreno La Quatra, who collaborated to the writing and revision of the teaching content
- Affiliation
  - The author and his staff are currently members of the Database and Data Mining Group at Dipartimento di Automatica e Informatica (Politecnico di Torino) and of the SmartData interdepartmental centre
    - https://dbdmg.polito.it
    - https://smartdata.polito.it

Thank you!