TFIDF & ElasticSearch

DS8003 – MGT OF BIG DATA AND TOOLS

Ryerson University

Instructor: Kanchana Padmanabhan

ELASTICSEARCH

- ElasticSearch is a highly scalable open source search engine with a REST API
- Example:
  - Index all documents in Twitter
  - Find all documents that have a specific keyword
- TFIDF – scoring system that is inherent to the search engine. It retrieves the most relevant documents
- Scoring can be modified

http://www.rittmanmead.com/2015/08/three-easy-ways-to-stream-twitter-data-into-elasticsearch/

https://www.elastic.co/use-cases/klout

Install & Run ELASTICSEARCH (on sandbox)

- Install java8 - wget --no-check-certificate --no-cookies --header "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u102-b14/jdk-8u102-linux-x64.tar.gz
- tar-xvzf jdk-8u102-linux-x64.tar.gz
- export JAVA_HOME=jdk1.8.0_102/
- export PATH=$PATH:$JAVA_HOME/bin
- wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.0.1.tar.gz
- tar –xvzf elasticsearch-5.0.1.tar.gz
- cd elasticsearch-5.0.1 && bin/elasticsearch -d
- curl -XGET "http://127.0.0.1:9200"

https://blog.codecentric.de/en/2014/02/elasticsearch-101/

https://www.elastic.co/downloads/elasticsearch

Indexing Documents

- There is a python API also available. For simplicity we will use CURL
- PUT Request

```
curl -XPUT "http://localhost:9200/index/type/id" -d'
{
document
}'
```

```
curl -XPUT "http://localhost:9200/movies/movie/1" -d'
{
    "title": "The Godfather",
    "director": "Francis Ford Coppola",
    "year": 1972
}'
```

https://elasticsearch-py.readthedocs.io/en/master/

https://www.elastic.co/guide/en/elasticsearch/reference/2.3/docs-index_.html

Indexing Documents

- Index and type are required while the ID part is optional.
- If we don't specify an ID ElasticSearch will generate one for us.
- However, if we don't specify an id we should use POST instead of PUT.
- **The index name and type** are arbitrary.
- If there isn't an index with that name on the server already one will be created using default configuration.

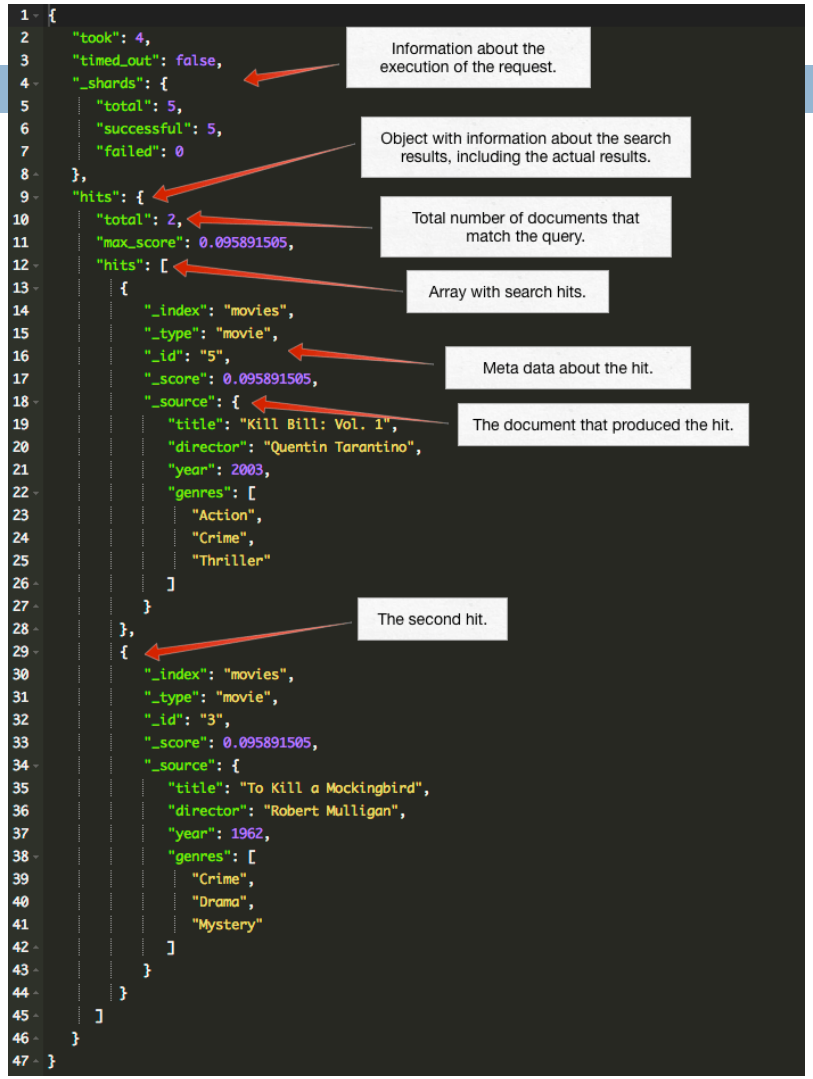
https://www.elastic.co/blog/index-vs-type

Search Documents

- Make POST requests to either of the following URLs:
- http://localhost:9200/_search - Search across all indexes and all types.
- http://localhost:9200/movies/_search - Search across all types in the movies index.
- http://localhost:9200/movies/movie/_search - Search explicitly for documents of type movie within the movies index.

```
curl -XPOST "http://localhost:9200/_search" -d'
{
    "query": {
        "query_string": {
            "query": "kill"
        }
    }
}'
```

Search Result




```
{
  "took": 4,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 0.095891505,
    "hits": [
      {
        "_index": "movies",
        "_type": "movie",
        "_id": "5",
        "_score": 0.095891505,
        "_source": {
          "title": "Kill Bill: Vol. 1",
          "director": "Quentin Tarantino",
          "year": 2003,
          "genres": [
            "Action",
            "Crime",
            "Thriller"
          ]
        }
      },
      {
        "_index": "movies",
        "_type": "movie",
        "_id": "3",
        "_score": 0.095891505,
        "_source": {
          "title": "To Kill a Mockingbird",
          "director": "Robert Mulligan",
          "year": 1962,
          "genres": [
            "Crime",
            "Drama",
            "Mystery"
          ]
        }
      }
    ]
  }
}
```

http://joelabrahamsson.com/elasticsearch-101/

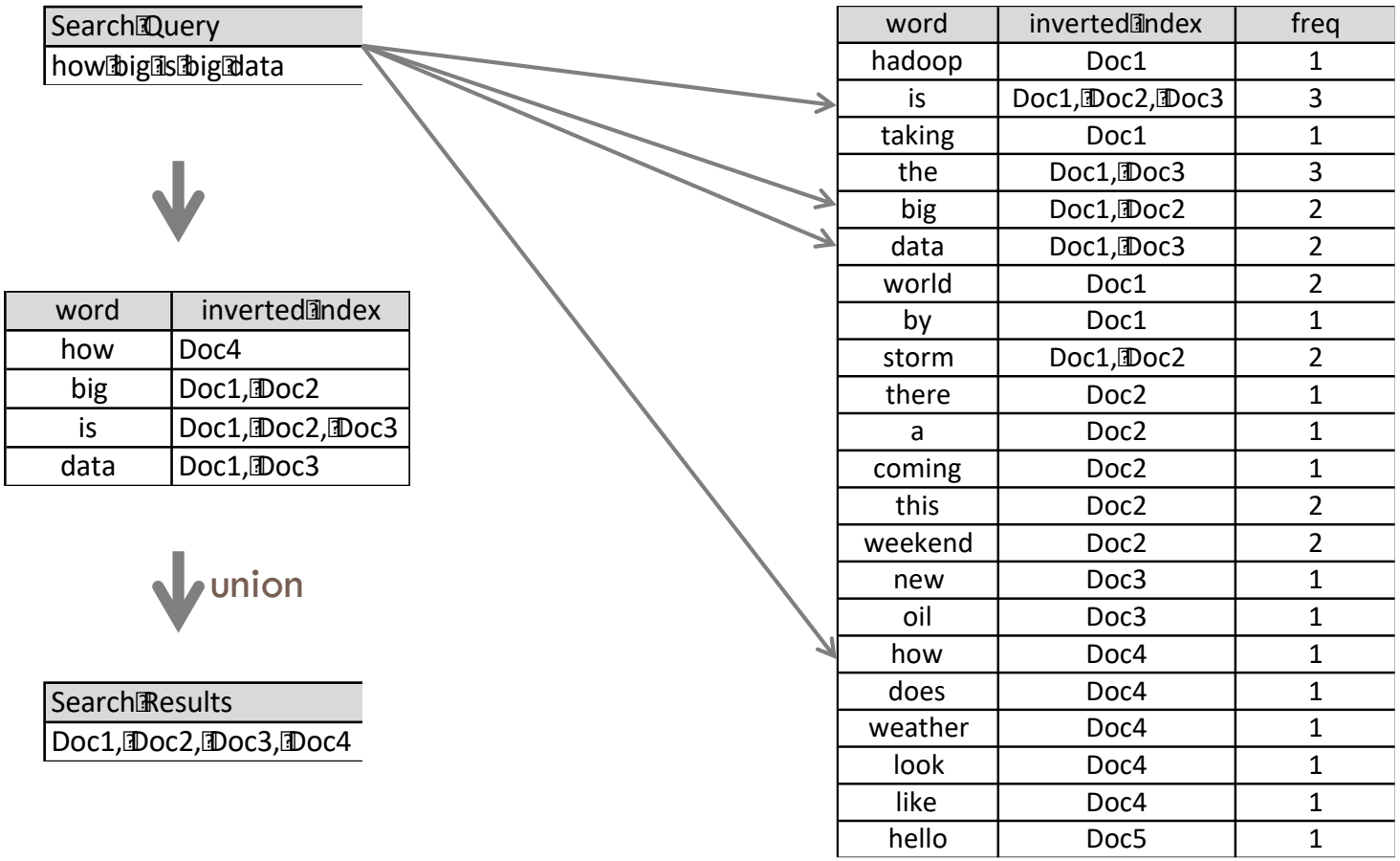
Search Engine In a Nutshell

- Inverted Index
  - Search engines like Google also use an inverted index for searching the web.
  - In fact, the need to build a web-scale inverted index led to the invention of MapReduce
- A list of documents that the term appear in

Documents	Text
Doc1	hadoop is taking the big data world by storm
Doc2	there is a big storm coming this weekend
Doc3	data is the new oil
Doc4	how does the weather look like this weekend
Doc5	hello world

word	inverted index	freq
hadoop	Doc1	1
is	Doc1, Doc2, Doc3	3
taking	Doc1	1
the	Doc1, Doc3	3
big	Doc1, Doc2	2
data	Doc1, Doc3	2
world	Doc1	2
by	Doc1	1
storm	Doc1, Doc2	2
there	Doc2	1
a	Doc2	1
coming	Doc2	1
this	Doc2	2
weekend	Doc2	2
new	Doc3	1
oil	Doc3	1
how	Doc4	1
does	Doc4	1
weather	Doc4	1
look	Doc4	1
like	Doc4	1
hello	Doc5	1

Inverted Index Document Retrieval



Retrieval does not consider the importance of word to document

How to consider importance of word to document?

- Counts of word in documents often don't capture what the document is about
- Doc 1: "*This is a document about a new area called 'big data.' This is a new and emerging area*"
- Most important words are "This", "is", "new" etc
- The word "big data" will often come towards the bottom

How to consider importance of word to document?

- You need to take into account two things
  - Frequency of word in document
  - Frequency of the word across all documents
- Importance should be given to words that frequently appear within a document and less frequently across documents

Use TFIDF instead of Counts

- Term Frequency – Inverse Document Frequency
  - puts more weight on relevant keywords
  - Considers the frequency of word in a document
  - Frequency of word across documents

- Calculate TFIDF value per term per document
- Given a word t in document d
- **tf(t, d)** = (Number of times term t appears in a document)
- **idf(t)** = **log**(Total number of documents / (Number of documents with term t in it))

$$\text{tf-idf}(t) = \text{tf}(t, d) \times \text{idf}(t)$$

Several variations in computing TF and IDF: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

TF-IDF Example:

1. Consider a document **d** containing 100 words wherein the word cat appears 3 times.
2. The term frequency (i.e., **tf(cat)**) for **cat** is then (3 / 100) = **0.03**.
3. Now, assume we have 10 million documents and the word cat appears in one thousand of these.
4. Then, the inverse document frequency (i.e., **idf(cat)**) is calculated as log(10,000,000 / (1,000)) = **4**.
5. Thus, the tf-idf weight is the product of these quantities:
6. **tf-idf(cat, d) = 0.03 * 4** = 0.12
7. This has to be computed for every (or selected) words in every document

TF-IDF - *Relevance Scoring*

TF_t ➔ Frequency of *t* in a document

$IDF_t = \log (N/DF_t)$

TF-IDF for the word 'hadoop' in Doc1:

$TF_{how} = 1$

$N = 5$

$TF_{how} = 1$

$IDF_{how} = log(5/1) = 0.699$

$TF\text{-}IDF_{how} = TF_{how} \times IDF_{how} = 0.699$

Doc1	hits	TF	IDF	TF-IDF
hadoop	1	1	0.699	0.699
is	3	1	0.222	0.222
taking	1	1	0.699	0.699
the	3	1	0.222	0.222
big	2	1	0.398	0.398
data	2	1	0.398	0.398
world	2	1	0.398	0.398
by	1	1	0.699	0.699
storm	2	1	0.398	0.398

Doc2	hits	TF	IDF	TF-IDF
there	1	1	0.699	0.699
is	3	1	0.222	0.222
a	1	1	0.699	0.699
big	2	1	0.398	0.398
storm	2	1	0.398	0.398
coming	1	1	0.699	0.699
this	2	1	0.398	0.398
weekend	2	1	0.398	0.398

Doc3	hits	TF	IDF	TF-IDF
data	2	1	0.398	0.398
is	3	1	0.222	0.222
the	3	1	0.222	0.222
new	1	1	0.699	0.699
oil	1	1	0.699	0.699

Doc4	hits	TF	IDF	TF-IDF
how	1	1	0.699	0.699
does	1	2	0.699	1.398
the	3	3	0.222	0.666
weather	1	4	0.699	2.796
look	1	5	0.699	3.495
like	1	6	0.699	4.194
this	2	7	0.398	2.786
weekend	2	8	0.398	3.184

Doc5	hits	TF	IDF	TF-IDF
hello	1	1	0.699	0.699
world	2	2	0.398	0.796

How to use TFIDF to find Relevant Search Results?

Query (Q): the big data

$$\text{Score(Query, Doc)} = \frac{|q \cap Doc|}{|q|} \sum_{q \in Q} TFIDF(q, Doc)$$

Score(Query, Doc1) = (0.398 + 0.398 + 0.222) * 3/3

Score(Query, Doc3) = (0.398 + 0.222) * 2/3

Score(Query, Doc2) = (0.398) * 1/3

Score(Query, Doc4) = (0.222) * 1/3

Doc1	hits	TF	IDF	TF-IDF
hadoop	1	1	0.699	0.699
is	3	1	0.222	0.222
taking	1	1	0.699	0.699
the	3	1	0.222	0.222
big	2	1	0.398	0.398
data	2	1	0.398	0.398
world	2	1	0.398	0.398
by	1	1	0.699	0.699
storm	2	1	0.398	0.398

Doc3	hits	TF	IDF	TF-ID
data	2	1	0.398	0.398
is	3	1	0.222	0.222
the	3	1	0.222	0.222
new	1	1	0.699	0.699
oil	1	1	0.699	0.699

Doc2	hits	TF	IDF	TF-IDF
there	1	1	0.699	0.699
is	3	1	0.222	0.222
a	1	1	0.699	0.699
big	2	1	0.398	0.398
storm	2	1	0.398	0.398
coming	1	1	0.699	0.699
this	2	1	0.398	0.398
weekend	2	1	0.398	0.398

Doc4	hits	TF	IDF	TF-IDF
how	1	1	0.699	0.699
does	1	2	0.699	1.398
the	3	3	0.222	0.666
weather	1	4	0.699	2.796
look	1	5	0.699	3.495
like	1	6	0.699	4.194
this	2	7	0.398	2.786
weekend	2	8	0.398	3.184

Doc5	hits	TF	IDF	TF-IDF
hello	1	1	0.699	0.699
world	2	2	0.398	0.796

There are variations to combining the score.

TF-IDF

- TF-IDF
  - http://horicky.blogspot.ca/2009/01/solving-tf-idf-using-map-reduce.html
- Solr vs. Elasticsearch
  - http://solr-vs-elasticsearch.com/