Enterprise Search

Lama Saeeda

lama.saeeda@fel.cvut.cz

January 2, 2019

Overview

Motivation

Motivation

Where is the search box?

Information systems need **Search** feature

Definition

Definition

Enterprise search

- is the practice of identifying and enabling specific content across the enterprise to be **indexed**, **searched**, and **displayed** to **authorized users**.
- is the organized retrieval of **structured** and **unstructured** data within your application.

Differences from normal web search

Enterprise vs. web search (Intranet vs. Internet)

- **Multiple data sources**
  websites, files, email, etc.
- **Collecting and indexing data**
  missed a key page?
- **Relevance and ranking algorithms**
  popular hits and page rank
- **Users**
  - Searchers are Knowledge workers
  - Context available: department, job, location...
- **Security**
  authenticated users
- **Single site, Single best document**
  federated search

Enterprise Search Components

Enterprise Search Components

- Content awareness and collecting data
- Content processing and analysis
- Indexing
- Query processing
- Matching

Collecting data

- Finding content and pulling it into the system
- **Crawlers** retrieve documents and other content
  - over protocols like HTTP
  - use adapters to connect to relational databases, document management systems, etc..

Content processing

- **Identification** sentences, determined by periods or other punctuation marks

  The operator operates successfully!

- **Tokenization** breaking up text into tokens (words, phrases, symbols, etc..)

  [The] [operator] [operates] [successfully]

- **Normalization** tokens to lower case to provide case-insensitive search

  [the] [operator] [operates] [successfully]

Content processing II

- **Stop-words removing** meaningless tokens, (there, so, other, etc..)
  - *[operator] [operates] [successfully]*
- **Stemming and lemmatization** to get the normal form of the word
  - *[operate] [operate] [success]*

.
.
.

- **Synonym expansion**: Controlled vocabulary, manually or automatically derived thesaurus, etc.. **Wordnet**
- **POS tagging** : the **book** on the table (noun), to **book** a flight (verb)

Indexing

- The resulting terms are stored in an index, instead of storing the full text of the document
- Contains the dictionary of all unique words in the corpus
- Groups information into logical categories that in turn can be searched and return results to users
- TF-IDF

Indexing - TF-IDF

- **TF: Term Frequency**, how frequently a term occurs in **one document**.
  TF = (Number of times term t appears in a document / Total number of terms in the document)
- **IDF: Inverse Document Frequency**, how important a term is in the **corpus** IDF = log (Total number of documents / Number of documents with term t in it)

Indexing - TF-IDF

$$TF * IDF(w) = TF(w).\frac{1}{DF(w)}$$

The word is more popular when it appears several times in a document

The word is more important if it appears in less documents

- $TF(w) \rightarrow$ term frequency (number of times a term occurs in a single document)
- $DF(w) \rightarrow$ document frequency (number of documents a term occurs in within the corpus)
- $TF * IDF \rightarrow$ relative importance of the word in the document

Indexing - TF-IDF

the following example is the example about indexing

Query: *the example*

$$TF_{the} = 2$$
$$TF_{example} = 2$$

$$IDF_{the} = 0$$
$$IDF_{example} = \frac{1}{7}$$

The total score of this doc against the query is:

$$score = TF_{the} \times IDF_{the} + TF_{example} \times IDF_{example}$$
$$= 2 \times 0 + 2 \times \frac{1}{7} = \frac{2}{7} = 0.2857142857$$

Searching

Enterprise search applications may allow

- general free-form keyword searching
- specialized query syntax to allow more specific queries
- a standardized query language like SQL or SPARQL

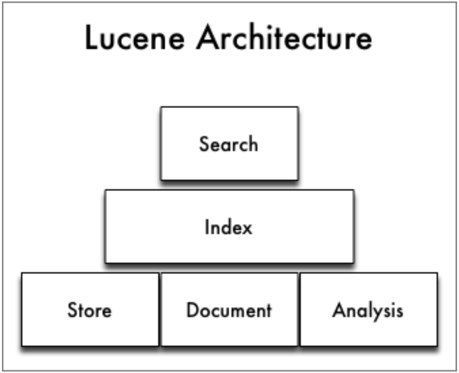
The query parser converts the query into a representation which can be used, along with the index, to determine matching results.

Query expansion for better performance (recall and precision)

Examples of enterprise search platforms and libraries

Lucene

- Java powerful open-source full-text search library
- Makes it easy to add full-text search capability to your application.
- **not** a complete application **but** a code library and API



Lucene - Simple Indexing example

- in-memory index from some strings.

Indexing

```
StandardAnalyzer analyzer = new StandardAnalyzer();
Directory index = new RAMDirectory();

IndexWriterConfig config = new IndexWriterConfig(analyzer);

IndexWriter w = new IndexWriter(index, config);
addDoc(w, "Lucene in Action", "193398817");
addDoc(w, "Lucene for Dummies", "55320055Z");
addDoc(w, "Managing Gigabytes", "55063554A");
addDoc(w, "The Art of Computer Science", "9900333X");
w.close();
```

Lucene - Simple Indexing example II

- **addDoc()** is what actually adds documents to the index
- use of **TextField** for content we want tokenized, and **StringFiel** for id fields and the like, which we don't want tokenized.

indexing - addDoc()

```
private static void addDoc(IndexWriter w, String title, String isbn) throws
IOException {
Document doc = new Document();
doc.add(new TextField("title", title, Field.Store.YES));
doc.add(new StringField("isbn", isbn, Field.Store.YES));
w.addDocument(doc);}
```

Lucene - Simple query example

- We read the query from stdin, parse it and build a lucene Query out of it.

query

```
String querystr = "your query keywords";
Query q = new QueryParser("title", analyzer).parse(querystr);
```

Lucene - Simple search example

- Using the Query we create a Searcher to search the index. Then a TopScoreDocCollector is instantiated to collect the top 10 scoring hits

search

```
int hitsPerPage = 10;
IndexReader reader = DirectoryReader.open(index);
IndexSearcher searcher = new IndexSearcher(reader);
TopDocs docs = searcher.search(q, hitsPerPage);
ScoreDoc[] hits = docs.scoreDocs;
```

Elasticsearch

- Open source search server powered by Lucene under the hood
- Written in Java
- Cross platform
- Scalability and distributed architecture
- HTTP REST API
- Schema-less JSON documents
- Developed by *Elasticsearch BV*
- Near real-time search

Elasticsearch

- Wikimedia
- Quora
- SoundCloud
- Github
- Netflix
- Foursquare

.

.

.

Elasticsearch - Introduction Example

- **Download** the latest distribution from `https://www.elastic.co/downloads/elasticsearch`
- **Unpack** it on your machine
- **Run** it, by launching **elasticsearch**
- **Lunch** it from the web browser `http://localhost:9200`

Elasticsearch - Introduction Example

Result in the browser


```
{
  "status" : 200,
  "name" : "Big Man",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "1.7.2",
    "build_hash" : "e43676b1385b8125d647f593f7202acbd816e8ec",
    "build_timestamp" : "2015-09-14T09:49:53Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.4"
  },
  "tagline" : "You Know, for Search"
}
```

Elasticsearch - Building a basic search app

Create an Index

```
PUT /myapp?pretty
```

Index a Document

```
PUT /myapp/tweet/1?pretty
{
  "name": "John Doe",
  "tweet": "I think elasticsearch is AWESOME",
  "date" : "2013-06-03",
  "loc" : {
   "lat": 13.4,
       "lon": 52.5
  }
}
```

Create an Index - Response


```
{
  "_index" : "myapp",
  "_type" : "tweet",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Get the Document

```
GET /myapp/tweet/1?pretty
```

Get the Document - Response

```
{
  "_index" : "myapp",
  "_type" : "tweet",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : { ...OUR TWEET... }
}
```

Update the Document

```
PUT /myapp/tweet/1?pretty
{
  "name": "Jahn Doe",
  "tweet": "I think elasticsearch is AWESOME",
  "date" : "2013-06-03",
  "loc" : {
   "lat": 13.4,
       "lon": 52.5
  }
}
```

Delete the Document - Response

```
Delete /myapp/tweet/1?pretty
```

PS: Response: "version" : 3

Indexing

$Doc\#1 \rightarrow$ _ *[operate] [operate] [success]*

'operate' : [1, 47, 72],
'success' : [1, 55, 92, 107],

Inverted Index

$$Doc\#1 \rightarrow \_ \textit{[operate] [operate] [success]}$$

```
inverted_index = {
'operate' : [1, 47, 72],
'success' : [1, 55, 92, 107],
'search' : [34, 92,, 119],
' zebra' : [15, 34, 55, 107],
}
```

Mapping

```
{
 "tweet": {
   "properties": {
     "name": { "type": "string" },
     "tweet": { "type": "string" },
     "date": { "type": "date" },
     "loc": { "type": "geo_point" },
}}}
```

PS: Do not change the mapping of existing field

Mapping

Full text : (defult)

```
{ "type": "string", index: "analyzed" }
```

Exact string

```
{ "type": "string", index: "not_analyzed" }
```

Not searchable

```
{ "type": "string", index: "no" }
```

Search the index - Empty Search

```
GET /myapp/_search
{
 "query": { "match_all": {} }
}
```

Response

```
{
 "took" : 2,
 "timed_out" : false,
 "_shards" : {
  "total" : 5,
  "successful" : 5,
  "failed" : 0
 },
 "hits" : {
 "total" : 14,
  "max_score" : 1.0,
  "hits" : [ {...}]
  }
}
```

Filters vs. Queries

Filters

- Exact matching
- binary yes/no
- fast
- cacheable

Queries

- full text search
- relevance scoring
- heavier
- not casheable

Query:

```
{ "match": {"tweet": "search" }}
```

Filter:

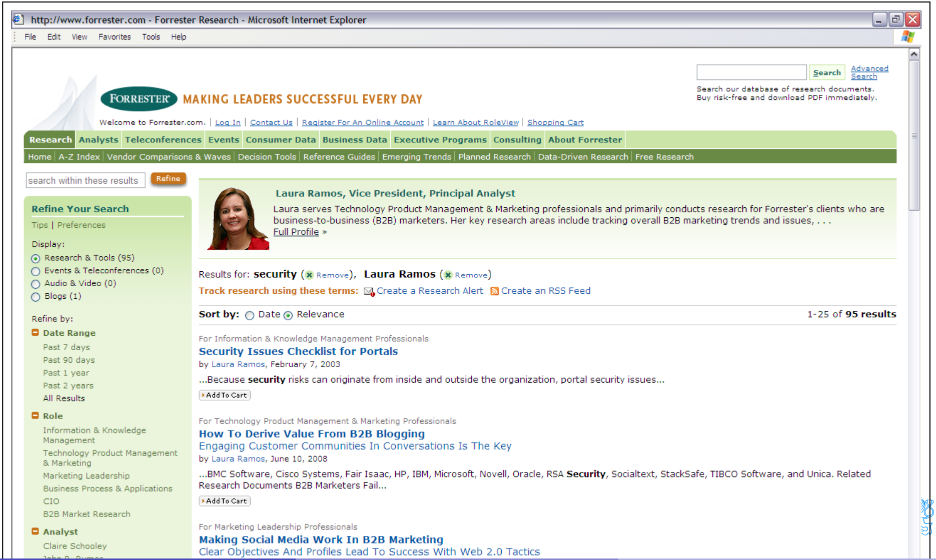
```
{ "term": {"date": "2018-1-3" }}
```

Filtered queries & Boolean queries

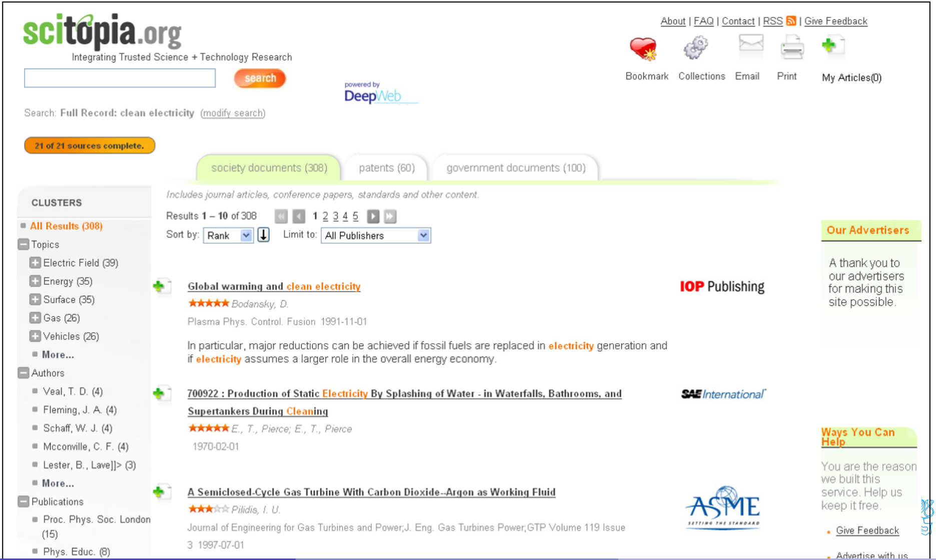
```
GET /bank/_search
{
  "query": {
    "bool": {
      "must": { "term": {"category" : "tech"} },
      "filter": {
        "range": {
          "salary": {
            "from": 20000,
            "to": 30000
          }
        }
      }
    }
  }
}
```

$$Boolean \rightarrow must, should, must\_not$$

Aggregations



Aggregations



Aggregations

```
GET /bank/_search
{
 "size": 0,
 "aggs": {
  "group_by_state": {
   "terms": {
    "field": "state"
   }
  }
 }
}
```

Aggregations

```
GET /bank/_search

{
  "size": 0,
  "aggs": {
    "group_by_age": {
      "range": {
        "field": "age",
        "ranges": [
          {
            "from": 20,
            "to": 30
          },
          {
            "from": 30,
            "to": 40
          },
          {
            "from": 40,
            "to": 50
          }
        ] },
```

```
"aggs": {
        "group_by_gender": {
          "terms": {
            "field": "gender.
                keyword"
          },
          "aggs": {
            "average_balance": {
              "avg": {
                "field": "balance"
              }
            }
          }
        }
      }
    }
  }
}
```

Solr

- Also built on Lucene
  - So similar feature set
  - Also exposes Lucene functionality, like Elastic Search, so easy to extend.
- A part of Apache Lucene project
- Perfect for Single Server search
- Clustering is there. But it's definitely not as simple as ElasticSearch
- Solr is for text search while Elasticsearch is for filtering and grouping, the analytical query workload, and not just text search.

Evaluation of search system

Evaluation of search system

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

<table>
<tr><td></td><td colspan="3">Documents Retrieved (search results)</td></tr>
<tr><td rowspan="3">Actual Documents (Should be retrieved)</td><td></td><td>Class = Yes</td><td>Class = No</td></tr>
<tr><td>Class = Yes</td><td>True Positive</td><td>False Negative</td></tr>
<tr><td>Class = No</td><td>False Positive</td><td>True Negative</td></tr>
</table>

What is bad search?

- No search box
- Too many hits: Return 10,000 hits when the average user looks at the top-20 only
- Bad scoring: The most relevant item is not at the top of the list
- Poor duplicate detection: Too many similar documents
- Inability to judge user intent: spell checking,recommendation system, auto complete.

The End

Thank You