Search Quality Evaluation

Tools and Techniques

Alessandro Benedetti, Software Engineer

Andrea Gazzarini, Software Engineer

2nd October 2018

Who we are

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

Alessandro Benedetti

- Search Consultant
- R&D Software Engineer
- Master in Computer Science
- Apache Lucene/Solr Enthusiast
- Semantic, NLP, Machine Learning Technologies passionate
- Beach Volleyball Player & Snowboarder

SeaSe

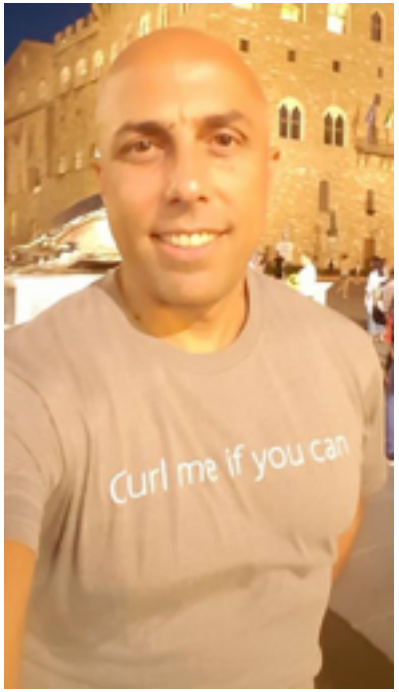
Who we are

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

Andrea Gazzarini, “Gazza”

- Software Engineer (1999-)
- “Hermit” Software Engineer (2010-)
- Java & Information Retrieval Passionate
- Apache Qpid (past) Committer
- Husband & Father
- Bass Player

Qpid



SeaSe

Search Services

- Open Source Enthusiasts
- Apache Lucene/Solr experts
- Community Contributors
- Active Researchers
- Hot Trends : Learning To Rank, Document Similarity, Measuring Search Quality, Relevancy Tuning

ADOREBEAUTY

alamy

BARCLAYS

DeviceBits

ComplyAdvantage

NetDimensions

Information Builders

Active Network
Ready to business

BAR FRIDGES
Cooling the Beers of a Nation
Australia

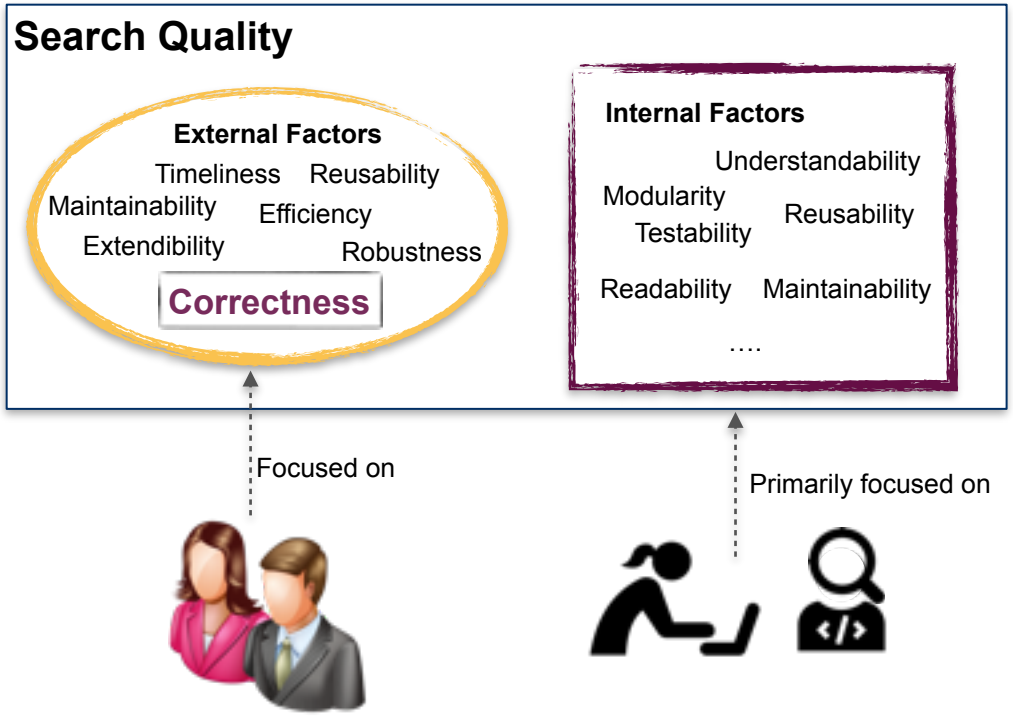
Agenda

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

- ✓ **Search Quality Evaluation**
  - **Context overview**
  - **Correctness**
  - **Evaluation Measures**
- ➢ Rated Ranking Evaluator (RRE)
- ➢ Future Works
- ➢ Q&A

SeaSe

Search Quality Evaluation



Context Overview

Search engineering is the **production** of **quality search systems**.

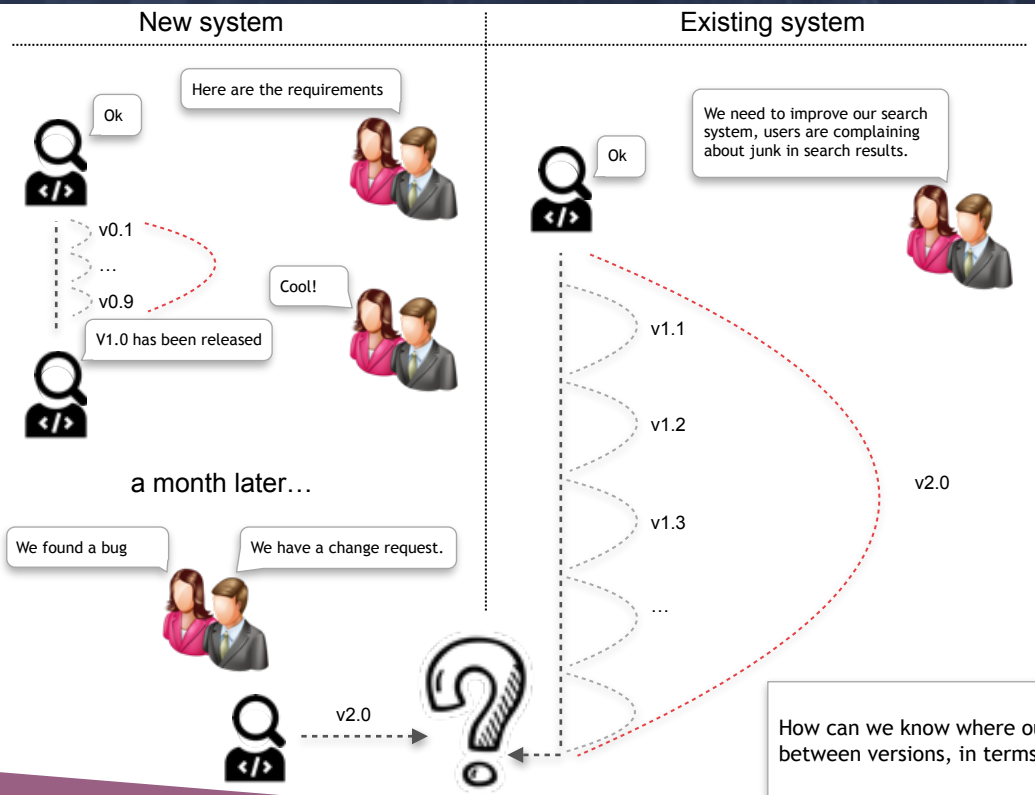
Search quality (and in general software quality) is a huge topic which can be described using **internal** and **external factors**.

In the end, **only external factors matter**, those that can be perceived by **users** and **customers**. But the key for getting **optimal levels** of those external factors are the **internal ones**.

One of the main differences between search and software quality (especially from a **correctness** perspective) is in the **ok / ko judgment**, which is, in general, more “deterministic” in case of software development.

SeaSe

Search Quality Evaluation: Correctness



Correctness

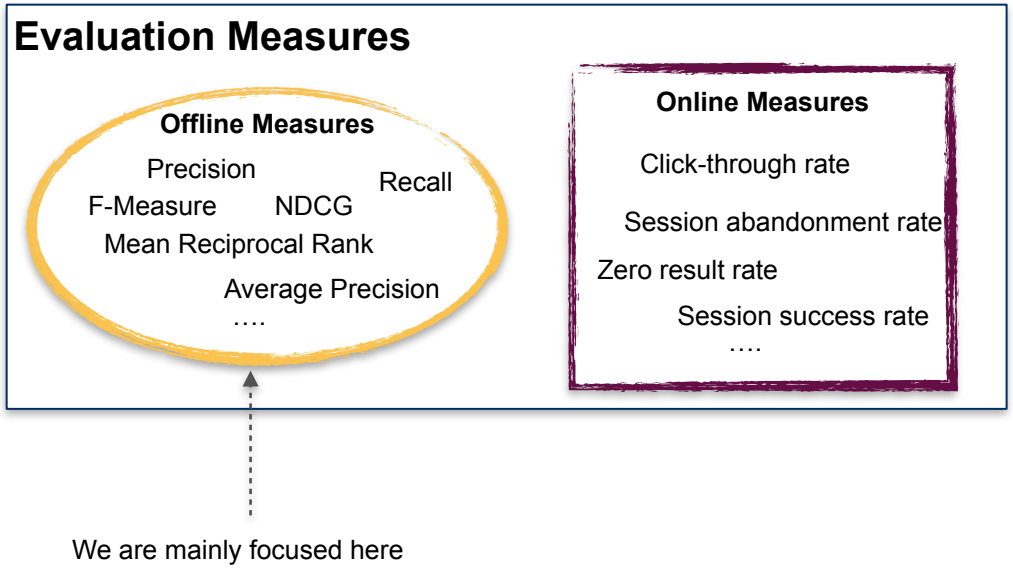
Correctness is the ability of a system to perform its exact task, as defined by its specification.

Search domain is critical from this perspective because correctness depends on arbitrary user judgments.

For each internal (gray) and external (red) iteration we need to find **a way to measure the correctness**.

Evaluation measures for an information retrieval system are used to assert how well the search results satisfied the user's query intent.

Search Quality Evaluation / Measures



Evaluation Measures

Evaluation measures for an information retrieval system try to formalise how well a search system satisfies its user information needs.

Measures are generally split into two categories: **online** and **offline measures**.

In this context we will focus on **offline measures.**

We will talk about something that can **help a search engineer** during his ordinary day (i.e. in those phases previously called “**internal iterations**”)

We will also see how the **same tool** can be used for a broader usage, like contributing in the **continuous integration pipeline** or even for **delivering value** to **functional stakeholders**.

SeaSe

Agenda

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

- Search Quality Evaluation

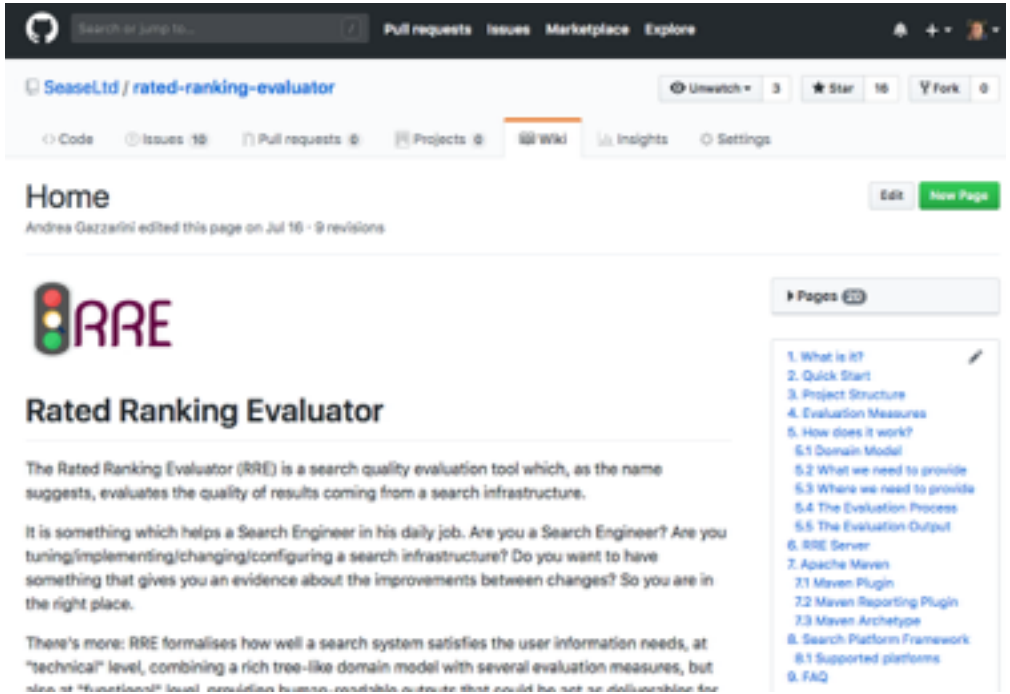
- ✓Rated Ranking Evaluator (RRE)
  - What is it?
  - How does it work?
  - Evaluation Process Input & Output
  - Challenges

- Future Works

- Q&A

SeaSe

https://github.com/SeaseLtd/rated-ranking-evaluator



RRE: What is it?

- A set of **search quality evaluation tools**
- A **search quality evaluation framework**
- **Multi** (**search**) platform
- Written in **Java**
- It can be used also in **non-Java projects**
- Licensed under **Apache 2.0**
- Open to **contributions**
- **Extremely dynamic!**

SeaSe

RRE: At a glance

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

Apache Lucene/Solr London Meetup

2 People	10 Modules	48950 Lines of Code	2 Months
2 People	10 Modules	67317 Lines of Code	5 Months

elasticsearch

SeaSe

RRE Ecosystem

The picture illustrates the **main modules** composing the RRE ecosystem.

All modules with a **dashed border** are planned for a **future release**.

RRE CLI has a double border because although the **rre-cli** module **hasn't been** developed, you can run **RRE from a command line** using **RRE Maven archetype**, which is part of the current release.

As you can see, the current implementation includes **two** target search platforms: **Apache Solr** and **Elasticsearch**.

The **Search Platform API module** provide a **search platform abstraction** for plugging-in additional search systems.

SeaSe

RRE: Available metrics

- Precision
- Recall
- Precision at 1 (P@1)
- Precision at 2 (P@2)
- Precision at 3 (P@3)
- Precision at 10 (P@10)
- Average Precision (AP)
- Reciprocal Rank
- Mean Reciprocal Rank
- Mean Average Precision (MAP)
- Normalised Discounted Cumulative Gain
- **F-Measure** >>>> **Compound Metric**

Available Metrics

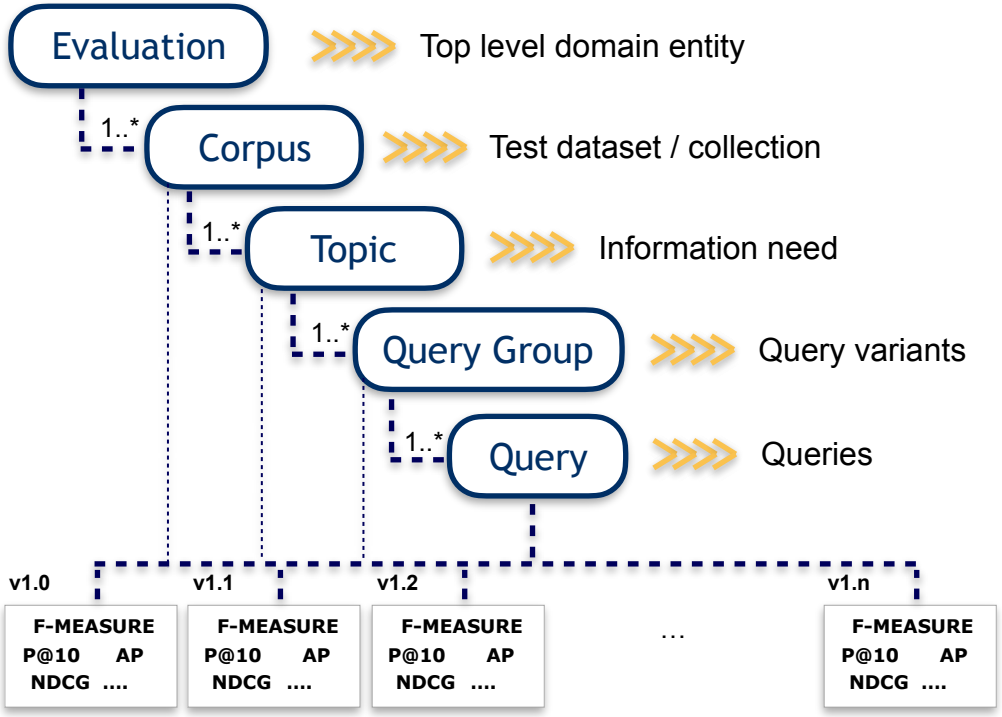
These are the RRE **built-in metrics** which can be used out of the box.

The most part of them are computed at query level and then aggregated at upper levels.

However, compound metrics (e.g. **MAP,** or **GMAP**) are **not explicitly declared** or defined, because the computation doesn't happen at query level. The result of the aggregation executed on the upper levels will automatically produce these metric.

For example, the **Average Precision** computed for Q1, Q2, Q3, Qn becomes the **Mean Average Precision** at Query Group or Topic levels.

RRE: Domain Model (1/2)



RRE Domain Model

RRE **Domain Model** is organized into a **composite** / **tree-like** structure where the relationships between entities are **always 1 to many**.

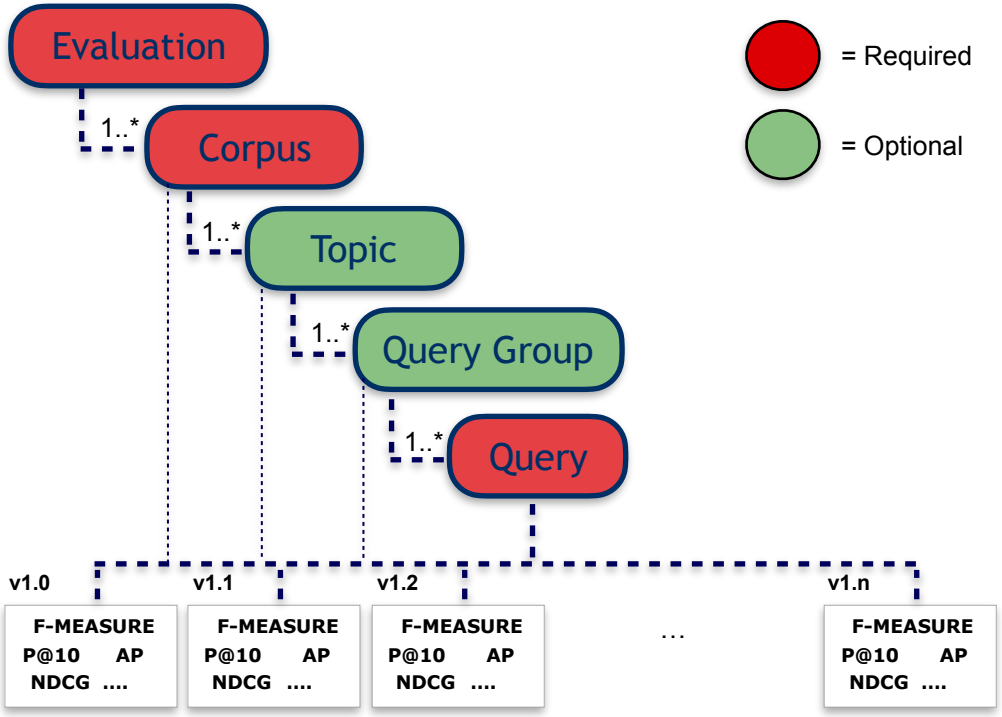
The top level entity is a placeholder representing an **evaluation execution**.

Versioned metrics are computed at **query level** and then reported, using an **aggregation function**, at upper levels.

The benefit of having a composite structure is clear: we can see a **metric value at different levels** (e.g. a query, all queries belonging to a query group, all queries belonging to a topic or at corpus level)

SeaSe

RRE: Domain Model (2/2)



RRE Domain Model

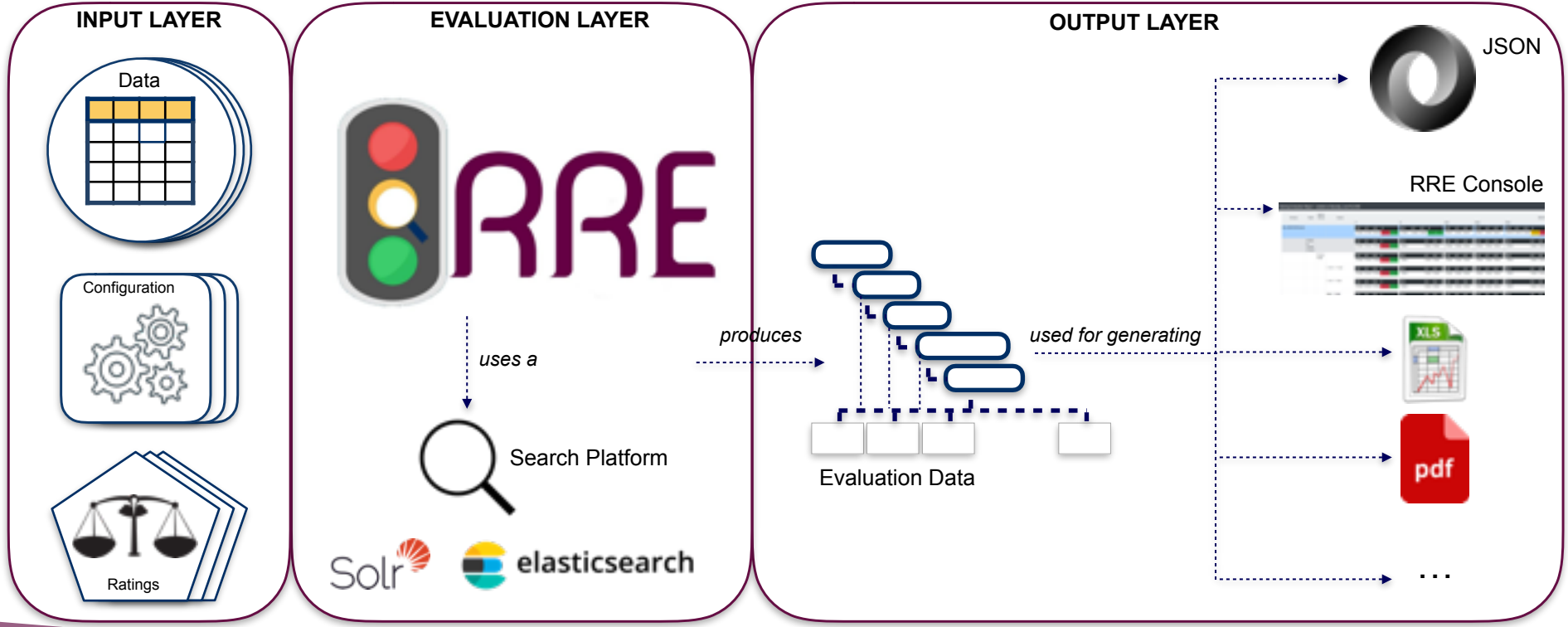
Although the domain model structure is able to capture complex scenarios, sometimes we want to model simpler contexts.

In order to avoid verbose and redundant ratings definitions it's possibile to omit some level. Specifically we can be in one of the following:

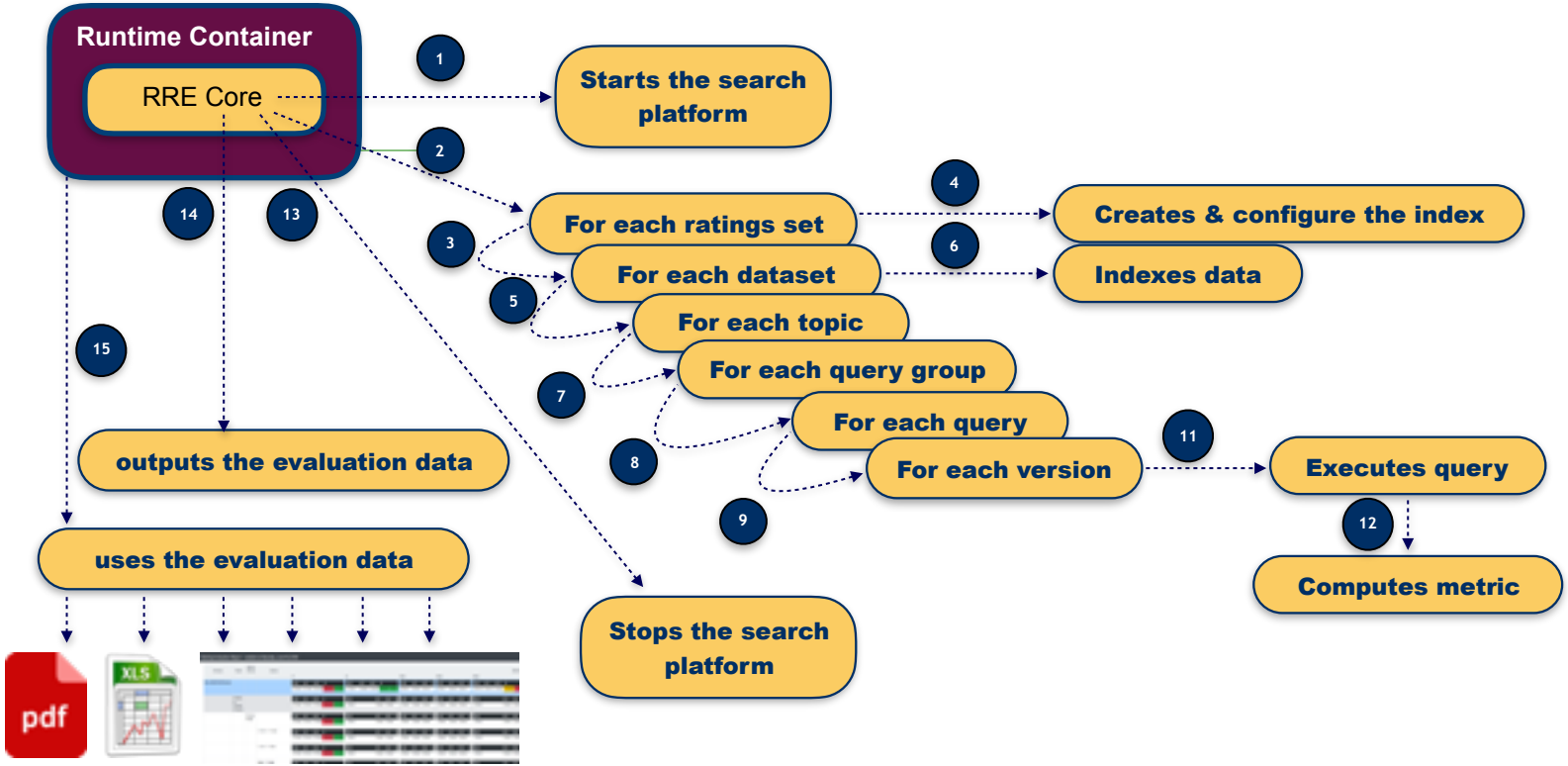
- only queries
- query groups and queries
- topics, query groups and queries

SeaSe

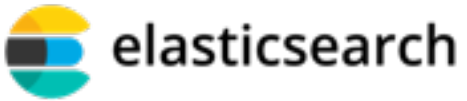
RRE: Evaluation process overview (1/2)



RRE: Evaluation process overview (2/2)




```
[
  {
    "id": 1,
    "name": "Fender Jazz Bass"
  },
  {
    "id": 2,
    "name": "Fender Precision Bass"
  },
  {
    "id": 3,
    "name": "Warwick Corvette"
  },
  {
    "id": 4,
    "name": "Warwick Thumb"
  }
]
```




```
{"index" : {"_index":"dataset1","_type":"doc","_id":"1"}}
{"name":"Fender Jazz Bass"}
{"index" : {"_index":"dataset1","_type":"doc","_id":"2"}}
{"name":"Fender Precision Bass"}
{"index" : {"_index":"dataset1","_type":"doc","_id":"3"}}
{"name":"Warwick Corvette"}
{"index" : {"_index":"dataset1","_type":"doc","_id":"4"}}
{"name":"Warwick Thumb"}
```

Corpora

An evaluation execution can involve **more than one datasets** targeting a given **search platform**.

A dataset consists consists of **representative** domain data; although a compressed dataset can be provided, generally it has a small/medium size.

Within RRE, **corpus, dataset, collection** are synonyms.

Datasets must be **located** under a **configurable folder**. Each dataset is then referenced in one or more **ratings file**.

Configuration Sets

The search platform **configuration evolves over time** (e.g. change requests, enhancements, bugs)

RRE encourages an incremental approach for managing the configuration instances. Even for internal or small iterations, each time we make a relevant change to the current configuration, it's better to **clone it and move forward** with a new version.

In this way we'll end up having t**he historical progression** of our system, and RRE will be able to **make comparisons**.

The evaluation process allows you to define **inclusion** / **exclusion** rules (i.e. include only version 1.0 and 2.0)

SeaSe

```
"queries": [
  {
    "template": "only_q.json",
    "placeholders": {
      "$query": "fender"
    }
  },
{
  "template": "filter_by_language.json",
  "placeholders": {
    "$query": "Fender",
    "$lang": "eng"
}
```

only_q.json

```
{
  "q" : "$query"
}
```

filter_by_language.json

```
{
  "q" : "$query",
  "fq" : "language:$lang"
}
```

```
"topics": [
  {
    "description": "Out of warranty",
    "query_groups": [
      {
        "name": "Out of warranty variants",
        "description": "The group tests several v
        "template": "query-shape-${version}.json"
        "queries": [
          {
            "placeholders": {
              "#$query##": "oow"
            }
          },
          {
            "placeholders": {
              "#$query##": "Oow"
            }
```

```
"function_score": {
  "query": {
    "bool": {
      "should": [
        {
          "constant_score": {
            "filter": {
              "match": {
                "classification_language_1": "#$query##"
              }
            },
            "boost": 3
          }
        },
        {
          "constant_score": {
```

Query templates

For each **query** or **query group**) it's possible to define a **template**, which is a kind of **query shape** containing one or more **placeholders**.

Then, in the ratings file you can reference one of those defined templates and you can provide **a value for each placeholder**.

Templates have been introduced in order to:

- allow a **common query management** between search platforms
- define **complex queries**
- define **runtime parameters** that cannot be statically determined (e.g. filters)

SeaSe

RRE: Ratings


```
{
  "index" : "corel",
  "corpora_file": "electric_basses.json",
  "id_field": "id",
  "topics" : [
    {
      "description": "Fender basses",
      "query_groups": [
        {
          "name": "Brand search",
          "description": "The group tests several searches on the Fender bran
          "queries": [
            {
              "template": "only_q.json",
              "placeholders": {
                "$query": "fender"
              }
            },
            {
              "template": "only_q.json",
              "placeholders": {
                "$query": "Fender"
              }
            }
          ],
          "relevant_documents": {
            "1": {
              "gain": 3
            },
            "2": {
              "gain": 3
            }
          }
        },
        {
          "name": "Jazz bass search",
          "description": "Several searches on a given model (Jazz bass)",
          "queries": [
```

OR

```
          ],
          "relevant_documents": {
            "3": [
              "1",
              "2"
            ]
```

Ratings

Ratings files associate the RRE domain model **entities** with r**elevance judgments**. A ratings file provides the association **between queries and relevant documents**.

There must be **at least one** ratings file (otherwise no evaluation happens). Usually there's a 1:1 relationship between a rating file and a dataset.

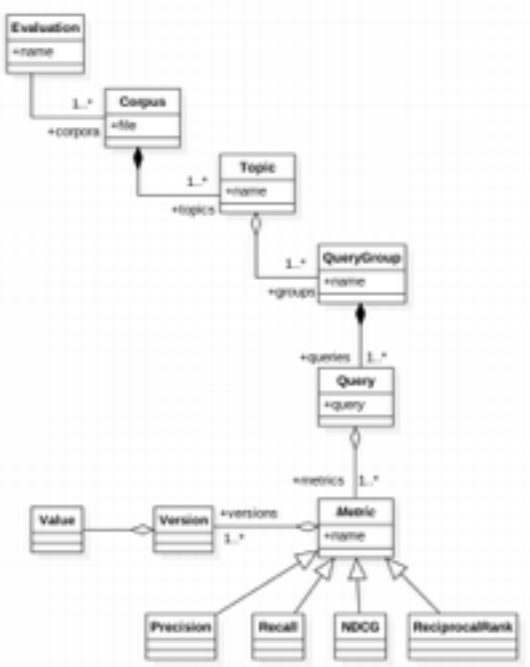
Judgments, the most important part of this file, consist of a **list of all relevant documents** for a **query group.**

Each listed document has a corresponding "**gain**" which is the **relevancy judgment** we want to assign to that document.

SeaSe

RRE: Evaluation Output


```
{
  "name" : "Ranking Evaluation Report - created on Friday, June 15, 2018",
  "metrics" : {...},
  "corpora" : [ {
    "name" : "bfa_15MAY2018.json",
    "metrics" : {...},
    "topics" : [ {
      "name" : "Fridges by number of doors",
      "metrics" : {...},
      "query-groups" : [ {
        "name" : "2 doors fridge",
        "metrics" : {...},
        "query-evaluations" : [ {
          "metrics" : {...},
          "query" : "2 doors fridge",
          "results" : {...}
        }, {
          "metrics" : {
            "P" : {
              "name" : "P",
              "versions" : {
                "v1.0" : {
                  "value" : "0.1546"
                },
                "v1.1" : {
                  "value" : "0.1539"
                },
                "v1.2" : {
                  "value" : "0.1798"
                }
              },
              "trend" : [ -0.0007, 0.0259 ]
            },
            "R" : {
              "name" : "R",
              "versions" : {
                "v1.0" : {
                  "value" : "1.0000"
                },
                "v1.1" : {
                  "value" : "1.0000"
                },
                "v1.2" : {
                  "value" : "1.0000"
                }
              },
              "trend" : [ 0.0000, 0.0000 ]
            },
            "AP" : {
              "name" : "AP",
              "versions" : {
                "v1.0" : {
                  "value" : "1.00000000"
```

Evaluation output

The **RRE Core** itself is a **library**, so it outputs its result as a **Plain Java object** that must be programmatically used.

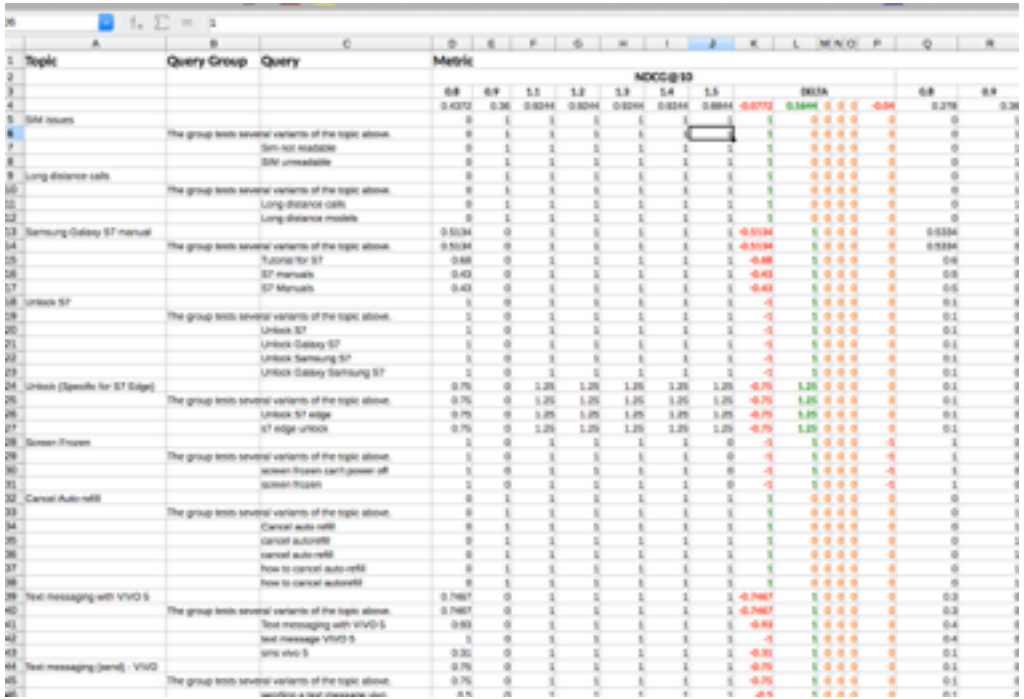
However when wrapped within a **runtime container**, like the **Maven Plugin**, the evaluation object tree is marshalled in **JSON format**.

Being interoperable, the JSON format can be used by some other component for producing a different **kind of output**.

An example of such usage is the **RRE Apache Maven Reporting Plugin** which can

- output a **spreadsheet**
- send the evaluation data to a running **RRE Server**

SeaSe



Workbook

The RRE domain model (topics, groups and queries) is on the **left** and each metric (on the right section) has a **value** for each **version / entity** pair.

In case the evaluation process includes multiple datasets, there will be a **spreadsheet** for each of them.

This output format is useful when

- you want to have (or maintain somewhere) **a snapshot** about how the system performed in a given moment
- the comparison includes a **lot of versions**
- you want to include **all available metrics**

SeaSe

RRE Server

The RRE console is a **SpringBoot/AngularJS** application which shows real-time information about **evaluation results**.

Each time a build happens, the **RRE reporting plugin** sends the evaluation result to a **RESTFul endpoint** provided by RRE Server.

The received data **immediately updates** the web dashboard with fresh data.

Useful during the **development / tuning phase iterations** (you don't have to open again and again the excel report)

RRE: RRE Server (2/2)

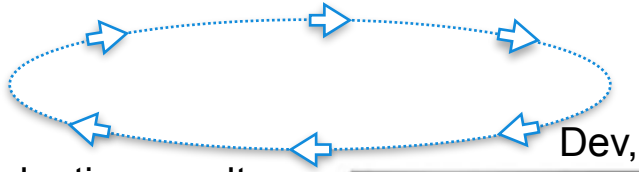
The evaluation data, at **query / version level**, collects the **top n search results**.

In the web console, under each query, there's a **little arrow** which allows to open / hide the section which contains those results.

In this way you can get immediately the meaning of each metric and its values between different versions.

In the example above, you can immediately see **why** there's a **loss of precision** (first metric) between v1.0, v1.1, which got fixed in v1.2

RRE: Iterative development & tuning



Check evaluation results

Dev, tune & Build

We are thinking about how to fill a third monitor

RRE: Challenges

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

"I think if we could create a simplified **pass/fail** report for the business team, that would be ideal. So they could understand the tradeoffs of the new search."

Do I have to write all judgments manually??

"Many search engines process the user query heavily before it's submitted to the search engine in whatever DSL is required, and if you don't retain some idea of the original query in the system how can you" relate the test results back to user behaviour?

How can I use RRE if I have a custom search platform?

Java is not in my stack

SeaSe

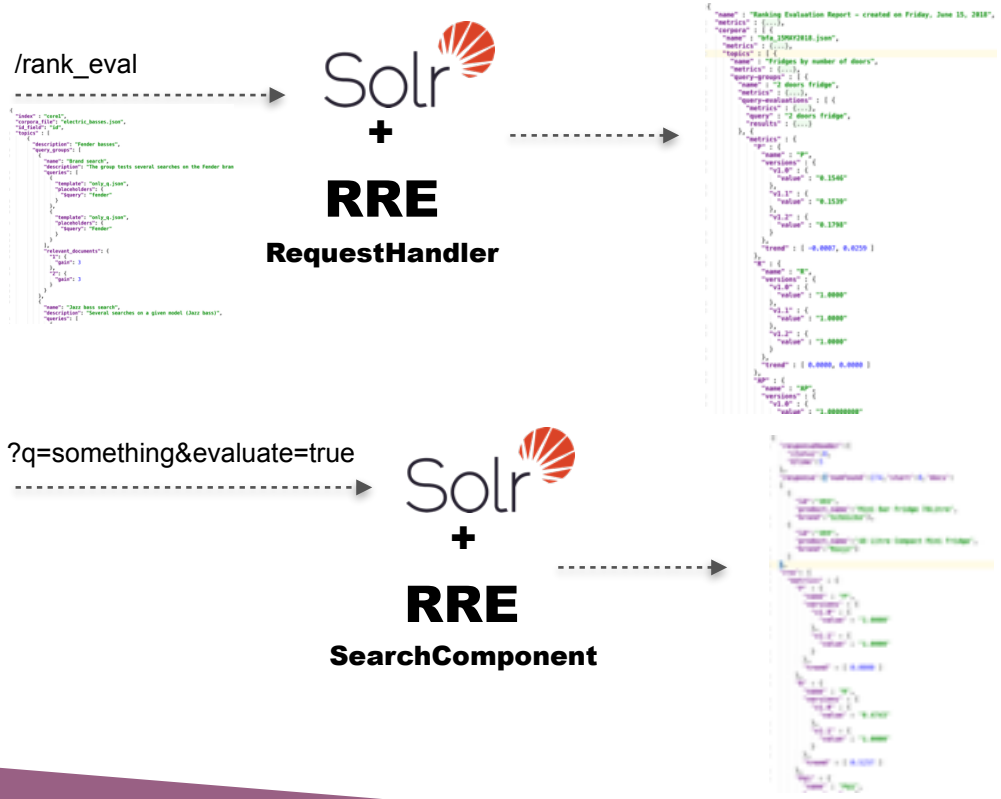
Agenda

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

- ➢ Search Quality Evaluation
- ➢ Rated Ranking Evaluator
- ✓ **Future Works / Idea**
- ➢ Q&A

SeaSe

Future Works: Solr Rank Eval API



Rank Eval API

The **RRE core** can be used for implementing a **RequestHandler** which will be able to expose a **Ranking Evaluation endpoint**.

That would result in the same functionality introduced in **Elasticsearch 6.2** [1] with some differences.

- rich tree data model
- metrics framework

Note that in this case it doesn't make so much sense to provide comparisons between versions.

As part of the same module there could be a **SearchComponent** for evaluating a single query interaction.

[1] https://www.elastic.co/guide/en/elasticsearch/reference/6.2/search-rank-eval.html

SeaSe

Future Works: Jenkins Plugin

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

Jenkins Plugin

RRE Maven plugin already produces the **evaluation data** in a **machine-readable format** (JSON) which can be consumed by another component.

The **Maven RRE Report plugin** or the **RRE Server** are just two examples of such consumers.

RRE can be **already executed** in a **Jenkins** CI build cycle (using the **Maven plugin**).

By means of a dedicated J**enkins plugin**, the evaluation data could be graphically displayed in the Jenkins dashboard. It could be even used for blocking builds which produce bad evaluation results.

SeaSe

Future Works: Building the input

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

The main input for RRE is the Ratings file, in JSON format.
Writing a comprehensive JSON to detail the ratings sets for your Search ecosystem can be expensive!

Judgement Collector UI

1. Explicit feedback from users judgements
2. An intuitive UI allow judges to run queries, see documents and rate them
3. Relevance label is explicitly assigned by domain experts

Users Interactions Logger

1. Implicit feedback from users interactions (Clicks, Sales …)
2. Log to disk / internal Solr instance for analytics
3. Estimate <q,d> relevance label based on Click Through Rate, Sales Rate

SeaSe

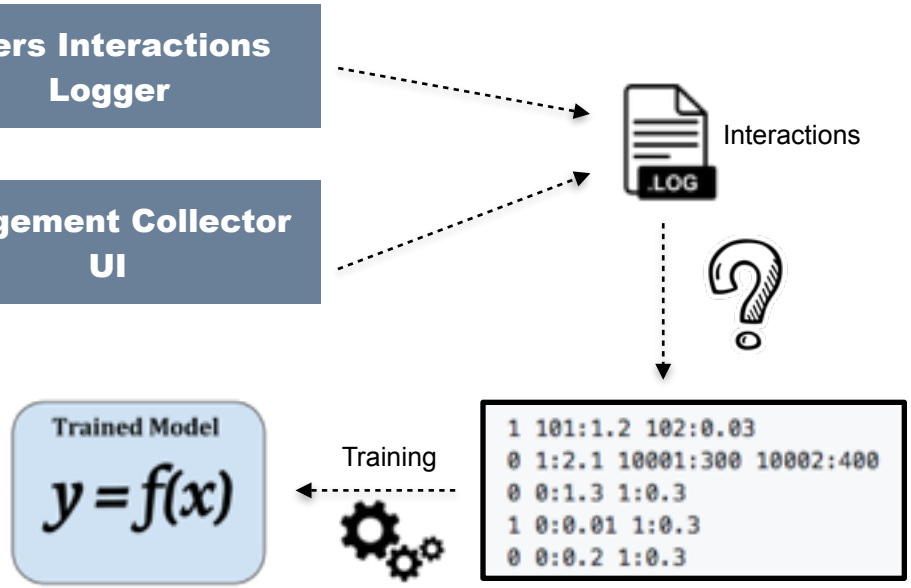
Future Works: Learning To Rank

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

Once you collected the ratings, could we use them to actively improve the quality metrics ?

Learning To Rank

"Learning to rank is the application of machine learning, typically supervised, semi-supervised or reinforcement learning, in the construction of ranking models for information retrieval systems." Wikipedia



SeaSe

Future Works: Training Set Building

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

Creating a Learning To Rank Training Set from the collected interactions is not going to be trivial.
It normally requires ad hoc data manipulation depending on the use case…
… but some steps could be automated and make available for a generic configurable approach

- Null feature sanitisation
- Query Id calculation
- Query document feature generation
- Single/Multi valued categorical feature encoding

Configuration

1. Ad Hoc category, Artificial values, keep NaN
   -> depends of Training Library to use
2. Optional Query Level features to be hashed as QueryId
3. Intersect related query and document level categorical features to generate Ordinal query-document features
4. Label Encoding ? One Hot Encoding? Binary Encoding? [1]
   ⚠ Dummy Variable Trap

[1] https://www.datacamp.com/community/tutorials/categorical-data

SeaSe

Future Works: Training Set Building

HAYSTACK
THE SEARCH RELEVANCE CONFERENCE!

What about the relevance label for each training vector ?
Can we estimate it from the interactions collected ?

- Interaction Type Counts
- Click Through Rate/Sales Through Rate calculation
- Relevance label normalisation

Configuration

1. Impressions? Clicks? Bookmarks? Add To Charts? Sales?
2. Define the objective: Clicks/Impressions ? Sales/Impressions?
3. Relevance Label : 0…4

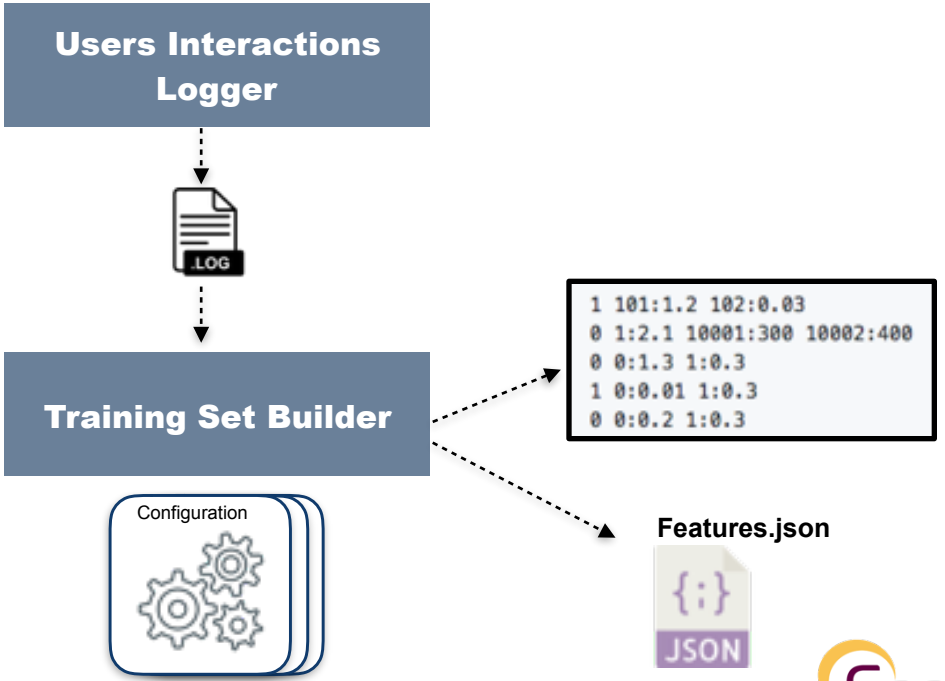
SeaSe

Future Works: Learning To Rank Solr Configs

Can the features.json configuration generation be automated?

Learning To Rank - Solr features.json

The features.json is a configuration file necessary for Solr Learning To Rank extension to work.
It is a configuration file that describes how the features that were used at training time for the model can be extracted at query time.
This file is coupled both with the training set features and the query time features.



- Search Quality Evaluation
- Rated Ranking Evaluator
- Future Works

✓ **Q&A**

Search Quality Evaluation

Tools and techniques

Alessandro Benedetti - Software Engineer
Andrea Gazzarini - Software Engineer

2nd October 2018

SeaSe