Elastic Search: Similarity & Scoring[1–4]

How Retrieved Documents are Ranked

Weimao Ke

Drexel University

Table of contents

1. Mapping and Built-in Similarity
2. Custom Similarities
3. Script Score Query
4. Practical Scoring with Relevance Signals

Mapping and Built-in Similarity

Mapping Similarity

With Mapping:

- You can configure a scoring or similarity function for each field.
- Basic similarity functions include: *BM25* (default), *classic* (TF*IDF), and *boolean*
- Custom similarities can be configured with related parameters.

This changes the way in which retrieved documents are ranked (sorted).

It is the most important function of a search engine.

Mapping Similarity

Example:

```
PUT my_index
{
  "mappings": {
    "properties": {
      "field1": {
        "type": "text"
      },
      "field2": {
        "type": "text",
        "similarity": "boolean"
      }
    }
  }
}
```

- By default, *field1* uses *BM25* similarity.
- *Field2* is configured to use the *boolean* similarity.

Mapping Similarity

You can change the default similarity for a new index:

```
PUT /index
{
  "settings": {
    "index": {
      "similarity": {
        "default": {
          "type": "boolean"
        }
      }
    }
  }
}
```

This is done when the index is created.

Mapping Similarity

You can change the default similarity of an existing index.

You will have to close the index to change it:

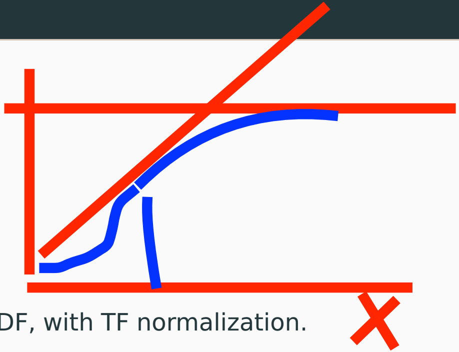
```
POST /index/_close

PUT /index/_settings
{
  "index": {
    "similarity": {
      "default": {
        "type": "boolean"
      }
    }
  }
}
```

And reopen it after the change:

```
POST /index/_open
```

Custom Similarities

Similarity Modules



BM25 similarity (default):

- Regarded as a version of TF*IDF, with TF normalization.
- k_1: controls TF normalization (saturation), default 1.2;
- b: controls document length normalization, default 0.75.

Similarity Modules

DFR (Divergence from Randomness) similarity:

- *basic_model*: possible values *g*, *if*, *in*, and *ine*;
- *after_effect*: possible values *b* and *l*;
- *normalization*: possible values *no*, *h*1, *h*2, *h*3, and *z*.

All options but the first option need a normalization value.

Similarity Modules

Other similarity functions include:

- DFI (Divergence from Independence) similarity
- IB (information based model) similarity
- LM Dirichlet similarity
- LM Jelinek Mercer similarity

Mapping Similarity

One can create a customized version of a built-in similarity function:

```
PUT /index
{
  "settings": {
    "index": {
      "similarity": {
        "my_similarity": {
          "type": "DFR",
          "basic_model": "g",
          "after_effect": "l",
          "normalization": "h2",
          "normalization.h2.c": "3.0"
        }
      }
    }
  }
}
```

Mapping Similarity

And then use the custom version with mapping:

```
PUT /index/_mapping
{
  "properties" : {
    "title" : { "type" : "text", "similarity" : "my_similarity" }
  }
}
```

Mapping Similarity

ElasticSearch also allows scripted similarity:

```
PUT /index
{
  "settings": {
    "number_of_shards": 1,
    "similarity": {
      "custom_tfidf": {
        "type": "scripted",
        "script": {
          "source": "
            double tf = Math.sqrt(doc.freq);
            double idf = Math.log((field.docCount+1.0)/(term.docFreq+1.0)) + 1.0;
            double norm = 1/Math.sqrt(doc.length);
            return query.boost * tf * idf * norm;"
        }
      }
    }
  },
```

Mapping Similarity

Scripted similarity (continued):

```
  "mappings": {
    "properties": {
      "field": {
        "type": "text",
        "similarity": "custom_tfidf"
      }
    }
  }
}
```

Mapping Similarity

Note:

- *doc.freq* is NOT document frequency (DF), but a term's frequency in the document (TF).
- *term.docFreq* is document frequency (DF).

Rules on scripted similarity:

- Returned scores must be positive;
- All other variables remaining equal, scores must not decrease when term frequencies (TF) increases;
- All other variables remaining equal, scores must not increase when document length increases.

Mapping Similarity

You may also use *weight_script* for more efficient scoring:

```
PUT /index
{
  "settings": {
    "number_of_shards": 1,
    "similarity": {
      "scripted_tfidf": {
        "type": "scripted",
        "weight_script": {
          "source": "double idf =
            Math.log((field.docCount+1.0)/(term.docFreq+1.0)) + 1.0;
            return query.boost * idf;"
        },
        "script": {
          "source": "
            double tf = Math.sqrt(doc.freq);
            double norm = 1/Math.sqrt(doc.length);
            return weight * tf * norm;"
        }
      }
    }
  },
```

Mapping Similarity

weight_script (continued):

```
  "mappings": {
    "properties": {
      "field": {
        "type": "text",
        "similarity": "scripted_tfidf"
      }
    }
  }
}
```

Script Score Query

Script Score Query

Query with a score script:

- *script_score* provides a custom score as part of a query request.
- Changes the way in which returned documents are scored and ranked for the query.

Script Score Query

Example script score query:

```
GET /_search
{
    "query" : {
        "script_score" : {
            "query" : {
                "match": { "message": "Best Video Tutorials" }
            },
            "script" : {
                "source" : "doc[’likes’].value / 10 "
            }
        }
     }
}
```

Suppose *doc*[′*likes*′] is a field with the count of likes.

Here it ranks documents according to the number of likes.

Script Score Query

Parameters for *script_score*:

- *query*: the query to search and return documents;
- *script*: the script used to compute the score of each returned document;
- *min_score* (optional): the threshold score for a document to be included in the search results (hits).

Note that *_score* variable has the current relevance score for each document and can be used as part of the new scoring function.

Predefined functions for *script_score*

Saturation:

$$saturation(value, k) = value/(k + value)$$

```
  "script" : {
      "source" : "saturation(doc[’likes’].value, 1)"
  }
```

that is $\frac{likes}{1+likes}$:

- 0 when $likes = 0$;
- 0.5 when $likes = 1$;
- 1 when $likes \to \infty$

Predefined functions for *script_score*

Sigmoid:

$$sigmoid(value, k, a) = value^a / (k^a + value^a)$$

```
"script" : {
    "source" : "sigmoid(doc[’likes’].value, 2, 1)"
}
```

that is $\frac{likes^1}{2^1+likes^1} = \frac{likes}{2+likes}$

Predefined functions for *script_score*

Random score:

randomScore(< *seed* >, < *fieldName* >)

```
"script" : {
    "source" : "randomScore(100)"
}
```

- This generates some randomness (noise) in the ranking, e.g. for random sampling in the search results.
- < *fieldName* > parameter, as the source of randomness, is optional. Document ID is used by default to generate the random score.

Predefined functions for *script_score*

Decay functions:

- Provides scoring based on some kind of distance for numeric values, dates, and geo locations.

An example for geo fields:

```
"script" : {
    "source" : "decayGeoExp(params.origin, params.scale,
        params.offset, params.decay, doc[’location’].value)",
    "params": {
        "origin": "40, −70.12",
        "scale": "200km",
        "offset": "0km",
        "decay" : 0.2
    }
}
```

Predefined functions for *script_score*

Functions for vector fields (experimental):

- Functions to be used on *dense_vector* and *sparse_vector* fields.
- A score is computed for every matched document so try to limit the number of docs with a query parameter.

An index with vector fields:

```
PUT my_index
{
  "mappings": {
    "properties": {
      "my_dense_vector": {
        "type": "dense_vector",
        "dims": 3
      },
      "my_sparse_vector" : {
        "type" : "sparse_vector"
      },
      "status" : {
        "type" : "keyword"
      }
    }
  }
```

Predefined functions for *script_score*

Example vector data:

```
PUT my_index/_doc/1
{
  "my_dense_vector": [0.5, 10, 6],
  "my_sparse_vector": {"2": 1.5, "15" : 2, "50": −1.1, "4545": 1.1},
  "status" : "published"
}

PUT my_index/_doc/2
{
  "my_dense_vector": [−0.5, 10, 10],
  "my_sparse_vector": {"2": 2.5, "10" : 1.3, "55": −2.3, "113": 1.6},
  "status" : "published"
}
```

Predefined functions for *script_score*

A script score query using cosine similarity:

```
GET my_index/_search
{
  "query": {
    "script_score": {
      "query" : {
        "bool" : {
          "filter" : {
            "term" : {
              "status" : "published"
            }
          }
        }
      },
      "script": {
        "source": "cosineSimilarity(params.query_vector, doc[’my_dense_vector’]) + 1.0",
        "params": {
          "query_vector": [4, 3.4, –0.2]
        }
      }
    }
  }
}
```

Predefined functions for *script_score*

Cosine similarity of sparse vectors:

```
GET my_index/_search
{
  "query": {
    "script_score": {
      "query" : {
        "bool" : {
          "filter" : {
            "term" : {
              "status" : "published"
            }
          }
        }
      },
      "script": {
        "source": "cosineSimilaritySparse(params.query_vector, doc[’my_sparse_vector’]) + 1.0",
        "params": {
          "query_vector": {"2": 0.5, "10" : 111.3, "50": -1.3, "113": 14.8, "4545": 156.0}
        }
      }
    }
  }
```

Predefined functions for *script_score*

Other functions for vector fields:

- *dotProduct* computes the dot product of two vectors;
- *dotProductSparse* is the sparse vector version for dot product calculation;
- *l*1*norm* or *l*1*normSparse* calculates the L^1 distance, i.e. Manhattan distance;
- *l*2*norm* or *l*2*normSparse* ccalculates the L^2 distance, i.e. Euclidean distance

Practical Scoring with Relevance Signals

Relevance Signals

Additional relevance signals:

- Relevance based on terms is not the only source of evidence.
- Other signals correlated with relevance: quality, reputation/popularity, importance, etc.
- Hyperlinks, PageRanks, # of likes, etc. are signals that can be used in relevance scoring (ranking).
- Can be implemented using *script_score* or *rank_feature* queries.

Relevance Signals

Scoring with PageRank (using *script_score*):

```
GET index/_search
{
    "query" : {
        "script_score" : {
            "query" : {
                "match": { "body": "University" }
            },
            "script" : {
                "source" :
                  "_score * saturation(doc[’pagerank’].value, 10)"
            }
        }
    }
}
```

that is $bm25\_score \times \frac{pagerank}{10+pagerank}$

Here *pagerank* must be mapped as a *numeric* field.

Relevance Signals

Scoring with PageRank (using *rank_feature*):

```
GET _search
{
    "query" : {
        "bool" : {
            "must": {
                "match": { "body": "University" }
            },
            "should": {
                "rank_feature": {
                    "field": "pagerank",
                    "saturation": {
                        "pivot": 10
                    }
                }
            }
        }
    }
}
```

Here *pagerank* must be mapped as a *rank_feature* field.

Relevance Signals

Comparing *script_score* vs. *rank_feature*:

- *script_score* is more flexible and offers more ways to combine text relevance with other signals.
- *rank_feature* is limited but more efficient (faster);
- *rank_feature* fields are indexed signals, based on which the query can skip non-competitive documents and find the top matches much faster.

Relevance Signals

More on *rank_feature*:

- *rank_feature* is typically used in the *should* (OR) clause of a *bool* query;
- this is to *add* the signal to the other scores.
- supports a few functions: saturation, logarithm, and sigmoid.
- must setup *rank_feature* or *rank_features* fields with the index.

Relevance Signals

Setup an index for *rank_feature*:

```
PUT /test
{
  "mappings": {
    "properties": {
      "pagerank": {
        "type": "rank_feature"
      },
      "url_length": {
        "type": "rank_feature",
        "positive_score_impact": false
      },
      "topics": {
        "type": "rank_features"
      }
    }
  }
}
```

positive_score_impact : *false* for URL’s negative impact on relevance; *rank_features* for the topics field that will contain a list of topics and their relevance.

Relevance Signals

Index documents with *rank_feature*(*s*):

```
PUT /test/_doc/1?refresh
{
  "url": "http://en.wikipedia.org/wiki/2016_Summer_Olympics",
  "content": "Rio 2016",
  "pagerank": 75.3,
  "url_length": 42,
  "topics": {
    "sports": 50,
    "brazil": 30
  }
}

PUT /test/_doc/2?refresh
{
  "url": "http://en.wikipedia.org/wiki/2016_Brazilian_Grand_Prix",
  "content": "Formula One motor race held on 13 November 2016",
  "pagerank": 38.9,
  "url_length": 47,
  "topics": {
    "sports": 35,
    "formula one": 65,
    "brazil": 20
  }
}
```

Relevance Signals

Query with *rank_feature*(*s*):

```
GET /test/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "2016"
          }
        }
      ],
      "should": [
        {
          "rank_feature": {
            "field": "pagerank"
          }
        },
```

Relevance Signals

Query with *rank_feature*(*s*) (continued):

```
        {
          "rank_feature": {
            "field": "url_length",
            "boost": 0.1
          }
        },
        {
          "rank_feature": {
            "field": "topics.sports",
            "boost": 0.4
          }
        }
      ]
    }
  }
}
```

boost (relative to 1.0) is to boost – increase (> 1) or decrease (< 1) – a relevance score.

Relevance Signals

Saturation – example $\frac{pagerank}{pagerank+8}$:

```
GET /test/_search
{
  "query": {
    "rank_feature": {
      "field": "pagerank",
      "saturation": {
        "pivot": 8
      }
    }
  }
}
```

Relevance Signals

Logarithm – example $\log 4 + \mathit{pagerank}$:

```
GET /test/_search
{
  "query": {
    "rank_feature": {
      "field": "pagerank",
      "log": {
        "scaling_factor": 4
      }
    }
  }
}
```

Relevance Signals

Sigmoid – example $\frac{pagerank^{0.6}}{pagerank^{0.6}+7^{0.6}}$:

```
GET /test/_search
{
  "query": {
    "rank_feature": {
      "field": "pagerank",
      "sigmoid": {
        "pivot": 7,
        "exponent": 0.6
      }
    }
  }
}
```

References

[1] elastic.co. Elasticsearch reference [7.5]: Index moduels: Similarity. `https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html`, . Accessed: 2020-1-16.

[2] elastic.co. Elasticsearch reference [7.5]: How to: Recipes: Incorporating static relevance signals into the score. `https://www.elastic.co/guide/en/elasticsearch/reference/current/static-scoring-signals.html`, . Accessed: 2020-1-16.

References ii

[3] elastic.co. Elasticsearch reference [7.5]: Query dsl: Script score query. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-script-score-query.html, . Accessed: 2020-1-16.

[4] elastic.co. Elasticsearch reference [7.5]: Mapping: Similarity. https://www.elastic.co/guide/en/elasticsearch/reference/current/similarity.html, . Accessed: 2020-1-16.