Elasticsearch Learning to Rank Documentation

Pere Urbon

May 14, 2018

Contents

1 Get started **3**

2 Installing **5**

3 Heeelp! **7**

4 Contents **9**

4.1 The Theory . . . 9
4.1.1 Ranking can be biased . . . 9
4.1.2 The FA*IR algorithm . . . 10
4.1.3 References . . . 11
4.2 The Plugin . . . 11
4.2.1 What the Fairsearch plugin does . . . 11
4.2.2 What the plugin does NOT . . . 11
4.3 How to use the fair-search plugin . . . 11
4.3.1 Assumptions and preconditions for this example . . . 12
4.3.2 How does a search looks like . . . 12
4.4 Advance usage . . . 13
4.4.1 Building M tables . . . 13
4.5 Contact . . . 15

5 Indices and tables **17**

The *Fairsearch* plug-in for Elasticsearch is an implementation of the FA*IR algorithm. It enables you to apply a *positive action* policy in which elements are re-ranked to ensure a fair representation of minorities or disadvantaged people.

This plugin has been developed by Pere Urbón in collaboration with researchers at TU Berlin and Pompeu Fabra University, with support from a grant by the Data Transparency Lab.

CHAPTER 1

Get started

- Check whether this plug-in is for you: *The Theory*, *The Plugin*
- Understand the fairness criterion applied: *The Theory*
- Use the plug-in to perform a re-ranking: *How to use the fair-search plugin*
- Advance usage, like build an M table: *Advance usage*

CHAPTER 2

Installing

Pre-built versions can be found here. Want a build for an ES version? Follow the instructions in the README for building or create an issue. Once you've found a version compatible with your Elasticsearch, you'd run a command such as:

```
./bin/elasticsearch-plugin install https://fair-search.github.io/fair-reranker/fairsearch-1.0-es6.1.2-SNAPSHOT.zip
```

(It's expected you'll confirm some security exceptions, you can pass -b to elasticsearch-plugin to automatically install)

CHAPTER 3

Heeelp!

- If you have questions or feedback, see *Contact*

CHAPTER 4

Contents

4.1 The Theory

4.1.1 Ranking can be biased

Core concepts: ranking bias

Search engines today are used to rank many different types of items, including items that represent *people*. Job recruiting search engines, marketplaces for consulting and other services, dating apps, etc. have at its core the idea of ranking/ordering people from most relevant to less relevant, which often means from "best" to "worst".

Traditional scoring methods such as TF-IDF or BM25 (the two most popular ones) can introduce a certain degree of bias; the main motivation of this plugin is to provider methods to search without having this problem.

A computer system is biased [Friedman 1996] if:

> It systematically and unfairly discriminate[s] against certain individuals or groups of individuals in favor of others. A system discriminates unfairly if it denies an opportunity or a good or if it assigns an undesirable outcome to an individual or a group of individuals on grounds that are unreasonable or inappropriate.

In algorithmic bias, an important concept is that of a *protected group*, which is a category of individuals protected by law, voluntary commitments, or other reasons. Search results are considered *unfair* if members of a protected group are systematically ranked lower than those of a non-protected group.

Examples where a fair search would be required are for instance, the US Equal Employment Opportunity Commission, which sets a goal of 12% of workers with disabilities in federal agencies in the US, while in Spain, a minimum of 40% of political candidates in voting districts exceeding a certain size must be women.

Real-world example: Job Search

Consider the three rankings in the table below, corresponding to searches for an "economist," "market research analyst," and "copywriter" in a job search engine, i.e., an online platform for jobs that is used by recruiters and headhunters to find suitable candidates.

Positions 1, 2, . . . , 10 are the top-10 ranking positions. A letter `m` indicates the candidate is a man, while `f` indicates the candidate is a woman.

Query	1	2	3	4	5	6	7	8	9	R10	Men:Women (top-10)	Men:Women (top-40)
Econ.	f	m	m	m	m	m	m	m	m	m	90%:10%	73%:27%
Analyst	f	m	f	f	f	f	f	m	f	f	20%:80%	43%:57%
Copywr.	m	m	m	m	m	m	f	m	m	m	90%:10%	73%:27%

While analyzing the extent to which candidates of both genders are represented as we go down these lists, we can observe that the proportions keep changing (compare the top-10 against the the top-40).

As a consequence, recruiters examining these lists will see different proportions depending on the point at which they decide to stop. This can cause under represented groups have not a fair outcome, so limiting the visibility.

4.1.2 The FA*IR algorithm

What is the fairness criterion applied by FA*IR?

A *prefix* of a list are the first elements of the list; for instance, the list *(A, B, C)* has prefixes *(A)*, *(A, B)*, and *(A, B, C)*.

The fairness criterion in FA*IR [Zehlike et al. 2017] requires that the number of protected elements *in every prefix* of the list corresponds to the number of protected elements we would expect if they where picked at random using Bernoulli trials (independent "coin tosses") with success probability p.

This correspondence is not exact, and there is a parameter α corresponding to the accepted probability of a Type I error, which means rejecting a fair ranking in this test. A typical value of α could be 0.1, or 10%.

Given p, α, and k, which is the total length of the list to be returned, an M-table is computed. This M-table indicates what is the minimum number of protected elements at every prefix.

Example

This example illustrates how the re-ranker works, but we will be omitting a correction on α that will be explained next.

Suppose p=*0.5*, this means that we would like a list in which the protected candidates are at least 50% of every prefix. Suppose α=*0.1*, meaning we accept a 10% of Type I error.

The M-table in this case is:

Position	1	2	3	4	5	6	7	8	9	10
M	0	0	0	1	1	1	2	2	3	3

This means that, among the top 3 elements, even if there is no protected item, we would still consider the list to be fair, because if you toss a fair coin (p=*0.5*) 3 times, the chance of getting "heads" 3 times is above 10% (remember α=*0.1*). However, among the top 4 items, at least one of them has to be protected, because if you toss a fair coin, the chance of getting *heads* 4 times is below 10%, hence, with this α it is not believable that the coin was fair in the first place.

The rest of the M table is easy to interpret; for instance: among the top-5 elements there has to be at least 1 protected, among the top-7 there must be 2 at least, and among the top-9 there must be 3 at least.

Corrections for multiple hypotheses testing

The FA*IR plug-in does not use directly the parameter α, but computes a *corrected* α, which is in general smaller. For instance, for *p=0.5*, α*=0.1*, *k=100*, the *corrected* α*=0.0207*.

The *corrected* α accounts for the fact that, in a list of size *k*, there will be *k* tests performed, one for every prefix (for instance, 100). Hence, the probability of failing in at least one prefix is larger than α (because there are 100 attempts being made). The correction mechanism is explained in the FA*IR paper [Zehlike et al. 2017].

4.1.3 References

[Friedman 1996] Friedman, B., & Nissenbaum, H. (1996). Bias in computer systems. ACM Transactions on Information Systems (TOIS), 14(3), 330-347.

[Zehlike et al. 2017] Zehlike, M., Bonchi, F., Castillo, C., Hajian, S., Megahed, M., and Baeza-Yates, R. (2017, November). FA*IR: A fair top-k ranking algorithm. Proc. CIKM 2017 (pp. 1569-1578). ACM Press.

4.2 The Plugin

4.2.1 What the Fairsearch plugin does

People search engines, as a main example of this plugin application, are not aware of the biases the traditional algorithms for search (aka TF/IDF or BM25) might be introducing in their search results. This will reduce the visibility of already disadvantaged groups corresponding to a legally protected category such as people with disabilities, racial or ethnic minorities, or an under-represented gender in a specific industry).

This plugin uses an efficient algorithm for producing a fair ranking given a protected attribute, i.e., a ranking in which the representation of the minority group does not fall below a minimum proportion *p* at any point in the ranking, while the utility of the ranking is maintained as high as possible.

This method can be used within an anti-discrimination framework such as positive actions. This is certainly, not the only way of achieving fairness, but this plugin provide an algorithm grounded in statistical tests that enables the implementation of a positive action policy in the context of search.

4.2.2 What the plugin does NOT

This plugin uses a fairness criterion that requires a couple of input parameters, but it does impose specific parameters for that fairness (e.g., the proportion *p*). Those must be set according to the context of your application.

4.3 How to use the fair-search plugin

Now it is time to finally perform a fair re-scoring.

The usual flow for the fairsearch plug-in is this one:

- a user executes a query in the search engine, and during this process,
- indicates s/he wants to apply the fairsearch plug-in.

To achive this we are going to use a functionality provided by Elasticsearch named *re-scoring*.

4.3.1 Assumptions and preconditions for this example

Lets suppose we have already in our search engine this set of documents:

```
Doc1 { body: "hello hello hello hello hello hello hello hello hello hello", gender: "m
↪" }
Doc3 { body: "hello hello hello hello hello hello hello hello hello bye", gender: "m"␣
↪}
Doc5 { body: "hello hello hello hello hello hello hello hello bye bye", gender: "m" }
Doc7 { body: "hello hello hello hello hello hello hello bye bye bye", gender: "m" }
Doc9 { body: "hello hello hello hello hello hello bye bye bye bye", gender: "m" }
Doc2 { body: "hello hello hello hello hello bye bye bye bye bye", gender: "f" }
Doc4 { body: "hello hello hello hello bye bye bye bye bye bye", gender: "f" }
Doc6 { body: "hello hello hello bye bye bye bye bye bye bye", gender: "f" }
Doc8 { body: "hello hello bye bye bye bye bye bye bye bye", gender: "f" }
Doc10 { body: "hello bye bye bye bye bye bye bye bye bye", gender: "f" }
```

In this example, women will be our protected category. As we see in the "body" of the documents above, the word "hello" occurs more in the ones having `gender=m` (male) than in the ones having `gender=f` (female).

4.3.2 How does a search looks like

Lets first imagine we execute a normal search for "hello", one without using the Fairsearch plugin. The results would look like this:

```
GET test/_search
{
    "query": {
        "match": {
            "body": "hello"
        }
    }
}
```

This request will return all documents that match the word hello, sorted by their relevance scoring. For this particular dataset we would get this results:

Doc1, Doc3, Doc5, Doc7, Doc9, Doc2, Doc4, Doc6, Doc8, Doc10

that if we take a close look these will be:

m, m, m, m, m, f, f, f, f, f

with all men as first top results, however as we could see in the Motivation section, there are many situations where we might aim for a fair result. To achieve this we will use the plug-in.

A request with the rescore function will look like this:

```
GET test/_search
{
    "query": {
        "match": {
            "body": "hello"
        }
    },
    "rescore": {
        "fair_rescorer": {
            "protected_key": "gender",
```

(continues on next page)

(continued from previous page)

```
            "protected_value": "f",
            "significance_level": 0.1,
            "min_proportion_protected": 0.6
        }
    }
}
```

this request is actually doing an Elasticsearch *match* query, could it by any other type of query, for example a *bool* or a *multi match*. then after the results are calculated (in every shard) it apply the fair topK algorithm.

This request will give you a response where the target number of protected elements will be scored in relevant places, that for our example will be:

Doc1, Doc3, Doc2, Doc5, Doc4, Doc7, Doc9, Doc6, Doc8, Doc10

in terms of gender:

m, m, f, m, f, m, m, f, f, f

with a much fair distribution of elements of the protected class (i.e., some women appear in the top positions).

4.4 Advance usage

4.4.1 Building M tables

The M tables are a core component of this plugin. They indicate at position *M[i]* the minimum number of protected elements that must be present among the top *i* elements to consider the ranking was fair.

In the plugin we operationalize this process by creating them inside elasticsearch as documents in their own internal store, otherwise the process of calculating them on every request would it be very costly.

Create a new M table

To create a new M table you can issue the next command:

POST _fs/_mtable/{proportion}/{alpha}/{k}

where the parameters are:

- proportion: The proportion of protected elements.
- alpha: The significance parameter (α) corresponding to the probability of rejecting a fair ranking.
- k: The expected size of returned documents in the search.

for example:

POST /_fs/_mtable/0.5/0.1/5

```
{
    "_index": ".fs_store",
    "_type": "store",
    "_id": "name(0.5,0.1,5)",
    "_version": 1,
    "result": "created",
    "forced_refresh": true,
    "_shards": {
```

(continues on next page)

(continued from previous page)

```
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 0,
    "_primary_term": 1
}
```

this will store a document in elasticsearch that will look like:

```
{
  "_index": ".fs_store",
  "_type": "store",
  "_id": "name(0.5,0.1,5)",
  "_score": 1,
  "_source": {
    "type": "mtable",
    "proportion": 0.5,
    "alpha": 0.1,
    "k": 5,
    "mtable": [
      0,
      0,
      0,
      0,
      1,
      1
    ]
  }
}
```

List all stored M tables

To list all stored M tables you can use this command:

```
GET _fs/_mtable
```

this will give you an answer like:

```
{
    "took": 7,
    "timed_out": false,
    "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
},
"hits": {
    "total": 3,
    "max_score": 1,
    "hits": [
    {
        "_index": ".fs_store",
        "_type": "store",
        "_id": "name(0.5,0.1,5)",
```

(continues on next page)

(continued from previous page)

```
        "_score": 1,
        "_source": {
            "type": "mtable",
            "proportion": 0.5,
            "alpha": 0.1,
            "k": 5,
            "mtable": [
             0,
             0,
             0,
             0,
             1,
             1
             ]
        }
    },
....
    ]
    }
}
```

Delete stored M tables

Currently there is no functionality offered to delete an specific mtable, you should probably also never do that yourself. However if you want to delete documents, use the standard document api from elastic and refer to the specific table document id.

4.5 Contact

The plugin and guide was built by the search and data consultant Pere Urbon in partnership with researchers from *Technische Universität Berlin* (Meike Zehlike and Tom Sühr) and *Universitat Pompeu Fabra* (Carlos Castillo).

This development was supported by a grant from the Data Transparency Lab.

The FA*IR algorithm was introduced in: Zehlike, M., Bonchi, F., Castillo, C., Hajian, S., Megahed, M., and Baeza-Yates, R. (2017, November). FA*IR: A fair top-k ranking algorithm. Proc. CIKM 2017 (pp. 1569-1578). ACM Press.

If you have any questions or feedback, please contact Pere Urbon **or** create an issue.

CHAPTER 5

Indices and tables

- genindex
- search