Related Searches Recommendation

(query-4-query recommendation)

Concept:

Query-4-Query recommendation is done on the searchhub signals data using the ***Matrix decomposition based Query-Query similarity***. Consider the analogy of user-movie recommendation system to understand the query-query recommendation. In case of the user-movie recommendation, we have a weight matrix which represents the rating each user have given to the movie. We then use matrix decomposition techniques like ALS Recommenders to get the similarities between users and movies independently. This decomposition enables us to recommend movies to a user.

In our application **Users** correspond to **Queries** and **Movies** correspond to **Document Clicked.**

However, we are actually interested in getting the queries that are highly related (similar not just textually) to each other and recommend these queries to the user based on his first query.

To draw the analogy we want to show **similar users** to the **current user**. In the *Matrix decomposition based Query-Query similarity* we specify how we want the weight matrix to be initialized and the collections where we want the **query-query** similarities to be stored.

This job is computed offline and the similarities are stored in the collections which makes things fast at runtime.

Common Question:

A common question that would come to mind is that why do we even need this computationally heavy matrix decomposition job when we could just use the autocorrect logic and show highly similar queries!

The answer is: Yes, we could do that but those recommendations would be based only on the textual similarity. Matrix decomposition method allows us to capture more complex query similarity relations like: **Qyery_1** -> **Doc1** clicked. **Query_2** -> **Doc1** clicked. Query_1 and Query_2 are similar in the sense that they led to the same document being clicked. The number of times such associations are encountered in the signals data determines the value of the weight in our matrix.

Working principle:

Query-recommendation is built on the queries we have already seen. If there is a completely new query we will not have it in the `query-similarities` and we won't be able to give recommendations. Another very common case I could think of is typo in the query we have already seen. For e.g. `physical operators` is a query we have recommendations for but the issued query is `pysiscal oprators`. If we directly search for recommendations we won't be able to get any.

To make things robust create auxiliary `ngram, edge-ngram` indexes on the `query-similarities` collection's `itemId` field.

When a search query **`pysiscal oprators`** is issued, we first use the autocorrect logic and find the best match for this query from the queries we have already seen. Get the top result, which will be **"physical operators"**, and then give the recommendation for this top query match from the 'query-similarities' collection. (We already have computed the recommendations and stored using the query-query similarity UI job in fusion 3.1.0)

Conclusion:

You can watch the video to get a feel of how this thing works as at this time we do not have sufficient data to give good quality recommendations.
https://drive.google.com/open?id=0B6eJKDmiUr-MRWprQlhjLVo0ajA

The sample_ui_query_recs.html makes api calls to fusion pipelines and gets the results back. It can be used as a reference to make any further changes in any UI of your choice.

Execution Instructions:

1. Delete the *version* column from the shubsignals dataset
2. Start fusion 3.1.0

```
$./fusion start
```

and index signals into Fusion 3.1.0 using the standard procedure.

3. Run

```
$python bootstrap.py --setup_query_recommendations
```

4. Start spark and set appropriate memory limits:

```
$curl -H 'Content-type:application/json' -X PUT -d '1'
http://localhost:8764/api/apollo/configurations/fusion.spark
.executor.memory.fraction

$curl -u admin:password123 -X PUT -H 'Content-type:
application/json' -d '10g'
'http://localhost:8764/api/apollo/configurations/fusion
.spark.worker.memory'

$curl -u admin:password123 -X PUT -H 'Content-type:
application/json' -d '10g'
```

'http://localhost:8764/api/apollo/configurations/fusion.spark.executor.memory'

5. `$./spark-master start`
6. `$./spark-worker start`
7. From the Fusion 3.1.0 start the Spark Job id “query-recommendation”
8. From the Fusion 3.1.0 UI crawl the datasource “query-recommendation-matching-indexer”

Some sanity checks: Verify if your spark master/worker are running and the collections are created before running the spark job and crawling the datasource.

9. Open the `sample_ui_query_recs.html` and enjoy!!