Realistic Search Engine Query Log Generation

Amirhossein Aleyasen, Chen Zhang

{aleyase2,czhang49}@illinois.edu

This project is on software track and we developed GenQL an open-source package that is available publicly on https://github.com/Aleyasen/GenQL . GenQL generates realistic search engine query logs using a given document corpus in large scale. In this report we provided details description of the implementation and short user manual. We also provided details user manual with examples in the Github page.

Introduction

Query logs are critical to conduct research in information retrieval area. Researchers in this area use query logs extensively to do empirical studies and prove their proposed theories in real-world applications. However, query logs from real search engines are hard to find. Companies usually do not share query logs due to privacy concerns even if they have been anonymized. As an example, AOL published a set of anonymous query logs in 2006 but deleted it after some users could be identified through associating some information with each other. In another example, Wikipedia published an anonymous query log dataset in 2012 but removed it after several days, stating that they no longer plann to publish any query log in the near future. Therefore, the only option for researchers is to find query logs that are released for specific and small systems; in which they can investigate queries carefully and remove any data that might be problematic in terms of privacy concerns.

Given these issues, in this project our goal was to generate realistic search engine query logs in large scale. We aimed to build a tool that its input is a corpus containing a list of documents for general or specific domains. The tool generates query logs based on this input corpus that follows real-world search engine query log characteristics.

Implementation Details

For implementing our algorithm, first of all we needed a simple search engine. There are great open-source search engines (e.g. Lucene) but since we wanted to modify the code for our usage we couldn't use sophisticated search engines that their code is

hard to modify. So we use a search engine code that was available on Github that provides some basic search engine functionalities for us (e.g. indexing documents, TF-IDF ranking, AND/OR queries, etc.).

The project has a Command-Line Interface (CLI) that user can specify the arguments using it. CLIRunner class parse the input arguments and run the desired process. There are two main task that user can do:

1- Generate search engine query logs

2- Evaluate the generated queries using some available human generated query logs.

Generate Query Logs

Method gen() in MainApp is responsible for this task. The inputs for this method are given in the following:

- *corpus*: Input corpus directory
- *generateQuerieFile*: Path for the output query file
- *qcount*: Number of queries that need to generate

First of all, the method indexes all the documents in the corpus directory. Then it repeats the following process *qcount* times to generate *qcount* queries:

1. Select k documents as seed.
2. Run a relevance feedback method (Roccio algorithm) to promote those k documents to top documents. This step will update the query.
3. Select top terms based on their contribution in the query and create a new query.
4. Submit the new query and go to step (2).
5. Repeat steps (2-4) for a specified number of iterations.
6. Store the top terms of the last generated query as the result.

In the relevance feedback method we use a modified version of Rocchio algorithm, we give more positive weights to relevance documents (the selected k documents) and negative weights to the top results that are not relevant. By doing this several times, we can expand the query somehow the N documents be the top results. Since Rocchio algorithms just add terms to the query, we remove the terms that has less contribution from the query. The formula for updating the query in Roccio in each step given in the following.

$$\overrightarrow{Q_m} = (a \cdot \overrightarrow{Q_o}) + \left(b \cdot \frac{1}{|D_r|} \cdot \sum_{\overrightarrow{D_j} \in D_r} \overrightarrow{D_j}\right) - \left(c \cdot \frac{1}{|D_{nr}|} \cdot \sum_{\overrightarrow{D_k} \in D_{nr}} \overrightarrow{D_k}\right)$$

That first part is query from last iteration, the second part is relevant document (the k documents) and the last part is non-relevant document (top documents in the result that are not among those k documents).

To specify query length, use can specify minimum and maximum query length and the implementation select query length randomly between those minimum and maximum values.

Evaluate The Generated Queries

Method eval() in MainApp is responsible for this task. The inputs for this method are given in the following:

- *corpus*: Input corpus directory
- *generateQuerieFile*: Path for the generated query file
- groundTruthQueriesFile: Path for the ground truth query file.

In eval() method, we first create an Index on corpus directory. Then we follow these steps to do evaluation:

1. Submit all the queries in the ground truth query file.
2. Store the ranked results for each of the queries in a temp directory.
3. Select top (default 10) results for each query.
4. Select the top results from step (3) as seed documents.
5. Run a relevance feedback method to promote those k documents to top documents.
6. Select top terms based on their contribution in the query and create a new query.
7. Submit the new query and go to step (5).
8. Repeat steps (5-7) for a specified number of iterations.
9. Compare the generated query and the query from ground truth.

In the relevance feedback method, we used the same method as the generate query log section. For evaluation, we use DBLP abstracts dataset (given in homework 5) as the corpus. Each abstract will be a document. We ask two graduate students to think about their questions in their research and come up with some queries about them. We use these queries as ground truth. Some of these queries and the generated queries using GenQL are given in Table 1.

Ground truth query	**Query generated by GenQL**

data integration for semi-structured data	data preparation parallel stream
network comparison for evolving network over time	network congestion conference coding controller
finding shortest path between nodes in very large graphs	graphs shortest path
named entity resolution for hierarchical entities	entities entity resolution named
big data applications in health care	health emergency preparation

Tutorial

In this section, we will provide details about Command-Line Interface (CLI) using with some examples. In the *bin* directory you can find *genql.sh* for Linux machines (and *genql.bat* for Window machines). To see all the available switches type:

```
genql -h
```

Output is given in the following that contains all the available arguments.

```
usage: genql.jar [-d <arg>] [-e] [-g] [-h] [-n <arg>][-t <arg>] [-o <arg>] [-q <arg>]

 -d,--dir <arg>        The directory for the corpus
 -e,--eval             Evaluate generated queries based on a ground-truth query log
 -g,--gen              Generate search query log
 -h,--help             Help
 -n,--count <arg>      Numbers of queries to generate
 -o,--output <arg>     Output file for the generated queries
 -q,--query <arg>      The file for the generated queries using -g option (only for evaluation)
 -t,--gtruth <arg>     The file for the ground-truth query log (only for evaluation). Each line of the file
contains a query
 -m,--min <arg>        Minimum query length
 -x,--max <arg>        Maximum query length
```

We provide some examples in the following.

Example 1: Simple query generation

```
genql -g -h 1000 -d /opt/corpus -o /opt/out/gen.out
```

Generate 1000 queries from a corpus located on */opt/corpus*. The queries will store in file */opt/out/gen.out*.

Example 2: Query generation with specific min/max query length

```
genql -g -h 1000 -d /opt/corpus -m 3 -x 7 -o /opt/out/gen2.out
```

Generate 1000 queries from a corpus located on */opt/corpus*. The query lengths will between 3 to 7 (inclusive). The queries will store in file */opt/out/gen2.out*.

Example 3: Evaluating the generated queries

```
genql -e -d /opt/corpus -q /opt/out/gen.out -t /opt/out/gtruth.txt
```

Generate queries from a corpus located on */opt/corpus* (number of generated queries is equal to number of queries in */opt/out/gtruth.txt*). Store the generated queries in /opt/out/gen.out and compare them with ground truth queries in */opt/out/gtruth.txt*