SearchTool: user guide

Ashley Williams [ashley.williams@pg.canterbury.ac.nz]

Version History

Version	Date	Author	Description
1.0	14.08.2017	A. Williams	Initial version

Contents

1 Overview

Certain blog articles hold potential value in software engineering research. However, this value is often left untapped due to the effort that is required in filtering the high quality blog articles from the vast quantity available. SearchTool is a tool that can be used to interact with Google's Custom Search API to help researchers (and anyone) filter this high quality, relevant content. It was made for finding practitioner blog articles, but with Google's Custom Search API whitelist feature, SearchTool can be used to help filter any part of the web.

1.1 How does it work?

Google's Custom Search API provides programmatic access to Google. We use this API to query Google and insert the results into a Mongo database. These results can then be exported for analysis, or for example, trend analysis can be conducted to visualise the change in results over time. We also utilise keyword indicators to optionally manipulate queries to improve the relevance of the results presented. This is explained later on in this guide.

2 Getting started

2.1 Google Custom Search API

First we need to set up an instance of Google's Custom Search API to serve as our gateway into Google.

1. Sign into https://developers.google.com/custom-search/ with your Google account
2. Go to the CSE control panel at https://cse.google.com/manage/all
3. Click the 'Add' button to add a new search engine
4. You will be prompted to add the sites that you want your search engine to search. This can be used as a whitelist of sites that you have previously found to be relevant. Later on, we can change the settings so that the whole internet is searched (although these whitelist sites will be favoured). (Step 9).
5. Choose a language for your search engine
6. Give your search engine a name
7. Click the 'Create' button to create your search engine
8. Now go back to your control panel and edit the search engine that you have just created, you will have more options such as; add a description for your search engine, edit your whitelist, add an exclusions list etc
9. Next to your 'Sites to search' list there is a combo box with 'Search only included sites' selected. If you want to search outside of your whitelist, change this to 'Search the entire web but emphasize included sites'

Going forward, you will need the search engine ID and an API key. The search engine ID can be found in the 'Details' section. The API key can be found by going to

https://developers.google.com/custom-search/json-api/v1/overview and clicking the 'Get a Key' button (you will be prompted to create a project).

2.1.1 A quick word on pricing

Just to explain upfront, the Custom Search API is free up to a certain limit. At the time of writing (August 2017), this is the statement from the CSE website:

"For CSE users, the API provides 100 search queries per day for free. If you need more, you may sign up for billing in the API Console. Additional requests cost $5 per 1000 queries, up to 10k queries per day."

If you require more than the above, Google has a paid service called Google Site Search (but you may have to make changes to the application if you want them to work together).

2.2 Prerequisites

To get started with this tool, download using the following command:

```
git clone https://github.com/zedrem/relevance_querying
```

Prior to running, you will need to have Python installed and access to a MongoDB implementation.

- Python - tested with version 3.6.1
- MongoDB - we have installed Mongo locally from MongoDB (https://www.mongodb.com/). However, you may also use a hosted solution such as Mongo Lab (https://mlab.com/)

Alternatively, you can use an online IDE like Cloud9 (https://c9.io) which comes with Python pre-installed. Instructions for setting up Mongo in Cloud9 can be found at https://community.c9.io/t/setting-up-mongodb/1717

2.2.1 Installing requirements

Once you have Python and Mongo ready to go, you need to install the required Pip packages. These can be found in the requirements.txt file. You may install them manually using:

```
pip install -r requirements.txt
```

Alternatively, run the provided MAKEFILE.

2.2.2 Updating the config file

Next, edit config.json, located in the config directory. Add the path to your database, your desired database name, search engine ID, and API key.

```
{
        "db_url": "mongodb://127.0.0.1/",
        "db_client": "relevance-querying",
        "search_engine_id": " search engine id here ",
        "api_key": " api key here "
}
```

2.3 First time use

Finally, run the setup file by navigating to the root directory in your terminal and using the command below:

```
python ./setup.py
```

This setup file tests the connection to your database, and then loads the example indicators and default config settings. More information on what these settings are can be found in the applications user guide (this document, see below), which is stored in the /docs directory.

The application is now setup and ready to use.

3 Running the app

3.1 Via the command line interface menu

To run the CLI menu, simply run the below command and follow the instructions in your terminal window:

```
python ./searchtool.py
```

When running SearchTool from the menu, you first have to run the query and then separately export the results to a CSV. Of course, you can also manipulate the results directly through the database.

3.2 Via the use of arguments

To just run a query, without having to navigate the menu, you can run the same script, only with providing arguments separated by a space:

```
python ./searchtool.py query="Hello world" indicators=TOP,BLG,ARG,EXP
start_date="02/06/2016" end_date="02/06/2017" number_of_results=100
```

The ‘query’ argument is required, while the other four arguments are optional. Not providing optional arguments will result in the default values being used. To find out what the optional arguments are for, see ‘Keyword indicators,’ ‘Querying specific timeframes’ and ‘Number of results’ in the Functionality and settings section of the documentation.

When running a query this way, the results are automatically exported once the query is complete. You can also access the results directly though the database.

4 Running a query

When a query is run, through either the menu or programmatically, the application converts the given query into a series of API calls depending on the settings. One query will usally result in multiple API calls being made (see ‘Single/Multi Query Mode’ and ‘Number of results’). The results of the API calls are stored in both, the query_results and query_archive collections in the database. The difference between these two collections is that query_results is dropped between each query and therefore only shows the results of the previous query. The query_archive collection is not dropped and continuously added to. This collection can be useful for analysing trends over time.

5 Viewing/Changing your settings

5.1 Point the application at a new database

Update /config/config.json with the new settings and run the setup.py file. The new database will be populated with the default config and will be used next time the application is started.

5.2 General settings

To change any other setting, you can open the menu and follow the instructions (see running the app), or you can import the settings.py library and run the commands outlined in this section.

First, to import the settings; in your terminal, navigate to the root directory of the project and open the Python terminal (by typing ‘python’ and hitting enter). Now run the following command:

```
import searchtool.settings as sets
```

You can now run the below functions, as long as you prefix them with ‘sets.’

Command	Description
sets.getDB()	returns the database object
sets.setAPIKey(key)	updates the API key in the database with the value of key
sets.getAPIKey()	returns the API key that is stored in the database
sets.setSearchEngineID(id)	updates the search engine ID in the database with the value of id
sets.getSearchEngineID()	returns the search engine ID that is stored in the database
sets.setQueryMode(mode)	updates the query mode in the database with the value of mode. Should be 'multi' or 'single'.
sets.getQueryMode()	returns the current query mode
sets.setNumberOfRuns(num)	updates the number of runs in the database with the value of num. This setting is for handling stochastic results, see ‘Repeating queries to handle stochastic results’ in functionality and settings.
sets.getNumberOfRuns()	returns the current set number of runs

5.3 Manually exporting to a CSV file

To manually export the query_results or query_archive collections, there are two options; through the menu, or programmatically. Exporting through the menu is the easiest solution, run the menu and simply select the options to export either collection. To export programmatically, open Python and import the settings as is shown in the ‘General settings’ section. Then run the following commands:

```
# to export the results collection
sets.exportResultsCSV()

# to export the archive collection
sets.exportArchiveCSV()
```

Once exported, the script will prompt you and provide a path. The CSV files will be located in the newly created ‘Output’ directory and stored as either ‘_results.csv’ or ‘_archive.csv,’ each prefixed with a timestamp.

6 Functionality and settings

6.1 Keyword indicators

Keyword indicators are a series of keywords that can be added to a query to improve the retrieval of relevant results. The example indicators that we provide are to aid the retrieval of practitioner blog articles that contain some reasoning based on personal experience. These indicators are broken up into four types; blog indicators that help us to find blog articles, topic indicators that help us to find articles about a certain topic, reasoning indicators that help us to find articles that provide some reasoning about a topic that we care about, and experience indicators that help us find articles that are supported by some personal experience.

6.1.1 How to specify indicators in a query

To specify which indicators you wish to include with a query, use the indicators argument with the indicator codes separated by a comma, as shown below.

```
python ./searchtool.py query="Hello world" indicators=TOP,BLG,ARG,EXP
```

The same comma separated format should be used when using the menu interface.

6.1.2 How to update the indicators

To update the indicators, edit the indicator files located at /config/indicators. Each indicator (which can be a phrase) should be stored on a line. Once you have saved the indicator text files, import the settings script and run the following commands:

```
# first import the settings
import searchtool.settings as sets

# first get the database object
db = sets.getDB()

collectionName = " name of the collection "
code = " the code of the collection "
fileName = " the path to the indicator file "

# load the indicators
sets.loadIndicators(db, collectionName, code, fileName)
```

Alternatively, you can update the indicators through the menu. However, there is no option in the menu to add new indicator types so you will need to repurpose the existing codes or manually interact with the database.

6.2 Repeating queries to handle stochastic results

Stochastic results (i.e getting different results from running the same query) may be caused by factors such as search engine optimisation, the user's history and the location of the data. We are unsure whether querying an API gives these stochastic results in the same way that the Google web search does. For this reason we have introduced a setting called ‘number_of_runs.’ This variable is an integer that determines how many times the entire query will be run. The results can then be merged or compared either in the database collection or in the exported CSV. Managing this setting is simple:

```
import searchtool.settings as sets

# get the current setting
number_of_runs = sets.getNumberOfRuns()

# update the setting
sets.setNumberOfRuns(3)
```

6.3 Single/Multi Query Mode

SearchTool has two types of query modes; single query mode and multi query mode.

6.3.1 Single Query Mode

In single query mode, the query string is combined with all of the indicators into a single query. At least one of each indicator must be present in the query along with the query string. The query is structured as follows:

```
query_string AND (indicator1_type1 OR indicator2_type1) AND (indicator1_type2 OR
indicator2_type2 OR indicator3_type2) AND etc.
```

Although a single query means less API calls, single query mode may not work on large query strings, or when working with a large number of indicators due to the maximum query length being exceeded. In these instances, multi query mode can be used.

6.3.2 Multi Query Mode

In multi query mode, multiple queries are generated to handle all variations of indicators. This list of generated queries is run against the API and the results can then be merged and compared in the database or exported CSV. For example:

```
generated_query_list = ["query_string AND ind1", "query_string AND ind2", "query_string
AND ind1 AND ind2"]
```

In multi query mode, each query is typically shorter than single query mode. However, it generates a lot more API calls overall.

This setting is managed through the settings script:

```
import searchtool.settings as sets

# get the current setting
query_mode = sets.getQueryMode()

# update the setting
sets.setQueryMode('multi') # or 'single'
```

6.4 Querying specific timeframes

Specific start and end dates can be set to only retrieve results within these designated timeframes. These are optional parameters that can be set when initiating the query (or when prompted through the menu). The dates must be in the format ‘dd/mm/yyyy’ to be valid. You may choose whether to provide just the start date (to get results that were published after that date), just the end date (to get results that were published before that date), both dates (to get results that were published between those dates), or none at all (to get all results). Any unspecified dated will be set to ‘off’ as default. Set the date parameters as shown below:

```
python ./searchtool.py query="Hello world" start_date="01/01/2007" end_date="01/05/2017"
```

6.5 Number of results

The number of results you wish to receive can also be specified when calling the query. This is an optional parameter whose default is to return 50 results. The number of results specified has to be a multiple of ten to be valid. The API only provides ten results at a time, so retrieving 100 results will lead to 10 API calls being made. The ‘start’ value in the results indicates which results came from which of these calls. Set the number of results parameter as shown below:

```
python ./searchtool.py query="Hello world" number_of_results=100
```

7 Resources

7.1 API guide

If you wish to extend the functionality of SearchTool, this guide shows the API options available:
https://developers.google.com/resources/api-libraries/documentation/customsearch/v1/python/latest/customsearch_v1.cse.html

7.2 API Quota

To check your API usage and quota limits, go to this URL and log in. Select the project that you have created for SearchTool, click ‘Custom Search API’ and navigate to the ‘Quotas’ tab: https://console.developers.google.com/apis/dashboard

8 License

This application has been released under the MIT license. More information can be found in the /LICENSE file.

9 Acknowledgements

Thanks to Adrien Aucher for his work in developing the early prototype of this tool.

10 Questions and problems?

If you have any questions, please contact the projects author via the issues section on GitHub, or by emailing: ashley.williams@pg.canterbury.ac.nz