coast_search Documentation

Release 0+untagged.47.g6216b3b.dirty

Ashley Williams

Dec 24, 2018

Contents

Contents:

CHAPTER 1

COAST_SEARCH

COAST_SEARCH allows you to generate and run queries against the Google Custom Search API.

- Free software: MIT license
- Documentation: https://coast-search.readthedocs.io.

1.1 Features

With COAST_SEARCH you can:

- Generate and run queries based on the multi-dimensional search logic outlined by Rainer and Williams in IST'18
- Define your own queries to run

1.2 Prerequisites

The tool is built in Python 3 and tested in versions 3.5 and 3.6.

You are required to set up and provide your own API keys and Search Engine ID's for searching against the Google Custom Search API. Details on how to do so can be found online here

1.3 Installation

To install COAST_SEARCH, run this command in your terminal:

```
$ pip install coast_search
```

This is the preferred method to install COAST_SEARCH, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

To install from source, visit our documentation.

CHAPTER 2

Installation

2.1 Stable release

To install COAST_SEARCH, run this command in your terminal:

```
$ pip install coast_search
```

This is the preferred method to install COAST_SEARCH, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From source

The source for COAST_SEARCH can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/zedrem/coast_search
```

Or download the tarball:

```
$ curl  -OL https://github.com/zedrem/coast_search/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Note: you may need to ensure that your setuptools version is greater than 12. You can upgrade with the following:

```
$ pip install --upgrade setuptools
```

CHAPTER 3

Modules

3.1 Query Generator

3.1.1 Introduction

The query generator module contains functions that can be used to generate a set of queries that follow the heuristics proposed by Rainer and Williams in IST'18

3.1.2 Usage

To use the query_generator module:

```
>>> import coast_search
>>> coast_search.query_generator.function(to_use)
```

or: .. code-block:: console

```
>>> from coast_search import query_generator
>>> query_generator.function(to_use)
```

3.1.3 Functions

Contains the functions required for generating the multiple queries from the config, given n number of dimensions and any constraints

`coast_search.query_generator.`**`add_api_config_to_queries`**(*generated_query_strings*, *search_engines*)

Merges the two parameters and returns a list of dicts that include the api config.

If only 1 API key is provided, it is assumed this is valid for many searches and is used for all queries If more than 1 is provided, then the number of keys provided needs to match the number of queries

Args:

generated_query_strings: The output from the generate_query_strings function.

search_engines: The search engines list that is found in the api_config file. See the documentation for usage guidelines (http://coast_search.readthedocs.io/).

Returns: result_list: Updated list of query data now including search engine/api info

`coast_search.query_generator.`**`add_to_result_list`**`(`*result_list*, *seg_id*, *logic*, *query*`)`

`coast_search.query_generator.`**`check_length`**`(`*seed*, *random*, *query_words*, *key_max*`)`

Google limits searches to 32 words, so we need to make sure we won't be generating anything longer Need to consider - number of words in seed - number of words in random phrase - number of words in the lists from the query Will raise exception if there are too many words

Args: seed: the seed for segment 1 random: the random query string for segment 0 query_words: object with key=name of dimension, value=list of keywords to use in query key_max: the maximum number of words (32 in Google's case)

Returns: bool: True for correct number of words, False for too many

`coast_search.query_generator.`**`generate_query_strings_n_dimensions`**`(`*dimensions_dict*, *seed='software'*, *key_max=32*`)`

Given dimensions and associated words, the seg1 seed and the max length of query, sets up and generates the query strings dynamically, depending on the number of dimensions.

Args: dimensions_dict: dictionary containing the dimensions data. key=name, value=list of words seed: seg1 seed key_max: the maximum number of words (32 in Google's case)

Returns: result_data: an object containing data about each of the segments (id, logic, query) Returns None if check_length returns False

`coast_search.query_generator.`**`generate_result_list`**`(`*dimensions_data*, *dimensions*, *seed*, *random*`)`

Given the dimensions information, dynamically generates logic & query for each segment.

Note: segment 0 will always contain the random query, and segment 1 will always contain the seed.

Args:

dimensions_data: a list of phrases (e.g. reasoning/experience indicators)

dimensions: a list of names of the given dimensions seed: the seed for seg 1 random: the random phrase for seg 0

Returns:

result_string: a string of all phrases AND'd together ready for a search engine.

`coast_search.query_generator.`**`get_random_query`**`(`*words_to_exclude*`)`

Segment 1 uses a random seed query. This function creates that seed query using the random_words library and returns it. Notes:

1. The random query returned wont contain any word that exists in any topic string or indicator list.
2. The random query string will always be three words long.

Args: words_to_exclude: the list of words from each of the dimensions to use as stoplist

Returns: qs: The generated random query.

`coast_search.query_generator.`**`neg_query_segment`**`(`*phrase_list*`)`

Given a list of phrases, returns a string of all the phrases negated. e.g. -"but" -"because" -"however" Args:

phrase_list: a list of phrases (e.g. reasoning/experience indicators)

Returns:

result_string: a string of all phrases AND'd together ready for a search engine.

`coast_search.query_generator.`**`pos_query_segment`**(*phrase_list*)

Given a list of phrases, returns a string of all the phrases AND'd together. e.g. ("but" AND "because" AND "however") Args:

phrase_list: a list of phrases (e.g. reasoning/experience indicators)

Returns:

result_string: a string of all phrases AND'd together ready for a search engine.

3.2 Search

3.2.1 Introduction

The search module contains functions for conducting searches against the Google Custom Search API, and then for parsing the results.

3.2.2 Usage

To use the utils module:

```
>>> import coast_search
>>> coast_search.search.function(to_use)
```

or: .. code-block:: console

```
>>> from coast_search import search
>>> search.function(to_use)
```

3.2.3 Functions

Title: search_command.py Author: Ashley Williams Description: A collection of functions that can be used for running searches. This module calls is called by init, so there is no need to import this module specifically. Refer to the documentation for details of how to use this module (http://coast_search.readthedocs.io/).

`coast_search.search.`**`deduplicate_urls`**(*json_data*)

function to create and return a list of deduplicated URLS Args:

json_data: json data result from queries

Returns: a list of deduplicated urls. If there is duplication across segments, also returns a warning

`coast_search.search.`**`extract_search_results_from_JSON`**(*json_data*)

Given the json output of the search queries, extracts the results(i.e. the URLS, titles from the search results) Args:

json_data: the json output result from the searches

Returns: json obj of the relevant extracted data

`coast_search.search.`**`get_object_to_write`**(*result*)

Returns the constructed object containing the search results data, for later analysis. Args:

> result: The output from running the query

Returns: object_to_write: the constructed object containing desired search result data

`coast_search.search.`**`queryAPI`**(*query*, *number_of_results*, *api_key*, *search_engine_id*, *segment_id*)

Query the API, return the results as a list of JSON objects. Refer to the documentation for usage guidelines and descriptions of what each parameter means (http://coast_search.readthedocs.io/). Args:

> query: The query string to run. number_of_results: The number of results you wish to be returned.
>
> > Note, the free version of the Custom Search API is limited to 100 searches per day. Each search returns 10 results.
>
> api_key: The api key of the search engine, provided by Google. search_engine_id: The id of the Custom Search Engine provided by
>
> > Google.
>
> segment_id: The segment which the results belong to.

Returns: results_list: The results from Google as a list of JSON objects

Err: In the event of an error, the error is printed to the stdout.

`coast_search.search.`**`run_all_queries`**(*query_dict_list*, *number_of_runs*, *number_of_results*, *day*, *search_backup_dir*)

Given a list of queries and configuration parameters, calls the method run_query for each query object in the given list. Args:

> query_dict_list: list of query data for all of the queries wanting to be searched number_of_runs: number of desired runs (from config file) number_of_results: number of desired results (from config file) day: Day number in search process (number of days since start date) search_backup_dir: location to store file output of searches

Returns: object containing results of all of the queries

`coast_search.search.`**`run_daily_search`**(*config_file*, *write_to_file_flag*)

Run a full daily search. This function can be set up as a cronjob (or scheduled task on Windows) to search over consecutive days. Refer to the documentation for usage guidelines and descriptions of how the config file should be structured (http://coast_search.readthedocs.io/). Args:

> **config_file: Path to a JSON file containing all relevant information for** conducting the searches.
>
> write_to_file_flag: boolean flag for writing to file

Returns: results from the search

`coast_search.search.`**`run_query`**(*query_string*, *number_of_runs*, *number_of_results*, *api_key*, *search_engine_id*, *segment_id*, *day*, *backup_dir*)

Runs the query against the Google Custom Search API. Writes the results to file and appends them to the extracted results list. Refer to the documentation for usage guidelines and descriptions of what each parameter means (http://coast_search.readthedocs.io/). Args:

> query_string: The query string to run. number_of_runs: The number of runs you wish to be repeat for each

day. Note, the free version of the Custom Search API is limited to 100 searches per day. Each search returns 10 results.

number_of_results: The number of results you wish to be returned. Note, the free version of the Custom Search API is limited to 100 searches per day. Each search returns 10 results.

api_key: The api key of the search engine, provided by Google. search_engine_id: The id of the Custom Search Engine provided by

Google.

segment_id: The segment which the results belong to. day: The day of the search period that the result has originated

from.

backup_dir: A directory that can be used for storing results as files.

Returns: extracted_results: list of results

`coast_search.search.`**`write_to_file`**(*name*, *result*, *directory*, *extension*)

Writes to results to a file Args:

name: desired filename result: directory: A directory that can be used for storing results

as files.

extension: the desired file extension, e.g: json or txt result: The output from running the query.

3.3 Utils

3.3.1 Introduction

The utils module contains some helper functions that are used by other modules. However, they are open to be utilised as you wish.

3.3.2 Usage

To use the utils module:

```
>>> import coast_search
>>> coast_search.utils.function(to_use)
```

or: .. code-block:: console

```
>>> from coast_search import utils
>>> utils.function(to_use)
```

3.3.3 Functions

Title: utils.py Author: Ashley Williams Description: A collection of generic utility functions that are used throughout coast by various modules relating to reading and writing to files.

`coast_search.utils.`**`get_from_file`**(*filename*)

Reads a file and returns each line as a list of strings.

Notes:

1. All double quotes are replaced with single quotes.
2. New line (

) characters are removed.

Args: filename: The path to the file you wish to read.

Returns: res: A list of strings, where each string is a line in the file.

`coast_search.utils.`**`get_from_file_list`**(*file_list*)

Given a list of file names, reads from each of these and returns a dictionary with filename: list of words :param file_list: :return:

`coast_search.utils.`**`get_json_from_file`**(*filename*)

Reads a JSON file and returns as an object.

Args: filename: The path to the JSON file you wish to read.

Returns: res: A JSON object, generated from the contents of the file.

Err: In the event of an error, the error is printed to the stdout.

`coast_search.utils.`**`number_of_days_past_start_date`**(*config*)

Gets the day number past the start date defined in the config (eg if the start date was a monday and today was a wednedsay the day would be 3 :param config: config file configs start_date property :return: number of days past start date

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/zedrem/coast_search/issues.

If you are reporting a bug, please include:

- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

coast_search could always use more documentation, whether as part of the official coast_search docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/zedrem/coast_search/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *coast_search* for local development.

1. Fork the *coast_search* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/coast_search.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv coast_search
   $ cd coast_search/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 coast_search tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5 and 3.6. Check https://travis-ci.org/zedrem/coast_search/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Maintainer

- Ashley Williams <ashley.williams@pg.canterbury.ac.nz>

5.2 Contributors

- Liz Richardson
- Stefan Hall
- Adrien Aucher

Want to contribute? See: CONTRIBUTING.rst

5.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 6

History

Date	Status
April 2016	Research on credibility begins, some initial scripts are written as part of various studies
First half of 2017	Adrien Aucher joins UC as an intern and works with Ashley Williams on the first version of this tool. It is only used internally at this point.
November 2018	Liz Richardson and Stefan Hall join the project to work on the first release.
December 2018	Version 1.0.0 released.

CHAPTER 7

Indices and tables

- genindex
- modindex
- *Introduction*

Python Module Index

Index