coast Documentation

Release 0+untagged.65.g29cd343.dirty

Ashley Williams

Apr 25, 2018

Contents

1 COAST **3**
- 1.1 Features . . . 3

2 Installation **5**

3 Usage **7**
- 3.1 In the terminal . . . 7
- 3.2 Modules . . . 7
  - 3.2.1 The Search module . . . 7
  - 3.2.2 The Extraction module . . . 11
  - 3.2.3 The Markers module . . . 11
  - 3.2.4 The Citations module . . . 11
  - 3.2.5 The Clarity of Writing module . . . 11

4 Contributing **13**
- 4.1 Types of Contributions . . . 13
  - 4.1.1 Report Bugs . . . 13
  - 4.1.2 Fix Bugs . . . 13
  - 4.1.3 Implement Features . . . 13
  - 4.1.4 Write Documentation . . . 13
  - 4.1.5 Submit Feedback . . . 14
- 4.2 Get Started! . . . 14
- 4.3 Pull Request Guidelines . . . 14

5 Credits **17**
- 5.1 Maintainer . . . 17
- 5.2 Contributors . . . 17

6 History **19**

7 Indices and tables **21**

Contents:

CHAPTER 1

COAST

A search tool that can be used to search for and analyse credible online articles.

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

1.1 Features

- Search for online articles, using specific reasoning and experience markers to give an indication of its credibility.
- Scrape the content of your search results ready to analyse.
- Apply a series of credibility measures to your extracted content.
- Report the results and amalgamate the credibility measures into a overall credibility rating.

CHAPTER 2

Installation

At the command line:

```
$ pip install coast
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv coast
$ pip install coast
```

CHAPTER 3

Usage

To use coast in a project:

```
import coast
```

3.1 In the terminal

Currently under development

3.2 Modules

3.2.1 The Search module

How we conduct searching

To find the best or better online articles, we try to determine online articles that are relevant and rigorous. For relevance, we rely on the topic model used by Google. For rigour, we further distinguish between reasoning and experience.

Relevance and rigour are inherently challenging to assess for research. Such challenges increase when using a search engine to find appropriate online content. Consider, for example, that a search engine:

- typically uses a keyword-based search query, and such queries do not necessarily allow for finer-grained searching;
- is likely to optimise the search results to the searcher, based on the search engine's history of prior searches by that searcher;
- is likely to maintain its own topic model to determine relevance of results e.g. whether an online article relates to software testing

COAST uses the Google Custom Search API[1] to automate the Google searches. Google, like other online search engines, uses a keyword based search. We therefore need to implement relevance and rigour in terms of keywords.

For the criterion of relevance, COAST uses a simple keyword search (e.g. the presence of the words <"software" AND "testing"> in an online document) and to allow the Google search engine's internal topic model(s) to find relevant content based on that simple keyword search.

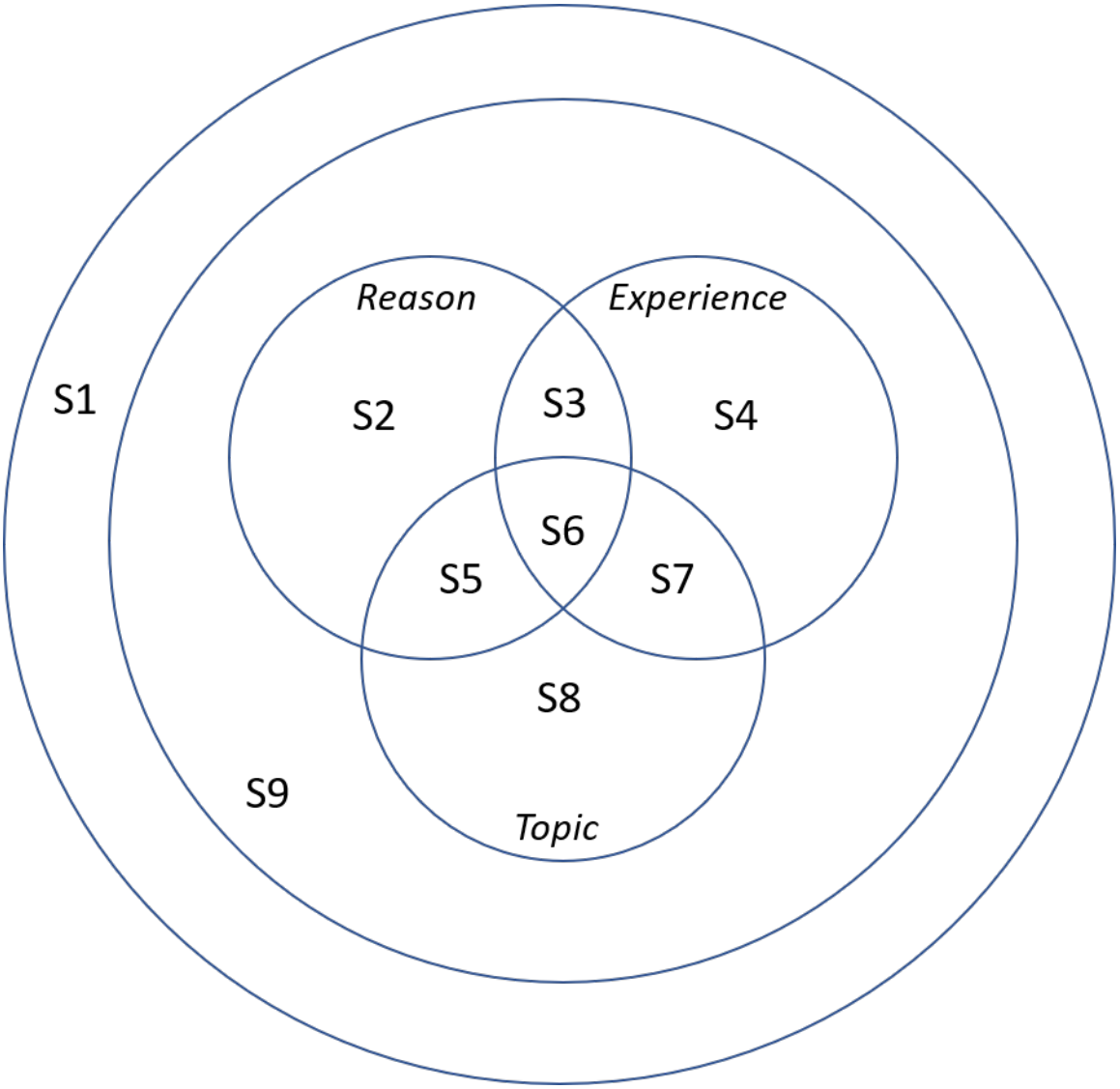
For the criterion of rigour, and specifically reasoning and experience, COAST uses a set of keywords to seek to limit searches based on reasoning and, separately, experience. We provide sample reasoning and experience indicators at https://github.com/zedrem/coast/sample_data.

COAST uses a logic of nine distinct queries, summarised in the table below and visualised in the figure (see also the example below for one of the actual search strings).

Table: Logic for each set of searches and resulting datasets (T=topic; R=reasoning; E=Experience; !=logical not)

Search Set	T	R	E	!T	!R	!E
1				x	x	x
2		x		x		x
3		x	x	x		
4			x	x	x	
5	x	x				x
6	x	x	x			
7	x		x		x	
8	x				x	x
9				o	x	x

[1] https://developers.google.com/custom-search/



Example: A search query for one of our search segments (S1: !(T + R + E))

```
todays_random_query + ' -"software" -"testing" -"i" -"me" -"we" -"us" -"my" -
↪"experience" -"experiences" -"experienced" -"our" -"but" -"because" -"for example" -
↪"due to" -"first of all" -"however" -"as a result" -"since" -"reason" -"therefore"'
```

To clarify, search set S1 would generate dataset S1. Ideally, we want the search engine to find online content that contains reasoning and experience relating to software testing. The search query for set S6 targets that ideal content. We conduct the other eight sets of searches to allow us to evaluate the quality of content in S6, and to evaluate the frequency of URL citations to research. For example, search S3 is intended to find online content that contains reasoning and experience, but where the content is not about software testing.

Setting up the search engines

In order to use COAST, you will need to set up 9 instances of Googles Custom Search API. Instructions on how to do this can be found at: https://developers.google.com/custom-search/json-api/v1/overview.

Once you have set up your nine search engines, you will need to create and record the API keys and search engine IDs in a JSON file. This must use the following format:

::

```
{
    "search_engines": [
        { "segment_id": 1, "name": "cse-1", "api_key": "api-key1", "search_engine_id":
            "search_engine_id1"
        }, {
            "segment_id": 2, "name": "cse-2", "api_key": "api-key2", "search_engine_id":
            "search_engine_id2"
    ]
}
```

Setting up the database

COAST also requires MongoDB to record all results. Instructions on how to set up MongoDB can be found at: https://docs.mongodb.com/manual/installation/.

Test that MongoDB is installed correctly and working before continuing.

Conducting the searches

The easiest way to use the search module is by using the *run_coast_daily_search* function. This can be wrapped up into simple Python script and ran as a cronjob (or scheduled task on Windows).

:: from coast import search

```
config_file_path = "./path/to/config/file.json"

search.run_coast_daily_search(config_file_path)
```

This function takes in a config file as a parameter, so first we need to create the config file. The config file is a JSON file that should contain the following keys:

- *start_date* - The date when the search period began (the first day you ran the search). This should be in the following format 'dd-mm-yyyy'
- *topic_file* - The path to a text file which contains all of the topic keywords & phrases that you wish to search on.
- *reasoning_file* - The path to a text file which contains all of the reasoning markers that you wish to use (We have provided a sample at: https://github.com/zedrem/coast/sample_data).
- *experience_file* - The path to a text file which contains all of the experience markers that you wish to use (We have provided a sample at: https://github.com/zedrem/coast/sample_data).
- *api_details_file* - The path to your API config file that you created in the 'Setting up the search engines' section.
- *db_url* - The URL to your MongoDB instance (e.g. http://localhost:27017).
- *db_client* - The name of the database instance to use (e.g. 'coast_test').
- *number_of_runs* - The number of times that each query will be run.
- *number_of_results* - The number of results to be returned by each search.
- *search_backup_dir* - The path to a directory where you will store a copy of the output from each query. This means that the data isn't lost if there is ever a problem writing to the database.

NOTE: The Google Custom Search API is limited to 100 free searches per day. So you have to keep this in mind when deciding your number_of_runs and number_of_results (each search returns 10 results). For example, one possible solution could be to use 10 runs of 100 results (10 pages); 10 * 10 = 100 total searches.

The resulting config file should look like as follows:

::

{ “start_date”: “20-04-2018”, “topic_file”: “./path/to/topic_file.txt”, “reasoning_file”: “./path/to/reasoning_file.txt”, “experience_file”: “./path/to/experience_file.txt”, “api_details_file”: “./api_config_test.json”, “db_url”: “mongodb://localhost:27017”, “db_client”: “coast_test”, “number_of_runs”: 3, “number_of_results”: 30, “search_backup_dir”: “./path/to/backup/dir/”

}

The full list of functions available is given below.

Function list

References

3.2.2 The Extraction module

3.2.3 The Markers module

Currently under development

3.2.4 The Citations module

Currently under development

3.2.5 The Clarity of Writing module

Currently under development

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/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 could always use more documentation, whether as part of the official coast 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/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* for local development.

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

```
$ git clone git@github.com:your_name_here/coast.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
$ cd coast/
$ 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 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 2.7, 3.3, 3.4, 3.5 and for PyPy. Check https://travis-ci.org/zedrem/coast/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

- Yann Le Norment <yann.lenorment@canterbury.ac.nz>

Want to contribute? See: CONTRIBUTING.rst

CHAPTER 6

History

Pre-release

CHAPTER 7

Indices and tables

- genindex
- modindex
- search