coast_core Documentation

Release v1.0.0+0.g6e97d76.dirty

Ashley Williams

Nov 20, 2018

Contents

Contents:

CHAPTER 1

COAST_CORE

COAST_CORE is a tool designed for aiding the credibility assessment of online articles. It is a collection of modules that are useful for assessing various aspects of credibility.

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

1.1 Features

COAST_CORE is made up of several modules for:

- N-Gram extraction
- Citation detection and classification
- Clarity of writing assessment
- Code detection
- Event detection
- Keyword detection
- Named entity detection

1.2 Prerequisites

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

There are two methods of named entity detection included as part of COAST_CORE. For running the Stanford named entity detection, you will need Java installed.

1.3 Installation

To install COAST_CORE, run this command in your terminal:

```
$ pip install coast_core
```

This is the preferred method to install COAST_CORE, 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_CORE, run this command in your terminal:

```
$ pip install coast_core
```

This is the preferred method to install COAST_CORE, 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_CORE can be downloaded from the Github repo.

You can either clone the public repository:

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

Or download the tarball:

```
$ curl  -OL https://github.com/zedrem/coast_core/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 Citations

3.1.1 Introduction

The citation module contains functions for doing the following:

- Given a block of HTML, it will extract all of the URLs by analysing the anchor <a> tags.
- Given the URL of the HTML being analysed, will determine which of the citations found are external resources.
- Given a JSON file of classifications, will classify each of the external URLs accordingly.

3.1.2 Usage

To use the citations module:

```
>>> import coast_core
>>> coast_core.citations.function(to_use)
```

or:

```
>>> from coast_core import citations
>>> citations.function(to_use)
```

3.1.3 Functions

A collection of functions that can be used for analysing the citations within results to other resources.

See the documentation and sample_data for examples (https://coast-core.readthedocs.io).

coast_core.citations.**classify_citations**(*external_uris*, *classification_config_file*)
Given a file containing a JSON object of key value {classification:[patterns]} pairs. Classify each of the citations for each article.

For example, given the following JSON:

```
{
  "research": ["reseaerchgate", "ieee.", "dx.doi.", "acm", "sciencedirect"]
}
```

All citations that contain any sub-string within the list will be classified as 'research' citations. A more detailed JSON example can be found in our test_data: https://github.com/zedrem/coast_core/blob/master/coast_core/resources/data/citations_classification.json

Parameters

- **external_uris** – A list of uris to classify.
- **classification_config_file** – A config file containing all classifications.

Returns A list of objects containing all classifications.

coast_core.citations.**compute_citation_binary_counts**(*classified_external_uris*, *classification_config_file*)
Take binary counts of each citation type.

Parameters

- **classified_external_uris** – a list of objects containing all classifications.
- **classification_config_file** – A config file containing all classifications.

Returns An object containing a binary count of each classification type.

coast_core.citations.**execute_full_citation_analysis**(*html*, *link*, *classification_config_file*)
Runs a complete end-to-end analysis of citations using all other functions.

Parameters

- **html** – The html to operate on.
- **link** – The link of the article being analysed.
- **classification_config_file** – A config file containing all classifications.

Returns An object containing all analysis.

coast_core.citations.**get_all_citations**(*html*)
Extract citations from a single articles HTML.

Parameters **html** – The html to operate on.

Returns A list of all URI's in lowercase form found in the article.

coast_core.citations.**get_an_articles_domain**(*link*)
For a given URL, parse and return the articles TLDN as a string.

Parameters **link** – The link to parse.

Returns The domain of the link.

coast_core.citations.**select_external_citations**(*link*, *all_uris*)
From a list of uri's, return those that are external to the domain of the link.

Parameters

- **link** – The link of the article being analysed.
- **all_uris** – A list of all URI's found in the article.

Returns A list of uris that are external to the domain of the link being analysed.

3.2 Clarity of writing

3.2.1 Introduction

The clarity of writing module assesses the readability, grammatical correctness and sentiment of a given body of text.

3.2.2 Usage

To use the module:

```
>>> import coast_core
>>> coast_core.clarity_of_writing.function(to_use)
```

or:

```
>>> from coast_core import clarity_of_writing
>>> clarity_of_writing.function(to_use)
```

3.2.3 Functions

A collection of functions that can be used for analysing the clarity of writing within an article

coast_core.clarity_of_writing.**analyse_readability_metrics**(*article_text*)
Use the textstat library to report multiple readability measures.

The readability metrics analysed are: * The Flesch Reading Ease Score. A score from 100 (very easy to read) to 0 (very confusing). * The grade score using the Flesch-Kincaid Grade Formula. For example a score of 9.3 means that a ninth grader would be able to read the document. * The FOG index of the given text * The SMOG index of the given text * The ARI(Automated Readability Index) which outputs a number that approximates the grade level needed to comprehend the text. For example if the ARI is 6.5, then the grade level to comprehend the text is 6th to 7th grade * The grade level of the text using the Coleman-Liau Formula * The grade level using the Lisear Write Formula * The grade level using the New Dale-Chall Formula.

Parameters **article_text** – The article text to operate on.

Returns An object containing all measures

coast_core.clarity_of_writing.**analyse_text_for_grammatical_metrics**(*article_text*)
Use the language_check library to check a body of text for grammatical issues.

Parameters **article_text** – The text to be analyse.

Returns The total number of grammatical issues found. A list containing details of each grammatical issue. A list of sentences, tokenized by NLTK.

coast_core.clarity_of_writing.**detect_language**(*text*)
Given a body of text, will use the langdetect library to detect the text language and return.

Parameters **text** – The body of text to analyse.

Returns The language code (e.g. EN for English).

coast_core.clarity_of_writing.**execute_clarity_of_writing_check**(*article_text*)
Runs a complete end-to-end analysis of clarity of writing using all other functions.

Parameters **article_text** – The article text to operate on.

Returns An object containing language, readability, grammar and sentiment

coast_core.clarity_of_writing.**run_sentiment_check**(*article_text*)
Run sentiment analysis over the article.

Parameters **article_text** – The article text to operate on.

Returns An object that contain polarity and subjectivity. Polarity, also known as orientation is he emotion expressed in the sentence. It can be positive, neagtive or neutral. Subjectivity is when text is an explanatory article which must be analysed in context.

3.3 Code detection

3.3.1 Introduction

The code detection module is used for identifying an extracting code examples within text. Regular expressions are used to identify the following features:

Feature	Regular Expression	Example
Arrow functions	`.(-|=)>.`	`Funct funct = ()-> { console.log("Hello"); }`
Full stops that don't have a space character either side	`\w\.\w`	`my_list.append(a_value)`
Camel case	`[A-Z][a-z0-9]+[A-Z][a-z0-9]\w+`	`MyFirstClass(args)`
Code comments	`\"\"\"|/(\*+)|//|\*+/|#|<!--|-->`	`# Here is a Python comment`
Curly brackets	`{|}`	`my_function(){...}`
Brackets that don't have a space either side	`\w\(.*?\)`	`my_function(type arg, type arg)`
Semi-colons	`.;`	`int i = 0;`
Uncommon characters	`(!|\+|-)=|\+|(\*|&|||=|<|>){1,2}|(_|:){2}`	`if(my_int > 0 and 'a' in __special_file.py:`
Words that are separated by an underscore	`[[:alnum:]]_[[:alnum:]]`	`some_words_separate_by_underscore = 5`
Square brackets that don't have a space either side	`\w\[.*?\]`	`for object in my_database['my_collection_name']`
Keywords	`(^|\s)" + keyword + "(\s|\(|\{|:|$)`	`if while else for each elif`

These default features (and keywords) are pulled from the patterns.json and keywords.txt files in coast_core/resources/data (https://github.com/zedrem/coast_core/tree/master/coast_core/resources/data). To add more features, you can simply add them to these files (Note that keywords can also be multiple words also).

3.3.2 Usage

To use the module:

```
>>> import coast_core
>>> coast_core.code_detection.function(to_use)
```

or:

```
>>> from coast_core import code_detection
>>> code_detection.function(to_use)
```

3.3.3 Functions

A collection of function that detect and analyse an article

coast_core.code_detection.**execute_code_detection**(*text*, *granularity='ALL'*)

Execute all the function of code detection analysis. You can choose what to return.

- ALL will return all the data we can get. This is the default value.
- BASIC will return the binary and the absolute data.
- FEATURES will return the detected features in the text
- LINES will return the lines data.

Parameters

- **text** – The text to operate on
- **granularity** – Will affect the returned data : ALL BASIC FEATURES LINES

Returns The return will depend of the granularity

coast_core.code_detection.**extract_absolute_data**(*lines_list*)

Extract the absolute data from lines.

Parameters **lines_list** – The list of line to operate on returned by the extract_lines_data function.

Returns An object containing the binary data of the lines.

coast_core.code_detection.**extract_binary_data**(*lines_list*)

Extract the binary data from lines.

Parameters **lines_list** – The list of line to operate on returned by the extract_lines_data function.

Returns An object containing the binary data of the lines.

coast_core.code_detection.**extract_features_by_words**(*text*)

Extract the features from words.

Parameters **text** – The text to operate on

Returns A list of features objects

coast_core.code_detection.**extract_lines_data**(*text*)

Extract the lines data from a text.

Parameters **text** – The text to operate on

Returns A list of lines objects

coast_core.code_detection.**extract_text_data**(*text*)
Extract the data of the text : total of characters, total of words, total of lines

Parameters **text** – The text to operate on

Returns an object containing th text data

coast_core.code_detection.**features_detection**(*word*)
Detect features in a word. Features come from the pattern.json in the resources directory

Parameters **word** – the word to operate on.

Returns the list of features of a word.

coast_core.code_detection.**percentage**(*string*, *total*)
Calculate the percentage of code in an string.

Parameters

- **string** – The string to operate on
- **total** – The total depending on what you base your percentage

Returns The percentage of code in the string

3.4 Events

3.4.1 Introduction

The event detection module analyses a body of text to identify events that have taken place. This is useful for identifying mentions of personal experience or story mining.

3.4.2 Usage

To use the module:

```
>>> import coast_core
>>> coast_core.events.function(to_use)
```

or:

```
>>> from coast_core import events
>>> events.function(to_use)
```

3.4.3 Functions

Extract event instances from text.

coast_core.events.**get_iverb_bigrams**(*text*)

Split a text into bigrams, following the pattern ("i", <<verb>>). VB - Verb, base form VBD - Verb, past tense VBG - Verb, gerund or present participle VBN - Verb, past participle - n VBP - Verb, non-3rd person singular present - n VBZ - Verb, 3rd person singular present - n

Parameters **text** – The text to analyse

Returns A dictionary containing 'I verb' bigrams and the total number of 'I verb' events.

coast_core.events.**get_timex_events**(*text*)
Given a body of text, returns a list of Timex events. Timex events are temporal events that are detected using regular expressions. Our timex library is a variation of the Timex module in NLTK_contrib: https://github.com/nltk/nltk_contrib/blob/master/nltk_contrib/timex.py

Parameters **text** – The text to operate on

Returns return the timex events

coast_core.events.**get_verb_events**(*text*)

Given a body of text, returns a list of verb events. VB - Verb, base form VBD - Verb, past tense VBG - Verb, gerund or present participle VBN - Verb, past participle - n VBP - Verb, non-3rd person singular present - n VBZ - Verb, 3rd person singular present - n

Parameters **text** – The text to analyse.

Returns The list of verb events.

coast_core.events.**run_all_event_analysis**(*article_text*)
Run all event analysis for all articles.

Parameters **article_text** – The text to analyse.

Returns An object containing timex events, verb events and iverb bigrams

coast_core.events.**timex_tag**(*text*, ***kwargs*)
Extract the timex events from a given body of text

Parameters **text** – The body of text to operate on

Returns A list of timex events as default, unless 'markup' argument is given. In which case, returns a markedup string.

3.5 Keyword detection

3.5.1 Introduction

The keyword (markers) module allows you to specify words and phrases to detect. For example, you may detect the presence of reasoning using the following list of markers: ['because', 'in my opinion', 'however'].

3.5.2 Usage

To use the module:

```
>>> import coast_core
>>> coast_core.markers.function(to_use)
```

or:

```
>>> from coast_core import markers
>>> markers.function(to_use)
```

3.5.3 Functions

A collection of functions that can be used for analysing the markers within result text.

coast_core.markers.**analyse_set_of_markers_for_a_given_article**(*article_text*, *list_of_markers*)

Given a link and a set of markers, return a frequency dictionary that shows how many times each marker appeared in the extracted article.

Parameters

- **article_text** – The text to search.
- **list_of_markers** – A list of markers to search for. Each marker is an ngram.

Returns A dictionary of markers and their frequency counts for a given link.

coast_core.markers.**run_all_markers**(*article_text*, *config_file*)

Runs a complete end-to-end analysis of markers using all other functions.

Parameters

- **article_text** – The text to search.
- **config_file** – A JSON file containing all relevant information for conducting the analysis. The config file should be structured as shown in the test data: https://github.com/zedrem/coast_core/blob/master/coast_core/resources/example/config_file.json. Each specific marker file should then be structured as shown in: https://github.com/zedrem/coast_core/blob/master/coast_core/resources/example/markers_experience_9.json.

Returns An object containing all markers found

3.6 Named entities

3.6.1 Introduction

The named entities module extracts named entites using NLTK, Stanford, and by just tagging text and extracting Personal Pronouns. Named entities are useful for identifying characters in stories, personal experience and events.

Note: In order to use the Stanford named entity detection, you will need to have Java installed.

3.6.2 Usage

To use the module:

```
>>> import coast_core
>>> coast_core.named_entities.function(to_use)
```

or:

```
>>> from coast_core import named_entities
>>> named_entities.function(to_use)
```

3.6.3 Functions

Title: named_entities.py

Author: Ashley Williams

Description: Extract named entities from text.

This module is called by init, so there is no need to import this module specifically.

coast_core.named_entities.**extract_all_named_entities**(*article_text*)
Extract the named entities for all extracted articles.

Parameters **article_text** – The article text to operate on.

Returns An object containing all named entities

coast_core.named_entities.**getNodes**(*parent*)
Never called externally, used to extract entities using nltk.

coast_core.named_entities.**get_nltk_named_entities**(*text*, *exception_list=[]*)
Returns a list of named entities in a given block of text using NLTK's averaged_perceptron_tagger.

Parameters

- **text** – The text to analyse.
- **exception_list** – A list of named entities to ignore.

Returns A list of named entities.

coast_core.named_entities.**get_pronouns**(*text*, *exception_list=[]*)

Returns a list of personal pronouns in a given block of text PRP - Personal pronouns PRP$ - Possessive pronoun

Parameters

- **text** – The text to analyse.
- **exception_list** – A list of named entities to ignore.

Returns A list of named entities.

coast_core.named_entities.**get_stanford_named_entities**(*text*, *exception_list=[]*)
Returns a list of named entities in a given block of text.

Parameters

- **text** – The text to analyse.
- **exception_list** – A list of named entities to ignore.

Returns A list of named entities.

3.7 Ngram Extraction

3.7.1 Introduction

The ngram_extraction module uses *nltk* to split a given block of text into ngrams.

3.7.2 Usage

To use the module:

```
>>> import coast_core
>>> coast_core.ngram_extraction.function(to_use)
```

or:

```
>>> from coast_core import ngram_extraction
>>> ngram_extraction.function(to_use)
```

3.7.3 Functions

A collection of functions that can be used for splitting the article into ngrams.

coast_core.ngram_extraction.**generate_ngrams**(*article_text*)

Split the given text into ngrams, returning an object that contains ngrams from one to six.

Parameters **article_text** – the block of text to operate on.

Returns

An object containing all ngrams up to 6 in the following structure:

```
{
    "unigrams": [list of unigrams],
    "bigrams": [list of bigrams],
    "trigrams": [list of trigrams],
    "fourgrams": [list of fourgrams],
    "fivegrams": [list of fivegrams],
    "sixgrams": [list of sixgrams]
}
```

3.8 Utils

3.8.1 Introduction

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

3.8.2 Usage

To use the citations module:

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

or: .. code-block:: console

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

3.8.3 Functions

A collection of generic utility functions that are used throughout coast by various modules relating to NLP tasks and the reporting and performance measures.

coast_core.utils.**get_from_file**(*path*)
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.

Parameters **path** – The path to the file you wish to read.

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

coast_core.utils.**get_json_from_file**(*path*)
Reads a JSON file and returns as an object. :param path: The path to the JSON file you wish to read. :return: A JSON object, generated from the contents of the file. :return: In the event of an error, the error is printed to the stdout.

coast_core.utils.**get_ngrams**(*text*, *number*)
Split a given body of text into ngrams.

Parameters

- **text** – The body of text to operate on.
- **number** – Specify the size of the ngram (e.g unigram, bigram etc).

Returns A list of ngrams.

coast_core.utils.**import_punkt**()
Import punkt

coast_core.utils.**penn_treebank_filter**(*article_text*, *filter_list*, *exception_list=[]*)
Returns a list of tuples that are tagged with any penn treebank tag from the filter list.

Parameters

- **article_text** – The text to analyse.
- **filter_list** – The tags to return.
- **exception_list** – A list of exception.

Returns A list of words containing any of the tags in the filter list.

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

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

```
$ git clone git@github.com:your_name_here/coast_core.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_core
$ cd coast_core/
$ 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_core 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 3.3, 3.4, 3.5, 3.6 and for PyPy. Check https://travis-ci.org/zedrem/coast_core/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
- 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.
April 2018	Yann Le Norment joins UC as an intern and works on a first public release.
May 2018	First public release!
August 2018	Version 0.1.2 released; fixing bugs and making things stable.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

R

S

T