pyrate Documentation

Release 0.1.7

Will Usher

Nov 09, 2017

Contents

Pyrate is a software architecture and suite of algorithms for the analysis of AIS data originating from ship-borne transceivers and collected by satellites and shore-based receivers. The different tools engage in an efficient and modular way, hence they are substitutable and extendable in a dynamic fashion. The primary goal is to validate and clean the dataset, extract information on shipping patterns and shipping routes. To make information easily discoverable, the data is stored in a variety of database types and formats.

CHAPTER 1

Contents

1.1 License

```
The MIT License (MIT)

Copyright (c) 2015 Julia Schaumeier, Sam Macbeth

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

1.2 Developers

- Julia Schaumeier
- Sam Macbeth
- Will Usher

1.3 pyrate

1.3.1 pyrate package

Subpackages

pyrate.algorithms package

Submodules

pyrate.algorithms.aisparser module

Parses the AIS data from csv of xml files and populates the AIS database

pyrate.algorithms.aisparser.**check_imo**(*imo*)

pyrate.algorithms.aisparser.**float_or_null**(*s*)

pyrate.algorithms.aisparser.**get_data_source**(*name*)

Guesses data source from file name.

If the name contains 'terr' then we guess terrestrial data, otherwise we assume satellite.

Parameters **name** (*str*) – File name

Returns 0 if satellite, 1 if terrestrial

Return type int

pyrate.algorithms.aisparser.**imostr**(*s*)

pyrate.algorithms.aisparser.**int_or_null**(*s*)

pyrate.algorithms.aisparser.**longstr**(*s*)

pyrate.algorithms.aisparser.**parse_file**(*fp*, *name*, *ext*, *baddata_logfile*, *cleanq*, *dirtyq*, *source=0*)

Parses a file containing AIS data, placing rows of data onto queues

Parameters

- **fp** (*str*) – Filepath of file to be parsed
- **name** (*str*) – Name of file to be parsed
- **ext** (*str*) – Extension, either '.csv' or '.xml'
- **baddata_logfile** (*str*) – Name of the logfile
- **cleanq** – Queue for messages to be inserted into clean table
- **dirtyq** – Queue for messages to be inserted into dirty table
- **source** (*int, optional, default=0*) – 0 is satellite, 1 is terrestrial

Returns

- **invalid_ctr** (*int*) – Number of invalid rows
- **clean_ctr** (*int*) – Number of clean rows
- **dirty_ctr** (*int*) – Number of dirty rows
- **time_elapsed** (*time*) – The time elapsed since starting the parse_file procedure

pyrate.algorithms.aisparser.**parse_raw_row**(*row*)
Parse values from row, returning a new dict with converted values

Parse values from row, returning a new dict with converted values converted into appropriate types. Throw an exception to reject row

Parameters **row** (*dict*) – A dictionary of headers and values from the csv file

Returns **converted_row** – A dictionary of headers and values converted using the helper functions

Return type dict

pyrate.algorithms.aisparser.**parse_timestamp**(*s*)

pyrate.algorithms.aisparser.**readcsv**(*fp*)
Returns a dictionary of the subset of columns required

Reads each line in CSV file, checks if all columns are available, and returns a dictionary of the subset of columns required (as per AIS_CSV_COLUMNS).

If row is invalid (too few columns), returns an empty dictionary.

Parameters **fp** (*str*) – File path

Yields **rowsubset** (*dict*) – A dictionary of the subset of columns as per *columns*

pyrate.algorithms.aisparser.**readxml**(*fp*)

pyrate.algorithms.aisparser.**run**(*inp*, *out*, *dropindices=True*, *source=0*)
Populate the AIS_Raw database with messages from the AIS csv files

Parameters

- **inp** (*str*) – The name of the repositor(-y/-ies) as defined in the global variable *INPUTS*
- **out** (*str*) – The name of the repositor(-y/-ies) as defined in the global variable *OUTPUTS*
- **dropindices** (*bool, optional, default=True*) – Drop indexes for faster insert
- **source** (*int, optional, default=0*) – Indicates terrestrial (1) or satellite data (0)

pyrate.algorithms.aisparser.**set_null_on_fail**(*row*, *col*, *test*)
Helper function which sets the column in a row of data to null on fail

Parameters

- **row** (*dict*) – A dictionary of the fields
- **col** (*str*) – The column to check
- **test** (*func*) – One of the validation functions in pyrate.utils

pyrate.algorithms.aisparser.**validate_row**(*row*)

pyrate.algorithms.aisparser.**xml_name_to_csv**(*name*)
Converts a tag name from an XML file to the corresponding name from CSV.

pyrate.algorithms.imolist module

pyrate.algorithms.imolist.**create_imo_list**(*aisdb*)
Create the imo list table from MMSI, IMO pairs in clean and dirty tables.

This method collects the unique MMSI, IMO pairs from a table, and the time intervals over-which they have been seen in the data. These tuples are then upserted into the *imo_list* table.

Removes cases where ships have clashing MMSI numbers within a time threshold.

On the clean table pairs with no IMO number are also collected to get the activity intervals of MMSI numbers. On the dirty table only messages specifying an IMO are collected.

Parameters **aisdb** (*postgresdb*) – The database upon which to operate

pyrate.algorithms.imolist.**run**(*_*, *out*)

pyrate.algorithms.vesselimporter module

Extracts a subset of clean ships into ais_extended tables

pyrate.algorithms.vesselimporter.**cluster_table**(*aisdb*, *table*)

Performs a clustering of the postgresql table on the MMSI index.

This process significantly improves the runtime of extended table generation.

pyrate.algorithms.vesselimporter.**filter_good_ships**(*aisdb*)

Generate a set of imo numbers and (mmsi, imo) validity intervals

Generate a set of imo numbers and (mmsi, imo) validity intervals for ships which are deemed to be 'clean'. A clean ship is defined as one which:

- Has valid MMSI numbers associated with it.
- For each MMSI number, the period of time it is associated with this IMO (via message number 5) overlaps with the period the MMSI number was in use.
- For each MMSI number, its usage period does not overlap with that of any other of this ship's MMSI numbers.
- That none of these MMSI numbers have been used by another ship (i.e. another IMO number is also associated with this MMSI)

Returns

- *valid_imos* – A set of valid imo numbers
- *imo_mmsi_intervals* – A list of (mmsi, imo, start, end) tuples, describing the validity intervals of each (mmsi, imo) pair

pyrate.algorithms.vesselimporter.**generate_extended_table**(*aisdb*, *intervals*, *n_threads=2*)

pyrate.algorithms.vesselimporter.**get_remaining_interval**(*aisdb*, *mmsi*, *imo*, *start*, *end*)

pyrate.algorithms.vesselimporter.**insert_message_stream**(*aisdb*, *interval*, *msg_stream*)

Takes a stream of messages for an MMSI over an interval, runs it through outlier detection and interpolation algorithms, then inserts the resulting stream into the ais_extended table.

pyrate.algorithms.vesselimporter.**interval_copier**(*db_options*, *interval_q*)

pyrate.algorithms.vesselimporter.**process_interval_series**(*aisdb*, *interval*)

pyrate.algorithms.vesselimporter.**run**(*inp*, *out*, *n_threads=2*, *dropindices=False*)

pyrate.algorithms.vesselimporter.**upsert_interval_to_imolist**(*aisdb*, *mmsi*, *imo*, *start*, *end*)

Module contents

pyrate.repositories package

Submodules

pyrate.repositories.aisdb module

class pyrate.repositories.aisdb.**AISExtendedTable**(*db*)
Bases: *pyrate.repositories.sql.Table*

create()

create_indices()

drop_indices()

class pyrate.repositories.aisdb.**AISdb**(*options*, *readonly=False*)
Bases: *pyrate.repositories.sql.PgsqlRepository*

action_log_spec = {'cols': [('timestamp', 'timestamp without time zone DEFAULT now()'), ('action', 'TEXT'), ('mm

clean_db_spec = {'cols': [('MMSI', 'integer'), ('Time', 'timestamp without time zone'), ('Message_ID', 'integer'), ('Na

clean_imo_list = {'cols': [('mmsi', 'integer NOT NULL'), ('imo', 'integer NULL'), ('first_seen', 'timestamp without t

create()
Create the tables for the AIS data.

dirty_db_spec = {'cols': [('MMSI', 'bigint'), ('Time', 'timestamp without time zone'), ('Message_ID', 'integer'), ('Nav

double_type = 'double precision'

get_message_stream(*mmsi*, *from_ts=None*, *to_ts=None*, *use_clean_db=False*, *as_df=False*)
Gets the stream of messages for the given mmsi, ordered by timestamp ascending

get_messages_for_vessel(*imo*, *from_ts=None*, *to_ts=None*, *use_clean_db=False*, *as_df=False*)

imolist_db_spec = {'cols': [('mmsi', 'integer NOT NULL'), ('imo', 'integer NULL'), ('first_seen', 'timestamp without

ship_info(*imo*)

sources_db_spec = {'cols': [('ID', 'SERIAL PRIMARY KEY'), ('timestamp', 'timestamp without time zone DEFAUL

status()

truncate()
Delete all data in the AIS table.

update()
Updates (non-destructively) existing tables to new schema

pyrate.repositories.aisdb.**load**(*options*, *readonly=False*)

pyrate.repositories.file module

class pyrate.repositories.file.**FileRepository**(*path*, *allowedExtensions=None*, *recursive=True*, *unzip=False*)
Bases: object

close()

iterfiles()
Iterate files in this file repository. Returns a generator of 3-tuples, containing a handle, filename and file extension of the current opened file.

status()

pyrate.repositories.file.**load**(*options*, *readonly=False*)

pyrate.repositories.sql module

Classes for connection to and management of database tables

PgsqlRepository

Sets up a connection to a pyrate database repository

Table

Used to encapsulate a pyrate database table

class pyrate.repositories.sql.**PgsqlRepository**(*options*, *readonly=False*)
Bases: object

connection()

class pyrate.repositories.sql.**Table**(*db*, *name*, *cols*, *indices=None*, *constraint=None*, *foreign_keys=None*)
Bases: object

A database table

copy_from_file(*fname*, *columns*)

create()
Creates tables in the database

create_indices()

drop_indices()

get_name()

insert_row(*data*)
Inserts one row into the table

insert_rows_batch(*rows*)
Inserts a number of rows into the table

Parameters **rows** (*list*) – A list of dicts of (column, value) pairs

status()
Returns the approximate number of records in the table

Returns

Return type integer

truncate()
Delete all data in the table.

pyrate.repositories.sql.**load**(*options*, *readonly=False*)

Module contents

pyrate.tools package

Submodules

pyrate.tools.resampler module

pyrate.tools.resampler.**convert_messages_to_hourly_bins**(*df*, *period='H'*, *fillnans=False*, *run_resample=True*)

Resample the messages to a new time-resolution.

Defaults to hourly.

Parameters

- **df** (*pandas DataFrame*) – A DataFrame of messages
- **period** (*string, optional*) – Indicates the period to resample over
- **fillnans** (*bool, optional*) – Defaults to False
- **run_resample** (*bool, optional*) – Defaults to True

Notes

Intended for use with the extended database

Called internally, one of the wrapper functions should be called

Module contents

Submodules

pyrate.cli module

Provides a command line interface to the pyrate library

The command line interface (CLI) expects that a configuration file named 'aistool.conf' is located in the current folder.

If the config file is not present, a runtime error is raised, and the commands *set_default* can be used to generate a default configuration file.

pyrate.cli.**main**()

The command line interface

Type *pyrate –help* for help on how to use the command line interface

pyrate.config_setter module

Generates a default config file in current folder

pyrate.config_setter.**gen_default_config**(**args*)
Generates a default config file in current folder

This command generates a default configuration file and folder structure in the current folder.

The folders generated are:

repositories To hold additional repository code for pyrate

algorithms To hold additional algorithm code for pyrate

aiscsv For AIS csv files (required by algorithms/aisparser.py)

baddata For AIS import logfiles (required by algorithms/aisparser.py)

pyrate.loader module

This module provides the Loader class which loads a pyrate session from a configuation file. This session can then run tasks on data repositories and algorithms.

class pyrate.loader.**Loader**(*config=None*)
Bases: object

The Loader joins together data repositories and algorithms, and executes operations on them.

execute_algorithm_command(*algname*, *command*, ***args*)
Execute the specified command on the specified algorithm

execute_repository_command(*reponame*, *command*, ***args*)
Execute the specified command on the specified repository.

get_algorithm(*name*)
Returns the algorithm module specified.

get_algorithm_commands(*algname*)
Returns a list of available commands for the specified algorithm

get_algorithms()
Returns a set of the names of available algorithms

get_data_repositories()
Returns a set of the names of available data repositories

get_data_repository(*name*, *readonly=False*)
Returns a loaded instance of the specified data repository.

get_repository_commands(*repo_name*)
Returns a list of available commands for the specified repository

pyrate.loader.**load_all_modules**(*paths*)
Load all modules on the given paths.

pyrate.loader.**load_module**(*name*, *paths*)
Load module name using the given search paths.

pyrate.utils module

pyrate.utils.**detect_location_outliers**(*msg_stream*, *as_df=False*)
Detects outlier messages by submitting messages to a speed test

The algorithm proceeds as follows:

1. Create a linked list of all messages with non-null locations (pointing to next message)

2. Loop through linked list and check for location outliers:
   - A location outlier is who does not pass the speed test (<= 50kn; link is 'discarded' when not reached in time)
   - No speed test is performed when:
     - distance too small (< 0.054nm ~ 100m; catches most positioning inaccuracies) => no outlier
     - time gap too big (>= 215h ~ 9d; time it takes to get anywhere on the globe at 50kn not respecting land) => next message is new 'start'

   If an alledged outlier is found its link is set to be the current message's link
3. The start of a linked list becomes special attention: if speed check fails, the subsequent link is tested

Line of thinking is: Can I get to the next message in time? If not 'next' must be an outlier, go to next but one.

Parameters

- **msg_stream** – A list of dictionaries representing AIS messages for a single MMSI number. Dictionary keys correspond to the column names from the *ais_clean* table. The list of messages should be ordered by timestamp in ascending order.
- **as_df** (*bool, optional*) – Set to True if *msg_stream* are passed as a pandas DataFrame

Returns The rows in the message stream which are outliers

Return type outlier_rows

pyrate.utils.**interpolate_passages**(*msg_stream*)
Interpolate far apart points in an ordered stream of messages.

Parameters **msg_stream** – A list of dictionaries representing AIS messages for a single MMSI number. Dictionary keys correspond to the column names from the *ais_clean* table. The list of messages should be ordered by timestamp in ascending order.

Returns **artificial_messages** – A list of artificial messages to fill in gaps/navigate around land.

Return type list

pyrate.utils.**is_valid_cog**(*cog*)
Validates course over ground

Parameters **cog** (*float*) – Course over ground

Returns

Return type True if course over ground is greater than zero and less than 360 degrees

pyrate.utils.**is_valid_heading**(*heading*)
Validates heading

Parameters **heading** (*float*) – The heading of the ship in degrees

Returns

Return type True if heading is greater than zero and less than 360 degrees

pyrate.utils.**is_valid_sog**(*sog*)
Validates speed over ground

Parameters **sog** (*float*) – Speed over ground

Returns

Return type True if speed over ground is greater than zero and less than 102.2

pyrate.utils.**speed_calc**(*msg_stream*, *index1*, *index2*)
Computes the speed between two messages in the message stream

Parameters

- **msg_stream** – A list of dictionaries representing AIS messages for a single MMSI number. Dictionary keys correspond to the column names from the *ais_clean* table. The list of messages should be ordered by timestamp in ascending order.
- **index1** (*int*) – The index of the first message
- **index2** (*int*) – The index of the second message

Returns

- **timediff** (*datetime*) – The difference in time between the two messages in datetime
- **dist** (*float*) – The distance between messages in nautical miles
- **speed** (*float*) – The speed in knots

pyrate.utils.**valid_imo**(*imo=0*)
Check valid IMO using checksum.

Parameters **imo** (*integer*) – An IMO ship identifier

Returns

Return type True if the IMO number is valid

Notes

Taken from Eoin O'Keeffe's *checksum_valid* function in pyAIS

pyrate.utils.**valid_latitude**(*lat*)
Check valid latitude.

Parameters **lon** (*integer*) – A latitude

Returns

Return type True if the latitude is valid

pyrate.utils.**valid_longitude**(*lon*)
Check valid longitude.

Parameters **lon** (*integer*) – A longitude

Returns

Return type True if the longitude is valid

pyrate.utils.**valid_message_id**(*message_id*)

pyrate.utils.**valid_mmsi**(*mmsi*)
Checks if a given MMSI number is valid.

Parameters **mmsi** (*int*) – An MMSI number

Returns

Return type Returns True if the MMSI number is 9 digits long.

pyrate.utils.**valid_navigational_status**(*status*)

Module contents

pyrate.**get_resource_filename**(*resource_name*)
Returns the absolute path associated with the file

Parameters **path** (*str*) – The expected file path

Returns **path** – The absolute file path

Return type str

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

L

M

P

R

S

T

U

V

X