Trouve Documentation

Release 0.6.0

Ry Whittington

Aug 26, 2023

Contents

Trouve specializes in finding discrete events in uniformly sampled, time-series data such as IoT and sensor data based on boolean conditional arrays. It currently only supports Python 3 on both Windows and Linux.

CHAPTER 1

Installation and Dependencies

Install `trouve` using the standard dependency manager `pip`:

```
pip install trouve
```

Dependencies:

- Toolz
- Numpy
- Pandas

Source code can be found on github

Trouve has been tested on both Windows and Linux on Python 3 only.

CHAPTER 2

Quickstart

2.1 Setup

Example events for quickstart.

```
>>> import numpy as np
>>> import trouve as tr
>>> import trouve.transformations as tt
>>> x = np.array([0, 1, 1, 0, 1, 0])
>>> example = tr.find_events(x > 0, period=1, name='example')
```

2.2 Finding Events

Find all occurrences where the `numpy.array` `x` is greater than zero. Assume the sample period is one second.

```
>>> sample_period = 1 #second
>>> example = find_events(x > 0, period=sample_period, name='example')
>>> len(example)
2
```

2.3 Applying Transformations

Transformation functions are applied in the specified order. Each transformation alters events inplace to avoid making unnecessary copies.

```
>>> deb = tt.debounce(2, 1)
>>> offset = tt.offset_events(0, 1)
>>> cond = x > 0
>>> deb_first = tr.find_events(cond, period=1,
```

(continues on next page)

(continued from previous page)

```
... transformations=[deb, offset])
>>> deb_first.to_array()
array([ 0.,  1.,  1.,  1.,  0.,  0.])
```

Note: Order matters with transformations.

Observe how the events change if the offset is applied before debouncing.

```
>>> offset_first = find_events(cond, period=1, transformations=[offset, deb])
>>> offset_first.to_array()
array([ 0.,  1.,  1.,  1.,  1.,  1.])
>>> offset_first == deb_first
False
```

2.4 Array Methods

Events provides several methods to produce array representations of events.

`numpy.ndarray` s via *Events.to_array* .

```
>>> example.to_array()
array([ 0.,  1.,  1.,  0.,  1.,  0.])



>>> example.to_array()
array([ 0.,  1.,  1.,  0.,  1.,  0.])



>>> example.to_array()
array([ 0.,  1.,  1.,  0.,  1.,  0.])
```

`pandas.Series` s via *Events.to_series* .

```
>>> example.to_series()
0    0.0
1    1.0
2    1.0
3    0.0
4    1.0
5    0.0
Name: example, dtype: float64
```

Boolean masks via

```
>>> example.to_series()
0    0.0
1    1.0
2    1.0
3    0.0
4    1.0
```

(continues on next page)

(continued from previous page)

```
5    0.0
Name: example, dtype: float64
```

Boolean masks via

```
>>> example.to_series()
0    0.0
1    1.0
2    1.0
3    0.0
4    1.0
5    0.0
Name: example, dtype: float64
```

Boolean masks via *Events.to_array* for use with the numpy.ma module.

```
>>> example.to_array(1, 0, dtype=np.bool)
array([ True, False, False,  True, False,  True], dtype=bool)
>>> x > 0
array([False,  True,  True, False,  True, False], dtype=bool)
```

2.5 Inspecting Events

The trouve.Events class implements __getitem__ which returns an *Occurrence* .

```
>>> first_event = example[0]
>>> first_event.duration
2
>>> x[first_event.slice]
array([1, 1])
```

trouve.Events is also an iterable through implementation of both __iter__ and __next__. Every iteration returns an *Occurrence* .

```
>>> for event in example:
...     print(event.duration)
2
1
```

2.6 Magic Methods

Trouve implements several magic methods including:

__len__ for determining the number of events found using len.

```
>>> len(example)
2
```

__str__ for printing a summary of the events with print.

```
>>> print(example)
example
Number of events: 2
Min, Max, Mean Duration: 1.000s, 2.000s, 1.500s
```

__eq__ for determining if two events are equal.

```
>>> example == example_2
True
```

Note: Equality compares _starts, _stops, _period and _condition_size of both Event``s. The event ``name does **not** have to be the same for both events.

__repr__ for help with trouble-shooting using repr.

```
>>> repr(example)
"Events(_starts=array([1, 4]), _stops=array([3, 5]), _period=1, name='example', _
↪condition_size=6)"
```

CHAPTER 3

Events

The primary function of `trouve` is to find events in time-series data and apply functional transformations in a specified order. The main function is *`find_events`*. This function takes in a conditional `bool` and then returns the class *`Events`*. The *`Events`* class finds each distinct occurrence and records it's start and stop index value. These values then allow a user to inspect each event in a Pythonic manner.

`trouve.find_events.`**`find_events`**`()`

Find events based off a condition

Find events based off a `bool` conditional array and apply a sequence of transformation functions to them. The `find_events` function is curried via `toolz.curry`. Most datasets are of the same sample rate, this is a convenience so that one can specify it once.

Parameters

- **`condition`** (`numpy.ndarray` or `pandas.Series` of `bool`) – Boolean conditional array.
- **`period`** (`float`) – Time in seconds between each data point. Requires constant increment data that is uniform across the array. (1/Hz = s)
- **`transformations`** (sequence of `callable` 's, optional) – Ordered sequence of transformation functions to apply to events. Transformations are applied via `toolz.pipe()`
- **`name`** (`str`, optional) – Default is `'events'`. User provided name for events.

Returns Returns events found from `condition` with any supplied `transformations` applied.

Return type *`trouve.events.Events`*

Examples

```
>>> import trouve as tr
>>> import trouve.transformations as tt
>>> import numpy as np
>>> deb = tt.debounce(2, 2)
```

(continues on next page)

(continued from previous page)

```
>>> offsets = tt.offset_events(-1,2)
>>> filt_dur = tt.filter_durations(3, 5)
>>> x = np.array([4, 5, 1, 2, 3, 4, 5, 1, 3])
>>> condition = (x > 2)
>>> no_transforms = tr.find_events(condition, period=1)
>>> events = tr.find_events(condition, period=1,
... transformations=[deb, filt_dur, offsets])
>>> no_transforms.to_array()  # doctest: +SKIP
array([ 1.,  1.,  0.,  0.,  1.,  1.,  1.,  0.,  1.])
>>> events.to_array()  # doctest: +SKIP
array([ 0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.])
```

class trouve.events.**Events**(*starts*, *stops*, *period*, *name*, *condition_size*)

Object to represent events found in time series data

A representation of events based off a `bool` conditional array.

name

User provided name for events.

Type `str`

_starts

The index for event starts

Type `np.array` of `int`

_stops

The index for event stops

Type `np.array` of `int`

_period

Time between each value of the original condition array

Type `float`

_condition_size

The size of the original condition array

Type `int`

durations

Return a `numpy.ndarray` of event durations in seconds.

Examples

```
>>> import trouve as tr
>>> x = np.array([2, 2, 4, 5, 3, 2])
>>> condition = x == 2
>>> events = tr.find_events(condition, period=1)
>>> events.to_array()  # doctest: +SKIP
array([1., 1., 0., 0., 0., 1.])
>>> print(events.durations)
[2 1]
```

to_array(*inactive_value=0*, *active_value=1*, *dtype=None*, *order='C'*)

Returns a `numpy.ndarray` identifying found events

Useful for plotting or building another mask based on identified events.

Parameters

- **inactive_value** (float, optional) – Default is 0. Value of array where events are not active.
- **active_value** (float, optional) – Default is 1. Value of array where events are active.
- **dtype** (numpy.dtype, optional) – Default is numpy.float64. The datatype of returned array.
- **order** (str, optional) – Default is 'C'. {'C', 'F'} whether to store multidimensional data in C- or Fortran-contiguous (row- or column-wise) order in memory.

Returns

An array where values are coded to identify when events are active or inactive.

Return type numpy.ndarray

Examples

```
>>> import trouve as tr
>>> x = np.array([2, 2, 4, 5, 3, 2])
>>> condition = x > 2
>>> print(condition)
[False False  True  True  True False]
>>> events = tr.find_events(condition, period=1)
>>> events.to_array()  # doctest: +SKIP
array([0., 0., 1., 1., 1., 0.])
```

to_series(*inactive_value=0*, *active_value=1*, *index=None*, *dtype=None*, *name=None*)

Returns a pandas.Series identifying found events

Useful for plotting and for filtering a pandas.DataFrame

Parameters

- **inactive_value** (float, optional) – Default is 0. Value of array where events are not active.
- **active_value** (float, optional) – Default is 1. Value of array where events are active.
- **index** (array-like or Index (1d)) – Values must be hashable and have the same length as data. Non-unique index values are allowed. Will default to RangeIndex(len(data)) if not provided. If both a dict and index sequence are used, the index will override the keys found in the dict.
- **dtype** (numpy.dtype or None) – If None, dtype will be inferred.
- **name** (str, optional) – Default is *Events.name*. Name of series.

Returns A series where values are coded to identify when events are active or inactive.

Return type pandas.Series

Examples

```
>>> import trouve as tr
>>> x = np.array([2, 2, 4, 5, 3, 2])
>>> condition = x > 2
```

(continues on next page)

(continued from previous page)

```
>>> print(condition)
[False False  True  True  True False]
>>> events = tr.find_events(condition, period=1)
>>> events.to_series()
0    0.0
1    0.0
2    1.0
3    1.0
4    1.0
5    0.0
Name: events, dtype: float64
```

__getitem__(*item*)

Get a specific *Occurrence*

Examples

```
>>> import numpy as np
>>> import trouve as tr
>>> x = np.array([0, 1, 1, 0, 1, 0])
>>> example = tr.find_events(x, period=1, name='example')
>>> first_event = example[0]
>>> print(first_event)
Occurrence(start=1, stop=2, slice=slice(1, 3, None), duration=2)
```

__len__()

Returns the number of events found

Redirects to *Events._starts* and returns `Events._starts.size`

Examples

```
>>> import numpy as np
>>> import trouve as tr
>>> x = np.array([0, 1, 1, 0, 1, 0])
>>> example = tr.find_events(x, period=1, name='example')
>>> len(example)
2
```

__repr__()

Return repr(self).

__str__()

Prints a summary of the events

Examples

```
>>> import numpy as np
>>> import trouve as tr
>>> x = np.array([0, 1, 1, 0, 1, 0])
>>> example = tr.find_events(x, period=1, name='example')
>>> print(example)
```

(continues on next page)

(continued from previous page)

```
example
Number of events: 2
Min, Max, Mean Duration: 1.000s, 2.000s, 1.500s
```

__eq__(*other*)

Determine if two Events objects are identical

Compares *Events._starts*, *Events._stops*, *Events._period* and `Events.condition.size` to determine if equality of two events. Events objects can have different names and still be equal.

Examples

```
>>> import numpy as np
>>> import trouve as tr
>>> x = np.array([0, 1, 1, 0, 1, 0])
>>> example = tr.find_events(x, period=1, name='example')
>>> other = tr.find_events(x, period=1, name='other')
>>> id(example) # doctest: +SKIP
2587452050568
>>> id(other) # doctest: +SKIP
2587452084352
>>> example == other
True
>>> example != other
False
```

class `trouve.events.`**Occurrence**(*start*, *stop*, *slice*, *duration*)

`trouve.events.Occurrence` is a `collections.namedtuple` that is returned by both *Events.__getitem__* and `Events.__next__`

Parameters:

- start (`int`): Index of the start of the occurrence
- stop (`int`): Index of the stop of the occurrence
- slice (`slice`): `slice` object for the entire occurrence
- duration (`float`): Duration in seconds of the occurrence

Examples:

```
>>> import numpy as np
>>> import trouve as tr
>>> x = np.array([0, 1, 1, 0, 1, 0])
>>> example = tr.find_events(x, period=1, name='example')
>>> first_event = example[0]
>>> print(first_event)
Occurrence(start=1, stop=2, slice=slice(1, 3, None), duration=2)
>>> first_event.start
1
>>> x[first_event.slice]
array([1, 1])
```

CHAPTER 4

Transformations

This page contains all available transformations, relevant functions, and classes available in `trouve`.

debounce([activate_debounce, ...])	Debounce activation and deactivation of events
filter_durations([min_duration, max_duration])	Filter out durations based on length of time active
offset_events([start_offset, stop_offset])	Apply an offset to event start and stops
merge_overlap(events)	Merge any events that overlap

4.1 Definitions

`trouve.transformations.`**`debounce`**(*activate_debounce=None*, *deactivate_debounce=None*)

Debounce activation and deactivation of events

Find an occurrence that is active for time >= activate_debounce and activate event. Deactivate event only after an occurrence is found that is inactive for time >= to deactivate_debounce. Filter out all events that fall outside of these bounds. This function is used to prevent short duration occurrences from activating or deactivating longer events. See mechanical debounce in mechanical switches and relays for a similar concept.

Parameters

- **`activate_debounce`** (`float`) – Default is `None`. Default value does not apply an activate_debounce. Minimum time in seconds an occurrence must be active to activate an event. (event active >= `activate_debounce`)
- **`deactivate_debounce`** (`float`) – Default is `None`. Default value does not apply an deactivate_debounce. Maximum time in seconds an occurrence must be inactive to deactivate an event. (event inactive >= `deactivate_debounce`)

Returns Partial function

Return type `callable`

Examples

```
>>> import trouve as tr
>>> import trouve.transformations as tt
>>> import numpy as np
>>> y = np.array([2, 3, 2, 3, 4, 5, 2, 3, 3])
>>> condition = y > 2
>>> events = tr.find_events(condition, period=1)
>>> deb = tt.debounce(2, 2)
>>> trans_events = tr.find_events(condition, period=1, transformations=[deb])
>>> events.to_array()  # doctest: +SKIP
array([ 0.,  1.,  0.,  1.,  1.,  1.,  0.,  1.,  1.])
>>> trans_events.to_array()  # doctest: +SKIP
array([ 0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.])
```

Raises `ValueError` – If `activate_debounce` or `deactivate_debounce` < 0

`trouve.transformations.`**`filter_durations`**(*min_duration=None*, *max_duration=None*)

Filter out durations based on length of time active

Filter out events that are < min_duration and > max_duration (time in seconds).

Parameters

- **`min_duration`** (`float`) – Default is `None`. Default value does not apply a min_duration filter. Filter out events whose duration in seconds is < min_duration.
- **`max_duration`** (`float`) – Default is `None`. Default value does not apply a max_duration filter. Filter out events whose duration in seconds is > max_duration.

Returns Partial function

Return type `callable`

Raises `ValueError` – If min_duration or max_duration is < 0

Examples

```
>>> import trouve as tr
>>> import trouve.transformations as tt
>>> y = np.array([2, 3, 2, 3, 4, 5, 2, 3, 3])
>>> condition = y > 2
>>> events = tr.find_events(condition, period=1)
>>> filt_dur = filter_durations(1.5, 2.5)
>>> trans_events = tr.find_events(condition, period=1, transformations=[filt_dur])
>>> events.to_array()  # doctest: +SKIP
array([ 0.,  1.,  0.,  1.,  1.,  1.,  0.,  1.,  1.])
>>> trans_events.to_array()  # doctest: +SKIP
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.])
```

`trouve.transformations.`**`offset_events`**(*start_offset=None*, *stop_offset=None*)

Apply an offset to event start and stops

Offset the starts and stops of events by the time in seconds specified by start_offset and stop_offset.

Parameters

- **`start_offset`** (`float`) – Default is `None`. Time in seconds to offset event starts. Value must be <= 0.

- **stop_offset** (*float*) – Default is None. Time in seconds to offset event stops. Value must be >= 0.

Returns Partial function

Return type callable

Raises ValueError – If start_offset > 0 or stop_offset < 0

Examples

```
>>> import trouve as tr
>>> import trouve.transformations as tt
>>> y = np.array([2, 2, 2, 3, 4, 5, 2, 2, 2])
>>> condition = y > 2
>>> events = tr.find_events(condition, period=1)
>>> offset = tt.offset_events(-1, 1)
>>> trans_events = tr.find_events(condition, period=1, transformations=[offset])
>>> events.to_array()  # doctest: +SKIP
array([ 0.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.])
>>> trans_events.to_array()  # doctest: +SKIP
array([ 0.,  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.])
```

trouve.transformations.**merge_overlap**(*events*)

Merge any events that overlap

Some events such as offset_events can cause events to overlap. If this transformation is applied, any events that overlap will become one contiguous event.

Parameters **events** (*trouve.events.Events*) –

Returns Any overlapping events merged into one event.

Return type *trouve.events.Events*

Examples

```
>>> import trouve as tr
>>> import trouve.transformations as tt
>>> y = np.array([2, 3, 2, 3, 4, 5, 2, 2, 2])
>>> condition = y > 2
>>> offset = tt.offset_events(-1, 1)
>>> events = tr.find_events(condition, period=1, transformations=[offset])
>>> merged_events = tr.find_events(condition,  period=1,
... transformations=[offset, merge_overlap])
>>> events.to_array()  # doctest: +SKIP
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.])
>>> merged_events.to_array()  # doctest: +SKIP
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.])
>>> len(events)
2
>>> len(merged_events)
1
```

CHAPTER 5

Tips and Tricks

Here are some recipes to effectively use Trouve to it's full potential.

5.1 Specify Sample Period for Reuse

If you're looking for multiple events in the same data set, then one shortcut is to specify the period once. The :any:`find_events` function is curried via `toolz.curry`, allowing a user to specify the period once for reuse.

```
>>> x = np.array([1, 1, 2, 0, 2])
>>> period = 1
>>> find_events = tr.find_events(period=period)
>>> events_1 = find_events(x == 1)
>>> events_1.as_array()
array([ 1.,  1.,  0.,  0.,  0.])
>>> events_2 = find_events(x == 2)
>>> events_2.to_array()
array([ 0.,  0.,  1.,  0.,  1.])
```

5.2 Multi-parameter Conditional Array

The condition can be as complicated as necessary. Using multiple inputs and the ampersand (`&`) or the pipe (`|`). The following example find events where x > 0 and y == 2, or z <= 1. `((x > 0) & (y == 2)) | (z <= 1)`

When using more than one parameter, you must put each expression in its own parenthesis

```
>>> x = np.array([1, 1, 0, 0, 1, 1, 0, 1, 0, 1])
>>> y = np.array([2, 2, 0, 0, 0, 0, 0, 0, 0, 2])
>>> z = np.array([2, 2, 2, 3, 3, 0, 3, 3, 3, 3])
>>> cond = ((x > 0) & (y == 2)) | (z <= 1)
>>> events = tr.find_events(cond, period=1)
>>> events.to_array()
```

(continues on next page)

(continued from previous page)

```
array([ 1.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.])


>>> z = np.array([2, 2, 2, 3, 3, 0, 3, 3, 3, 3])
>>> cond = ((x > 0) & (y == 2)) | (z <= 1)
>>> events = tr.find_events(cond, period=1)
>>> events.as_array()
array([ 1.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.])
```

5.3 Events and the `numpy.ma` Module

The `Events.as_mask` method was developed to integrate directly with `numpy.ma.MaskedArray` and `numpy.ma.masked_where`. The `numpy.ma` module makes things like summing or finding min/max of arrays based on your condition.

```
>>> x = np.array([-1, 1, -1, -1, 1, 1, -1, 1, -1, 1])
>>> cond = x == 1
>>> events = tr.find_events(cond, period=1)
>>> mask = events.as_mask()
>>> np.ma.masked_where(mask, x)
masked_array(data = [-- 1 -- -- 1 1 -- 1 -- 1],
             mask = [ True False  True  True False False  True False  True False],
       fill_value = 999999)

>>> masked_x = np.ma.MaskedArray(x, mask)
>>> masked_x.sum()
5
>>> x.sum()
0
```

5.4 Getting Events into a `pandas.DataFrame`

The `pandas.DataFrame` data structure and `trouve` fit nicely together. You can loop through each occurrence and append a statistical description to the dataframe. This is helpful you your trying to pull features out of time-series data for a machine learning algorithm, or you want to describe all events found in a data set and then use `pandas` idioms to further process them.

```
>>> x = np.array([-1, 1, -1, -1, 1, 1, -1, 1, -1, 1])
>>> y = np.array([1, 2, 3, 4, 5, 4, 3, 2, 1, 0])
>>> cond = x == 1
>>> events = tr.find_events(cond, period=1)
>>> columns = ['duration', 'ave_y_value', 'y_value_at_event_start']
>>> df = pd.DataFrame(index=pd.RangeIndex(len(events)), columns=columns)
>>> for i, occurrence in enumerate(events):
...     df.iloc[i] = dict(
...         duration=occurrence.duration,
...         ave_y_value= y[occurrence.slice].mean(),
...         y_value_at_event_start=y[occurrence.start]
...     )
>>> df
  duration ave_y_value y_value_at_event_start
```

(continues on next page)

(continued from previous page)

```
0         1            2                       2
1         2          4.5                       5
2         1            2                       2
3         1            0                       0
```

5.5 Finding Inverse Events

If you're interested in when events aren't active, then you can use the inverse of the condition. This would be helpful if you wanted to know the average, min, or max time between events.

```
>>> x = np.array([-1, 1, -1, -1, 1, 1, -1, 1, -1, 1])
>>> cond = x == 1
>>> events = find_events(cond, period=1)
>>> inv_events = find_events(~cond, period=1)
>>> events.as_array()
array([ 0.,  1.,  0.,  0.,  1.,  1.,  0.,  1.,  0.,  1.])
>>> inv_events.to_array()
array([ 1.,  0.,  1.,  1.,  0.,  0.,  1.,  0.,  1.,  0.])
```

5.6 `Events.durations` Tips

Total time in seconds events are active.

```
>>> x = np.array([-1, 1, -1, -1, 1, 1, -1, 1, -1, 1])
>>> cond = x == 1
>>> events = tr.find_events(cond, period=1)
>>> events.durations.sum()
5
```

Occurrence rate: Occurrences/second

```
>>> x = np.array([-1, 1, -1, -1, 1, 1, -1, 1, -1, 1])
>>> cond = x == 1
>>> period = 1
>>> events = tr.find_events(cond, period=period)
>>> len(events) / (x.size * period)
0.4
```

Creating a histogram of event lengths

```
>>> x = np.array([-1, 1, -1, -1, 1, 1, -1, 1, -1, 1])
>>> cond = x == 1
>>> events = ftr.ind_events(cond, period=1)
>>> np.histogram(events.durations, [0, 0.5, 1, 1.5, 2, 2.5])
(array([0, 0, 3, 0, 1], dtype=int64), array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5]))
```

CHAPTER 6

Change Log

6.1 0.6.0

- Apply `toolz.curry` to :any:`trouve.find_events`

6.2 0.5.2

- Fixed bug where events with no occurrences failed with *transformations.merge_overlap* applied to them

6.3 0.5.1

- Fixed issue where deprecated methods in 0.5.0 didn't issue deprecation warnings

6.4 0.5.0

Events methods

- Deprecate `Events.as_array`, use *`Events.to_array`*
- Deprecate `Events.as_series`, use *`Events.to_series`*
- Deprecate `Events.as_mask`, use *`Events.to_array`* with `inactive_value=1`, `ative_values=` and `dtype=np.bool`

Transformations

- Deprecate passing transformation functions as *args to `trouve.find_events`. Pass them to the explicit transformations keyword arguments

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

Symbols

D

E

F

M

N

O

T