
sciquence Documentation

Release 0.1.0

Krzysztof Joachimiak

Sep 04, 2017

1	Intro	1
2	Installation	3
3	sciquence API	5
3.1	sciquence.sequences	5
3.2	sciquence.dtw	14
3.3	sciquence.sax_vsm	15
3.4	sciquence.sliding_window	15
3.5	sciquence.representation	18
3.6	sciquence.text_processing	19
3.7	sciquence.data_structures	20
4	Indices and tables	23

CHAPTER 1

Intro

Sciquence is a python module created especially to work with time series and other types of sequences. It mimics scikit-learn API, but introduces its own extensions as well.

<http://www.timeseriesclassification.com/index.php>

CHAPTER 2

Installation

To install current bleeding-edge `sciquence` version, simply use command:

```
sudo pip install git+https://github.com/krzjoa/sciquence.git
```


sciquence.sequences

Cutting & trimming

<i>seq</i> (array)	Cut input array into sequences consisting of the same elements
<i>nseq</i> (array)	Returns sequences consisting of zeros
<i>pseq</i> (array)	Returns sequences consisting of ones
<i>specseq</i> (array, element)	Return sequences consisting of specific tag
<i>seqi</i> (array)	Get list of sequences and corresponding list of indices
<i>nseqi</i> (array)	Get list of sequences and corresponding list of indices
<i>pseqi</i> (array)	Get list of sequences and corresponding list of indices
<i>chunk</i> (array, chunk_size)	Split numpy array into chunks of equal length.

sciquence.sequences.seq

`sciquence.sequences.seq(array)`

Cut input array into sequences consisting of the same elements

Parameters `array` (*ndarray*) – Numpy array

Returns `seq_list` – List of sequences

Return type list of ndarray

Examples

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0])
```

```
>>> print sq.seq(x)
[array([1, 1, 1, 1, 1, 1]), array([0, 0, 0, 0, 0, 0]), array([1, 1, 1, 1, 1]),
↪array([0, 0, 0, 0])]
```

sciquence.sequences.nseq

sciquence.sequences.nseq(array)

Returns sequences consisting of zeros

Parameters array (array-like) – Numpy array

Returns seq_list – List of negative sequences

Return type list of ndarray

Examples

```
>>> from sciquence import sequences as sq
>>> import numpy as np
>>> x = np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0])
>>> print sq.nseq(x)
[array([0, 0, 0, 0, 0, 0]), array([0, 0, 0, 0])]
```

sciquence.sequences.pseq

sciquence.sequences.pseq(array)

Returns sequences consisting of ones

Parameters array (array-like) – Numpy array

Returns seq_list – List of positive sequences

Return type list of ndarray

Examples

```
>>> from sciquence import sequences as sq
>>> import numpy as np
>>> x = np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0])
>>> print sq.pseq(x)
[array([1, 1, 1, 1, 1, 1]), array([1, 1, 1, 1, 1])]
```

sciquence.sequences.specseq

sciquence.sequences.specseq(array, element)

Return sequences consisting of specific tag

Parameters

- array (ndarray) – Numpy array
- element (object) – Element

Returns `seq_list` – List of sequences consisting of specific tag

Return type list of ndarray

Examples

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 44, 44, 44, 44, 44, 1, 1,
↪ 0, 0, 0, 0])
>>> print sq.specseq(x, 44)
[array([44, 44, 44, 44, 44])]
```

sciquence.sequences.seqi

`sciquence.sequences.seqi` (*array*)

Get list of sequences and corresponding list of indices

array: ndarray Numpy array

seq_list: list of ndarray List of sequences

idx_list: list of ndarray List of sequences indices

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 44, 44, 44, 44, ↪
↪44, 1, 1, 0, 0, 0, 0])
>>> print sq.seqi(x)
```

```
[[0, 1, 2, 3, 4, 5], [6, 7, 8, 9, 10, 11], [12], [13, 14, 15, 16, 17], [18, 19], [20, 21, 22, 23]]
```

sciquence.sequences.nseqi

`sciquence.sequences.nseqi` (*array*)

Get list of sequences and corresponding list of indices

array: ndarray Numpy array

seq_list: list of ndarray List of sequences

idx_list: list of ndarray List of sequences indices

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 44, 44, 44, 44, ↪
↪44, 1, 1, 0, 0, 0, 0])
>>> print sq.nseqi(x)
```

```
[[0, 1, 2, 3, 4, 5], [6, 7, 8, 9, 10, 11], [12], [13, 14, 15, 16, 17], [18, 19], [20, 21, 22, 23]]
```

sciquence.sequences.pseqi

sciquence.sequences.pseqi (array)

Get list of sequences and corresponding list of indices

Parameters **array** (ndarray) – Numpy array

Returns

- **seq_list** (list of ndarray) – List of sequences
- **idx_list** (list of ndarray) – List of sequences indices

Examples

sciquence.sequences.chunk

sciquence.sequences.chunk (array, chunk_size)

Split numpy array into chunks of equal length.

Parameters

- **array** (ndarray) – A numpy array
- **chunk_size** (int) – Desired length of a single chunk

Returns **chunks** – Chunks of equal length

Return type list of ndarray

Examples

```
>>> import numpy as np
>>> import sciquence.sequences as sq
>>> x = np.array([1,2,3,4,5,6,7,8,9,10])
>>> sq.chunk(x, 3)
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9]), array([10])]
```

Comparing

<code>lseq_equal(lseqa, lseqb)</code>	Compare two lists of ndarrays
<code>shapes_equal(*arrays)</code>	Check if all the arrays have the same shape.
<code>size_equal(*arrays, **kwargs)</code>	Check if all the arrays have the same length along the particular axis.

sciquence.sequences.lseq_equal

sciquence.sequences.lseq_equal (lseqa, lseqb)

Compare two lists of ndarrays

Parameters

- **lseqa** (list of ndarray) – List of sequences
- **lseqb** (list of ndarray) – List of sequences

Returns `ans` – True if lists equal, otherwise False

Return type bool

Examples

```
>>> from sciquence import sequences as sq
>>> import numpy as np
>>> x = [np.array([1, 2, 3, 4]), np.array([6, 7, 8])]
>>> y = [np.array([1., 2.8, 3., 4.]), np.array([6.1, 7., 8.5])]
>>> z = [np.array([1, 2, 3, 4]), np.array([6, 7, 8])]
>>> print sq.lseq_equal(x, y)
False
>>> print sq.lseq_equal(x, z)
True
```

`sciquence.sequences.shapes_equal`

`sciquence.sequences.shapes_equal(*arrays)`

Check if all the arrays have the same shape.

Parameters `arrays` (*ndarrays*) – Numpy arrays

Returns `are_equal` – True if all the arrays have the same shape, otherwise False

Return type bool

Examples

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.random.rand(1, 2)
>>> y = np.random.rand(1, 2)
>>> z = np.random.rand(1, 2, 3)
>>> sq.shapes_equal(x, y)
True
>>> sq.shapes_equal(x, y, z)
False
```

`sciquence.sequences.size_equal`

`sciquence.sequences.size_equal(*arrays, **kwargs)`

Check if all the arrays have the same length along the particular axis.

Parameters

- `arrays` (*ndarrays*) – Numpy arrays
- `kwargs` –
 - `axis: int` Axis index, default: 0

Returns `are_equal` – True if all the arrays have the same shape, otherwise False

Return type bool

Examples

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.random.rand(1, 2)
>>> y = np.random.rand(1, 2)
>>> z = np.random.rand(1, 2, 3)
>>> v = np.random.rand(2, 2, 3)
>>> sq.shapes_equal(x, y)
True
>>> sq.shapes_equal(x, y, z)
True
>>> sq.shapes_equal(x, y, z, v)
True
```

Sorting

`parallel_sort(*arrays, **kwargs)`

Parallel sort.

sciquence.sequences.parallel_sort

`sciquence.sequences.parallel_sort(*arrays, **kwargs)`

Parallel sort. It always uses values from the first array to sort all them.

arrays: lists or/and ndarrays Numpy arrays (at least one)

kwargs:

- **reverse:** bool If True, reversed sort is performed

Returns `sorted_arrays` – New arrays, parallelly sorted accordingly to the first array’s elements

Return type ndarrays

Examples

```
>>> import sciquence.sequences as sq
>>> import numpy as np
>>> x = np.array([2., 3., 5., 1.45, 6, 4.2])
>>> y = np.array([0, 1, 0, 0, 0, 1])
>>> z = np.array([[0, 1, 56, 67, 90, 100],
>>>               [78, 34, 13, 49, 25, 101]]).T
>>> print sq.parallel_sort(x, y, z)
[array([ 1.45,  2.   ,  3.   ,  4.2  ,  5.   ,  6.   ]),
 array([0, 0, 1, 1, 0, 0]),
 array([[ 67,  49],
        [  0,  78],
        [  1,  34],
        [100, 101],
        [ 56,  13],
        [ 90,  25]])]
```

Input transformations

<code>rnn_input(X, y, window_size[, step])</code>	Prepare input for recurrent neural network.
<code>seq2seq_input(X, y, window_size[, step, ...])</code>	Prepare input for sequence2sequence recurrent neural network.

sciquence.sequences.rnn_input

`sciquence.sequences.rnn_input(X, y, window_size, step=1)`

Prepare input for recurrent neural network. Input is prepared from two parallel time series: series of feature vectors and series of labels.

Parameters

- **X** (`ndarray (time_series_length, n_features)`) – A time series of feature vectors
- **y** (`ndarray (time_series_length,)`) – A time series of labels
- **window_size** (`int`) – Integer odd number
- **step** (`int`) – Step of sliding window

Returns

- **new_X** (`ndarray (n_samples, n_timesteps, n_features)`) – Input for recurrent neural network
- **new_y** (`ndarray (n_samples,)`) – Time series of labels

sciquence.sequences.seq2seq_input

`sciquence.sequences.seq2seq_input(X, y, window_size, step=1, output_dim=1)`

Prepare input for sequence2sequence recurrent neural network. Input is prepared from two parallel time series: series of feature vectors and series of labels.

Parameters

- **X** (`ndarray (time_series_length, n_features)`) – A time series of feature vectors
- **y** (`ndarray (time_series_length,)`) – A time series of labels
- **window_size** (`int`) – Integer odd number
- **step** (`int`) – Step of sliding window

Returns

- **new_X** (`ndarray (n_samples, n_timesteps, n_features)`) – Input for recurrent neural network
- **new_y** (`ndarray (n_samples, n_timesteps, output_dim)`) – Time series of labels

Searching

<code>mslc</code>	Given a length n real sequence, finds the consecutive subsequence of length at most U with the maximum sum in O(n) time.
-------------------	--

Continued on next page

Table 3.5 – continued from previous page

<code>longest_segment</code>	Find the longest subsequence which scores above a given threshold in $O(n)$
<code>max_avg_seg</code>	Given a length n real sequence, finding the consecutive subsequence of length at least L with the maximum average can be done in $O(n \log L)$ time.

sciquence.sequences.ms1c

`sciquence.sequences.ms1c()`

Given a length n real sequence, finds the consecutive subsequence of length at most U with the maximum sum in $O(n)$ time.

Parameters

- **A** (*list of float*) – List of float numbers
- **U** (*int*) – Sum upper bound

Returns `ln_pointers` – List of left-negative pointers

Return type list of int

References

Lin Y.L., Jiang T., Chaoc K.M. (2002).

Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis

http://www.csie.ntu.edu.tw/~kmchao/papers/2002_jcss.pdf

sciquence.sequences.longest_segment

`sciquence.sequences.longest_segment()`

Find the longest subsequence which scores above a given threshold in $O(n)$

Parameters

- **sequence** (*ndarray*) – A sequence
- **alpha** (*float*) – Floating-point threshold being a lower bound for searched segment

Returns `segment` – The longest segment with sum above given threshold

Return type ndarray

Examples

```
>>> from sciquence.sequences import longest_segment
>>> import numpy as np
>>> X = np.array([-1, -2, -3, -23, -45, -3, -4, 5, -56, 67, 1, 3, 4, 5])
>>> ls1 = longest_segment(X, 30)
>>> print ls1, sum(ls1)
[67 1 3 4 5] 80
# Next, we change -56 into -50
>>> Z = np.array([-1, -2, -3, -23, -45, -3, -4, 5, -50, 67, 1, 3, 4, 5])
```



```
>>> ls2 = longest_segment(Z, 30)
[-4  5 -50  67  1  3  4  5] 31
```

Notes

Keep in mind that this algorithm maximizes segment length, not the segment total sum.

References

Csűrös M. (2008).

A linear-time algorithm for finding the longest segment which scores above a given threshold

<https://arxiv.org/pdf/cs/0512016.pdf>

sciquence.sequences.max_avg_seq

sciquence.sequences.max_avg_seq()

Given a length n real sequence, finding the consecutive subsequence of length at least L with the maximum average can be done in $O(n \log L)$ time. In other words, function maximizes subsequence average, keeping its length equal or greater given value L

Parameters

- **A** (*ndarray*) – List of float numbers
- **L** (*int*) – Minimal subsequence length

Returns

- **start** (*int*) – First slice index of found subsequence
- **stop** (*int*) – Second slice index of found subsequence

Examples

```
>>> from sciquence.sequences import max_avg_seq
>>> import numpy as np
>>> X = np.array([-1, -2, -3, -23, -45, -3, -4, 5, 50, 67, 1, 3, 4, 5])
>>> max_avg_seq(X, 3)
(7, 10)
>>> print X[7:10]
[ 5 50 67]
# We change 50 into -50
>>> Z = np.array([-1, -2, -3, -23, -45, -3, -4, 5, -50, 67, 1, 3, 4, 5])
(9, 12)
>>> print Z[9:12]
[67  1  3]
# In last example, we replace -3 with 600
>>> V = np.array([-1, -2, 600, -23, -45, -3, -4, 5, -50, 67, 1, 3, 4, 5])
(0, 3)
>>> print V[0:3]
[-1 -2 600]
```

References

Lin Y.L., Jiang T., Chao K.M. (2002). *Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis*

http://www.csie.ntu.edu.tw/~kmchao/papers/2002_jcss.pdf

sciquence.dtw

Similarities

<code>dtw</code>	Measure similarities between two sequences.
<code>segmental_dtw</code>	Find similarities between two sequences.

sciquence.dtw.dtw

`sciquence.dtw.dtw()`

Measure similarities between two sequences.

When computing Dynamic Time Warping path, we are looking for the lowest cost path from (0, 0) to (len(A), len(B) point).

Parameters

- **A** (`np.ndarray (a_rows, n_columns)`) – A sequence
- **B** (`np.ndarray (b_rows, n_columns)`) – A sequence
- **metric** (`function(np.ndarray, np.ndarray)`) – A distance function with two parameters, which returns double

Returns

- **warping_path** (`list of tuple`) – Points of warping path
- **distance** (`double`) – A distance between two sequences

Examples

```
>>> from sciquence.dtw import dtw
>>> import numpy as np
>>> from scipy.spatial.distance import cosine
>>> A = np.random.rand(5, 3)
>>> B = np.random.rand(8, 3)
>>> warp_path, distance = dtw(A, B, cosine)
```

References

sciquence.dtw.segmental_dtw

`sciquence.dtw.segmental_dtw()`

Find similarities between two sequences.

Segmental DTW algorithm extends ide of Dynamic Time Warping method, and looks for the best warping path not only on the main diagonal, but also on the other. It facilitates performing not only the comparison of the whole sequences, but also discovering similarities between subsequences of given sequences A and B.

Parameters

- **A**(*ndarray* (*n_samples*, *n_features*)) – First sequence
- **B**(*ndarray* (*n_samples*, *n_features*)) – Second sequence
- **min_path_len**(*int*) – Minimal length of path
- **metric**(*str*) – Metric name

Returns **matchings** – List of matching sequences

Return type list of list of tuple

See also:

dtw ()

References

Park A. S. (2006).

Unsupervised Pattern Discovery in Speech: Applications to Word Acquisition and Speaker Segmentation

https://groups.csail.mit.edu/sls/publications/2006/Park_Thesis.pdf

sciquence.sax_vsm

Symbolic Aggregate Approximation - Vector Space Model

SAX_VSM

sciquence.sliding_window

Generators

<i>wingen</i> (X, window_size[, step])	Generate subsequences from a single sequence.
<i>raw_wingen</i> (X, window_size[, step])	Generate subsequences from a single sequence.
<i>multi_wingen</i> (X, window_size, step)	Generate subsequences from a single sequence.

sciquence.sliding_window.wingen

`sciquence.sliding_window.wingen` (*X*, *window_size*, *step=1*)

Generate subsequences from a single sequence. Generator usage reduces memory consumption.

Parameters

- **X**(*ndarray* (*n_samples*, *n_features*)) – Array of size
- **window_size**(*int*) – Size of sliding window

- **step** (*int*) – Size of sliding window step

Yields subsequence (*ndarray (window_size, n_features)*) – Subsequence from X sequence

Examples

```
>>> from sciquence.sliding_window import wingen
>>> import numpy as np
>>> X = np.array([[1, 2, 3,],
>>>                [11, 12, 13],
>>>                [21, 22, 23],
>>>                [31, 32, 33]])
>>> print wingen(X, 2, 1).next()
>>> [[ 1  2  3]
>>>  [11 12 13]]
```

sciquence.sliding_window.raw_wingen

sciquence.sliding_window.**raw_wingen** (*X, window_size, step=1*)

Generate subsequences from a single sequence. Generator usage reduces memory consumption.

Parameters

- **X** (*ndarray (n_samples, n_features)*) – Array of size
- **window_size** (*int*) – Size of sliding window
- **step** (*int*) – Size of sliding window step

Yields subsequence (*ndarray (window_size, n_features)*) – Subsequence from X sequence

Examples

```
>>> from sciquence.sliding_window import wingen
>>> import numpy as np
>>> X = np.array([[1, 2, 3,],
>>>                [11, 12, 13],
>>>                [21, 22, 23],
>>>                [31, 32, 33]])
>>> print wingen(X, 2, 1).next()
>>> [[ 1  2  3]
>>>  [11 12 13]]
```

sciquence.sliding_window.multi_wingen

sciquence.sliding_window.**multi_wingen** (*X, window_size, step*)

Generate subsequences from a single sequence. Generator usage reduces memory consumption.

Parameters

- **X** (*ndarray (n_sequences, n_samples, n_features)*) – or list of ndarray (*n_samples, n_features*) 3-dimensional array or list of sequences
- **window_size** (*int*) – Size of sliding window

- **step** (*int*) – Size of sliding window step

Yields subsequence (*ndarray (window_size, n_features)*) – Subsequence from X sequence

Seekers

<code>Seeker(estimator[, train_pipeline, ...])</code>	
<code>SlidingWindow([window_size, shift])</code>	Class dedicated for simple sliding window transformations.

sciquence.sliding_window.Seeker

class sciquence.sliding_window.**Seeker** (*estimator, train_pipeline=None, test_pipeline=None, window_sizes=[3], windows_shifts=[1], bin_threshold=0.5, proba_combiner=None*)

`__init__` (*estimator, train_pipeline=None, test_pipeline=None, window_sizes=[3], windows_shifts=[1], bin_threshold=0.5, proba_combiner=None*)

Methods

<code>__init__</code> (<i>estimator[, train_pipeline, ...]</i>)	
<code>fit(X, y)</code>	Transform train set with the train_pipeline and fit passed estimator
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	
<code>predict_log_proba(X)</code>	
<code>predict_proba(X)</code>	
<code>set_params(**params)</code>	Set the parameters of this estimator.

sciquence.sliding_window.SlidingWindow

class sciquence.sliding_window.**SlidingWindow** (*window_size=5, shift=1*)

Class dedicated for simple sliding window transformations. It transforms only the one input sequence: for parallel X and y vectors processing see WindowsMaker

Parameters

- **window_size** (*int, default 5*) – Size of sliding window
- **shift** (*int*) – Shift of sliding window, default 1

window_size_
int – Size of sliding window

shift_
int – Shift of sliding window

Examples

```
>>> from sciquence.sliding_window import SlidingWindow
>>> import numpy as np
```

```
>>> X = np.array([0, 1, 23, 3, 4, 67, 89, 11, 2, 34])
>>> print SlidingWindow(window_size=3, shift=4).transform(X)
[array([1, 2, 3]), array([ 5, 11, 12])]
```

References

Dietterich Thomas G. Machine Learning for Sequential Data: A Review

<http://web.engr.oregonstate.edu/~tgd/publications/mlsd-ssspr.pdf>

See also:

WindowsMaker

`__init__` (*window_size=5, shift=1*)

Methods

<code>__init__</code> (<i>window_size, shift</i>)	
<code>fit</code> (<i>X</i> , <i>y</i>)	Mock method, does nothing.
<code>fit_transform</code> (<i>X</i> , <i>y</i>)	Fit to data, then transform it.
<code>get_params</code> (<i>[deep]</i>)	Get parameters for this estimator.
<code>set_params</code> (<i>**params</i>)	Set the parameters of this estimator.
<code>transform</code> (<i>X</i> , <i>y</i>)	Transform sequence into sequence of contextual sliding_window

sciquence.representation

Piecewise Aggregate Approximation

<code>paa</code> (<i>sequence, window</i> , <i>adjust</i>)	Piecewise Aggregate Approximation
--	-----------------------------------

sciquence.representation.paa

`sciquence.representation.paa` (*sequence, window, adjust=True*)

Piecewise Aggregate Approximation

PAA is a method of time series representation. Every time point in the time series is quantized into the mean value in the given time range of length *N*.

Parameters

- **sequence** (*ndarray (n_timesteps, 1)*) – A sequence
- **window** (*int*) – Window length
- **adjust** (*bool, default True*) – Adjust size

Returns `paa_representation` – PAA representation of input sequence

Return type `ndarray`

Examples

```
>>> import numpy as np
>>> from sciquence.representation import paa
>>> np.random.seed(42)
>>> random_time_series = np.random.rand(50)
>>> print paa(random_time_series, window=10)
[ 0.52013674  0.39526784  0.40038724  0.50927069  0.40455702]
```

References

Symbolic Aggregate Approximation

<code>sax(sequence, window[, alphabet_size, adjust])</code>	Symbolic Aggregate Approximation.
---	-----------------------------------

sciquence.representation.sax

sciquence.representation.**sax** (*sequence*, *window*, *alphabet_size=5*, *adjust=True*)
 Symbolic Aggregate Approximation.

Transform time series into a string.

Parameters

- **sequence** (*ndarray*) – One-dimensional numpy array of arbitrary length
- **window** (*int*) – Length of sliding window
- **alphabet_size** (*int*) – Number of Gaussian breakpoints
- **adjust** (*bool*, *default True*) – Compute only for equal-size chunks

Returns `sax_representation` – A SAX representation

Return type `str`

Examples

```
>>> import numpy as np
>>> from sciquence.representation import sax
>>> np.random.seed(42)
>>> random_time_series = np.random.rand(50)
>>> print sax(random_time_series, 10, alphabet_size=5)
dcccc
```

References

sciquence.text_processing

Text to vector

<i>WordEncoder()</i>	Class used for for transforming text data into
----------------------	--

sciquence.text_processing.WordEncoder

class sciquence.text_processing.**WordEncoder**
 Class used for for transforming text data into word indices

`__init__()`

Methods

<code>__init__()</code>	
<code>fit(X[, y])</code>	Fit WordEncoder object
<code>fit_transform(X[, y])</code>	Fit WordEncoder and transform list of tokenized sentences
<code>inverse_transform(X)</code>	'
<code>partial_fit(X[, y])</code>	Partially fit WordEncoder to the given word set
<code>transform(X[, y])</code>	Transform list of tokenized sentences (or raw text) into lists of indices

sciquence.data_structures

Text to vector

<i>MultiDict(*args, **kwds)</i>	A dictionary, which allows to get multiple elements at once
---------------------------------	---

sciquence.data_structures.MultiDict

class sciquence.data_structures.**MultiDict** (*args, **kwds)
 A dictionary, which allows to get multiple elements at once

Examples

```
>>> md = MultiDict([('a', 1), ('b', 2), ('c', 3)])
>>> print md[['c', 'a']]
(3, 1)
```

`__init__` (*args, **kwds)

Initialize an ordered dictionary. The signature is the same as regular dictionaries, but keyword arguments are not recommended because their insertion order is arbitrary.

Methods

<code>__init__(*args, **kwargs)</code>	Initialize an ordered dictionary.
<code>clear()</code> -> None. Remove all items from od.)	
<code>copy()</code> -> a shallow copy of od)	
<code>fromkeys(S, ...)</code>	If not specified, the value defaults to None.
<code>get((k,d))</code> -> D[k] if k in D, ...)	
<code>has_key(k)</code> -> True if D has a key k, else False)	
<code>items()</code> -> list of (key, value) pairs in od)	
<code>iteritems()</code>	od.iteritems -> an iterator over the (key, value) pairs in od
<code>iterkeys()</code> -> an iterator over the keys in od)	
<code>itervalues()</code>	od.itervalues -> an iterator over the values in od
<code>keys()</code> -> list of keys in od)	
<code>pop((k,d))</code> -> v, ...)	value. If key is not found, d is returned if given, otherwise KeyError
<code>popitem()</code> -> (k, v), ...)	Pairs are returned in LIFO order if last is true or FIFO order if false.
<code>setdefault((k,d))</code> -> od.get(k,d), ...)	
<code>update((E, ...)</code>	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
<code>values()</code> -> list of values in od)	
<code>viewitems(...)</code>	
<code>viewkeys(...)</code>	
<code>viewvalues(...)</code>	

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

Symbols

`__init__()` (sciquence.data_structures.MultiDict method), 20

`__init__()` (sciquence.sliding_window.Seeker method), 17

`__init__()` (sciquence.sliding_window.SlidingWindow method), 18

`__init__()` (sciquence.text_processing.WordEncoder method), 20

C

`chunk()` (in module sciquence.sequences), 8

D

`dtw()` (in module sciquence.dtw), 14

L

`longest_segment()` (in module sciquence.sequences), 12

`lseq_equal()` (in module sciquence.sequences), 8

M

`max_avg_seq()` (in module sciquence.sequences), 13

`mslc()` (in module sciquence.sequences), 12

`multi_wingen()` (in module sciquence.sliding_window), 16

MultiDict (class in sciquence.data_structures), 20

N

`nseq()` (in module sciquence.sequences), 6

`nseqi()` (in module sciquence.sequences), 7

P

`paa()` (in module sciquence.representation), 18

`parallel_sort()` (in module sciquence.sequences), 10

`pseq()` (in module sciquence.sequences), 6

`pseqi()` (in module sciquence.sequences), 8

R

`raw_wingen()` (in module sciquence.sliding_window), 16

`rnn_input()` (in module sciquence.sequences), 11

S

`sax()` (in module sciquence.representation), 19

Seeker (class in sciquence.sliding_window), 17

`segmental_dtw()` (in module sciquence.dtw), 14

`seq()` (in module sciquence.sequences), 5

`seq2seq_input()` (in module sciquence.sequences), 11

`seqi()` (in module sciquence.sequences), 7

`shapes_equal()` (in module sciquence.sequences), 9

`shift_` (sciquence.sliding_window.SlidingWindow attribute), 17

`size_equal()` (in module sciquence.sequences), 9

SlidingWindow (class in sciquence.sliding_window), 17

`specseq()` (in module sciquence.sequences), 6

W

`window_size_` (sciquence.sliding_window.SlidingWindow attribute), 17

`wingen()` (in module sciquence.sliding_window), 15

WordEncoder (class in sciquence.text_processing), 20