sciquence Documentation

Release 0.1.0

Krzysztof Joachimiak

Jul 19, 2019

User documentation

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
```

CHAPTER 3

sciquence API

3.1 sciquence.sequences

3.1.1 Cutting

seq(array)	Cut input array into sequences consisting of the same elements
nseq(array)	Returns sequences consisting of zeros
pseq(array)	Returns sequences consisting of ones
specseq(array, element)	Return sequences consisting of specific tag
seqi(array)	Get list of sequences and corresponding list of indices
nseqi(array)	Get list of negative sequences indices (consisting of zeroes)
pseqi(array)	Get list of positive sequences indices (consisting of ones)
specseqi(array, elem)	Get list of sequences indices, consisting of specific element
chunk(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.nseq(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

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

Returns

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

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

sciquence.sequences.nseqi

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

Get list of negative sequences indices (consisting of zeroes)

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

Returns

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

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.seqi(x)
[array([ 6,  7,  8,  9, 10, 11]), array([17, 18, 19, 20])]
```

sciquence.sequences.pseqi

sciquence.sequences.**pseqi**(*array*)

Get list of positive sequences indices (consisting of ones)

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

Returns

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

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.seqi(x)
[array([0, 1, 2, 3, 4, 5]), array([12, 13, 14, 15, 16])]
```

sciquence.sequences.specseqi

sciquence.sequences.**specseqi**(*array*, *elem*)

Get list of sequences indices, consisting of specific element

Parameters

- **array** (*ndarray*) – Numpy array
- **elem** (*object*) – A sequence element

Returns

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

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.seqi(x)
[array([13, 14, 15, 16, 17])]
```

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])]
```

3.1.2 Comparing

lseq_equal(lseqa, lseqb)	Compare two lists of ndarrays
shapes_equal(*arrays)	Check if all the arrays have the same shape.
size_equal(*arrays, **kwargs)	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
```

(continues on next page)

(continued from previous page)

```
>>> 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)
```

(continues on next page)

(continued from previous page)

```
True
>>> sq.shapes_equal(x, y, z, v)
True
```

3.1.3 Sampling

random_slice(array_len, slice_length)	Choose a random slice of given length

sciquence.sequences.random_slice

sciquence.sequences.**random_slice**(*array_len*, *slice_length*)

Choose a random slice of given length

Parameters

- **array_len** (*int*) – Array length
- **slice_length** (*int*) – Length of subsequence

Returns **slice** – A subsequence slice

Return type slice

Examples

```
>>> import numpy as np
>>> import sciquence.sequences as sq
>>> print sq.random_slice(54, 6)
slice(15, 21, None)
```

3.1.4 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, parallely 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]])]
```

3.1.5 Sliding window

wingen(X, window_size[, step, raw])	Generate subsequences from a single sequence.

sciquence.sequences.wingen

sciquence.sequences.**wingen**(*X*, *window_size*, *step=1*, *raw=False*)

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
- **raw** (*bool*) – If true, the last window will be yielded even if shorter than

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

Examples

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

3.1.6 Searching

mslc	Given a length n real sequence, finds the consecutive subsequence of length at most U with the maximum sum in O(n) time.
longest_segment	Find the longest subsequence which scores above a given threshold in O(n)
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.

sciquence.sequences.mslc

sciquence.sequences.**mslc**()

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)
```

(continues on next page)

(continued from previous page)

```
[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)
```

(continues on next page)

(continued from previous page)

```
>>> print V[0:3]
[ -1  -2 600]
```

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

3.2 sciquence.similarities

3.2.1 Similarities

dtw(A, B, metric)	Measure similarities between two sequences.
segmental_dtw	Find similarities between two sequences.

sciquence.similarities.dtw

sciquence.similarities.**dtw**(*A*, *B*, *metric*)

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.similarities.segmental_dtw

sciquence.similarities.**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 comparision 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

3.3 sciquence.postprocessing

3.3.1 Binarization

ClasswiseBinarizer(thresholds)	Performing binarization classwise.
binarize_classwise(X, thresholds)	Binarization performed classwise.

sciquence.postprocessing.ClasswiseBinarizer

class sciquence.postprocessing.**ClasswiseBinarizer**(*thresholds*)

Performing binarization classwise.

It may be used for binarize independently multiple class in the tagging tasks.

Parameters **thresholds** (*list of float or numpy.ndarray*) – Binarization thresholds for all the classes

__init__(*thresholds*)

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(thresholds)	x.__init__(...) initializes x; see help(type(x)) for signature
fit(X[, y])	Does nothing
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.
transform(X[, y, copy])	Perform classwise binarization, i.e.

sciquence.postprocessing.binarize_classwise

sciquence.postprocessing.**binarize_classwise**(*X*, *thresholds*)

Binarization performed classwise.

Parameters

- **X** (*numpy.ndarray*) – Probabilities vector
- **thresholds** (*list of float or numpy.ndarray*) – Binarization thresholds for all the classes

Examples

```
>>> import numpy as np
>>> X = np.array(
>>> [[ 0.04344385  0.24317802  0.81423947],
>>> [ 0.30503777  0.08385118  0.48402043],
>>> [ 0.38695257  0.64501778  0.19023201],
>>> [ 0.49452506  0.35440145  0.74149338],
>>> [ 0.25147325  0.14294654  0.6648142 ],
>>> [ 0.99852846  0.75026559  0.43106003],
>>> [ 0.33369685  0.41158767  0.86865335],
>>> [ 0.07741532  0.90428353  0.87152301],
>>> [ 0.79609158  0.47617837  0.1890651 ],
>>> [ 0.14287567  0.52800364  0.10957203]]
>>> )
>>> X_binarized = ClasswiseBinarizer(thresholds=[.5, .4, .3]).transform(X)
>>> print X_binarized
>>> [[ 0.  0.  1.],
>>> [ 0.  0.  1.],
>>> [ 0.  1.  0.],
>>> [ 0.  0.  1.],
>>> [ 0.  0.  1.],
>>> [ 1.  1.  1.],
>>> [ 0.  1.  1.],
>>> [ 0.  1.  1.],
>>> [ 1.  1.  0.],
>>> [ 0.  1.  0.]]
```

3.4 sciquence.representation

3.4.1 Piecewise Aggregate Approximation

paa(sequence, window[, adjust])	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** (*numpy.ndarray*) – A sequence (n_timesteps, 1)
- **window** (*int*) – Window length
- **adjust** (*bool*) – Adjust size. Default: True

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

3.4.2 Symbolic Aggregate Approximation

sax(sequence, window[, alphabet_size, adjust])	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** (*numpy.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

3.5 sciquence.text_processing

3.5.1 Text to vector

Word2Idx()	Class used for for transforming text data into word indices

sciquence.text_processing.Word2Idx

class sciquence.text_processing.**Word2Idx**

Class used for for transforming text data into word indices

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__()	x.__init__(...) initializes x; see help(type(x)) for signature
fit(X[, y])	Fit WordEncoder object
fit_transform(X[, y])	Fit WordEncoder and transform list of tokenized sentences (or raw text) into lists of indices
inverse_transform(X)	’
partial_fit(X[, y])	Partially fit WordEncoder to the given word set
transform(X[, y])	Transform list of tokenized sentences (or raw text) into lists of indices

3.6 sciquence.data_structures

3.6.1 Text to vector

MultiDict(**kwds)	A dictionary, which allows to get multiple elements at once

sciquence.data_structures.MultiDict

class sciquence.data_structures.**MultiDict**(***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__(***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

__init__(**kwds)	Initialize an ordered dictionary.
clear()	
copy()	
fromkeys(S[, v])	If not specified, the value defaults to None.
get(k[,d])	
has_key(k)	
items()	
iteritems()	od.iteritems -> an iterator over the (key, value) pairs in od
iterkeys()	
itervalues()	od.itervalues -> an iterator over the values in od
keys()	
pop(k[,d])	value.
popitem()	Pairs are returned in LIFO order if last is true or FIFO order if false.
setdefault(k[,d])	
update([E,]**F)	If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v
values()	
viewitems()	
viewkeys()	
viewvalues()	

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

C

D

L

M

N

P

R

S

W