khiva Documentation

Release v0.5.0

Shapelets.io

Apr 30, 2020

Contents:

CHAPTER 1

Quick Start

1.1 Install Khiva

First of all, the Khiva C++ library should be installed:

Shapelets.

Then, install the compiled Khiva package that is hosted on the Python Package Index (PyPI) with pip:

```
pip install khiva
```

1.2 Dive in

Dive quickly into Khiva with the following example: First, set the backend and device you want to use. There is a backend and a device set by default:

```
from khiva.library import *
set_backend(KHIVABackend.KHIVA_BACKEND_OPENCL)
set_device(0)
```

Then, you can create an array in the device:

```
from khiva.array import *
a = Array.from_list([1, 2, 3, 4, 5, 6, 7, 8], dtype.s32)
a.display()
```

The previous lines print the dimensions and the content of the created array:

[8 1 1 1]
1.0000
2.0000
3.0000
4.0000
5.0000
6.0000
7.0000
8.0000

Once the array is created in device memory, we can concatenate operations with this array in an asynchronous way and receive the data only in the host when *to_list()*, *to_numpy()* or *to_pandas()* (the latter only supports bi-dimensional time series) functions are called.

```
a = a.to_pandas()
print(a)
```

The result is the next one:

	0
1	1.0
2	2.00
3	3.00
4	4.00
5	5.00
6	6.00
7	7.00
8	8.00

Now, let's dive into the asynchronous usage of the library. Khiva library provides us several time series analysis functionalities which include features extraction, time-series re-dimension, distance calculations, motifs and discords detection, tools for similarity study, statistical parameters extraction or time series normalization.

All these functionalities can be concatenated to improve the performance, so you can get the data just in the moment that you do not use the functions of this library:

```
from khiva.matrix import *
stomp_result = stomp(Array.from_numpy(np.array([11, 10, 10, 10, 10, 10, 10, 10, 10,␣
↪10, 10, 11]), dtype.s32),
                     Array.from_numpy(np.array([9, 10, 10, 10, 10, 10, 10, 10, 10,
↪ 10, 10, 9]), dtype.s32),
                     3)
find_best_n_discords_result = find_best_n_discords(stomp_result[0],
                                                   stomp_result[1], 2)
a = find_best_n_discords_result[2].to_numpy()
print(a)
```

The previous produces the following output:

```
[1.73190141 1.73185158] [8 8] [0 9]
```

The first numpy array represents the minimum distances between the subsequences of length 3 between the two time-series. The second numpy array represents the location of those subsequences in the first time-series and the third one represents the indices in the second time-series.

We want to highlight the possibility of using the library for computing the functions in different backends and with different devices, knowing that the operations should be executed in the same device where the array was created.

```
#Adding operations in the different backends and devices.
from khiva.features import *
set_backend(KHIVABackend.KHIVA_BACKEND_OPENCL)
set_device(0)
a = Array.from_list([1, 2, 3, 4, 5, 6, 7, 8], dtype.s32)
b = mean(a)

set_device(1)
c = Array.from_list([1, 2, 3, 4, 5, 6, 7, 8], dtype.s32)
d = mean(c)

set_backend(KHIVABackend.KHIVA_BACKEND_CPU)
set_device(0)
e = Array([1, 2, 3, 4, 5, 6, 7, 8])
f = mean(e)

#Retrieving the results of the previous operations
set_backend(KHIVABackend.KHIVA_BACKEND_OPENCL)
set_device(0)
print(b.to_numpy())

set_device(1)
print(d.to_numpy())

set_backend(KHIVABackend.KHIVA_BACKEND_CPU)
set_device(0)
print(f.to_numpy())
```

The output is the next one:

4.5
4.5
4.5

Note that the data type used by default is floating point of 32 bits in order to avoid problems with the different devices, but it can be changed deliberately.

The available data types are the next ones:

Data type	Explanation
f32	32 bits Float
c32	32 bits Complex
f64	64 bits Double
c64	64 bits Complex
b8	8 bits Boolean
s32	32 bits Int
32u	32 bits Unsigned Int
u8	8 bits Unsigned Int
s64	64 bits Int
u64	64 bits Unsigned Int
s16	16 bits Int
u16	16 bits Unsigned Int

There are functions that do not support 32 bits floating point data type, so it is necessary to indicate the data type. The following is an example function requiring a 32-bit signed integer array:

```
cwt_coefficients_result = cwt_coefficients(Array.from_list([[0.1, 0.2, 0.3], [0.1, 0.
↪2, 0.3]], dtype.s32),
                                           Array.from_list([1, 2, 3], dtype.s32), 2,␣
↪2).to_numpy()
print(cwt_coefficients_result)
```

The output is:

```
[0.26517162 0.26517162]
```

1.3 Limitations

This open-source library provides a very good performance, but it has got memory limitations. For cases where you need to apply a time series analysis over a huge amount of data and in short-term fashion, please, contact us).

1.4 Let's Rock!

Now, you have the basic concepts to start using the library. Please, follow the documentation of each function to know how to use them. Each function has its corresponding tests so you can check how to use each of them.

Furthermore, we provide use cases and examples that you can use to learn where and how to apply the library.

CHAPTER 2

Khiva

2.1 khiva

2.1.1 Submodules

2.1.1.1 khiva.array

class khiva.array.**Array**(*array_reference*)

Bases: object

as_type(*dtype*)

Converts the array to a desired array with a desired type.

Parameters **dtype** – The desired KHIVA data type.

Returns An array with the desired data type.

copy()

Performs a deep copy of the array.

return: An identical copy of self.

display()

Displays the data stored in the KHIVA array.

static from_arrayfire(*arrayfire*)

Creates a KHIVA array from an array of ArrayFire. This method increments the reference count of the ArrayFire's array passed.

Parameters **arrayfire** – An ArrayFire array.

Returns a KHIVA array.

static from_list(*input_list*, *khiva_type*)

Creates a KHIVA array from a Python list.

Parameters

- **input_list** – A Python list.
- **khiva_type** – The KHIVA type of the elements of the list.

Returns a KHIVA array.

static from_numpy(*array*, *khiva_type*)
Creates a KHIVA array from a Pandas dataframe.

Parameters

- **input_list** – A Numpy multidimensional array.
- **khiva_type** – The KHIVA type of the elements of the Pandas dataframe.

Returns a KHIVA array.

static from_pandas(*dataframe*, *khiva_type*)
Creates a KHIVA array from a Pandas dataframe.

Parameters

- **input_list** – A Pandas dataframe.
- **khiva_type** – The KHIVA type of the elements of the Pandas dataframe.

Returns a KHIVA array.

get_col(*index*)
Gets a desired column.

Parameters **index** – Index of the desired column.

Returns The desired column.

get_cols(*first*, *last*)
Gets a sequence of columns using the first column index and the last column index, both columns included.

Parameters

- **first** – First column of the subsequence of columns.
- **last** – Last column of the subsequence of columns.

Returns A subsequence of columns between 'first' and 'last'.

get_dims()
Gets the dimensions of the KHIVA array.

Returns The dimensions of the KHIVA array.

get_row(*index*)
Gets a desired row.

Parameters **index** – Index of the desired row.

Returns The desired row.

get_rows(*first*, *last*)
Gets a sequence of rows using the first row index and the last row index, both rows included.

Parameters

- **first** – First row of the subsequence of rows.
- **last** – Last row of the subsequence of rows.

Returns A subsequence of rows between 'first' and 'last'.

get_type()
Gets the type of the KHIVA array.

Returns The type of the KHIVA array.

join(*dim*, *other*)
Joins the first and second KHIVA arrays along the specified dimension. :param dim: The dimension along which the join occurs. :param other: The second input array. :return: KHIVA Array with the result of this operation.

matmul(*other*)
Matrix multiplication.

Parameters **other** – KHIVA Array

Returns The matrix multiplication between these two KHIVA Arrays.

to_arrayfire()
Creates an Arrayfire array from this KHIVA array. This need to be used carefully as the same array reference is going to be used by both of them. Once the Arrayfire array is created, the destructor of the KHIVA array is not going to free the allocated array.

Returns an Arrayfire Array

to_list()
Converts the KHIVA array to a list.

Returns KHIVA array converted to list.

to_numpy()
Converts the KHIVA array to a numpy array.

The returned numpy array shape matches the Array dimensions as follows:

- For an Array with dims equal to [4, 2, 1, 1] the numpy shape will be (2, 4).
- For an Array with dims equal to [4, 3, 2, 1] the numpy shape will be (2, 3, 4).
- For an Array with dims equal to [4, 1, 2, 3] the numpy shape will be (3, 2, 1, 4).

Returns KHIVA array converted to numpy.array.

to_pandas()
Converts the KHIVA array to a pandas data frame.

Returns KHIVA array converted to a pandas data frame.

transpose(*conjugate=False*)
Transpose the KHIVA Array.

Parameters **conjugate** – Indicates if the transpose is conjugated or not.

Returns The transposed KHIVA Array.

class khiva.array.**dtype**
Bases: enum.Enum

KHIVA array available types.

b8 = 4
Boolean. khiva.dtype

c32 = 1
32 bits Complex. khiva.dtype

c64 = 3
64 bits Complex. khiva.dtype

f32 = 0
Float. khiva.dtype

f64 = 2
64 bits Double. khiva.dtype

s16 = 10
16 bits Int. khiva.dtype

s32 = 5
32 bits Int. khiva.dtype

s64 = 8
64 bits Integer. khiva.dtype

u16 = 11
16 bits Unsigned int. khiva.dtype

u32 = 6
32 bits Unsigned Int. khiva.dtype

u64 = 9
64 bits Unsigned Int. khiva.dtype

u8 = 7
8 bits Unsigned Int. khiva.dtype

2.1.1.2 khiva.dimensionality

khiva.dimensionality.**paa**(*a*, *bins*)
Piecewise Aggregate Approximation (PAA) approximates a time series X of length n into vector $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_M)$ of any arbitrary length $M \leq n$ where each of $\bar{x}_i$ is calculated as follows:

$$\bar{x}_i = \frac{M}{n} \sum_{j=n/M(i-1)+1}^{(n/M)i} x_j.$$

Which simply means that in order to reduce the dimensionality from n to M, we first divide the original time series into M equally sized frames and secondly compute the mean values for each frame. The sequence assembled from the mean values is the PAA approximation (i.e., transform) of the original time series.

Parameters

- **a** – Set of points.
- **bins** – Sets the total number of divisions.

Returns KHIVA array of points with the reduced dimensionality.

khiva.dimensionality.**pip**(*a*, *number_ips*)
Calculates the number of Perceptually Important Points (PIP) in the time series.

[1] Fu TC, Chung FL, Luk R, and Ng CM. Representing financial time series based on data point importance. Engineering Applications of Artificial Intelligence, 21(2):277-300, 2008.

Parameters

- **a** – KHIVA array whose dimension zero is the length of the time series.
- **number_ips** – The number of points to be returned.

Returns KHIVA array with the most Perceptually Important number_ips.

khiva.dimensionality.**pla_bottom_up**(*ts*, *max_error*)

Applies the Piecewise Linear Approximation (PLA BottomUP) to the time series.

[1] Zhu Y, Wu D, Li Sh (2007). A Piecewise Linear Representation Method of Time Series Based on Feature Points. Knowledge-Based Intelligent Information and Engineering Systems 4693:1066-1072.

Parameters

- **ts** – Expects a khiva array containing the set of points to be reduced. The first component of the points in the first column and the second component of the points in the second column.
- **max_error** – The maximum approximation error allowed.

Returns The reduced number of points.

khiva.dimensionality.**pla_sliding_window**(*ts*, *max_error*)

Applies the Piecewise Linear Approximation (PLA Sliding Window) to the time series.

[1] Zhu Y, Wu D, Li Sh (2007). A Piecewise Linear Representation Method of Time Series Based on Feature Points. Knowledge-Based Intelligent Information and Engineering Systems 4693:1066-1072.

Parameters

- **ts** – Expects a khiva array containing the set of points to be reduced. The first component of the points in the first column and the second component of the points in the second column.
- **max_error** – The maximum approximation error allowed.

Returns The reduced number of points.

khiva.dimensionality.**ramer_douglas_peucker**(*a*, *epsilon*)

The Ramer–Douglas–Peucker algorithm (RDP) is an algorithm for reducing the number of points in a curve that is approximated by a series of points. It reduces a set of points depending on the perpendicular distance of the points and epsilon, the greater epsilon, more points are deleted.

[1] Urs Ramer, "An iterative procedure for the polygonal approximation of plane curves", Computer Graphics and Image Processing, 1(3), 244–256 (1972) doi:10.1016/S0146-664X(72)80017-0.

[2] David Douglas & Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), 112–122 (1973) doi:10.3138/FM57-6770-U75U-7727

Parameters

- **a** – KHIVA array with the x-coordinates and y-coordinates of the input points (x in column 0 and y in column 1).
- **epsilon** – It acts as the threshold value to decide which points should be considered meaningful or not.

Returns KHIVA array with the x-coordinates and y-coordinates of the selected points (x in column 0 and y in column 1).

khiva.dimensionality.**sax**(*a*, *alphabet_size*)

Symbolic Aggregate approXimation (SAX). It transforms a numeric time series into a time series of symbols with the same size. The algorithm was proposed by Lin et al.) and extends the PAA-based approach inheriting the original algorithm simplicity and low computational complexity while providing satisfactory sensitivity and selectivity in range query processing. Moreover, the use of a symbolic representation opened a door to the existing wealth of data-structures and string-manipulation algorithms in computer science such as hashing, regular expression, pattern matching, suffix trees, and grammatical inference.

[1] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003) A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. San Diego, CA. June 13.

Parameters

- **a** – KHIVA array with the input time series.
- **alphabet_size** – Number of element within the alphabet.

Returns KHIVA array of points with the reduced dimensionality.

khiva.dimensionality.**visvalingam**(*a*, *num_points*)

Reduces a set of points by applying the Visvalingam method (minimun triangle area) until the number of points is reduced to numPoints.

[1] M. Visvalingam and J. D. Whyatt, Line generalisation by repeated elimination of points, The Cartographic Journal, 1993.

Parameters

- **a** – KHIVA array with the x-coordinates and y-coordinates of the input points (x in column 0 and y in column 1).
- **num_points** – Sets the number of points returned after the execution of the method.

Returns KHIVA array with the x-coordinates and y-coordinates of the selected points (x in column 0 and y in column 1).

2.1.1.3 khiva.distances

khiva.distances.**dtw**(*tss*)

Calculates the Dynamic Time Warping Distance.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns Array with an upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

khiva.distances.**euclidean**(*tss*)

Calculates euclidean distances between time series.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns Array with an upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

khiva.distances.**hamming**(*tss*)

Calculates Hamming distances between time series.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns Array with an upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

khiva.distances.**manhattan**(*tss*)

Calculates Manhattan distances between time series.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns Array with an upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

khiva.distances.**sbd**(*tss*)

Calculates the Shape-Based distance (SBD). It computes the normalized cross-correlation and it returns the value that maximizes the correlation value between time series.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns Array with an upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

khiva.distances.**squared_euclidean**(*tss*)

Calculates the non squared version of the euclidean distance.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns Array with an upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

2.1.1.4 khiva.features

class khiva.features.**FftCoefficientResult**(*real*, *imag*, *abs*, *angle*)

Bases: tuple

abs
Alias for field number 2

angle
Alias for field number 3

imag
Alias for field number 1

real
Alias for field number 0

class khiva.features.**LinearTrendResult**(*pvalue*, *rvalue*, *intercept*, *slope*, *stdrr*)

Bases: tuple

intercept
Alias for field number 2

pvalue
Alias for field number 0

rvalue
Alias for field number 1

slope
Alias for field number 3

stdrr
Alias for field number 4

khiva.features.**abs_energy**(*arr*)
Calculates the sum over the square values of the time series.

Parameters **arr** (*khiva.array*) – KHIVA array with the time series.

Returns KHIVA array with the absEnergy.

khiva.features.**absolute_sum_of_changes**(*arr*)
Calculates the value of an aggregation function f_agg (e.g. var or mean) of the autocorrelation (Compare to http://en.wikipedia.org/wiki/Autocorrelation#Estimation), taken over different all possible lags (1 to length of x)

Parameters **arr** (*khiva.array*) – KHIVA array with the time series.

Returns KHIVA array with the absolute sum of changes.

khiva.features.**aggregated_autocorrelation**(*arr*, *aggregation_function*)
Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.

Parameters

- **arr** – A KHIVA array with the time series.
- **aggregation_function** – Function to be used in the aggregation. It receives an integer which indicates the function to be applied. 0 : mean, 1 : median 2 : min, 3 : max, 4 : stdev, 5 : var, default : mean

Returns KHIVA array that contains the aggregated correlation for each time series.

khiva.features.**aggregated_linear_trend**(*arr*, *chunk_size*, *aggregation_function*)
Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.

Parameters

- **arr** – A KHIVA array with the time series.
- **chunk_size** – The chunk size used to aggregate the data.
- **aggregation_function** – Function to be used in the aggregation. It receives an integer which indicates the function to be applied: 0 : mean, 1 : median 2 : min, 3 : max, 4 : stdev, default : mean

Returns (pvalue: KHIVA array with the pvalues for all time series. rvalue: KHIVA array with the rvalues for all time series. intercept: KHIVA array with the intercept values for all time series. slope: KHIVA array with the slope for all time series. stdrr: KHIVA array with the stderr values for all time series.)

khiva.features.**approximate_entropy**(*arr*, *m*, *r*)
Calculates a vectorized Approximate entropy algorithm. https://en.wikipedia.org/wiki/Approximate_entropy For short time-series this method is highly dependent on the parameters, but should be stable for N > 2000, see: Yentes et al. (2012) - The Appropriate Use of Approximate Entropy and Sample Entropy with Short Data Sets Other shortcomings and alternatives discussed in: Richman & Moorman (2000) - Physiological time-series analysis using approximate entropy and sample entropy

Parameters

- **arr** – A KHIVA array with the time series.
- **m** – Length of compared run of data.
- **r** – Filtering level, must be positive.

Returns KHIVA array with the vectorized approximate entropy for all the input time series in tss.

khiva.features.**auto_correlation**(*arr*, *max_lag*, *unbiased*)
Calculates the autocorrelation of the specified lag for the given time series.

Parameters

- **arr** – KHIVA array with the time series.
- **max_lag** – The maximum lag to compute.
- **unbiased** – Determines whether it divides by n - lag (if true) or n (if false).

Returns KHIVA array with the autocorrelation value for the given time series.

khiva.features.**auto_covariance**(*arr*, *unbiased=False*)
Calculates the auto-covariance the given time series.

Parameters

- **arr** – KHIVA array with the time series.
- **unbiased** – Determines whether it divides by n - lag (if true) or n (if false).

Returns KHIVA array with the auto-covariance value for the given time series.

khiva.features.**binned_entropy**(*arr*, *max_bins*)
Calculates the binned entropy for the given time series and number of bins.

Parameters

- **arr** – KHIVA array with the time series.
- **max_bins** – The number of bins.

Returns KHIVA array with the binned entropy value for the given time series.

khiva.features.**c3**(*arr*, *lag*)
Calculates the Schreiber, T. and Schmitz, A. (1997) measure of non-linearity for the given time series

Parameters

- **arr** – KHIVA array with the time series.
- **lag** – The lag.

Returns KHIVA array with non-linearity value for the given time series.

khiva.features.**cid_ce**(*arr*, *z_normalize*)
Calculates an estimate for the time series complexity defined by Batista, Gustavo EAPA, et al (2014). (A more complex time series has more peaks, valleys, etc.)

Parameters

- **arr** – KHIVA array with the time series.
- **z_normalize** – Controls wheter the time series should be z-normalized or not.

Returns KHIVA array with the complexity value for the given time series.

khiva.features.**count_above_mean**(*arr*)
Calculates the number of values in the time series that are higher than the mean.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the number of values in the time series that are higher than the mean.

khiva.features.**count_below_mean**(*arr*)
Calculates the number of values in the time series that are lower than the mean.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA Array with the number of values in the time series that are lower than the mean.

khiva.features.**cross_correlation**(*xss*, *yss*, *unbiased*)
Calculates the cross-correlation of the given time series.

Parameters

- **xss** – KHIVA array with the time series.
- **yss** – KHIVA array with the time series.
- **unbiased** – Determines whether it divides by n - lag (if true) or n (if false).

Returns KHIVA array with cross-correlation value for the given time series.

khiva.features.**cross_covariance**(*xss*, *yss*, *unbiased*)
Calculates the cross-covariance of the given time series.

Parameters

- **xss** – A KHIVA array with time series.
- **yss** – A KHIVA Array with time series.
- **unbiased** – Determines whether it divides by n - lag (if true) or n (if false).

Returns KHIVA array with the cross-covariance value for the given time series.

khiva.features.**cwt_coefficients**(*tss*, *widths*, *coeff*, *w*)
Calculates a Continuous wavelet transform for the Ricker wavelet, also known as the "Mexican hat wavelet".

Parameters

- **tss** – KHIVA array with the time series.
- **widths** – Widths. It accepts a list of lists or a numpy array with one or several widths.
- **coeff** – Coefficient of interest.
- **w** – Width of interest.

Returns KHIVA Array with the result of calculated coefficients.

khiva.features.**energy_ratio_by_chunks**(*arr*, *num_segments*, *segment_focus*)
Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series. segmentFocus should be lower than the number of segments.

Parameters

- **arr** – KHIVA array with the time series.
- **num_segments** – The number of segments to divide the series into.
- **segment_focus** – The segment number (starting at zero) to return a feature on.

Returns KHIVA array with the energy ratio by chunk of the time series.

khiva.features.**fft_aggregated**(*arr*)
Calculates the spectral centroid(mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the spectral centroid (mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.

khiva.features.**fft_coefficient**(*arr*, *coefficient*)
Calculates the fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast fourier transformation algorithm.

Parameters

- **arr** – KHIVA array with the time series.
- **coefficient** – The coefficient to extract from the FFT.

Returns Tuple with: real: KHIVA array with the real part of the coefficient. imag: KHIVA array with the imaginary part of the coefficient. abs: KHIVA array with the absolute value of the coefficient. angle: KHIVA array with the angle of the coefficient.

khiva.features.**first_location_of_maximum**(*arr*)
Calculates the first relative location of the maximal value for each time series.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the first relative location of the maximum value to the length of the time series, for each time series.

khiva.features.**first_location_of_minimum**(*arr*)
Calculates the first location of the minimal value of each time series. The position is calculated relatively to the length of the series.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the first relative location of the minimal value of each series.

khiva.features.**friedrich_coefficients**(*arr*, *m*, *r*)
Coefficients of polynomial $h(x)$, which has been fitted to the deterministic dynamics of Langevin model: Largest fixed point of dynamics $argmax_x h(x) = 0$ estimated from polynomial $h(x)$, which has been fitted to the deterministic dynamics of Langevin model:

$$\dot(x)(t) = h(x(t)) + R(N)(0, 1)$$

as described by [1]. For short time series this method is highly dependent on the parameters.

[1] Friedrich et al. (2000): Physics Letters A 271, p. 217-222 Extracting model equations from experimental data.

Parameters

- **arr** – KHIVA array with the time series.
- **m** – Order of polynom to fit for estimating fixed points of dynamics.
- **r** – Number of quantiles to use for averaging.

Returns KHIVA array with the coefficients for each time series.

khiva.features.**has_duplicate_max**(*arr*)
Calculates if the maximum within input time series is duplicated.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array containing True if the maximum value of the time series is duplicated and false otherwise.

khiva.features.**has_duplicate_min**(*arr*)
Calculates if the minimum of the input time series is duplicated.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array containing True if the minimum of the time series is duplicated and False otherwise.

khiva.features.**has_duplicates**(*arr*)
Calculates if the input time series contain duplicated elements.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array containing True if the time series contains duplicated elements and false otherwise.

khiva.features.**index_mass_quantile**(*arr*, *q*)
Calculates the index of the mass quantile.

Parameters

- **arr** – KHIVA array with the time series.
- **q** – The quantile.

Returns KHIVA array with the index of the mass quantile q.

khiva.features.**kurtosis**(*arr*)
Returns the kurtosis of tss (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the kurtosis of each time series.

khiva.features.**large_standard_deviation**(*arr*, *r*)
Checks if the time series within tss have a large standard deviation.

Parameters

- **arr** – KHIVA array with the time series.
- **r** – The threshold.

Returns KHIVA array containing True for those time series in tss that have a large standard deviation.

khiva.features.**last_location_of_maximum**(*arr*)
Calculates the last location of the maximum value of each time series. The position is calculated relatively to the length of the series.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the last relative location of the maximum value of each series.

khiva.features.**last_location_of_minimum**(*arr*)
Calculates the last location of the minimum value of each time series. The position is calculated relatively to the length of the series.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array the last relative location of the minimum value of each series.

khiva.features.**length**(*arr*)
Returns the length of the input time series.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array the length of tss.

khiva.features.**linear_trend**(*arr*)
Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.

Parameters **arr** – KHIVA array with the time series.

Return a tuple with pvalue: KHIVA array the pvalues for all time series. rvalue: KHIVA array The rvalues for all time series. intercept: KHIVA array the intercept values for all time series. slope:

KHIVA array the slope for all time series. stdrr: KHIVA array the stderr values for all time series.

khiva.features.**local_maximals**(*arr*)
Calculates all Local Maximals fot the time series in array.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the calculated local maximals for each time series in arr.

khiva.features.**longest_strike_above_mean**(*arr*)
Calculates the length of the longest consecutive subsequence in tss that is bigger than the mean of tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the length of the longest consecutive subsequence in the input time series that is bigger than the mean.

khiva.features.**longest_strike_below_mean**(*arr*)
Calculates the length of the longest consecutive subsequence in tss that is below the mean of tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the length of the longest consecutive subsequence in the input time series that is below the mean.

khiva.features.**max_langevin_fixed_point**(*arr*, *m*, *r*)
Largest fixed point of dynamics $argmax_x h(x) = 0$ estimated from polynomial $h(x)$, which has been fitted to the deterministic dynamics of Langevin model

$$\dot(x)(t) = h(x(t)) + R(N)(0, 1)$$

as described by

Friedrich et al. (2000): Physics Letters A 271, p. 217-222 *Extracting model equations from experimental data*

Parameters

- **arr** – KHIVA array with the time series.
- **m** – Order of polynom to fit for estimating fixed points of dynamics.
- **r** – Number of quantiles to use for averaging.

Returns KHIVA array with the largest fixed point of deterministic dynamics.

khiva.features.**maximum**(*arr*)
Calculates the maximum value for each time series within tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the maximum value of each time series within tss.

khiva.features.**mean**(*arr*)
Calculates the mean value for each time series within tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the mean value of each time series within tss.

khiva.features.**mean_absolute_change**(*arr*)
Calculates the mean over the absolute differences between subsequent time series values in tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the mean over the absolute differences between subsequent time series values.

khiva.features.**mean_change**(*arr*)
Calculates the mean over the differences between subsequent time series values in tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the mean over the differences between subsequent time series values.

khiva.features.**mean_second_derivative_central**(*arr*)
Calculates mean value of a central approximation of the second derivative for each time series in tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the mean value of a central approximation of the second derivative for each time series.

khiva.features.**median**(*arr*)
Calculates the median value for each time series within tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the median value of each time series within tss.

khiva.features.**minimum**(*arr*)
Calculates the minimum value for each time series within tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the minimum value of each time series within tss.

khiva.features.**number_crossing_m**(*arr*, *m*)
Calculates the number of m-crossings. A m-crossing is defined as two sequential values where the first value is lower than m and the next is greater, or viceversa. If you set m to zero, you will get the number of zero crossings.

Parameters

- **arr** – KHIVA array with the time series.
- **m** – The m value.

Returns KHIVA array with the number of m-crossings of each time series within tss.

khiva.features.**number_cwt_peaks**(*arr*, *max_w*)
This feature calculator searches for different peaks. To do so, the time series is smoothed by a ricker wavelet and for widths ranging from 1 to :math:'max_w‘. This feature calculator returns the number of peaks that occur at enough width scales and with sufficiently high Signal-to-Noise-Ratio (SNR).

Parameters

- **arr** – KHIVA array with the time series.
- **max_w** – The maximum width to consider.

Returns KHIVA array with the number of peaks for each time series.

khiva.features.**number_peaks**(*arr*, *n*)
Calculates the number of peaks of at least support n in the time series tss. A peak of support n is defined as a subsequence of $tsswhereavalueoccurs, whichisbiggerthanits : math$: neighbours to the left and to the right.

Parameters

- **arr** – KHIVA array with the time series.
- **n** – The support of the peak.

Returns KHIVA array with the number of peaks of at least support n.

khiva.features.**partial_autocorrelation**(*arr*, *lags*)

Calculates the value of the partial autocorrelation function at the given lag. The lag k' partial autocorrelation of a time series $\{x_t, t = 1 \dots T\}$ equals the partial correlation of x_t and x_{t-k}, adjusted for the intermediate variables $\{x_{t-1}, \dots, x_{t-k+1}\}$ ([1]). Following [2], it can be defined as:

$$\alpha_k = \frac{Cov(x_t, x_{t-k}|x_{t-1}, \dots, x_{t-k+1})}{\sqrt{Var(x_t|x_{t-1}, \dots, x_{t-k+1})Var(x_{t-k}|x_{t-1}, \dots, x_{t-k+1})}}$$

with (a) $x_t = f(x_{t-1}, \dots, x_{t-k+1})$ and (b) $x_{t-k} = f(x_{t-1}, \dots, x_{t-k+1})$ being AR(k-1) models that can be fitted by OLS. Be aware that in (a), the regression is done on past values to predict x_t whereas in (b), future values are used to calculate the past value x_{t-k}. It is said in [1] that "for an AR(p), the partial autocorrelations α_k will be nonzero for $k <= p$ and zero for $k > p$." With this property, it is used to determine the lag of an AR-Process.

[1] Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). Time series analysis: forecasting and control. John Wiley & Sons. [2] https://onlinecourses.science.psu.edu/stat510/node/62

Parameters

- **arr** – KHIVA array with the time series.
- **lags** – KHIVA array with the lags to be calculated.

Returns KHIVA array with the partial autocorrelation for each time series for the given lag.

khiva.features.**percentage_of_reoccurring_datapoints_to_all_datapoints**(*arr*, *is_sorted*)

Calculates the percentage of unique values, that are present in the time series more than once.

$$len(differentvaluesoccurringmorethanonce)/len(differentvalues)$$

This means the percentage is normalized to the number of unique values, in contrast to the percentage_of_reoccurring_values_to_all_values.

Parameters

- **arr** – KHIVA array with the time series.
- **is_sorted** – Indicates if the input time series is sorted or not. Defaults to false.

Returns KHIVA array with the percentage of unique values, that are present in the time series more than once.

khiva.features.**percentage_of_reoccurring_values_to_all_values**(*arr*, *is_sorted*)

Calculates the percentage of unique values, that are present in the time series more than once.

$$\frac{number\ of\ data\ points\ occurring\ more\ than\ once}{number\ of\ all\ data\ points)}$$

This means the percentage is normalized to the number of unique values, in contrast to the percentage_of_reoccurring_datapoints_to_all_datapoints.

Parameters

- **arr** – KHIVA array with the time series.
- **is_sorted** – Indicates if the input time series is sorted or not. Defaults to false.

Returns KHIVA array with the percentage of unique values, that are present in the time series more than once.

khiva.features.**quantile**(*arr*, *q*, *precision=100000000.0*)
Returns values at the given quantile.

Parameters

- **arr** – KHIVA array with the time series.
- **q** – Khiva array with the percentile(s) at which to extract score(s). One or many.
- **precision** – Number of decimals expected.

Returns Values at the given quantile.

khiva.features.**range_count**(*arr*, *min*, *max*)
Counts observed values within the interval [min, max).

Parameters

- **arr** – KHIVA array with the time series.
- **min** – Value that sets the lower limit.
- **max** – Value that sets the upper limit.

Returns KHIVA array with the values at the given range.

khiva.features.**ratio_beyond_r_sigma**(*arr*, *r*)
Calculates the ratio of values that are more than $r * std(x)$ (so r sigma) away from the mean of x.

Parameters

- **arr** – KHIVA array with the time series.
- **r** – Number of times that the values should be away from.

Returns KHIVA array with the ratio of values that are more than $r * std(x)$ (so r sigma) away from the mean of x.

khiva.features.**ratio_value_number_to_time_series_length**(*arr*)
Calculates a factor which is 1 if all values in the time series occur only once, and below one if this is not the case. In principle, it just returns:

$$\frac{number\ unique\ values}{number\ values}$$

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the ratio of unique values with respect to the total number of values.

khiva.features.**sample_entropy**(*arr*)
Calculates a vectorized sample entropy algorithm. https://en.wikipedia.org/wiki/Sample_entropy https://www.ncbi.nlm.nih.gov/pubmed/10843903?dopt=Abstract For short time-series this method is highly dependent on the parameters, but should be stable for N > 2000, see: Yentes et al. (2012) - The Appropriate Use of Approximate Entropy and Sample Entropy with Short Data Sets Other shortcomings and alternatives discussed in: Richman & Moorman (2000) - Physiological time-series analysis using approximate entropy and sample entropy.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the same dimensions as tss, whose values (time series in dimension 0) contains the vectorized sample entropy for all the input time series in tss.

khiva.features.**skewness**(*arr*)
Calculates the sample skewness of tss (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array containing the skewness of each time series in tss.

khiva.features.**spkt_welch_density**(*arr*, *coeff*)

Estimates the cross power spectral density of the time series array at different frequencies. To do so, the time series is first shifted from the time domain to the frequency domain.

Welch's method computes an estimate of the power spectral density by dividing the data into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms. [1] P. Welch, "The use of the fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms", IEEE Trans. Audio Electroacoust. vol. 15, pp. 70-73, 1967. [2] M.S. Bartlett, "Periodogram Analysis and Continuous Spectra", Biometrika, vol. 37, pp. 1-16, 1950. [3] Rabiner, Lawrence R., and B. Gold. "Theory and Application of Digital Signal Processing" Prentice-Hall, pp. 414-419, 1975.

Parameters

- **arr** – KHIVA array with the time series.
- **coeff** – The coefficient to be returned.

Returns KHIVA array containing the power spectrum of the different frequencies for each time series in arr.

khiva.features.**standard_deviation**(*arr*)

Calculates the standard deviation of each time series within tss.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array with the standard deviation of each time series within tss.

khiva.features.**sum_of_reoccurring_datapoints**(*arr*, *is_sorted=False*)

Calculates the sum of all data points, that are present in the time series more than once.

Parameters

- **arr** – KHIVA array with the time series.
- **is_sorted** – Indicates if the input time series is sorted or not. Defaults to false.

Returns KHIVA array with the sum of all data points, that are present in the time series more than once.

khiva.features.**sum_of_reoccurring_values**(*arr*, *is_sorted=False*)

Calculates the sum of all values, that are present in the time series more than once.

Parameters

- **arr** – KHIVA array with the time series.
- **is_sorted** – Indicates if the input time series is sorted or not. Defaults to false.

Returns KHIVA array with the sum of all values, that are present in the time series more than once.

khiva.features.**sum_values**(*arr*)

Calculates the sum over the time series arr.

Parameters

- **arr** – KHIVA array with the time series.
- **is_sorted** – Indicates if the input time series is sorted or not. Defaults to false.

Returns KHIVA array with the sum of values in each time series.

khiva.features.**symmetry_looking**(*arr*, *r*)
Calculates if the distribution of tss *looks symmetric*. This is the case if

$$|mean(tss) - median(tss)| < r * (max(tss) - min(tss))$$

Parameters

- **arr** – KHIVA array with the time series.
- **r** – The percentage of the range to compare with.

Returns KHIVA array denoting if the input time series look symmetric.

khiva.features.**time_reversal_asymmetry_statistic**(*arr*, *lag*)
This function calculates the value of:

$$\frac{1}{n - 2lag} \sum_{i=0}^{n-2lag} x_{i+2 \cdot lag}^2 \cdot x_{i+lag} - x_{i+lag} \cdot x_i^2$$

which is

$$\mathbb{E}[L^2(X)^2 \cdot L(X) - L(X) \cdot X^2]$$

where $\mathbb{E}$ is the mean and L is the lag operator. It was proposed in [1] as a promising feature to extract from time series.

Parameters

- **arr** – KHIVA array with the time series.
- **lag** – The lag to be computed.

Returns KHIVA array containing the count of the given value in each time series.

khiva.features.**value_count**(*arr*, *v*)
Counts occurrences of value in the time series tss.

Parameters

- **arr** – KHIVA array with the time series.
- **v** – The value to be counted.

Returns KHIVA array containing the count of the given value in each time series.

khiva.features.**variance**(*arr*)
Computes the variance for the time series array.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array containing the variance in each time series.

khiva.features.**variance_larger_than_standard_deviation**(*arr*)
Calculates if the variance of array is greater than the standard deviation. In other words, if the variance of array is larger than 1.

Parameters **arr** – KHIVA array with the time series.

Returns KHIVA array denoting if the variance of array is greater than the standard deviation.

2.1.1.5 khiva.library

class khiva.library.**KHIVABackend**
Bases: enum.Enum

KHIVA Backend.

KHIVA_BACKEND_CPU = 1
CPU Backend.

KHIVA_BACKEND_CUDA = 2
CUDA Backend.

KHIVA_BACKEND_DEFAULT = 0
Default Backend.

KHIVA_BACKEND_OPENCL = 4
OPENCL Backend.

class khiva.library.**KhivaLibrary**
Bases: object

instance = None

khiva.library.**get_backend**()
Gets the active backend.

Returns The active backend. KHIVABackend type.

khiva.library.**get_backend_info**()
Gets information from the current backend.

Returns A string with information from the current backend.

khiva.library.**get_backends**()
Gets the available backends.

Returns The available backends.

khiva.library.**get_device_count**()
Gets the devices count.

Returns The devices count.

khiva.library.**get_device_id**()
Gets the active device.

Returns The active device.

khiva.library.**set_backend**(*backend*)
Sets the KHIVABackend.

Parameters **backend** – The desired backend. KHIVABackend type.

khiva.library.**set_device**(*device*)
Sets the device.

Parameters **device** – The desired device.

khiva.library.**version**()
Returns a string with the current version of the library.

Returns A string with the current version of the library.

2.1.1.6 khiva.linalg

khiva.linalg.**lls**(*a*, *b*)
Calculates the minimum norm least squares solution x ($||Ax - b||^2$) to $Ax = b$. This function uses the singular value decomposition function of Arrayfire. The actual formula that this function computes is $x = VD\dagger U^T b$. Where U and V are orthogonal matrices and $D\dagger$ contains the inverse values of the singular values contained in D if they are not zero, and zero otherwise.

Parameters

- **a** – KHIVA array with the coefficients of the linear equation problem to solve. It accepts a list of lists or a numpy array with one or several time series.
- **b** – KHIVA array with the measured values.

Returns KHIVA array with the solution to the linear equation problem minimizing the norm 2.

2.1.1.7 khiva.matrix

class khiva.matrix.**BestNResult**(*distances*, *indexes*, *subsequence_indexes*)
Bases: tuple

distances
Alias for field number 0

indexes
Alias for field number 1

subsequence_indexes
Alias for field number 2

class khiva.matrix.**BestNResultOcurrences**(*distances*, *indexes*)
Bases: tuple

distances
Alias for field number 0

indexes
Alias for field number 1

class khiva.matrix.**MatrixProfileResult**(*profile*, *index*)
Bases: tuple

index
Alias for field number 1

profile
Alias for field number 0

khiva.matrix.**find_best_n_discords**(*profile*, *index*, *m*, *n*, *self_join=False*)
This function extracts the best N motifs from a previously calculated matrix profile.

Parameters

- **profile** – KHIVA array with the matrix profile containing the minimum distance of each subsequence.
- **index** – KHIVA array with the matrix profile index containing where each minimum occurs.
- **m** – Subsequence length value used to calculate the input matrix profile.
- **n** – Number of discords to extract.

- **self_join** – Indicates whether the input profile comes from a self join operation or not. It determines whether the mirror similar region is included in the output or not.

Returns KHIVA arrays with the discord distances, the discord indexes and the subsequence indexes.

khiva.matrix.**find_best_n_motifs**(*profile*, *index*, *m*, *n*, *self_join=False*)

This function extracts the best N discords from a previously calculated matrix profile.

Parameters

- **profile** – KHIVA array with the matrix profile containing the minimum distance of each subsequence.
- **index** – KHIVA array with the matrix profile index containing where each minimum occurs.
- **m** – Subsequence length value used to calculate the input matrix profile.
- **n** – Number of motifs to extract.
- **self_join** – Indicates whether the input profile comes from a self join operation or not. It determines whether the mirror similar region is included in the output or not.

Returns KHIVA arrays with the motif distances, the motif indexes and the subsequence indexes.

khiva.matrix.**find_best_n_occurrences**(*query_time_series*, *time_series*, *number_of_occurrences*)

Calculates the N best matches of several queries in several time series.

The result has the following structure:

- 1st dimension corresponds to the nth best match.
- 2nd dimension corresponds to the number of queries.
- 3rd dimension corresponds to the number of time series.

For example, the distance in the position (1, 2, 3) corresponds to the second best distance of the third query in the fourth time series. The index in the position (1, 2, 3) is the index of the subsequence which leads to the second best distance of the third query in the fourth time series.

Parameters **query_time_series** – Array whose first dimension is the length of the query time series and the second

dimension is the number of queries. :param time_series: Array whose first dimension is the length of the time series and the second dimension is the number of time series. :param number_of_occurrences: Number of matches to return. :return: KHIVA arrays with the distances and indexes.

khiva.matrix.**get_chains**(*time_series*, *subsequence_length*)

Calculate all the chains within *tss* using a subsequence length of *m*.

[1] Yan Zhu, Makoto Imamura, Daniel Nikovski, and Eamonn Keogh. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. IEEE ICDM 2017

Parameters

- **time_series** – Time series to compute the chains within them.
- **subsequence_length** – Subsequence length.

Returns

The calculated chains in a KHIVA array with the following topology: - 1st dimension corresponds to the chains indexes flattened. - 2nd dimension:

- [0] corresponds to all the indexes in the chains flattened

- [1] corresponds to the index of the chain that the value in [0] belongs to.
- 3rd dimension corresponds to the number of time series.

Notice that the size of the first dimension is the maximum possible size which is n - m + 1. If the number of values belonging to a chain is lower than the maximum, the remaining values and indexes are 0. It implies that 0 is an invalid chain index.

khiva.matrix.**mass**(*query_time_series*, *time_series*)

Mueen's Algorithm for Similarity Search.

The result has the following structure:

- 1st dimension corresponds to the index of the subsequence in the time series.
- 2nd dimension corresponds to the number of queries.
- 3rd dimension corresponds to the number of time series.

For example, the distance in the position (1, 2, 3) correspond to the distance of the third query to the fourth time series for the second subsequence in the time series.

Parameters **query_time_series** – Array whose first dimension is the length of the query time series and the second

dimension is the number of queries. :param time_series: Array whose first dimension is the length of the time series and the second dimension is the number of time series. :return: KHIVA array with the distances.

khiva.matrix.**matrix_profile**(*first_time_series*, *second_time_series*, *subsequence_length*)

Calculate the matrix profile between *ta* and *tb* using a subsequence length of *m*.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Parameters

- **first_time_series** – KHIVA array with the first time series.
- **second_time_series** – KHIVA array with the second time series.
- **subsequence_length** – Length of the subsequence.

Returns KHIVA arrays with the profile and index.

khiva.matrix.**matrix_profile_self_join**(*time_series*, *subsequence_length*)

Calculate the matrix profile between *t* and itself using a subsequence length of *m*. This method filters the trivial matches.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Parameters

- **time_series** – The query and reference time series in KHIVA array format.
- **subsequence_length** – Lenght of the subsequence

Returns KHIVA arrays with the profile and index.

khiva.matrix.**stomp**(*first_time_series*, *second_time_series*, *subsequence_length*)

Stomp algorithm to calculate the matrix profile between *ta* and *tb* using a subsequence length of *m*.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Parameters

- **first_time_series** – KHIVA array with the first time series.
- **second_time_series** – KHIVA array with the second time series.
- **subsequence_length** – Length of the subsequence.

Returns KHIVA arrays with the profile and index.

khiva.matrix.**stomp_self_join**(*time_series*, *subsequence_length*)

Stomp algorithm to calculate the matrix profile between *t* and itself using a subsequence length of *m*. This method filters the trivial matches.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Parameters

- **time_series** – The query and reference time series in KHIVA array format.
- **subsequence_length** – Lenght of the subsequence

Returns KHIVA arrays with the profile and index.

2.1.1.8 khiva.normalization

khiva.normalization.**decimal_scaling_norm**(*tss*)

Normalizes the given time series according to its maximum value and adjusts each value within the range (-1, 1).

Parameters **tss** – KHIVA array with the time series.

Returns KHIVA array with the same dimensions as tss, whose values (time series in dimension 0) have been normalized by dividing each number by :math::*10^j* , where j is the number of integer digits of the max number in the time series.‘

khiva.normalization.**decimal_scaling_norm_in_place**(*tss*)

Same as decimal_scaling_norm, but it performs the operation in place, without allocating further memory.

Parameters **tss** – KHIVA array with the time series.

khiva.normalization.**max_min_norm**(*tss*, *high=1.0*, *low=0.0*, *epsilon=1e-08*)

Normalizes the given time series according to its minimum and maximum value and adjusts each value within the range [low, high].

Parameters

- **tss** – KHIVA array with the time series.
- **high** – Maximum final value (Defaults to 1.0).
- **low** – Minimum final value (Defaults to 0.0).
- **epsilon** – Safeguard for constant (or near constant) time series as the operation implies a unit scale operation between min and max values in the tss.

Returns KHIVA array with the same dimensions as tss where the time series have been adjusted for zero mean and one as standard deviation.

khiva.normalization.**max_min_norm_in_place**(*tss*, *high=1.0*, *low=0.0*, *epsilon=1e-08*)
Same as max_min_norm, but it performs the operation in place, without allocating further memory.

Parameters

- **tss** – KHIVA array with the time series.
- **high** – Maximum final value (Defaults to 1.0).
- **low** – Minimum final value (Defaults to 0.0).
- **epsilon** – Safeguard for constant (or near constant) time series as the operation implies a unit scale operation between min and max values in the tss.

khiva.normalization.**mean_norm**(*tss*)
Normalizes the given time series according to its maximum-minimum value and its mean. It follows the following formulae:

$$\acute{x} = fracx - mean(x)max(x) - min(x).$$

Parameters **tss** – KHIVA array with the time series.

Returns An array with the same dimensions as tss, whose values (time series in dimension 0) have been normalized by substracting the mean from each number and dividing each number by $max(x) - min($, in the time series.

khiva.normalization.**mean_norm_in_place**(*tss*)
Normalizes the given time series according to its maximum-minimum value and its mean. It follows the following formulae:

Parameters **tss** – KHIVA array with the time series.

khiva.normalization.**znorm**(*tss*, *epsilon=1e-08*)
Calculates a new set of time series with zero mean and standard deviation one.

Parameters

- **tss** – KHIVA array with the time series.
- **epsilon** – Minimum standard deviation to consider. It acts as a gatekeeper for those time series that may be constant or near constant.

Returns KHIVA array with the same dimensions as tss where the time series have been adjusted for zero mean and one as standard deviation.

khiva.normalization.**znorm_in_place**(*tss*, *epsilon=1e-08*)
Adjusts the time series in the given input and performs z-norm in place (without allocating further memory).

Parameters

- **tss** – KHIVA array with the time series.
- **epsilon** – epsilon Minimum standard deviation to consider. It acts as a gatekeeper for those time series that may be constant or near constant.

2.1.1.9 khiva.polynomial

khiva.polynomial.**polyfit**(*x*, *y*, *deg*)
Least squares polynomial fit. Fit a polynomial $p(x) = p[0] * x^{deg} + ... + p[deg]$ of degree deg to points (x, y). Returns a vector of coefficients p that minimises the squared error.

Parameters

- **x** – KHIVA array with the x-coordinates of the M sample points $(x[i], y[i])$.

- **y** – KHIVA array with the y-coordinates of the sample points.
- **deg** – Degree of the fitting polynomial

Returns KHIVA array with the polynomial coefficients, highest power first.

khiva.polynomial.**roots**(*p*)

Calculates the roots of a polynomial with coefficients given in p. The values in the rank-1 array p are coefficients of a polynomial. If the length of p is $n + 1$ then the polynomial is described by:

$$p[0] * x^n + p[1] * x^{n-1} + ... + p[n-1] * x + p[n]$$

Parameters **p** – KHIVA array with the polynomial coefficients.

Returns KHIVA array with the roots of the polynomial.

2.1.1.10 khiva.regression

khiva.regression.**linear**(*xss*, *yss*)

Calculates a linear least-squares regression for two sets of measurements. Both arrays should have the same length.

Parameters

- **xss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- **yss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns slope Slope of the regression line. intercept Intercept of the regression line. rvalue Correlation coefficient. pvalue Two-sided p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic. stderrest Standard error of the estimated gradient.

2.1.1.11 khiva.regularization

khiva.regularization.**group_by**(*tss*, *aggregation_function*, *n_columns_key=1*, *n_columns_value=1*)

Group by operation in the input array using n_columns_key columns as group keys and n_columns_value columns as values. The data is expected to be sorted. The aggregation function determines the operation to aggregate the values.

Parameters

- **tss** – KHIVA array with the time series.
- **aggregation_function** – Function to be used in the aggregation. It receives an integer which indicates the function to be applied. 0 : mean, 1 : median 2 : min, 3 : max, 4 : stdev, 5 : var, default : mean
- **n_columns_key** – Number of columns conforming the key.
- **n_columns_value** – Number of columns conforming the value (they are expected to be consecutive to the column keys).

Returns KHIVA array with the values of the group keys aggregated using the aggregation_function.

2.1.1.12 khiva.statistics

khiva.statistics.**covariance**(*tss*, *unbiased=False*)
Returns the covariance matrix of the time series contained in tss.

Parameters

- **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- **unbiased** – Determines whether it divides by n - 1 (if false) or n (if true).

Returns The covariance matrix of the time series.

khiva.statistics.**kurtosis**(*tss*)
Returns the kurtosis of tss (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns The kurtosis of tss.

khiva.statistics.**ljung_box**(*tss*, *lags*)
The Ljung–Box test checks that data whithin the time series are independently distributed (i.e. the correlations in the population from which the sample is taken are 0, so that any observed correlations in the data result from randomness of the sampling process). Data are no independently distributed, if they exhibit serial correlation. The test statistic is:

$$Q = n\,(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k}$$

where n is the sample size, $\hat{\rho}k$ is the sample autocorrelation at lag k, and h is the number of lags being tested. Under H_0 the statistic Q follows a $\chi^2(h)$. For significance level α, the $criticalregion$ for rejection of the hypothesis of randomness is:

$$Q > \chi^2_{1-\alpha,h}$$

where $\chi^2_{1-\alpha,h}$ is the α-quantile of the chi-squared distribution with h degrees of freedom.

[1] G. M. Ljung G. E. P. Box (1978). On a measure of lack of fit in time series models. Biometrika, Volume 65, Issue 2, 1 August 1978, Pages 297–303.

Parameters

- **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- **lags** – Number of lags being tested.

Returns Array containing the Ljung-Box statistic test.

khiva.statistics.**moment**(*tss*, *k*)
Returns the kth moment of the given time series.

Parameters

- **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- **k** – The specific moment to be calculated.

Returns The kth moment of the given time series.

khiva.statistics.**quantile**(*tss*, *q*, *precision=1e-08*)
Returns values at the given quantile.

Parameters

- **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series. NOTE: the time series should be sorted.
- **q** – Percentile(s) at which to extract score(s). One or many.
- **precision** – Number of decimals expected.

Returns Values at the given quantile.

khiva.statistics.**quantiles_cut**(*tss*, *quantiles*, *precision=1e-08*)
Discretizes the time series into equal-sized buckets based on sample quantiles.

Parameters

- **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series. NOTE: the time series should be sorted.
- **quantiles** – Number of quantiles to extract. From 0 to 1, step 1/quantiles.
- **precision** – Number of decimals expected.

Returns Matrix with the categories, one category per row, the start of the category in the first column and the end in the second category.

khiva.statistics.**sample_stdev**(*tss*)
Estimates standard deviation based on a sample. The standard deviation is calculated using the "n-1" method.

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

Returns The sample standard deviation.

khiva.statistics.**skewness**(*tss*)
Calculates the sample skewness of tss (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).

Parameters **tss** – Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series. NOTE: the time series should be sorted.

Returns Array containing the skewness of each time series in tss.

CHAPTER 3

FAQ

1. **What are the Khiva Array restrictions?**

   Khiva Arrays can have up to 4 dimensions, each one needs to have the same length and the same type.

2. **Can I use the Khiva package without having a GPU for computation?**

   Of course. It is right that Khiva algorithms are designed to run on GPU, but a CPU backend could be set.

3. **What OS are supported?**

   Nowadays, Khiva is supported on Windows, Linux and MacOS.

CHAPTER 4

Authors

4.1 Core Development Team

- Cuesta Merino, David (david.cuesta@shapelets.io)
- Ruíz-Ferrer, Justo (justo.ruiz@shapelets.io)
- Torreño Tirado, Óscar (oscar.torreno@shapelets.io)
- Vilches Reina, Antonio (antonio.vilches@shapelets.io)

4.2 Contributions

- Sánchez de Ybargüen, Luis (luis.sanchez@shapelets.io)

CHAPTER 5

License

```
Mozilla Public License, version 2.0

1. Definitions

1.1. "Contributor"

     means each individual or legal entity that creates, contributes to the
     creation of, or owns Covered Software.

1.2. "Contributor Version"

     means the combination of the Contributions of others (if any) used by a
     Contributor and that particular Contributor's Contribution.

1.3. "Contribution"

     means Covered Software of a particular Contributor.

1.4. "Covered Software"

     means Source Code Form to which the initial Contributor has attached the
     notice in Exhibit A, the Executable Form of such Source Code Form, and
     Modifications of such Source Code Form, in each case including portions
     thereof.

1.5. "Incompatible With Secondary Licenses"
     means

     a. that the initial Contributor has attached the notice described in
        Exhibit B to the Covered Software; or

     b. that the Covered Software was made available under the terms of
        version 1.1 or earlier of the License, but not also under the terms of
        a Secondary License.
```

(continues on next page)

(continued from previous page)

```
1.6. "Executable Form"

     means any form of the work other than Source Code Form.

1.7. "Larger Work"

     means a work that combines Covered Software with other material, in a
     separate file or files, that is not Covered Software.

1.8. "License"

     means this document.

1.9. "Licensable"

     means having the right to grant, to the maximum extent possible, whether
     at the time of the initial grant or subsequently, any and all of the
     rights conveyed by this License.

1.10. "Modifications"

     means any of the following:

     a. any file in Source Code Form that results from an addition to,
        deletion from, or modification of the contents of Covered Software; or

     b. any new file in Source Code Form that contains any Covered Software.

1.11. "Patent Claims" of a Contributor

      means any patent claim(s), including without limitation, method,
      process, and apparatus claims, in any patent Licensable by such
      Contributor that would be infringed, but for the grant of the License,
      by the making, using, selling, offering for sale, having made, import,
      or transfer of either its Contributions or its Contributor Version.

1.12. "Secondary License"

      means either the GNU General Public License, Version 2.0, the GNU Lesser
      General Public License, Version 2.1, the GNU Affero General Public
      License, Version 3.0, or any later versions of those licenses.

1.13. "Source Code Form"

      means the form of the work preferred for making modifications.

1.14. "You" (or "Your")

      means an individual or a legal entity exercising rights under this
      License. For legal entities, "You" includes any entity that controls, is
      controlled by, or is under common control with You. For purposes of this
      definition, "control" means (a) the power, direct or indirect, to cause
      the direction or management of such entity, whether by contract or
      otherwise, or (b) ownership of more than fifty percent (50%) of the
      outstanding shares or beneficial ownership of such entity.
```

(continues on next page)

(continued from previous page)

```
2. License Grants and Conditions

2.1. Grants

     Each Contributor hereby grants You a world-wide, royalty-free,
     non-exclusive license:

     a. under intellectual property rights (other than patent or trademark)
        Licensable by such Contributor to use, reproduce, make available,
        modify, display, perform, distribute, and otherwise exploit its
        Contributions, either on an unmodified basis, with Modifications, or
        as part of a Larger Work; and

     b. under Patent Claims of such Contributor to make, use, sell, offer for
        sale, have made, import, and otherwise transfer either its
        Contributions or its Contributor Version.

2.2. Effective Date

     The licenses granted in Section 2.1 with respect to any Contribution
     become effective for each Contribution on the date the Contributor first
     distributes such Contribution.

2.3. Limitations on Grant Scope

     The licenses granted in this Section 2 are the only rights granted under
     this License. No additional rights or licenses will be implied from the
     distribution or licensing of Covered Software under this License.
     Notwithstanding Section 2.1(b) above, no patent license is granted by a
     Contributor:

     a. for any code that a Contributor has removed from Covered Software; or

     b. for infringements caused by: (i) Your and any other third party's
        modifications of Covered Software, or (ii) the combination of its
        Contributions with other software (except as part of its Contributor
        Version); or

     c. under Patent Claims infringed by Covered Software in the absence of
        its Contributions.

     This License does not grant any rights in the trademarks, service marks,
     or logos of any Contributor (except as may be necessary to comply with
     the notice requirements in Section 3.4).

2.4. Subsequent Licenses

     No Contributor makes additional grants as a result of Your choice to
     distribute the Covered Software under a subsequent version of this
     License (see Section 10.2) or under the terms of a Secondary License (if
     permitted under the terms of Section 3.3).

2.5. Representation

     Each Contributor represents that the Contributor believes its
     Contributions are its original creation(s) or it has sufficient rights to
```

(continues on next page)

(continued from previous page)

```
     grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

     This License is not intended to limit any rights You have under
     applicable copyright doctrines of fair use, fair dealing, or other
     equivalents.

2.7. Conditions

     Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in
     Section 2.1.


3. Responsibilities

3.1. Distribution of Source Form

     All distribution of Covered Software in Source Code Form, including any
     Modifications that You create or to which You contribute, must be under
     the terms of this License. You must inform recipients that the Source
     Code Form of the Covered Software is governed by the terms of this
     License, and how they can obtain a copy of this License. You may not
     attempt to alter or restrict the recipients' rights in the Source Code
     Form.

3.2. Distribution of Executable Form

     If You distribute Covered Software in Executable Form then:

     a. such Covered Software must also be made available in Source Code Form,
        as described in Section 3.1, and You must inform recipients of the
        Executable Form how they can obtain a copy of such Source Code Form by
        reasonable means in a timely manner, at a charge no more than the cost
        of distribution to the recipient; and

     b. You may distribute such Executable Form under the terms of this
        License, or sublicense it under different terms, provided that the
        license for the Executable Form does not attempt to limit or alter the
        recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

     You may create and distribute a Larger Work under terms of Your choice,
     provided that You also comply with the requirements of this License for
     the Covered Software. If the Larger Work is a combination of Covered
     Software with a work governed by one or more Secondary Licenses, and the
     Covered Software is not Incompatible With Secondary Licenses, this
     License permits You to additionally distribute such Covered Software
     under the terms of such Secondary License(s), so that the recipient of
     the Larger Work may, at their option, further distribute the Covered
     Software under the terms of either this License or such Secondary
     License(s).

3.4. Notices

     You may not remove or alter the substance of any license notices
```

(continues on next page)

(continued from previous page)

```
     (including copyright notices, patent notices, disclaimers of warranty, or
     limitations of liability) contained within the Source Code Form of the
     Covered Software, except that You may alter any license notices to the
     extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

     You may choose to offer, and to charge a fee for, warranty, support,
     indemnity or liability obligations to one or more recipients of Covered
     Software. However, You may do so only on Your own behalf, and not on
     behalf of any Contributor. You must make it absolutely clear that any
     such warranty, support, indemnity, or liability obligation is offered by
     You alone, and You hereby agree to indemnify every Contributor for any
     liability incurred by such Contributor as a result of warranty, support,
     indemnity or liability terms You offer. You may include additional
     disclaimers of warranty and limitations of liability specific to any
     jurisdiction.

4. Inability to Comply Due to Statute or Regulation

   If it is impossible for You to comply with any of the terms of this License
   with respect to some or all of the Covered Software due to statute,
   judicial order, or regulation then You must: (a) comply with the terms of
   this License to the maximum extent possible; and (b) describe the
   limitations and the code they affect. Such description must be placed in a
   text file included with all distributions of the Covered Software under
   this License. Except to the extent prohibited by statute or regulation,
   such description must be sufficiently detailed for a recipient of ordinary
   skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You
     fail to comply with any of its terms. However, if You become compliant,
     then the rights granted under this License from a particular Contributor
     are reinstated (a) provisionally, unless and until such Contributor
     explicitly and finally terminates Your grants, and (b) on an ongoing
     basis, if such Contributor fails to notify You of the non-compliance by
     some reasonable means prior to 60 days after You have come back into
     compliance. Moreover, Your grants from a particular Contributor are
     reinstated on an ongoing basis if such Contributor notifies You of the
     non-compliance by some reasonable means, this is the first time You have
     received notice of non-compliance with this License from such
     Contributor, and You become compliant prior to 30 days after Your receipt
     of the notice.

5.2. If You initiate litigation against any entity by asserting a patent
     infringement claim (excluding declaratory judgment actions,
     counter-claims, and cross-claims) alleging that a Contributor Version
     directly or indirectly infringes any patent, then the rights granted to
     You by any and all Contributors for the Covered Software under Section
     2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user
     license agreements (excluding distributors and resellers) which have been
     validly granted by You or Your distributors under this License prior to
     termination shall survive termination.
```

(continues on next page)

(continued from previous page)

```
6. Disclaimer of Warranty

   Covered Software is provided under this License on an "as is" basis,
   without warranty of any kind, either expressed, implied, or statutory,
   including, without limitation, warranties that the Covered Software is free
   of defects, merchantable, fit for a particular purpose or non-infringing.
   The entire risk as to the quality and performance of the Covered Software
   is with You. Should any Covered Software prove defective in any respect,
   You (not any Contributor) assume the cost of any necessary servicing,
   repair, or correction. This disclaimer of warranty constitutes an essential
   part of this License. No use of  any Covered Software is authorized under
   this License except under this disclaimer.

7. Limitation of Liability

   Under no circumstances and under no legal theory, whether tort (including
   negligence), contract, or otherwise, shall any Contributor, or anyone who
   distributes Covered Software as permitted above, be liable to You for any
   direct, indirect, special, incidental, or consequential damages of any
   character including, without limitation, damages for lost profits, loss of
   goodwill, work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses, even if such party shall have been
   informed of the possibility of such damages. This limitation of liability
   shall not apply to liability for death or personal injury resulting from
   such party's negligence to the extent applicable law prohibits such
   limitation. Some jurisdictions do not allow the exclusion or limitation of
   incidental or consequential damages, so this exclusion and limitation may
   not apply to You.

8. Litigation

   Any litigation relating to this License may be brought only in the courts
   of a jurisdiction where the defendant maintains its principal place of
   business and such litigation shall be governed by laws of that
   jurisdiction, without reference to its conflict-of-law provisions. Nothing
   in this Section shall prevent a party's ability to bring cross-claims or
   counter-claims.

9. Miscellaneous

   This License represents the complete agreement concerning the subject
   matter hereof. If any provision of this License is held to be
   unenforceable, such provision shall be reformed only to the extent
   necessary to make it enforceable. Any law or regulation which provides that
   the language of a contract shall be construed against the drafter shall not
   be used to construe this License against a Contributor.


10. Versions of the License

10.1. New Versions

      Mozilla Foundation is the license steward. Except as provided in Section
      10.3, no one other than the license steward has the right to modify or
      publish new versions of this License. Each version will be given a
      distinguishing version number.
```

(continues on next page)

(continued from previous page)

```
10.2. Effect of New Versions

      You may distribute the Covered Software under the terms of the version
      of the License under which You originally received the Covered Software,
      or under the terms of any subsequent version published by the license
      steward.

10.3. Modified Versions

      If you create software not governed by this License, and you want to
      create a new license for such software, you may create and use a
      modified version of this License if you rename the license and remove
      any references to the name of the license steward (except to note that
      such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary
      Licenses If You choose to distribute Source Code Form that is
      Incompatible With Secondary Licenses under the terms of this version of
      the License, the notice described in Exhibit B of this License must be
      attached.

Exhibit A - Source Code Form License Notice

      This Source Code Form is subject to the
      terms of the Mozilla Public License, v.
      2.0. If a copy of the MPL was not
      distributed with this file, You can
      obtain one at
      http://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular file,
then You may include the notice in a location (such as a LICENSE file in a
relevant directory) where a recipient would be likely to look for such a
notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - "Incompatible With Secondary Licenses" Notice

      This Source Code Form is "Incompatible
      With Secondary Licenses", as defined by
      the Mozilla Public License, v. 2.0.
```

CHAPTER 6

Changelog

KHIVA uses Semantic Versioning

6.1 Version 0.2.2

Added

- Join function for KHIVA arrays.

Fixed

- to_arrayfire and from_arrayfire functions.
- KShape for double precision data.

6.2 Version 0.2.0

Added

- KMeans algorithm.
- KShape Algorithm.
- Added Ljung-Box test.
- Installation script for Windows.
- SBD distance function.
- Header checks in all header files from core library and bindings.

Changed

- Implementation improvement of stomp function and find motifs and discords functions.
- Readme and installation Guide have been improved.

Fixed

- PAA method.
- Crosscorrelation function to work with several time series.
- Conan script to work with new conan version.
- Documentation generator to work with new PIP version.
- Cmake path for windows in installation guide.

6.3 Version 0.1.0

Added - Binding for Matlab. - Statistics namespace. - Features namespace. - Dimensionality namespace. - Polynomial namespace. - LinAlg namespace. - Normalization namespace. - Regression namespace. - Regularization namespace. - Support for Windows and Linux (Ubuntu 16.04 LTS). - Documentation using breathe. - Async. memory management. - Operators for Khiva Arrays class for all bindings (Java, Python, C++, R)

Removed - Simplification namespace.

6.4 Version 0.0.1

Added - We have set Arrayfire as an abstraction layer to run on top of accelerators that are able to run OpenCL. - Implementation of STOMP method. - (Mueen's Algorithm for Similarity Search) MASS algorithm implementation. - Implementation of Simplification algorithms (Visvalingam and PIP). - Benchmarks Suite (based on Google Benchmark). - Test Suite (Google Test). - New Features Algorithms. - Binding for Python, R, Java. - Matrix namespace. - Distances namespace. - Simplification namespace.

CHAPTER 7

How to contribute

We have just started! Our aim is to make the Khiva library the reference library for time series analysis in the fastest fashion. To achieve this target we need your help!

All contributions, bug reports, bug fixes, documentation improvements, enhancements and ideas are welcome. If you want to add one or two interesting feature calculators, implement a new feature selection process or just fix 1-2 typos, your help is appreciated.

If you want to help, just create a pull request on our github page.

7.1 Guidelines

7.1.1 Branching model

Our branching model has one permanent branch, **master**. We aim at using *master* as the main branch, where all features are merged. In this sense, we also use the master branch to contain the release versions of the Python Khiva library by means of git tags.

7.1.2 Contribution process

In order to contribute to the code base, we follow the next process:

1. The main branch is *master*, every developer should pull the current status of the branch before stating to develop any new feature. *git pull*

2. Create a new branch with the following pattern "feature/[name_of_the_feature]" *git checkout -b feature/example_feature*

3. Develop the new feature on the the new branch. It includes testing and documentation. *git commit -a -m "Bla, Bla, Bla"*

git push

4. Open a Pull Request to merge the feature branch in to *master*. Currently, a pull request has to be reviewed at least by one person.

5. Finally, delete the feature branch.

6. Move back to *master* branch. *git checkout master*

7. Pull the latest changes. *git pull*

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

G

H

I

J

K

L

M