stmetrics

Release 0.1.5.0

Feb 24, 2021

Introduction

CHAPTER 1

About

The **stmetrics** is a python package that provides the extraction of state-of-the-art time-series features. These features can be used for remote sensing time-series image classification and analysis.

Producing reliable land use and land cover maps to support the deployment and operation of public policies is a necessity, especially when environmental management and economic development are considered. To increase the accuracy of these maps, satellite image time-series (provided as Analysis Ready Data - ARD) have been used, as they allow the understanding of land cover dynamics through the time.

This package is developed under the Brazil Data Cube project that is part of the "Environmental Monitoring of Brazilian Biomes project", funded by the Amazon Fund through the financial collaboration of the Brazilian Development Bank (BNDES) and the Foundation for Science, Technology and Space Applications (FUNCATE) no. 17.2.0536.1. Brazil Data Cube is the successor of the research project e-sensing, funded by FAPESP (Fapesp 2014/08398-6).

CHAPTER 2

License

MIT License

Copyright (c) 2019 INPE.

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

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

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

CHAPTER 3

Contributors

- Anderson Soares
- Thales Körting
- Hugo Bendini
- Leila Fonseca
- Daiane Vaz
- Tatiana Uehara
- Michel Chaves
- Sarah Lechler

CHAPTER 4

Publications

Soares, A. R., Bendini, H. N., Vaz, D. V., Uehara, T. D., Neves, A. K., Lechler, S., Körting, T. S., Fonseca, L. M. G. STMETRICS: A PYTHON PACKAGE FOR SATELLITE IMAGE TIME-SERIES FEATURE EXTRACTION. Conference: 2020 IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2020).

Uehara, T. D. T., Soares, A. R., Quevedo, R. P., Körting, T. S., Fonseca, L. M. G., & Adami, M. LAND COVER CLASSIFICATION OF AN AREA SUSCEPTIBLE TO LANDSLIDES USING RANDOM FOREST AND NDVI TIME SERIES DATA. Conference: 2020 IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2020).

Soares, A. R., Körting, T. S., Fonseca, L. M. G., Bendini, H. N. SIMPLE NONLINEAR ITERATIVE TEMPORAL CLUSTERING. IEEE Transactions on Geoscience and Remote, 2020 (Early Access).

CHAPTER 5

Metrics module

Module for computing metrics

stmetrics.metrics.**get_metrics**(*series, metrics_dict={'basics': ['all'], 'fractal': ['all'], 'polar': ['all']}, nodata=-9999, show=False*)

This function performs the computation of the basic, polar and fractal metrics available in the stmetrics package.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **metrics_dict** (*dictionary*) – Dictionary with metrics to be computed.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns time_metrics Dicitionary of metrics.

stmetrics.metrics.**sits2metrics**(*dataset, metrics={'basics': ['all'], 'fractal': ['all'], 'polar': ['all']}, num_cores=-1*)

This function performs the computation of the metrics using multiprocessing.

Parameters

- **dataset** (*rasterio dataset, numpy array (ZxMxN) - Z is the time series lenght or xarray.Dataset*) – Time series.
- **metrics_dict** (*dictionary*) – Dictionary with metrics to be computed.
- **num_cores** (*integer*) – Number of cores to be used. Value -1 means all cores available.

Returns image Numpy matrix of metrics or xarray.Dataset with the metrics as an dataset. The orders of the dimensions, follows the dictionary provided.

CHAPTER 6

Polar metrics

Module for computing polar metrics, that are derived from a time wheel legend, projecting the values of a timeseries to angles in the interval `[0, 2pi]`.

stmetrics.polar.**angle**(*timeseries*, *nodata=-9999*)

Angle - The main angle of the closed shape created by the polar visualization. If two angle are the same, the first one is presented.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return angle The main angle of time series.

stmetrics.polar.**area_q1**(*timeseries*, *nodata=-9999*)

Area of the closed shape over the first quadrant.

Parameters

- **timeseries** (*numpy.ndarra*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return area_q1 Area of polygon that covers quadrant 1.

stmetrics.polar.**area_q2**(*timeseries*, *nodata=-9999*)

Area_Q2 - Area of the closed shape over the second quadrant.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return area_q2 Area of polygon that covers quadrant 2.

stmetrics.polar.**area_q3**(*timeseries*, *nodata=-9999*)

Area of the closed shape over the thrid quadrant.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return area_q3 Area of polygon that covers quadrant 3.

stmetrics.polar.**area_q4**(*timeseries*, *nodata=-9999*)
Area of the closed shape over the fourth quadrant.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return area_q4 Area of polygon that covers quadrant 4.

stmetrics.polar.**area_season**(*timeseries*, *nodata=-9999*)
Partial area of the shape, proportional to some quadrant of the polar representation.

This metric returns the area of the polygon on each quadrant.

area2—-area1 | | area3—-area4

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return area The area of the time series that intersected each quadrant that represents a season.

stmetrics.polar.**area_ts**(*timeseries*, *nodata=-9999*)
Area - Area of the closed shape.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return area_ts Area of polygon.

stmetrics.polar.**csi**(*timeseries*, *nodata=-9999*)
Cell Shape Index - This is a dimensionless quantitative measure of morphology, that characterize the standard deviation of an object from a circle.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return shape_index Quantitative measure of morphology.

Note: Rational of this metric:

After polar transformation time series usually have a round shape, which can be releate do cell in some cases. That's why cell shape index is available here.

---

stmetrics.polar.**ecc_metric**(*timeseries*, *nodata=-9999*)
Return values close to 0 if the shape is a circle and 1 if the shape is similar to a line.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.

- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return eccentricity Eccentricity of time series after polar transformation.

stmetrics.polar.**get_seasons**(*x*, *y*)

This function polygons that represents the four season of a year. They are used to compute the metric area_season.

Parameters

- **x** (*numpy.array*) – x-coordinate in polar space.
- **y** (*numpy.array*) – y-coordinate in polar space.

Returns tuple of polygons Quadrant polygons

stmetrics.polar.**gyration_radius**(*timeseries*, *nodata=-9999*)

Gyration_radius - Equals the average distance between each point inside the shape and the shape's centroid.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return gyration_radius Average distance between each point inside the shape and the shape's centroid.

stmetrics.polar.**polar_balance**(*timeseries*, *nodata=-9999*)

Polar_balance - The standard deviation of the areas per season, considering the 4 seasons.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return polar_balance Standard deviation of the areas per season.

stmetrics.polar.**polar_plot**(*timeseries*, *nodata=-9999*)

This function create a plot of time series in polar space.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns plot Plot of time series in polar space.

stmetrics.polar.**symmetric_distance**(*time_series_1*, *time_series_2*, *nodata=-9999*)

This function computes the difference between two time series in the polar space.

Parameters

- **timeseries1** – Time series.
- **timeseries2** – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns dist Distance between two time series.

stmetrics.polar.**ts_polar**(*timeseries, funcs=['all'], nodata=-9999, show=False*)

This function compute 9 polar metrics:

- Area - Area of the closed shape.
- Angle - The main angle of the closed shape created after transformation.

- Area_q1 - Partial area of the shape, proportional to quadrant 1 of the polar representation.
- Area_q2 - Partial area of the shape, proportional to quadrant 2 of the polar representation.
- Area_q3 - Partial area of the shape, proportional to quadrant 3 of the polar representation.
- Area_q4 - Partial area of the shape, proportional to quadrant 4 of the polar representation.
- Polar_balance - The standard deviation of the areas per season, considering the 4 seasons.
- Eccenticity - Return values close to 0 if the shape is a circle and 1 if the shape is similar to a line.
- Gyration_radius - Equals the average distance between each point inside the shape and the shape's centroid.
- CSI - This is a dimensionless quantitative measure of morphology, that characterize the standard deviation of an object from a circle.

To visualize the time series on polar space use: ts_polar(timeseries, show=True)

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*boolean*) – nodata of the time series. Default is -9999.
- **show** – This inform that the polar plot must be presented.

Returns out_metrics Dictionary with polar metrics values.

Tip: Check the original publication of the metrics: Körting, Thales & Câmara, Gilberto & Fonseca, Leila. (2013). Land Cover Detection Using Temporal Features Based On Polar Representation.

CHAPTER 7

Basic metrics

Module for computing basic metrics, including sum of values, average value, amplitude (max - min), and so on. We provide a function to compute all metrics in a single call (`ts_basics`).

`stmetrics.basics.`**`abs_sum_ts`**(*timeseries*, *nodata=-9999*)

Sum of absolute values of the time series. All values are converted to absolute (positive) values, and are summed.

Parameters

- **`timeseries`** (*numpy.ndarray*) – Time series.
- **`nodata`** (*int*) – nodata of the time series. Default is -9999.

Returns Sum of absolute values of the time series.

`stmetrics.basics.`**`amd_ts`**(*timeseries*, *nodata=-9999*)

Computes the mean of the absolute derivative of time series.

Regarding to vegetation it provides information on the growth rate of vegetation, allowing discrimination of natural cycles from crop cycles.

Parameters

- **`timeseries`** (*numpy.ndarray*) – Time series.
- **`nodata`** (*int*) – nodata of the time series. Default is -9999.

Returns The mean of the absolute derivative of time series.

`stmetrics.basics.`**`amplitude_ts`**(*timeseries*, *nodata=-9999*)

The difference between the maximum and minimum values of the time series.

Parameters

- **`timeseries`** (*numpy.ndarray*) – Time series.
- **`nodata`** (*int*) – nodata of the time series. Default is -9999.

Returns Amplitude of values of time series.

`stmetrics.basics.`**`fqr_ts`**(*timeseries*, *nodata=-9999*)

Computes the first quartile of a time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The first quartile of the time series.

stmetrics.basics.**fslope_ts**(*timeseries*, *nodata=-9999*)

Maximum value of the first slope of the time series. It indicates when some abrupt change happened in the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The maximum value of the first slope of time series.

stmetrics.basics.**iqr_ts**(*timeseries*, *nodata=-9999*)

Computes the interquaritle range of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The interquaritle range of the time series.

stmetrics.basics.**max_ts**(*timeseries*, *nodata=-9999*)

Maximum value of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns Maximum value of time series.

stmetrics.basics.**mean_ts**(*timeseries*, *nodata=-9999*)

Average value (mean) of the time series, considering only valid values. When nodata is found, it is not included in N value (for all functions).

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns Mean value of time series.

stmetrics.basics.**min_ts**(*timeseries*, *nodata=-9999*)

Minimum value of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns Minimum value of time series.

stmetrics.basics.**mse_ts**(*timeseries*, *nodata=-9999*)

Computes mean spectral energy of a time series.

Mean spectral energy density computes the energy of the time series that is distributed with frequency. High frequency time series usually have lower spectral energy.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The mean spectral energy of the time series.

stmetrics.basics.**skew_ts**(*timeseries*, *nodata=-9999*)

Measures the asymmetry of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The asymmetry of time series.

stmetrics.basics.**sqr_ts**(*timeseries*, *nodata=-9999*)

Computes the second quartile of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The second quartile of the time series.

stmetrics.basics.**std_ts**(*timeseries*, *nodata=-9999*)

Standard deviation of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns Standard deviation of time series.

stmetrics.basics.**sum_ts**(*timeseries*, *nodata=-9999*)

Sum of values of the time series.

When dealing with vegetation analysis, this value can be a proxy of the annual production of vegetation.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns Sum of values of time series.

stmetrics.basics.**tqr_ts**(*timeseries*, *nodata=-9999*)

Computes the third quartile of a time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Returns The third quartile of the time series.

stmetrics.basics.**ts_basics**(*timeseries, funcs=['all'], nodata=-9999*)

This function compute all basic metrics in a single call, returning a dictionary:

- "Max" - Maximum value of the time series.
- "Min" - Minimum value of the time series.
- "Mean" - Average value of the time series.
- "Std" - Standard deviation of the time series.
- "Sum" - Sum of values over a cycle. Usually is an indicator of the annual production of vegetation.
- "Amplitude" - The difference between the time series's maximum and minimum values.
- "MSE" - Mean Spectral Energy.
- "First_slope" - Maximum value of the first slope of the cycle.
- "Skew" - Measures the asymmetry of the time series.
- "AMD" - Absolute mean derivative (AMD).
- "AbsSum" - Absolute Sum of values over of the time series.
- "IQR" - Interquaritle range (IQR) of the time series.
- "FQR" - First quartile of the time series.
- "SQR" - Second quartile of the time series.
- "TQR" - Third quaritle of the time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*real number*) – nodata of the time series. Default is -9999.

Returns Dictionary of basic metrics

CHAPTER 8

Fractal metrics

Module for computing fractal metrics

stmetrics.fractal.**dfa_fd**(*timeseries*, *nvals=None*, *overlap=True*, *order=1*, *nodata=-9999*)

Detrended Fluctuation Analysis (DFA) measures the Hurst parameter H, which is very similar to the Hurst Exponent (HE). The main difference is that DFA can be used for non-stationary time series.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nvals** (*int*) – Sizes of subseries to use.
- **overlap** (*Boolean*) – if True, there will be a 50% overlap on windows otherwise non-overlapping windows will be used.
- **order** (*Boolean*) – Polynomial order of trend to remove.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return dfa Detrended Fluctuation Analysis.

Note: This function uses the Detrended Fluctuation Analysis (DFA) implementation from the Nolds package. Due to time series characteristcs we use by default the 'RANSAC' fitting method as it is more robust to outliers. For more details regarding the hurst implementation, check Nolds documentation page.

---

stmetrics.fractal.**hurst_exp**(*timeseries*, *nvals=None*, *nodata=-9999*)

Computes the Hurst Exponent (HE) by a standard rescaled range (R/S) approach. HE is a self-similarity measure that assesses long-range dependence in a time series. It can be used to determine whether the time series is more, less, or equally likely to increase if it has increased in previous steps.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nvals** (*int*) – Sizes of subseries to use.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return hurst The Hurst Expoent (HE).

Note: This function was adapted from the package Nolds. Due to time series characteristcs we use by default the 'RANSAC' fitting method as it is more robust to outliers. For more details regarding the hurst implementation, check Nolds documentation page.

---

stmetrics.fractal.**katz_fd**(*timeseries*, *nodata=-9999*)

Katz fractal dimension.

It is defined by: .. math:: K = frac{log_{10}(n)}{log_{10}(d/L)+log_{10}(n)} where L is the total length of the time series and d is the Euclidean distance between the first point in the series and the point that provides the furthest distance with respect to the first point.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return kfd Katz fractal dimension.

Note: This function was adapted from the package entropy available at: https://github.com/raphaelvallat/entropy.

Tip: To know more about it: Michael J. Katz, Fractals and the analysis of waveforms, Computers in Biology and Medicine, volume 18, Issue 3,1988, Pages 145-156,ISSN 0010-4825, https://doi.org/10.1016/0010-4825(88)90041-8. Esteller, R. et al. (2001). A comparison of waveform fractal dimension algorithms. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 48(2), 177-183. Goh, Cindy, et al. "Comparison of fractal dimension algorithms for the computation of EEG biomarkers for dementia." 2nd International Conference on Computational Intelligence in Medicine and Healthcare (CIMED2005). 2005.

---

stmetrics.fractal.**ts_fractal**(*timeseries, funcs=['all'], nodata=-9999*)

This function computes 4 fractal dimensions and the hurst exponential.

- DFA: measures the Hurst parameter H, which is similar to the Hurst exponent.
- HE: self-similarity measure that assess long-range dependence in a time series.
- KFD: This algorirhm computes the FD using Katz algorithm.

Parameters

- **timeseries** (*numpy.ndarray*) – Time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return out_metrics Dictionary with fractal metrics values.

CHAPTER 9

Spatial Module

Module for spatial functions, considering a set of polygons obtained by image segmentation.

stmetrics.spatial.**aspect_ratio**(*geom*)

This function computes the aspect ratio of a given geometry.

The Aspect Ratio is the ratio of the length and the width of the minimum bounding rectangle of a polygon.

Parameters **geom** (*shapely.geometry.Polygon*) – Polygon geometry

Returns **aspect_ratio** Polygon aspect_ratio.

stmetrics.spatial.**distance**(*c_series*, *ic*, *jc*, *subim*, *S*, *m*, *rmin*, *cmin*, *window=None*, *max_dist=None*, *max_step=None*, *max_diff=None*, *penalty=None*, *psi=None*, *pruning=False*)

This function computes the spatial-temporal distance between two pixels using the DTW distance.

Parameters

- **c_series** (*numpy.ndarray*) – average time series of cluster.
- **ic** (*int*) – X coordinate of cluster center.
- **jc** (*int*) – Y coordinate of cluster center.
- **subim** (*int*) – Block of image from the cluster under analysis.
- **S** (*int*) – Pattern spacing value.
- **m** (*float*) – Compactness value.
- **rmin** (*int*) – Minimum row.
- **cmin** (*int*) – Minimum column.
- **window** – Only allow for maximal shifts from the two diagonals smaller than this number. It includes the diagonal, meaning that an Euclidean distance is obtained by setting window=1.
- **max_dist** – Stop if the returned values will be larger than this value.
- **max_step** – Do not allow steps larger than this value.

- **max_diff** – Return infinity if length of two series is larger.
- **penalty** – Penalty to add if compression or expansion is applied.
- **psi** – Psi relaxation parameter (ignore start and end of matching). Useful for cyclical series.
- **use_pruning** – Prune values based on Euclidean distance.

Returns D numpy.ndarray distance.

stmetrics.spatial.**distance_fast**(*c_series*, *ic*, *jc*, *subim*, *S*, *m*, *rmin*, *cmin*, *window=None*, *max_dist=None*, *max_step=None*, *max_diff=None*, *penalty=None*, *psi=None*)

This function computes the spatial-temporal distance between two pixels using the dtw distance with C implementation.

Parameters

- **c_series** (*numpy.ndarray*) – average time series of cluster.
- **ic** (*int*) – X coordinate of cluster center.
- **jc** (*int*) – Y coordinate of cluster center.
- **subim** (*int*) – Block of image from the cluster under analysis.
- **S** (*int*) – Pattern spacing value.
- **m** (*float*) – Compactness value.
- **rmin** (*int*) – Minimum row.
- **cmin** (*int*) – Minimum column.
- **window** – Only allow for maximal shifts from the two diagonals smaller than this number. It includes the diagonal, meaning that an Euclidean distance is obtained by setting window=1.
- **max_dist** – Stop if the returned values will be larger than this value.
- **max_step** – Do not allow steps larger than this value.
- **max_diff** – Return infinity if length of two series is larger.
- **penalty** – Penalty to add if compression or expansion is applied.
- **psi** – Psi relaxation parameter (ignore start and end of matching). Useful for cyclical series.

Returns D numpy.ndarray distance.

stmetrics.spatial.**dtw_filter**(*dataset*, *kernel_size=3*, *window=None*, *max_dist=None*, *max_step=None*, *max_length_diff=None*, *penalty=None*, *psi=None*, *pruning=False*)

This function performs a spatio-temporal filtering of datacube using the DTW distance.

Parameters

- **dataset** (*shapely.geometry.Polygon*) – SITS dataset.
- **kernel_size** (*int*) – Size of convolutional kernel.
- **window** – Only allow for maximal shifts from the two diagonals smaller than this number. It includes the diagonal, meaning that an Euclidean distance is obtained by setting window=1.
- **max_dist** – Stop if the returned values will be larger than this value.
- **max_step** – Do not allow steps larger than this value.

- **max_diff** – Return infinity if length of two series is larger.
- **penalty** – Penalty to add if compression or expansion is applied.
- **psi** – Psi relaxation parameter (ignore start and end of matching). Useful for cyclical series.
- **use_pruning** – Prune values based on Euclidean distance.

Returns edge Edge image as numpy.ndarray.

stmetrics.spatial.**extract_features**(*dataset, segmentation, features=['mean', 'std', 'min', 'max', 'majority', 'area', 'perimeter', 'width', 'length', 'aspect_ratio', 'symmetry', 'compactness', 'rectangular_fit'], nodata=-9999*)

This function computes features using polygon geometries.

Regarding image features, this function computes 5 features: Mean, Standard Deviation, Minimum, Maximum and Majority (mode). Along side with the image features, 8 shape features can be computed for each polygon: Area, Perimeter, Width, Length, Aspect Ratio ratio, Symmetry, Compactness and Rectangular fit.

Parameters

- **dataset** (*Rasterio, Xarray.Dataset or string*) – Images or path to images that compose time series.
- **segmentation** (*geopandas.Dataframe*) – Spatio-temporal Segmentation.
- **features** (*list*) – List of features to be extracted
- **nodata** (*int*) – Nodata value

Returns segmentation GeoPandas DataFrame with the features.

stmetrics.spatial.**init_cluster_hex**

This function initialize the clusters for SNITC using a hexagonal pattern.

Parameters

- **rows** (*int*) – Number of rows of image.
- **columns** (*int*) – Number of columns of image.
- **ki** (*int*) – Number of desired superpixel.
- **img** (*numpy.ndarray*) – Input image.
- **bands** (*int*) – Number of bands (lenght of time series).

Returns C ND-array containing cluster centres information.

Returns S Spacing between clusters.

Returns l Matrix label.

Returns d Distance matrix from cluster centres.

Returns k Number of superpixels that will be produced.

stmetrics.spatial.**init_cluster_regular**

This function initialize the clusters for SNITC using a square pattern.

Parameters

- **rows** (*int*) – Number of rows of image.
- **columns** (*int*) – Number of columns of image.
- **ki** (*int*) – Number of desired superpixel.

- **img** (*numpy.ndarray*) – Input image.
- **bands** (*int*) – Number of bands (lenght of time series).

Returns C ND-array containing cluster centres information.

Returns S Spacing between clusters.

Returns l Matrix label.

Returns d Distance matrix from cluster centres.

Returns k Number of superpixels that will be produced.

stmetrics.spatial.**length**(*geom*)

This function computes the lenght of a geometry.

Parameters **geom** (*shapely.geometry.Polygon*) – Polygon geometry.

Returns length Polygon length.

stmetrics.spatial.**postprocessing**(*raster*, *S*)

Post processing function to enforce connectivity.

Parameters

- **raster** (*numpy.ndarray*) – Labelled image.
- **S** (*int*) – Spacing between superpixels.

Returns final Labelled image with connectivity enforced.

stmetrics.spatial.**rectangular_fit**(*geom*)

This function computes the rectangular fit of a geometry. The rectangular fit is defined as:

$$RectFit = (AR - AD)/AO$$

where AO is the area of the original object, AR is the area of the equal rectangle (AO = AR) and AD is the overlaid difference between the equal rectangle and the original object (Sun et al. 2015).

Parameters **geom** (*shapely.geometry.Polygon*) – Polygon geometry

Returns rectangular_fit Polygon rectangular fit.

Tip: To know more about it:

Sun, Z., Fang, H., Deng, M., Chen, A., Yue, P. and Di, L. "Regular Shape Similarity Index:Novel Index for Accurate Extraction of Regular Objects From Remote Sensing Images," IEEE Transactions on Geoscience and Remote Sensing, v.53, 2015, p. 3737. doi:10.1109/TGRS.2014.2382566

---

stmetrics.spatial.**reock_compactness**(*geom*)

This function computes the reock compactness of a given geometry.

The Reock Score (R) is the ratio of the area of a polygon P to the area of a minimum bounding cirle (AMBC) that encloses the geometry. This score falls within the range of [0,1] and high values indicates a more compact geometry.

$$Reock = A_p/A_{MBC}$$

Parameters **geom** (*shapely.geometry.Polygon*) – Polygon geometry.

Returns reock Polygon reock compactness.

Tip: To know more about it:

Reock, Ernest C. 1961. "A note: Measuring compactness as a requirement of legislative apportionment." Midwest Journal of Political Science 1(5), 70–74.

stmetrics.spatial.**seg_metrics**(*dataframe, bands=None, metrics_dict={'basics': ['all'], 'fractal': ['all'], 'polar': ['all']}, features=['mean'], num_cores=-1*)

This function compute time series metrics from a geopandas with time features. Currently, basic, polar and fractal metrics are extracted. but you can set the metrics you to compute using a dictionary.

Parameters

- **dataframe** (*pandas DataFrame*) – Pandas DataFrame with time series information.
- **bands** (*list*) – List of bands from which the metrics should be computed.
- **metrics_dict** (*dictionary*) – Dictionary of metrics to be computed.
- **features** (*list*) – List of features to be used for computation. This parameter allows you to use the features extracted with `extract_features` function and compute metrics over image features (mean, max, min, std and mode). If it is None, the code expect that the DataFrame has only one variable.

Returns out_dataframe Geopandas dataframe with the features added.

stmetrics.spatial.**snitc**(*dataset*, *ki*, *m*, *nodata=0*, *scale=10000*, *iter=10*, *pattern='hexagonal'*, *output='shp'*, *window=None*, *max_dist=None*, *max_step=None*, *max_diff=None*, *penalty=None*, *psi=None*, *pruning=False*)

This function create spatial-temporal superpixels using a Satellite Image Time Series (SITS). Version 1.4

Parameters

- **image** (*Rasterio dataset object or a xarray.DataArray.*) – SITS dataset.
- **k** (*int*) – Number or desired superpixels.
- **m** (*int*) – Compactness value. Bigger values led to regular superpixels.
- **nodata** (*float*) – If you dataset contain nodata, it will be replace by this value. This value is necessary to be possible the use the DTW distance. Ideally your dataset must not contain nodata.
- **scale** (*int*) – Adjust the time series, to 0-1. Necessary to distance calculation.
- **iter** (*int*) – Number of iterations to be performed. Default = 10.
- **pattern** (*int*) – Type of pattern initialization. Hexagonal (default) or regular (as SLIC).
- **output** (*string*) – Type of output to be produced. Default is shp (Shapefile). The two possible values are shp and matrix (returns a numpy array).
- **window** – Only allow for maximal shifts from the two diagonals smaller than this number. It includes the diagonal, meaning that an Euclidean distance is obtained by setting window=1.
- **max_dist** – Stop if the returned values will be larger than this value.
- **max_step** – Do not allow steps larger than this value.
- **max_diff** – Return infinity if length of two series is larger.
- **penalty** – Penalty to add if compression or expansion is applied.

- **psi** – Psi relaxation parameter (ignore start and end of matching). Useful for cyclical series.

Returns segmentation Segmentation produced.

..Note:: Reference: Soares, A. R., Körting, T. S., Fonseca, L. M. G., Bendini, H. N. Simple Nonlinear Iterative Temporal Clustering. IEEE Transactions on Geoscience and Remote, 2020 (Early Access).

stmetrics.spatial.**symmetry**(*geom*)

This function computes the symmetry of a given geometry.

Symmetry is calculated by dividing the overlapping area (AO), between a polygon P and its reflection across the horizontal axis by the area of the polygon P (A_p). The range of this score falls between [0,1] and a score closer to 1 indicates a more compact and regular geometry.

$$Symmetry = AO/A_p$$

Parameters **geom** (*shapely.geometry.Polygon*) – Polygon geometry

Returns symmetry Polygon symmetry.

stmetrics.spatial.**update_cluster**

This function update clusters.

Parameters

- **img** (*numpy.ndarray*) – Input image.
- **la** (*numpy.ndarray*) – Matrix label.
- **rows** (*int*) – Number of rows of image.
- **columns** (*int*) – Number of columns of image.
- **bands** (*int*) – Number of bands (lenght of time series).
- **k** (*int*) – Number of superpixel.

Returns C_new ND-array containing updated cluster centres information.

stmetrics.spatial.**width**(*geom*)

This function computes the width of a geometry.

Parameters **geom** (*shapely.geometry.Polygon*) – Polygon geometry.

Returns width Polygon width.

stmetrics.spatial.**write_pandas**(*segmentation*, *transform*, *crs*)

This function creates a GeoPandas DataFrame of the segmentation.

Parameters

- **segmentation** (*numpy.ndarray*) – Segmentation numpy array.
- **transform** (*list*) – Transformation parameters.
- **crs** (*PROJ4 dict*) – Coordinate Reference System.

Returns gdf Segmentation as a geopandas geodataframe.

CHAPTER 10

Utility functions

Utility module for stmetrics

stmetrics.utils.**bdc2xarray**(*cube_path*, *list_bands*)

This function reads a path with BDC ARD (Brazil Data Cube Analysis Ready Data) and creates an xarray dataset.

Parameters

- **cube_path** (*string*) – Path of folder with images.
- **list_bands** (*list*) – List of bands that will be available as xarray.

Return cube_dataset Xarray dataset.

stmetrics.utils.**check_input**(*timeseries*)

This function checks the input and raises one exception if it is too short or has the wrong type.

Parameters **timeseries** (*numpy.ndarray.*) – Your time series.

Raises **ValueError** – When timeseries is not valid.

stmetrics.utils.**create_polygon**(*timeseries*)

This function converts a time series to the polar space.

If the time series has lenght smaller than 3, it can not be properly converted to the polar space.

Parameters **timeseries** (*numpy.ndarray*) – Your time series.

Return polygon Shapely polygon of time series without spikes.

stmetrics.utils.**fixseries**(*timeseries*, *nodata=-9999*)

This function ajusts the time series to polar transformation.

As some time series may have very significant noises (such as spikes), when coverted to polar space it may produce an inconsistent geometry. To avoid this issue, this function removes this spikes.

Parameters

- **timeseries** (*numpy.ndarray*) – Your time series.
- **nodata** (*int*) – nodata of the time series. Default is -9999.

Return fixed_timeseries Numpy array of time series without spikes.

stmetrics.utils.**get_list_of_points**(*timeseries*)

This function creates a list of angles based on the time series that is used to convert a time series to a geometry.

Parameters **timeseries** (*numpy.ndarray*) – Your time series.

Return list_of_observations Numpy array of lists of observations after polar transformation.

Return list_of_angles Numpy array of lists of angles after polar transformation.

stmetrics.utils.**list_metrics**()

This function lists the available metrics in stmetrics.

CHAPTER 11

Dependencies

11.1 Installing Python

The following instructions assume you have installed Python as packaged in the Anaconda Python distribution. The stmetrics in different enviroments using Anaconda.

11.2 Open Command prompt

With *Anaconda Prompt* update conda package manager to the latest version:

```
conda update conda
```

11.3 Create a conda virtual environment (recommended)

To do it, use:

```
conda create -n new_env python=3.7
```

where `new_env` is the name of the environment.

After this, activate the environment with:

```
conda activate new_env
```

11.4 Install proper 3rd-party packages

Before installing **smetrics** make sure that you have correctly installed Shapely, Rasterio and Geopandas.

To install using conda, please use:

```
conda config --add channels conda-forge

conda install shapely

conda install rasterio

conda install geopandas
```

11.5 stmetrics dependecies on Windows

11.5.1 Installing C++ compiler

As previously stated, the package requires the mingw-w64 compiler. To install mingw-w64 compiler type:

```
conda install libpython m2w64-toolchain -c msys2
```

This will install

- `libpython` package which is needed to import mingw-w64. <https://anaconda.org/anaconda/libpython>
- mingw-w64 toolchain. <https://anaconda.org/msys2/m2w64-toolchain>

Hint: `libpython` creates automatically `distutils.cfg` file, but if it failed use the following instructions to setup it manually. Go to environment Lib path in `Anaconda3\\envs\\new_env\\Lib\\distutils` and create a `distutils.cfg` file with a text editor (e.g. Notepad) and add the following lines:

```
[build]
compiler=mingw32
```

To find the correct `distutils` path, run the following lines in `python`:

```
>>> import distutils
>>> print(distutils.__file__)
```

11.5.2 Install dtaidistance package

The dtaidistance package is mandatory for stmetrics. However, due to some issues, Windows users need to compile and install directly from source. This was tested with version 1.2.4.

Caution: Make sure that you have numpy and cython already installed!

- Download the source from https://github.com/wannesm/dtaidistance
- Compile the C extensions: `python setup.py build_ext --inplace`
- Install into your site-package directory: `python setup.py install`

Caution: If OpenMP is not installed in your system you can use:

```
python3 setup.py --noopenmp build_ext --inplace
```

However, if it is not installed, make sure the C++ compiler was properly installed.

Hint: If after installation fast computation using DTW be not available, follow the steps from this page: https://dtaidistance.readthedocs.io/en/latest/usage/installation.html#from-pypi

11.6 stmetrics on Linux

At this moment we don't have reports on issues regarding ubuntu installation.

Follow the installation for 3rd paty dependecies as describe above.

For dtaidistance package just make sure your compiler has openmp and use:

```
pip install dtaidistance[numpy]
```

CHAPTER 12

Installation

stmetrics is mostly written in python, however, due to the spatial-temporal segmentation algorithm available it requires C++ compiler. Please, prior to install, check the required dependecies at *Dependencies*. Configuring this compiler is the critical step in getting the spatial-temporal algorithm running.

12.1 stmetrics on Windows

stmetrics is supported under Windows with the following caveats:

- Python 3.5 or higher
- MSVC compiler is not supported.

12.2 stmetrics on Linux

At this moment we don't have reports on issues regarding ubuntu installation.

12.2.1 Installing stmetrics

You can use pip install to obtain the latest stmetrics directly from our git repository:

```
pip install git+https://github.com/brazil-data-cube/stmetrics
```

CHAPTER 13

Spatio-temporal Metrics

In this notebook, we present some functionalities to perform Geographic Object Image Analysis (GEOBIA) with Earth Observation Data Cubes produced by the Brazil Data Cube (BDC) project using the STMETRICS package.

This notebook was designed to present some functionalities of the **stmetrics** python package. It provides the state-of-the-art features extraction methods for Satellite Image Time Series (SITS) that can be used for remote sensing time-series image classification and analysis.

Produce reliable land use and land cover maps to support the deployment and operation of public policies is a necessity, especially when environmental management and economic development are considered. To increase the accuracy of

these maps, satellite image time-series have been used, as they allow the understanding of land cover dynamics through the time.

13.1 Getting started

Our first step is to install the **"stmetrics"**. To do this you can easily run

```
!pip install git+https://github.com/brazil-data-cube/stmetrics
```

Moreover, to use the datacube produced by the BDC project we need to install the `stac.py` package. You can do it using pip.

```
!pip install stac.py
```

!pip install git+https://github.com/brazil-data-cube/stmetrics.git

[1]:
```
#import modules
import stac
import numpy
import pandas
import rasterio
import stmetrics
import multiprocessing as mp
import matplotlib.pyplot as plt

#This is just to remove some annoying possible warnings
import warnings
warnings.filterwarnings('ignore')
```

13.2 Import test image

Our test image is one of the datacubes produced by the Brazil Data Cube Project. To access the project products we will use the `stac.py` package. It can be installed using pip.

```
!pip install stac.py
```

Along with stac we must use some utility function to build a small datacube for our test. For this we use the following functions:

Hint: the **stmetrics** package uses *numpy*, *rasterio* and *geopandas* in most of its functions. Use help or check the documentation to undestand it better https://stmetrics.readthedocs.io/en/latest/.

[2]:
```
import xarray

def longlat2window(lon, lat, dataset):
    """
    Args:
        lon (tuple): Tuple of min and max lon
        lat (tuple): Tuple of min and max lat
        dataset: Rasterio dataset

    Returns:
        rasterio.windows.Window
    """
```

(continues on next page)

(continued from previous page)

```
    from pyproj import Proj

    from rasterio.warp import transform
    from rasterio.windows import Window

    p = Proj(dataset.crs)
    t = dataset.transform
    xmin, ymin = p(lon[0], lat[0])
    xmax, ymax = p(lon[1], lat[1])
    col_min, row_min = ~t * (xmin, ymin)
    col_max, row_max = ~t * (xmax, ymax)
    return Window.from_slices(rows=(numpy.floor(row_max), numpy.ceil(row_min)),
                              cols=(numpy.floor(col_min), numpy.ceil(col_max)))

def file_to_da(filepath, bbox, crop=True):
    import re
    import numpy
    import pandas
    import rasterio
    import xarray
    from affine import Affine
    from rasterio.warp import transform

    #Open image as xarray.DataArray
    da = xarray.open_rasterio(filepath)

    #find datetime
    match = re.findall(r'\d{4}-\d{2}-\d{2}', filepath)[-1]
    da.coords['time'] = match

    # Compute the lon/lat coordinates and build meshgrid
    ny, nx = len(da['y']), len(da['x'])
    x, y = numpy.meshgrid(da['x'], da['y'])

    # Rasterio works with 1D arrays
    lon, lat = transform(da.crs, {'init': 'EPSG:4326'}, x.flatten(), y.flatten())
    lon = numpy.asarray(lon).reshape((ny, nx))
    lat = numpy.asarray(lat).reshape((ny, nx))

    #add spatial coordinates to xarray
    da.coords['lon'] = (('y', 'x'), lon)
    da.coords['lat'] = (('y', 'x'), lat)

    #Crop xarray if requested
    if crop == True:

        #Get coordinates
        w, s, e, n  = bbox.split(',')

        #To crop we just use where over lat/long
        mask_lon = (da.lon >= float(w)) & (da.lon <= float(e))
        mask_lat = (da.lat >= float(s)) & (da.lat <= float(n))

        #get cropped xarray.DataArray
        da = da.where(mask_lon & mask_lat, drop=True)

    return da
```

(continues on next page)

(continued from previous page)

```
def bdc2xray(stac, collection, bbox, time, bands=['ndvi']):

    box = ",".join(str(x) for x in bbox)

    collection = stac.collection(collection)
    items = collection.get_items(filter={'bbox':box,'datetime':time, 'limit':1000})

    list_of_datasets = []

    dataset = xarray.Dataset()

    #get links to images in Brazil Data Cube
    for band in bands:

        list_of_data_arrays = []

        for item in range(len(items.features)):
            dataarray = file_to_da(items.features[item].assets[band]['href'], box )
            list_of_data_arrays.append(dataarray)

        dataset[band] = xarray.concat(list_of_data_arrays, dim='time')

        #load xarray
        list_of_datasets.append(dataset)

    bdc_xray = xarray.merge(list_of_datasets)

    return bdc_xray
```

13.3 Connect to BDC-stac

Our first step is obtain a datacube, here we will use one of the datacubes produced by the **BDC project**. To do so, we will use the stac service provided by the project and the `stac.py` package, which is also developed by the BDC team.

[3]:
```
bdc_stac = stac.STAC("http://brazildatacube.dpi.inpe.br/stac/")
bdc_stac
```

[3]:
```
stac("http://brazildatacube.dpi.inpe.br/stac/")
```

The above result display the list of collections and datacube available.

Now let's define a bouding box of our interest area. This one is located in the MATOPIBA region in Brazil. For simplication purpose, for this test only NDVI will be used.

[4]:
```
w = -45.90
n = -12.20
e = -45.20
s = -12.90
bbox = ( w,s,e,n )

my_bands = ['NDVI']
timeline = '2018-09-01/2019-08-31'
```

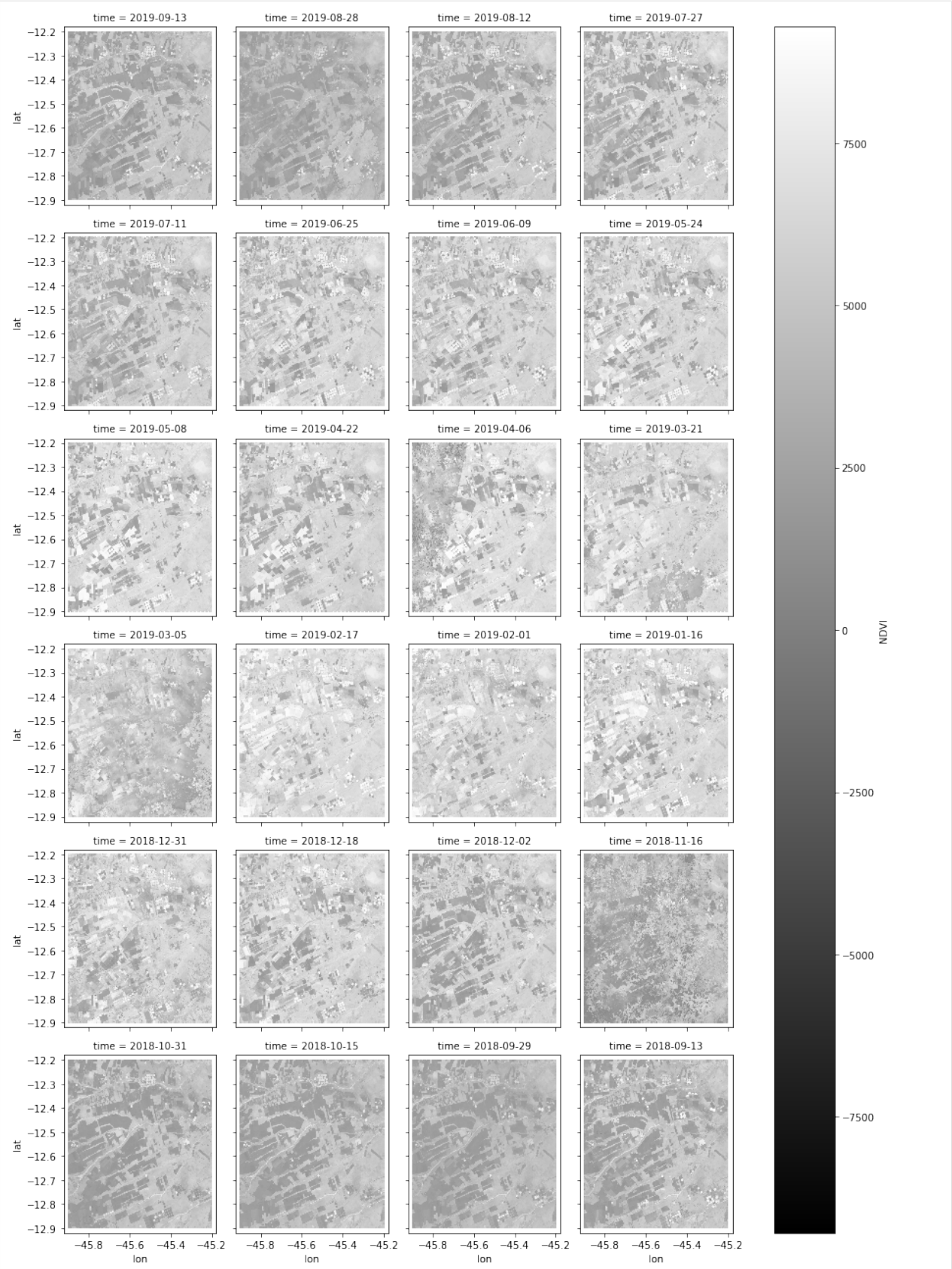
Now we will use the utility functions available above to create an **xarray.Dataset** using `stac.py`.

[5]:
```
xarray = bdc2xray(bdc_stac, "CB4_64_16D_STK-1", bbox, timeline, my_bands)
```

Let's plot the xarray

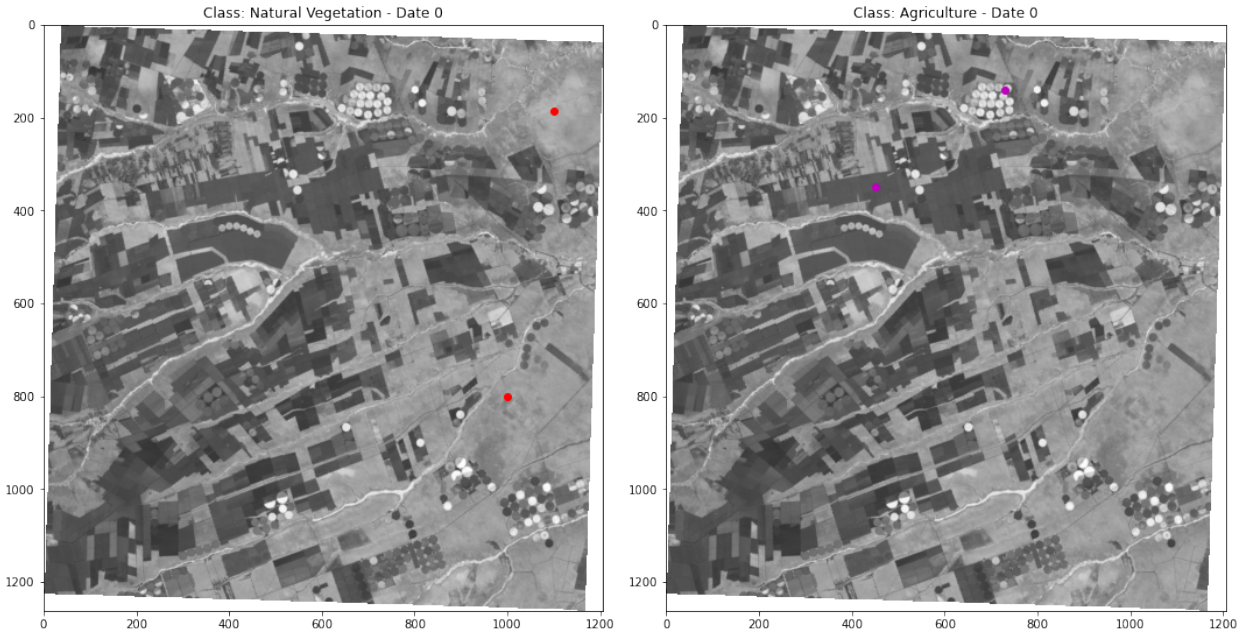
[6]:
```
xarray.NDVI.plot(x="lon", y="lat", col="time", col_wrap=4, cmap='gray')
```

[6]:
```
<xarray.plot.facetgrid.FacetGrid at 0xf72bf48>
```


time = 2019-09-13
time = 2019-08-28
time = 2019-08-12
time = 2019-07-27
time = 2019-07-11
time = 2019-06-25
time = 2019-06-09
time = 2019-05-24
time = 2019-05-08
time = 2019-04-22
time = 2019-04-06
time = 2019-03-21
time = 2019-03-05
time = 2019-02-17
time = 2019-02-01
time = 2019-01-16
time = 2018-12-31
time = 2018-12-18
time = 2018-12-02
time = 2018-11-16
time = 2018-10-31
time = 2018-10-15
time = 2018-09-29
time = 2018-09-13
lat
lon
−12.2
−12.3
−12.4
−12.5
−12.6
−12.7
−12.8
−12.9
−45.8
−45.6
−45.4
−45.2
NDVI
7500
5000
2500
0
−2500
−5000
−7500

Now lets select some sample points to be used in our examples.

[7]:
```
img = numpy.squeeze(xarray.NDVI.values)
fig= plt.figure(figsize=(15, 15))  # width, height in inches
sub = fig.add_subplot(1, 2, 1)
sub.imshow(img[0,:,:], cmap = 'gray')
plt.scatter(1100,185,color='r')
plt.scatter(1000,800,color='r')
plt.title('Class: Natural Vegetation - Date 0')
sub = fig.add_subplot(1, 2, 2)
sub.imshow(img[0,:,:], cmap = 'gray')
plt.scatter(450,350,color='m')
plt.scatter(730,140,color='m')
plt.title('Class: Agriculture - Date 0')
plt.tight_layout()
```



13.4 Time Series extraction

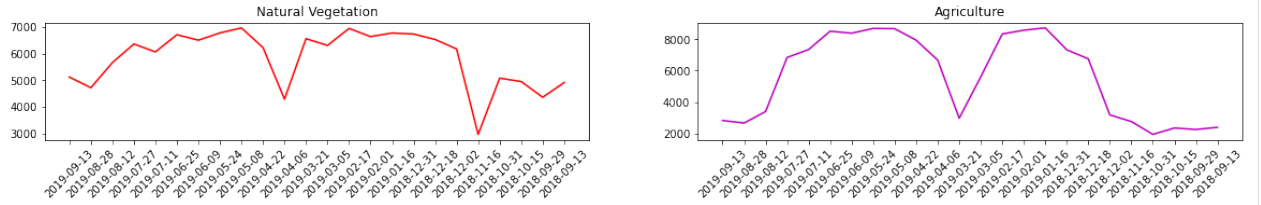
Now let's use our "samples" to extract some time series profiles.

[8]:
```
import matplotlib.dates as mdates

fn = img[:,1100,185] #Natural vegetation
agr = img[:,450,350] #Agriculture pattern

#get dates from xarray
dates = xarray.time.values

fig, ax = plt.subplots(1,2, figsize=(20,2))
ax[0].plot(dates,fn,color = 'r')
ax[0].title.set_text('Natural Vegetation')
ax[0].set_xticklabels(dates, rotation=45)
```

(continues on next page)

(continued from previous page)

```
ax[1].plot(dates,agr, color = 'm')
ax[1].title.set_text('Agriculture')
ax[1].set_xticklabels(dates, rotation=45);
```



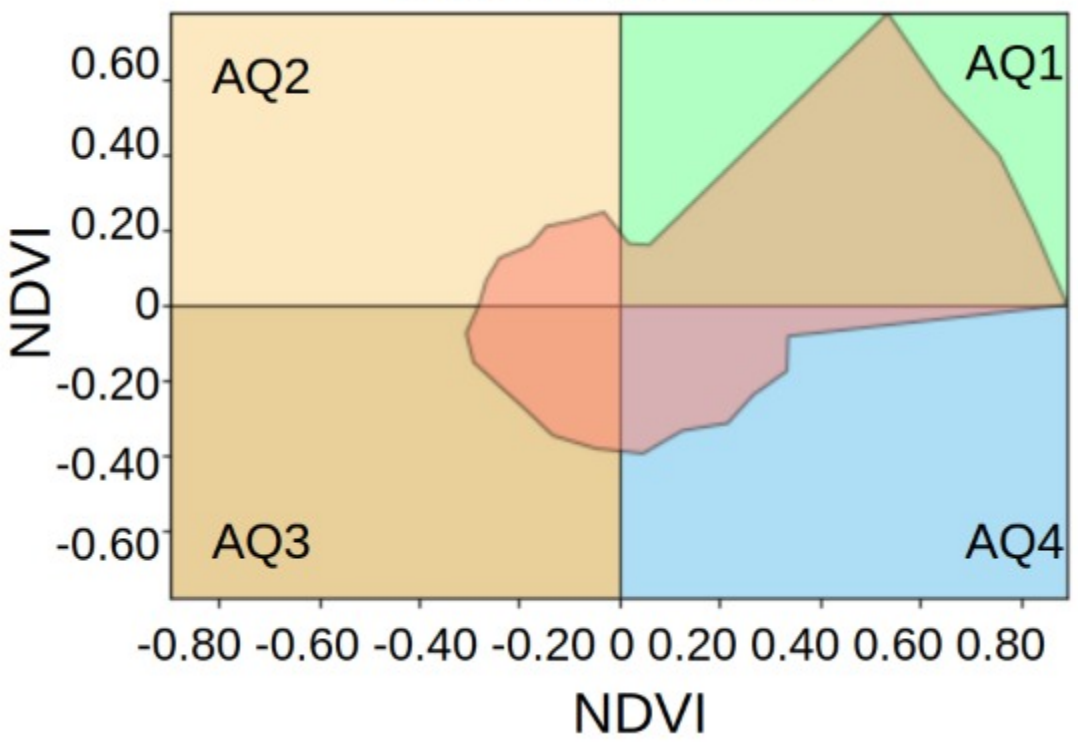
13.5 Polar plot

The stmetrics is currently composed by 4 modules:

- Metrics - With some functions to compute the all metrics available
- Basics - That has the implementation of the basics metrics
- Polar - That has the implementation of the polar metrics proposed by Körting (2013).
- Fractal - That has the implementatio of fractal metrics that are currently under assessment.

The polar approach proposed by Körting (2013) convert the time series to a polar coordinate system. This way we can plot the series in this space just to look at it and get some insights about the time series. For this, we can use the funtion `polar_plot`.

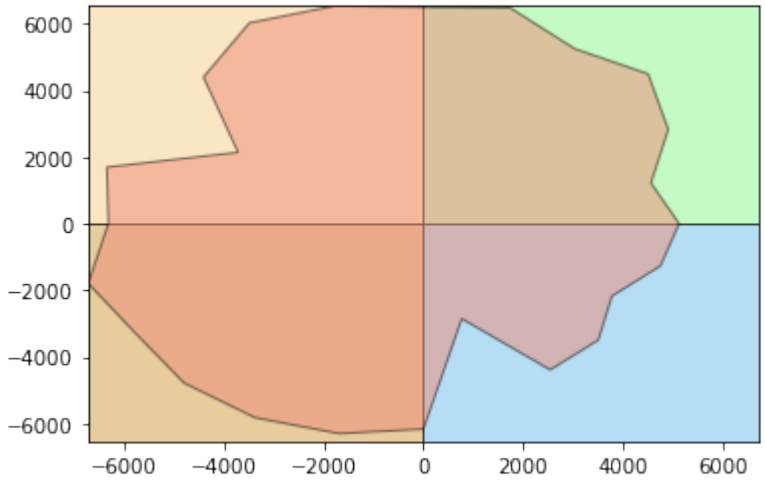
The plot as you will see is composed by the series over four quadrants. The idea here is observe a stronger response over a specific period. Imagine that your time series encompass a year of observations, this way each quadrant represents 3 months.



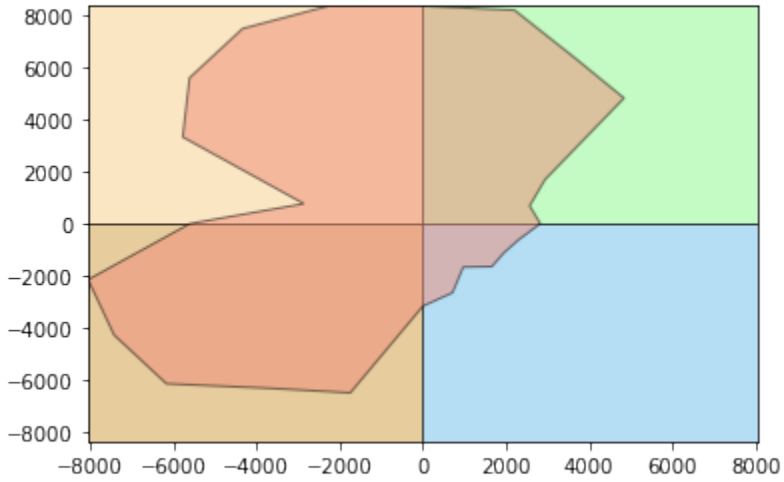
Reference: Körting, Thales & Câmara, Gilberto & Fonseca, Leila. (2013).Land Cover Detection Using Temporal Features Based On Polar Representation.

[9]:
```
#Plotting our time series samples
#Natural Vegetation
print('Natural Vegetation')
stmetrics.polar.polar_plot(fn)

print('Agriculture')
#Pasture
stmetrics.polar.polar_plot(agr)
```

```
Natural Vegetation
```



```
Agriculture
```



13.6 Metrics computation

The stmetrics has two functions to compute the metrics. The `get_metrics` function was designed to be used for compute the metrics of one time series. Along with the metrics, using the function you can also see the polar plot. Just remember, one series each time.

The `sits2metrics` function was developed to compute the metrics over images, as the name states. For this function we use multiprocessing package to improve perfomance. Don't worry if it is using the whole campacity of your system, it was designed to do it.

NOTE: This process may take sometime, tha package is computing **28 metrics!**

[10]:
```
im_metrics = stmetrics.metrics.sits2metrics(xarray)
```

[13]:
```
im_metrics
```

[13]:
```
<xarray.Dataset>
Dimensions:       (band: 1, metric: 28, time: 24, x: 1207, y: 1263)
Coordinates:
  * band          (band) int32 1
  * y             (y) float64 9.965e+06 9.965e+06 ... 9.884e+06 9.884e+06
  * x             (x) float64 5.866e+06 5.866e+06 ... 5.943e+06 5.943e+06
  * time          (time) object '2019-09-13' '2019-08-28' ... '2018-09-13'
    lon           (y, x) float64 -45.92 -45.92 -45.92 ... -45.18 -45.18 -45.18
    lat           (y, x) float64 -12.2 -12.2 -12.2 -12.2 ... -12.9 -12.9 -12.9
  * metric        (metric) object 'max_ts' 'min_ts' ... 'hurst_exp' 'katz_fd'
Data variables:
    NDVI          (time, band, y, x) float64 nan nan nan nan ... nan nan nan nan
    NDVI_metrics  (metric, y, x) float64 nan nan nan nan nan ... nan nan nan nan
```

13.7 Image Metrics

The output of `get_metrics` and `sits2metrics` follows the same order and can be acessed with the `list_metrics` function that is available at `utils` module.

[11]:
```
stmetrics.utils.list_metrics()
```

[11]:
```
['max_ts',
 'min_ts',
 'mean_ts',
 'std_ts',
 'sum_ts',
 'amplitude_ts',
 'mse_ts',
 'fslope_ts',
 'skew_ts',
 'amd_ts',
 'abs_sum_ts',
 'iqr_ts',
 'fqr_ts',
 'tqr_ts',
 'sqr_ts',
 'ecc_metric',
 'gyration_radius',
 'area_ts',
 'polar_balance',
 'angle',
 'area_q1',
 'area_q2',
 'area_q3',
 'area_q4',
 'csi',
 'dfa_fd',
 'hurst_exp',
 'katz_fd']
```

13.8 Now let's plot the results!

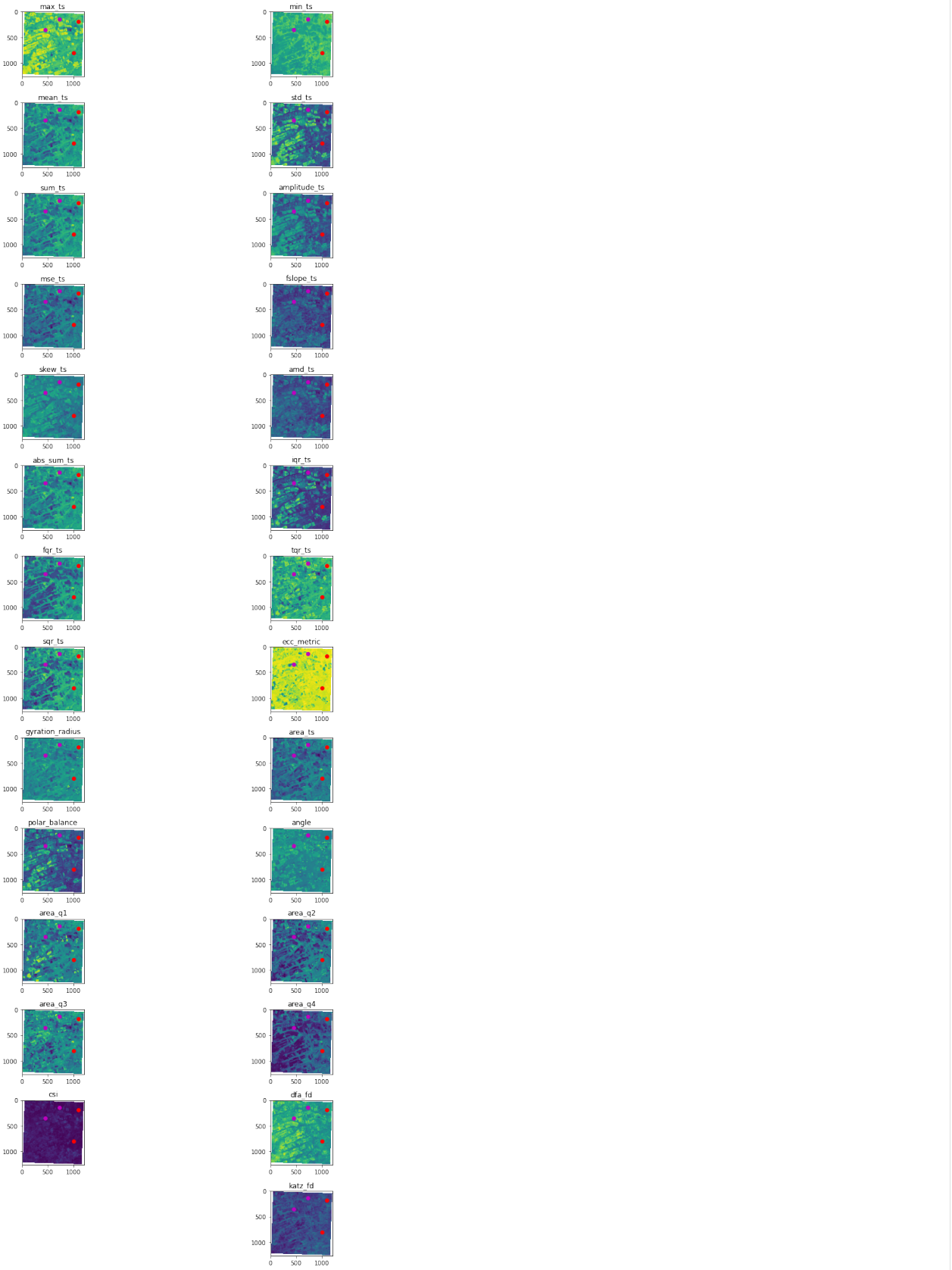
Don't worry if some metric is too noise or you don't see a clear pattern, some features are not always representative.

[19]:
```
header = stmetrics.utils.list_metrics()

img = im_metrics.NDVI_metrics.values

plt.figure(1,figsize=(15,30))

for b,n in zip(range(1, img.shape[0]+1),header):
    plt.subplot(14,2,b,)
    plt.imshow(img[b-1,:,:])
    plt.scatter(1100,185,color='r')
    plt.scatter(1000,800,color='r')
    plt.scatter(450,350,color='m')
    plt.scatter(730,140,color='m')
    plt.tight_layout()
    plt.title(n)
```

(continues on next page)

(continued from previous page)

```
plt.show()
```


max_ts
min_ts
mean_ts
std_ts
sum_ts
amplitude_ts
mse_ts
fslope_ts
skew_ts
amd_ts
abs_sum_ts
iqr_ts
fqr_ts
tqr_ts
sqr_ts
ecc_metric
gyration_radius
area_ts
polar_balance
angle
area_q1
area_q2
area_q3
area_q4
csi
dfa_fd
katz_fd
0
500
1000

13.9 That's all!

Soon we will have more examples and usages to perform time series classifications.

Keep watching our repo on github https://github.com/brazil-data-cube/stmetrics and our documentation https://stmetrics.readthedocs.io/en/latest/!

If you find any problems in the package, please, submit an issue on github.

Thanks for checking!

CHAPTER 14

Benchmark

We assessed the performance of two main functions of stmetrics: `get_metrics` and `sits2metrics`. For that, we used a core i7-8700 CPU @ 3.2 GHz and 16GB of RAM. With this test, we wanted to assess the performance of the package to compute the metrics available under different scenarios.

We compared the time and memory performance of those functions using different approaches. For `get_metrics` function, we assessed the performance using a random time series, created with NumPy, with different lengths. For the `sits2metrics` function, we used images with different dimensions in columns and rows, maintaining the same length.

14.1 Install stmetrics

!pip install git+https://github.com/brazil-data-cube/stmetrics.git

14.2 `get_metrics`

To evaluate the performance of `get_metrics` function, we implemented a simple test using a random time series built with `NumPy` package, using the following code.

[1]:
```
import time
from stmetrics import metrics
import numpy
import matplotlib.pyplot as plt
```

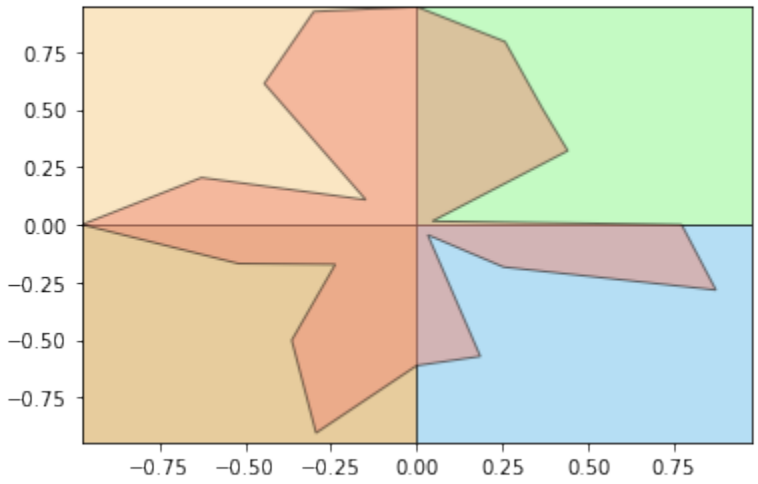
The `get_metrics` function was designed to be used for compute the metrics of one time series. The stmetrics is currently composed by 4 modules:

- Metrics - With some functions to compute the all metrics available
- Basics - That has the implementation of the basics metrics

- Polar - That has the implementation of the polar metrics proposed by Körting (2013).
- Fractal - That has the implementatio of fractal metrics that are currently under assessment.

Along with the metrics, `get_metrics` function also returns the polar plot of the inpute time series.

[2]:
```
metrics.get_metrics(numpy.random.rand(1,20)[0], show = True)
```



[2]:
```
{'basics': {'max_ts': 0.9759792296756558,
  'min_ts': 0.04808538721672284,
  'mean_ts': 0.6109056388733799,
  'std_ts': 0.29056364848092775,
  'sum_ts': 12.218112777467597,
  'amplitude_ts': 0.9278938424589329,
  'mse_ts': 9.152658668516812,
  'fslope_ts': 0.7243171421102885,
  'skew_ts': -0.54738857482128,
  'amd_ts': 0.3327714629554783,
  'abs_sum_ts': 12.218112777467597,
  'iqr_ts': 0.36797126163222327,
  'fqr_ts': 0.42877007608749085,
  'tqr_ts': 0.8754980682995873,
  'sqr_ts': 0.6213153669212852},
 'polar': {'ecc_metric': 0.9751625623899488,
  'gyration_radius': 0.832372517625857,
  'area_ts': 1.192937763415025,
  'polar_balance': 0.06548526239149761,
  'angle': 3.306939635357677,
  'area_q1': 0.3946454162385842,
  'area_q2': 0.2641239104991063,
  'area_q3': 0.3158387113661037,
  'area_q4': 0.21832972531123107,
  'shape_index': 4.293564433772728,
  'fill_rate': 0.8128851173945503,
  'fill_rate2': 0.4292998579836149,
  'symmetry_ts': 5.302775225959127},
 'fractal': {'dfa_fd': 0.30195613396763243,
  'hurst_exp': 0.7610313450623636,
  'katz_fd': 2.3097390361381698}}
```

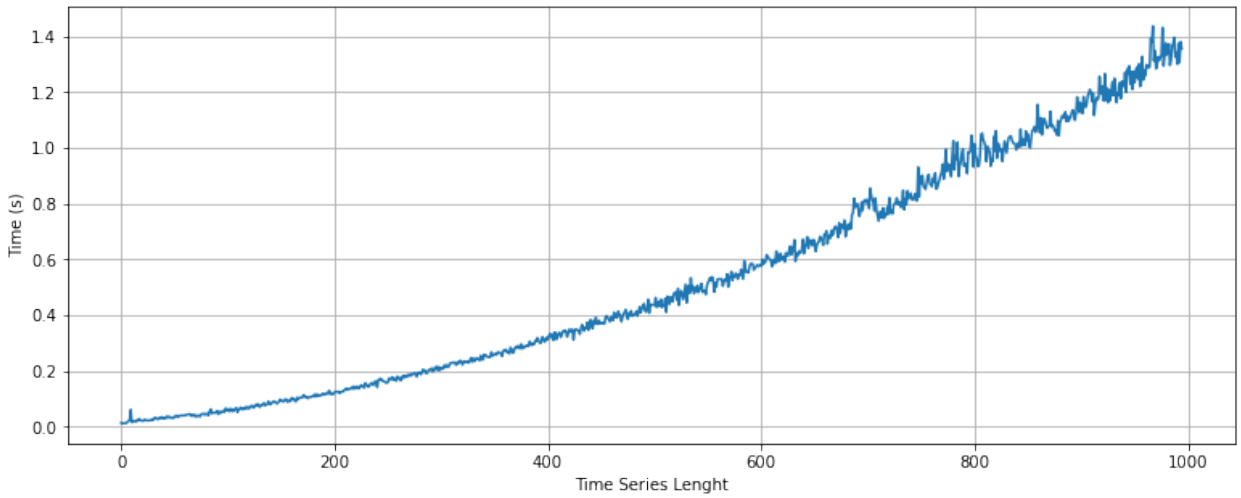
[3]:
```
tempos = []
```

(continues on next page)

(continued from previous page)

```
for i in range(5,1000):
    start = time.time()
    metrics.get_metrics(numpy.random.rand(1,i)[0])
    end = time.time()
    tempos.append(end - start)
```

[4]:
```
figure = plt.figure(figsize=(13,5))
plt.plot(tempos)
plt.ylabel('Time (s)')
plt.xlabel('Time Series Lenght')
plt.grid()
plt.show()
```



As shown above, the `get_metrics` function presents a quadratic response regarding the length of the time series. It is able to compute the metrics for a time series with 1,000 data points in less than **two second**. This beahaviour is explained by some polar metrics that requires more computational time, for example the `symmetry_ts` function. For the following versions, we will try to improve the performance of the package.

14.3 sits2metrics

To evaluate the `sits2metrics` function we used a sample image with the following dimensions: 249x394 and 12 dates. With this test, we aim to assess how the size of the image impacts the total time to compute the metrics.

This function uses the multiprocessing library to speed up the process. According to the previous test, a time series with 12 dates as our sample requires 0.015s to compute the metrics for one pixel, therefore using a single core this should require 1,318s or approximately 21minutes. With the parallel implementation, according to our tests, the package performs the same task in 6 minutes.

[5]:
```
import rasterio
```

[6]:
```
sits = rasterio.open('https://github.com/tkorting/remote-sensing-images/blob/master/
↪evi_corte.tif?raw=true').read()
```

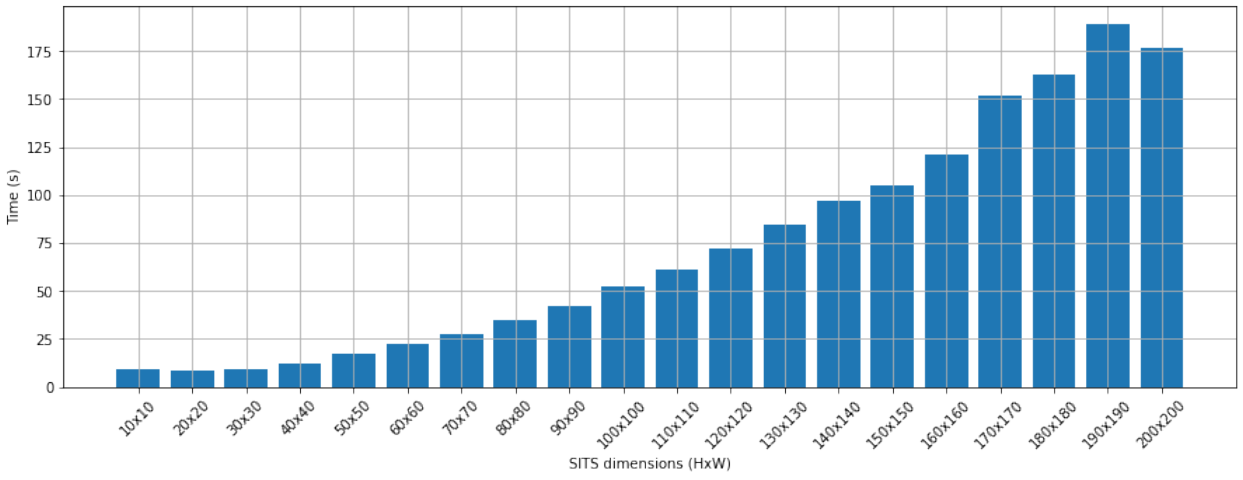
[7]:
```
tempos_sits = []
dim         = []
```

(continues on next page)

(continued from previous page)

```
for i in range(10,210,10):
    dim.append(str(i)+'x'+str(i))
    start = time.time()
    metrics.sits2metrics(sits[:,:i,:i])
    end = time.time()
    tempos_sits.append(end - start)
```

[8]:
```
fig = plt.figure(figsize=(15,5))
plt.bar(dim, tempos_sits)
plt.ylabel('Time (s)')
plt.xlabel('SITS dimensions (HxW)')
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



Note: This documentation is not finished. Parts of the description are incomplete and may need corrections. Please, come back later for the definitive documentation.

stmetrics is a python package that aims at making the process of feature extraction of state-of-the-art time-series as simple as possible.

It provides functions to support time-series analyzes that includes not only the feature extraction but also spatio-temporal analysis through a spatio-temporal segmentation and filtering approaches that are a first step to the full exploitation of the spatio-temporal information.

The documentation presented here summarize the technical aspects of the package. The methodoogical application of the package and it's methods are available at the *Publications* section.

Python Module Index

S

Index

T

U

W