MethodsX 9 (2022) 101643

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

Automatic detection of vegetation cover changes in urban-rural interface areas

Bruno Barbosa[a,b,*], Jorge Rocha[a,b], Hugo Costa[c,d], Mário Caetano[c,d]

[a] Centre for Geographical Studies, Institute of Geography and Spatial Planning, University of Lisbon, Lisbon, Portugal
[b] Associated Laboratory TERRA, Lisbon, Portugal
[c] Direção-Geral do Território, Lisbon, Portugal
[d] NOVA Information Management School (NOVA IMS), University NOVA of Lisbon, Lisbon, Portugal

A B S T R A C T

The present work started from the need to streamline the process of monitoring changes in vegetation in the in urban-rural interface fuel management bands, defined by Portuguese legislation as areas where the existing biomass must be totally or partially removed. The model developed uses a time series of Sentinel 2 satellite images to search for changes in the vegetation cover in a 100 m buffer around built-up areas. The use of satellite data allows analysing large areas and speeds up the task of identifying the places where fuel management took place and the places where there is a need to carry out such management. The objective of the proposed method is to give a script in Python language that can verify the cleanliness of vegetation in the fuel management ranges through multi-temporal analysis of satellite images.

- The paper presents a step-by-step procedure for a Sentinel 2 time series vegetation index analysis.
- Automated routine to detection of spatiotemporal vegetation changes based on statistical parameters.
- Used Python language to do geoprocessing analysis.

A R T I C L E I N F O
Method name: Statistical Meaningful Vegetation Changes
Keywords: Sentinel 2, Time Series, Vegetation index, Vegetation change, Python
Article history: Received 21 December 2021; Accepted 20 February 2022; Available online 24 February 2022

* Corresponding author at: Centre for Geographical Studies, Institute of Geography and Spatial Planning, University of Lisbon, Lisbon, Portugal.
E-mail address: bruno.misson@gmail.com (B. Barbosa).

https://doi.org/10.1016/j.mex.2022.101643

Specifications table

Subject Area:	*Geographical Sciences*
More specific subject area:	*Spatial data analysis*
Method name:	*Statistical Meaningful Vegetation Changes*
Name and reference of original method:	*Not applicable*
Resource availability:	*Python script available as supplementary material*

Used data

The data used as inputs for the model are.

- Land Use map.
- Buffer of 600 m of the built areas, in raster format.
- Buffer of 100 m of the built areas, in raster format.
- Time series of vegetation indices.

Method details

The developed method, adapted from [1], consist in search statistically significant differences across the Normalized Difference Vegetation Index (NDVI) time series of satellite data obtained from Sentinel 2, using a moving window through each pixel.

The analysis uses three time series information. For each available image we check: (i) the vegetation index value in each pixel located within the study area; (ii) the vegetation index mean value of neighbouring pixels within a given distance radius, located outside of the study area and of the same land use class; and (iii) the difference between (i) and (ii).

For each of this information and at each date of time series we applied a Welch's t-test [2]. This test identifies significant differences between the means of two data sets with different sizes and variances. The data sets are defined by a time window with a group of images from after and before each date. This groups could have a minimum of 2 dates and a maximum of 8 dates to realize the Welch's t-test. The moving temporal window is set to 60 days, to avoid comparing values of distant dates and minimise problems with months when there are few or no images with valid values, e.g., because of the presence of clouds.

A change is set to occur when the test points significant differences on the same date for the 'pixel (i)' value and the 'difference (iii)' value but not to the 'mean (ii)' value. We aim to find only the changes that indicate the decrease of the vegetation index, and this information is associated with negative values of 't'. With these criteria the pixels are classified with (1) where one change occur and (0) in the unchanged areas. For each analyzed image a Boolean raster is generated indicating change (1) and no change (0).

Script construction

For the development of this methodology, we use the Python version 3.9.2 [3]. According with [4] there are some features that make Python an ideal tool for geoprocessing analysis, like: (i) is an easy-to-learning language; (ii) is object-oriented; (iii) the abundance of available resources, because it is an open-source language, presents a vast community of programmers who make information available online; and (iv) its embedded in many geoprocessing software, e.g., ArcGIS, GRASS GIS, QGIS and PostGreSQL/PostGIS.

To achieve the proposed objectives, ten specific functions were developed that helped collect and produce the information necessary for the analysis. Developed scripts were based on modules 'os' and 'glob', gdal' 'numpy' 'scipy' 'time' and 'datetime' [5–11].

Developed functions

raster2array()

This function uses the gdal module to extract information from an input raster file. First, a variable containing a numpy array with the values of each pixel is generated and then the spatial data information is organized in a dictionary. This is the metadata with the characteristics of the selected raster file, during the script this information will be used to verify the spatial relations between the images.

world2Pixel()

This function checks the first point of intersection between two given raster files and converts this information into numbers of row and column of a numpy array. The information of the *gdal.GetGeoTransform()* object that contain the coordinates and cell size (i.e., spatial resolution) of a first raster file, e.g., (530250.0, 10.0, 0.0, 4471020.0, 0.0, -10.0), and the coordinates of the upper left corner (minimum X and maximum Y) of a second raster, are used to do that.

saveRaster()

This function converts the numpy array values into to a raster file format. Initially, a new raster file is created without information with the same format as the input template file and then fills it with the information provided by the metadata. The number of rows and columns, the projection and the geotransformation are used to establish the position of this new raster in space. Subsequently the values are filled with the data available in an input numpy array and saved in a selected folder.

clipRaster()

This function organizes three other functions created, *raster2array(), world2Pixel()* and *saveRaster()*, to generate a new raster with the data clipped of a input raster where it overlay with a second raster file selected.

window()

This function creates the (n x n) pixel window required to incorporate the desired radius size according to the spatial resolution of a raster. In the case of Sentinel-2, as each cell in the image has 10 m resolution the expression is given by:

$$n = \left(\left(\frac{radius}{10}\right) * 2\right) + 1 \tag{1}$$

arrayMean()

This function is responsible for calculating the arithmetic mean around a window of (n x n) of a pixel. Here we need three inputs information, the array with the pixel values, the mask with outside study area and the size of (n x n) window.

It starts by generating two new arrays, the first *outdata* with null values and the same size of the input array and the other a new *arrayMask*, its result by the multiplication of the input array and the input mask, because the purpose is to calculate the average of outside the study area.

After that we use two *for* looping, the first with the range of rows [i] and the second with the range of columns [j], obtained of the array shape size. The propose of that is to calculate the mean in all array cell inside a moving window of (n x n).

The Fig. 1 simplifies this idea in a matrix of 5 rows and 4 columns where we would like to calculate the mean in a window (3×3) at points P1 and P2.

P1	m1		
m1	m1		
	m2	m2	m2
	m2	P2	m2
	m2	m2	m2

Fig. 1. Array (5,4).

In Python the first value of an numpy array object is always at the index position zero, so point P1 is located at position (0,0) and its mean is obtained from the range: array[0:2, 0:2]. That is, the values between rows 0 and 1 and between columns 0 and 1. The mean of P1 is given by (P1 + 3m1)/4. Point P2 is located at position (3,2) of the array, so the values to calculate its mean are in the range: array[2:5, 1:4], that is, the values that are between rows 2 and 4 and between columns 1 and 3. The mean of P2 is given by (P2 + 8m2)/12.

This example shows that each cell has a different range that must be elaborated according to the position of the cell in array. In the case presented the window (3×3) of point P1 cannot be complete as it would involve values in the row and column -1 that do not exist. If the script tries to access the values in missing rows or columns, an empty set is generated, and the mean is not calculated. Because of this, we must lay down two rules to avoid this error:

1. If the index of rows or columns is less than zero, use zero.
2. If the index of rows or columns is greater than the maximum of rows and columns, use the maximum value.

To ensure that the generated matrix displays values only in the allowed cells, a third rule is stipulated.

3. If the input array value in row [i] and column [j] is null, the generated array *outdata* will display a null value at this same position.

Finally, when the corresponding range is generated for the row [i] and column [j], the mean is obtained using this range in the values of the array matrix. This process obtains the average of neighbouring pixels in each radius located outside the study area. This process requires processing time, as all the cells of the input matrix are covered and in each of them a new calculation is made, based on different intervals.

dicArray()

Generate a dictionary containing numpy arrays with information collected and produced from a set of rasters. The key for each item in this dictionary is a datetime object with the date of each image. Four information is associated with each key, the raster name, the vegetation index value in a pixel, the vegetation index outside mean and the difference between these two values.

The first process performed is the inversion of the array of the study area, because the average that we want to calculate considers the pixels located outside the study area. Subsequently, for each item in the given entry set raster, a dictionary key is generated, based on raster time data. To this key is associated the array containing the pixel values, obtained through the *raster2array()*. If a raster has all null values it is deleted from processing.

Then, for each land use class, multiplication between pixel values and the land use mask is made. The result of this calculation is inserted into the *arrayMean()* function with the inverted array of the study area and the window which the mean around each pixel will be calculated. This process meets the assumption of the mean of neighbours of the pixel, outside the study area in the same land use class.

sets()

This function is responsible for select which images are part of the analysis from the established time window. There is a minimum and maximum number of images used and they must respect a time limit. This function generates two lists, one containing the set of images before the analysis date, with this included, and the other the set of images after the analysis date.

arraySet()

This function is responsible for take the value of the pixel, of the mean and of the difference and store that into lists for each established date in a set. Here we use as input the dictionary created by *dicArray()* and the list created by *sets()*. As the list contains the datetime objects, which are the dictionary key, for each date of the list we can take the information contained in the dictionary and stored in a new list, which is returned at the end of the function.

welchTest

Here the script performs the Welch's t-test calculation for two sets of input data arrays. The test used here is a function belonging to the scipy module and returns the values of t and p for each item in a set of dates. In this case the function is given as: *scipy.stats.ttest_ind(arrayA, arrayB, axis = 0, nan_policy = 'omit')*. The 'axis' = 0 indicates that the calculation should be in the third dimension, i.e., between the cells located in the same position in the different input arrays. The 'nan_policy' = 'omit' indicates that the operation does not take the null values into account, this factor reduces the effect of the absence of values caused by interference with the satellite image signal. As a result, two new arrays are generated, one with the t values and the other with the p values for each analyzed cell.

Subsequently a Boolean matrix is generated where the values less than or equal to established p-Value will be (1) and all the others will be (0). These values indicate whether there has been a significant change between the values of the sets. In other words, it checks if there was a significant change in the vegetation index values between two sets of dates. The information if there was a decrease in the vegetation index is indicated by the t signal.

General script

The first step is importing all the necessary modules and generate all global variables. We chose to generate the functions presented in a separate file and import them into the script as if they were a module.

After, are collected the information of two raster used as a mask for which the images with vegetation index values will be 'cut out', represented by the buffers of 600 and 100 m of the built areas.

Subsequently, information from the land use and occupation map is collected. First it is cut to the 600 m mask around the built areas using the *clipRaster()* function. After clipping, the information of this new raster is stored in new variables via *raster2array()*.

Land use classes are individualised in a dictionary that has as keys the code of the land use classes. It is verified which land use classes exist in the study area if isn't the key is eliminated. Otherwise, a new check is made if the key in question belongs to the list of uses used in the analysis, in this case the cells where the values are equal receive value (1) and the others receive null value.

The following are cut out all vegetation index images to the 600 m mask around the built areas. With those new images is created the dictionary containing the information required, namely: the name, pixel value, outside study area mean and difference between these values, through the *dicArray()*.

Now the data are grouped for the calculation of the Welch t-test. A list is generated that will receive all the keys in the *dicData()* dictionary, using the native Python function *sorted()* we ensures that the items are entered in chronological order in the list. Subsequently, a looping is made to

perform the statistical test for each data. In this looping are deleted the first two and the last two dates. This is done because the Welch's t-test needs at least two values in each input set.

In this looping a new variable is created in each iteration with the information about day, month and year of each image analyzed. Next are set the limits of the time window, the minimum of 2 images, the maximum of 8 images in a period of maximum 60 days later and before the date of the image analyzed. The *sets()* function is used to generate two lists containing sets of earlier and later dates that obey the established limits. The data from these two generated lists will be used in the *arraySet()* function to access arrays with pixel, outside mean and difference values between them for each date of the sets. With this set is made the Welch t-test through the *welchTest()* function.

Following, operations are carried out with the Boolean arrays generated. The *bin_fnc1* variable receive the multiplication between the pixel array and difference array, and the *bin_fnc2* variable receive the inverse array of the outside mean value. These two variables are then multiplied to determine the areas of change in vegetation.

Finally, the information of the day, month and year of the datetime object is removed to compose the name of the raster that is generated through the *saveRaster()* function and saved in the specified folder.

Conclusion

The proposed process of analyzing data obtained from satellite imagery showed the versatility of using the Python language. The functions developed and the ones used in existing modules made it possible to create a logical and functional structure to achieve the desired goals. The option for a free programming language was given to avoid the use of geoprocessing software, which often has expensive licenses, and the model can be easily replicated by other users. The model proved to be efficient, as it was able to detect changes in the selected study areas. Finally, the possibility of jointly processing a large volume of information in a single environment demonstrated the potential of this approach for spatial analysis.

Acknowledgments

This work was financed by national funds through FCT—Portuguese Foundation for Science and Technology, I.P., under the framework of the Project "FORESTER - Data fusion of sensor networks and fire spread modelling for decision support in forest fire suppression" [PCIF / SSI / 0102 / 2017] and is relative to a scientific Research Fellowship BI_27_20.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.mex.2022.101643.

References

[1] M.L. Campagnolo, D. Oom, M. Padilla, J.M.C. Pereira, A patch-based algorithm for global and daily burned area mapping, Remote Sens. Environ. 232 (January) (2019) 111288, doi:10.1016/j.rse.2019.111288.
[2] B.L. Welch, The generalization of 'student's' problem when several different population variances are involved, Biometrika 34 (1947) 28–35.
[3] Python™: (2022) https://www.python.org/.
[4] L. Tateosian, Python for ArcGIS, 1, Springer International Publishing, 2015, doi:10.1007/978-3-319-18398-5.
[5] OS - Miscellaneous operating system interfaces: https://docs.python.org/3/library/os.html. (acesssed on 03/02/2022).
[6] GLOB - Unix style pathname pattern expansion: https://docs.python.org/3/library/glob.html. (acessed on 03/02/2022).
[7] GDAL/ORG: https://gdal.org/. (acessed on 03/02/2022).

[8] NumPy Developers: https://numpy.org/doc/stable/. (acessed on 03/02/2022).
[9] SciPy API: https://docs.scipy.org/doc/scipy/reference/. (acessed on 03/02/2022).
[10] TIME - Time access and conversions: https://docs.python.org/3/library/time.html. (acessed on 03/02/2022).
[11] DATETIME- Basic date and time types: https://docs.python.org/3/library/datetime.html. (acesssed on 03/02/2022).