Ruprecht-Karls-Universität Heidelberg

Fakultät für Chemie und Geowissenschaften

Geographisches Institut

SS 2019

Course: Introduction to Computer Science for Geographers

Docent: Christina Ludwig

NDVI calculation with tile-based processing

Pascal Wolf

pascal_wolf89@gmx.de

Matrikelnummer: 3547467

Master of Science Geographie

Fachsemester: 2

Table of Contents

1 Introduction

Climate change is everywhere. There are several methods to display how critical the situation is. One concerning vegetation is the Normalized Difference Vegetation Index (NDVI). The therefore valuable open source raster images are however large maybe too large for the processing hardware. This project focuses on the calculation of the NDVI by managing with large datasets.

2 Structure of the script

The script has three functions. Two functions are for the tile-wise NDVI calculation and one function is for the difference calculation. The script has also automated search for satellite images and check of user input via CMD. The end products of the script are two NDVI calculated rasterfiles with the same bounding box but different in time and one rasterfile where the difference of the two NDVI rasterfiles is calculated.

While scripting some best practices and good enough practices were applied. Good enough practices contained Data Management like saving and copying the script to code without fear (Wilson et al. 2017, p. 2). Concerning Software some processes are decomposed into functions for reusing (Wilson et al. 2017, p. 7). Best practices were making incremental changes while coding and adding Error Handling to catch expected Errors (Wilson et al. 2014, pp. 3-5).

2.1 NDVI calculation

The function tiled_writing does the NDVI calculation by splitting the raster datasets sequentially in tiles. First the red band and near-infrared band of one raster will be opened in reading mode. The basic raster metadata is copied out of one band and is applied to the output raster so the NDVI calculated raster matches the original raster scene. The dtype of the metadata will be updated to Float32 as the original Uint16 is not suitable for NDVI calculations.

The NDVI is calculated within a for-loop because the bands will be split up in tiles sequentially. To calculate the NDVI the dataset objects must be changed to an ndarray. The window parameter for the rasterio.read() function is a Cartesian product of input iterables that will be created within the function calc_tiles.

The function calc_tiles first fetches the coordinates of the upper left corner of the raster dataset. The values of the user input for the tiles size will be added in column direction and subtracted in row direction. Then the width and height of the tiles can be calculated with the index() method.

Afterwards the Cartesian product can be created out of two sequences of numbers with the range() function. The first sequence starts with 0, increments by the calculated width value and ends with the highest column value. The second sequence starts with 0, increments by the calculated height value and ends with the highest row value.

As the tiling in most cases does not cover the entire raster and some pixels remain it is necessary to create a window (bounding_window) that covers the full extent of the raster and intersect it afterwards with the tiling windows. The intersection() method returns windows that affect the remaining pixels.

Finally the window parameter will be yielded by looping through the Cartesian product. To test if it works correctly the windows can be printed by adding print(window) inside the for-loop within the tiled_ndvi function.

2.2 Difference calculation

The difference calculation was written inside the function calc_difference regarding to reusability. First the two created NDVI raster files are opened in reading mode. As in case of the written raster files the metadata is copied out of one dataset for the output raster. As the metadata has already dtype Float32, no update is needed.

When the datasets are converted to ndarrays the difference calculation can be done with the np.subtract()function. For this both dimensions of the ndarrays have to be equal. So the second dataset will be converted to ndarray by adding the out_shape parameter with width and height value of the first dataset.

Finally the difference raster can be written with the same extent of the first raster dataset.

2.3 Check User Input

There are 11 input arguments a user shall give. To check if the amount of input is correct an if-statement is integrated. It says if the amount is less than 11 the script should stop at this point and remind the user what and how many input arguments he shall give.

The script also features Error Handling by raising an Exception for the ValueError. It checks if the value for the type is correct. This is important as the types for the input values differ from each other. If the value is inaccurate the script should stop here and the user will be pointed what he did wrong.

2.4 Automated download of satellite images

The script offers automated download of satellite images. It is achieved with the help of the opensource python package satsearch. With the implied function Search() the scenes will be downloaded if the parameters lead to a result. Most of the parameters are user inputs like the coordinates of the bounding box and the date or date range. Only the cloud cover value is stable as a low coverage is viable for NDVI calculations.

As it is possible that no scene was found Error Handling is necessary or otherwise it would result in an Error. So Raising an Error is implemented. If no scenes in both searches was found the script stops here and tells the user that no Landsat scene was found.

3 Difficulties

While coding there have been several problems. One concerns the automated download of satellite images. There are two kinds of satellite scenes that can be downloaded: Landsat and Sentinel. If you want to read a downloaded Sentinel raster dataset you cannot read it as ndarray as it throws an Error (RasterioIOError).

For this the script offers a loop through the list of found scenes. It goes through the scenes properties and searches for the entry "landsat". The first found item stops the loop and the red and near-infrared band of the scene will be fetched.

Another complicacy concerns the difference calculation. As the Landsat scences have no consistent transformation matrix – different coordinates and extent – one of the NDVI calculated rasters have to be aligned to the other raster.

The first step was managed by transferring the ndarrays to the same dimension. The second step is to re-project one of the ndarray so that it fits the other. This will be achieved by the warp.reproject() function of the rasterio.warp module. For this a destination (proj_band2) has to be created with the same shape and transformation matrix of the first raster so that the source raster can be re-projected to the destination raster. Afterwards the destination and the first raster resp. ndarray are overlapping and the difference can be calculated.

4 Conclusion

The script offers the search for satellite images by letting the user decide over the search parameter, the calculation of the NDVI for both images and the difference calculation. It manages the NDVI calculation by splitting the raster datasets in tiles so that the RAM can handle with the large satellite images. But the smaller the tile size the longer the process takes. The script also manages the difference calculation for the two NDVI calculated satellite images by aligning one raster to another.

The difference calculation works primarily that the accurate values are compared. What get left out are those values that do not overlap. This means that on the output of the difference calculation there are some pixels on the borders that are not desired. A method to disable those values for the difference calculation can be considered in future work.

Publication bibliography

Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., Haddock S. H. D., Huff, K. D., Mitchell, I. M., Plumbley, M. D., Waugh, B., White, E. P., Wilson, P. (2014). Best Practices for Scientific Computing. PLOS Biology, 12(1), e1001745. https://doi.org/10/qtt

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. PLOS Computational Biology, 13(6), e1005510. https://doi.org/10/gbkbwp