SAR Surface Water Mapping Documentation

Release 1.0

Water Survey of Canada

Oct 13, 2020

CONTENTS

CHAPTER

ONE

OVERVIEW OF THE PROJECT

1.1 Introduction

The National Hydrologic Services (NHS) of Canada is responsible for the upkeep and distribution of data from some 2500 gauging stations strategically places across the country to monitor Canadian water resources. While the data these stations collect is important to the regions they are located in, a majority of water bodies and systems in this country are completely unmonitored. Many of these water bodies, especially within the prairie regions, exibit dramatic shifts in waterbody volume and extent thoughout the year, which can have significant impacts on the surrounding region. Thus, the included algorithm was developed to monitor these volatile waterbodies using SAR, specifically data from the RADARSAT Constellation Mission (RCM).

RCM is a trio of radar satellites launched in June of 2019, that began collecting data in early 2020. SAR was chosen for an operational monitoring program such as this due to its ability to image at any time of day and through cloud cover, versus optical sensors that require sunlight and direct sight of the surface. Unlike Canada's previous generations of radar satellite (RADARSAR-1 launched in 1995, and RADARSAT-2 launched in 2007), having three satellites in a constellation allows for access to nearly the entire country each day. This large accessability allows NHS to monitor highly volatile waterbodies, such as those in the prairies, on a biweekly basis with high spatial resolution products (5m).

The waterbodies themselves are detected using open-source Random Forest machine learning which, because of its robustness, is able to accommodate a variety of SAR sensors and beam-modes. The algorithm is trained on the Global Surface Water (GSW) Dataset (https://global-surface-water.appspot.com), which gives each 30x30m pixel on Earth a occurrence value between 0 and 100. These values were derrived from 30 years of Landsat data, and give a percentage of how often the pixel was water in the record. Scenes are trained on themselves using pixels within the scene determined from GSW to have a greater than 90% probability of being water. Results are then filtered to remove false-positives, and presented in both raster and vector formats. Additional details on the methods of the algorithm are presented by Millard et al. 2020.

The following documentation presents instructions to setup the algorithm, and an accumulation of code comments for reference and troubleshooting. An overview of the science behind the algorithm is presented by Millard et al. 2020.

1.2 Reference

Koreen Millard, Nicholas Brown, Douglas Stiff & Alain Pietroniro (2020): Automated surface water detection from space: a Canada-wide, open-source, automated, near-real time solution, Canadian Water Resources Journal / Revue canadienne des ressources hydriques, DOI: 10.1080/07011784.2020.1816499

CHAPTER

TWO

SETTING UP THE ALGORITHM

2.1 User Requirements:

1) A basic understanding of the command line
2) Python 3.7 (Preferably Anaconda) installed.
3) RCM, RADARSAT-2, or Sentinel-1 raw products

Optional: - Access to a GIS

2.2 Required Python Libraries

GDAL, Pandas, GeoPandas, RasterStats, SciKit-Learn, SciKit-Image, Cython, NumPy, SciPy, Requests, h5py, TensorFlow

The recommended method is to create a new Anaconda environment and install Python 3.7 with the above libraries.

2.3 Directory Setup

Besides those included with the GitHub download, the following directories will be needed for the execution of the script:

rawInput : Where raw SAR zipfiles are placed

finishedPre: Where the results of the preprocessor are placed

ouptut : Where the results of the random forest classifier are placed

h5 : Where the training data for each scene is created and temporarily stored

DEM : Directory DEM files for orthorectification are downloaded into

logs : Where log files are placed

waterTiles : Location GSW tiles for RF training are stored

tmp : Location for temporary files created by the scripts.

Make sure to edit the config file ("classify.ini") with paths to the above directories!

2.4 Configuration file

The config file used for this project is named **classify.ini**, and is contained within the main *SSWM* directory. Please see *classify.ini* for notes on what each variable represents.

Under *LaunchPreprocessor*, a satellite profile must be specified. This can be one of the following:

RS2 : Fully implemented, expects **product.xml** in main directory of data product.

RCM : Partially implemented, expects a **.tif** file in the main directory of data product (This file must be created by the user via independant preprocessing, replaces the manifest.safe). Since the RCM manifest.safe cannot be read by the libraries used in these scripts, some preprocessing (including SAR calibration) are required by the user beforehand. The script performs speckle-filtering and orthorectification, thus the user is mainly responsible for product calibration.

S1 : Partially implemented, expects manifest.safe in main directory of data product. WARNING: Extensive testing with Sentinel-1 has yet to be performed, use this mode at your own risk!

2.5 Other Data Requirements

GSW Tiles

The training data for the random forest classifier, GSW tiles, are located here: https://global-surface-water.appspot.com/download Download any tiles needed for your study area and place them into the waterTiles directory.

NOTE: The scripts expect the GSW tiles in a specific naming convension (occurrence_*LongBoundary*_*LatBoundary*.tif, Ex. occurrence_100W_50N.tif). If changes in the naming convension of these tiles occurrs in the future, the tiles will have to be renamed to this convension in order for the scripts to run properly.

DEM Files

DEM tiles are used in the preprocessor to orthorectify the scenes. The DEM tiles required for each scene are automatically determined and downloaded by the script! This is done in the *DEM.py* (documented below) and *NTS.py* (undocumented!) scripts. Within *DEM.py*, there are three choices of DEM, one of which must be configured. The three options are:

CDED [Canadian Digital Elevation Data] Only use if study area is fully within Canadian lands, if any portion of the scene exists outside of the Canadian landmass, the script will crash!

SRTM [Shuttle Radar topography mission (https://e4ftl01.cr.usgs.gov/MEASURES/SRTMGL1.003/2000.02.11/,] requires an account with https://urs.earthdata.nasa.gov/) Available world-wide. Place login and password in the header of the **DownloadSRTM** function in *DEM.py* (Change *login* and *password* from **None** to your own)

NED [USGS National Elevation Dataset (https://catalog.data.gov/dataset/usgs-national-elevation-dataset-ned)] Available for North America Only

You can select the appropriate DEM type for the script to download by changing the DEM type to either CDED, SRTM, or NED in the config file.

Cython

The filtering procedured conducted on the imagery during the preprosessing utilizes the Cython library to increase efficiency. It is highly recommended that you read up on this module here: (https://cython.org/) as it can be quite tricky to setup properly. This document will not go over how to setup Cython, but will describe how to setup the **PSPOL** module to run properly. The **PSPOL** module requires Cython inorder to run, and setting it up is as easy as the following:

1) Navigate to the **PSPOL** directory in the terminal.

2) In the terminal, execute the following command: "python setup.py build_ext –inplace" (Make sure the "python" command in the terminal is set to the same python executable you will be running the remaining code with)

If all goes well, **PSPOL** should be ready to go with no further action required. If an error message occurrs while running the above command, it will be related to the installation of Cython dependancies. Please see the Cython page for help.

2.6 Launching the Tasks

As noted before, there are two "tasks" associated with this package: Preprocessor and RandomForestClassifier. Each of these tasks has a section within the config file ("Classify.ini"). Launching of both tasks begins by running the script *check_directory.py*. In the first few lines of this script (Line 17-18) is an area for user input. The path to the config file, along with the task to be launch need to be specified. Input your path to "Classify.ini", then set **task** equal to either "LaunchPreprocessor" or "LaunchClassifier" to run the preprocessor and rf classifier respectivily. Once these two lines are configured, save the file and run it. This should result in the creation of a *manifest.txt* file within the data directory for the given task (either rawInput or finishedPre for LaunchPreprosessor and LaunchClassifier respectively) listing the files to process. The chosen job is then automatically called by this script, and is continually run until all files listed in the manifest are processed. Once the code finishes, if another task is to be run, simply change the **task** on line 18 of *check_directory.py*, save the script, and run it again.

CHAPTER

THREE

DOCUMENTATION

3.1 Launch

Checks a directory for the existence of certain files.

`launch_preprocess.`**`preprocess`**(*folder*, *DEM_directory*, *finished_result*, *DEMType*, *logger*, *satellite='RS2'*)

Run preprocessing on next file and move results to correct directory

Gets the next file from the manifest, processes it and moves the results to the target folders.

Parameters

folder : str

DEM_directory [str] Path to directory containing DEM files in appropriate folder hierarchy

finished_raw [str] Path to folder where original file should be moved after processing

finished_result [str] Path to folder where results should be saved

DEMType: DEM type the preprocessor will use (See documentation for options)

satellite [str] Which satellite profile should be used. One of “RS2” or “RCM”

`launch_preprocess.`**`untar`**(*cur_file*, *folder*, *s1=False*)

Extract files from archive

Returns

tuple

(1) Path to extracted VRT file

(2) Path to working directory containing VRT and image files

This script is used to create a hdf5 file from an image, train a random forest clasifier and then classify the image

`launch_forest.`**`clean_up`**(*extracted_directory*)

Remove extracted files and move archive to backup directory

`launch_forest.`**`failure`**(*output_h5*, *exdir*, *cur_file*, *images_output_dir*, *msg*)

Create a flag file to indicate that the processing was aborted.

`launch_forest.`**`get_cur_file`**(*folder*)

Read current file from textfile manifest

Parameters

folder [str] Path to job directory containing manifest and preprocessed image archives

Returns

tuple

(1) Path at which to create next file to process

(2) Directory to which the archived files should be extracted

`launch_forest.`**`untar_VRT`**(*cur_file*)

Extract files from archive

Returns

tuple

(1) Path to extracted VRT file

(2) Path to working directory containing VRT and image files

3.2 PreProcess

This file contains functions to download and mosaic DEM tiles from a variety of providers

`DEM.`**`DEMproj4`**(*product*)

return proj4 string or equivalent for DEM product

`DEM.`**`NED_tile_name`**(*lon*, *lat*, *fext=''*, *v='2013'*)

Build name of NED DEM file

Parameters

lon [int] Longitude (whole number) for corner of DEM tile

lat [int] Lattiude (whole number) for corner of DEM tile

fext [str] File extension to append

v [str] Which version of the dataset to use. One of [“2013”, “2017”]. Not all tiles are available in each version.

Returns

str File name of DEM tile for NED

Example

NED_tile_name(-110, 49, “.zip”, “2017”) NED_tile_name(-110, 49, “.zip”, “2013”)

`DEM.`**`NED_tiles_from_extent`**(*ext*)

Get a list of NED tiles required to cover a spatial extent

Parameters

ext [dict] Dictionary with the following keys: {xmin, xmax, ymin, ymax} corresponding to the spatial extent in WGS84 decimal degrees

Returns

list List of tile names required to cover specified spatial extent

Examples

E = {‘xmin’: -110 ,’xmax’: -108,’ymin’: 48 ,’ymax’: 51 } NED_tiles_from_extent(E)

`DEM.`**`NTS_tiles_from_extent`**(*ext*, *scale=1*)

Determine which NTS tiles are required to cover a target spatial extent

Parameters

ext [dict] Dictionary with the following keys: {xmin, xmax, ymin, ymax} corresponding to the spatial extent in WGS84 decimal degrees

scale : int

Examples

ext = {'ymin': 52, 'ymax': 53, 'xmin' : -114, 'xmax' : -112} NTS_tiles_from_extent(ext)

DEM.**SRTM_tile_name**(*lon*, *lat*)
Build name of SRTM DEM file

Parameters

lon [int] Longitude (whole number) for corner of DEM tile. Negative numbers correspond to W

lat [int] Lattiude (whole number) for corner of DEM tile. Negative numbers correspond to

Returns

str File name of DEM tile for SRTM

Example

SRTM_tile_name(-110, 49)

class DEM.**SessionWithHeaderRedirection**(*username*, *password*)

rebuild_auth(*prepared_request*, *response*)
When being redirected we may want to strip authentication from the request to avoid leaking credentials. This method intelligently removes and reapplies authentication where possible to avoid credential loss.

DEM.**create_DEM_mosaic**(*DEM*, *DEM_dir*, *dstfile*, *product='NED'*, *vrt_only=False*, *format='GTiff'*)
Create a Mosaic from a list of DEM urls or NTS tiles. Missing tiles will be downloaded

DEM.**create_DEM_mosaic_from_extent**(*ext*, *dstfile*, *DEM_dir*, *product='CDED'*, *vrt_only=False*)
Generate DEM mosaic covering a extent

Parameters

ext [dict] Dictionary with the following keys: {xmin, xmax, ymin, ymax} corresponding to the spatial extent in WGS84 decimal degrees

dstfile [str] Path to file to create

DEM_dir [str] Directory where DEM files are saved

product [str] DEM source to use. One of ("NED", "CDED").

vrt_only [boolean] Whether to create a VRT as an output file or

Returns

str Path to mosaicked DEM

Examples

from os import path home = path.expanduser('~') ext = {'ymin': 52, 'ymax': 53, 'xmin' : -114, 'xmax' : -112} create_DEM_mosaic_from_extent(ext,

dstfile = path.join(home, 'mosaic.tif'), DEM_dir = path.join(home, 'DEM'), product = "CDED", vrt_only = False)

DEM.**degree_tiles_from_extent**(*ext*, *tile_function*, *xoff=0*, *yoff=0*)
Get a list of raster tiles required to cover a spatial extent

Parameters

ext [dict] Dictionary with the following keys: {xmin, xmax, ymin, ymax} corresponding to the spatial extent in WGS84 decimal degrees

tile_function [function] function that takes lon, lat as keyword arguments and returns tile name

Returns

list List of tile names required to cover specified spatial extent

Examples

E = {'xmin': -110 ,'xmax': -108,'ymin': 48 ,'ymax': 51 } NED_tiles_from_extent(E)

`DEM.`**`downloadSRTM`**(*url*, *destfile*, *username=None*, *password=None*, *retry=5*)
Downloads an SRTM tile.

Parameters

url [str] path to SRTM tile

username [str (optional)] USGS Earthdata username. If missing, looks for the existence of a .netrc file in your home directory

password [str (optional)] USGS Earthdata username. If missing, looks for the existence of a .netrc file in your home directory

retry [int] how many times to retry downloading

If username / password are not provided, the function requires a netrc file in your home directory (named either '_netrc' or '.netrc')

with the following contents: machine <hostname> login <login> password <password>

`DEM.`**`download_and_unzip`**(*url*, *destfile*, *exdir*, *rmzip=True*)
Downloads and unzips a file

Parameters

url [str] Url path

destfile [str] Filepath of output zipfile

exdir [str] The directory to which files are extracted

rmzip [boolean] Whether or not to remove zipfile after extraction.

Returns

str path(s) to target tiles

`DEM.`**`download_multiple_DEM`**(*DEM*, *DEM_dir*, *product='NED'*)
Download a list of DEM URLs. If they exist already, they are not downloaded

Parameters

DEM [list] List of DEM urls (NED) or NTS tiles (CDED)

DEM_dir [str] Path to which files are downloaded

product [str] Which DEM tile series should be downloaded: ('NED', 'CDED')

Returns

list a list of file paths for target DEMs

`DEM.`**`download_single_DEM`**(*DEM_id*, *DEM_dir*, *replace=False*, *product='NED'*)
Download a DEM tile

Parameters

DEM_id [str] Name or NTS sheet of tile to download. If product is "NED" or "SRTM", a name should be specified, but if product is "CDED", then a NTS sheet should be.

DEM_dir [str] Path to which files are downloaded

replace [boolean] Whether or not existing files should be re-downloaded and overwritten

product [str] Which DEM tile series should be downloaded: ('NED', 'CDED')

Returns

list List of file paths to DEM files. There may be more than one file per single zipped tile.

DEM.**egm96_to_wgs84_heights**(*dem*, *geoid*)
Convert heights above the EGM96 geoid to heights above the WGS84 ellipsoid, for example, as required to use the RADARSAT-2 rational function model.

DEM.**get_spatial_extent**(*raster_path*, *target_EPSG=4326*, *tol=0.1*)
Get the spatial extent of a raster file.

If the file is not georeferenced (e.g. for raw radarsat 2), this function attempts to use the GCPs in the image . Howver, this doesn't always produce exact results, so it is advisable to use an extra buffer tolerance in your spatial extent (maybe ~0.1 decimal degrees)

Parameters

raster_path [str] Path to raster for which a spatial extent is desired

target_EPSG [int] EPSG code specifying coordinate system for output file

tol [float] By how many decimal degrees to buffer spatial extent

Returns

dict Dictionary with the following keys: {xmin, xmax, ymin, ymax} corresponding to the spatial extent in WGS84 decimal degrees

DEM.**get_tile_path_CDED**(*NTS*)
Get FTP path for a CDED NTS tile

Parameters

NTS [str] Name of NTS sheet for which a DEM is desired

Returns

str FTP location for CDED DEM tile

get_tile_path_CDED("079D01") get_tile_path_CDED("079D")

DEM.**get_tile_path_NED**(*lon=None*, *lat=None*, *name=None*, *test=True*)
Get http path to download a USGS NED tile over a particular coordinate

Only valid over North America (lon is always assumed to be positive)

Parameters

lon [int] Longitude (whole number) for corner of DEM tile

lat [int] Latitude (whole number) for corner of DEM tile

name [str, optional] If provided, use the name directly to produce link

test [boolean] Whether or not to test whether or not the file path is valid. There are two possible nomenclature styles for NED tiles. If test is False then the link may be invalid.

Returns

str HTTP location for CDED DEM tile

Example

get_tile_path_NED(112, 56) get_tile_path_NED(-112, 56) get_tile_path_NED(name='n56w112')

`DEM.`**`get_tile_path_SRTM`**`(`*-110*, *49*`)`

`filters.`**`energy`**`(`*img*, *window*`)`

Apply energy texture filter to an image. Does not modify original.

Parameters

img [numpy array] Array to which filter is applied

window [int] Size of filter

Returns

array Filtered array with float32 datatype (done internally to avoid overflow)

`filters.`**`enhanced_lee`**`(`*img*, *looks*, *window=7*, *df=1*`)`

Apply enhanced lee filter to image. Does not modify original.

Enhanced lee filter following Lopes et al. (1990) (PCI implementation)

Parameters

img [numpy array] Array to which filter is applied

window [int] Size of filter

looks [int] Number of looks in input image

df [int] Number of degrees of freedom

Returns

array

R = Im for Ci <= Cu R = Im * W + Ic * (1-W) for Cu < Ci < Cmax R = Ic for Ci >= Cmax where: W = exp (-Damping Factor (Ci-Cu)/(Cmax - Ci)) Cu = SQRT(1/Number of Looks) Ci = S / Im Cmax = SQRT(1+2/Number of Looks) Ic = center pixel in the kernel Im = mean value of intensity within the kernel S = standard deviation of intensity within the kernel

`filters.`**`enhanced_lee_filter`**`(`*d*, *rg_win*, *az_win*, *nlooks*, *damp=1*`)`

PCI-style Enhanced Lee filter after Lopes, 1990.

`filters.`**`filter_image`**`(`*file*, *output=None*, *filter='lee'*, ***kwargs*`)`

Parameters

file [str][] File to filter (may have multiple bands)

filter [str] Name of filter to use on each band

output [str] Path to output file. If none, overwrites input file

`filters.`**`lee_filter`**`(`*img*, *window=5, 5*`)`

Apply a Lee filter to a numpy array. Modifies original array

Parameters

img [numpy array] Array to which filter is applied

window [int] Size of filter

`filters.`**`lee_filter2`**(*img*, *window=3, 3*)

Apply a Lee filter to a numpy array. Does not modify original.

Code is based on: https://stackoverflow.com/questions/39785970/speckle-lee-filter-in-python

PCI implementation is found at http://www.pcigeomatics.com/geomatica-help/references/pciFunction_r/python/P_fle.html

Parameters

img [numpy array] Array to which filter is applied

window [int] Size of filter

Returns

array filtered array

`filters.`**`moving_window_sd`**(*data*, *window*, *return_mean=False*, *return_variance=False*)

This is Ben's implementation Calculate a moving window standard deviation (and mean)

`filters.`**`window_stdev`**(*img*, *window*, *img_mean=None*, *img_sqr_mean=None*)

Calculate standard deviation filter for an image

Parameters

img [numpy array] Array to which filter is applied

window [int] Size of filter

img_mean [array, optional] Mean of image calculated using an equally sized window. If not provided, it is computed.

img_sqr_mean [array, optional] Mean of square of image calculated using an equally sized window. If not provided, it is computed.

The function is based on code from: http://nickc1.github.io/python,/matlab/2016/05/17/Standard-Deviation-(Filters)-in-Matlab-and-Python.html

`orthorectify.`**`orthorectify_dem_rpc`**(*input*, *output*, *DEM*, *dtype=None*)

Orthorectify raster using rational polynomial coefficients and a DEM

Parameters

input [str] Path to image to orthorectify

output [str] Path to output image

DEM [str] Path to DEM

dtype [int] GDAL data type for output image (UInt16=2, Float32=6 etc.)

Returns

boolean True if it completes sucessfully

`preprocess.`**`preproRCM_bd`**(*folder*, *DEM_dir*, *cleanup=True*, *product='CDED'*, *filter=True*)

Preprocess RCM scenes that have been converted to *.tif files

This assumes files have been converted to (amplitude * 20k) values and have embedded GCPs

Parameters

folder [str] Path to product.xml file for Radarsat-2 image

DEM_dir [str] Path to directory containing DEM files in

cleanup [boolean] Whether intermediate files should be deleted

product [str] Which DEM product to use.

Returns

str Path to zipped output files

`preprocess.`**`preproRS2`**(*product_xml*, *DEM_dir*, *cleanup=True*, *product='CDED'*)
Preprocess Radarsat-2 file in preparation for classification

Parameters

product_xml [str] Path to product.xml file for Radarsat-2 image

DEM_dir [str] Path to directory containing DEM files in appropriate folder hierarchy

cleanup [boolean] Whether intermediate files should be deleted

product [str] Which DEM product to use.

Returns

str Path to zipped output files

`preprocess.`**`preproS1`**(*folder*, *DEM_dir*, *cleanup=True*, *product='CDED'*)
Preprocess Radarsat-2 file in preparation for classification

Parameters

product_xml [str] Path to product.xml file for Radarsat-2 image

DEM_dir [str] Path to directory containing DEM files in appropriate folder hierarchy

cleanup [boolean] Whether intermediate files should be deleted

product [str] Which DEM product to use.

Returns

str Path to zipped output files

`preutils.`**`ProcessSLC`**(*product_xml*)
Convert SLC values to raw DN values

Checks whether a RS-2 product.xml file is associated with SLC data and if so, converts the two-channel (i,q) *.tif images into single-channel (amplitude) images. Also updates the product.xml file data type attribute from 'Complex' to 'Magnitude Detected'

Parameters

product_xml [str] file path pointing to product.xml file

Returns

boolean True if completed successfully

`class` `preutils.`**`RCM`**
Class for accessing information about an RCM dataset

`classmethod path_to_xml`(*folder*)
given a standard folder with RCM data, find the product.xml file

`class` `preutils.`**`RS2`**
Class for accessing information about an RS2 dataset

`classmethod lut`(*product_xml*, *norm='Sigma'*)
given product_xml path, find calibration LUTs norm : str

one of 'Beta', 'Gamma', 'Sigma' (default)

classmethod path_to_xml (*folder*)
given a standard folder with RS-2 data, find the product.xml file

classmethod product_xml_imagery_files (*xml*)
Return a list of which imagery files are associated with a RS-2 product.xml file

classmethod product_xml_pol_modes (*xml*)
Return a list of polarization modes associated with an RS-2 product.file

class preutils.**Radar**
Generic class for RS2 and RCM folder structures

classmethod TIF_channels (*tif*)
Get count how many channels are in an image

preutils.**ReIm2Amp** (*re*, *im*, *inplace=True*)
Convert complex components to their modulus

Addes small value to the result to ensure it is positive because the code is based on the SLC2IMG algorithm from PCI such that DN = int(sqrt(I*I + Q*Q) + 0.5)

Parameters

re [numpy array] Numpy array of shape (m,n) corresponding to the real component of a complex number.

im [] Numpy array of shape (m,n) corresponding to the imaginary component of a complex number.

inplace [boolean] Whether the inputs should be modified in-place. If true, the final result is stored in the re array

Returns

array Modulus of real and imaginary arrays (with shape [m,n])

class preutils.**S1**
Class for accessing information about an RCM dataset

preutils.**SLC2IMG** (*image_file*, *output*)
Convert SLC (Re, Im) raster to amplitude.

The code is based on the SLC2IMG algorithm from PCI such that DN = int(sqrt(I*I + Q*Q) + 0.5)

Parameters

image_file [str] Path to imagery file, usually a tiff

output [str] Path to output file

preutils.**calibrate** (*array*, *lut*, *complex=False*, *scale=20000*)
apply LUT calibration to a radar array. Modifies array in-place

Parameters

array [array-like] m x n array of raw DN values or modulus for SLC (DNi**2 + DNq**2)**0.5

lut [str] path to xml for LUT

complex [bool] does the array represent the modulus of complex (SLC) data?

scale [int] scaling factor used to store results as int16 (default is 20000, same as CIS for visual interpretation)

preutils.**calibrate_in_place** (*file*, *lut*, *complex*, *scale*, *band=[1]*)
Apply LUT calibration to file and change in-place

preutils.**cloneRaster** (*img*, *newRasterfn*, *ret=True*, *all_bands=True*, *coerce_dtype=None*, *copy_data=False*)
make empty raster container from gdal raster object. Does not copy data

Parameters

img [osgeo.gdal.Dataset] An open gdal raster object

newRasterfn str Filename of raster to create

ret [boolean] Whether to return a file handle. If False, closes file

all_bands [boolean] Whether or not all bands should be copied or just the first one

Returns

a handle for the new raster file (if ret is True)

`preutils.`**`copy_band_metadata`**(*src*, *dst*, *bands*)
Copy band metadata from one osgeo.gdal.Dataset to another

Parameters

src [osgeo.gdal.Dataset] An open gdal raster object

dst [osgeo.gdal.Dataset] A gdal raster object that is open for writing

bands [int] How many bands are in the image

`preutils.`**`copy_georeferencing`**(*src*, *dst*)
Copy geotransform and/or GCPs from one osgeo.gdal.Dataset to another

Parameters

src [osgeo.gdal.Dataset] An open gdal raster object

dst [osgeo.gdal.Dataset] A gdal raster object that is open for writing

`preutils.`**`copy_metadata`**(*src*, *dst*)
Copy metadata from one osgeo.gdal.Dataset to another

Parameters

src [osgeo.gdal.Dataset] An open gdal raster object

dst [osgeo.gdal.Dataset] A gdal raster object that is open for writing

`preutils.`**`createvalidpixrast`**(*img*, *dst*, *band*)
Create valid pixel raster (0 or 1) for a gdal raster band

`preutils.`**`get_blocksize_options`**(*img*)
Get raster blocksize information as a string that can be passed to gdal

`preutils.`**`incidence_angle_from_gains`**(*beta_gains*, *sigma_gains*, *complex=False*)
calculate incidence angle array

`preutils.`**`incidence_angle_from_xml`**(*beta_xml*, *sigma_xml*, *nrow*, *complex=False*)
Calculate incidence angle from lutBeta and lutSigma xml files

Parameters

beta_xml [str] path to lutBeta.xml file

sigma_xml [str] path to lutSigma.xml file

nrow [int] number of rows in output array (number of lines in original image)

complex [boolean] whether or not the xml files represent complex data (in which case beta and sigma values are squared before dividing)

Returns

an array of dimension (M,N) with M = nrow and N = the number of values represented by each the xml files.

`preutils.`**`interpolate_steps`**`(`*array*, *step*`)`

interpolate array with desired step size

`preutils.`**`interpolator`**`(`*y*`)`

Interpolate missing (nan) values in an array

Parameters

y [array] Array which may contain nan values

Returns

array equal-length array with nan values replaced with imputed data

Example

import numpy as np a = np.array([1, np.nan, np.nan, 4, np.nan, 6, 7], dtype='float32') interpolator(a)

`preutils.`**`read_calibration_gains`**`(`*xml*`)`

Read calibration info from RCM or RS2 lut*.xml

Parameters

xml [str] Path to look-up table (e.g. sigma.xml)

Returns

tuple tuple consisting of: (1) gains (array) (2) offset (int) (3) stepsize (int)

`preutils.`**`read_lut_array`**`(`*xml*, *nrow*`)`

`preutils.`**`reproject_image_to_master`**`(`*master*, *src*, *dst*`)`

This function reprojects an image (`src`) to match the extent, resolution and projection of another (`master`) using GDAL. The newly reprojected image is a GDAL VRT file for efficiency. A different spatial resolution can be chosen by specifyign the optional `res` parameter. The function returns the new file's name.

Parameters

master: str A filename (with full path if required) with the master image (that that will be taken as a reference)

src: str A filename (with path if needed) with the image that will be reprojected

res: float, optional The desired output spatial resolution, if different to the one in `master`.

Returns

The reprojected filename

code credit: https://github.com/jgomezdans/eoldas_ng_observations

`preutils.`**`write_array_like`**`(`*img*, *newRasterfn*, *array*, *dtype=6*, *ret=True*, *driver='GTiff'*, *copy_metadata=False*`)`

write numpy array to gdal-compatible raster.

Parameters

img [osgeo.gdal.Dataset or str] An open gdal raster object or path to file

newRasterfn [str] Filename of raster to create

array [array] array to be written with shape (nrow[y], ncol[x], band)

dtype [int] What kind of data should raster contain?

ret [logical] Whether to return a file handle. If false, closes file

Returns

osgeo.gdal.Dataset a handle for the new raster file

3.3 Forest

For RandomForest water classification of radar images

class forest.**imgchunker**(*img*, *by_y=5000*)

Splits image into chunks for memory-safer processing

This object takes raster arrays of dimension (m,n,p) and yields 'chunks' with dimension (i, j) with 1 < i < m*n and 1 < j < p. The smaller chunks can then be classified without running out of memory.

The last chunk is usually smaller than the rest unless by_y is chosen to evenly divide the number of image rows.

Parameters

img [str] Path to gdal-compatible raster image

by_y [int] How many rows should be returned during each iteration

build_band_dict(*img*)

Get indices of image bands that will be used in classification

chunkerator()

Generate image chunks for classification

During the classification process, this function 'feeds' the classifier pieces of the input image. The last piece of the image is usually smaller than the rest.

Yields

tuple A tuple containing (1) An array corresponding to a chunk of the input image, and (2) the y-offset of the chunk relative to the upper-left corner of the original image.

static get_chunk(*img*, *ix*, *offx*, *offy*, *lnx*, *lny*)

Get a slice of an image for classification.

Images are classified in pieces to prevent memory overflow

Parameters

ix [list] indices (1-based) for image bands that are to be used.

offx [int] X offset from which to begin reading image. Referenced to upper left corner

offy [int] Y offset from which to begin reading image. Referenced to upper left corner.

lnx [int] How many columns to read beginning from x offset

lny [int] How many rows to read beginning from y offset

Returns

array an array corresponding to a slice of the raster array with dimensions (m,n,p) where m=lnx, n=lny and p=len(ix)

open(*img*)

Open an image and collect some parameters

reshape_chunk(*chunk*)

Flatten a 3-d array so it can be fed into a random forest classifier

Parameters

chunk [array-like] 3-d array with dimensions (m, n, p)

Returns

array-like 2-d array with dimensions (m*n, p). Each row corresponds to a pixel and each column corresponds to an image band

class forest.**metric**(*labels*, *predictions*)

A class to hold various statistics about a binary classification

Parameters

labels [array-like (1-d)] vector of feature labels

predictions [array-like (1-d)] vector of predicted categories equal in length to labels

Example

labels = np.array([True, True, True, True, True, False, False, False]) predictions = np.array([True, False, False, True, True, True, False, False]) M = metric(labels, predictions) print(M)

add_dict(*dct*, *name*)

Add custom statistics

Dictionaries are added to the 'extras' attribute are written to the output file when save_report() is called.

Parameters

dct [dict] Dictionary of statistics to add

name [str] Header for set of statistics when it is written to a file

calculate_metrics(*labels*, *predictions*)

Calculates accuracy, precision, F1, recall and specificity

confusion_matrix(*labels*, *predictions*)

Build confusion matrix for labels and predictions

save_report(*txtfile*)

Saves all calculated statistics to a textfile.

Includes confusion matrix, derived statistics (F1, Accuracy etc..) and any custom statistics that were added.

Parameters

txtfile [str] path to output file

class forest.**training_dataset**

A class to hold data for random forest training and evaluation.

Makes use of hdf5 files to store training data.

Attributes

training_data [array-like] features for training samples

testing_data [array-like] features for testing samples

training_targets [array-like] 1-d vector of feature labels for training samples

testing_targets [array-like] 1-d vector of feature labels for testing samples

classmethod from_h5(*h5f*, *nland*, *nwater*, *eval_frac*, *max_L2W_ratio=None*)

Create training dataset object from h5 file

classmethod from_many_h5(*h5list*, *nland*, *nwater*, *eval_frac*, *max_L2W_ratio=None*)

Create training dataset object from multiple h5 files

proportional_h5_sample(*h5f*, *nwater=1000*, *eval_frac=0.2*, *max_L2W_ratio=15*)

Sample water/land pixels proportional to their abundance in the image

Parameters

h5f [str] Path to hdf5 file created using PixStats or a list of h5 files

nwater [int] Number of water pixels to sample

eval_frac [float] Fraction between 0 and 1 that should be set aside for evaluation

max_L2W_ratio [int] Maximum ratio of land to water pixels. Used to prevent very large sample sizes in scenes where water is very sparse.

`sample_from_h5`(*h5f*, *nland=1000*, *nwater=1000*, *eval_frac=0.2*)

Read data from an HDF5 file.

This is done to more easily sample values with known labels without having to read the entire dataset into memory. Assigns each data point to either water or land depending on what is

Parameters

h5f [str] path to hdf5 file created using PixStats or a list of h5 files

nland [int] number of water pixels to sample

nwater [int] number of water pixels to sample

eval_frac [float] fraction between 0 and 1 that should be set aside for evaluation

Returns

None Stores data in training_dataset object

`split_sample`(*sample*, *eval_frac=0.2*)

Randomly split sample into training and test subsamples

Returned samples are shuffled relative to the input sample

Parameters

sample [array-like] an array of samples with dimension (m, n)

eval_frac [numeric] fraction of rows to allocate to testing data

Returns

tuple Two arrays with dimension (m - j, n) and (j, n) where j~=eval_frac*m

class `forest.`**`waterclass_RF`**(***rfargs*)

RandomForest classifier for open-water classification of (radar) images

Parameters

****rfargs** [] keyword arguments passed to sklearn.ensemble.RandomForestClassifier

`evaluate`()

Evaluate the current random forest model using current test data

`predict_chunked`(*imfile*, *outfile*, *chunksize=5000*)

Classify an image piece-by-piece to avoid running out of RAM

Parameters

imfile [str] Path to input image file

outfile [str] Path to output raster containing probability of water

chunksize [int] How many rows to process at once during classification

`predict_features`(*imfile*, *outfile*)

Use current RF model to produce binary classification of an image

predict_probabilities(*imfile*, *outfile*)
Use current RF model to produce probabilistic classification of an image

save_evaluation(*file*)
Save current evaluation statistics to a text file

test_from_h5(*h5f*, *nwater=200*, *output=None*)
Test current model using a random sample from an hdf5 file

Parameters

h5f [str] Path to hdf5 file containing training data

nwater [int] The number of water pixels to draw from

output [str, optional] File path to output classification statistics. Ignored if None

Returns

metric metric() object with classification statistics

train_from_h5(*h5f*, *nland=1000*, *nwater=1000*, *eval_frac=0.2*)
Train a random forest using a data sample from an hdf5 file Optionally set aside some of the sample for evaluation

Parameters

h5f [str] path to hdf5 file created using PixStats or a list of h5 files

nland [int] number of water pixels to sample

nwater [int] number of water pixels to sample

eval_frac [float] fraction between 0 and 1 that should be set aside for evaluation

Postprocessing for probability images generated using random forest classification

postprocess.**grow_regions**(*input*, *output*, *window=3*, *val=50*)
Threshold water classification and grow lakes by 1 pixel

postprocess.**max_filter_inplace**(*img_path*, *band=1*, *size=3*)
Run a maximum filter on a raster file and changes the values in-place

postprocess.**modefilter**(*input*, *output*, *window=7*)
Threshold water classification at 50% water likelihood

postprocess.**polygonize**(*output*, *rast*, *pythonexe='python'*, *gdalpolypath='/usr/bin/gdal'*, *fmt='GPKG'*, *shell=False*)
Convert raster to polygons

The input raster should be equal to 1 wherever a polygon is desired and zero elsewhere. The raster nodata value should also be set to zero for maximum performance

Parameters

output [str] path to output polygon file with file extension

rast [str] path to raster file that will be polygonized

pythonexe [str] path to python executable

gdalpolypath [str] path to gdal_polygonize.py file

fmt [str] GDAL-compatible format for output polygons

shell [boolean] Passed to subprocess. Experimental.

Returns

int return code for the subprocess.call function

`postprocess.`**`postprocess`**(*classified_img*, *output_poly*, *pythonexe*, *gdalpolypath*, *window=7*)

Postprocess a classified probability image to remove false positives

using a techinque inspired by Bolanos et al. (2013)

Parameters

classified_img [str] path to classified probability image

output_poly [str] path to output GPKG file

pythonexe [str] path to python executable

gdalpolypath [str] path to gdal_polygonize.py file

`postprocess.`**`postprocess_highestimate`**(*classified_img*, *output_poly*, *pythonexe*, *gdalpolypath*)

Polygonize regions without filtering

`postprocess.`**`rasterize_inplace`**(*rast*, *inshape*, *prefill=0*)

Overwrites a raster with the output of a polygon rasterization

Parameters

rast [str] path to EXISTING raster file that will store values from rasterized

inshape [str] path to vector dataset that will be rasterized

prefill [int] Value to write to raster before writing polygonization result

`postprocess.`**`raststats`**(*inshape*, *raster*)

calculate mean and max value of a raster in each polygon

`postprocess.`**`set_nodata`**(*file*, *nodata=0*)

Set nodata value for raster file

Parameters

file [str] path to EXISTING raster file

nodata [numeric] value to set as nodata for input raster

`postprocess.`**`threshold`**(*input*, *val=50*)

Threshold a raster image and return the new array

3.4 Training and Testing

`TrainingTestingutils.`**`bin_ndarray`**(*ndarray*, *new_shape*, *operation='sum'*)

Bins an ndarray in all axes based on the target shape, by summing or averaging.

Keeps the dtype of the input array Number of output dimensions must match number of input dimensions.

Example

```
>>> m = np.arange(0,100,1).reshape((10,10))
>>> n = bin_ndarray(m, new_shape=(5,5), operation='sum')
>>> print(n)
[[ 22  30  38  46  54]
 [102 110 118 126 134]
 [182 190 198 206 214]
 [262 270 278 286 294]
 [342 350 358 366 374]]
```

`TrainingTestingutils.`**`consume`**(*iterator*, *n=None*)

"Advance the iterator n-steps ahead. If n is none, consume entirely. From python.org manual 9.7.2

`TrainingTestingutils.`**`rebin`**(*a*, *new_shape*)

Resizes a 2d array by averaging or repeating elements, new dimensions must be integral factors of original dimensions *Parameters*

a [array_like] Input array.

new_shape [tuple of int] Shape of the output array

Returns

rebinned_array [ndarray] If the new shape is smaller of the input array, the data are averaged, if the new shape is bigger array elements are repeated

See Also

resize : Return a new array with the specified shape.

Examples

```
>>> a = np.array([[0, 1], [2, 3]])
>>> b = rebin(a, (4, 6)) #upsize
>>> b
array([[0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [2, 2, 2, 3, 3, 3],
       [2, 2, 2, 3, 3, 3]])
>>> c = rebin(b, (2, 3)) #downsize
>>> c
array([[ 0. ,  0.5,  1. ],
       [ 2. ,  2.5,  3. ]])
```

class `GSWInterpolator.`**`GSWInterpolator`**(*sat_f_name*, *gsw_dir*, *output_dir=None*, *data=None*, *use_cols_vector=None*)

Handle Global Surface Water files

Parameters

sat_f_name [str] File path to satellite imagery scene for which water mask is to be interpolated

gsw_dir [str] File path to location of global surface water *.tif files

output_dir [str] File path to desired location of output files

data : use_cols_vector :

`array_to_raster`(*array*)

Array > Raster Save a raster from a C order array (column-major).

Parameters

array : ndarray

Returns

a tuple of the gdal file handle for the raster dataset and a gdal RasterBand object corresponding to the input array

`get_covering_global_surface_water_file_names`(*min_lat*, *max_lat*, *min_lon*, *max_lon*)

Assuming we're running in the path where there is a 'coverage' directory containing all the RS2 BBOX and convex hulls.

`get_water_presence_for_points`(*min_lat*, *max_lat*, *min_lon*, *max_lon*, *pts*)

interpolate_water_presence(*min_lat*, *max_lat*, *min_lon*, *max_lon*, *save=True*, *get_data=True*)

class PixStats.**PixStats**(*f_path*, *output_dir=None*, *gsw_path=None*, *images_output_dir=None*, *fst_converter_path=None*)

Get satellite images ready for neural net processing

Parameters

f_path [str] imagery file to be converted

output_dir : str

gsw_path [str] Path to directory containing global surface water data

images_output_dir : str

fst_converter_path [str] File path with location to save the files in FST format

classmethod get_file_pol(*file*)

get valid polarizations for a file

get_predict_data(*index=0*, *num_procs=8*, *polarization='HH'*, *water_weight=1.0*)

Put image data into a dictionary that can be fed as features into an tf.estimator.inputs.numpy_input_fn object

Returns

A dictionary whose keys correspond to the names of image bands

get_stats(*write_water_mask=True*, *write_hist=False*)

Create hdf5 file and a priori water mask for radar image

Creates a *.tif file corresponding to the 89-100% confidence interval for water in the GSW product. Also takes any pixels with water likelihood equal to zero or greater than 89 and writes them to an hdf5 file (these become the data on which the model will be trained)

Creates histograms of water/non-water pixel values

get_valid_bands()

build dictionary to describe order of bands

prepare_from_geotif(*classified_img*, *convert_probabilities=False*, ***kwargs*)

Equivalent to prepare_fst_info but used when pol_fst_array doesn't exist

Parameters

classified_img [str] path to gdal-supported raster

class SRIDConverter.**SRIDConverter**

convert coordinates from one coordinate reference system to another using a spatial reference identifier (SRID)

classmethod convert_from_coordinates_check_geo(*coordinates*, *src_srs*, *dest_srid=4326*)

Returns

A tuple (x, y) consisting of x and y coordinates

PYTHON MODULE INDEX

INDEX

H

I

L

M

N

O

P

R

S

T

U

W