SAR Surface Water Mapping

Release 1.1

National Hydrological Service

Jul 31, 2025

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 over 2200 gauging stations strategically placed across the country to monitor Canadian water resources. While the data these stations collect is important to the regions they are located in, the 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 dynamic 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 (RADARSAT-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). In addition to RCM, support for ESA's Sentinel-1 SAR missions can be used in conjunction, increasing the flexibility and availability of SAR data products used for surface water mapping.

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 folowing 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

COMMON SETUP AND OPERATION

2.1 Python

The following libraries in at-least Python 3.10 are needed to run the program:

- NumPy
- SciPy
- Pandas
- Geopandas
- GDAL
- RasterStats
- Rasterio
- SciKit-Learn
- SciKit-Image
- Requests

2.2 Environment & config

Included with the code is a config file where the user provides the necessary settings required by SSWM in order to run. The config is split into two sections, each with their own focus as follows:

The Directories section is for input of the various directories needed by the program. Each of these should be created as a new folder within the working directory of SSWM.

The Params section is for additional information required by the program to run. Mainly, the user is able to select which type of DEM to use during preprocessing; Shuttle Radar Topography Mission (SRTM) or Canadian Digital Elevation Data (CDED), and which SAR mission is being processed; Radarsat-2 (RS2), Radarsat Constellation Mission (RCM) or Sentinel-1 (S1).

All individual config fields Table 1.

Table 1: Config fields

Field	Section	Purpose	Data type
DEM_dir	Directories	A location where DEM files are stored	Full Path
watch_folder	Directories	A location where SAR data to be processed is placed	Full Path
gsw_path	Directories	A location where GSW tiles for training are stored	Full Path
output	Directories	Where classified results (rast & vect) are placed	Full Path
tmp	Directories	Location for temporary files	Full Path
log_dir	Directories	Location for log files	Full Path
DEMType	Params	DEM product to use in preprocessing	SRTM or CDED
satellite_profile	Params	SAR sensor being processed	RS2, RCM, or S1
num_procs	Params	Number of processors to use during classification	int

2.3 SAR Data

The current version of SSWM (1.1) supports SAR data from the following missions: Radarsat-2, Radarsat Constellation Mission (RCM), and Sentinel-1. Functionality with Radarsat-2 has not been tested in this version however, thus, using this data is not guaranteed to produce accurate results. Additionally, the RCM preprocessor expects as input a multiband raster product (IE GeoTiff) that is pre-calibrated, thus the user is responsible for conducting this initial preprocessing. If this is unobtainable, it is recommended to use Sentinel-1 data, as the Sentinel-1 preprocessor performs all necessary preprocessing steps (including calibration) using OrfeoToolbox. There is currently no intention to add support for additional SAR missions, however, a new function in the preprocessor for any given SAR mission could be added by the user if desired. Simply utilize the Sentinel-1 preprocessor as a template.

2.4 OrfeoToolbox

SSWM utilizes OrfeoToolbox (OTB, https://www.orfeo-toolbox.org/) to perform some SAR preprocessing, mainly calibration and orthorectification of Sentinel-1 products. To install, download the latest build for your operating system from here: https://orfeo-toolbox.org/packages/, and unzip the contents to a permanent location. No installer needs to be run. While OrfeoToolbox does have a Python API, is it difficult to set up due to overlapping dependancies with pre-build Python environments. Instead, SSWM calls the necessary OTB functions via the command line. To do this, the full path to the OTB bin folder must be added as a system *Path* variable. For example, using OTB version 9.0.0 that was extracted to the C: drive on Windows, the following would need to be added to the system Path variable: **C:OTB-9.1.0-Win64bin**.

2.5 Computational Power

Many of the operations performed by SSWM are computationally intensive, something that is exacerbated when processing Sentinel-1 data given its resolution and scene size. Several setting within the Python scripts can be modified to improve performance and computation success:

1) For preprocessing completed using OrferToolbox, the amout of RAM available to the toolbox can be modified. OTB will automatically tile the data product for processing based on the amount of RAM that is available, thus, increasing or decreasing OTB's RAM allotment will dictate the speed of processing, but also makes it possible for less powerful systems to perform operations. Within the preprocessing.py script, a global 'MAX_RAM' parameter has been defined. The current default value is 2500 MB.

2) After training, SAR scenes are classified in chunks to prevent memory overflow. This value is currently set to chunks of 1000 rows, and can be modified via the global parameter in launch_forest.py. If execution of SSWM fails at this stage due to memory overflow, reduce this value.

3) The number of processor used during classification, which is set by the user in the config file, also dictates the speed

at-which scenes are processed. This should be modified in conjunction with the chunksize to find the ideal parameter set for processing on your specific machine.

2.6 GSW Tiles

The Global Surface Water dataset (GSW, doi:10.1038/nature20584) is used to train the random forest classifier. The dataset is global, and can be downloaded as a series of 10x10 degree tiles from here: https://global-surface-water.appspot. com/download. The user must download the occurrence product for each tile within their ROI, and place them into the waterTiles directory defined earlier.

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

2.7 DEM Products

DEM files are used by the preprocessor for orthorectification of the SAR scenes. The DEM tiles needed for each SAR scene are automatically downloaded if they are not already available within the DEM folder. Two DEM options are available; Shuttle Radar Topography Mission (SRTM) and Canadian Digital Elevation Data (CDED). Downloading CDED tiles does not require an account, however, they are only available within Canada, thus, if any portion of the SAR scene exists outside of Canada, strange behavior will occur. SRTM tiles are available globally to 60 degrees North, but require an EarthData account to access (https://urs.earthdata.nasa.gov/). If any portion of the SAR data extends above 60N, strange behavior will occur. For Python to access EarthData products, a **.netrc** file is needed in the users Home directory. To make this, create a text file in your Home directory with the name .netrc, and add the following line to it:

machine urs.earthdata.nasa.gov login <username> password <password>

where <username> and <password> are your EarthData credentials.

2.8 Launching SSWM

Before running SSWM, ensure that the desired SAR scenes are placed into the water_folder (keep the zipped). Next, open the check_directory.py script, and add the full path to the config file (classify.ini) to the config variable (line 17). Then, run check_directory.py using the Python environment created earlier.

CHAPTER

THREE

FUNCTIONS

3.1 Pre-Process

SSWM.preprocess.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 data folder

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

SSWM.preprocess.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

`SSWM.preprocess.preprocess.preproS1`(*folder*, *DEM_dir*, *cleanup=True*, *product='CDED'*)

Preprocess Sentinel-1 file in preparation for classification

Parameters

folder
[str] Path to data folder

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

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

`SSWM.preprocess.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)

`SSWM.preprocess.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 SSWM.preprocess.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.

SSWM.preprocess.DEM.**create_DEM_mosaic**(*DEM*, *DEM_dir*, *dstfile*, *product='CDED'*, *vrt_only=False*, *format='GTiff'*)

Create a Mosaic from a list of DEM urls or NTS tiles. Missing tiles will be downloaded

SSWM.preprocess.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 ("SRTM", "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)

SSWM.preprocess.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 }

SSWM.preprocess.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>

`SSWM.preprocess.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

`SSWM.preprocess.DEM.`**`download_multiple_DEM`**(*DEM*, *DEM_dir*, *product='CDED'*)

Download a list of DEM URLs. If they exist already, they are not downloaded

Parameters

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

DEM_dir
[str] Path to which files are downloaded

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

Returns

list
a list of file paths for target DEMs

`SSWM.preprocess.DEM.`**`download_single_DEM`**(*DEM_id*, *DEM_dir*, *replace=False*, *product='CDED'*)

Download a DEM tile

Parameters

DEM_id
[str] Name or NTS sheet of tile to download. If product is "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: ('SRTM', 'CDED')

Returns

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

`SSWM.preprocess.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.

`SSWM.preprocess.DEM.`**`get_spatial_extent`**`(`*raster_path*, *target_EPSG=4326*, *tol=0.5*`)`

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

`SSWM.preprocess.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

3.1.1 Example

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

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

`SSWM.preprocess.filters.`**`enhanced_lee_filter`**`(`*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

`SSWM.preprocess.filters.`**`filter_image`**`(`*img*, *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

`SSWM.preprocess.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

`SSWM.preprocess.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

`SSWM.preprocess.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)

`SSWM.preprocess.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

`SSWM.preprocess.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

`SSWM.preprocess.orthorectify.`**`orthorectify_otb`**(*input*, *output*, *DEMFolder*, *gridspacingx*, *ram=1000*)

Orthorectify raster using orfeotoolbox Ortho

3.1.2 Parameters

input
[str] Path to image to orthorectify

output
[str] Path to output image

DEM
[str] Path to DEM

gridspacing
[float] pixel size of deformation grid used for the ortho

3.1.3 Returns

`SSWM.preprocess.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 `SSWM.preprocess.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 `SSWM.preprocess.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 `SSWM.preprocess.preutils.`**`Radar`**

Generic class for RS2 and RCM folder structures

classmethod TIF_channels(*tif*)

Get count how many channels are in an image

`SSWM.preprocess.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 SSWM.preprocess.preutils.`**`S1`**

Class for accessing information about an RCM dataset

`SSWM.preprocess.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

`SSWM.preprocess.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)

`SSWM.preprocess.preutils.`**`calibrateS1`**(*img*, *lut='sigma'*)

Using Orfeotoolbox, calibrate an S1 product. Calibration is performed per data band, and are then stacked.

`SSWM.preprocess.preutils.`**`calibrate_in_place`**(*file*, *lut*, *complex*, *scale*, *band=[1]*)

Apply LUT calibration to file and change in-place

SSWM.preprocess.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)

SSWM.preprocess.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

SSWM.preprocess.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

SSWM.preprocess.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

SSWM.preprocess.preutils.**createvalidpixrast**(*img*, *dst*, *band*)

Create valid pixel raster (0 or 1) for a gdal raster band

SSWM.preprocess.preutils.**get_blocksize_options**(*img*)

Get raster blocksize information as a string that can be passed to gdal

`SSWM.preprocess.preutils.`**`incidence_angle_from_gains`**`(`*beta_gains*, *sigma_gains*, *complex=False*`)`

calculate incidence angle array

`SSWM.preprocess.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.

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

interpolate array with desired step size

`SSWM.preprocess.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)

`SSWM.preprocess.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)

`SSWM.preprocess.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

`SSWM.preprocess.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

3.2 Random-Forest

class `SSWM.trainingTesting.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 :

`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.

class `SSWM.trainingTesting.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_stats_and_sample`(*valseed*, *nwater*, *nland*, *max_L2W_ratio*, *write_water_mask=False*)

Sample an image, we no longer use H5 files

`get_valid_bands`()

build dictionary to describe order of bands

For RandomForest water classification of radar images

class `SSWM.forest.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 SSWM.forest.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 SSWM.forest.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

sample_from_image(*cur_file*, *exdir*, *gsw_path*, *valseed*, *nland=1000*, *nwater=1000*, *eval_frac=0.2*, *max_L2W_ratio=10*)

sample n & m pixels water and land pixels respectivally from the scene, split into training and testing sets needed for RF

Parameters

cur_file
[str] path to image file

exdir: str
path to dataset directory

gsw_path
[str] path to gsw files needed to create mask for training

valseed
[int] seed for random sampling

nland
[int] number of land 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

max_l2w_ratio
[int] maximum allowed ratio of land pixels to water pixels in the training dataset

`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 SSWM.forest.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

train_from_image(*cur_file*, *exdir*, *gsw_path*, *valseed*, *nland=1000*, *nwater=1000*, *eval_frac=0.2*)

Train a random forest by sampling directly from an image Optionally set aside some of the sample for evaluation

Parameters

cur_file
[str] path to image file

exdir: str
path to dataset directory

gsw_path
[str] path to gsw files needed to create mask for training

valseed
[int] seed for random sampling

nland
[int] number of land 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

SSWM.forest.postprocess.**grow_regions**(*input*, *output*, *window=3*, *val=50*)

Threshold water classification and grow lakes by 1 pixel

SSWM.forest.postprocess.**max_filter_inplace**(*img_path*, *band=1*, *size=3*)

Run a maximum filter on a raster file and changes the values in-place

SSWM.forest.postprocess.**modefilter**(*input*, *output*, *window=7*)

Threshold water classification at 50% water likelihood

SSWM.forest.postprocess.**postprocess**(*classified_img*, *output_poly*, *extrasTXT*, *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

extrasTXT
[str] path containing RF model quality metrics

window
[int] window size to use for filtering (Default 7)

`SSWM.forest.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

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

calculate mean and max value of a raster in each polygon

`SSWM.forest.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

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

Threshold a raster image and return the new array

PYTHON MODULE INDEX

S

INDEX

I

L

M

N

O

P

R

S

T

W