Assignment 4

Solution

1. 1. Build mask

Create a function that reads a single QA file and returns a Boolean numpy array (True: good observations, False: invalid observations). Good observations are clear land, water, and snow pixels (See the codes in the exercise description).

In [27]:
```
import numpy as np
import os
import fnmatch
from osgeo import gdal
from matplotlib import pyplot as plt
from matplotlib import patches as mpatches
from matplotlib import colors
```

In [28]:
```
def readMaskAsArray(fname):

    '''
    0 => clear land pixel
    1 => clear water pixel
    2 => cloud shadow
    3 => snow
    4 => cloud
    255 => no observation
    '''

    # open geotiff file
    ds = gdal.Open(fname)

    # open first band
    band = ds.GetRasterBand(1)

    # retrieve data from first band
    array = band.ReadAsArray()

    # use logical expression to create a binary mask of
    # True: clear pixels, and False: cloud and invalid pixels
    mask = np.logical_or(array < 2, array == 3)
    ds = None

    return mask
```

2. Mask image

Create a function that takes the filename of a single SR image and outputs a numpy array containing the masked-out image data. Inside the function, you should identify the corresponding QA file. Use the function from task 1 to retrieve the correct mask array.

In [36]:
```
def maskImage(sr_file):

    # create file name for QA file based on what we know about the
    # file naming convention
    qa_file = sr_file.replace('sr', 'qa')

    # open reflectance file
    src_ds = gdal.Open(sr_file)

    # retrieve 6-band reflectance array
    # shape: (6, 1000, 1000)
    sr_array = src_ds.ReadAsArray()

    # retrieve boolean cloud mask
    # shape: (1000, 1000)
    qa_array = readMaskAsArray(qa_file)

    # here are two ways to reshape the mask to match the three dimensions
    # of the reflectance data. However, because we want to broadcast the
    # left most dimension of the reflectance array, the reshaping is done
    # automatically.
    # qa_array1 = qa_array.reshape(1, qa_array.shape[0], qa_array.shape[1])
    # qa_array2 = qa_array[np.newaxis(), :, :)

    # V1: apply mask to reflectance array by multiplication
    # masked values become 0.
    masked_array = sr_array * qa_array

    return masked_array
```

3. Mean image

Create a function that takes SR file names as input and outputs a mean SR image, i.e., a six-band image where each band and pixel represent the mean across time. Utilize the functions you created in task (1) and (2). Hint: A NaN is a special value for float arrays only.

In [46]:
```
def meanImage(sr_files, out_file=None):

    # collect all images in a list
    result = []
    for sr_file in sr_files:
        # print(os.path.basename(sr_file))
        result.append(maskImage(sr_file))

    # stack all images into a 4-dimensional array
    # stack.shape: (6, 1000, 1000, 17)
    stack = np.stack(result, axis=3)

    # create mask of invalid pixel values
    invalidValues = stack == 0

    # replace invalid values with NaN's
    # This requires converting the integer array
    # to a float32 array
    stack = stack.astype(np.float32)
    stack[invalidValues] = np.NaN

    # Calculate the mean across the 4th dimension (time: axis=3)
    # mean.shape: (6, 1000, 1000)
    mean = np.nanmean(stack, axis=3, dtype=np.float32)

    if out_file is not None:

        src_ds = gdal.Open(sr_files[0])

        n_layers = mean.shape[0]

        drv = gdal.GetDriverByName('GTiff')
        dst_ds = drv.Create(out_file,
                            src_ds.RasterXSize, src_ds.RasterYSize, n_layer
s,
                            gdal.GDT_Float32, [])

        for i in range(n_layers):
            dst_band = dst_ds.GetRasterBand(i + 1)

            # write the data
            dst_band.WriteArray(mean[i, :, :], 0, 0)

            # flush data to disk, set the NoData value and calculate stats
            dst_band.FlushCache()

        dst_ds.SetGeoTransform(src_ds.GetGeoTransform())
        dst_ds.SetProjection(src_ds.GetProjectionRef())

        dst_ds = None
        src_ds = None
```

Split vertex array into time-axis (year) and spectral-axis (fit)

In [61]:
```
sr_files = []
path = os.getcwd()
for dirpath, dirnames, files in os.walk(path):
    for fname in files:
        if fname.endswith('sr_clip.tif'):
            sr_files.append(os.path.join(dirpath, fname))

meanImage(sr_files, out_file='meanImage.tif')
```

```
/Users/dirk/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:24: Ru
ntimeWarning: Mean of empty slice
```

4. Max NDVI Composite

Create a function that takes SR file names as input and write a maximum NDVI composite as GeoTIFF. In a maximum NDVI composite each pixel contains the band values that correspond with the date of maximum NDVI. Hence, each pixel will represent the original band values that were recorded at the date of maximum vegetation greenness.

In [59]:
```
def maxNdviComposite(sr_files, out_file=None):

    # collect all images in a list
    result = []
    for sr_file in sr_files:
        # print(os.path.basename(sr_file))
        result.append(maskImage(sr_file))

    # stack all images into a 4-dimensional array
    # stack.shape: (6, 1000, 1000, 17)
    stack = np.stack(result, axis=3)

    # create mask of invalid pixel values
    invalidValues = stack == 0

    # replace invalid values with NaN's
    # This requires converting the integer array
    # to a float32 array
    stack = stack.astype(np.float32)
    stack[invalidValues] = np.NaN

    # Calcualte NDVI arcross all time steps at once
    # ndvi.shape: (1000, 1000, 17)
    red = stack[2, :, :, :]
    nir = stack[3, :, :, :]
    ndvi = (nir - red) / (nir + red)

    # The previous division may result in non-finite values
    # for some pixels (e.g. division by 0). Let's replace invalid values
    # with a very low NDVI:
    ndvi[np.isnan(ndvi)] = -1

    # get array indices along the time axis
    # where ndvi is maximum, i.e. each pixel
    # value in the resulting index array
    # represents the position along the time axis.
    # inds.shape: (1000, 1000)
    inds = np.argmax(ndvi, axis=2)

    # reshape inds to match the 4 dimensions of the reflectance
    # image (variable: stack)
    # inds.shape: (1, 1000, 1000, 1)
    inds = inds[np.newaxis, :, :, np.newaxis]

    # maxNdvi.shape: (6, 1000, 1000, 1)
    maxNdvi = np.take_along_axis(stack, inds, axis=3)

    # remove dimensions of length=1
    # maxNdvi.shape: (6, 1000, 1000)
    maxNdvi = maxNdvi.squeeze()

    if out_file is not None:

        src_ds = gdal.Open(sr_files[0])

        n_layers = maxNdvi.shape[0]

        drv = gdal.GetDriverByName('GTiff')
        dst_ds = drv.Create(out_file,
                            src_ds.RasterXSize, src_ds.RasterYSize, n_layer
s,
                            gdal.GDT_Float32, [])

        for i in range(n_layers):
            dst_band = dst_ds.GetRasterBand(i + 1)

            # write the data
            dst_band.WriteArray(maxNdvi[i, :, :], 0, 0)
```

In [60]:
```
sr_files = []
path = os.getcwd()
for dirpath, dirnames, files in os.walk(path):
    for fname in files:
        if fname.endswith('sr_clip.tif'):
            sr_files.append(os.path.join(dirpath, fname))

maxNdviComposite(sr_files, out_file='maxNdviComposite.tif')
```