

Lecture 08

GEE Change Detection

Part 1: two-date image differencing

2024-04-22

Sébastien Valade



UNIVERSIDAD NACIONAL
AUTÓNOMA DE
MÉXICO

1. Introduction

2. Case example: detection of a wild fire

1. Select pre/post-event images
2. Preprocess images
3. Compute change map
4. Analyze change map

1. Introduction

2. Case example: detection of a wild fire

1. Select pre/post-event images
2. Preprocess images
3. Compute change map
4. Analyze change map

Change detection:

⇒ **Change detection** in remote sensing consists in *capturing differences in images acquired at different times in order to assess how landscape conditions have changed*. Examples:

- changes in land cover → e.g., deforestation, urban sprawl, desertification, polar ice loss, etc.
- changes after natural disasters → e.g., floods, fires, eruptions, etc.

⇒ Questions which can be addressed:

- has a change occurred?
- what area is affected?
- what is the nature/severity of the change?

⇒ Challenges which arise: separate the “changes of interest” from the “other changes”

- changes related to *seasonal conditions*
- changes related to *image acquisition conditions*:
 - scene illumination (e.g., sun angle, sensor position)
 - atmospheric effects (e.g., clouds)
 - sensor health and processing algorithm (e.g., leading to radiometric inconsistencies)

Change detection:

⇒ **Change detection** in remote sensing consists in *capturing differences in images acquired at different times in order to assess how landscape conditions have changed*. Examples:

- changes in land cover → e.g., deforestation, urban sprawl, desertification, polar ice loss, etc.
- changes after natural disasters → e.g., floods, fires, eruptions, etc.

⇒ Questions which can be addressed:

- has a change occurred?
- what area is affected?
- what is the nature/severity of the change?

⇒ Challenges which arise: separate the “changes of interest” from the “other changes”

- changes related to *seasonal conditions*
- changes related to *image acquisition conditions*:
 - scene illumination (e.g., sun angle, sensor position)
 - atmospheric effects (e.g., clouds)
 - sensor health and processing algorithm (e.g., leading to radiometric inconsistencies)

Change detection:

⇒ **Change detection** in remote sensing consists in *capturing differences in images acquired at different times in order to assess how landscape conditions have changed*. Examples:

- changes in land cover → e.g., deforestation, urban sprawl, desertification, polar ice loss, etc.
- changes after natural disasters → e.g., floods, fires, eruptions, etc.

⇒ Questions which can be addressed:

- has a change occurred?
- what area is affected?
- what is the nature/severity of the change?

⇒ Challenges which arise: separate the “changes of interest” from the “other changes”

- changes related to *seasonal conditions*
- changes related to *image acquisition conditions*:
 - scene illumination (e.g., sun angle, sensor position)
 - atmospheric effects (e.g., clouds)
 - sensor health and processing algorithm (e.g., leading to radiometric inconsistencies)

Naive method: two-date image differencing

- ⇒ Easiest way to detect changes is to perform *image differencing* between two images (pre- and post-event), by simply subtracting the spectral values (or spectral indices) of the pre-image from that of the post-image, pixel by pixel.
- ⇒ The exercise here consists in detecting the changes related the wild fires which affected the region of Palermo in July 2023. The workflow will be as followed:
 1. Select images
 2. Preprocess images
 3. Compute change map
 4. Analyze change map

1. Introduction

2. Case example: detection of a wild fire

1. Select pre/post-event images
2. Preprocess images
3. Compute change map
4. Analyze change map

2.1. Select pre/post-event images

Step 1: Select pre/post-event images

1. Select two images (before/after the event), trying to minimize the impact of:

- atmospheric conditions: select images with the *least cloud cover* possible
- seasonal conditions: select images acquired during the same season

```
# Select image collection and bands
image_collection = (ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
    .select(
        ['B2', 'B3', 'B4', 'B8', 'B11', 'B12'], # selected bands
        ['blue', 'green', 'red', 'nir', 'swir1', 'swir2']) # renamed bands (for convenience)
    )

# Select pre-event image
point = ee.Geometry.Point([13.33, 38.13]) # select region of interest
ti, tf = '2019-01-01', '2022-01-01' # select time interval
ti_month, tf_month = 8, 10 # select month interval (season)

image_pre = (image_collection
    .filterBounds(point)
    .filterDate(ti, tf)
    .filter(ee.Filter.calendarRange(ti_month, tf_month, 'month'))
    .sort('CLOUD_COVER') # sort collection by cloud cover
    .first()) # select least clouded image
image_pre_date = ee.Date(image_pre.get('system:time_start')).format('YYYY-MM-dd'). getInfo()
```

2.1. Select pre/post-event images

Step 1: Select pre/post-event images

1. Select two images (before/after the event), trying to minimize the impact of:
 - atmospheric conditions: select images with the *least cloud cover* possible
 - seasonal conditions: select images acquired during the same season
2. Clip region of interest (optional)

```
# Clip region of interest
lon_min, lon_max = -98.67, -98.574
lat_min, lat_max = 18.978, 19.068
roi = ee.Geometry.Rectangle([lon_min, lat_min, lon_max, lat_max])

image_pre = image_pre.clip(roi)
image_post = image_post.clip(roi)
```

2.1. Select pre/post-event images

Step 1: Select pre/post-event images

1. Select two images (before/after the event), trying to minimize the impact of:
 - atmospheric conditions: select images with the *least cloud cover* possible
 - seasonal conditions: select images acquired during the same season
2. Clip region of interest (optional)
3. Display result

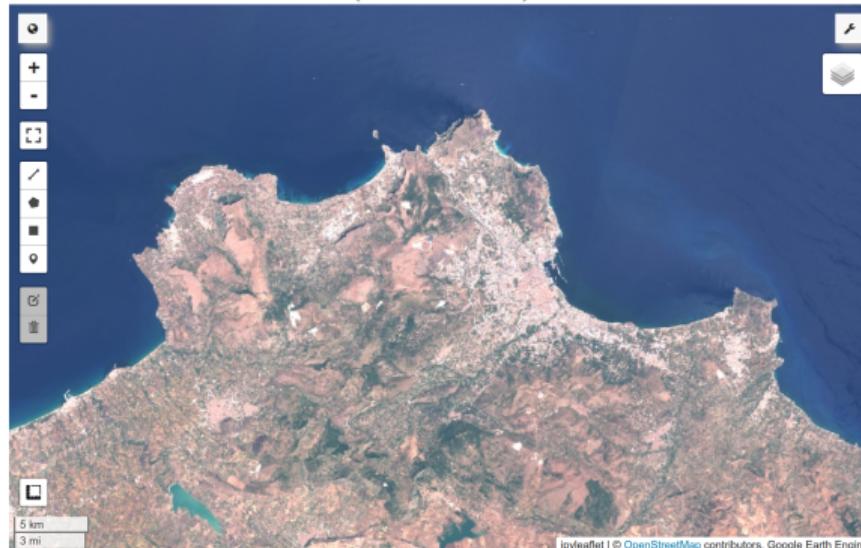
```
Map = geemap.Map()
Map.centerObject(point, 11)
Map.addLayerControl()

vis_params = {'bands': ['red', 'green', 'blue'], 'min': 0, 'max': 2000}
Map.addLayer(image_pre, vis_params, f'Pre-event ({image_pre_date})')
Map.addLayer(image_post, vis_params, f'Post-event ({image_post_date})')
Map
```

2.1. Select pre/post-event images

Step 1: Select pre/post-event images

Pre-event image
(2019-08-01)



Post-event image
(2023-08-03)



2.2. Preprocess images

Step 2: Preprocess images

⇒ *image preprocessing* should be achieved on images before continuing the *change detection* workflow, in order to ensure that each pixel records the same type of measurement at the same location over time.

This typically includes:

- *image co-registration*
⇒ ensures that images are in the same projection and have the same pixel size (resampling)
- *radiometric and atmospheric corrections*
⇒ ensures that the pixel values are comparable (e.g., convert digital numbers (DN) to reflectance values, calculated either at the *top of the atmosphere* (TOA) or at the *surface*, with or without *atmospheric correction*)
- *illumination correction*
⇒ correct *local solar incidence* (depends on sensor inclination + sun elevation/azimuth + terrain slope/aspect) → see Carty (2019) Chapter 5

NB: notice the *sunglint* in the post-event image, caused by the specular reflection of sunlight off the water surface directly towards the satellite sensor, which results in bright silvery pixels. (The MSI instrument onboard Sentinel-2 has different detectors which acquire the scene with slightly different viewing angles, thereby resulting in different sunglint patterns. Metadata stores information on sensor/sun viewing angles).

- *cloud and shadow masking*
⇒ remove pixels affected by clouds/shadows

2.2. Preprocess images

Step 2: Preprocess images

⇒ luckily, the most important preprocessing steps have been applied to the images available in GEE. EX:

- **Sentinel-2 (MSI)**

- Top-of-Atmosphere Reflectance: ‘COPERNICUS/S2_HARMONIZED’
 - = Level 1-C processing
 - = top-of-atmosphere reflectance (TOA), orthorectified, harmonized¹
- Surface Reflectance: ‘COPERNICUS/S2_SR_HARMONIZED’
 - = Level 2-A processing
 - = surface reflectance, orthorectified, atmospherically corrected, harmonized¹

- **Sentinel-1 (SAR)**

- Ground Range Detected SAR (log-scaling): ‘COPERNICUS/S1_GRD’
 - = Level 1 GRD processing
 - = backscattered intensity with log scaling ($I_{dB} = 10 * \log_{10}(I)$), single-polarization (VV or VH), sampled in ground range, orthorectified (terrain corrected using DEM), calibrated (thermal noise removal + radiometric calibration)
- Ground Range Detected SAR (log-scaling): ‘COPERNICUS/S1_GRD_FLOAT’
 - = same as ‘COPERNICUS/S1_GRD’ but without log scaling

¹The “harmonized” designation means that the band-dependent offset added to reflectance bands (affecting data after 2022/01/24, processing baseline 04.00) has been removed.

2.3. Compute change map

Step 3: Compute change map

- ⇒ the “naive approach” to computing *change maps* is to subtract bands (or subtract band compositions such as spectral indices) between the pre-event and post-event images.
- ⇒ the choice of the bands (or band compositions) to subtract greatly depends on the type of change to detect (i.e. flooding, fire, urban sprawl, etc.)
- ⇒ in this example we want to detect a *wild fire*, so we will subtract the *NBR* (Normalized Burn Ratio) between the pre-event and post-event images, where $NBR = \frac{NIR - SWIR}{NIR + SWIR}$

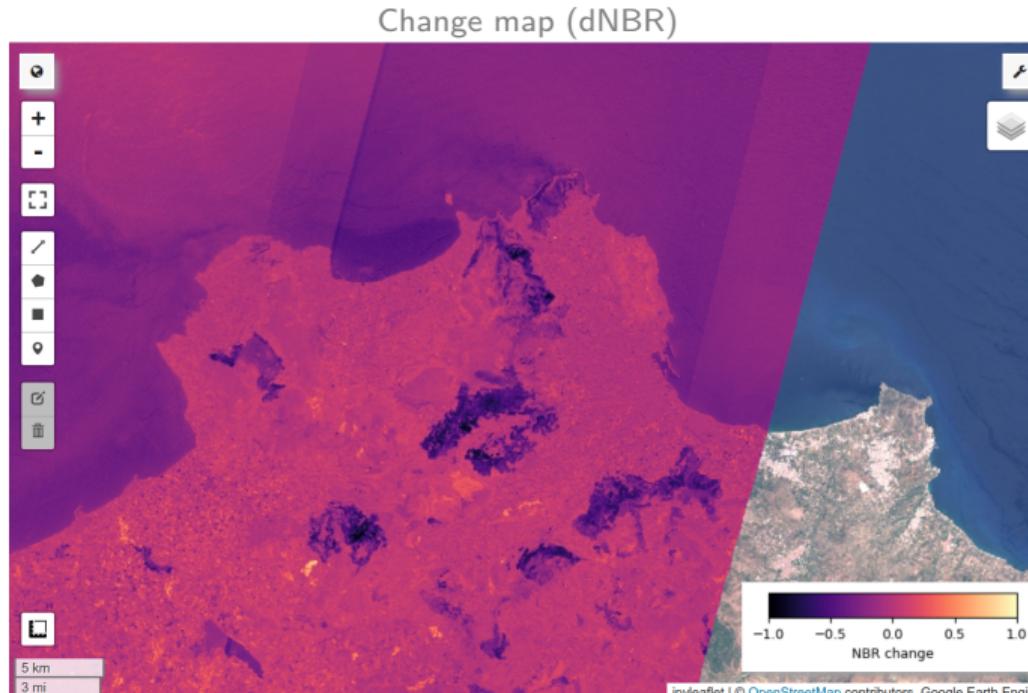
```
# Calculate Normalized Burn Ratio (NBR) for pre-event and post-event images
nbr_pre = image_pre.normalizedDifference(['nir', 'swir2']).rename('nbr_pre')
nbr_post = image_post.normalizedDifference(['nir', 'swir2']).rename('nbr_post')

# Calculate difference between pre-event and post-event NBR
img_change = nbr_post.subtract(nbr_pre).rename('change')

# Display result
vis_params = {'palette': 'magma', 'min': -1, 'max': 1}
Map.addLayer(img_change, vis_params, 'Change map')
Map.add_colorbar(vis_params, label="NBR change", layer_name='Change map')
```

2.3. Compute change map

Step 3: Compute change map



2.4. Analyze change map

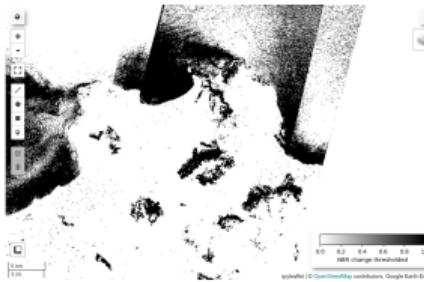
Step 4: Analyze change map

- ⇒ The goal here is to isolate the regions of the change map that correspond to the burned area
- ⇒ This can be achieved by:
 1. **threshold** the change map
 - ⇒ select threshold to binarize image in burned/non-burned areas
 2. **mask** the thresholded map
 - ⇒ make pixels with value = 0 invalid (i.e. not burned)
 3. **update** the burned mask
 - ⇒ exclude from the mask the pixels corresponding to water bodies (e.g. lakes, rivers, etc.)
 4. **mask** the change map using the burned mask
 - ⇒ get the *severity map* of the burned regions

2.4. Analyze change map

Step 4: Analyze change map

1. Thresholded change map



```
# Threshold change map
lt_threshold, lt_newval = -0.25, 1 # select threshold and new value = 1 (= valid)
img_change_thresh = ee.Image(0)      # create image filled with 0 (= invalid)
img_change_thresh = img_change_thresh.where(img_change.lte(lt_threshold), lt_newval)
Map.addLayer(img_change_thresh, {'palette':['white', 'black']}, 'Change map thresholded')
```

2. Mask of burned area



```
# Get mask of burned area
mask_burned = img_change_thresh.selfMask() # make pixels with value = 0 invalid
Map.addLayer(mask_burned, {'palette':['white', 'black']}, 'Burned mask')
```

2.4. Analyze change map

3. Water mask



```
# Get water bodies (to exclude from burned mask)
water_mask = (ee.Image("JRC/GSW1_1/GlobalSurfaceWater")
              .select('occurrence') # frequency with which water was present (since 1984)
              .gte(50)
            ) # pixel values: 1=water / None=not-water
Map.addLayer(water_mask, {'palette':['white', 'blue']}, 'water')
```

4. Updated burned mask



```
# Exclude water bodies from burned mask
# => need to invert water mask (we need invalid values (= 0) where water is):
#   .unmask => converts None values to 0
#   .eq(0)  => tests if pixel=0 => where water used to be (=1), sets False (=0=invalid)
water_mask_invert = water_mask.unmask(0).eq(0)
mask_burned = mask_burned.updateMask(water_mask_invert) # Update mask
Map.addLayer(mask_burned, {'palette':['white', 'black']}, 'Burned mask (water excluded)')
```

2.4. Analyze change map

5. Burned regions severity



```
# Get burn severity
# => mask `change map` using mask `mask_burned`
burned_severity = img_change.updateMask(mask_burned)

# Get min/max in burned_severity masked image
minMax = burned_severity.reduceRegion(
    reducer=ee.Reducer.minMax(),
    geometry=burned_severity.geometry(),
    maxPixels=1e10,
)
min = minMax.get('change_min'). getInfo()      # property name = <band_name>_min
max = minMax.get('change_max'). getInfo()      # property name = <band_name>_max
vis_params = {'palette':'magma', 'min':min, 'max':max}
Map.addLayer(burned_severity, vis_params, 'Burned regions severity')
```

2.4. Analyze change map

