Estimate crop area based on crop mask (single year)

Author: Hannah Kerner (hkerner@umd.edu) and Adebowale Daniel Adebayo (aadebowaledaniel@gmail.com)

Description:

This notebook performs the following steps:

1. Copies existing crop map from Google cloud storage
2. Clips crop map to a regional boundary (admin1, admin2 shape or user-defined bounding box)
3. Thresholds the crop map to a binary mask of 0 (noncrop) or 1 (crop)
4. Creates a random stratified sample from the crop mask for labeling in CEO
5. Computes the confusion matrix between the labeled reference sample and the crop mask
6. Calculates the crop and noncrop area and accuracy estimates based on Olofsson et al., 2014

Note:

This notebook can be either be use on Google Colab or your local computer. Therefore, if you are using your local computer, skip the Colab Setup step and start with the Local Setup section.

If your map size is >7GB consider running this notebook on your personal computer or a virtual machine with RAM >12GB.

```
# Clone the crop-mask repository
# Skip this step if you have already cloned the repository or running locally
# email = input("Github email: ")
# username = input("Github username: ")

# !git config --global user.email $username
# !git config --global user.name $email

# from getpass import getpass
# token = getpass('Github Personal Access Token:')
!git clone https://$username:$token@github.com/nasaharvest/crop-mask.git
%cd crop-mask
```

```
Cloning into 'crop-mask'...
warning: redirecting to https://github.com/nasaharvest/crop-mask.git/
remote: Enumerating objects: 11257, done.
remote: Counting objects: 100% (1910/1910), done.
remote: Compressing objects: 100% (713/713), done.
remote: Total 11257 (delta 1255), reused 1757 (delta 1183), pack-reused 9347
Receiving objects: 100% (11257/11257), 115.66 MiB | 22.20 MiB/s, done.
Resolving deltas: 100% (7309/7309), done.
/content/crop-mask
```

Colab Setup

- Note: You must be logged into Colab with the same account that you will use to authenticate.
- You need to authenticate your google account in order to access the cloud storage where the map is saved.

```
# Authenticate Google Cloud
# from google.colab import auth
# print("Logging into Google Cloud")
# auth.authenticate_user()
```

```
# Install required packages
# Skip this step if you have already installed the packages in your local environment
!pip install geopandas -q
!pip install seaborn -q
!pip install rasterio -q
!pip install cartopy -q
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.6/20.6 MB 43.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 51.3 MB/s eta 0:00:00
```

Local Setup

- Check setting up a local environment with conda here.

```
# Import libraries
import os
import sys
import numpy as np
import seaborn as sns
from shapely.geometry import box
import geopandas as gpd
```

```
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
# Import crop area estimation functions
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)

from src.area_utils import (
    load_ne,
    load_raster,
    binarize,
    cal_map_area_class,
    estimate_num_sample_per_class,
    generate_ref_samples,
    reference_sample_agree,
    compute_confusion_matrix,
    compute_area_estimate,
    create_area_estimate_summary,
    create_confusion_matrix_summary
)
```

- NOTE: You can skip this step if you already have/downloaded your map, then change the `map_path = os.path.basename(bucket_uri.value)` to `map_path = "relative_path_to_your_map"` in the Load the crop mask.
- Paste the map gsutil URI (file path in the cloud storage) to download/copy the map into local storage in Colab or your personal computer.

```
# Download the map from the cloud storage by providing bucket URI
# Example: gs://crop-mask-final-maps/2016/China/epsg32652_Heilongjiang_2016.tif
```

```
# import ipywidgets as widgets
# bucket_uri = widgets.Text(description="Bucket URI:", placeholder="Paste the crop map bucket uri or file path: gs://", layout=
# bucket_uri
```

```
# !gsutil du -h $bucket_uri.value
```

```
# Download the map
# !gsutil cp $bucket_uri.value .
```

Load Region of Interest(ROI)

- Note: If the ROI is an administrative boundary and the map has not been clipped to it, the following steps download one (note: this functionality is available for admin1 or admin2 boundaries).
- If you want to use the dimensions of a bounding box instead of a shapefile, you may define a bounding box in the next cell.

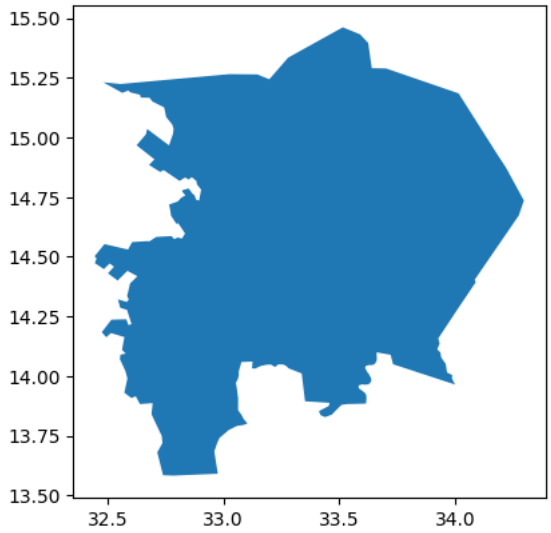
```
# country_iso_code = 'ETH' # Can be found https://www.iso.org/obp/ui/#search under the Alpha-3 code column
# adm1_of_interest = ['Tigray']

# gadm2_path = f'https://geodata.ucdavis.edu/gadm/gadm4.1/json/gadm41_{country_iso_code}_2.json.zip'
# roi = gpd.read_file(gadm2_path)
# roi = roi.query('NAME_1 in @adm1_of_interest')
# roi.head()
```

```
# Optionally restrict ROI to an admin2 boundary
# adm2_of_interest = ["Mi'irabawi"]
# roi = roi.query('NAME_2 in @adm2_of_interest')


# Merge selected region(s) into a single polygon
# roi = roi.dissolve()


# roi.plot()


#########################################################
## MOUNTING THE GDRIVE, TO ACCESS FILES             ###
#########################################################

from google.colab import drive
drive.mount('/gdrive')

#/gdrive/Sudan-Maps
```

```
Mounted at /gdrive
```

```
aoi = gpd.read_file('/gdrive/MyDrive/Sudan-Maps/area_of_interest.geojson')
aoi = aoi[aoi['NAME_1']=='Al Jazirah']
roi = aoi.dissolve()
roi.plot()
```

```
<Axes: >
```




```
# Optionally specify bounding box boundaries to clip to
# Note that these boundaries must be in the same CRS as the raster
# You can get this from bboxfinder, e.g.: http://bboxfinder.com/#10.277000,36.864900,10.835100,37.191000

def getFeatures(gdf):
    """Function to parse features from GeoDataFrame in such a manner that rasterio wants them"""
    import json
    return [json.loads(gdf.to_json())['features'][0]['geometry']]

minx, miny, maxx, maxy = # your optional bbox bounds, e.g.
                         # 249141.6217,840652.3433,272783.1953,855138.2342
target_crs = #'EPSG:XXXXX'
bbox = box(minx, miny, maxx, maxy)
geodf = gpd.GeoDataFrame({'geometry': bbox}, index=[0], crs=target_crs)
roi = getFeatures(geodf)
```

Load the crop mask

- Loads the map from the .tif file as a numpy array. If ROI is specified above, an array masked with ROI is returned; else, the whole map extent is returned as a numpy array.
- To make sure your rasters are projected using the local UTM zone (e.g., EPSG:326XX where XX is the 2-digit UTM zone), you will be prompted to input the EPSG code for the ROI if the map has not already been projected (i.e., the map CRS is EPSG:4326).
- The projected map will be saved as `prj_{the base name}.tif`.

```
####################################
## Loadin the map from drive       ###
####################################
os.listdir('/gdrive/MyDrive/Sudan-Maps')
```

```
['2022-ceo-sample.csv',
 'area_of_interest.geojson',
 '2023-ceo-samples.csv',
 'Maps_2022',
 'Maps_2023',
 'crop_area_estimation.ipynb']
```

```
# map_path = os.path.basename(bucket_uri.value) # uncomment if you just downloaded from cloud storage
map_path = "/gdrive/MyDrive/Sudan-Maps/Maps_2023/aljazirah_merged_2023_epsg32637.tif" # uncomment and replace with the path to y
assert os.path.isfile(map_path), "Map file not found"
```

```
map_array, map_meta = load_raster(map_path, roi) #uncomment to clip with roi
# map_array, map_meta = load_raster(map_path) #uncomment to load without clipping; using the map extent
```

```
Map CRS is EPSG:32637. Loading map into memory.
Clipping to boundary.
The pixel size is 9.884 meters
```

```
UTM Zone 35T: EPSG 32635
UTM Zone 36T: EPSG 32636
UTM Zone 37T: EPSG 32637
UTM Zone 38T: EPSG 32638
```

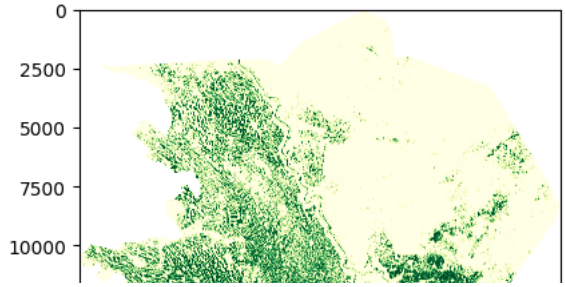
Binarize the predicted maps to create crop masks

- Convert the maps to 1 where p >= 0.5 and 0 where p < 0.5.
- Leave no-data values (255 if using the example above) as is - this should be automatically handled since the rasters were loaded as masked arrays.
- Cast the type to uint8 since they should now have values of 0, 1, or 255/masked.

```
if map_array.data.dtype == "uint8": # If the map is already binarized
    binary_map = map_array.data
else:
    binary_map = binarize(map_array, map_meta)
```

```
# Plot the map to make sure it looks as expected
# This may take a while depending on the size of the map,
# so you may choose not to run this every time.
import matplotlib.pyplot as plt
plt.imshow(binary_map, cmap='YlGn', vmin=0, vmax=1)
```

```
<matplotlib.image.AxesImage at 0x79851b78cd90>
```



Calculate the mapped area for each class

```
crop_area_px, noncrop_area_px = cal_map_area_class(binary_map, unit='pixels')
crop_area_ha, noncrop_area_ha = cal_map_area_class(binary_map, unit='ha')
crop_area_frac, noncrop_area_frac = cal_map_area_class(binary_map, unit='fraction')
```

```
    Crop pixels count: 60567764, Non-crop pixels count: 189547662 pixels
                Total counts: 250115426 pixels
    Crop area: 605677.64 ha, Non-crop area: 1895476.62 ha
                 Total area: 2501154.26 ha
    Crop area: 0.24 fraction, Non-crop area: 0.76 fraction
```

Create random stratified reference sample from change map strata following best practices

First we need to determine the number of total samples we want to label for our reference dataset.

We use the method identified by Olofsson et al. in Good practices for estimating area and assessing accuracy of land change (eq 13) to determine sample size:

$$n \approx (\Sigma(W_i S_i) / S(\hat{O}))^2$$

Where	
W_i	Mapped proportion of class i
S_i	Standard deviation $\sqrt{(U_i(1-U_i))}$
U_i	Expected user's accuracy for class i
$S(\hat{O})$	Desired standard error of overall accuracy
n	Sample size

If you have already used an independent validation or test set to estimate the user's accuracy (precision) for each class, you can plug those values into this equation. If you have not already calculated it, you will need to make a guess (it is better to make a conservative guess since an overestimation may lead to fewer points than are actually needed to achieve low standard errors). See the example calculation below for user's accuracy of both classes of 0.63 and a standard error of 0.02.

```
u_crop = widgets.Text(description="u_crop:",
                      placeholder="Expected user's accuracy (precision) for crop class",
                      layout=widgets.Layout(height="5em", width="50%"))
u_noncrop = widgets.Text(description="u_noncrop:",
                      placeholder="Expected user's accuracy (precision) for non-crop class",
                      layout=widgets.Layout(height="5em", width="70%"))
stderr = 0.02
```

```
n_crop_sample, n_noncrop_sample = estimate_num_sample_per_class(crop_area_frac, noncrop_area_frac, u_crop, u_noncrop)
```

Now we can randomly draw sample locations using this allocation from each of the map strata.

```
# from util import sample_df
generate_ref_samples(binary_map, map_meta, n_crop_sample, n_noncrop_sample)
```

Label the reference samples in CEO

This step is done in Collect Earth Online. First you need to create a labeling project with the shapefile we just created (two copies for consensus). Once all of the points in both sets have been labeled, come back to the next step.

See the instructions for labeling planted area points here.

Load the labeled reference samples and get the mapped class for each of the reference samples

There should be two sets of labels for the reference sample. We compare the labels from each set to filter out labels for which the labelers did not agree, and thus we can be confident about the true label.

Upload the labeled reference sample and paste the relative paths.

```
# paths to the labeled reference samples
ceo_set_1 = '/content/ceo-Sudan-Feb-2023---Feb-2024-(Set-1)-sample-data-2023-11-18.csv'
ceo_set_2 = '/content/ceo-Sudan-Feb-2023---Feb-2024-(Set-1)-sample-data-2023-11-18.csv'
```

```
ceo_geom = reference_sample_agree(binary_map, map_meta, ceo_set_1, ceo_set_2)
```

```
    Number of NANs/ missing answers in set 1: 0
    Number of NANs/ missing answers in set 2: 0
    The number of rows in the reference sets are equal.
    Number of samples that are in agreement: 1196 out of 1196 (100.00%)
```

```
ceo_geom = ceo_geom[ceo_geom['Mapped class'] != 255]
```

```
ceo_geom.shape
```

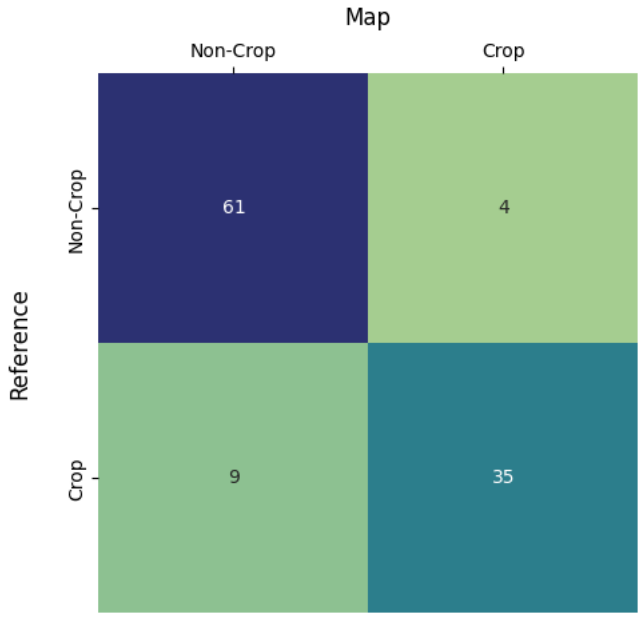
```
    (109, 15)
```

Compute the confusion matrix between the mapped classes and reference labels

```
cm = compute_confusion_matrix(ceo_geom)
labels = ["Non-Crop", "Crop"]
```

```
def plot_confusion_matrix(cm,labels) -> None:
    """Pretty prints confusion matrix.

    Expects row 'Reference' and column 'Prediction/Map' ordered confusion matrix.

    Args:
        cm:
            Confusion matrix of reference and map samples expressed in terms of
            sample counts, n[i,j]. Row-column ordered reference-row, map-column.
        labels:
            List-like containing labels in same order as confusion matrix. For
            example:

            ["Stable NP", "PGain", "PLoss", "Stable P"]

            ["Non-Crop", "Crop"]

    """

    _, ax = plt.subplots(nrows=1, ncols=1)
    sns.heatmap(cm, cmap="crest", annot=True, fmt="d", cbar=False, square=True, ax=ax)
    ax.xaxis.tick_top()
    ax.xaxis.set_label_coords(0.50, 1.125)
    ax.yaxis.set_label_coords(-0.125, 0.50)
    ax.set_xticklabels(labels=labels)
    ax.set_yticklabels(labels=labels)
    ax.set_xlabel("Map", fontsize=12)
    ax.set_ylabel("Reference", fontsize=12)
    plt.tight_layout()
```

```
plot_confusion_matrix(cm, labels)
```




```
confusion_summary = create_confusion_matrix_summary(cm, labels)
```

```
                        Non-Crop  Crop
    False Positive Rate     0.20  0.06
    True Positive Rate      0.94  0.80
    Accuracy                0.60  0.40
```

Adjust mapped area using confusion matrix to compute area estimates

```
# Marginal pixel totals
a_j = np.array([noncrop_area_px, crop_area_px], dtype = np.int64)

# Pixel size
px_size = map_meta["transform"][0]

# Area estimate
estimates = compute_area_estimate(cm, a_j, px_size)
```

U_j is the user's accuracy (i.e., precision) for each mapped class expressed in terms of area proportion.

```
u_j, err_u_j = estimates["user"]
print(f"User's accuracy and 95% CI\n{u_j.round(2)}\n{(err_u_j).round(2)}")
```

```
    User's accuracy and 95% CI
    [0.87 0.9 ]
    [0.08 0.1 ]
```

P_i is the producer's accuracy (i.e., recall) for each reference class, also expressed in terms of area proportion.

```
p_i, err_p_i = estimates["producer"]
print(f"Producer's accuracy and 95% CI\n{p_i.round(2)}\n{(err_p_i).round(2)}")
```

```
    Producer's accuracy and 95% CI
    [0.96 0.69]
    [0.03 0.13]
```

O is the overall accuracy.

```
acc, err_acc = estimates["accuracy"]
print(f"Overall accuracy and 95% CI\n{acc.round(2)} \u00B1 {(err_acc).round(2)}")
```

```
    Overall accuracy and 95% CI
    0.88 ± 0.06
```

A_i is the area estimate for each class.

```
a_i, err_a_i = estimates["area"]["pr"]
print(f"Estimated area [proportion] and 95% CI of area [proportion] \n{np.stack([a_i, err_a_i]).round(2)}")
```

```
Estimated area [proportion] and 95% CI of area [proportion]
[[0.69 0.31]
 [0.06 0.06]]
```

A_{px} is the adjusted area estimate in units of pixels.

```
a_px, err_px = estimates["area"]["px"]
print(f"Estimated area [pixels] and 95% CI of area [pixels] \n{np.stack([a_px, err_px]).round()}")
```

```
Estimated area [pixels] and 95% CI of area [pixels]
[[1.71389327e+08 7.87260990e+07]
 [1.60702690e+07 1.60702690e+07]]
```

A_{ha} is the adjusted area estimate in units of hectares.

```
a_ha, err_ha = estimates["area"]["ha"]
print(f"Estimated area [ha] and 95% CI of area [ha] \n{np.stack([a_ha, err_ha]).round()}")
```

```
Estimated area [ha] and 95% CI of area [ha]
[[1674485.  769159.]
 [ 157008.  157008.]]
```

Summary of the final estimates and 95% confidence interval of area [ha], user's accuracy, and producer's accuracy for each class.

```
summary = create_area_estimate_summary(a_ha, err_ha, u_j, err_u_j, p_i, err_p_i, columns = ["Non-Crop", "Crop"])
```

```
                         Non-Crop       Crop
Estimated area [ha]  1674485.15  769159.24
95% CI of area [ha]   157007.60  157007.60
User's accuracy            0.87       0.90
95% CI of user acc.        0.08       0.10
Producer's accuracy        0.96       0.69
95% CI of prod acc.        0.03       0.13
```