Stahl, A.T., Fremier, A.K., Heinse, L., 2021. Cloud-Based Environmental Monitoring to Streamline Remote Sensing Analysis for Biologists. *BioScience*. https://doi.org/10.1093/biosci/biab100

User Guide and links to online resources: https://labs.wsu.edu/ecology/research-projects/cbem-user-library/

Scripts and documentation with updates hosted on GitHub: https://github.com/ATStahl/CBEM

Supplementary Materials

These scripts and subsequent updates will be hosted on Github (https://github.com/ATStahl/CBEM).
All lines without black background in JavaScript can be copied and pasted directly in the Earth Engine Code Editor (https://code.earthengine.google.com/)

Script 1: Inspect imagery, classify cover, and evaluate accuracy

This script executes the following tasks as implemented in the study area, eastern Washington State, USA.

I. Search for all available Sentinel-2 satellite imagery in the study area over the time period of interest.
II. Create an average (median) composite image for each specified time interval, discarding the cloudiest images and replacing any remaining cloudy pixels with pixels from another date.
III. Compute indices of vegetation vigor from the composite images.
IV. Build and train a model to classify land cover with selected spectral bands and indices from the composite images.
V. Apply the model to classify land cover in other areas or timeframes (compared to the image that was used to train the model.
VI. Evaluate the accuracy of the model with an independent set of validation polygons.

Annotations above each block of code indicate what those lines accomplish and how they can be adapted to different study areas or timeframes. Note that "//" marks text that will be disregarded by GEE, so those lines can be copied directly into the Code Editor for quick reference. In some instances, "//" are used to prevent lines of code from being executed during a given model run. This helps to manage performance and keep each run of the script within the memory limitations of GEE. (Comment lines will be shown in green text in the Code Editor).

```
// The purpose of this script is to create a cloud-free
    // Sentinel-2 satellite composite image for the study area.
// It averages spectral data over the time interval of interest:
    // 15 August to 1 October 2018.
// It then builds a model using spectral bands B4 and B11 combined with two
    // indices of vegetation vigor (NDVI and NDRE[1]) from the input image.
// The user draws and labels polygons to indicate 4 cover classes: impervious
    // surfaces, water, brown vegetation, green vegetation.
// The labeled polygons are input into a Random Forest classifier with 100 trees.
// The classifier is then applied to classify each of the late season
```

[1] NDVI = Normalized Difference Vegetation Index = (Near infrared - Red) / (Near infrared + Red). This is used to quantify "greenness" and is used to track vegetation vigor.
NDRE = Normalized Difference Red Edge Index = (Near infrared – Red edge) / (Near infrared + Red edge). This is used to further refine measurements of vegetation vigor, particularly for understory or mid-late season crops.

```
    // (15 August to 1 October) composite images: 2016, 2017, 2018, 2019
// The user draws and labels validation polygons with the 4 cover classes on
    // one of the classified images (other than the input image). These are
    // input for the confusion matrix, used to evaluate accuracy
// Finally, it exports classified late season composite images to Assets or
    // Google Drive for subsequent analysis.

//***NOTE: Some lines are commented out with "//" so that they will not use
    // computing power unless needed in the current run. The user can then
    // choose which lines to activate for each run for efficiency in producing
    // desired outputs.
```

"Assets" provide a way of importing GIS files that were created outside of GEE. They also enable the user to save outputs from a previous run of the script so that they can be imported back into the script for quick analysis with minimal memory usage. The comment lines below are simply a note to indicate that some of the objects used in this script are stored as Assets, either before any lines are executed (e.g., the study area outline) or during the execution of the script (e.g., hand-drawn polygons and classified image composites).

```
// Imported assets that accompany this script will include training polygons,
    // classified image composites (after classification step has run),
    // and polygons for validation.
```

Below we have added two lines of code to create a simple rectangular polygon outlining the study are for this script. It covers the area analyzed by Stahl et al. 2021. (The shapefile for that study area is available on Github). If you wish to analyze a different area, you have three options.

(1) Enter bounding coordinates (longitude, latitude in decimal degrees) to outline a study area anywhere on the globe.
(2) Draw a polygon or rectangle on the map. Exit drawing, then hover over the geometry layer to open the geometry settings, rename it "ROI" and choose to import it as a FeatureCollection.
(3) If you have a shapefile for your study area, you can import it as an Asset (upload the 6 files that comprise the shapefile, leaving out the ".sbx" if there is one. Alternatively, you can upload a single zip file containing the 6 files in the shapefile). After it has been ingested (check Task pane for progress), import the Asset into this script, and then assign it to a variable by clicking "table" and replacing it with "ROI" – those steps will change the study area in this script.

Note that a larger study area may take longer to process requests.

```
//***Set the boundaries of the area for image querying and analysis***
    // If changing to another study area with option (2) or (3) above, delete or
```

```
76      // comment out the following two lines of code (lines 78-85).
77      // To set the study area boundaries with latitude and longitude coordinates,
78        // replace the coordinates below with the coordinates of the bounding polygon
79        // for your area. Enter longitude, then latitude for each point.
80      var studyAreaGeometry = ee.Geometry.Polygon([
81        [[-119.1098, 47.7328],
82        [-115.5251, 47.7328],
83        [-115.5251, 45.3685],
84        [-119.1098, 45.3685]]
85        ]);
86
87      var ROI = ee.FeatureCollection(studyAreaGeometry);
88
```

In this script, we will apply a function to mask clouds each time we create a composite image. The cloud mask function in the lines below is written near the top of the script so that it will be available when called later on. Sentinel imagery has a band labeled 'QA60' that can be used to mask clouds; that is what is used here. Other image sources such as Landsat imagery have searchable code snippets to deal with clouds.

```
95  // Function "maskS2clouds" to mask clouds using the Sentinel-2 QA60[2] (cloud mask)
96      // band. This function first selects the QA60 band for the image and assigns
97      // it to variable "qa".
98        function maskS2clouds(image) {
99          var qa = image.select('QA60');
100       // Bits 10 and 11 correspond to opaque clouds and cirrus clouds,
101             // respectively. For each pixel, 0 indicates no clouds present, 1
102             // indicates clouds are present. Here we assign each type of cloud bit
103             // in our image area to a variable.
104         var cloudBitMask = 1 << 10;
105         var cirrusBitMask = 1 << 11;
106
107         // Next, we create a masked image including only pixels for which both
108             // cloud flags are set to zero, indicating clear conditions.
```

[2] QA = quality; 60 indicates 60x60 meter spatial resolution. For more background on the use of the QA60 band for cloud masking, see resources linked from the User Library.

```
        var mask = qa.bitwiseAnd(cloudBitMask).eq(0).and(
                   qa.bitwiseAnd(cirrusBitMask).eq(0));
        // Return the masked data, scaled by dividing by 10000 for convenience,
            // without the QA bands (because we no longer need them) and including
            // the time of image capture.
        return image.updateMask(mask).divide(10000)
            .select("B.*")
            .copyProperties(image, ["system:time_start"]);

            }  //***include the closing bracket to complete this function!
```

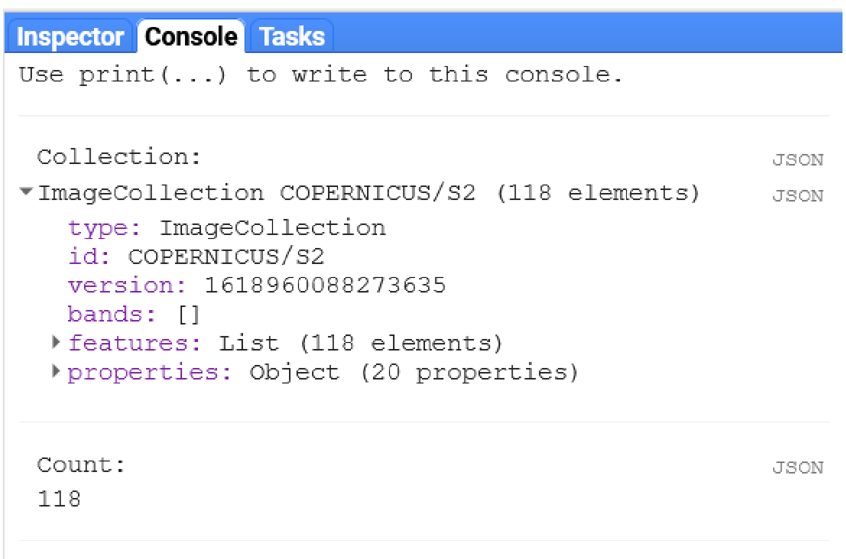
The following lines are used to query the Sentinel-2 image collection. Because the study area is semi-arid and the time interval was during the dry season, we were able to reliably use the top of atmosphere (Level 1C) product. Sentinel-2 imagery pre-processed for surface reflectance (Level-2A) is becoming available on GEE and may be more appropriate to use in some settings. First, we well query and create composite image to train the classifier. Then we will repeat the same process to query and composite an image to classify.

```
//***Query Sentinel-2 image collection***

// * Training image *
// Create a variable to store the training image collection overlapping the
  // study area (variable "ROI" that was assigned above), filtered to the time
  // period of interest
    var trainingCollection = ee.ImageCollection('COPERNICUS/S2')
        .filter(ee.Filter.bounds(ROI))
        .filterDate('2018-08-15', '2018-10-01')

// Pre-filter to get fewer cloudy granules (here, we are including only images with
    // less than 20% cloud cover -- you can change to other thresholds if needed,
    // e.g. 50 or 60% for a frequently cloudy area)
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))

// Apply the cloud mask to include only cloud-free pixels in each image using the
    // "maskS2clouds" function created above.
    .map(maskS2clouds);   // (the line of code ends here with semicolon)
```

```
// Average the images to create a single composite image by taking the median value
    // of each cloud-free pixel for all images queried.
    var trainingMedian = trainingCollection.median();

// Clip the composite image to the study area outline (ROI)— we now have a single
    // image for the study extent and timeframe of interest.
    var trainingImage = trainingMedian.clipToCollection(ROI);


// * Image to be classified *
// Here we repeat the process used above to generate a training image. First, we
    // create a variable to store the training image collection overlapping the
    // study area (ROI), filtered to the time period of interest (change dates
    // in lines below as needed).
  var classifyCollection = ee.ImageCollection('COPERNICUS/S2')
      .filter(ee.Filter.bounds(ROI))
      .filterDate('2019-08-15', '2019-10-01')  //change dates in parentheses
                                                       // as desired

// Pre-filter to get less cloudy granules. Use the same cloud threshold as for
      training image above (here it's set to 20).
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
    .map(maskS2clouds);

// Average the images by taking the median value of each pixel to create a
    // single composite image.
    var classifyMedian = classifyCollection.median();

// Clip the composite image to the study area outline (ROI)
    var classifyImage = classifyMedian.clipToCollection(ROI);


//To view information about available imagery (replace "classifyCollection"
    // below with "trainingCollection" or another image collection variable
```

```
    // to get information about it.
print('Collection: ', classifyCollection);      //comment this line out after
      //you have the information so it does not use memory in subsequent runs.


//Get the number of images. Replace "classifyCollection" with another image
    // collection variable to get information about it.
var count = classifyCollection.size();
print('Count: ', count);            //comment this line out after you have the
                // information so it does not use memory in subsequent runs.
```

Pause here. Copy all code lines above this point and paste them into the Code Editor. Save the script with a name of your choosing and click Run. In the Console, you will see information appear about the Image Collection and the number of images found in the query (see example in image below). Once you have viewed the information, you can choose to comment out these "print" lines to save memory on subsequent runs.



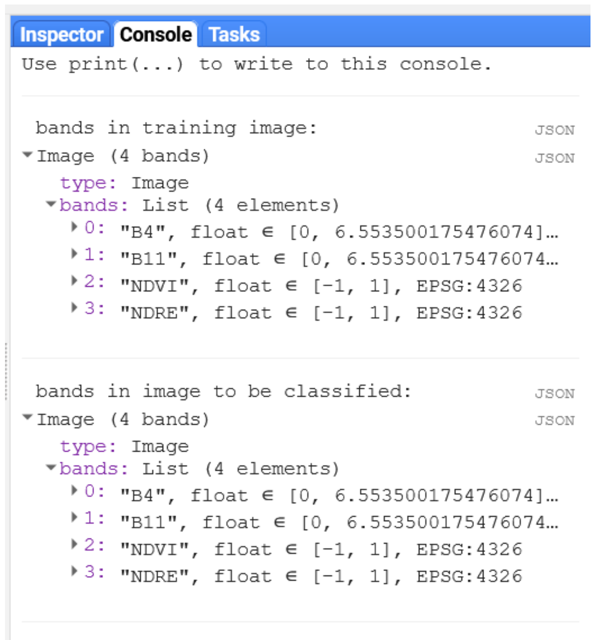
Next, we demonstrate how to compute vegetation indices. Here we compute the Normalized Difference Vegetation Index (NDVI) and the Normalized Difference Red Edge Index (NDRE). We then add them as bands to the images that will be trained and classified so they can be included in the random forest classification.

```
//***Compute NDVI and NDRE and add as bands to median composite images.***
//*Use the normalizedDifference(A, B) to compute (A - B) / (A + B) for each
    // index. We create functions to compute NDRE and NDVI and add the index
    // values as bands. Then these functions can be called repeated as needed for
```

```
    // subsequent images.
    var addNDRE = function(image) {
      var ndre = image.normalizedDifference(['B8', 'B5']).rename('NDRE');
      return image.addBands(ndre);
    };

    var addNDVI = function(image) {
      var ndvi = image.normalizedDifference(['B8', 'B4']).rename('NDVI');
      return image.addBands(ndvi);
    };

    //*Call the functions created above to compute and add NDVI and NDRE bands to
      // the image composites that will be used for training and classification.
    var ndre_train = addNDRE(trainingImage).select('NDRE');
    var ndvi_train = addNDVI(trainingImage).select('NDVI');
    var ndre_classify = addNDRE(classifyImage).select('NDRE');
    var ndvi_classify = addNDVI(classifyImage).select('NDVI');
    var indexParam = {min: -1, max: 1, palette: ['black', 'white']};

//***Set the map center location and zoom level, then add map layers.***
// Notes: You can use the Inspector to find coordinates and zoom level on the
    // map, then update these values in the Map.setCenter line accordingly.
    // In the lines below, “false” indicates that the map layer will not
    // be displayed by default (in Layers, the box will be unchecked), which
    // saves loading time. Image visualization parameters can be manually
    // adjusted and imported as a variable for subsequent script runs. To do
    // this, hover over the layer on the map and click the settings symbol.

// Here, we add true color and color infrared map layers for the training image,
    // as well as the vegetation indices for both training and classification
    // images. One could similarly display the true color and color infrared
    // layers for other images by substituting “trainingImage” in a Map.addLayer
    // line with the variable storing the desired image.
    Map.setCenter(-117.3052, 46.5685, 8);
    Map.addLayer(trainingImage, {bands: ['B4', 'B3', 'B2'], max: 0.5, gamma: 2},
```

```
        'Sentinel true color training image', false);
    Map.addLayer(trainingImage, {bands: ['B8', 'B4', 'B3'], max: 0.5, gamma: 2},
        'Sentinel color infrared training image', false);
    Map.addLayer(ndvi_train, indexParam,
        'NDVI in image sampled for training', false);
    Map.addLayer(ndre_train, indexParam,
        'NDRE in image sampled for training', false);
    Map.addLayer(ndvi_classify, indexParam,
        'NDVI in image to classify', false);
    Map.addLayer(ndre_classify, indexParam,
        'NDRE in image to classify', false);


//***Create multiband rasters to create customized image composites for training
    // and classification. Here we selected two bands and two vegetation indices
    // to use for classification. The remote sensing literature can provide
    // guidance on the best inputs to use for a given study.

//*Assign variables for each spectral band of interest and concatenate (link) with
    // spectral indices to create a single multiband image for training and
    // classification, respectively.
    var B4_train = trainingImage.select('B4');  // Red band
    var B11_train = trainingImage.select('B11');  // short wave infrared band
    var bandsTraining = ee.Image.cat(
          [B4_train, B11_train, ndvi_train, ndre_train]);
    print('bands in training image: ', bandsTraining);
              //comment out "print" line above after checking image bands

    var B4_classify = classifyImage.select('B4');
    var B11_classify = classifyImage.select('B11');
    var bandsClassify = ee.Image.cat(
          [B4_classify, B11_classify, ndvi_classify, ndre_classify]);
    print('bands in image to be classified: ', bandsClassify);
            //comment out "print" line above after checking image bands
```

Pause here. Copy and paste the lines above into the Code Editor. Save the script and click Run. In the Console, you will see information about the spectral bands or indices that are included in the images to be used for training and classification. This is one way to check that the code has run successfully thus far. (See example in image below.) Once you have viewed the information, you can choose to comment out these "print" lines to save memory on subsequent runs.



The next section goes through a script that executes the land cover classification in the accompanying article (Stahl et al. in review). Before these code lines can be used, one or more datasets is needed for model training and validation. These input data can be generated from existing field data or spatial datasets related to land cover that are available for the area and timeframe to be used for model training. In this case, we used visual inspection and local knowledge of the study area to hand-draw polygons representing each cover class. For model training, we referred primarily to Sentinel-2 satellite imagery during the timeframe of interest. For model validation, we used Sentinel-2 satellite imagery supplemented with visual inspection of Google Earth imagery, NAIP (National Agricultural Imagery Program, US Department of Agriculture) aerial imagery. Through visual interpretation, we identified areas as open water, impervious surfaces, green vegetation, or brown vegetation (including bare soil). The training polygons from this study are available as a zipped shapefile in the Github repository. You can import that shapefile into this script, import your own file with reference data or draw and label your own polygons in the map pane.

NOTE: you will not be able to run the remaining lines in this script until you have assigned one of these reference data sets (for the area you are analyzing) to the variable 'polygons'. If you do try to run without that step, you will receive an error message, such as "'polygons' is not defined in this scope."

```
//***Image classification (into 4 land cover classes: open water, impervious
    // surfaces, green vegetation, brown vegetation or bare soil)

// Use these bands for prediction. (These bands should have matching names and
```

```
301      // order to the concatenated images created above.)
302      var bands = ['B4', 'B11', 'NDVI', 'NDRE'];
303
304
305  // *Make a FeatureCollection from the hand-made geometries and assign it to a
306      // variable (“polygons”). (Here, we assigned a value of 0 to both “Other”
307      // (impervious surfaces) and “OpenWater” because they were not the focus of
308      // our analysis. One could designate a separate class for open water by
309      // coding it differently than “Other”)
310
311  // *Note: to run the remaining lines in this script, one would
312      // either draw polygons in the map viewer and label them by cover class
313      // (GreenVeg, BrownVeg, etc.) or ingest and import the shapefile of training
314      // polygons from github, then assign it to the variable "polygons" and
315      // delete the line of code below.
316      // If using hand-drawn polygons, remove comment marks to run the line of code
317      // below (lines 320-325). This will assign the polygons to a feature
318      // collection to be stored as the variable “polygons”.
319          //var polygons = ee.FeatureCollection([
320          //  ee.Feature(Other, {'class': 0}),
321          //  ee.Feature(OpenWater, {'class': 0}),
322          //  ee.Feature(BrownVeg, {'class': 1}),
323          //  ee.Feature(GreenVeg, {'class': 2}),
324          //  ]);
325
326
327  // *Get the class values for all pixels in each polygon in the training.
328  var training = bandsTraining.sampleRegions({
329    // Get the sample from the polygons FeatureCollection.
330    collection: polygons,
331    // Keep this list of properties from the polygons.
332    properties: ['class'],
333    // Set the scale to get Sentinel pixels in the polygons. (Adjust the scale
334       according to the spatial resolution of input imagery.)
335    scale: 10
```

Stahl, A.T., Fremier, A.K., Heinse, L., 2021. Cloud-Based Environmental Monitoring to Streamline Remote Sensing Analysis for Biologists. *BioScience*. https://doi.org/10.1093/biosci/biab100

User Guide and links to online resources: https://labs.wsu.edu/ecology/research-projects/cbem-user-library/

Scripts and documentation with updates hosted on GitHub: https://github.com/ATStahl/CBEM

```
336 });
337
338 // *Create a random forest classifier with 100 trees.
339 var classifier = ee.Classifier.smileRandomForest(100);
340
341 // *Train the classifier using the labeled training pixels created above (lines
342       // 328-336) and the bands indicated on line 302.
343       var trained = classifier.train(training, 'class', bands);
344
345
346 // *Classify the composite images. To use only minimum memory needed per run of
347     // the script, we recommend completing only one image classification per
348     // script run. One can then export the classified image to Assets (using
349     // Export line below) and subsequently import it into this or another script
350     // as needed. The completed classification line should then be commented
351     // out, deleted, or updated with a new image to classify.
352
353   // uncomment the line of code that follows to classify the composite image
354       // "bandsTraining" that was sampled for training (assuming that only
355       // selected portions of this image, e.g., the areas within hand-drawn
356       // polygons, were sampled for training)
357     // var trainingClassified = bandsTraining.classify(trained);
358         // remove "//" before "var" in line above to classify the training image
359
360
361   // uncomment the line below to classify the composite image "bandsClassify",
362       // note that you will see an error message saying "imageClassified is not
363       // defined in this scope as long as the line below is commented out.
364       // The classifier cannot be used until a set of training polygons has
365       // been either imported or hand-drawn in the Code Editor.
366   //var imageClassified = bandsClassify.classify(trained);
367 // Display classification results.
368     // Set visualization parameters for the classified images.
369     var ClassParam = {min: 0, max: 2, palette: ["373e8d","ffc772","20b82c"],
370                     opacity: 0.6};
```

```
// Remove comment marks from lines below as needed to display classification
    // results for current script run.
//Map.addLayer(trainingClassified, ClassParam, 'training image classified', false);
//Map.addLayer(imageClassified, ClassParam, 'image classified', false);
```

To minimize processing time and to avoid going over memory limits per script run, we recommend iterating through classifications and exporting each classified image to Assets. The lines below can be used to export the image that was classified in the current script run. The Tasks pane will show the ‘description’ indicated in the Export function. Click RUN and a dialog box will open. There you can specify where you wish to send the exported image--to your Earth Engine Assets for use in the Code Editor, or to your Google Drive as a TIFF for download.

```
//***Export classified image***
// Set image: current image to export, update description)--> choose option to
    // save as an Earth Engine Asset or TIFF in Google Drive. To follow
    // subsequent steps in this script, save classified images to your Assets.

       Export.image.toDrive({
               image: imageClassified,    //if this throws an error, check  that
                          // the line creating imageClassified is uncommented
               description: 'Late19classified_100trees', //enter name
               scale: 10,           //adjust as appropriate
               maxPixels: 1e9,      // adjust if needed, often need to set to 1e10
               region: ROI
               });

// After each classified image has been exported to Assets, it can be imported
    // and displayed for subsequent analysis (remove // before "Map" for each
    // corresponding classified image after import). In the accompanying article,
    // we classified four images and exported each to Assets, then imported into
    // this script. We assigned each image to a variable, such that “class16” is
    // the classified image composite from late summer 2016, and so on.
//Map.addLayer(class16, ClassParam, '2016 classification--imported Asset', false);
//Map.addLayer(class17, ClassParam, '2017 classification--imported Asset', false);
//Map.addLayer(class18, ClassParam, '2018 classification--imported Asset', false);
//Map.addLayer(class19, ClassParam, '2019 classification--imported Asset', false);
```

```
//Map.addLayer(polygons, {}, 'training polygons', false);  //uncomment this line to
    //show the training polygon outlines in the map display when the script is run.


//***Evaluating Accuracy***
// FeatureCollection to evaluate classification accuracy. First, create
    // hand-drawn polygons using Google Earth, Sentinel, or other
    // higher-resolution imagery if available for the area and timeframe of
    // interest. (For this approach, the total number of validation pixels must
    // be less than 5000.) Here the polygons were labeled EvalNonVeg,
    // EvalBrownVeg, or EvalGreenVeg and assigned values corresponding to the
    // classification above. (A shapefile containing example validation polygons
    // from the 2019 classified image is available on GitHub).

 // If using hand-drawn polygons, remove comment marks to run the line below.
    // var polyEval = ee.FeatureCollection([
      // ee.Feature(EvalNonVeg, {'vclass': 0}),
      // ee.Feature(EvalBrownVeg, {'vclass': 1}),
      // ee.Feature(EvalGreenVeg, {'vclass': 2}),
   //]);

// Sample specified classification results (class 16, class17, class18 or
    // class19 to validation areas (not to exceed 5000 pixels).
    var validation = class19.sampleRegions({
      collection: polyEval,
      properties: ['vclass'],
      scale: 10,
    });


//Compare the cover class of validation data against the classification result
    //(with a 2D error matrix).
    var testAccuracy = validation.errorMatrix('vclass', 'classification');

  //Print the error matrix to the console (uncomment line below to run)
```

```
    //print('Validation error matrix: ', testAccuracy);

  //Print the overall accuracy to the console (uncomment line below to run)
    //print('Validation overall accuracy: ', testAccuracy.accuracy());
```

Below is an example of what the print output to the Console looks like:

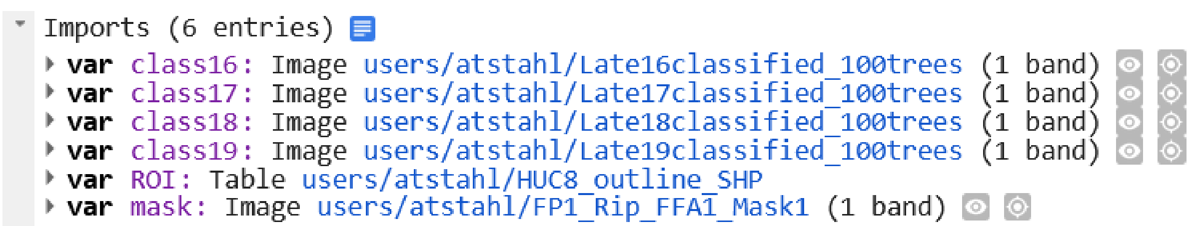
```
Validation error matrix:
[[619,8,2],[0,2613,193],[0,0,1731]]
  0: [619,8,2]
  1: [0,2613,193]
  2: [0,0,1731]

Validation overall accuracy:
0.9607046070460704
```

```
//***Additional information***
// Additional information can be extracted from the objects created in the
    // Code Editor. For example, to calculate the area of the training polygons
    // in square meters:
    // First, create an "image" of pixels with area in m^2
     var img = ee.Image.pixelArea().clip(polygons);

    // then use reducer to compute sum of areas in polygons.
     var area2 = img.reduceRegion({
      reducer: ee.Reducer.sum(),
      geometry: polygons,
      scale: 10,  // adjust as appropriate
      maxPixels: 1E13   //adjust if needed
     });

     // Display the results. (Remove comment marks before "print" in the line
        // below to display the area value).
    //print('area of training polygons: ', ee.Number(area2.get('area')).getInfo() +
      ' m2');
```

Script 2: Create change classes from images that were classified in Script 1

Note: This script will not work unless there are already classified images to import (see Script 1 or import classified images from another source).

```
//This is the Classification/Uncertainty script that was used to create
    // Figure 3e,f in Stahl et al. (2021), also shown on the User Library page.

// *** The purpose of this script is to generate uncertainty classes by querying
    // 2016-2019 composite images classified by a classifier trained on a 2018
    // image composite.It then computes area-based statistics for the uncertainty
    // classes.

// Before running this script, one must import classified images for each year
    // from Assets, here each is assigned to a variable named "class19",
    // "class18", and so on. We also imported a shapefile of the study area
    // (HUC8_outline_SHP) and a georeferenced TIFF file indicating riparian
    // areas (FP1_Rip_FFA1_Mask1). See example list of imports in image below.
```




```
// Set visualization parameters for the classified images.
var ClassParam = {min: 0, max: 2, palette: ["373e8d","ffc772","20b82c"],
                        opacity: 0.6};

// Set map center and display classified images for visual reference.
Map.setCenter(-117.4, 46.5, 8); //update coordinates for study area
Map.addLayer(class19, ClassParam, 'Class 2019');
Map.addLayer(class18, ClassParam, 'Class 2019');
Map.addLayer(class17, ClassParam, 'Class 2019');
Map.addLayer(class16, ClassParam, 'Class 2016');
```

```
// ***Create uncertainty classes using an expression, display.
  // first, concatenate classified images into single multiband image.
  var concatYears = ee.Image.cat([class16, class17, class18, class19]);
  print(concatYears);         //comment out unless needed to check output

  // Select and rename bands from the defaults to more user-friendly names.
  var diffYears = concatYears.select(
      ['classification', 'classification_1', 'classification_2',
      'classification_3'], // old names
      ['class16', 'class17', 'class18', 'class19']               // new names
  );
  print(diffYears);       //comment out unless needed to check output


// Set color palette for the change classes we will create.
  var palette = ['white', // 0 = not classified
                 'black',  // 1 = "non-vegetated" in all 4 years
                 'yellow', // 2 = "senesced" in all 4 years
                 'green', // 3 = "evergreen" in all 4 years
                 'magenta', // 4 = "evergreen" in at least 1 year, "senesced" in at
      least one year
                 'gray', // 5 = "other" in at least 1 year, "senesced" or "other" in
      other years
                 'blue']; // 6 = "other" in at least 1 year, "evergreen" in other
      years


// Create a series of nested conditional statements to create the desired change
      // classes.
var stabilityExp = diffYears.expression(
   "(b('class16') == 0) && (b('class17') == 0) && (b('class18') == 0) &&
      (b('class19') == 0) ? 1" +
    ": (b('class16') == 1) && (b('class17') == 1) && (b('class18') == 1) &&
      (b('class19') == 1) ? 2" +
      ": (b('class16') == 2) && (b('class17') == 2) && (b('class18') == 2) &&
      (b('class19') == 2) ? 3" +
```

```
        ": (b('class16') == 2) && ((b('class17') == 1) || (b('class18') == 1) ||
    (b('class19') == 1)) ? 4" +
          ": (b('class17') == 2) && ((b('class16') == 1) || (b('class18') == 1) ||
    (b('class19') == 1)) ? 4" +
            ": (b('class18') == 2) && ((b('class16') == 1) || (b('class17') == 1)
    || (b('class19') == 1)) ? 4" +
              ": (b('class19') == 2) && ((b('class16') == 1) || (b('class17') == 1)
    || (b('class18') == 1)) ? 4" +
                ": (b('class16') == 0) && ((b('class17') < 2) && (b('class18') < 2)
    && (b('class19') < 2)) ? 5" +
                  ": (b('class17') == 0) && ((b('class16') < 2) && (b('class18') <
    2) && (b('class19') < 2)) ? 5" +
                    ": (b('class18') == 0) && ((b('class16') < 2) && (b('class17')
    < 2) && (b('class19') < 2)) ? 5" +
                      ": (b('class19') == 0) && ((b('class16') < 2) &&
    (b('class17') < 2) && (b('class18') < 2)) ? 5" +
                        ": (b('class16') == 0) && ((b('class17') == 2) ||
    (b('class18') == 2) || (b('class19') == 2)) ? 6" +
                          ": (b('class17') == 0) && ((b('class16') == 2) ||
    (b('class18') == 2) || (b('class19') == 2)) ? 6" +
                            ": (b('class18') == 0) && ((b('class16') == 2) ||
    (b('class17') == 2) || (b('class19') == 2)) ? 6" +
                              ": (b('class19') == 0) && ((b('class16') == 2) ||
    (b('class17') == 2) || (b('class18') == 2)) ? 6" +
     ": 0"
);

// Display the cover change classification as a map layer using the color palette.
Map.addLayer(stabilityExp, {min: 0, max: 6, palette: palette},
      'stability classes 2016-2019', false);

// *Compute area of each cover change class for the study area.
    // NOTE: the following section of code can be repeated for any subset of
    // the study area. To do so replace "ROI" with the area of interest.

    // Clip the change classification to the study area.
    var class_ROI = stabilityExp.clipToCollection(ROI);

    // Add an area band (m^2) to the classified image so that we can compute areas.
```

```
    var addArea = ee.Image.pixelArea().addBands(class_ROI);

    // Use a Reducer to compute the area occupied by each cover change class in
      // the study area. The Reducer sums the pixels in each change class.
      var class_areas = addArea.reduceRegion({
            reducer: ee.Reducer.sum().group({
             groupField: 1,
              groupName: 'class_ROI',
             }),
             geometry: ROI,
             scale: 10,
             bestEffort: true,
      });

      // Display the area calculation outputs in the Console.
      print('area per uncertainty class', class_areas);


// Compute area of each transition class for riparian areas only. NOTE: these
    // lines require the user to provide a file to use for a mask (in this case,
    // we used a TIFF in which all riparian area cells had a value of 1.)
    // We imported it and assigned it to the variable "mask".

    // Mask the change classification to show only riparian areas in the study
      area.
    var class_masked = stabilityExp.updateMask(mask);

    // Add a band to the classified image so that we can compute areas.
    var addArea_rip = ee.Image.pixelArea().addBands(class_masked);

    // Use a Reducer to compute the area occupied by each cover change class in
      // the study area.
    var rip_class_areas = addArea_rip.reduceRegion({
        reducer: ee.Reducer.sum().group({
          groupField: 1,
```

```
          groupName: 'class_masked',
        }),
        geometry: ROI,
        scale: 10,
        bestEffort: true,
      });

    // Display the area calculation outputs in the Console.
    print('riparian area per uncertainty class', rip_class_areas);

// Export cover change classification. This line can be used to export the
    // cover change classification to Google Drive, where it can be downloaded
    // as a georeferenced TIFF file, or to Assets, from where it can be Imported
    // into other GEE scripts for further analysis, to share with others or to be
    // accessed by GEE Apps.
    Export.image.toDrive({
               image: class_ROI,
               description: 'StabilityClass_ROI',
               scale: 10,
               maxPixels: 1e9,
               region: ROI
               });
```

Note that when you try to view the exported image in Drive or a photo app, it will likely show up blank or all black. You will need to visualize the output raster in GIS software like ArcMap or ArcGIS Pro in order to view the change classes. Instructions for doing this are provided in the GitHub repository and the User Library.

Data Sources

Theobald DM, Mueller D, Norman J. 2013. Detailed datasets on riparian and valley-bottom attributes and condition for the Great Northern and Northern Pacific. Available from https://databasin.org/galleries/58411c761def4a54a477bebc48a57db1 (accessed May 19, 2015)

United States Geological Survey (USGS). 2013. National Hydrography Geodatabase. Available from https://ecology.wa.gov/ (accessed June 16, 2017).

Whitman County 2017. Whitman County Voluntary Stewardship Program Work Plan. Available from https://scc.wa.gov/vsp/ (accessed March 13, 2020).