
gee-ccdc-tools

Release 0.0.1

Erick Bullock, Paulo Arevalo

Sep 14, 2020

CONTENTS

- 1 Background 3**
 - 1.1 Background 3
- 2 Documentation 5**
- 3 Tutorials 7**
 - 3.1 CCDC Results Visualization Tutorial (GUI) 7
 - 3.2 Land cover tutorial (GUI) 12
 - 3.3 Land cover tutorial (API) 20
 - 3.4 Obtaining coefficients, changes and synthetic images (API) 30
- 4 API REFERENCE 35**
 - 4.1 API REFERENCE 35
- Index 49**

This is the documentation for the toolbox to use the Continuous Change Detection and Classification (CCDC) algorithm on the Google Earth Engine.

CCDC is used to monitor changes in land cover, land use, or condition using dense time series of satellite imagery. As the name suggests, there are two primary components: change detection and classification. CCDC has primarily been applied using Landsat data, but these tools are designed to be data agnostic. This toolbox provides applications and a Javascript API for performing all necessary steps in using CCDC, which include:

- Visualizing pixel-based time series and CCDC model fits
- Performing change detection for entire study regions
- Viewing CCDC coefficient images
- Producing “synthetic” imagery
- Land cover classification
- Creating maps of land cover/use change

This documentation and the tools are a work in progress. For any questions contact us at: parevalo@bu.edu or bullocke@bu.edu.

Contents

BACKGROUND

1.1 Background

TODO

DOCUMENTATION

TUTORIALS

3.1 CCDC Results Visualization Tutorial (GUI)

By Paulo Arévalo. May 14, 2020

To facilitate easy access to our API we have created a series of graphical user interfaces (GUIs) that require no coding by the user. These GUIs can be used for calculating CCDC model parameters (i.e. regression coefficients), displaying and interacting with CCDC coefficients and corresponding pixel time series, and classification of the model parameters. This tutorial will demonstrate the GUI for exploring Landsat time series and temporal segments fitted by the CCDC algorithm, as well as visualizing coefficients of the temporal segments, predicted images and change information.

In this guide you will learn how to:

- Explore time series of Landsat observations for a single pixel, as well as the temporal segments fitted on them by the CCDC algorithm.
- Visualize different coefficients of the temporal segments over space.
- Visualize images predicted from the temporal segments.
- Visualize change information.

The tool use in this tutorial can be found [here](#).

The tool might look like in the image below when you load it for the first time. To make sure you can visualize the map, please lower the separator bar that divides the map area and the time series chart area shown below.

The panel on the left controls the parameters for running CCD interactively for any point in the world with Landsat coverage. The panel on the right controls the interaction with the loaded CCDC results, and allows you to create predicted (synthetic) images, visualize maps of CCDC coefficients, and map the changes detected by the algorithm.

3.1.1 Creating charts of time series and interacting with them

If you want to visualize time series of Landsat spectral bands or a subset of indices derived from them given location, use the panel on the left to set up the desired band and time range (Box 1). Other CCD parameters can be left at their default parameters. In the example below, time series of the SWIR1 band are displayed after clicking on the map. You will notice that there are some segments missing in the chart. If this happens, you need to increase the Num segments parameter (e.g. to 10) in the Visualization params section (Box 2) and click on the pixel again. Creating the chart might take a little longer. You can also click on the points in the chart and they will be added to the map according to the visualization parameters selected for the RGB combination (Box 3). Right now any changes made there are not set on the fly, you need to set them before clicking on the map for them to take effect. In the image below, I clicked in one of the points and the image loaded with the default RGB combination.

The left panel also allows you to add any of your own assets (either image or feature collection) to the map. Given the current limitations imposed by GEE, the assets need to be publicly shared to be “seen” by the app, or they need to be

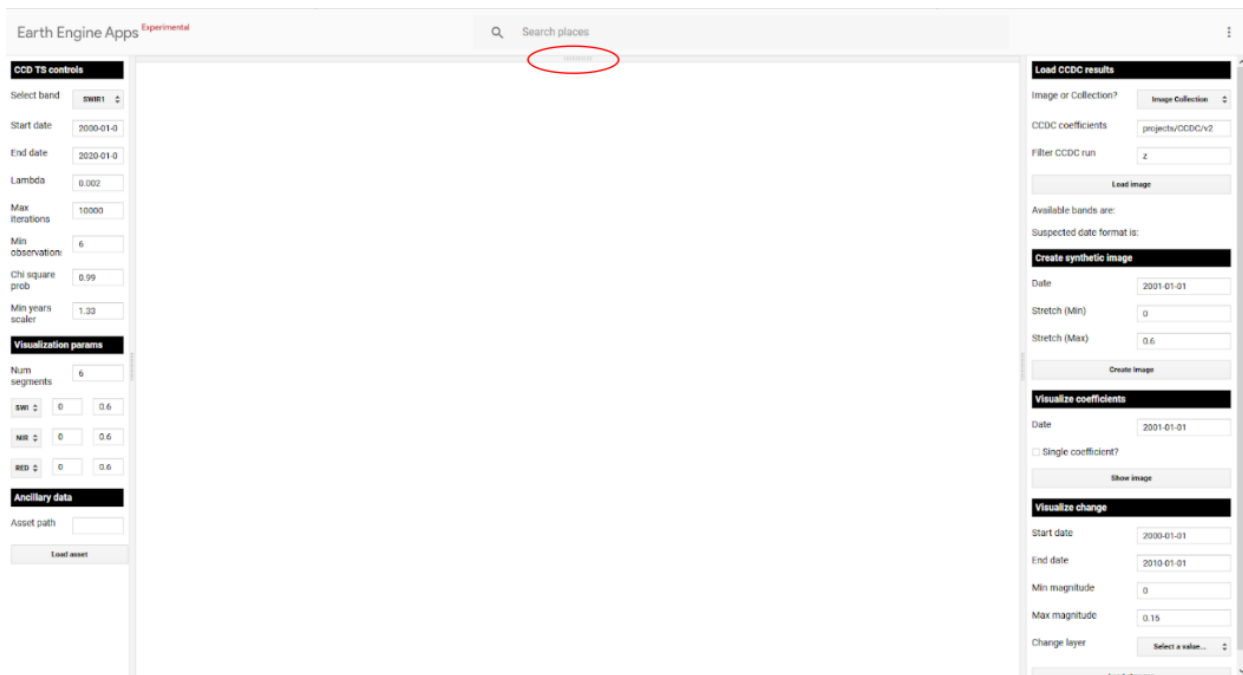


Fig. 1: Lower the horizontal division on top if the map is not visible..

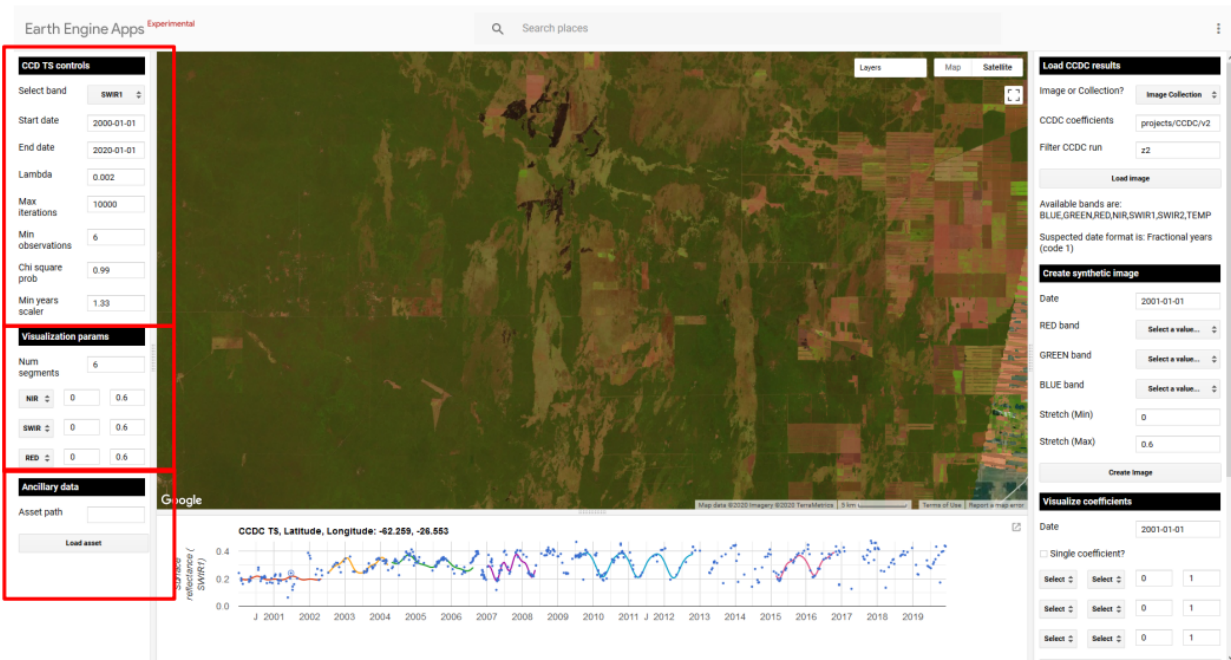


Fig. 2: Time series of a pixel with agricultural dynamics in Brazil

shared by the owner of the app. In the example below I changed the band to display NBR time series and modified the start date to begin in 1985. I set an RGB combination of NIR/SWIR1/RED that will be used to display the images loaded from the time series chart. Finally, I clicked on a pixel, and then clicked in the point show in the time series chart to visualize the image for that date.

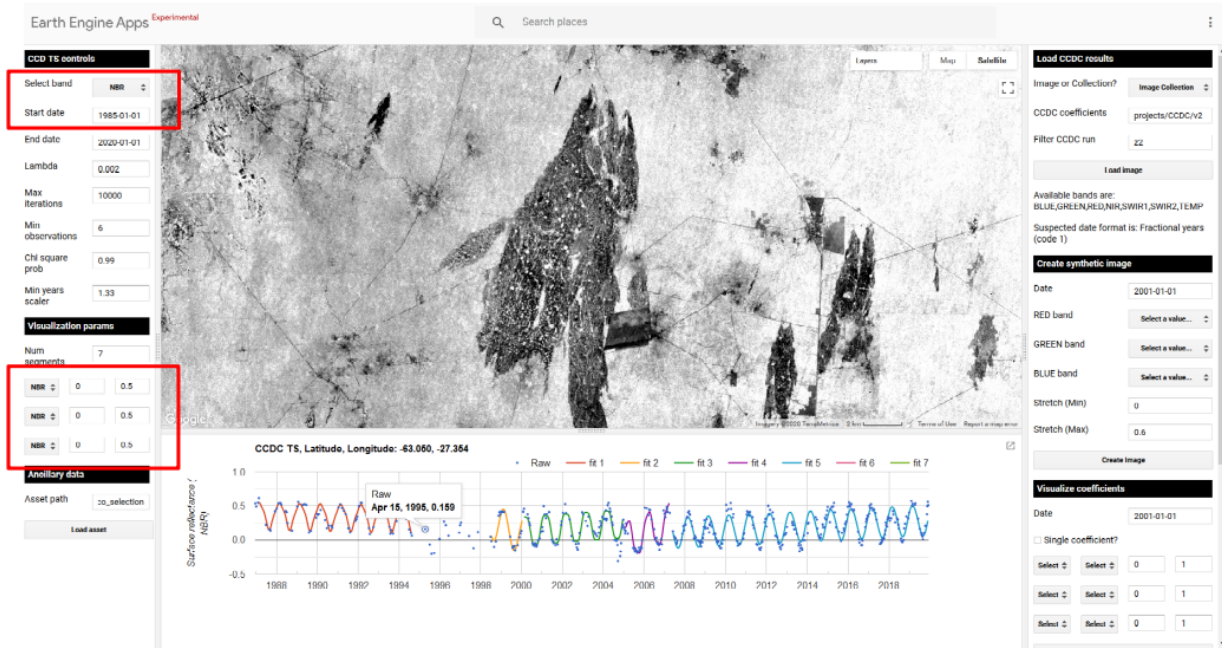


Fig. 3: Setting the parameters to run CCD on the fly and visualize images from the time series chart

3.1.2 Loading CCDC results

In order to use the subpanels on the right panel to generate maps of predicted images, coefficients and changes, you need to load pre-existing CCDC results. After this step is done, the rest of the sections in the tools can be used in any order. Keep in mind that this step is not necessary to visualize the time series of clicked points in the map (left panel of the tool). To load pre-existing CCDC results, look at the top right panel in the app, it must look like this:

The first few parameters describe the format of the CCDC results. First, are they saved as a single image or a collection? Next is the path to the CCDC results. Even though they are not officially public yet, we can interact with some of the CCD results that have been executed by Google. The default values, particularly the “z” in the filter CCDC run, contain results for the period between 1999-2020. After setting the desired run prefix, you can click on the Load image button. When the two fields below the button show their corresponding information, you will be able to interact with the rest of the options in the control panel in any order.

Fig. 4: Select which CCDC results to load using this panel. Once loaded, it will display the available band names and suspected date format of the results, if stored in the metadata at the time of creation.

3.1.3 Visualizing CCD coefficients and change information

Once the results have loaded, you can use the subpanels in the right as follows, in any order:

Generate predicted images: You can use the *Create synthetic image* panel to generate a predicted surface reflectance image for the date you specify. This is done by finding the intersecting temporal segment and using the coefficients to generate a predicted image for that date. The image will be displayed using the RGB combination specified using the dropdown boxes.

Generate maps of CCDC coefficients: You can use the *Visualize coefficients* panel to query and visualize the model coefficients and RMSE that intersect a given date. You can either visualize individual coefficient and specify the min and max values to stretch the visualization to, or you can create RGB images of different bands and min/max stretch values. In the image below you can see the RMSE of the nearest segment circa 1995 for a location in Brazil. You can see the fire scars are visible in the loaded image. You can experiment with changing the bands, coefficients and RGB combination.

Visualize change information: You can use the *Visualize change* panel to generate the following change layers:

- **Max change magnitude:** Value of the largest detected change for the specified time period and spectral band, as measured by the difference between the end and start point of adjacent temporal segments.
- **Time of max magnitude of change:** For the given date range and spectral band, visualize the time when the max detected change magnitude occurred.
- **Number of changes:** For the specified time period, display the total number of changes detected.

The image below shows an example of the timing of max magnitude of change for the period 1994-1997 in the SWIR1 band, capturing the extent of the fire scars shown before very clearly.

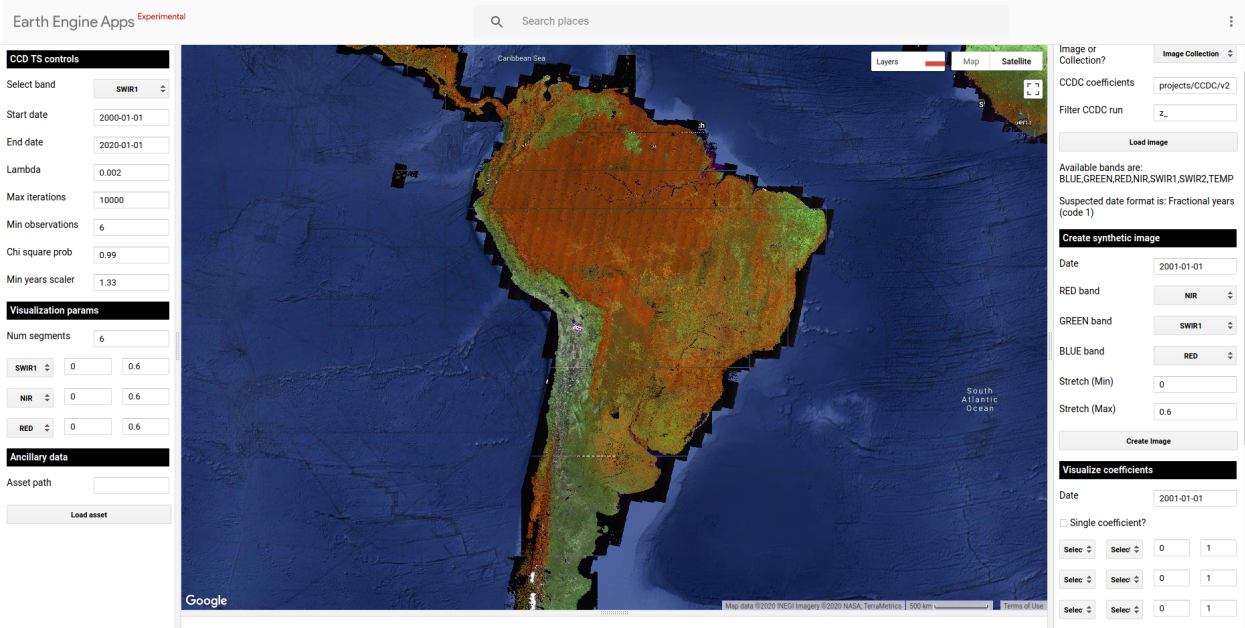


Fig. 5: Example of a predicted (synthetic) image circa 2001-01-01 for South America. The RGB color combination is NIR-SWIR1-RED

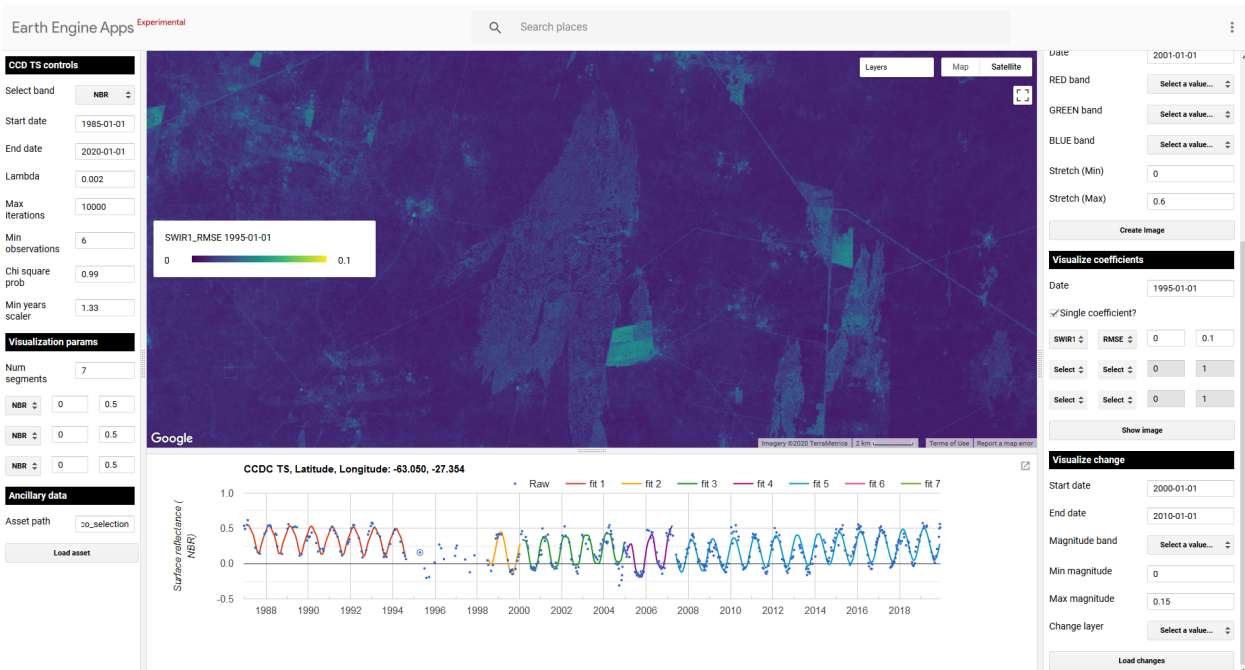


Fig. 6: Example of an image showing the RMSE of the fitted model circa 1995 for a region in Brazil, with its corresponding legend on the left, and the time series of the NBR index and fitted models for a clicked pixel in that general area.

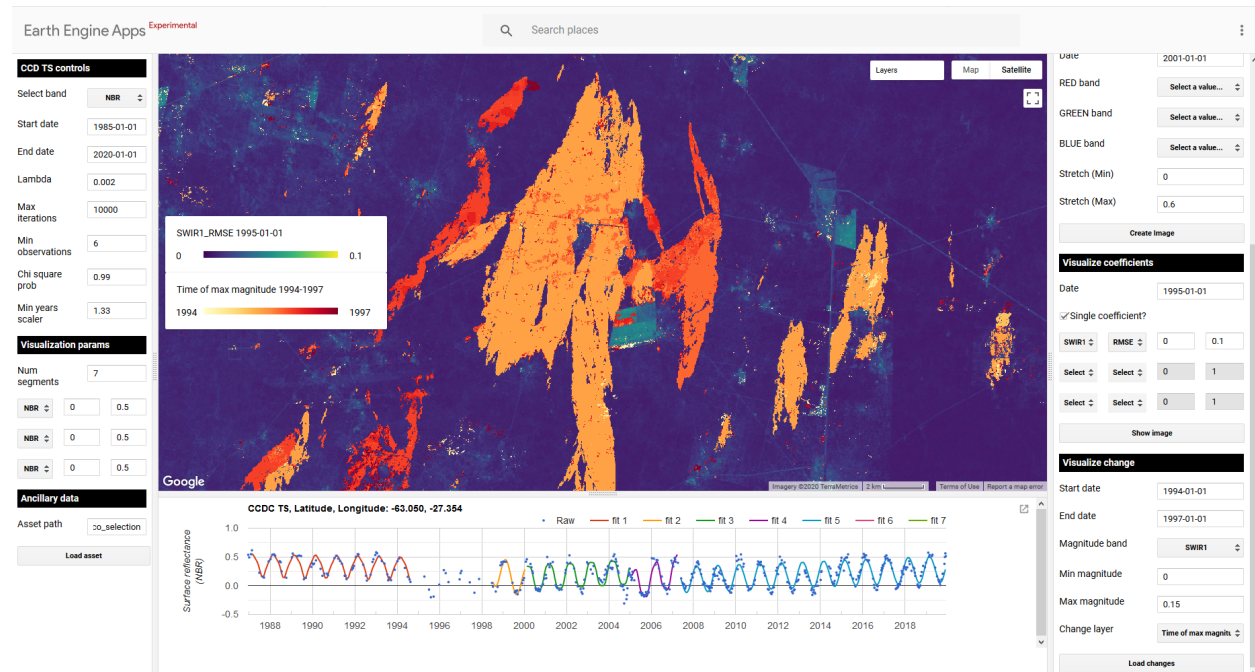


Fig. 7: Map of the timing of max magnitude of change between 1994-1997 for the SWIR1 band, delineating the fire scars in this region of Brazil.

3.2 Land cover tutorial (GUI)

3.2.1 CCDC Classification Interface

By Eric Bullock May 7, 2020

To facilitate easy access to our API we have created a series of graphical user interfaces (GUIs) that require no coding by the user. These guis can be used for calculating CCDC model parameters (i.e. regression coefficients), displaying and interacting with CCDC coefficients and corresponding pixel time series, and classification of the model parameters. This tutorial will demonstrate the gui for creating land cover and land cover change maps.

In this tutorial you will:

- Classify CCDC segments based on their model parameters and ancillary data
- Extract a land cover map for a specific date
- Calculate land cover change between two or more dates

I provide example training data and CCDC coefficients for seven countries in East Africa (Rwanda, Uganda, Ethiopia, Tanzania, Kenya, Zambia, and Malawi).

The first tool that be used in this tutorial can be found [here](#).

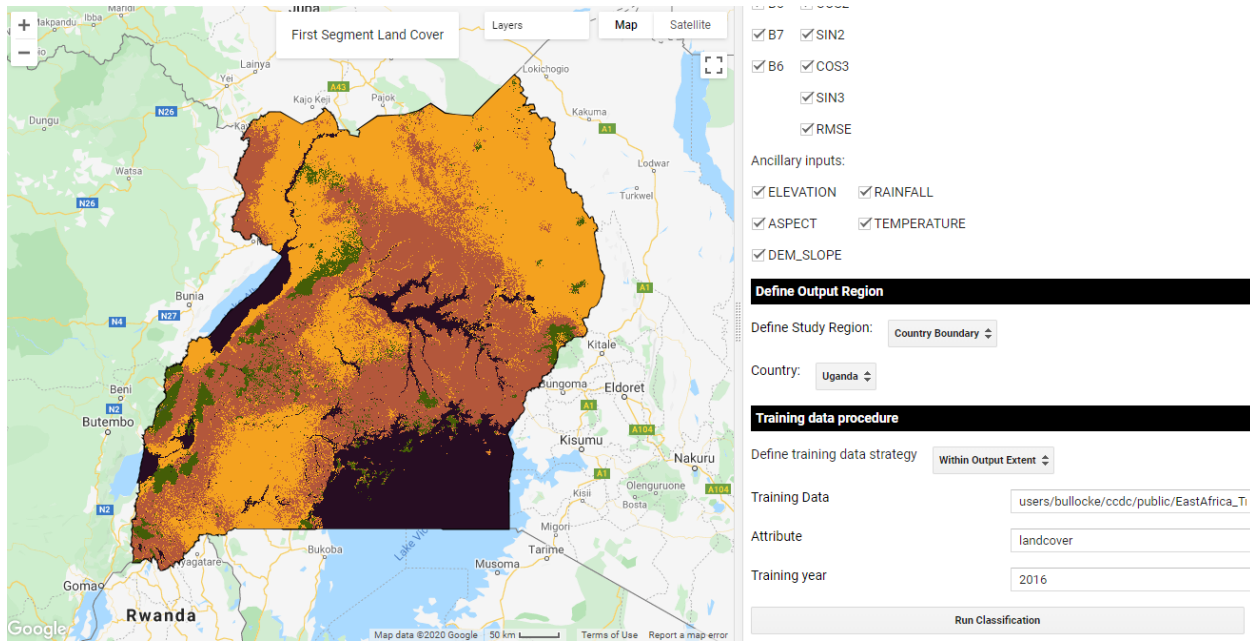


Fig. 8: Classification of CCDC model parameters for Uganda.

3.2.2 1. Classify time series segments

This first step is to classify the time series segments that are the output from the CCDC algorithm. Users should refer to Zhu and Woodcock (2014) for a detailed explanation of the CCDC algorithm. For a refresher, however, the example belows shows the CCDC model fit for the time series of one pixel. On the x-axis is time, and the y-axis is reflectance. Each observation corresponds to a different image at that same location. The parameters of the harmonic regression models, are what we are classifying in this procedure. Thus, we are not classifying Landsat observations, but rather the temporally-continuous model segments. It should be noted that these models are unique for each pixel, and therefore have a unique start and end time depending on land cover or condition change.

This result of part 1 of this tutorial will be an image with bands corresponding to the pixel's nth land cover label for nbands. In other words, band 1 is the first segment's classification, band 2 is the second, and so on. Theoretically, a pixel can have dozens of segments. That is very rare, however, since the changes correspond to land change occurring within that pixel. Thus, to reduce computational intensity we limit the number of segments that are classified in this application to 6 per pixel.

The first step is to load the app, you should see a panel like this appear:

These first few parameters describe the format of the CCDC results. First, are they saved as a single image or a collection? Next is the path to the CCDC results. In this example, we provide an example of an ImageCollection of results with the path 'projects/LCMS/SERVIR_CCDC'. Finally, you must specify the date format that the results were run with. For the example dataset they are in the format of ordinal years (0). Click Load.

Next are the parameters of the machine learning classifier and predictor variables. Uncheck any bands, coefficients, or ancillary data that you do not wish to be used as inputs to classification. The terrain inputs are from the [30m SRTM global DEM](#), while the climate inputs are from the [WorldClim BIO Variables V1](#).

The next option lets you decide how to define the region to classify and export. As you'll see, there are many options. Most of them revolve around a global grid that we have created for the Global Land Cover Mapping and Estimation (GLanCE) project. More information on the GLanCE grid grid can be found on the [project website](#).

There are four ways you can specify a tile to run in addition to manually defining the study region or selecting a country. The simplest option is to choose "Tile Intersecting Point", and then click somewhere on the map. You will

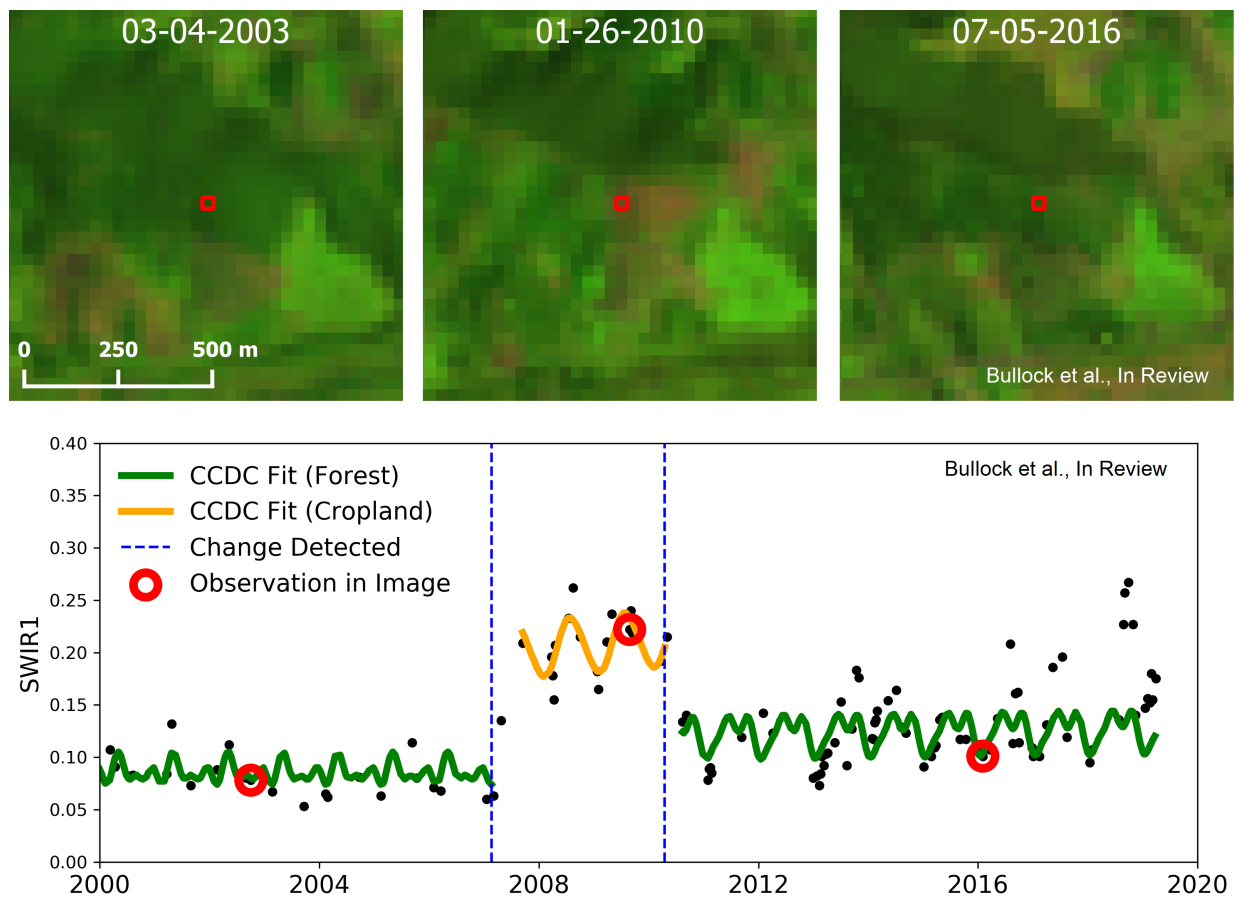


Fig. 9: Example CCDC time series for a pixel that is deforested but later return to secondary forest.

Classify CCDC Segments (BETA)

Load CCDC results

Image or Collection?

Image Collection

CCDC coefficients

projects/LCMS/SERVIR_CC

Date Format?

Fractional Years (1)

Load

Fig. 10: The first step in the app is to load the coefficients.

Classification Parameters

Dates (YYYY-MM-DD);
Comma separated

2001-01-01

Classifier

Select a value...

Predictor Variables

CCDC Model Parameters

☒ B1 ☒ INTP

☒ B2 ☒ SLP

☒ B3 ☒ COS

☒ B4 ☒ SIN

☒ B5 ☒ COS2

☒ B7 ☒ SIN2

☒ B6 ☒ COS3

☒ SIN3

☒ RMSE

Ancillary inputs:

☒ ELEVATION ☒ RAINFALL

☒ ASPECT ☒ TEMPERATURE

☒ DEM_SLOPE

Fig. 11: Next comes parameter specification.

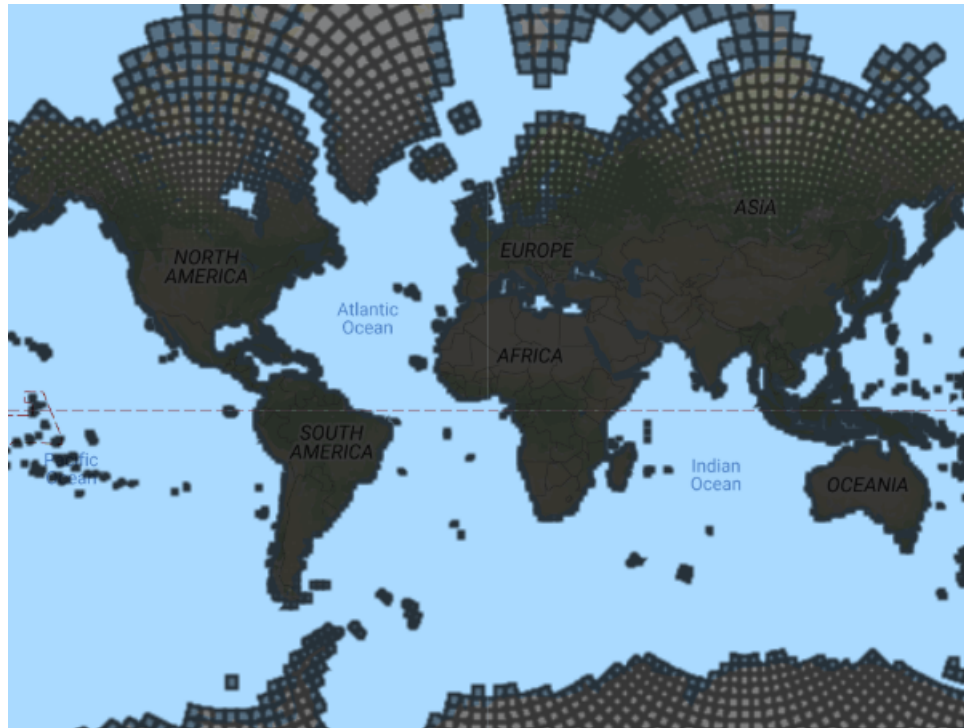


Fig. 12: The global GLanCE grid.

see the grid overlapping the location you selected loaded as the study region.

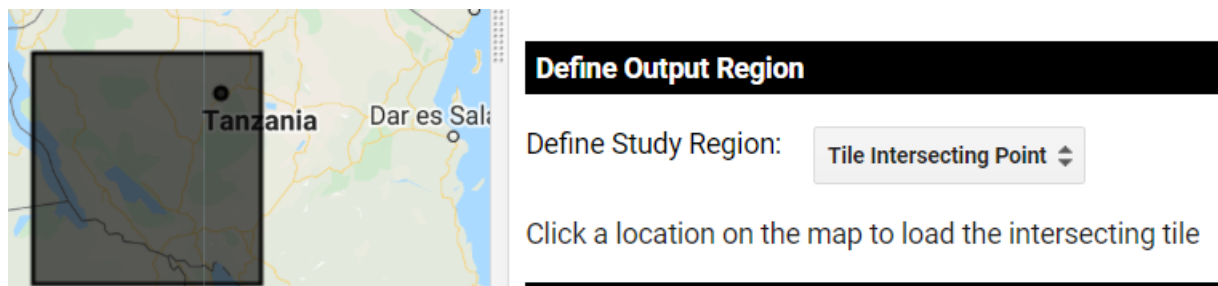


Fig. 13: Define study region from the global grid.

Alternatively, you can manually define the study region by clicking on five points on the map that define the borders.

The other options are to manually define output grids based on their tile IDs, drawing on the map to specify multiple grids, or selecting a country. If multiple grids are selected then each grid will be submitted as a separate task. If a country is selected then the entire country boundary will be the study region.

The final set of parameters relate to the training data. An example training dataset is provided as a FeatureCollection for the seven countries noted above. The training data requires that each point has an attribute identifying the land cover label, and must also correspond to a specific year for training. You have the option to use the entire FeatureCollection or only the points that fall within the study region.

Note, the classification runs quicker if the predictor data for each training point is saved in the feature's properties (as opposed to being calculated on the fly). I recommend doing this process in a separate task, and then using the data with the predictors attached to quickly try classification parameters. You should see in the Console a note about whether or not the predictor data must be sampled as the training points. If so, you can also submit a task that will save this

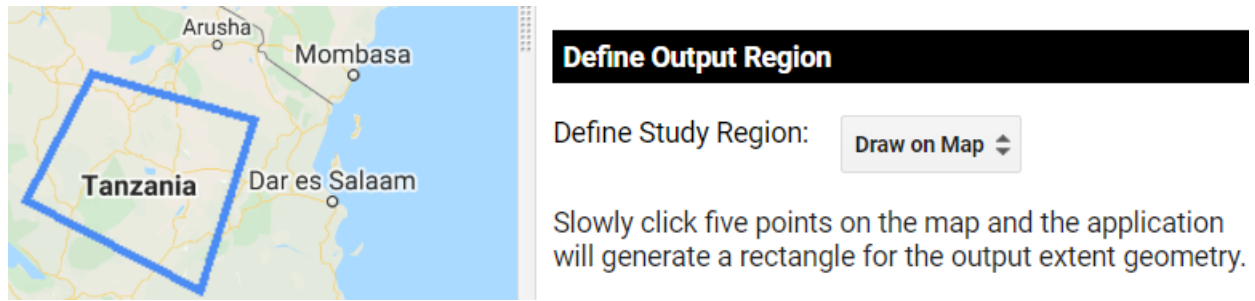


Fig. 14: Manually define study region.

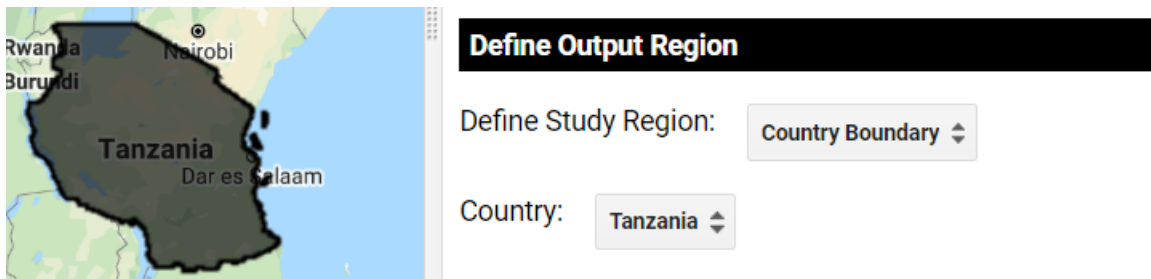


Fig. 15: Or an entire country!

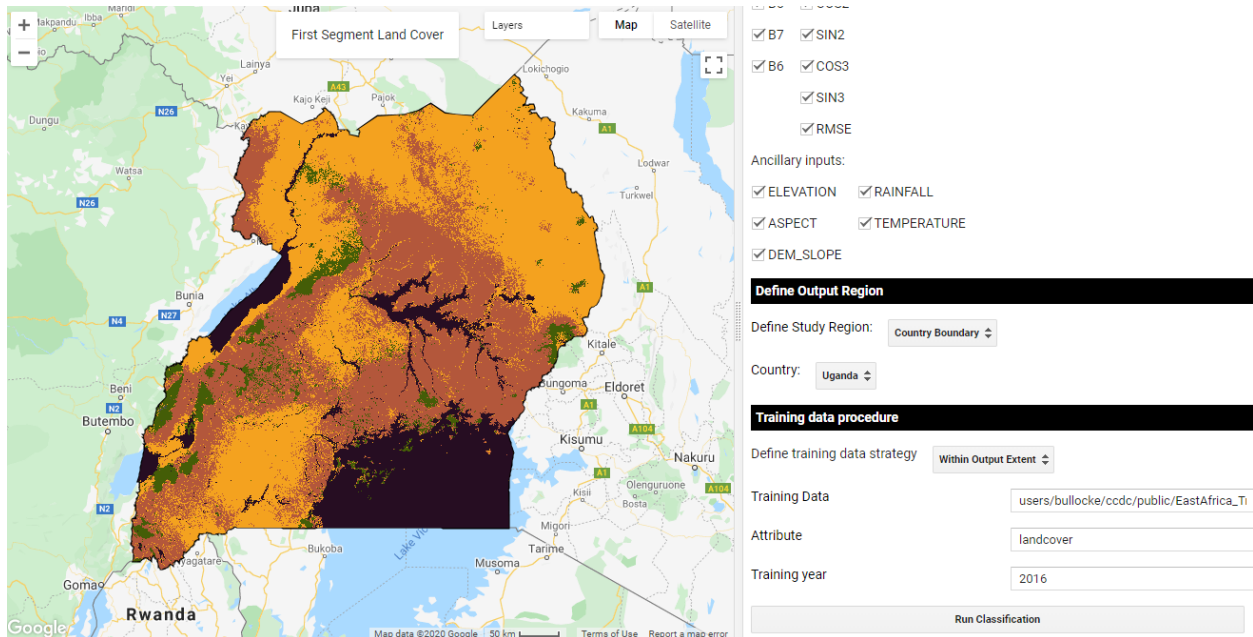


Fig. 16: Classification of the first CCDC segment.

calculation for future use.

Finally, when you click ‘Run Classification’, the classification corresponding to the first segment period gets displayed on the map. In this case, the models correspond to ~1999. The full classification stack can be exported as a task that should appear with the description “classification_segments”.

3.2.3 2. Create Land Cover and Land Cover Change Maps

Once the task has completed processing, we can use it to make landcover maps at any date in time for the study region. This asset can be used directly in the [Landcover Application](#). This application is relatively simple - all you need to do is specify the path to the segment image created above and voilà! The dates should be entered in the format ‘YYYY-MM-DD’ and separated by commas, for example “2001-01-15, 2001-07-21, 2014-12-10”. Each band in the output image will correspond to a different date’s classification.

Fig. 17: Land cover classification for 2001-01-01.

This app also has the function to add a change between that represents conversion from one or multiple classes at a specified date to a specified class or group of classes. You must first specify the starting and ending dates and the land cover class # labels for the corresponding dates. For example, the following examples shows the pixels (red) that

are class 1 (forest) in 2001-01-01, and are either class 2, 3, 4, or 5 in 2014-01-15. In other words, deforestation from January 2001 to June 2016. You can also specify a single value for the Class (To) box, for example just using 3 would map conversion from 1 to 3, or forest to cropland. If these boxes are left empty then just the land cover maps will be created.

Finally, the tool allows you to specify some visualization parameters. This step is very straightforward, just list the land cover names and corresponding numeric value, and optionally provide a palette.



Fig. 18: Land cover in 2001 and deforestation between 2001 and 2014.

By default, an opaque hillshade layer is loaded on top of the classification. I find this helps provide context when viewing landscapes that I am unfamiliar with. However, you can simply turn this off in the layers tab.

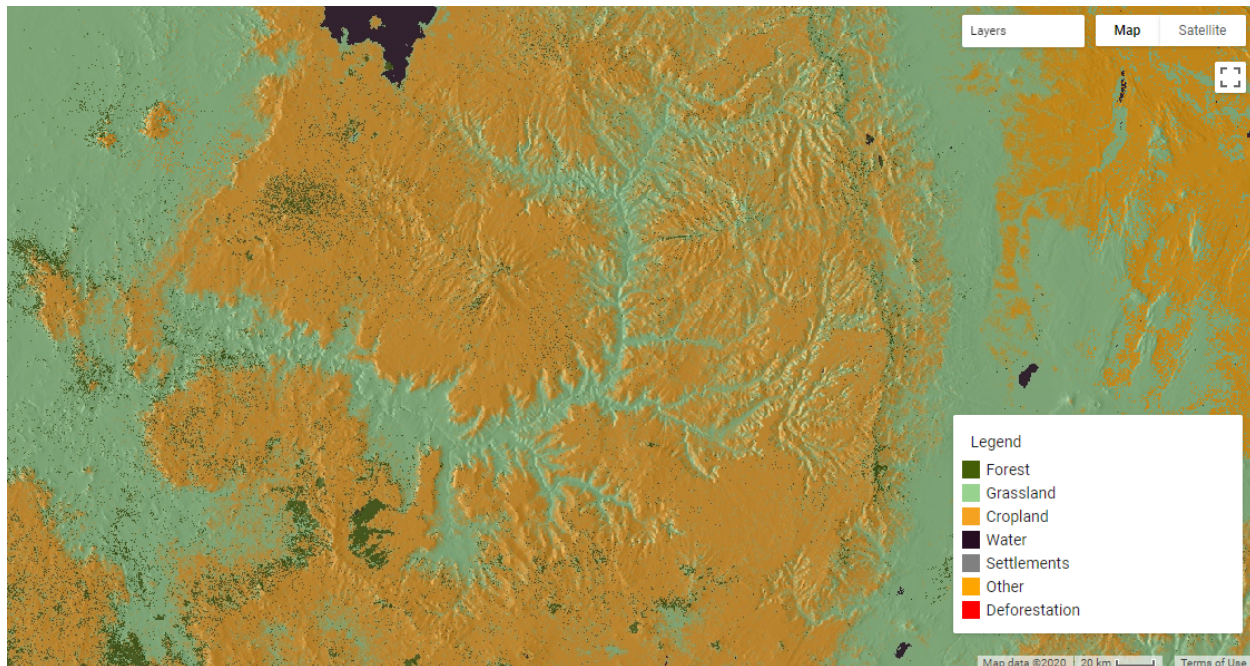


Fig. 19: Classified map with a hillshade overlay.

3.3 Land cover tutorial (API)

3.3.1 Part 1: Submit Change Detection

Continuous Change Detection and Classification (CCDC) is a land change monitoring that was designed to operate on dense time series of Landsat data. Until recently, use of CCDC has been limited to high-performance computing clusters due to the heavy computation requirements. However, thanks to the outstanding engineers at Google and the US Forest Service, it is now available on the Google Earth Engine (GEE). For a detailed description of the methodology please refer to:

Zhu, Z., & Woodcock, C. E. (2014). Continuous change detection and classification of land cover using all available Landsat data. *Remote sensing of Environment*, 144, 152-171.

CCDC, as the name implies, has two major components: change detection and classification. This part of the tutorial will focus on the change detection component. Spectral change is detected on a pixel level by testing for structural breaks on the pixel's time series. In GEE, this process is referred to as 'temporal segmentation', as the pixel-level time series are segmented according to periods of unique reflectance. It does so by fitting harmonic regression models to all spectral bands in the time series. The model-fitting starts at the beginning of the time series, and moves forward in time in an "online" approach to change detection. The coefficients are used to predict future observations, and if the residuals of the future observations exceed a statistical boundary for numerous consecutive observations then a change is detected. After the change, a new regression model is fit and the process continues until the end of the time series.

I am not going to go into all of the details of the algorithm, because they are explained in the manuscript above. If this is all new to you, I encourage you to experiment with our CCDC time series viewer app:

<https://glance.earthengine.app/view/tstools>

Simply navigate to a location (or choose User Location to navigate to your current location), pick a Landsat spectral band, and click on the map to see the CCDC fit to the location you clicked. For example, here is an example of the SWIR1 time series of a location in Rwanda that was forest, converted to cropland, and then converted again to a plantation.

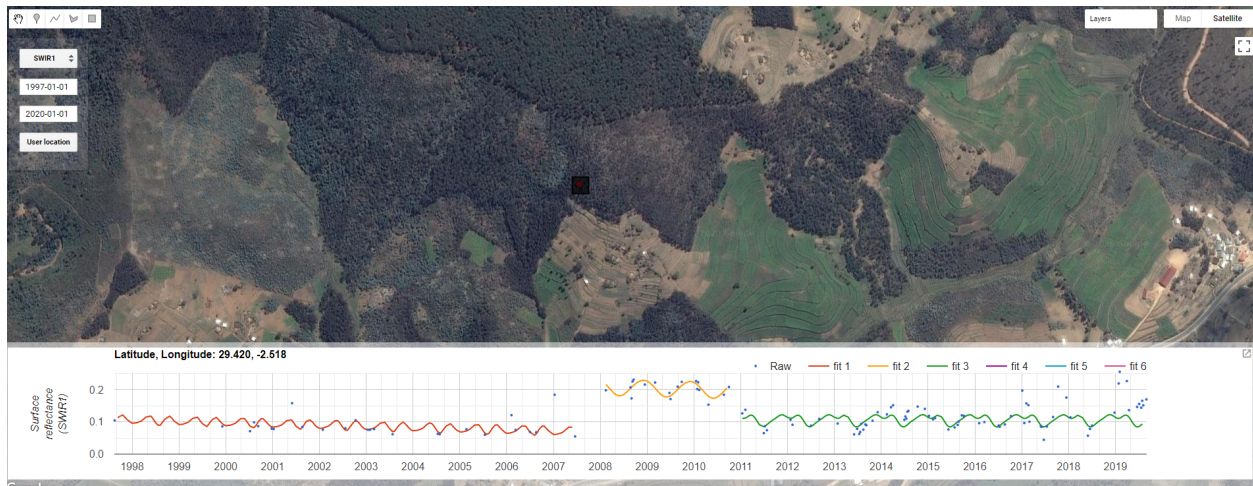


Fig. 20: Figure 1. TSTools Earth Engine App

In that example, there were two changes detected: one in 2007 and one in 2010. At this point there are no classification labels associated with the changes, they are simply structural breaks found by the algorithm. This example is for a single pixel, however, and we wish to perform analysis over all the pixels in a study region. This tutorial will demonstrate how to do just that.

CCDC API

To facilitate processing of CCDC mapping on GEE, we have developed an API that contains functions for generating input data, running CCDC, filtering the results, creating “synthetic” Landsat imagery, and classifying the change detection results. The API functions can be found [here](#), and loaded from the following repository:

```
var utils = require('projects/GLANCE:ccdcUtilities/api')
```

Building and image stack

The CCDC function in GEE can take any `ee.ImageCollection` that has been masked for clouds and cloud shadows. CCDC contains an internal cloud masking algorithm and is rather robust against missed clouds, but the cleaner the data the better. If you wish to build your own Image collection, I refer you to the GEE Examples repo. Alternatively, to obtain an image collection of Landsat 4, 5, 7, and 8 data that is masked using the `cfmask` band you can use the `getLandsat` function in the `Inputs` module of our api. The parameters for building an image collection and running CCDC live within a separate [parameter file](#)

```
// Load example parameter file
var params = require('projects/GLANCE:Tutorial/params.js')

// Filter by date and a location in Brazil
var filteredLandsat = utils.Inputs.getLandsat()
    .filterBounds(params.StudyRegion)
    .filterDate(params.ChangeDetection.start, params.ChangeDetection.end)

print(filteredLandsat.size())
```

The console should show that there are around images in the collection. It should be noted that CCDC uses all available Landsat data, even if part of the image is cloudy! That is because there can be many usable, cloud-free pixels even if a majority of the image is cloudy. Since CCDC operates on the pixel time series, those observations are still usable.

Now, we can use this Image Collection into the `ee.Algorithms.TemporalSegmentation.Ccdc` algorithm and retrieve a multi-dimensional array containing model coefficients, model RMSE, and change information for every detected segment. That means that the dimensions for one pixel can be different than another, depending on the number of model breaks. Documentation on the CCDC parameters are in the GEE Docs, so I will not elaborate on them here.

```
// First define parameters
var changeParams = {
  collection: filteredLandsat,
  breakpointBands: params.ChangeDetection.breakpointBands,
  tmaskBands: params.ChangeDetection.tmaskBands,
  minObservations: params.ChangeDetection.minObservations,
  chiSquareProbability: params.ChangeDetection.chiSquareProbability,
  minNumOfYearsScaler: params.ChangeDetection.minNumOfYearsScaler,
  dateFormat: params.ChangeDetection.dateFormat,
  lambda: params.ChangeDetection.lambda,
  maxIterations: params.ChangeDetection.maxIterations
}

var results = ee.Algorithms.TemporalSegmentation.Ccdc(changeParams)
print(results)
```

And like that, you have run the change detection component of CCDC! A quick note on the output bands:

- `tStart`: The start date of each model segment.
- `tEnd`: The end date of each model segment.
- `tBreak`: The model break date if a change is detected.
- `numObs`: The number of observations used in each model segment.
- `changeProb`: A numeric value representing the multi-band change probability.
- `*_coefs`: The regression coefficients for each of the bands in the image collection.
- `*_rmse`: The model root-mean-square error for each segment and input band.
- `*_magnitude`: For segments with changes detected, this represents the normalized residuals during the change period.

The array can now be saved as an array image. In my experience, array images require the ‘pyramidingPolicy’ to be ‘sample’.

The next part of the tutorial we will go through the process of formatting training data to be used in classification.

3.3.2 Part 2: Prepare training data

The second step in performing land cover analysis using CCDC is collecting training data. There are a few requirements for the training data for it to work with the rest of the process. This tutorial demonstrates how to ensure your data meets these requirements.

Training data requirements:

- The data must be an Earth Engine FeatureCollection of point geometries.
- An attribute in each point must contain a numeric value indicating the associated land cover.
- An attribute in each point must contain a year that corresponds to the land cover label.

Optional additional steps

- The predictor data as attributes for each point and for the year that corresponds to the land cover label.
- Unique sample IDs as attributes

Importing your training data as an Earth Engine Asset

I am not going to go into details, as that has been documented in depth on [Google's Developer Page](#). However, it's worth making sure your data is the correct format. You can do that by printing out the first feature.

```
// Load example parameter file
var params = require('projects/GLANCE:Tutorial/params.js')

var trainingData = ee.FeatureCollection(params.Classification.trainingPath)
print(trainingData.first())
```

In the console, you should see information on the first training point. Select the feature and then select 'geometry'. Make sure the 'type' is 'Point', like in the figure below. If it is anything else (such as Rectangle or Polygon) then you must convert your data to points before continuing.

Creating a numeric land cover attribute

The land cover label must be numeric, so it cannot be a string (such as "forest") or a numeric string (or a number that is written in string format). To check the type of your attribute select the 'JSON' button on the right side of the console to expand the json representation of the feature. The button is circled in red below:

```
▼ Feature (Point, 3 properties)
  type: Feature
  ▸ geometry: Point (-161.97, 61.57)
  ▼ properties: Object (3 properties)
    lc_string: forest
    numeric: 1
    numeric_string: 1
```

JSON

Fig. 21: img1

You should see the JSON of the feature appear:

Note that my feature has three attributes: lc_string, numeric, and numeric_string. The lc_string attribute will not work because the classifiers require numeric class property. In the first picture, where the output is formatted, the numeric and numeric_string both look like they'd work. But when viewing the JSON representation, it can be seen that the numeric_string still has quotations, and thus is still formatted as a string. Therefore, only the numeric attribute would work for this tutorial.

```

{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      -161.97499523948434,
      61.57136990530492
    ]
  },
  "properties": {
    "lc_string": "forest",
    "numeric": 1,
    "numeric_string": "1"
  }
}

```

Formatted

Fig. 22: img2

If you do not have a numeric attribute indicating land cover you can convert a string attribute to numeric using the 'remap' Earth Engine (EE) function. The code to do so is rather straightforward, but we have a helper function in our API. For this example, I'll demonstrate how it's done using the EE function and our CCDC API, and hereforth I'll mostly be relying on the API.

In pure EE code, and using the class attribute name in the above example and assuming our only classes are 'forest', 'agriculture', and 'water', you can convert them to numeric values in a new attribute 'landcover' with the following code:

```

trainingData = trainingData.map(function(feat) {
  return feat.set('landcover', feat.get('lc_string'))
  .remap(['forest', 'agriculture', 'water'], [1, 2, 3], 'landcover')
})

```

This can also be done using the 'remapLC' function in our API. See the [API documentation](#) for a full description of available functions.

```

// First load the API file
var utils = require('projects/GLANCE:ccdcUtilities/api')

trainingData = utils.Classification.remapLC(
  fakeFC, 'lc_string', 'landcover', ['forest', 'agriculture', 'water'], [1, 2, 3])

print('First training point: ', trainingData.first())

```

Note that there should now be an attribute called 'landcover' that is numeric.

Add a year attribute

In order to determine the CCDC coefficients to use as predictors for training the classifier, each point needs to have a year attribute to query the CCDC results by. For example, if you know a training point corresponds to forest in 2014, then the CCDC coefficients for 2014 will be used for training. If all of your training data corresponds a certain year, you can add the attribute with a simple bit of code. In this example the training data corresponds to 2014 and it is assigned to an attribute named 'year'.

```

trainingData = trainingData.map(function(feat) {
  return feat.set('year', 2014)}
)

```


Get predictor data for each training point

We need to extract predictor data for each training point before we can apply a classifier. We can do this either manually right before submitting the classification, or we can extract the predictors in advance and store them as properties of each of the training points. For this tutorial we will use the second way, because it enables a faster classification that will load “on the fly”. This is useful for testing classification parameters. The function to do this is called ‘getTrainingCoefsAtDate’. First, however, we need to construct the CCDC coefficient image to sample from.

```
// Define bands to use in classification
var bands = params.Classification.bandNames

// Define coefficients to use in classification
var coefs = params.Classification.coefs

// Segment ids
var segs = params.Classification.segs

// Property corresponding to year of training data
var yearProperty = params.Classification.yearProperty

// CCDC change detection results from the first part of this tutorial.
var coefImage = ee.ImageCollection(params.Classification.changeResults)
  .filterBounds(params.StudyRegion).mosaic()

// Load ccd image stack with coefficients and change information

var ccdImage = utils.Classification.loadResults(
  params.Classification.resultFormat,
  params.Classification.changeResults,
  params.StudyRegion)

print('CCD Image:', ccdImage)

// Finally, get ancillary topographic and climate data
var ancillary = utils.Inputs.getAncillary()
```

Now that we have the CCDC image we can calculate the predictor data for each point, filter features that return no data, and export the results as an asset.

```
var trainingData = utils.Classification.getTrainingCoefsAtDate(
  trainingData, coefs, bands, yearProperty, ancillary, ccdImage, segs)

// Filter points with no data
var testBand = params.Classification.bandNames[0] + '_' + params.Classification.
  ↪coefs[0]
trainingData = trainingData.filter(ee.Filter.notNull([testBand]))

print('First training point with predictors:', trainingData.first())

Export.table.toAsset({
  collection: trainingData,
  description: 'trainingDataProcessed',
  assetId: params.Classification.trainingPathPredictors})
```

You should now see in the feature attributes all of the predictor data that can be used for classification.

Add unique IDs as attributes

Another optional, but recommended, step is assigning each sample with a unique ID as an attribute. EE gives each point an ID, but they can be long and seemingly random. The ‘assignIDs’ function in our API will shuffle the sample and assign a unique ID to a given attribute name.

```
trainingData = utils.Classification.assignIds(trainingData, 'ID')
```

3.3.3 Part 3: Classify time series segments

The third step in performing land cover classification using CCDC is to use the training data from step 2 in a machine learning classifier to classify each CCDC *segment*. Note that each pixel can have a different number of segments depending on the number of changes detected. That is why the coefficients for CCDC are stored in n-dimensional arrays, because each pixel can have a different number of dimensions depending on the changes detected. This means that the process is *slightly* more complicated than a simple supervised classification, but this tutorial will go through it all.

A code example using GLANCE data and parameters can be found here:

<https://code.earthengine.google.com/?scriptPath=projects%2FGLANCE%3ATutorial%2FPart%202.%20Classification>

Classification requirements:

- The training data must be specified in the format described in Part 2 of this tutorial.
- A machine learning classifier.

Converting the CCDC coefficient data to an image that can be classified

The API functions ‘loadResults’ in the Classification module and ‘getAncillary’ in the Inputs module can be used to create the CCDC stack with ancillary data to classify.

```
// First load the API file
var utils = require('projects/GLANCE:ccdcUtilities/api')

// Define a couple parameters
var bandNames = ["BLUE", "GREEN", "RED", "NIR", "SWIR1", "SWIR2", "TEMP"]
var inputFeatures = ["INTP", "SLP", "PHASE", "AMPLITUDE", "RMSE"]
var ancillaryFeatures = ["ELEVATION", "ASPECT", "DEM_SLOPE", "RAINFALL", "TEMPERATURE"]
var numberOfSegments = 6
var classProperty = 'landcover'
var trainProp = .2
var seed = Math.ceil(Math.random() * 1000)
var studyArea = ee.Geometry.Polygon(
  [[[-65.11727581440459, -8.755437491733284],
    [-65.11727581440459, -13.240578578777912],
    [-59.470303158154586, -13.240578578777912],
    [-59.470303158154586, -8.755437491733284]]], null, false);
var trainingDataPath = 'PATH/TO/YOUR/TRAINING/DATA'
var classifier = ee.Classifier.smileRandomForest({
  numberOfTrees: 150,
  variablesPerSplit: null,
  minLeafPopulation: 1,
```

(continues on next page)

(continued from previous page)

```

    bagFraction: 0.5,
    maxNodes: null
  })

  // Obtain the CCDC change detection array
  var ccdcArray = 'PATH/TO/YOUR/CCDC/ARRAY'

  // Next, turn array image into image
  var imageToClassify = utils.CCDC.buildCcdImage(ccdcArray, numberOfSegments, bandNames)

  // Now get ancillary data
  var demImage = ee.Image('USGS/SRTMGL1_003').rename('ELEVATION')
  var slope = ee.Terrain.slope(demImage).rename('DEM_SLOPE')
  var aspect = ee.Terrain.aspect(demImage).rename('ASPECT')
  var bio = ee.Image('WORLDCLIM/V1/BIO')
    .select(['bio01', 'bio12'])
    .rename(['TEMPERATURE', 'RAINFALL'])
  var ancillary = ee.Image.cat([demImage, slope, aspect, bio])

```

Next, we can actually do the classification! We’ve already defined the parameters above, so we can then use the ‘classifySegments’ function to classify the CCDC segments.

```

// Now do the actual classification add the first segments classification to the map

// Get training data as FC
var trainingData = ee.FeatureCollection(trainingDataPath)

// Optionally filter by study area
trainingData = trainingData.filterBounds(studyArea)

var results = utils.Classification.classifySegments(
  imageToClassify, numberOfSegments, bandNames, ancillary, ancillaryFeatures,
  trainingData, classifier, studyArea, classProperty, inputFeatures)
  .clip(studyArea)

// Get a legend and visualization parameters from the api.
var viz = utils.Results.viz
var legend = utils.Results.legend

Map.addLayer(results.select(0), viz, 'Seg1 Classification')
Map.add(legend)

```

And just like that, we can get a classified land cover map! The layer added represents the first segment land cover.

3.3.4 Part 4: Land Cover Mapping

The output of part 3 was a stack of land cover classifications organized by CCDC models. Each pixel contains different model start and end times, so the land cover label for each band corresponds to different time periods for each pixel. Well that’s not very helpful, is it?

Part 4 of this tutorial demonstrates how to go from the classification “stack” to a map of land cover at a specific year, or change between years.



Fig. 23: img1

Mapping Requirements

- A classified ‘stack’ of as demonstrated in Part 3 of this tutorial

To go from a classified image stack to a classification at a date is relatively straightforward. To get a land cover classification at a specific date we can use the ‘getLcAtDate’ function in our API.

```
var utils = require('projects/GLANCE:ccdcUtilities/api')
var classificationStack = '/PATH/TO/IMAGE/STACK'
var dateOfClassification = '2014-03-27'
var matchingDate = classUtils.getLcAtDate(classificationStack,
  dateOfClassification)
```

This can easily be extended to map change between two dates. In this example we calculate the post-deforestation land cover between 2000 and 2018

```
var class2000 = utils.Classification.getLcAtDate(classificationStack,
  '2000-01-01')

var class2018 = utils.Classification.getLcAtDate(classificationStack,
  '2018-01-01')

var deforestation = class2000.eq(5)
  .and(class2018.neq(5))

Map.addLayer(deforestation.selfMask(),
  {palette: 'red'},
  'Deforestation')

var postDefClass = class2018.updateMask(deforestation)

var viz = utils.Results.viz

Map.addLayer(postDefClass,
  viz,
  'Post-Deforestation Class')
```

Note that the post-disturbance land cover is almost entirely from the ‘Herbaceous’ class.

In the above example, the Forest class corresponds to the number 5. This process can be repeated to map any type of land cover change for the classes in your legend. For example, the following example shows expansion of river water west of Porto Velho (Cyan are pixels that were converted from non-water to water)..

```
var regrowth = class2000.neq(5).and(class2000.neq(0))
  .and(class2018.eq(5))

Map.addLayer(regrowth.selfMask(),
  {palette: 'cyan'},
  'Regrowth')
```

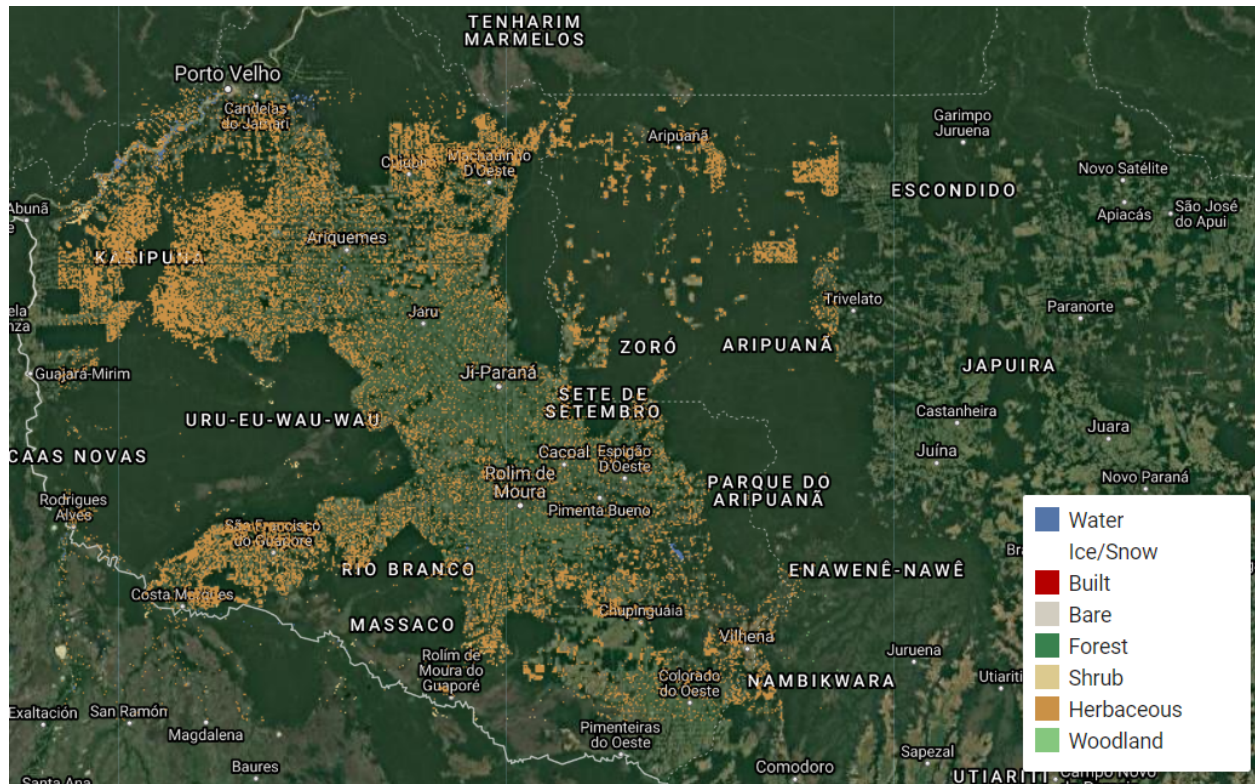



Fig. 24: img1

3.4 Obtaining coefficients, changes and synthetic images (API)

3.4.1 Access the API and load CCDC results

This tutorial requires access to the API. It also requires access to any results obtained from running the CCDC algorithm on Google Earth Engine (GEE). For this tutorial we can use the results created by the GEE team for the entire globe. As of May 27, 2020, these results include multiple runs with varying inputs and configurations, and therefore we need to filter the results for a specific model run before we can process them. A model run we have been using so far is the 'z_' model, but others include the BRDF-corrected 'brdf'. Until the GEE team publishes an official public dataset, we can use the 'z_' or 'brdf' for testing purposes. Finally, if we want to look at the results for a specific location we can filter them using the built-in `filterBounds` function, or remove it if we want to see the results for the current map view extent.

```
// First load the API file
var utils = require('projects/GLANCE:ccdcUtilities/api')

// Load the global results computed by the GEE team
var ccdcCol = ee.ImageCollection('projects/CCDC/v2')
var ccdc = ccdcCol.filterMetadata('system:index', 'starts_with', 'z_')
                  .filterBounds(geometry)
                  .mosaic()
```

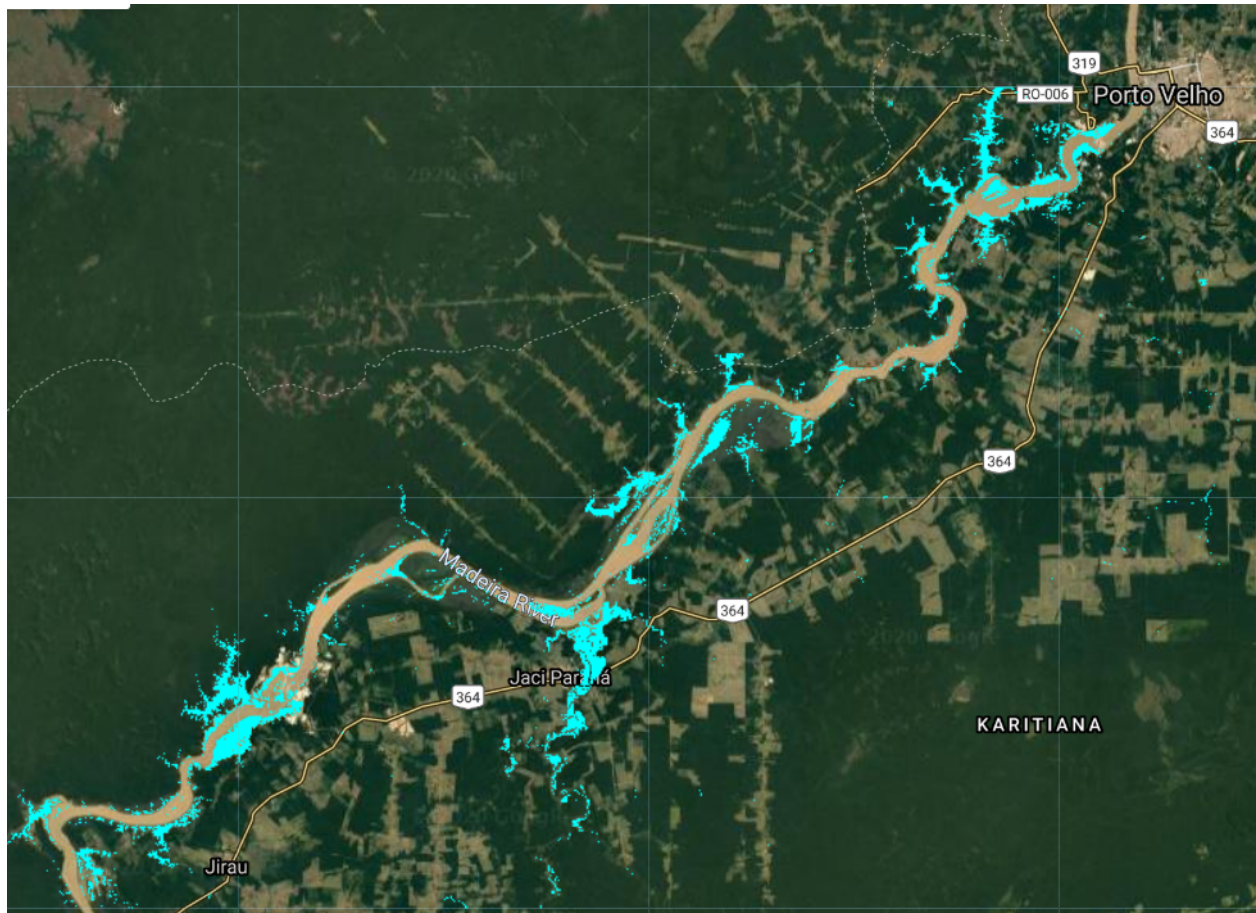


Fig. 25: New Water

3.4.2 Obtain CCDC coefficients and synthetic images

The CCDC module provides functions that facilitate obtaining any coefficient for any point in the time within the range of the results. In the case of the global results generated by the GEE team, the time period corresponds to 1999 to 2019. Since the CCDC algorithm can be run in three different time formats, it is important to know which format was used to encode the results. The results used here were computed using fractional years, therefore we need to convert the date we want to obtain coefficients for to that format. This can be done using the `Dates` module:

Get date in the right format

```
var inputDate = '2001-12-30'
var dateParams = {inputFormat: 3, inputDate: inputDate, outputFormat: 1}
var formattedDate = utils.Dates.convertDate(dateParams)
```

In the example above, we convert the input date into fractional year, corresponding to the `outputFormat` 1. Other output formats are: 0 for Julian days, and 2 for Unix time.

Obtain CCDC results in 'regular' image format

The CCDC outputs are stored as array images to facilitate storing the variable-length arrays that are computed by the algorithm, as it is not known in advance how many temporal segments will be obtained for each pixel. However, operating on those arrays and displaying them tends to be slower than using a regular `ee.Image()`. For this reason, we convert the array image results into a regular image using the `utils.CCDC.buildCcdImage` function. The function expects the CCDC results, the number of segments we want to extract, and the names of the spectral bands.

```
// Spectral band names. This list contains all possible bands in this dataset
var BANDS = ['BLUE', 'GREEN', 'RED', 'NIR', 'SWIR1', 'SWIR2', 'TEMP']

// Names of the temporal segments
var SEGS = ["S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9", "S10"]

// Obtain CCDC results in 'regular' ee.Image format
var ccdImage = utils.CCDC.buildCcdImage(ccdc, SEGS.length, BANDS)
```

Get coefficients

The resulting image will contain the `ee.Image()` version of the results with the number of coefficients specified. A lower number of segments requested will speed up subsequent processing, but may result in some missing segments for highly dynamic areas, such as agricultural lands in California. For several locations, 10 segments seems to be a good compromise. With this image, we can request any set of bands and coefficients for a the date we selected above. You can read the API documentation to specify the other parameters of the `get_multi_coefs` function.

```
// Define bands to select.
var SELECT_BANDS = ['RED', 'NIR']

// Define coefficients to select. This list contains all possible segments
var SELECT_COEFS = ["INTP", "SLP", "COS", "SIN", "COS2", "SIN2", "COS3", "SIN3", "RMSE", ""]

// Obtain coefficients
var coefs = utils.CCDC.getMultiCoefs(ccdImage, formattedDate, SELECT_BANDS, SELECT_COEFS, true, SEGS, 'after')
```


Compute synthetic image

The regression models can be used to calculate the surface reflectance of any of the bands for any point in time within the data time range (i.e. 1999 to 2019 in our case). This image is called a synthetic image, and it is computed with the `getMultiSynthetic` function.

```
// Bands to get surface reflectance for
var SUB_BANDS = ['RED', 'NIR', 'SWIR1', 'SWIR2']

// Obtain synthetic image
var synt = utils.CCDC.getMultiSynthetic(ccdImage, formattedDate, 1, BANDS, SEGS)
```

Get change information

Finally, to obtain change information we can use the `filterMag` function. The function expects the CCDC results in the regular image format, start and end dates in the correct date format, the spectral band for which to get the information, and the list of segments defined previously.

```
var changeStart = '2001-01-01'
var changeEnd = '2018-12-31'
var startParams = {inputFormat: 3, inputDate: changeStart, outputFormat: 1}
var endParams = {inputFormat: 3, inputDate: changeEnd, outputFormat: 1}
var formattedStart = utils.Dates.convertDate(startParams)
var formattedEnd = utils.Dates.convertDate(endParams)

var filteredChanges = utils.CCDC.filterMag(ccdImage, formattedStart, formattedEnd,
  ↪ 'SWIR1', SEGS)
```

The image `filteredChanges` contains three bands:

1. 'MAG': Represents the magnitude of the largest change for the specified time range and band.
2. 'tBreak': Represents the date when the change with the largest magnitude occurred.
3. 'numTbreak': Represents the total number of changes in the specified time period.

API REFERENCE

4.1 API REFERENCE

4.1.1 CCDC

buildSegmentTag (*nSegments*)

Create sequence of segment strings

Arguments

- **nSegments** (*Integer*) – Number of segments to create labels for

Returns {*ee.List*} List of segment names (e.g. S1, S2)

buildBandTag (*tag*, *bandList*)

Create sequence of band names for a given string tag

Arguments

- **tag** (*string*) – String tag to use (e.g. 'RMSE')
- **bandList** (*array*) – List of band names to combine with tag

Returns {*ee.List*} List of band names combined with tag name

buildMagnitude (*fit*, *nSegments*, *bandList*)

Extract CCDC magnitude image from current CCDC result format

Arguments

- **fit** (*ee.Image*) – Image with CCD results
- **nSegments** (*number*) – Number of segments to extract
- **bandList** (*array*) – Client-side list with band names to use

Returns {*ee.Image*} Image with magnitude of change per segment per band

buildRMSE (*fit*, *nSegments*, *bandList*)

Extract CCDC RMSE image from current CCDC formatted results

Arguments

- **fit** (*ee.Image*) – Image with CCDC results
- **nSegments** (*number*) – Number of segments to extract
- **bandList** (*array*) – Client-side list with band names to use

Returns {*ee.Image*} Image with RMSE of each segment per band

buildCoefs (*fit, nSegments, bandList*)

Extract CCDC Coefficients from current CCDC formatted result

Arguments

- **fit** (*ee.Image*) – Image with CCD results
- **nSegments** (*number*) – Number of segments to extract
- **bandList** (*array*) – Client-side list with band names to use

Returns {*ee.Image*} Image with coefficients per band

buildStartEndBreakProb (*fit, nSegments, tag*)

Extract data for CCDC 1D-array, non-spectral bands (tStart, tEnd, tBreak, changeProb or numObs)

Arguments

- **fit** (*ee.Image*) – Image with CCD results
- **nSegments** (*integer*) – Number of segments to extract
- **tag** (*string*) – Client-side string to use as name in the output bands

Returns {*ee.Image*} Image with values for tStart, tEnd, tBreak, changeProb or numObs

buildCcdImage (*fit, nSegments, bandList*)

Transform ccd results from array image to “long” multiband format

Arguments

- **fit** (*ee.Image*) – Image with CCD results
- **nSegments** (*number*) – Number of segments to extract
- **bandList** (*array*) – Client-side list with band names to use

Returns {*ee.Image*} Image with all results from CCD in ‘long’ image format

getSyntheticForYear (*image, date, dateFormat, band*)

Create synthetic image for specified band

Arguments

- **image** (*ee.Image*) – Image with CCD results in long multi-band format
- **date** (*number*) – Date to extract the segments for, in the format that ccd was run in
- **dateFormat** (*number*) – Code of the date format that ccdc was run in (e.g. 1 for frac years)
- **band** (*string*) – Band name to use for creation of synthetic image

Returns {*ee.Image*} Synthetic image for the given date and band

getMultiSynthetic (*image, date, dateFormat, band*)

Create synthetic image for a list of bands

Arguments

- **image** (*ee.Image*) – Image with CCD results in long multi-band format
- **date** (*number*) – Date to extract the segments for, in the format that ccd was run in
- **dateFormat** (*number*) – Code of the date format that ccdc was run in (e.g. 1 for frac years)
- **band** (*array*) – List of bands to get synthetic data for

Returns {ee.Image} Synthetic image for the given date and bands

fillNoData (*fit*, *nCoefs*, *nBands*, *clipGeom*)

Note: Deprecated: (No longer necessary)

Replace nodata in CCD output and fill with zeros Assumes current CCDC result format

Arguments

- **fit** (*ee.Image*) – Image with CCD results
- **nCoefs** (*number*) – Number of coefficients present in the results
- **nBands** (*number*) – Number of spectral bands used to produce the results
- **clipGeom** (*ee.Geometry*) – Geometry of the image that is being masked

Returns {ee.Image} Image with nodata areas replaced with zeros

dateToDays (*strDate*)

Note: Deprecated: (No longer necessary)

Return a date as days from 01-01-0000

Arguments

- **strDate** (*String*) – Date in the format accepted by ee.Date

Returns ee.Number – Date expressed as days since 01-01-0000

filterCoefs (*ccdResults*, *date*, *band*, *coef*, *behavior*)

Filter coefficients for a given date using a mask

Arguments

- **ccdResults** (*ee.Image*) – CCD results in long multi-band format
- **date** (*string*) – Date in the same format that CCD was run with
- **band** (*string*) – Band to select.
- **coef** (*string*) – Coef to select. Options are “INTP”, “SLP”, “COS”, “SIN”, “COS2”, “SIN2”, “COS3”, “SIN3”, “RMSE”, “MAG”
- **behavior** (*String*) – Method to find intersecting (‘normal’) or closest segment to given date (‘before’ or ‘after’) if no segment intersects directly

Returns ee.Image – Single band image with the values for the selected band/coefficient

normalizeIntercept (*intercept*, *start*, *end*, *slope*)

Normalize the intercept to the middle of the segment time period, instead of the 0 time period.

Arguments

- **intercept** (*ee.Image*) – Image band representing model intercept
- **start** (*ee.Image*) – Image band representing model slope date
- **end** (*ee.Image*) – Image band representing model end date

- **slope** (*ee.Image*) – Image band representing model slope

Returns *ee.Image* – Image band representing normalized intercept.

getCoef (*ccd, date, bandList, coef, behavior*)

Get image of with a single coefficient for all bands

Arguments

- **ccd** (*ee.Image*) – results CCD results in long multi-band format
- **date** (*string*) – Date in the same format that CCD was run with
- **bandList** (*array*) – List of all bands to include.
- **coef** (*array*) – Coef to select. Options are “INTP”, “SLP”, “COS”, “SIN”, “COS2”, “SIN2”, “COS3”, “SIN3”, “RMSE”, “MAG”
- **behavior** (*string*) – Method to find intersecting (‘normal’) or closest segment to given date (‘before’ or ‘after’) if no segment intersects directly

Returns *ee.Image* – coefs Image with the values for the selected bands x coefficient

applyNorm (*bandCoefs, segStart, segEnd*)

Apply normalization to intercepts

Arguments

- **bandCoefs** (*ee.Image*) – Band x coefficients image. Must include slopes
- **segStart** (*ee.Image*) – Image with dates representing the start of the segment
- **segEnd** (*ee.Image*) – Image with dates representing the end of the segment

Returns *ee.Image* – bandCoefs Updated input image with normalized intercepts

getMultiCoefs (*ccd, date, bandList, coef_list, cond, segNames, behavior*)

Get image of with bands x coefficients given in a list

Arguments

- **ccd** (*ee.Image*) – results CCD results in long multi-band format
- **date** (*string*) – Date in the same format that CCD was run with
- **bandList** (*array*) – List of all bands to include. Options are “B1”, “B2”, “B3”, “B4”, “B5”, “B6”, “B7”
- **coef_list** (*list*) – List of coefs to select. Options are “INTP”, “SLP”, “COS”, “SIN”, “COS2”, “SIN2”, “COS3”, “SIN3”, “RMSE”, “MAG”
- **cond** (*boolean*) – Normalize intercepts? If true, requires “INTP” and “SLP” to be selected in *coef_list*.
- **segNames** (*ee.List*) – List of segment names to use.
- **behavior** (*string*) – Method to find intersecting (‘normal’) or closest segment to given date (‘before’ or ‘after’) if no segment intersects directly

Returns *ee.Image* – coefs Image with the values for the selected bands x coefficients

getChanges (*ccd, startDate, endDate, segNames*)

Filter segments with change in a given range

Arguments

- **ccd** (*ee.Image*) – results CCD results in long multi-band format

- **startDate** (*Number*) – Start date in the format that was used to run CCD
- **endDate** (*Number*) – End date in the format that was used to run CCD
- **segNames** (*ee.List*) – List of segment names matching the number of segments in the bands

Returns **ee.Image** – Mask image indicating which pixel/segments have changes in the specified time range.

filterMag (*ccd, startDate, endDate, band, segNames*)

Obtain change with largest magnitude, timing of that break, and total number of breaks for a given date range and band

Arguments

- **ccd** (*ee.Image*) – results CCD results in long multi-band format
- **startDate** (*number*) – Start date in the format that was used to run CCD
- **endDate** (*number*) – End date in the format that was used to run CCD
- **band** (*string*) – Spectral band
- **segNames** (*ee.List*) – List of segment names matching the number of segments in the bands

Returns **ee.Image** – Image with three bands indicating: 1) Magnitude of the largest break for the given date range 2) Timing of largest break (in the time units CCDC was run in) 3) Total number of breaks in the date range

phaseAmplitude (*ccd, bands, sinName, cosName*)

Get phase and amplitude for a single spectral band

Arguments

- **ccd** (*ee.Image*) – results CCD results in long multi-band format
- **bands** (*List*) – List with the name of the bands for which to calculate ampl. and phase
- **sinName** (*String*) – Band suffix of the desired sine harmonic coefficient (e.g ‘_SIN’)
- **cosName** (*String*) – Band suffix of the desired sine harmonic coefficient (e.g ‘_COS’)

ccdc.newPhaseAmplitude (*ccd, sinExpr, cosExpr*)

Get phase and amplitude. Replace old function with this.

Arguments

- **ccd** (*ee.Image*) – results CCD results in long multi-band format
- **sinExpr** (*String*) – Regular expression of the sine harmonic coefficient (e.g ‘.*SIN.*’) for all harmonics
- **cosExpr** (*String*) – Regular expression of the cosine harmonic coefficient (e.g ‘.*COS.*’) for all harmonics. Must retrieve the same number of bands as sinExpr

4.1.2 Inputs

getLandsat (*options, start, end, targetBands*)

Get Landsat images for a specific region Possible bands and indices: BLUE, GREEN, RED, NIR, SWIR1, SWIR2, NDVI, NBR, EVI, EVI2, BRIGHTNESS, GREENNESS, WETNESS

Arguments

- **options** (*ee.Dict*) – Parameter file containing the keys below
- **start** (*String*) – First date to filter images
- **end** (*String*) – Last date to filter images
- **targetBands** (*list*) – Bands and indices to return

Returns *ee.ImageCol*. Masked image collection with L4, L5, L7, and L8

generateCollection (*geom, startDate, endDate*)

Generate and combine filtered collections of Landsat 4, 5, 7 and 8

Arguments

- **geom** (*ee.Image*) – Geometry used to filter the collection
- **startDate** (*String*) – Initial date to filter the collection
- **endDate** (*String*) – Final date to filter the collection

Returns *ee.ImageCollection* – Filtered Landsat collection

doIndices (*collection*)

Calculate spectral indices for all bands in collection

Arguments

- **collection** (*ee.ImageCollection*) – Landsat image collection

Returns *ee.ImageCollection* – Landsat image with spectral indices

makeLatGrid (*options, minY, maxY, minX, size*)

Create a grid with features corresponding to latitudinal strips

Arguments

- **options** (*Dictionary*) – parameter file
- **minY** (*Number*) – minimum latitude coordinate
- **maxY** (*Number*) – maximum latitude coordinate
- **minX** (*Number*) – minimum longitude coordinate
- **maxX** – maximum longitude coordinate
- **size** (*Number*) – size of features in units of latitudinal degrees

Returns *ee.FeatureCollection* – grid of features along latitudinal lines

makeLonGrid (*options, minY, maxY, minX, size*)

Create a grid with features corresponding to longitudinal strips

Arguments

- **options** (*Dictionary*) – parameter file
- **minY** (*Number*) – minimum latitude coordinate
- **maxY** (*Number*) – maximum latitude coordinate

- **minX** (*Number*) – minimum longitude coordinate
- **maxX** – maximum longitude coordinate
- **size** (*Number*) – size of features in units of latitudinal degrees

Returns **ee.FeatureCollection** – grid of features along longitudinal lines

makeLonLatGrid (*minY, maxY, minX, size*)

Create a grid with features corresponding to longitudinal strips

Arguments

- **minY** (*Number*) – minimum latitude coordinate
- **maxY** (*Number*) – maximum latitude coordinate
- **minX** (*Number*) – minimum longitude coordinate
- **maxX** – maximum longitude coordinate
- **size** (*Number*) – size of features in units of latitudinal degrees

Returns **ee.FeatureCollection** – grid of features along longitudinal lines

getAncillary ()

Get ancillary data for training and classification.

Returns **ee.Image** – Multi-band image containing ancillary layers

getS2 (*roi*)

Get Sentinel-2 surface reflectance data. Taken directly from GEE examples repo.

Arguments

- **roi** (*ee.Geometry*) – target study area to filter data

Returns (*ee.ImageCollection*) Sentinel-2 SR and spectral indices

getS1 (*mode="ASCENDING", focalSize=3*)

Get Sentinel 1 data

Arguments

- **mode** (*string*) – orbital pass mode ('ASCENDING' or 'DESCENDING')
- **focalSize** (*number*) – window size for focal mean (1 means no averaging)

Returns **ee.ImageCollection** – Sentinel 1 collection with VH, VV, and ratio bands smoothed with focal mean

calcNDFI (*Surface*)

Calculate NDFI using endmembers from Souza et al., 2005

Arguments

- **Surface** (*ee.Image*) – reflectance image with 6 bands (i.e. not thermal)

Returns **ee.Image** – NDFI transform

makeCcdImage (*metadataFilter, segs, numberOfSegments, bandNames, inputFeatures*)

Make a ccd image from the most recent known global run

Arguments

- **metadataFilter** (*String*) – Which ccdc run prefix to use
- **segs** (*List*) – List with the segment names

- **numberOfSegments** (*Number*) – Max number of segments to retrieve from the CCDC results
- **bandNames** (*List*) – List with the band names to use
- **inputFeatures** (*List*) – List with the CCDC features to extract

Returns **ee.Image** – Filtered CCDC results in ‘long’ format

calcNDVI (*image*)

Calculate NDVI for an image

Arguments

- **image** (*ee.Image*) – Landsat image with NIR and RED bands

Returns **ee.Image** – NDVI image

calcNBR (*image*)

Calculate NBR for an image

Arguments

- **image** (*ee.Image*) – Landsat image with NIR and SWIR2 bands

Returns **ee.Image** – NBR image

calcEVI (*image*)

Calculate EVI for an image

Arguments

- **image** (*ee.Image*) – Landsat image with NIR, RED, and BLUE bands

Returns **ee.Image** – EVI transform

calcEVI2 (*image*)

Calculate EVI2 for an image

Arguments

- **image** (*ee.Image*) – Landsat image with NIR and RED

Returns **ee.Image** – EVI2 transform

tcTrans (*image*)

Tassel Cap coefficients from Crist 1985

Arguments

- **image** (*ee.Image*) – Landsat image with BLUE, GREEN, RED, NIR, SWIR1, and SWIR2

Returns **ee.Image** – 3-band image with Brightness, Greenness, and Wetness

makeAutoGrid (*geo, size*)

Create a grid with features overlaying the bounding box of a geometry

Arguments

- **geo** (*ee.Geometry*) – geometry to use as spec for grid
- **size** (*Number*) – size of features in units of degrees

Returns **ee.FeatureCollection** – grid of features along

4.1.3 Classification

getMiddleDate (*fc, startProp, endProp, middleProp*)

Get the middle segment date of training data

Arguments

- **fc** (*ee.FeatureCollection*) – Training data feature collection
- **startProp** (*string*) – Property name of segment start year
- **endProp** (*String*) – Property name of segment end year
- **middleProp** (*String*) – Property name of calculated middle attribute

Returns *ee.FeatureCollection* – Training data with middleProp attribute

makeGrids (*region, count, size, seed*)

Make random grids in a region of interest

Arguments

- **region** (*ee.Geometry*) – study region bounding geometry
- **count** (*number*) – number of random grids
- **size** (*number*) – length of one side of grid in m²
- **seed** (*number*) – random number seed or 'random'

Returns *ee.FeatureCollection* – feature collection of random grids

getBinaryLabel (*fc, property, targetClass*)

Convert training data to binary label for target class

Arguments

- **fc** (*ee.FeatureCollection*) – Training data feature collection
- **property** (*string*) – Property label indicating class label
- **targetClass** (*number*) – Class to retain as 1 in binary label

Returns *ee.FeatureCollection* – Training data where 1 = targetClass and 0 equals all other classes

getClassProbs (*fc, coefsToClassify, classList, classifier, property*)

Get class probability for each class in training data

Arguments

- **fc** (*ee.FC*) – feature collection of training data
- **coefsToClassify** (*ee.Image*) – multi-band image of coefficients to classify
- **classList** (*list*) – classes to test probability of
- **classifier** (*ee.Classifier*) – in 'PROBABILITY' mode
- **property** (*string*) – label defining class in training data

Returns *ee.Image* – image with each band being class probability for each input class

getTrainingCoefsAtDate (*trainingData, coefNames=['INTP','SLP','COS','SIN','RMSE','COS2','SIN2','COS3','SIN3'],
bandList=['BLUE','GREEN','RED','NIR','SWIR1','SWIR2'], dateProp-
erty='\"Start_Year\"', extraBands=null, ccdImage=null, segs=['\"S1\", \"S2\",
\"S3\", \"S4\", \"S5\", \"S6\"'], ccdcDateFmt=1, trainingDateFmt=1, scale=30,
landsatParams={start: '1990-01-01',end: '2020-01-01'})*

Get coefficients at a given date for each feature in collection

Arguments

- **trainingData** (*ee.FeatureCollection*) – training data points to extract coefficients for
- **coefNames** (*List*) – coefficient abbreviated names in order of results
- **bandList** (*List*) – list of input band names in order
- **dateProperty** (*string*) – property name containing date in features
- **extraBands** (*List*) – ancillary bands to add as predictors
- **ccdcImage** (*ee.Image*) – Use ccdc coefficients instead of calculating on the fly
- **segs** (*List*) – Segment identifiers for ccdcImage parameter
- **ccdcDateFmt** (*number*) – date format of ccdc date format
- **trainingDateFmt** (*number*) – training data date format (according to ccdc syntax)
- **scale** (*number*) – spatial scale to sample training points at
- **landsatParams** (*Object*) – parameters for ‘getLandsat’ function

Returns *ee.FeatureCollection* – training data with coefficients corresponding to specific date

remapLC (*feats, inLabel, outLabel, inList, outList*)

Remap training labels to GLANCE level 1 land cover

Arguments

- **feats** (*ee.FeatureCollection*) – training data feature collection
- **inLabel** (*string*) – attribute name containing land cover strings
- **outLabel** (*string*) – attribute name for output numeric land cover
- **inList** (*list*) – list of input land cover string values
- **outList** (*list*) – list of output land cover numeric values

Returns *ee.FeatureCollection* – training data feature collection with numeric ‘outLabel’ column in each feature

assignIds (*sample, attributeName="ID"*)

Shuffle the sample and assign sample ID

Arguments

- **sample** (*ee.FeatureCollection*) – training data of point samples
- **attributeName** (*string*) – name to assign ID attribute to

Returns *ee.FeatureCollection* – training data shuffled with unique ID attribute

makeLegend (*classes, palette, title="Legend", width="250px", position="bottom-right"*)

Make a legend widget

Arguments

- **classes** (*array*) – list of input classes
- **palette** (*array*) – list of color palette
- **title** (*string*) – legend title (optional)
- **width** (*string*) – width of panel (optional)
- **position** (*string*) – position on map (optional)

Returns `ui.Panel` – legend panel to display on map

classifyCoefs (*imageToClassify, bandNames, ancillary, ancillaryFeatures, trainingData, classifier, studyArea, classProperty="LC_Num", coefs, trainProp=.4, seed='random'*)

Classify single set of CCDC coefficients. Useful for quick parameter testing and debugging.

Arguments

- **imageToClassify** (*ee.Image*) – Single set of ccdc coefficients to classify
- **bandNames** (*array*) – list of band names to classify
- **ancillary** (*array*) – list of ancillary predictor data
- **ancillaryFeatures** (*ee.Image*) – ancillary data image
- **trainingData** (*ee.FeatureCollection*) – training data
- **classifier** (*ee.Classifier*) – earth engine classifier with parameters
- **studyArea** (*ee.Geometry*) – boundaries of region to subset training data, null uses all data.
- **classProperty** (*string*) – attribute name with land cover label
- **coefs** (*array*) – list of coefficients to classify
- **trainProp** (*float*) – proportion of data to use subset for training
- **seed** (*number*) – seed to use for the random column generator

Returns `ee.Image` – classified image

classifySegments (*imageToClassify, numberOfSegments, bandNames, ancillary, ancillaryFeatures, trainingData, classifier, studyArea, classProperty="LC_Num", coefs, trainProp=.4, seed='random', subsetTraining=true*)

Classify stack of CCDC coefficient, band-separated by segment

Arguments

- **imageToClassify** (*ee.Image*) – ccdc coefficient stack to classify
- **numberOfSegments** (*number*) – number of segments to classify
- **bandNames** (*array*) – list of band names to classify
- **ancillary** (*array*) – list of ancillary predictor data
- **ancillaryFeatures** (*ee.Image*) – ancillary data image
- **trainingData** (*ee.FeatureCollection*) – training data
- **classifier** (*ee.Classifier*) – earth engine classifier with parameters
- **studyArea** (*ee.Geometry*) – boundaries of region to subset training data, null uses all data.
- **classProperty** (*string*) – attribute name with land cover label
- **coefs** (*array*) – list of coefficients to classify
- **trainProp** (*float*) – proportion of data to use subset for training
- **seed** (*number*) – seed to use for the random column generator
- **subsetTraining** (*boolean*) – true to subset training to geometry, false to not

Returns `ee.Image` – classified stack of CCDC segments

parseConfMatrix (*im*, *attribute*="confMatrix")

Parse confusion matrix from string

Arguments

- **im** (*ee.Image*) – classified image with confusion matrix in metadata
- **attribute** (*string*) – name of attribute with confusion matrix

accuracyProcedure (*trainingData*, *imageToClassify*, *predictors*, *bandNames*, *ancillary*, *classifier*,
classProperty="LC_Num", *seed*='random', *trainProp*=.4)

Calculate accuracy metrics using a subset of the training data

Arguments

- **trainingData** (*ee.FeatureCollection*) – training data
- **imageToClassify** (*ee.Image*) – ccdc coefficient stack to classify
- **predictors** (*array*) – list of predictor variables as strings
- **bandNames** (*array*) – list of band names to classify
- **ancillary** (*array*) – list of ancillary predictor data
- **classifier** (*ee.Classifier*) – earth engine classifier with parameters
- **classProperty** (*string*) – attribute name with land cover label
- **seed** (*number*) – seed to use for the random column generator
- **trainProp** (*float*) – proportion of data to use subset for training

Returns **ee.ConfusionMatrix** – a confusion matrix as calculated by the train/test subset

getLcAtDate (*segs*, *date*, *numberOfSegments*, *ccdVersion*, *metadataFilter*, *behavior*, *bandNames*, *inputFeatures*)

Calculate landcover at a date based on pre-classified segments

Arguments

- **segs** (*ee.Image*) – classified ccd segment image
- **date** (*string*) – date of land cover to retrieve in format ‘YYYY-MM-DD’
- **numberOfSegments** (*number*) – number of segments in classification image
- **ccdVersion** (*string*) – version of ccd used for classification
- **metadataFilter** (*string*) – metadata used for classification of ccd
- **behavior** (*string*) – behavior when date is in between segments (‘none’, ‘before’, ‘after’)
- **bandNames** (*array*) – list of band names (such as “BLUE”, “GREEN”)
- **inputFeatures** (*array*) – list of input feature names (such as “INTP” and “RMSE”)

Returns **ee.Image** – matchingDate landcover classification image at date specified in parameter

getMode (*folder*, *matchingString*)

Get mode classification from a stack of overlapping result files

Arguments

- **folder** (*string*) – the path to the folder containing the result files
- **matchingString** (*string*) – an identifier in the result file names

Returns **ee.Image** – band-wise mode classification

4.1.4 Dates

msToDays (*ms*)

milliseconds since epoch (01-01-1970) to number of days

Arguments

- **ms** (*Number*) – ms since 01-01-1970)

Returns **ee.Number** – milliseconds since epoch

dateToJdays (*str_date*)

Convert Date to julian days (i.e. days since 01-01-0001)

Arguments

- **str_date** (*String*) – Date string in yyyy-mm-dd format

Returns **ee.Number** – Julian day

jdaysToms (*jdays*)

Convert julian day (i.e. days since 01-01-0001) to ms since 1970-01-01

Arguments

- **jdays** (*Number*) – Julian day

Returns **ee.Number** – ms since 1970-01-01

jdaysToDate (*jdays*)

Convert julian day (i.e. days since 01-01-0001) to ee.Date

Arguments

- **jdays** (*Number*) – Julian day

Returns **ee.Date** – ee.Date

msToJdays (*ms*)

Convert ms since 1970-01-01 to julian day (i.e. days since 01-01-0001)

Arguments

- **ms** (*Number*) – ms since 1970-01-01

Returns **ee.Number** – Julian day

msToFrac (*ms*)

Convert ms since 1970-01-01 to fractional year

Arguments

- **ms** (*Number*) – ms since 1970-01-01

Returns **ee.Number** – Fractional year

msToDate (*ms*)

Convert ms to ee.Date

Arguments

- **ms** (*number*) – jdays as milleconds

Returns **ee.Date** – ee.Date

fracToms (*frac*)

Convert fractional time to ms since 1970-01-01. DOES NOT ACCOUNT FOR LEAP YEARS

Arguments

- **frac** (*Number*) – Fractional year

Returns **ee.Number** – ms since 1970-01-01

convertDate (*options*)

Convert between any two date formats

Arguments

- **options** (*Dictionary*) – parameter dictionary

Returns **Object** – output reformatted date

4.1.5 Change

A

accuracyProcedure() (built-in function), 46
 applyNorm() (built-in function), 38
 assignIds() (built-in function), 44

B

buildBandTag() (built-in function), 35
 buildCcdImage() (built-in function), 36
 buildCoefs() (built-in function), 35
 buildMagnitude() (built-in function), 35
 buildRMSE() (built-in function), 35
 buildSegmentTag() (built-in function), 35
 buildStartEndBreakProb() (built-in function), 36

C

calcEVI() (built-in function), 42
 calcEVI2() (built-in function), 42
 calcNBR() (built-in function), 42
 calcNDFI() (built-in function), 41
 calcNDVI() (built-in function), 42
 ccdc.newPhaseAmplitude() (ccdc method), 39
 classifyCoefs() (built-in function), 45
 classifySegments() (built-in function), 45
 convertDate() (built-in function), 48

D

dateToDays() (built-in function), 37
 dateToJdays() (built-in function), 47
 doIndices() (built-in function), 40

F

fillNoData() (built-in function), 37
 filterCoefs() (built-in function), 37
 filterMag() (built-in function), 39
 fracToms() (built-in function), 47

G

generateCollection() (built-in function), 40
 getAncillary() (built-in function), 41
 getBinaryLabel() (built-in function), 43

getChanges() (built-in function), 38
 getClassProbs() (built-in function), 43
 getCoef() (built-in function), 38
 getLandsat() (built-in function), 40
 getLcAtDate() (built-in function), 46
 getMiddleDate() (built-in function), 43
 getMode() (built-in function), 46
 getMultiCoefs() (built-in function), 38
 getMultiSynthetic() (built-in function), 36
 getS1() (built-in function), 41
 getS2() (built-in function), 41
 getSyntheticForYear() (built-in function), 36
 getTrainingCoefsAtDate() (built-in function), 43

J

jdaysToDate() (built-in function), 47
 jdaysToms() (built-in function), 47

M

makeAutoGrid() (built-in function), 42
 makeCcdImage() (built-in function), 41
 makeGrids() (built-in function), 43
 makeLatGrid() (built-in function), 40
 makeLegend() (built-in function), 44
 makeLonGrid() (built-in function), 40
 makeLonLatGrid() (built-in function), 41
 msToDate() (built-in function), 47
 msToDays() (built-in function), 47
 msToFrac() (built-in function), 47
 msToJdays() (built-in function), 47

N

normalizeIntercept() (built-in function), 37

P

parseConfMatrix() (built-in function), 45
 phaseAmplitude() (built-in function), 39

R

remapLC() (built-in function), 44

T

`tcTrans()` (*built-in function*), [42](#)