Jess Strzempko GEOG 296 Advanced Raster GIS
Final Project 5.1.20

Tutorial: Habitat Suitability Modeling using Random Forest Classification in Google Earth Engine

Introduction

In recent decades, machine learning models have emerged as a prevalent mode of inductive modeling in the GIS community. Inductive or empirical models are ones in which known information is collected, observed for patterns, and applied to develop new theoretical frameworks. They are often defined in opposition to deductive models in which a hypothesis is formulated based on accepted paradigms and is tested using observable data to see if this information follows the general theory. In the field of habitat biodiversity or suitability modeling, a specific application of empirical modeling, data can be in the form of species observations/sightings which are then related to environmental variables to produce potential species range maps.

The technique this tutorial will use is a form of supervised classification called Random Forest where the model is trained using species observations to generate an output map of potential species presence (Komodo, 2018). Random decision forests are an extension of Classification Tree Analysis (CTA) which are most often applied to remotely sensed data for land cover classification purposes. In a single classification tree, the input data (in this case pixels) is partitioned into increasingly homogeneous groups (classes) based on their relation to a set of predictor variables (spectral bands). While this method is robust to outliers, carries no assumptions, and is simple to understand, it is very unstable with small changes in the input producing very different decision trees.

In response, random forest classification was developed as an extension of CTA where multiple decision trees are generated using random, but equally sized subsets of the data (with replacement) with majority voting often deciding the final class of a pixel (Brieman, 2011). In Google Earth Engine, various parameters of the Random Forest classifier can be modified to alter output. Several of these include the number of decision trees to create, the fraction of the input to "bag" per tree (the random subset selected), and the maximum number of lead nodes (classes) in each tree. With the ability to easily invoke and run classification algorithms across large, openly sourced datasets, Earth Engine presents ample geoprocessing opportunities to explore.

Objectives

- Familiarize users with Google Earth Engine modeling applications
- Walk users through a simple application of the Random Forest machine learning algorithm to habitat biodiversity modeling (HBM)
- Highlight opportunities for users to explore beyond the scope of the tutorial by building off existing scripts

Study Area

In this tutorial, the distribution of the *Vicugna vicugna* (vicuña) will be modeled. The vicugna is a member of the camel family with relation to alpacas, llamas, and guanacos (Encyclopædia Britannica). They are found in South America in the Andes mountain range of southern Peru, western Bolivia,

northwestern Argentina, and northern Chile. Though the vicugna has been hunted for centuries primarily for its fur, recent conservation efforts and a listing as vulnerable on the International Union for Conservation of Nature's (IUCN) Red List have resulted in recovering population size. Through the use of observation points and environmental variables, we will create a potential species range map for the vicugna.

Data

Data for this tutorial is sourced from class resources (TerrSet tutorial data) as well as datasets readily available through Google Earth Engine. The first dataset to import is a point shapefile of observations of vicugna in South America. These were originally part of Exercise 6-3 HBM: Maxent in Chapter 6 of the TerrSet Tutorial. In order for Google Earth Engine to read these files, they need to be run through SHAPEIDR in TerrSet to convert from a .vct to a .shp file type. At this stage, "absence" points were added to the dataset in regions where it would be extremely unlikely to find vicugna (e.g. Amazon rainforest, lake, or coast) using ArcMap. In future applications, the location of these points should be based off expert or field-based recommendations. These points were given a presence value of 0 while original observation points were given a presence value of 1. Next, this shapefile was sent to a zipped folder so that Google Earth Engine could read it as a single file. This zipped folder is provided with this final project for ease of access.

The analysis will also be based off open source environmental variables gathered from the Google Earth Engine search bar. Users should first search for WorldClim BIO Variables V1, a set of 19 global "bioclimatic" variables, compiled by University of California, Berkeley (Fick & Hijmans, 2017). They represent average values of various climatic variables for the years 1970-2000 and are available at a 1-kilometer spatial resolution. In the tutorial, users will learn how to select bands corresponding to annual mean temperature and annual precipitation.

Additionally, users will search for and import USGS Landsat 8 Surface Reflectance Tier 1 data (USGS EROS, 2013). From these bands of atmospherically corrected surface reflectance, users will select the visible, near-infrared, short-wave infrared, and thermal infrared bands (effectively excluding the aerosol attributes, pixel quality attributes, and radiometric saturation quality assessment bands). Landsat 8 data is available at 30-meters spatial resolution and will be filtered for low cloud cover across 2019 to create composite bands.

Lastly, for visualization purposes, the LSIB: Large Scale International Boundary Polygons, Detailed dataset will be used to select a region of interest and visualize country boundaries in relation to the analysis outputs (US Dept of State, 2017).

Methodology

It is recommended that users have a working knowledge of JavaScript for Earth Engine before starting this tutorial. Sample code (that can be cut and pasted into the code editor) will be provided to guide the user along, but previous exposure to this mode of analysis and relevant syntax will enhance learning. Provided below are several introductory sites to get started.

Introduction to JavaScript for Earth Engine: https://developers.google.com/earth-engine/tutorial_js_01

(good overview of commonly used statements and basic data types)
Get Started with Earth Engine: https://developers.google.com/earth-engine/getstarted#the-code-editor
(helpful in exploring/locating parts of the Google Earth Engine Application Programming Interface (API))

Additionally, users will need an Earth Engine account. It is easiest if you have an existing google account. Google "Sign up for Google Earth Engine" or visit https://signup.earthengine.google.com and an email will be sent to your inbox when you are approved. Visit code.earthengine.google.com to get started.

Results
Link to final code: https://code.earthengine.google.com/10a6b8ef6aca75843c27ec161cf16a92

A. Import Vicguna Data
    1. To begin, we are going to **import the vicugna shapefile** from a zipped folder. Navigate to the Asset tab on the left side of the screen and click the large red drop-down menu labeled "New." Select Shapefile (.shp, .shx, .dbf, .prj, .zip) under the Table Upload option and the window "Upload a new shapefile asset" will appear (Figure 1).
    2. You can upload multiple files at one time, but they must all correspond to the same shapefile. Rather than selecting each file separately, we will be uploading a zipped folder which contains all relevant files. Under source files, hit the select button and navigate to where your vicugna zipped folder is on your computer and select to open it. Name the asset "vicguna," leave all other properties as default, and scroll down to select upload.
    3. At this point, your Tasks tab on the right side of the screen may blink orange indicating that there are unseen tasks occurring. Click on this tab and you should see your asset being ingested. This may take a few minutes.
    4. When Earth Engine is done ingesting your asset, it will appear blue in the Tasks tab. If it hasn't already appeared in your Assets tab then hit the refresh button and "vicguna" should be added to the list. When you hover over an asset, three icons will appear: Share, Rename, Import into Script. Select the arrow to Import into Script and the "vicguna" table with its pathname will be added to your list of imports at the top of the code editor. Change the variable name to "vicguna."
    5. Visualize this data by adding this statement to your code and running the script (Ctrl + Enter) or hitting the run button at the top of the code editor:
        i. `Map.addLayer(vicugna, {}, 'Vicugna Data');` (if you need help understanding this code, type Map.addLayer into the Docs search bar on the left side of the screen. This documentation gives the inputs, outputs, and arguments to a statement – for instance, we want to add our layer "vicugna" to the map with default palette {}, and name it "Vicugna Data")
    6. On the right side of the screen, select the Inspector tab. Now, when you click on features in the map, information about them will appear in the Console tab. Investigate the data by clicking on points. Pay particular attention to the value of the 'presence' property.
    7. Next, we will create a variable called "label" in which we will store the property 'presence.' When the data was created, locations were vicugna were observed were

given a value of 0, while areas of extreme unlikely for this species were given a value of 0 in a column labeled 'presence.' We will base the creation of our classification off this property.

i. `var label = 'presence';`

B. Decide on a Region of Interest

1. Next, we will import data on world country boundaries and select a region of interest to bound our classification. In the search bar at the top of the screen (search places and datasets), type in "LSIB: Large Scale International Boundary Polygons, Detailed" and hit enter. Select the first option to appear (should be for the year 2017). Read the description and look through the table schema for this data before selecting the blue import button.
2. Similar to the vicugna table, this data should appear in your imports. Name it "world."
3. To minimize runtime and increase simplicity of the tutorial, we will select only the country of Peru as our region of interest. To do so, filter the world layer by selecting the feature where the property 'COUNTRY_NA' = 'Peru.'
    i. `var roi = world.filter(ee.Filter.eq('COUNTRY_NA', 'Peru'));`
4. Center the map on this region of interest and set it to a zoom level of 5.
    i. `Map.centerObject(roi, 5);`
5. You can also try out a larger analysis region defined by the user rather than country boundaries (Figure 2). To do so, navigate to the map below and look to the upper left-hand corner. Here you can manage and create geometry imports (points, lines, polygons, and rectangles). Zoom out so your map includes all of Peru and Bolivia and draw a rectangle that encompasses these countries. Under geometry imports, select "edit layer properties" and rename the layer "vicguna_region." It will appear with your world and vicugna imports from previous steps. Comment out the previous roi variable (by adding // before the statement) and add this code:
    i. `var roi = vicugna_region;`
    ii. Visualize this new roi: `Map.addLayer(roi, {}, 'Region of Interest');`
6. Return to the original roi by removing the // and adding them in front of the the vicugna_region roi. You can only define a variable once.

C. Import Environmental Variables

1. Similar to before, search for WorldClim BIO Variables V1 in the search bar. Explore this data and the bands it contains. We will be selecting bio01 and bio12 for analysis (Figure 3).
2. Import the image and name it "bioclim."
3. Next, we will create a new variable called "temp_bioclim" from the mean annual temperature band (bio01). Type and run this code:
    i. `var temp_bioclim = bioclim.select('bio01');`
4. We can also create visualization parameters with which to explore the data, by setting minimum and maximum display values and assigning a palette of colors. This visualization variable can be added to our Map.addLayer function.

i. var temp_vis = {min: -230.0, max: 300.0, palette: ['blue', 'purple', 'cyan', 'green', 'yellow', 'red']};
ii. Map.addLayer(temp_bioclim, temp_vis, 'Annual Mean Temperature');

5. Once you've explore the worldwide temperature trends, we will clip this layer to our roi to increase processing speed. Comment out the previous statement to remove the worldwide data from the map.
    i. var temp = temp_bioclim.clip(roi);
6. Conduct a similar process to obtain and visualize annual precipitation (bio12). Make sure to comment out extraneous statements when done exploring.
    i. var precip_bioclim = bioclim.select('bio12');
    ii. var precip_vis = {min: 0, max: 10000, palette: ['white', '00008B', 'black']};
    iii. var precip = precip_bioclim.clip(roi);
    iv. Map.addLayer(precip, precip_vis, 'Annual Precipitation');

(Optional Addition of Landsat Data)

7. Similar to previous steps, search for USGS Landsat 8 Surface Reflectance Tier 1 and import this data. If you've ever worked with Landsat data before, Earth Engine gives easy access to all Landsat image collections. Name this import "L8."
8. Next, we want to create a variable of Landsat 8 data filtered for 2019 over our roi.
    i. var L8_filter = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi).filterDate('2019-01-01', '2019-12-31');
9. Currently, we have an image collection consisting of surface reflectance for all Landsat bands for all of 2019 over our roi. We will want to create a function to filter out cloudy imagery using a cloud mask. We will use bits 3 and 5 (cloud shadow and cloud) in the pixel quality assessment (QA) band to filter for images where the flags are set to 0 indicating clear conditions. We will scale the image, so values are [0,1].
    i. function maskL8(image) {
    var cloudShadowBitMask = ee.Number(2).pow(3).int();
    var cloudsBitMask = ee.Number(2).pow(5).int();
    var qa = image.select('pixel_qa');
    var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
    .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
    return image.updateMask(mask).divide(10000);}
10. We will then map this function over the 2019 data, take the median pixel values (so as to avoid the influence of outliers), and clip to our roi.
    i. var L8_median = L8_filter.map(maskL8)
    .reduce(ee.Reducer.median())
    .clip(roi);
11. Create a true color composite of the median bands to visualize the images you have just created. We can utilize the visualization parameters to create a composite of 3 bands (Figure 4).
    i. Map.addLayer(L8_median,
    {bands: ['B4_median', 'B3_median', 'B2_median'], min: 0, max: 0.2},
    'Landsat 8 Composite');

D. Create a Composite of the Layers
    1. In order to run the classification, we will need all of our environmental variables in one variable that we can reference. If you included the Landsat 8 bands, you need to create a new variable called “L8_bands” which excludes the pixel qa, aerosol, and radarsat bands. If you did not run the Landsat 8 steps, then you can simply add the temperature data to the precipitation data to create a new composite variable. In both cases, we need a variable “bands” which includes the names of all the bands in this composite variable.
        i. First option
        ```
        var enviro = temp.addBands(precip);
        var bands = enviro.bandNames();
        ```
        ii. Second option
        ```
        var L8_bands = L8_median
        .select('B1_median', 'B2_median', 'B3_median', 'B4_median',
        'B5_median', 'B6_median', 'B7_median', 'B10_median', 'B11_median');
        var enviro = L8_bands.addBands(temp).addBands(precip);
        var bands = enviro.bandNames();
        ```

E. Sample the Input Imagery to get Training Data
    1. Next, we will need to create training data by sampling our environmental variables at the points in the vicugna dataset. This will create a feature collection of training data which we can use to train the classifier. We will use the sampleRegions statement across our vicguna selection, making sure that our data is separated across the label property (representative of our ‘presence’ classes).
        i.
        ```
        var training = enviro.select(bands).sampleRegions({
        collection: vicugna,
        properties: [label],
        scale: 30});
        ```

F. Train and Run a Discrete Classifier
    1. In this first section, we will train and run a discrete Random Forest classifier, meaning that the output will contain presence (value = 1) and absence (value = 0) regions. In this case, if greater than 50% of the decision trees mark a pixel a certain value, then it will be given this value. The output map will be a Boolean/binary image. This can be useful when generating species range maps as it can aid in generating a range perimeter.
    2. For our run of the Random Forest classifier (called smileRandomForest in GEE), we will set the classification to use 20 decision trees with all other parameters at default.
        i.
        ```
        var classifier_disc = ee.Classifier.smileRandomForest(20)
        .train({
        features: training,
        classProperty: label,
        inputProperties: bands});
        ```
    3. Once we have created the discrete classifier, we can run it on our “enviro” image and display the resulting Map.

i.
```
var classified_disc = enviro.select(bands).classify(classifier_disc);
Map.addLayer(classified_disc,
{min: 0, max: 1, palette: ['FFFFFF','031A00']},
'Random Forest Discrete Classification');
```

4. Explore the output (Figures 5 & 6). Use the inspector tool and the pan option to move around your image and select regions to investigate their value. When might a map like this be useful?

G. Train and Run a Soft Classifier (probabilities)
   1. To compare outputs, we will also run a soft classifier and scale the output probabilities to [0,100] to see gradients of habitat suitability for the vicugna. The same color palette was used for both classifiers to produce a contrast between results.
      i.
```
var classifier_prob = ee.Classifier.smileRandomForest(20).
setOutputMode('PROBABILITY')
.train({
features: training,
classProperty: label,
inputProperties: bands});
```
   2. Again, we will run the classifier on the data and display the output.
      i.
```
var classified_prob = enviro.select(bands).classify(classifier_prob).multiply(100);
Map.addLayer(classified_prob,
{min:0, max:100, palette: ['FFFFFF','C6AC94','8D8846','395315','031A00']},
"Random Forest Probability");
```
   3. Explore the output (Figure 5 & 6). Use the inspector tool to select certain regions. Compare their values to the discretely classified image. What advantages or disadvantages does this soft classification provide?

H. Confusion Matrix representing Re-substitution Accuracy
   1. Earth Engine recommends in its supervised classification tutorial/help documents that you create a confusion matrix (table that describes performance of the algorithm by comparing output data with test data for which values are known). Unfortunately, this data does not have another dataset available for validation (in the way land cover classes might) meaning that resubstitution error is simply the error rate obtained from comparing the training data to the output. The output confusion matrix and accuracy measurement will appear in your Console tab on the right side of the screen.
      i.
```
var trainAccuracy = classifier_disc.confusionMatrix();
print('RF Resubstitution error matrix: ', trainAccuracy);
print('RF Training overall accuracy: ', trainAccuracy.accuracy());
```
   2. Because each pixel has to be in a certain class for comparison, we can only conduct this on our hard/discrete classified image. Resubstitution error is optimistic and often unrealistic. Further validation data would be needed to generate a cross-validation matrix, which would provide more accurate assessments.

I. Additional Step

1. For visualization purposes, create an empty image to paint the roi features into. This will allow us to see country boundaries as outlines (which will be less disruptive to the analysis).
    i. Map.addLayer(ee.Image().paint(roi, 1, 2), {}, 'Region of Interest');

J. Export Image
    1. To export the image to your google drive, set the parameters as follows:
        i. Export.image.toDrive({
        image: classified_prob,
        description: 'Random Forest Probability',
        maxPixels: 1e20, scale: 500, region: roi,
        folder: 'Advanced_Raster'});
    2. Once you've run the script with this statement, a separate option for you to run the export will appear in your Tasks tab where you can decide on resolution, location, and name of the output image (Figure 7).

Conclusions

While the Random Forest classifier is one of many classification methods available in Earth Engine, it is perhaps the most easily understood machine learning algorithm because of its basis in simpler methods. The simple construction and modification of scripts combined with easy access to open source data via the Earth Engine search bar makes this program especially useful in this geoprocessing application. The script from this tutorial can be used with different data or can be expanded upon to test the classification methods with a set of new parameters. Ultimately, the application of Random Forest classification to habitat suitability/biodiversity modeling offers abundant opportunity to explore species distribution through the lens of biodiversity factors.

References

Breiman, L. (2001). Random Forests. *Machine Learning, 45*, 5–32.

Encyclopædia Britannica, The Editors of. "Vicuña." *Encyclopædia Britannica*, Encyclopædia Britannica, inc., 11 February 2020, https://www.britannica.com/animal/vicuna.

Fick, S. E., & R. J. Hijmans. (2017). WorldClim 2: new 1km spatial resolution climate surfaces for global land areas. *International Journal of Climatology, 37*(12): 4302-4315.

Komodo, ipta Kreatif. (2018). Proposed workflow to for habitat modelling using GBIF mediated data and Google Earth Engine. https://komodowebhost.com/index.php/features-mainmenu-47/geographic-information-system/289-proposed-workflow-to-for-habitat-modelling-using-gbif-mediated-data-and-google-earth-engine

United States Department of State, Office of the Geographer. (2017). LSIB: Large Scale International Boundary Polygons, Detailed [Data file]. Retrieved from: https://developers.google.com/earth-engine/datasets/catalog/USDOS_LSIB_2017.

United States Geological Survey (USGS) Earth Resources Observation and Science (EROS) Center. (2013). Landsat 8 Surface Reflectance Tier 1 [Remote-Sensing Image]. Retrieved from: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC08_C01_T1_SR

Figures

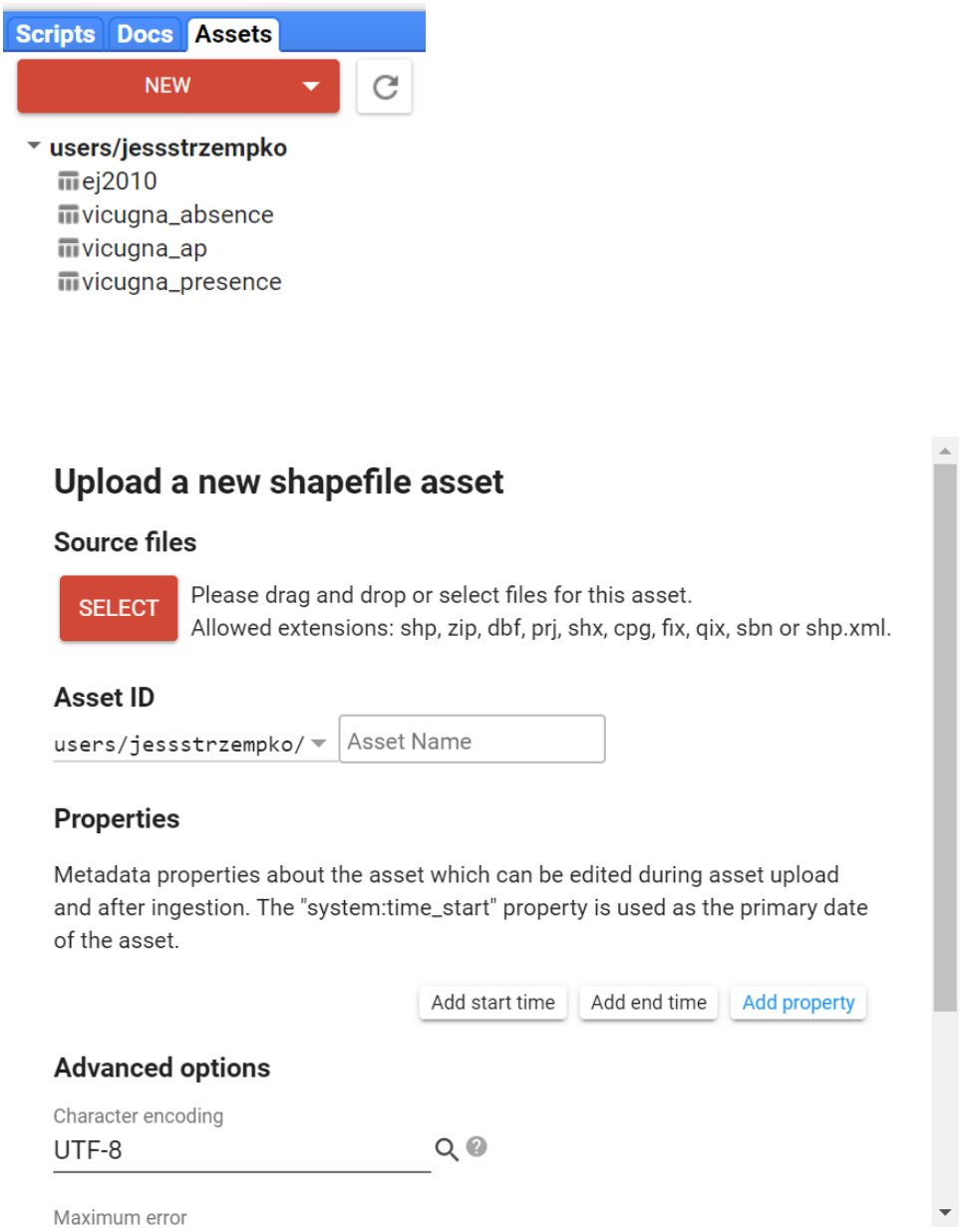

Figure 1. Screenshots of uploading the shapefile as an asset.

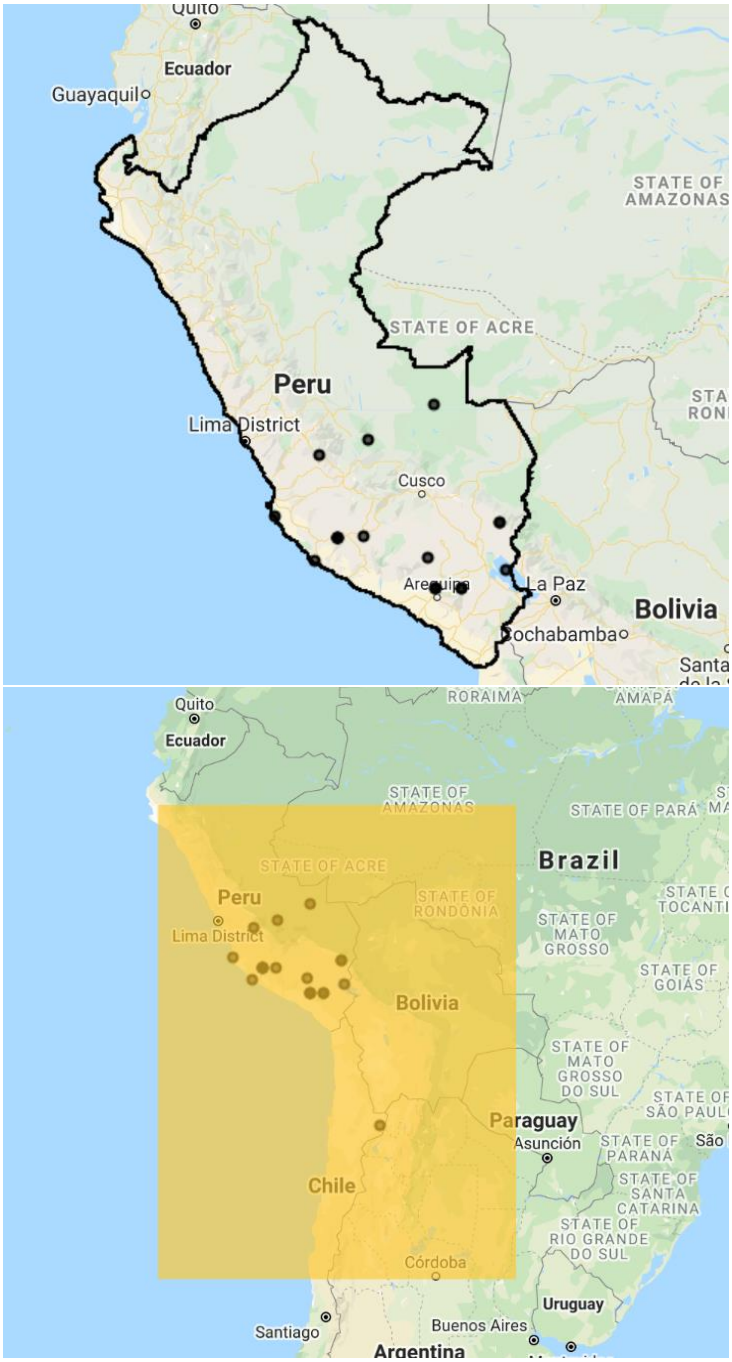

Figure 2. Representations of the 2 possible regions of interest.

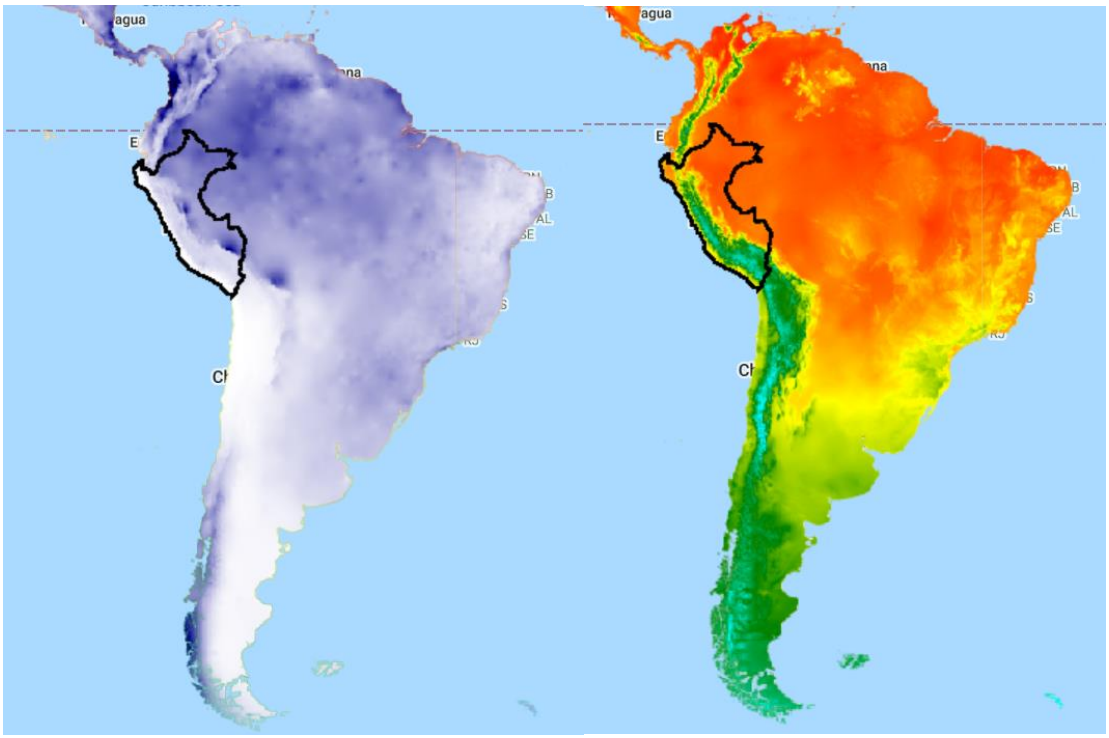

Figure 3. Visualizations of bioclimatic variables (precipitation and temperature).

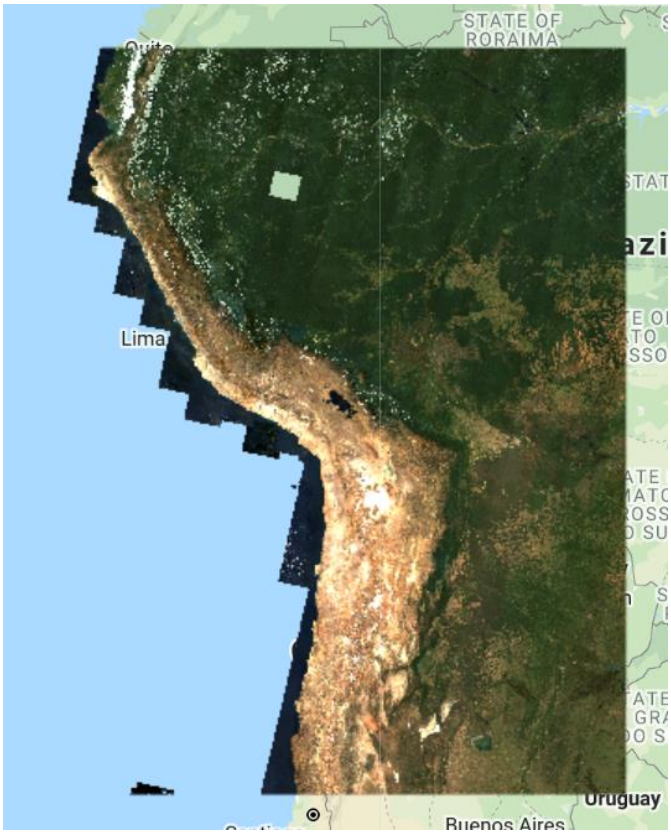

Figure 4. True color composite of L8_median.

Figure 5. Output maps for Peru.

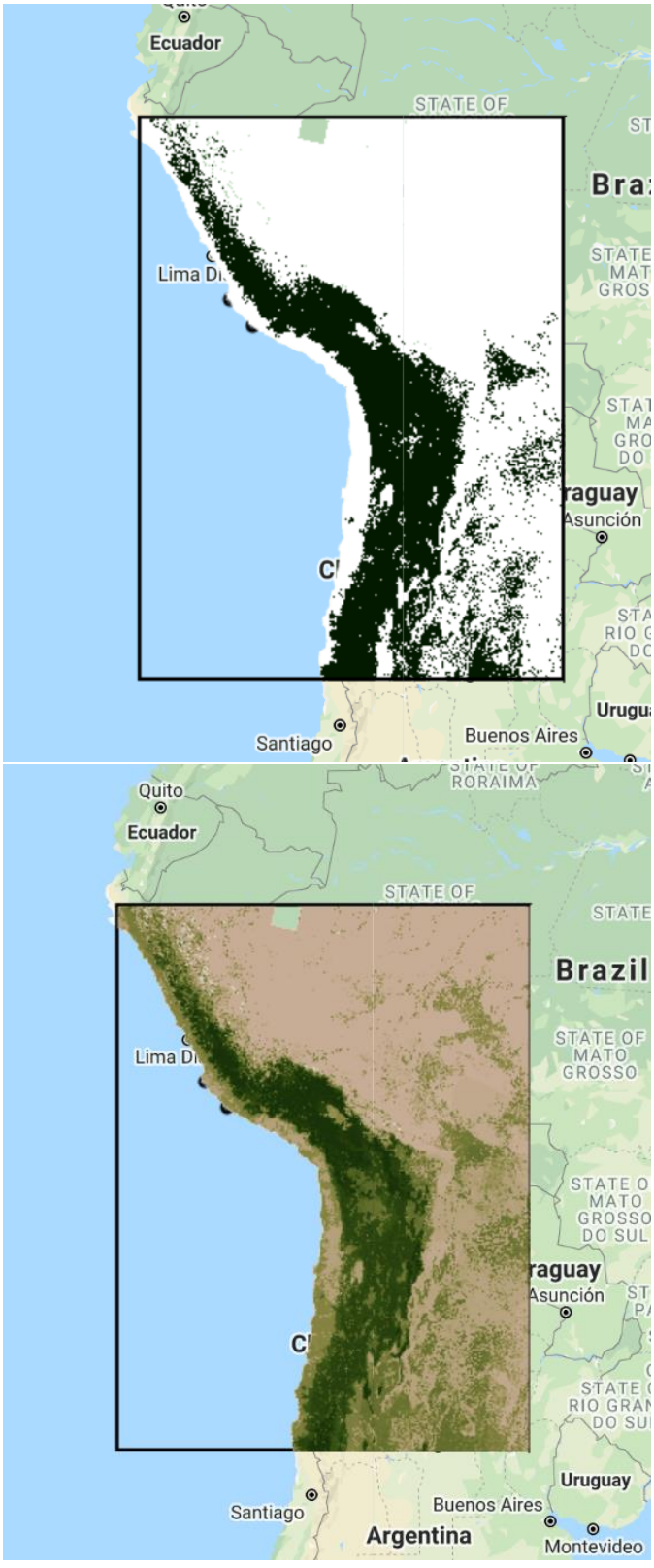

Figure 6. Output maps for a user defined region of interest.

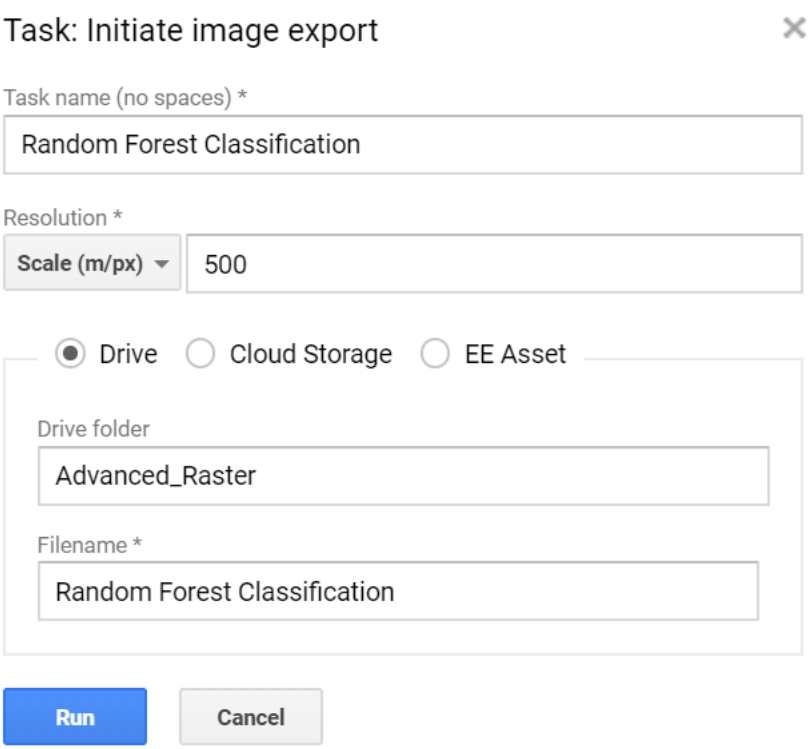

Figure 7. Window to export image.