

EXERCISE 4.1

Run a Random Forest Soil Classification



Introduction

The previous exercises have taught us how to 1) access Landsat Imagery to produce a seasonal composite, 2) calculate spectral indices using Landsat spectral bands, and 3) use other GEE resources to compile topographic and climatic data. The next step is to use this newfound information to perform a Random Forest classification in Earth Engine.

The region of interest for this exercise is Ryan Flats, Texas – all the necessary data will be provided in the course folder. Throughout this exercise, we will be classifying seven (7) soil types during this classification, producing useful accuracy assessment charts and statistics, and validating our classification.

Objectives

- Use **DSM_lib** of functions to load in Landsat composite and stack predictor layers
- Run a Random Forest classification with an understanding of how parameters effect the model
- Conduct model assessment and validation by interpreting statistics and figures
- Display final classification and legend on map

Required Data:

- **TX_boundary.shp** – shapefile representing example area of interest
- **TX_pedons.shp** – shapefile of training data for Ryan Flats, Texas



Prerequisites

- **Completion of Exercise 1-3 (you can review code by accessing the 02_ExerciseCompleteScripts folder within the course repository)**
- **Google Chrome installed on your machine**
- **An approved Google Earth Engine account**
- **Follow the links below to gain read access to the GEE code repositories we will refer to in the script.**
 - [Click here to gain access to the GTAC module repository](#)
 - [Click here to gain access to the GTAC training repository](#)



USDA Non-Discrimination Statement

In accordance with Federal civil rights law and U.S. Department of Agriculture (USDA) civil rights regulations and policies, the USDA, its Agencies, offices, and employees, and institutions participating in or administering USDA programs are prohibited from discriminating based on race, color, national origin, religion, sex, gender identity (including gender expression), sexual orientation, disability, age, marital status, family/parental status, income derived from a public assistance program, political beliefs, or reprisal or retaliation for prior civil rights activity, in any program or activity conducted or funded by USDA (not all bases apply to all programs). Remedies and complaint filing deadlines vary by program or incident.

Persons with disabilities who require alternative means of communication for program information (e.g., Braille, large print, audiotope, American Sign Language, etc.) should contact the responsible Agency or USDA's TARGET Center at (202) 720-2600 (voice and TTY) or contact USDA through the Federal Relay Service at (800) 877-8339. Additionally, program information may be made available in languages other than English.

To file a program discrimination complaint, complete the USDA Program Discrimination Complaint Form, AD-3027, found online at [How to File a Program Discrimination Complaint](#) and at any USDA office or write a letter addressed to USDA and provide in the letter all of the information requested in the form. To request a copy of the complaint form, call (866) 632-9992. Submit your completed form or letter to USDA by: (1) mail: U.S. Department of Agriculture, Office of the Assistant Secretary for Civil Rights, 1400 Independence Avenue, SW, Washington, D.C. 20250-9410; (2) fax: (202) 690-7442; or (3) email: program.intake@usda.gov.

USDA is an equal opportunity provider, employer, and lender.



Table of Contents

EXERCISE 4.1	1
Run a Random Forest Soil Classification.....	1
Part 1: Prepare Landsat composite & predictor layers	5
Part 2: Prepare training/validation data.....	7
Part 3: Run the Random Forest Classification	8
Part 4: Add classification to map, create a legend	10
Part 5: Create model assessment statistics and figures	13
Part 6: Perform validation on classification.....	15
Part 7: Export	16
Part 8: Discussion.....	18

Part 1: Prepare Landsat composite & predictor layers

The first step is to prepare the Landsat image composite and compile all the predictor layers. In the past 3 exercises, we practiced loading different dataset that is useful for classification – spectral indices, topographic data, and climate data. Now, we will combine all the layers calculated in the previous exercises into one image for our classification. We will load in a library full of functions that will allow us to easily load the data we need.

A. Load the exercise script and input data

1. In the course repository, navigate to the script named **ex4.1_RFclassification** (located in the users/USFS_GTAC/GTAC-Training repository in the folder location: DigitalSoilMapping, 02_Exercises, and 01_ExerciseWorksheets). Save it to your own repository.
2. Open the script and inspect it. The first thing you will notice is the heading, describing the title, authorship, date modified, and summary of the script.
3. Then, you will see a blank ‘skeleton’ script, which you will be completing during this exercise.
4. Ensure that you have uploaded the **TX_boundary.shp** file to your assets folder and imported to your script, making changes to the import name as necessary (hint: change the name from **table** to **TX_boundary**).

B. Prepare the composite + predictor layers

1. First, we must load in the user editable variables. Again, these will be the same as in the first exercises. Copy these lines of code below the comment that reads “**User Editable Variables.**” Note that we have already included comments describing what each variable represents.

```
var year = 2019; // Start year for composite
var startJulian = 100; // Starting Julian Date
var endJulian = 272; // Ending Julian date
var compositingPeriod = 0; // Number of years into the future to include
var compositeArea = TX_boundary.geometry().bounds(); // ROI shapefile/asset or polygon
var roiName = 'RyanFlats_TX'; // Give the study area a descriptive name.
var exportToDrive = 'no'; // Option to export landsat composite to drive
var crs = 'EPSG:32613'; // EPSG number for output projection. 32613 = WGS84/UTM Zone 13N. For more info- http://spatialreference.org/ref/epsg/
```

2. We have combined the commands we wrote in the previous exercises into a library of functions that load the composite and other predictor layers for a given area. To use these functions, we must load in the library. Under the comment that reads “**Load in Library of functions**”, paste:

```
var loadData = require('users/USFS_GTAC/GTAC-
Training:DigitalSoilMapping/03_Library/DSM_Lib');
```

3. Use the function and the parameters to load the Landsat composite. Here, we create the object *inImage* by referencing the library that we loaded, calling the function *getComp*, and then providing the input parameters. Copy this line of code into your script below the line that reads **“Use function to load Landsat composite.”**

```
var inImage = loadData.getComp(compositeArea, year, compositingPeriod,
startJulian, endJulian, exportToDrive, roiName, crs);
```

4. Next, we will use more functions from the library to add in our other predictor layers. The layers we’re loading include the spectral indices from exercise 2, and the topographic and climate data from exercise 3. Then, we ‘stack’ them so all the layers exist in one image object.
5. Under the comment that reads **“Load in other predictor layers, stack them all”**, paste:

```
var inSpectral = loadData.getSpectralIndices(inImage, crs);
var inSpectralTopo = loadData.getTopo(inSpectral, crs);
var inSpectralTopoClimate = loadData.getClimate(inSpectralTopo, year,
compositingPeriod, crs);
```

6. Since we’re dealing with lots of difference layers from different datasets, it’s very important that we ensure all the input data has the same projection and scale before running a model. To do this, we will reproject the *inStack* to the ‘crs’ specified and a 30 m scale. Under the comment that reads **“Reproject predictor layers to the same projections and scale”**, paste:

```
var inStack = inSpectralTopoClimate.reproject(crs, null, 30);
```

7. Go back and add any additional comments that will help you remember what the code is doing, or clarify your understanding.
8. To view our predictor layers in the console, paste the following under the comment that reads **“Print and check it out!”**:

```
print("Predictor Layers:", inStack);
```

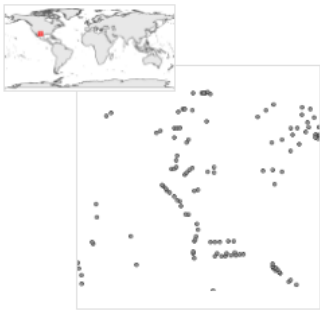
9. Click **Save** to save the code.
10. Click **Run**. Only click Run once and be patient. GEE is slow about loading libraries.
11. An object called **“Predictor Layers”** will appear in the console. Click the down arrows next to **“Image”** and then next to **“bands”** to inspect this object. Observe that we have created an image that contains 23 total bands: 6 bands from the Landsat composite, 8 bands of spectral indices, 4 bands of topographic derivatives, and 6 bands of climate data!

Part 2: Prepare training/validation data

A. Load in the AOI pedons shapefile

1. Before we started this exercise, you should have loaded the **TX_pedons.shp** into GEE as an asset. Now, we're going to import that asset into our script so we can use it in our classification.
2. Go to the assets tab in the left panel, and find the shapefile of interest (**ryanflat_pedons_all.shp**), and click it. This screen should come up:

Table: TX_RyanFlats_pedons



Feature Index	Class_num (Float)	soil (String)	system:index (String)	veg (String)	x (Float)	y (Float)
0	1	Barlite		Junk	570438	3353840
1	1	Barlite		Junk	574423	3354180
2	1	Barlite		Junk	574064	3354370
3	1	Barlite		Junk	574133	3354430
4	1	Barlite		Junk	573630	3354810
5	1	Barlite		Junk	573617	3354880
6	1	Barlite		Junk	573428	3354990
7	1	Barlite		Junk	573508	3355010
8	1	Barlite		Junk	573334	3355090
9	1	Barlite		Junk	573406	3355130

Table ID: users/julbateman/NRCS_GEE_DSM/Texas/TX_RyanFlats_pedons

Date: Start date: NA, End date: NA

File Size: 16.59KB

Number of Features: 121

*Limited to the first 10 features.

IMPORT DELETE SHARE CLOSE

- i. Navigate to the **FEATURES** tab and explore the different attributes of the shapefile. Since we are classifying soil type in this exercise, we are interested in the 'Class_num' field (which is directly correlated to the 'soil' class).
 - ii. Click the blue **IMPORT** button at the bottom right corner to add it to your script.
3. When you go back to your script, you'll see a new table has been added to your list of Imports at the top. Change the name of the new shapefile from **table** to **TX_pedons**.
 - i. Now you should have two imports: **TX_boundary** and **TX_pedons**.

B. Prepare training and validation data using pedons

1. Using the **TX_pedons** imported variable, we will now add that as a feature collection so that we can use it for our classification.
2. For the purposes of this exercise, we're going to split this dataset up so 70% of the points are used as training data and the other 30% is used as validation data. Under the comment that reads "Load in training data, separate it 70%/30% for training/validation", paste:

```
var data = ee.FeatureCollection(TX_pedons, 'geometry');
```

```
var datawithColumn = data.randomColumn('random');
var split = 0.7; // separate 70% for training, 30% for validation
var trainingData = datawithColumn.filter(ee.Filter.lt('random', split));
var validationData = datawithColumn.filter(ee.Filter.gte('random', split));
```

3. If you're curious to see what these objects look like, paste the follow in your code to print the datasets in your console. Note the separation of the pedon dataset into training and validation data.

```
print('Pedon dataset:', data);
print('Validation Data:', validationData);
print('Training Data:', trainingData);
```

4. Good job! Now we've added in our training and validation data points for our classification. Don't forget to save your work as you go!!

Part 3: Run the Random Forest Classification

A. Select the predictor layers of interest

1. For the purposes of this exercise, we are going to be using all the predictor layers that we calculated in Part 1. BUT, it's important to know that you could easily choose what predictor layers to include in your classification (HINT: this will be useful when you use your own data tomorrow!)
2. Under the comment that reads "**Select predictor layers to include in classification**", paste:

```
var bands = inStack.bandNames(); //All bands on included here
```

B. Collect training data

1. Our next step is to use our training points to get the value of each predictor layer at that exact location. This extracts the values of our 24 predictor layers to each of our training points. Our model will then run across the 24 predictor layers, attempting to model which combinations of predictor variables and values are associated with each of the soil types in our classification. Paste the following under the comment that reads "**Intersect training points with predictor layers to get training data**":

```
var training = inStack.select(bands).sampleRegions({
  collection: trainingData,
  properties: ['Class_num'],
  scale: 30
});
```


Note: we set the scale for training data to 30 m – keeping it consistent with the predictor layer projection we applied earlier.

- i. If you're interested in exploring the **sampleRegions** command further, simply type **"ee.Image.sampleRegions"** into the **Docs** search bar in the left panel.

C. Run the RF classifier

1. Then, we will use that training data to create and Random Forest Classifier.

- i. Under the comment that reads **"Make RF classifier, train it"**, paste:

```
var classifier = ee.Classifier.smileRandomForest(100, null, 1, 0.5, null, 0)
  .setOutputMode('CLASSIFICATION')
  .train({
    features: training,
    classProperty: 'Class_num',
    inputProperties: bands
  });
```

- ii. Below you'll see the documentation for our Random Forest model. This is how we know how to set important parameters. For example, in our case, we're setting **numberOfTrees = 100**. Keep this information in mind if you want to customize your model in the future.

```
ee.Classifier.smileRandomForest(numberOfTrees, variablesPerSplit, minLeafPopulation, bagFraction, maxNodes, seed)
```

Creates an empty Random Forest classifier.

Arguments:

- **numberOfTrees (Integer):**
The number of decision trees to create.
- **variablesPerSplit (Integer, default: null):**
The number of variables per split. If unspecified, uses the square root of the number of variables.
- **minLeafPopulation (Integer, default: 1):**
Only create nodes whose training set contains at least this many points.
- **bagFraction (Float, default: 0.5):**
The fraction of input to bag per tree.
- **maxNodes (Integer, default: null):**
The maximum number of leaf nodes in each tree. If unspecified, defaults to no limit.
- **seed (Integer, default: 0):**
The randomization seed.

Returns: Classifier

2. Finally, now we're going to classify our image using the classifier we just created. Under the comment that reads **"Classify image"**, paste:

```
var classified = inStack.select(bands).classify(classifier);
```

3. As of now, we have classified an image, but we haven't added it to our map yet, so if you ran this code nothing new would print in your console or in the map (that's coming next!). Don't forget to save your work!

Part 4: Add classification to map, create a legend

A. Add final classification to map

1. Before we display the classification on our map, we must create a palette so that each class has a specific color. Under the comment that reads **"Make palette for 7 soil types"**, paste:

```
var palette = [
  '8B0000', // Barlite 1
  'FFA500', // Berrend 2
  '4169E1', // Chilimol 3
  '006400', // Marfa 4
  '808000', // Murray 5
  'BC8F8F', // Musquiz 6
  'FFFF00', // Phantom 7
];
```

- i. If you'd like to add to and/or customize this palette for future classification, visit this [GitHub site which provides color palettes for import to GEE](#).
2. Now that we've defined our palette and we know what classes and colors correspond, we can add our classification to our map. Under the comment that reads **"Display the classification"**, paste:

```
Map.addLayer(classified, {palette: palette, min: 1, max: 7}, 'classification');
```

- i. Note that our palette range goes from 1 to 7 – this is simply because we have 7 classes specified with our **"Class_num"** that has the same range.

B. Make a legend, add it to map

1. It's always useful to have a legend display on your map, especially when your dealing with a lot of classes and colors.
2. The following code may look intimidating for some, but most of it is simply creating the structure and other aesthetic details of the legend. (We'll dive into customizing the legend at a later date!). For now, just copy the following code under the comment that says **"**** Make a Legend ****"** to make a print your legend:

```
// Create the panel for the legend items.
```

```
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create and add the legend title.
var legendTitle = ui.Label({
  value: 'Legend',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
  }
});
legend.add(legendTitle);

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {
  // Create the label that is actually the colored box.
  var colorBox = ui.Label({
    style: {
      backgroundColor: '#' + color,
      // Use padding to give the box height and width.
      padding: '8px',
      margin: '0 0 4px 0'
    }
  });

  // Create the label filled with the description text.
```

```

var description = ui.Label({
  value: name,
  style: {margin: '0 0 4px 6px'}
});

return ui.Panel({
  widgets: [colorBox, description],
  layout: ui.Panel.Layout.Flow('horizontal')
});});

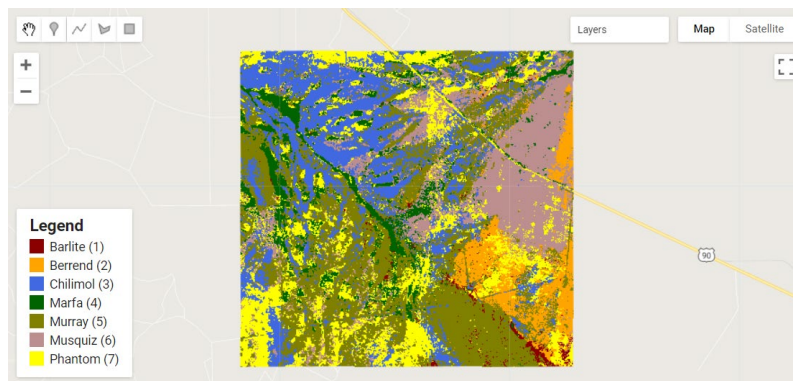
// Display names for each legend item updated here
legend.add(makeRow('8B0000', 'Barlite (1)')); //dark red
legend.add(makeRow('FFA500', 'Berrend (2)')); //orange
legend.add(makeRow('4169E1', 'Chilimol (3)')); //blue
legend.add(makeRow('006400', 'Marfa (4)')); //green
legend.add(makeRow('808000', 'Murray (5)')); //olive
legend.add(makeRow('BC8F8F', 'Musquiz (6)')); //rose brown
legend.add(makeRow('FFFF00', 'Phantom (7)')); //yellow

// Add the legend to the map.
Map.add(legend);

// Zoom to the classification on the map
Map.centerObject(compositeArea, 12);

```

3. Great job! When you run this, your classification will appear as a layer on your map, and the legend will be on the left side. Woohoo!



Part 5: Create model assessment statistics and figures

A. Perform assessment on classification

1. Next up, we're going to be doing a quick assessment on our classifier to see how well it did.
 - i. Note, this is not technically an "accuracy assessment" since we only have limited training and testing data. We can assess how well the model does relative to our training data, but we can't truly determine how "accurate" it is.
 - ii. We're going to be calculating an overall accuracy value, and a kappa value. Under the comment that reads "**Get model assessment statistics, print them**", paste:

```
var trainAccuracy = classifier.confusionMatrix();
print('Training overall accuracy:', trainAccuracy.accuracy()); // Overall
accuracy of classification based on training data
print('Training Kappa:', trainAccuracy.kappa()); //kappa value of classification
based on training data
//print('Resubstitution error matrix:', trainAccuracy); //Confusion matrix
representing substitution accuracy
```

- iii. You'll notice that we calculate a confusion matrix in order to obtain these statistics. We have commented out the bottom line, which prints the confusion matrix for this portion of the model assessment. The confusion matrix for the training data is called a "resubstitution matrix." Feel free to uncomment and investigate how this confusion matrix compares to the one prepared from the validation data.

B. Plot assessment tools

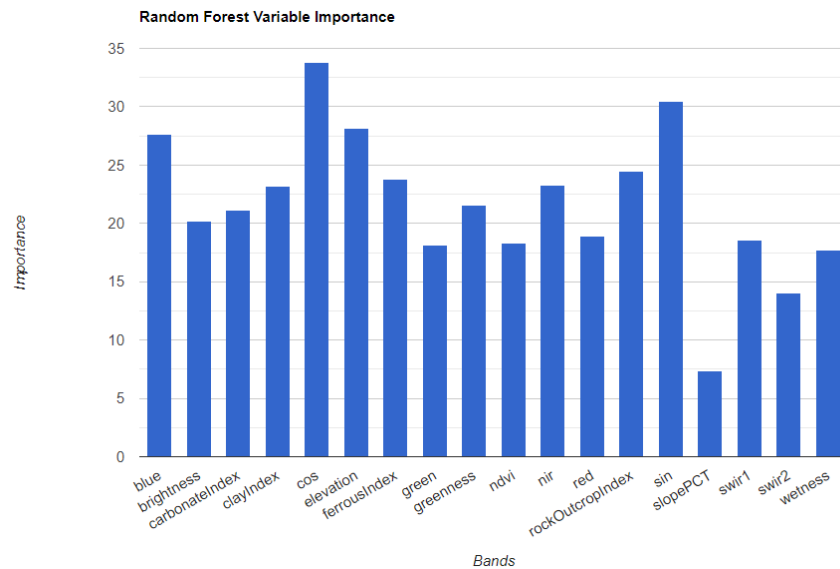
Data visualizations are an extremely important part of assessing how well a model performed and can provide a lot of insight.

1. The first plot we're going to make is a histogram of variable importance. This is a useful visual, especially when we're using more than 20 predictive layers in a model. It gives us the ability to see what variables helped the model, and what ones may not have. Under the comment that says "****** Variable Importance Histogram******", paste:
 - i. The "Classifier information" will print to the console and display values for variable importance, the number of trees in the model, numeric representations of the trees in the model, as well as the out-of-bag (OOB) error estimate. The OOB error is another way of evaluating model performance, and gives the mean error in predicting samples that were not included in a particular "bag" or decision tree.

```
// Get variable importance
var dict = classifier.explain();
print("Classifier information:", dict);
```

```
var variableImportance = ee.Feature(null, ee.Dictionary(dict).get('importance'));
// Make chart, print it
var chart =
ui.Chart.feature.byProperty(variableImportance)
.setChartType('ColumnChart')
.setOptions({
title: 'Random Forest Variable Importance',
legend: {position: 'none'},
hAxis: {title: 'Bands'},
vAxis: {title: 'Importance'}
});
print(chart);
```

ii. Once you run your code, your variable importance plot should look like this:



2. Next, we're going to make a histogram that shows how many pixels in our study area were classified to each class. As you could imagine, this is a useful visual to see if you had any dominant classes. Under the comment that reads ****** Histogram of Pixels per Class ******, paste:

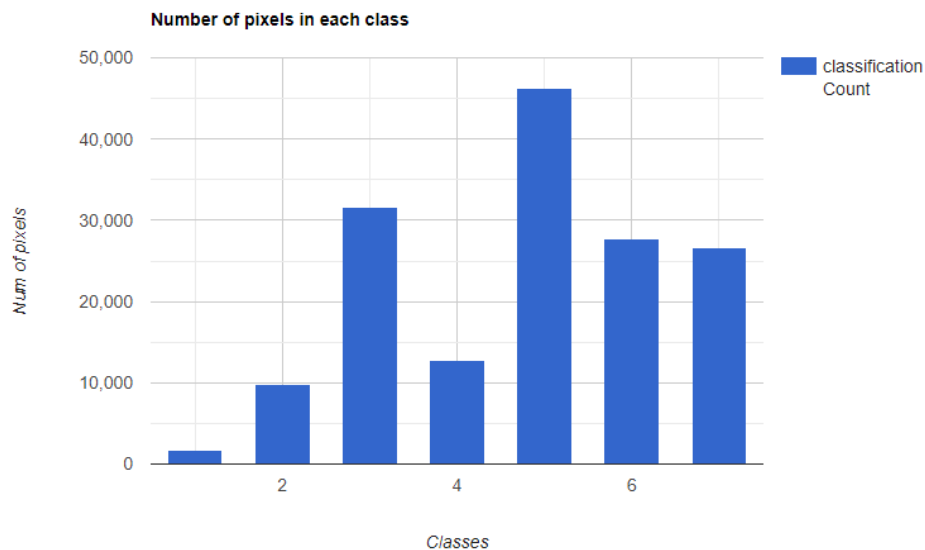
```
// Set chart options
var options = {
lineWidth: 1,
pointSize: 2,
```

```

hAxis: {title: 'Classes'},
vAxis: {title: 'Num of pixels'},
title: 'Number of pixels in each class'
};
// Get pixel chart, print it
var classPixelChart = ui.Chart.image.histogram({
  image: ee.Image(classified),
  region: compositeArea,
}).setOptions(options);
print(classPixelChart);

```

After running it, your pixel per class histogram should look like this:



Part 6: Perform validation on classification

A. Compute validation statistics

1. Now that we have our classification and performed an evaluation of the data, we should also perform a validation using the validation data we set aside earlier. This validation allows us to compare (ideally) “real” data to our classification, to see if it classified known points correctly.
2. We simply extract the predicted class at each validation point that we set aside at the beginning of our modeling exercise, and see how they compare to the observed class. Under the comment that reads “**Sample predicted classification points for validation**”, paste:

```

var validation = classified.sampleRegions({
  collection: validationData,

```

```

properties: ['Class_num'],
scale: 30,
});

```

- Then, to assess our validation, we make another confusion matrix with this data. Paste the following under the comment that reads “**Compare validation data to classification**”:

```

var testAccuracy = validation.errorMatrix('Class_num', 'classification');
print('Validation Confusion Matrix:', testAccuracy); // Confusion matrix based on
validation data
print('Validation overall accuracy:', testAccuracy.accuracy()); //Overall
accuracy of classification based on validation data
print('Validation Kappa:', testAccuracy.kappa()); //kappa value of classification
based on validation data

```

Part 7: Export

A. Export your files

- Now that you’ve created and evaluated your model, you can export it for future use—take it to ArcMap or your favorite GIS.
 - The export command you will use will depend on whether you are using Earth Engine with a personal gmail account, or a USDA account.
 - If you are using a personal gmail account, use the **Export.image.toDrive()** function.
 - If you are using a USDA account, use the **Export.image.toCloudStorage()** function, being sure to specify a cloud storage bucket.
 - Both options are provided below for you. If you wish to use the **Export.image.toCloudStorage()** function, simply delete the **/*** and ***/** before and after the code block, in order to uncomment it.
- Copy this code below the comment that reads “****** Export classification ******”.

```

// create file name for export
var exportName = roiName + '_' + 'DSM_classification';

// If using gmail: Export to Drive
Export.image.toDrive({image: classified,
  description: exportName,
  folder: "DigitalSoilMapping",
  fileNamePrefix: exportName,
  region: compositeArea,

```



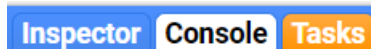
```

scale: 30,
crs: crs,
maxPixels: 1e13});

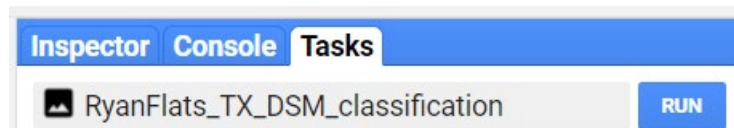
/*
// If using USDA acct: Export to Cloud Storage
// Export loss layers to cloud storage
Export.image.toCloudStorage({image: classified,
    description: exportName,
    bucket: cloudStorageBucket,           // update with name of Cloud
Storage Bucket here
    fileNamePrefix: exportName,
    region: compositeArea,
    scale: 30,
    crs: crs,
    maxPixels: 1e13});
*/

```

3. After you run the script, the Tasks tab on the right side of the pane will turn orange, indicating that there are export tasks that can be run.



4. Click on the Tasks tab.
5. Locate the appropriate task in the pane and click "Run".



6. In the window that pops up, you will see the export parameters. We have already specified these in our script. Check to make sure everything looks ok, and click "Run." The export may take upwards of 10 minutes to complete, so be patient!

Task: Initiate image export ✕

Task name (no spaces) *

Resolution *

Scale (m/px) ▾ 30

Drive
 Cloud Storage
 EE Asset

Drive folder

Filename *

7. Notice that we are sending the export to a folder in your Google Drive called “DigitalSoilMapping.” If this folder doesn’t already exist, this command will create it.
8. Navigate to your google drive, locate the DigitalSoilMapping folder, and click to open it.
9. Right click to download the file, which should be titled “RyanFlats_TX_DSM_classification.tif”.
Now, you can open this up in your GIS of choice.

Part 8: Discussion

A. Get thinking!

1. Now that we’ve completed our random forest classification and have run our final script, we should assess our results.
 - i. Looking through the console at the figures and statistics, is anything catching your attention? Are you surprised by any of your results?
 - ii. Observe your mapped final classification – is anything standing out? Would you say this is a “good” classification? What can you do to improve it?
2. These are just a few questions to get the gears moving!

Congratulations! You have successfully completed this exercise. You have used a variety of techniques to perform a soil classification using random forests in Google Earth Engine.