Lab 06 – Supervised Classification

Image classification is the task of categorizing multiband raster images into a fixed set of information classes to create a thematic map (1)(2). Images, by themselves, do not explicitly convey information. We can extract information from imagery visually based on our own experience, but the visual process is subjective and therefore difficult to convey the same information in a standardized way to someone else. Further, standardized categories of land cover extracted from imagery offer more help to land managers, and scientists as they work to manage and understand landscapes.

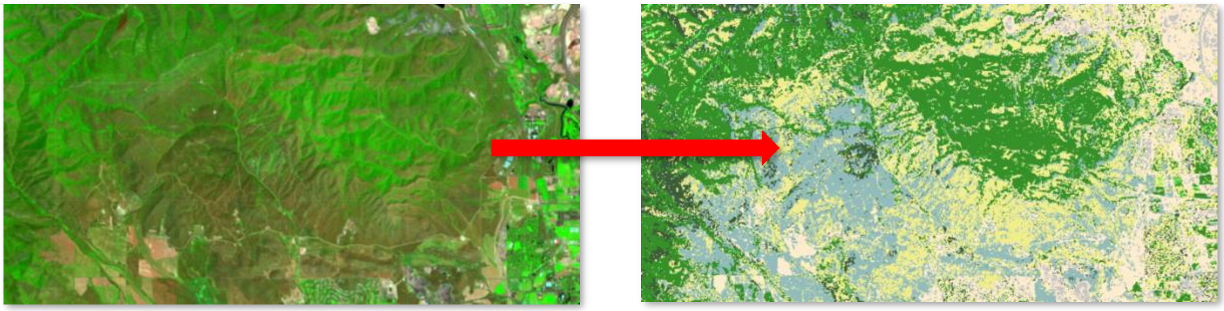


Image classification falls under the general category of machine learning and/or computer vision and is used extensively in the world of facial recognition, conversion of scanned text to real text, etc. In our world, image classification is used to map land cover types into categories that best fit our land management objectives.

We will discuss various classification techniques in class. As far as lab is concerned, we will focus on the tools available in Google Earth Engine. GEE has a number of classification algorithms representing the latest techniques in classification methodology. These techniques focus on the use of Decision Trees, Support Vector Machine (SVM), Bayesian (probabilistic), and Minimum Distance classifiers. Here are the different classification methods within GEE:

ee.Classifier.decisionTree
ee.Classifier.decisionTreeEnsemble
ee.Classifier.gmoMaxEnt
ee.Classifier.libsvm
ee.Classifier.minimumDistance
ee.Classifier.smileCart
ee.Classifier.smileNaiveBayes
ee.Classifier.smileRandomForest

The general workflow for classification is:

1. Collect training data. Assemble features that stores the known class label as well as numeric values for the predictors.
2. Train the classifier using the training data.
3. Apply the classifier to an image.
4. Estimate classification error with independent validation data.

In order to apply this general workflow, we must develop a script in GEE that will bring the data together, train the classifier, apply the classifier, and then evaluate accuracy.

Here is the "pseudo-code", or the process flow for our script. Work on each step until you get it working the way you want it to work and then move onto the next:

1. **Obtain training data**. These are points or polygons stored in a GEE feature collection that contains the geographic locations and identification of the land cover types we wish to map. I will provide a shapefile that you can upload into your GEE account as an "**Asset**". This shapefile consists of points that I placed on a high-resolution image and then visually interpreted the land cover from that image. The categories that we will map include:

   ['Bare', 'Deciduous Tree', 'Developed', 'Gambel Oak', 'Grassland', 'Juniper', 'Sagebrush']

2. **Identify and load imagery**. The remote sensing instrument that you will use is up to you. However, some things to remember are that the imagery should contain the spectral data that is appropriate to map land cover, but most importantly, the images have to have been collected within the same time frame as your training data. Further, the imagery should represent the most appropriate time of year (spring/summer/fall, etc.) to map your categories.

   a. **Mask clouds from imagery**

   b. **Reduce the collection to a specific area.** I used the bounding area of my training data. You can do this by displaying the training data and then drawing a rectangle to enclose all of the points.

   c. **Select appropriate bands (the QA bands are prob. not appropriate.)** For Landsat 8, for instance, I would choose bands 1-7 since they represent the reflected spectral bands.

   d. **Reduce the image collection to a single cloud-free image using the .median() reducer.**

3. **Intersect training data points with the image.** The training data that I provided includes only four attributes: PointID, CoordX, CoordY, and Type. The two coordinates are eastings and northings of each point in NAD83, UTM, Zone 12. The Type attribute is the attribute that holds the land cover type that I interpreted for each point. We need to intersect those points with the image so that the spectral values for each of those points are transferred to the attribute table. Our classifier will use the spectral responses in each band as the predictor variables and the "Type" attribute as the dependent variable.

   Here's the bit of code that intersects the points with the image using the command sampleRegions:

```
// Overlay the points on the imagery to get training data.
var training = image.sampleRegions({
  collection: points,
  properties: ['GridCode'],
  scale: 30
});
print('Training Data',training)
```

4. **Train the classifier.** There are a number of classifiers to choose from. I will use ee.Classifier.smileRandomForest. Look under the Docs tab in the ee.Classifier menu to become acquainted with this classifier. I've also added a few references below to help you understand how the classifiers work. Here's the classifier code snippet:

```
// Train a CART classifier with default parameters.
var classif = ee.Classifier.smileRandomForest({
          numberOfTrees: 5})
           .setOutputMode('CLASSIFICATION')
           .train({
             features: training,
             classProperty: 'GridCode',
             inputProperties: bands,
             subsampling: 1,
             subsamplingSeed: 0
                 });
```

Here’s a bit of code that prints out results of the classifier so we can evaluate what is going on. The output of the classifier is not an image or a feature collection, it’s a GEE object.

```
//##################################################################################################
//#################### PRINT THE RF RESULTS ##########################################
print('Confusion Matrix',classif.confusionMatrix());

var explain = classif.explain();
print('Explain',explain);

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

5. **Classify the image.** After the classifier has been trained, we need to apply it to the image. Be aware that there are two steps here. 1) train the classifier with discrete samples, and 2) apply the classifier to every pixel in the image.

```
// Classify the image with the same bands used for training.
var classified = image.classify(classif);
print('Classified',classified)
```

6. **Display results** – you know how to do this already

7. **Evaluate the classification accuracy**. This is done two ways and we will talk about accuracy assessment in more detail in class. Essentially, the accuracy is determined by using the classified image in the field, and/or comparing the map to a set of independent locations to see if what is on the ground was mapped correctly on the image. We don’t have that data yet, so another technique is to use the classifier (smileRandomForest) to do it for us. The Random Forest classifier uses an “ensemble learning method”. This essentially means that it uses the available training data to

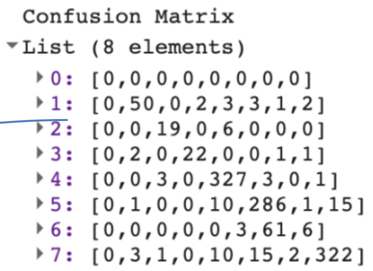
generate a number of trees using subsets of the input training data. These multiple trees are then used to “make a group decision” about the most probable outcome given the input data.

Since each tree uses a subset of the input data, the data that is not used for that particular tree can be used to estimate the error for that tree by testing the tree output against that unused data to see if it can accurately predict it.

Remember that in the code snippet above, I set the **numberOfTrees** to 5 (I just chose that out of the air, you can change it). These 5 trees were then used to “vote” for the most appropriate outcome. The smileRandomForest code also generates a table which sums up the number of times the trees voted correctly and incorrectly on a specific outcome. This table is called a “**Confusion Matrix**”. My code above printed the confusion matrix to the console in the form of a list. If you open that list, there should be 8 elements (the first is 0 which we don’t care about). Elements 1-7 correspond to the grid code values that represent the land cover types:

1. Bare
2. Deciduous Tree
3. Developed
4. Gambel Oak
5. Grassland
6. Juniper
7. Sagebrush

You should have something like this:



```
Confusion Matrix
▾List (8 elements)
  ▸0: [0,0,0,0,0,0,0,0]
  ▸1: [0,50,0,2,3,3,1,2]
  ▸2: [0,0,19,0,6,0,0,0]
  ▸3: [0,2,0,22,0,0,1,1]
  ▸4: [0,0,3,0,327,3,0,1]
  ▸5: [0,1,0,0,10,286,1,15]
  ▸6: [0,0,0,0,0,3,61,6]
  ▸7: [0,3,1,0,10,15,2,322]
```

Which, if copied, pasted into Excel and reformatted we should have this (I prettied it up and added a few things):

Actual

			Bare	**Decid. Trees**	**Devel**	**Gambel Oak**	**Grassl**	**Junip**	**Sage**			
			1	**2**	**3**	**4**	**5**	**6**	**7**	**Total**	**Users Acc.**	
Predicted	**Bare**	**1**	**50**	0	2	3	3	1	2	**61**	82.0%	
	Decid. Trees	**2**	0	**19**	0	6	0	0	0	**25**	76.0%	
	Developed	**3**	2	0	**22**	0	0	1	1	**26**	84.6%	
	Gambel Oak	**4**	0	3	0	**327**	3	0	1	**334**	97.9%	
	Grassland	**5**	1	0	0	10	**286**	1	15	**313**	91.4%	
	Juniper	**6**	0	0	0	0	3	**61**	6	**70**	87.1%	
	Sagebrush	**7**	3	1	0	10	15	2	**322**	**353**	91.2%	
		Total	**56**	**23**	**24**	**356**	**310**	**66**	**347**	**1182**		**Overall Acc.**
	Producers Accuracy		89.3%	82.6%	91.7%	91.9%	92.3%	92.4%	92.8%	Sum. Diag. =	1087	**92.0%**

Q1. Write your script and send me the link.

Q2. Copy the confusion matrix to Excel and reformat it like I did above.

Q3. What is the overall accuracy?

Q4. What types had the lowest accuracy and how and why do you think they were confused?

References:

1. https://cs231n.github.io/classification/
2. https://desktop.arcgis.com/en/arcmap/latest/extensions/spatial-analyst/image-classification/what-is-image-classification-.htm
3. https://developers.google.com/earth-engine/guides/classification
4. https://sites.google.com/site/mcgillgis2paulauskasmichael/projet-final/methods-3-gee
5. https://en.wikipedia.org/wiki/Confusion_matrix
6. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
7. http://web.pdx.edu/~nauna/resources/9-accuracyassessment.pdf