Lecture 07
GEE Image Classification:
supervised & unsupervised classification

2024-04-15

Sébastien Valade

Universidad Nacional Autónoma de México

Previous lecture:

GEE image manipulation:

⇒ band arithmetic (spectral indices), thresholds, masks, reducers

Today:

GEE image classification:

⇒ assign a *class* to each pixel in the image

⇒ supervised & unsupervised learning

Previous lecture:

GEE image manipulation:

$\Rightarrow$ band arithmetic (spectral indices), thresholds, masks, reducers

Today:

GEE image classification:

$\Rightarrow$ assign a *class* to each pixel in the image

$\Rightarrow$ supervised & unsupervised learning

Image Classification:

- ⇒ **Image classification** in remote sensing, is a task that involves categorizing all pixels in an image into a finite number of **classes**
- ⇒ its most common application is *land use & land cover (LULC)* classification, whereby all pixels are categorized in predefined land cover classes (e.g., water, forest, urban, etc.)

Image Classification:

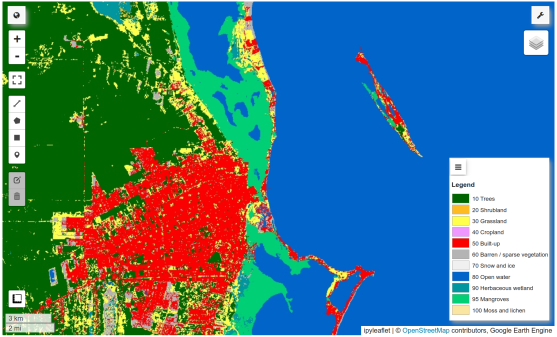
⇒ **Image classification** in remote sensing, is a task that involves categorizing all pixels in an image into a finite number of **classes**

⇒ its most common application is *land use & land cover (LULC)* classification, whereby all pixels are categorized in predefined land cover classes (e.g., water, forest, urban, etc.)

Natural color image

→

Land cover classification (ESA World Cover)



⇒ Several techniques exist to classify the image (Li et al. 2014, Kavzoglu et al. 2009 (ed2), 2025 (ed3)):
- **Pixel-based techniques**: classify each pixel individually based on spectral properties
- **Object-based techniques**: classify groups of pixels (objects) based on spectral, spatial, and cor
- **Deep Learning techniques**: use neural networks to learn features and classify images

⇒ These techniques generally fall in the broad field of **Machine Learning**, whereby *statistical algorithms are able to learn from data, and generalize to unseen data*, thus performing tasks (i.e., classification) without explicit instructions

⇒ The learning approach can be either **supervised** or **unsupervised**:
- **Supervised classification**: requires *training data* (labeled data)
- **Unsupervised classification**: does *not* require *training data*

⇒ In this lecture, we will focus on the *implementation in GEE* of **pixel-based classification** using both **supervised and unsupervised** techniques

⇒ Several techniques exist to classify the image (Li et al. 2014, Kavzoglu et al. 2009 (ed2), 2025 (ed3)):

- **Pixel-based techniques**: classify each pixel individually based on spectral properties
- **Object-based techniques**: classify groups of pixels (objects) based on spectral, spatial, and cor
- **Deep Learning techniques**: use neural networks to learn features and classify images

⇒ These techniques generally fall in the broad field of **Machine Learning**, whereby *statistical algorithms are able to learn from data, and generalize to unseen data*, thus performing tasks (i.e., classification) without explicit instructions

⇒ The learning approach can be either **supervised** or **unsupervised**:

- **Supervised classification**: requires *training data* (labeled data)
- **Unsupervised classification**: does *not* require *training data*

⇒ In this lecture, we will focus on the *implementation in GEE* of **pixel-based classification** using both **supervised and unsupervised** techniques

1.1. Image classification overview

⇒ Several techniques exist to classify the image (Li et al. 2014, Kavzoglu et al. 2009 (ed2), 2025 (ed3)):

- **Pixel-based techniques**: classify each pixel individually based on spectral properties
- **Object-based techniques**: classify groups of pixels (objects) based on spectral, spatial, and cor
- **Deep Learning techniques**: use neural networks to learn features and classify images

⇒ These techniques generally fall in the broad field of **Machine Learning**, whereby *statistical algorithms are able to learn from data, and generalize to unseen data*, thus performing tasks (i.e., classification) without explicit instructions

⇒ The learning approach can be either **supervised** or **unsupervised**:

- **Supervised classification**: requires *training data* (labeled data)
- **Unsupervised classification**: does *not* require *training data*

⇒ In this lecture, we will focus on the *implementation in GEE* of **pixel-based classification** using both **supervised and unsupervised** techniques

⇒ Several techniques exist to classify the image (Li et al. 2014, Kavzoglu et al. 2009 (ed2), 2025 (ed3)):

- **Pixel-based techniques**: classify each pixel individually based on spectral properties
- **Object-based techniques**: classify groups of pixels (objects) based on spectral, spatial, and cor
- **Deep Learning techniques**: use neural networks to learn features and classify images

⇒ These techniques generally fall in the broad field of **Machine Learning**, whereby *statistical algorithms are able to learn from data, and generalize to unseen data*, thus performing tasks (i.e., classification) without explicit instructions

⇒ The learning approach can be either **supervised** or **unsupervised**:

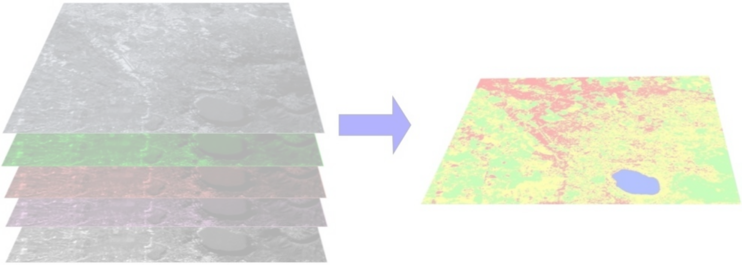
- **Supervised classification**: requires *training data* (labeled data)
- **Unsupervised classification**: does *not* require *training data*

⇒ In this lecture, we will focus on the *implementation in GEE* of **pixel-based classification** using both **supervised and unsupervised** techniques

Pixel-based classification:

⇒ With the pixel-based classification method, pixels are classified *individually*, i.e. without spatial context from the neighboring pixels

⇒ Classification is based on the *pixel spectral properties* (i.e., reflectance values in each band), and/or on their transformations (i.e., spectral indices, principal components, etc.)



<u>Pixel-based classification</u>:

⇒ With the pixel-based classification method, pixels are classified *individually*, i.e. without spatial context from the neighboring pixels

⇒ Classification is based on the *pixel spectral properties* (i.e., reflectance values in each band), and/or on their transformations (i.e., spectral indices, principal components, etc.)

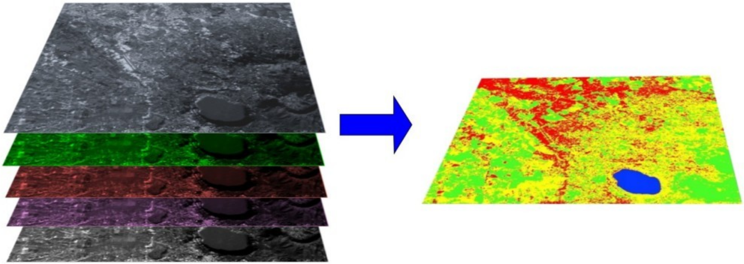
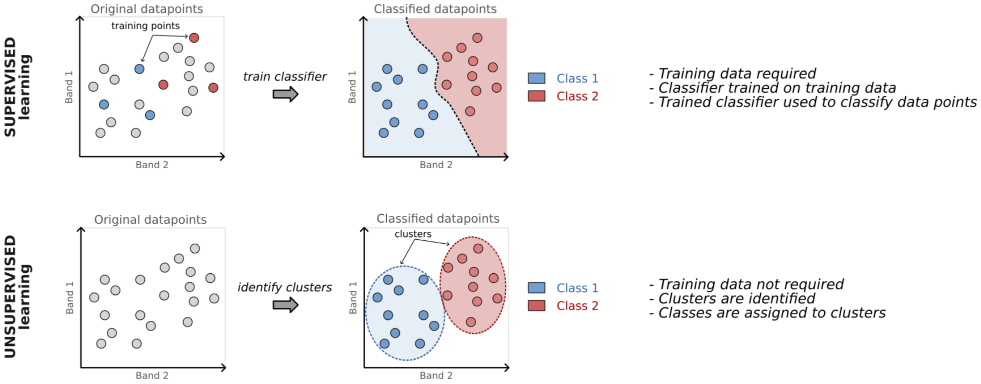


Image Source: link

Pixel-based classification:

⇒ **Supervised** vs. **Unsupervised** learning:



Workflow of supervised classification

1. **Select image to classify**

2. **Collect training data**
   ⇒ a *training dataset* is collection of *labeled data*, that is input-output pairs, where the *input* is the data provided to the model, and the *output* is the corresponding target or label that the model is expected to predict.

3. **Select prediction bands**
   ⇒ *prediction bands* correspond to the bands used to extract the spectral information to classify each pixels.

   EX: use the bands provided in the product (including bands in the visible range, and possibly in the infrared range), and possibly derived bands (e.g., spectral indices, or principal components derived from a PCA analysis)

4. **Select a classifier and train it on the training data**
   ⇒ a *classifier* is a *statistical model* that learns to map input pixels to output classes based on the provided labels.

   EX: commonly used classifiers are Random Forest, Support Vector Machine, K-Nearest Neighbors, CART, etc.

5. **Classify the image using the *trained classifier***
   The trained classifier is applied to the image, and each pixel is assigned a class based on the classifier's prediction.

Workflow of supervised classification

1. **Select image to classify**

2. **Collect training data**
   ⇒ a *training dataset* is collection of *labeled data*, that is input-output pairs, where the *input* is the data provided to the model, and the *output* is the corresponding target or label that the model is expected to predict.

3. **Select prediction bands**
   ⇒ *prediction bands* correspond to the bands used to extract the spectral information to classify each pixels.

   EX: use the bands provided in the product (including bands in the visible range, and possibly in the infrared range), and possibly derived bands (e.g., spectral indices, or principal components derived from a PCA analysis)

4. **Select a classifier and train it on the training data**
   ⇒ a *classifier* is a *statistical model* that learns to map input pixels to output classes based on the provided labels.

   EX: commonly used classifiers are Random Forest, Support Vector Machine, K-Nearest Neighbors, CART, etc.

5. **Classify the image using the *trained classifier***
   The trained classifier is applied to the image, and each pixel is assigned a class based on the classifier's prediction.

Workflow of supervised classification

1. **Select image to classify**

2. **Collect training data**
   ⇒ a *training dataset* is collection of *labeled data*, that is input-output pairs, where the *input* is the data provided to the model, and the *output* is the corresponding target or label that the model is expected to predict.

3. **Select prediction bands**
   ⇒ *prediction bands* correspond to the bands used to extract the spectral information to classify each pixels.

   EX: use the bands provided in the product (including bands in the visible range, and possibly in the infrared range), and possibly derived bands (e.g., spectral indices, or principal components derived from a PCA analysis)

4. **Select a classifier and train it on the training data**
   ⇒ a *classifier* is a *statistical model* that learns to map input pixels to output classes based on the provided labels.

   EX: commonly used classifiers are Random Forest, Support Vector Machine, K-Nearest Neighbors, CART, etc.

5. **Classify the image using the *trained classifier***
   The trained classifier is applied to the image, and each pixel is assigned a class based on the classifier's prediction.

Workflow of supervised classification

1. **Select image to classify**

2. **Collect training data**

   ⇒ a *training dataset* is collection of *labeled data*, that is input-output pairs, where the *input* is the data provided to the model, and the *output* is the corresponding target or label that the model is expected to predict.

3. **Select prediction bands**

   ⇒ *prediction bands* correspond to the bands used to extract the spectral information to classify each pixels.

   EX: use the bands provided in the product (including bands in the visible range, and possibly in the infrared range), and possibly derived bands (e.g., spectral indices, or principal components derived from a PCA analysis)

4. **Select a classifier and train it on the training data**

   ⇒ a *classifier* is a *statistical model* that learns to map input pixels to output classes based on the provided labels.

   EX: commonly used classifiers are Random Forest, Support Vector Machine, K-Nearest Neighbors, CART, etc.

5. **Classify the image using the *trained classifier***

   The trained classifier is applied to the image, and each pixel is assigned a class based on the classifier's prediction.

Workflow of supervised classification

1. **Select image to classify**

2. **Collect training data**
   ⇒ a *training dataset* is collection of *labeled data*, that is input-output pairs, where the *input* is the data provided to the model, and the *output* is the corresponding target or label that the model is expected to predict.

3. **Select prediction bands**
   ⇒ *prediction bands* correspond to the bands used to extract the spectral information to classify each pixels.

   EX: use the bands provided in the product (including bands in the visible range, and possibly in the infrared range), and possibly derived bands (e.g., spectral indices, or principal components derived from a PCA analysis)

4. **Select a classifier and train it on the training data**
   ⇒ a *classifier* is a *statistical model* that learns to map input pixels to output classes based on the provided labels.

   EX: commonly used classifiers are Random Forest, Support Vector Machine, K-Nearest Neighbors, CART, etc.

5. **Classify the image using the *trained classifier***
   The trained classifier is applied to the image, and each pixel is assigned a class based on the classifier's prediction.

Step 1: select the image to classify

```python
# Select region of interest (lon, lat)
roi = ee.Geometry.Point(-86.85, 21.17)

# Filter the Sentinel-2 collection and select the least cloudy image
image = (ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED')
            .filterBounds(roi)
            .filterDate('2020-01-01', '2024-10-01')
            .filter(ee.Filter.lt('CLOUD_COVERAGE_ASSESSMENT', 10))
            .sort("CLOUD_COVERAGE_ASSESSMENT")
            .first()
            )
```

Step 2: collect training samples

NB: the built-in tool in `GEEMAP` called "Collect training samples" is suffering bugs in the Google Colab environment (in particular, it does not store the "property" & "value" fields in the `user_rois` object). The approach suggested below is a workaround to collect training samples.

⇒ For each *land cover class* (see table below), repeat the following:

1. *Select* training pixels using the `Draw a marker` tool on the interactive map
2. *Convert* the collected samples (`Map.user_rois`) to a `GeoPandasDataframe`, and create a new column called "class" storing the numeric value corresponding to the land cover class
3. *Export* the `GeoPandasDataframe` to a `GeoJson` file on your Google Drive for future use
4. *Combine* all collected samples in a unique `FeatureCollection`: once steps 1-3 have been performed for each class, *import* all `GeoJson` files and combine in a unique `FeatureCollection`, which will store all the collected samples along with their class

⇒ In this example, we will be using the following *land cover classes* (feel free to adapt to your image):

Class	*Description*
0	Vegetation
1	Urban
2	Water
3	Grassland

Step 2: collect training samples

NB: the built-in tool in `GEEMAP` called "Collect training samples" is suffering bugs in the Google Colab environment (in particular, it does not store the "property" & "value" fields in the `user_rois` object). The approach suggested below is a workaround to collect training samples.

⇒ For each *land cover class* (see table below), repeat the following:

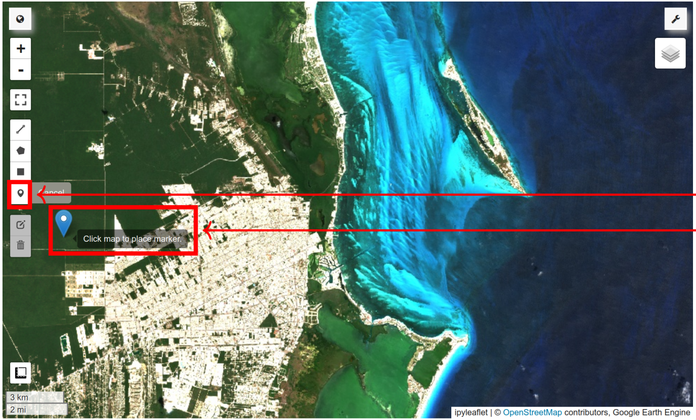
1. *Select* training pixels using the `Draw a marker` tool on the interactive map
2. *Convert* the collected samples (`Map.user_rois`) to a `GeoPandasDataframe`, and create a new column called "class" storing the numeric value corresponding to the land cover class
3. *Export* the `GeoPandasDataframe` to a `GeoJson` file on your Google Drive for future use
4. *Combine* all collected samples in a unique `FeatureCollection`: once steps 1-3 have been performed for each class, *import* all `GeoJson` files and combine in a unique `FeatureCollection`, which will store all the collected samples along with their class

⇒ In this example, we will be using the following *land cover classes* (feel free to adapt to your image):

Class	*Description*
0	Vegetation
1	Urban
2	Water
3	Grassland

Step 2: collect training samples

1. *Select* training pixels using the `Draw a marker` tool on the interactive map



a. select tool "Draw a marker"

b. pick on pixel corresponding to the "land cover class" (e.g., vegetation)

Step 2: collect training samples

2. *Convert* the collected samples (`Map.user_rois`) to a `GeoPandasDataframe`, and create a new column called "class" storing the numeric value corresponding to the land cover class

```
# === Convert collected training samples to GeoPandasDataframe

# Set class of the collected smaples
class_value, class_label = 0, 'vegetation'

# Convert ROIs to geodataframe
gdf = geemap.ee_to_gdf(Map.user_rois)

# Add class column (will be saved in "properties" field in json)
gdf['class'] = class_value
```

Step 2: collect training samples

3. *Export* the GeoPandasDataframe to a GeoJson file on your Google Drive for future use

NB: to mount your Google Drive in the Google Colab jupyter environment, either run the command below, or click on "Folder" icon in the vertical toolbar on the left-hand side of the screen, then click on the icon representing a folder w the Google Drive logo inside.

```
# Mount Google Drive programmatically
from google.colab import drive
drive.mount('/content/drive')

# Export GeoDataFrame to GeoJson
p_geojson = '/content/drive/MyDrive/training_samples/'
f_geojson = 'training_samples_class-{}_{}.geojson'.format(class_value, class_label)
gdf.to_file(p_geojson+f_geojson, driver='GeoJSON')
```

Step 2: collect training samples

4. *Combine* all collected samples in a unique `FeatureCollection`: once steps 1-3 have been performed for each class, *import* all `GeoJson` files and combine in a unique `FeatureCollection`, which will store all the collected samples along with their class

```python
# Import GeoJson as individual feature collections
p_geojson = '/content/drive/MyDrive/training_samples/'
fc_vegetation = geemap.geojson_to_ee(p_geojson+'training_samples_class-0_vegetation.geojson')
fc_urban = geemap.geojson_to_ee(p_geojson+'training_samples_class-1_urban.geojson')
fc_water = geemap.geojson_to_ee(p_geojson+'training_samples_class-2_water.geojson')
fc_grass = geemap.geojson_to_ee(p_geojson+'training_samples_class-3_grass.geojson')

# Display the FeatureCollections (optional)
Map.addLayer(fc_vegetation, {'color':'green'}, "vegetation")
Map.addLayer(fc_urban, {'color':'red'}, "urban")
Map.addLayer(fc_water, {'color':'blue'}, "water")
Map.addLayer(fc_grass, {'color':'orange'}, "grass")

# Combine all feature collections in a unique training FeatureCollection
fc_trainingSamples = (ee.FeatureCollection([
    fc_vegetation, fc_urban, fc_water, fc_grass
    ]).flatten())
```

Step 2: collect training samples

4. *Combine* all collected samples in a unique `FeatureCollection`: once steps 1-3 have been performed for each class, *import* all `GeoJson` files and combine in a unique `FeatureCollection`, which will store all the collected samples along with their class

Collected training samples

Step 3: select prediction bands

⇒ *Select the prediction bands* to be used for the classification. In the example below, we only use bands from the Sentinel-2 product, but you could also use derived bands (e.g., spectral indices, principal components, etc.)

⇒ *Sample the training data points* using these bands: use the sampleRegions method to sample the bands information at each training data point

NB: use display(fc_classifierTraining) to visualize the content of the object: by expanding the first feature (i.e. first training data point), you should see 1) the data of the prediction bands for that pixel, and 2) the class information you assigned to it.

```
# Select prediction bands (Sentinel-2 product)
predictionBands = [
    'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B8A', 'B9', 'B11', 'B12'
]

# Sample prediction bands at each training point
fc_classifierTraining = (image.select(predictionBands)
                          .sampleRegions(
                              collection=fc_trainingSamples,
                              properties=['class'],
                              scale=20
                          ))
```

Step 4: select and train the classifier

⇒ The classifier is a *statistical model* which contains mathematical rules linking the pixel's spectral information to its class

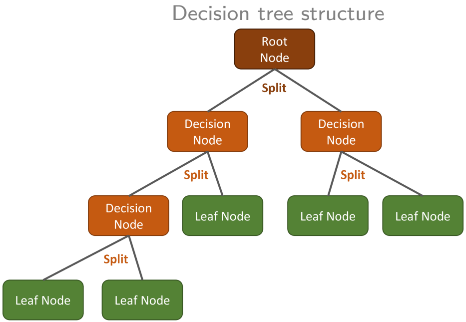
⇒ Selecting the appropriate classifier can be tricky. Various classifiers are available in GEE, e.g.: CART (Classification and Regression Trees), Random Forest, Naive Bayes, etc.

NB: the classifiers in GEE have names starting with "smile" (`ee.Classifier.smileCart()`), which stands for "Statistical Machine Intelligence and Learning Engine".

Step 4: select and train the classifier

⇒ In the following example, we use the CART classifier (Classification and Regression Trees), which is a *Decision Tree Method* introduced by Breiman et al. in 1984.

⇒ the CART algorithm *recursively splits* the data into subsets based on the most important feature (using e.g. the *Gini Index*). (See FU-Berlin for more details, from where the images below are taken).

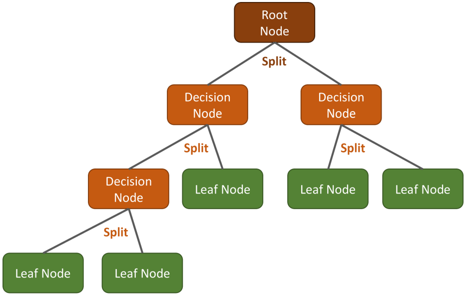


Step 4: select and train the classifier

⇒ In the following example, we use the CART classifier (Classification and Regression Trees), which is a *Decision Tree Method* introduced by Breiman et al. in 1984.

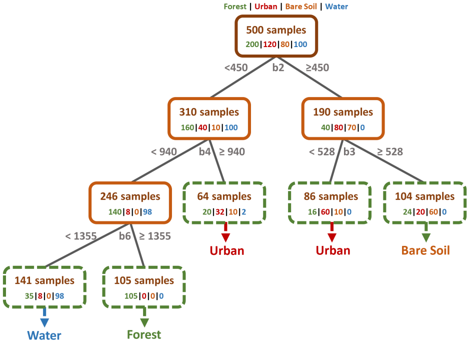
⇒ the CART algorithm *recursively splits* the data into subsets based on the most important feature (using e.g. the *Gini Index*). (See FU-Berlin for more details, from where the images below are taken).

Decision tree structure



→

Decision tree applied to land classification of satellite image



Step 4: select and train the classifier

⇒ In the following example, we use the CART classifier (Classification and Regression Trees), which is a *Decision Tree Method* introduced by Breiman et al. in 1984.

⇒ the CART algorithm *recursively splits* the data into subsets based on the most important feature (using e.g. the *Gini Index*). (See FU-Berlin for more details, from where the images below are taken).

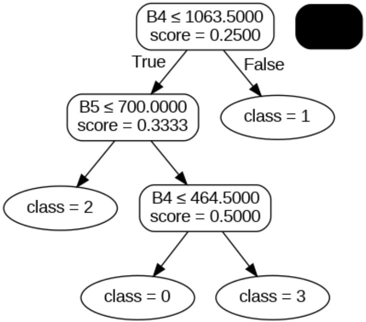
⇒ implementation in GEE:

```
# Instantiate CART classifier
classifier = ee.Classifier.smileCart()

# Train classifier
classifier = classifier.train(features=fc_classifierTraining,
                              classProperty='class',
                              inputProperties=predictionBands)
```

Step 4: select and train the classifier

⇒ Optional: you can use the command `display(classifier)` to display the basic characteristics of the classifier (bands, properties, and classifier name), and the command `classifier.explain()` to describe the results of the trained classifier (e.g., display decision rules of the trained classifier, plot as a *dot graph*, etc.)

Dot graph of trained CART decision rules




```python
import pydotplus
from IPython.display import Image

# Describe the results of the trained classifier
classifier_explained = classifier.explain()

# Plot decision tree
dot_data = classifier_explained.get('dot').getInfo()
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Step 5: classify the image using the trained classifier

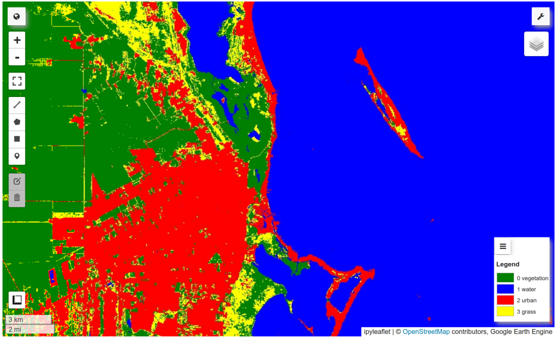
⇒ once the classifier has been trained, you can use it to predict the class of each pixel in the image

```
# Classify the image using the trained classifier
image_classified = image.select(predictionBands).classify(classifier)

# Visualize the classified image and add custom legend to the map
vis_params = {'min': 0, 'max': 3, 'palette': ['green', 'red', 'blue', 'yellow']}
Map.addLayer(image_classified, vis_params, 'Land Cover Classification (CART)')

legend_dict = {
    '0 vegetation': '008000',    # 'green' hex code
    '1 water': '0000FF',         # 'blue' hex code
    '2 urban': 'FF0000',         # 'red' hex code
    '3 grass': 'FFFF00'          # 'yellow' hex code
}
Map.add_legend(legend_title="Land Cover Classification", legend_dict=legend_dict)
Map
```

Supervised classification results

Natural color image

Land cover classification (CART)



Supervised classification results

⇒ Unsatisfied with the results? Here are some options to improve the classification:

- **Training data**: increase number of training points to have a more representative sampling of the classes
- **Predictors**: add spectral indices to the "prediction bands"
  EX: adding the NDVI index (dedicated to quantifying vegetation health) is likely to improve the common misclassification between grass and urban classes.
- **Classifier hyperparameter**: model "hyperparameters" are set to default values, but can be tuned (e.g. for classification trees, you can tune the *number of leaves* in the trees)
- **Classifier selection**: try using a different classifier
  EX: the Random Forest (RF) algorithm (Breiman 2001) builds on the concept of decision trees in the CART algorithm, by constructing multiple decision trees (hence the term "forest"), making it more powerful

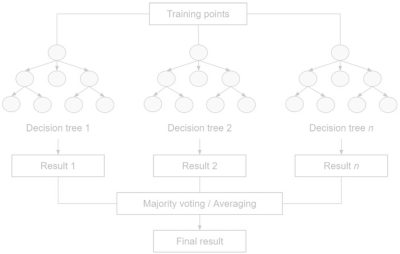


Image Source: Cardille et al. 2024

Supervised classification results

⇒ Unsatisfied with the results? Here are some options to improve the classification:

- **Training data**: increase number of training points to have a more representative sampling of the classes
- **Predictors**: add spectral indices to the "prediction bands"
  EX: adding the NDVI index (dedicated to quantifying vegetation health) is likely to improve the common misclassification between grass and urban classes.
- **Classifier hyperparameter**: model "hyperparameters" are set to default values, but can be tuned (e.g. for classification trees, you can tune the *number of leaves* in the trees)
- **Classifier selection**: try using a different classifier
  EX: the Random Forest (RF) algorithm (Breiman 2001) builds on the concept of decision trees in the CART algorithm, by constructing multiple decision trees (hence the term "forest"), making it more powerful

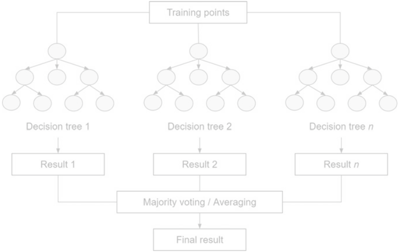


Image Source: Cardille et al. 2024

Supervised classification results

⇒ Unsatisfied with the results? Here are some options to improve the classification:

- **Training data**: increase number of training points to have a more representative sampling of the classes
- **Predictors**: add spectral indices to the "prediction bands"
  EX: adding the NDVI index (dedicated to quantifying vegetation health) is likely to improve the common misclassification between grass and urban classes.
- **Classifier hyperparameter**: model "hyperparameters" are set to default values, but can be tuned (e.g. for classification trees, you can tune the *number of leaves* in the trees)
- **Classifier selection**: try using a different classifier
  EX: the Random Forest (RF) algorithm (Breiman 2001) builds on the concept of decision trees in the CART algorithm, by constructing multiple decision trees (hence the term "forest"), making it more powerful

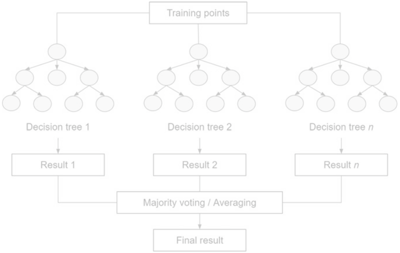


Image Source: Cardille et al. 2024

Supervised classification results

⇒ Unsatisfied with the results? Here are some options to improve the classification:

- **Training data**: increase number of training points to have a more representative sampling of the classes
- **Predictors**: add spectral indices to the "prediction bands"
  EX: adding the NDVI index (dedicated to quantifying vegetation health) is likely to improve the common misclassification between grass and urban classes.
- **Classifier hyperparameter**: model "hyperparameters" are set to default values, but can be tuned (e.g. for classification trees, you can tune the *number of leaves* in the trees)
- **Classifier selection**: try using a different classifier
  EX: the Random Forest (RF) algorithm (Breiman 2001) builds on the concept of decision trees in the CART algorithm, by constructing multiple decision trees (hence the term "forest"), making it more powerful

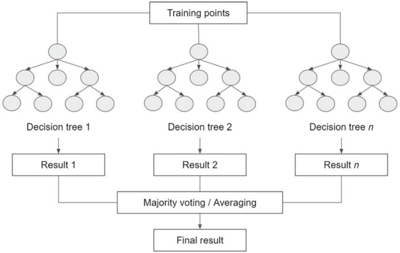
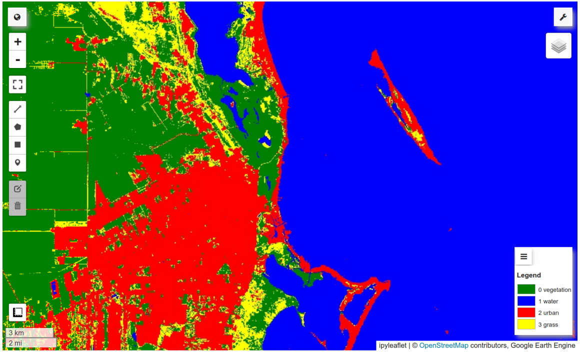


Image Source: Cardille et al. 2024

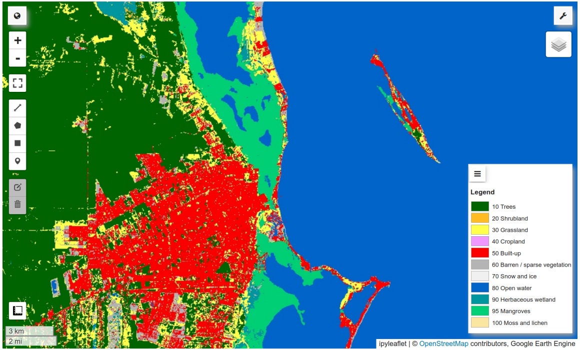
Compare with land cover image collections available in GEE

⇒ compare your classification with the ESA World Cover collection, which is a global map of land use and land cover derived from ESA's Sentinel-2 imagery at 10m.

Land cover classification (CART)



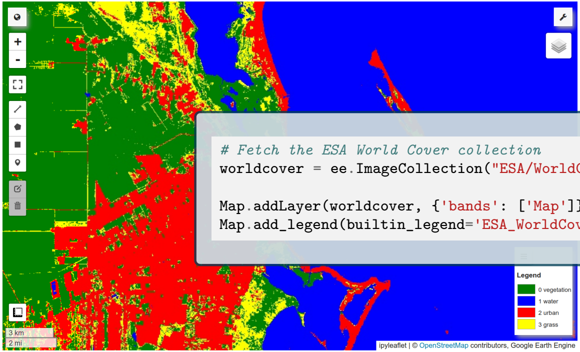
Land cover classification (ESA World Cover)



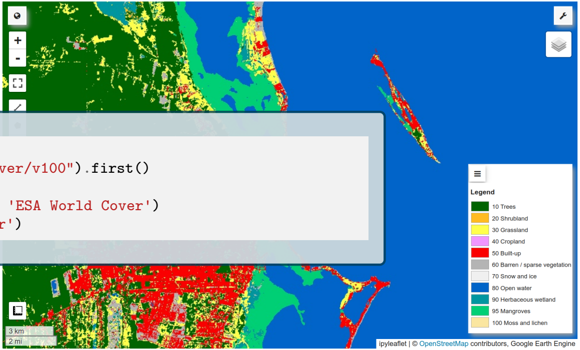
Compare with land cover image collections available in GEE

⇒ compare your classification with the ESA World Cover collection, which is a global map of land use and land cover derived from ESA's Sentinel-2 imagery at 10m.

Land cover classification (CART)



Land cover classification (ESA World Cover)

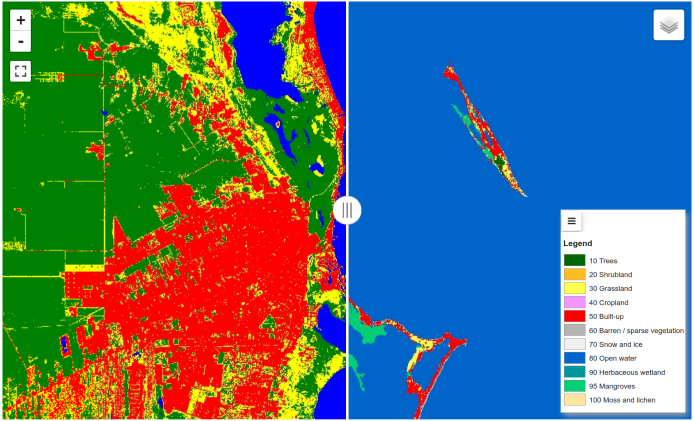



```python
# Fetch the ESA World Cover collection
worldcover = ee.ImageCollection("ESA/WorldCover/v100").first()

Map.addLayer(worldcover, {'bands': ['Map']}, 'ESA World Cover')
Map.add_legend(builtin_legend='ESA_WorldCover')
```

Compare with land cover image collections available in GEE

⇒ try using geemap's `split_map` method to easily compare the two classifications using a slider

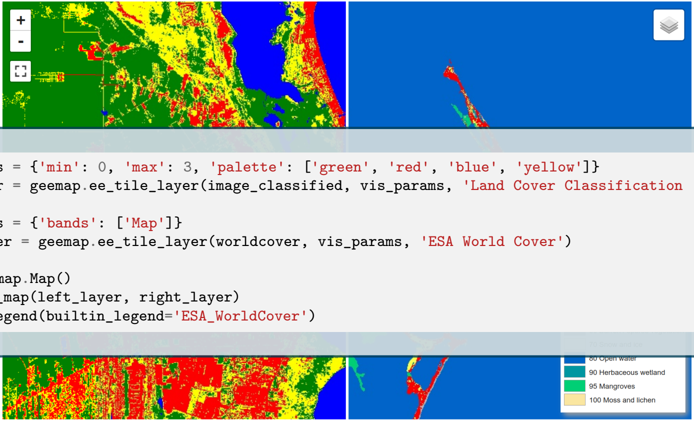
Slider to compare classification images



Compare with land cover image collections available in GEE

⇒ try using geemap's split_map method to easily compare the two classifications using a slider

Slider to compare classification images




```
vis_params = {'min': 0, 'max': 3, 'palette': ['green', 'red', 'blue', 'yellow']}
left_layer = geemap.ee_tile_layer(image_classified, vis_params, 'Land Cover Classification (CART)')

vis_params = {'bands': ['Map']}
right_layer = geemap.ee_tile_layer(worldcover, vis_params, 'ESA World Cover')

Map = geemap.Map()
Map.split_map(left_layer, right_layer)
Map.add_legend(builtin_legend='ESA_WorldCover')
```

Workflow of *unsupervised* classification

⇒ contrary to *supervised* algorithms (where the model learns from labeled training data provided by the user), *unsupervised* algorithms are "self-taught" as they do not rely on labeled data: instead, they attempt to find groups (i.e., clusters, classes) within the unlabeled data.

⇒ the workflow in GEE is as follows:

1. **Select image to classify**
2. **Collect randomly sampled points (unlabeled data)**
   ⇒ randomly sample pixels from the image, which will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties
3. **Select clustering algorithm and "train" it on the unlabeled data**
   ⇒ the *k-means* clustering algorithm is commonly used in remote sensing.
4. **Classify the image using the *trained clusterer***

Workflow of *unsupervised* classification

⇒ contrary to *supervised* algorithms (where the model learns from labeled training data provided by the user), *unsupervised* algorithms are "self-taught" as they do not rely on labeled data: instead, they attempt to find groups (i.e., clusters, classes) within the unlabeled data.

⇒ the workflow in GEE is as follows:

1. **Select image to classify**
2. **Collect randomly sampled points (unlabeled data)**
   ⇒ randomly sample pixels from the image, which will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties
3. **Select clustering algorithm and "train" it on the unlabeled data**
   ⇒ the *k-means* clustering algorithm is commonly used in remote sensing.
4. **Classify the image using the *trained clusterer***

Workflow of *unsupervised* classification

⇒ contrary to *supervised* algorithms (where the model learns from labeled training data provided by the user), *unsupervised* algorithms are "self-taught" as they do not rely on labeled data: instead, they attempt to find groups (i.e., clusters, classes) within the unlabeled data.

⇒ the workflow in GEE is as follows:

1. **Select image to classify**
2. **Collect randomly sampled points (unlabeled data)**
   ⇒ randomly sample pixels from the image, which will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties
3. **Select clustering algorithm and "train" it on the unlabeled data**
   ⇒ the *k-means* clustering algorithm is commonly used in remote sensing.
4. **Classify the image using the *trained clusterer***

Workflow of *unsupervised* classification

⇒ contrary to *supervised* algorithms (where the model learns from labeled training data provided by the user), *unsupervised* algorithms are "self-taught" as they do not rely on labeled data: instead, they attempt to find groups (i.e., clusters, classes) within the unlabeled data.

⇒ the workflow in GEE is as follows:

1. **Select image to classify**
2. **Collect randomly sampled points (unlabeled data)**
   ⇒ randomly sample pixels from the image, which will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties
3. **Select clustering algorithm and "train" it on the unlabeled data**
   ⇒ the *k-means* clustering algorithm is commonly used in remote sensing.
4. **Classify the image using the *trained clusterer***

Workflow of *unsupervised* classification

⇒ contrary to *supervised* algorithms (where the model learns from labeled training data provided by the user), *unsupervised* algorithms are "self-taught" as they do not rely on labeled data: instead, they attempt to find groups (i.e., clusters, classes) within the unlabeled data.

⇒ the workflow in GEE is as follows:

1. **Select image to classify**
2. **Collect randomly sampled points (unlabeled data)**
   ⇒ randomly sample pixels from the image, which will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties
3. **Select clustering algorithm and "train" it on the unlabeled data**
   ⇒ the *k-means* clustering algorithm is commonly used in remote sensing.
4. **Classify the image using the *trained clusterer***

Step 1: select the image to classify

⇒ use the image you used during the supervised classification

Image to classify

Step 2: Collect randomly sampled points (unlabeled data)

⇒ randomly sample pixels from the image (using *all* bands) → these will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties

⇒ use the image.sample method to sample the image in random position (unlike the image.sampleRegion method used in the supervised classification, which sampled pixels at the training point locations) → this will create a *FeatureCollection* of random points

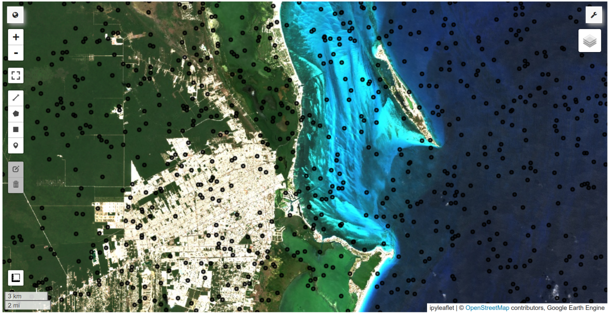
```
# Sample all image bands in random points

fc_training = image.sample(
                        region=image.geometry(),  # region to sample (take image footprint)
                        numPixels=1000,            # number of pixels to sample
                        geometries=True,           # add coordinates of sampled pixel in feature
                        # scale=10,                # nominal scale in meters to sample in
                    )

Map.addLayer(fc_training, {'color':'black'}, "Randomly sampled points")
```

Step 2: Collect randomly sampled points (unlabeled data)

⇒ randomly sample pixels from the image's prediction bands → these will constitute the *unlabeled dataset* in which to find clusters with similar spectral properties

⇒ use the image.sample method to sample the image in random position (unlike the image.sampleRegion method used in the supervised classification, which sampled pixels at the training point locations) → this will create a *FeatureCollection* of random points

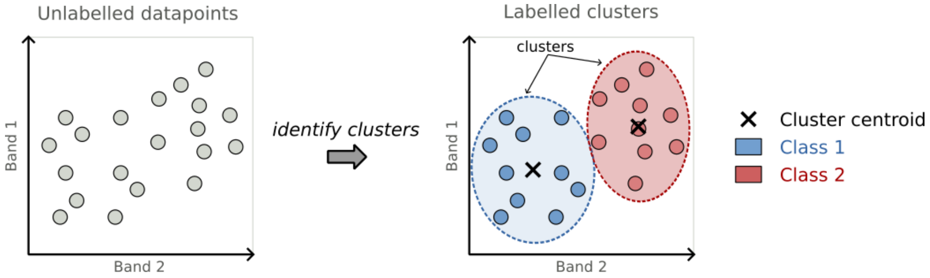
Randomly sampled points



Step 3: Select clustering algorithm and "train" it on the unlabeled data

⇒ in the example below we use the famous **k-means clustering algorithm** (ee.Clusterer.wekaKMeans)

⇒ the algorithm uses an *iterative grouping strategy* to identify *groups of pixels* (**clusters**) close to each other in the *spectral space*:

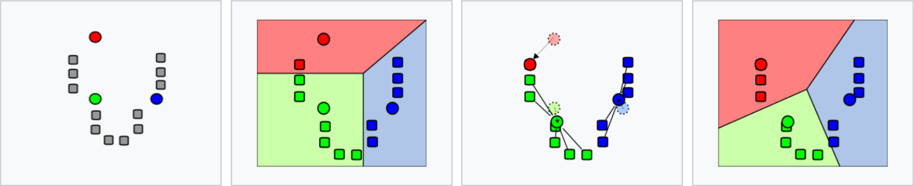


Step 3: Select clustering algorithm and "train" it on the unlabeled data

⇒ in the example below we use the famous **k-means clustering algorithm** (ee.Clusterer.wekaKMeans)

⇒ the algorithm uses an *iterative grouping strategy* to identify *groups of pixels* (**clusters**) close to each other in the *spectral space*:

⇒ the standard algorithm (a.k.a. *naive k-means*) iterative procedure can be illustrated as follows (source):



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Step 3: Select clustering algorithm and "train" it on the unlabeled data

⇒ instantiate the **k-means clustering algorithm** and train it on the randomly collected data (unlabeled)

```
# Instantiate the clustering algorithm
clusterer = ee.Clusterer.wekaKMeans(nClusters=4)

# Train the clustering algorithm
clusterer = clusterer.train(fc_training)
```

Step 4: Classify the image using the trained clusterer

⇒ apply the clusterer to the image and plot classified image

NB: because the clustering algorithm has no a-priori on the "meaning" of each class, in the example below we assign random colors to the classes (leaving it to the user to interpret the results).

```
# Classify the image using the trained clusterer
image_classified_kmeans = image.cluster(clusterer)

# Display the classification with random colors
Map.addLayer(image_classified_kmeans.randomVisualizer(), {}, 'Land cover classification (K-means)')
```

Unsupervised classification results

Natural color image

→

Land cover classification (K-means)