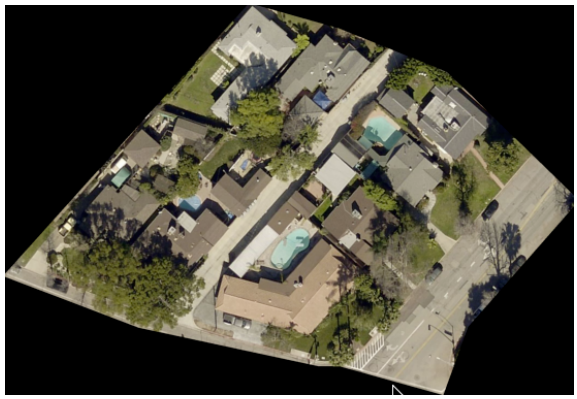
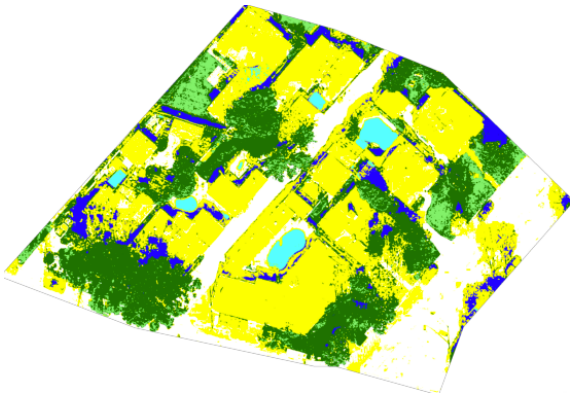
Lab 5: Integrating GIS and random forest for classification

In this lab, you will develop a random forest model to classify the land cover of a remote sensing image.



Input image



Classification Result

The data you will work with is an Eagle View aerial image "la_eagleview_small.tif". You have a training data "training_data_ev.tif", which is the same image with some areas labeled by a human annotator. Your goal is to use the training data from "training_data_ev.tif" to train a random forest model, and then apply the trained random forest model to the entire image to perform the classification.

Task 1 (20 pts): Use the rasterio package to load and visualize "la_eagleview_small.tif" and "training_data_ev.tif" respectively. You will need to visualize both images (5 pts), and show their height (5 pts), width (5 pts), and band count (5 pts). Use the variable name "land_image" to store the data from "la_eagleview_small.tif". Use the variable name "land_label" to store the data from "training_data_ev.tif".

The values in our training data "Training_data_ev.tif" are 0,1,2,3,4,5,6. These are labels indicating the land cover type of a pixel. More specifically, we have {'pool': 1, 'street': 2, 'grass': 3, 'roof': 4, 'tree': 5, 'shadow': 6, 'no data':0}. To get a better idea of how the training data look like, we may want to overlay these two images. To do so, you can use the following code:

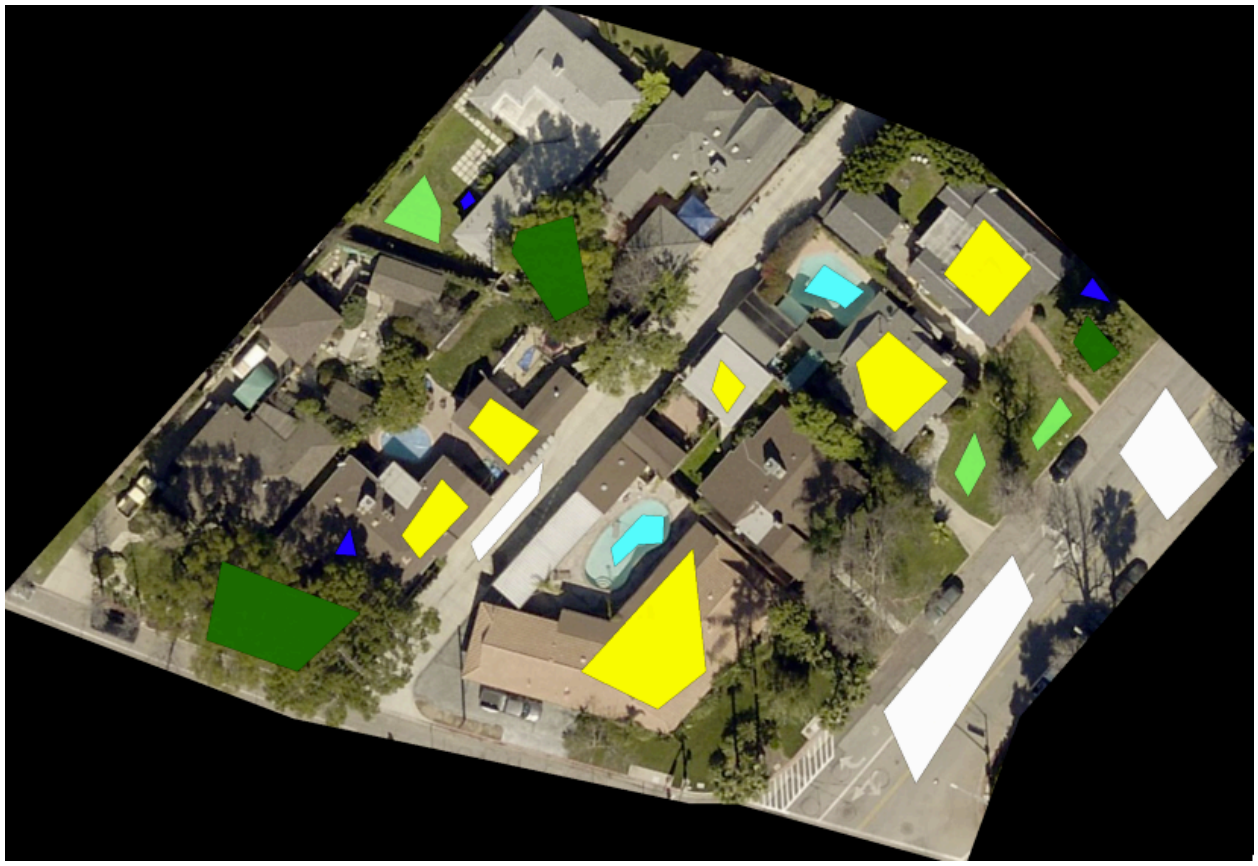
This lab is revised based on the data from GeoHackWeek: https://geohackweek.github.io/

```
plt.figure(figsize = (15,15))


# show the areial image
blue = land_image.read(3, masked=True)
green = land_image.read(2, masked=True)
red = land_image.read(1, masked=True)
rgb_land = np.dstack((red, green, blue))
plt.imshow(rgb_land)


# show the labeled training data above the image
# create a color palette to color the labels. This is because we may want to use some intuitive
# colors to show the label (e.g., blue for pool)
palette = np.array([[0, 0, 0, 0],  # no data
                    [0, 255, 255,250], # pool
                    [255, 255, 255,250],  # street
                    [100,238,100,250], #grass
                    [255, 255, 0,250], #roof
                    [0,100,0,250], #tree
                    [0, 0, 255,250] #shadow
                   ])
land_label_data = land_label.read(1)
plt.imshow(palette[land_label_data])
```

You should get a figure that looks like below:



As you can see, the training data are simply polygons drawn by a human annotator who indicated the land cover type of all the pixels within a polygon (e.g., roof). So what we can do is to use the pixels labeled out by the human annotator (i.e., label pixel value > 0) to train a model, and then apply the trained model to the entire image. The input features of this model (a random forest model) is the R, G, B values of each pixel (so you have three input features), and the output of the model is a classification value from 1 to 6.

To prepare our training data, we can use the following code:

```
# Get the labeled training data for each band
red_train = red[land_label_data>0]
blue_train = blue[land_label_data>0]
green_train = green[land_label_data>0]
X_label = np.column_stack((red_train, blue_train,green_train))   # put the three features as
                                                                 # three columns of the
                                                                 #matrix
```

```
# Get the labeled value
y_label = land_label_data[land_label_data>0]


# Split to training and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_label, y_label, test_size=0.2, random_state=42)
```

Task 2 (30 pts): Build a random forest model, set the *min_samples_leaf* parameter to 10. Train the model using the training data.

Task 3 (30 pts): Make predictions using your trained model on the test data, and evaluate the result using two metrics accuracy and confusion matrix. You can use the following code to import these two metric functions:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

Study how to use these two metric functions yourself, and calculate the accuracy score and confusion matrix of your trained model. If you are unfamiliar with confusion matrix, you can search it online. You can find one explanation here: http://www.50northspatial.org/classification-accuracy-assessment-confusion-matrix-method

Finally, we can apply the trained model to the entire image, and visualize the classification result. We can use the following code:

```
# prepare all the pixels of this image
X_whole = np.column_stack((red.ravel(), blue.ravel(), green.ravel()))
y_whole_pred = forest_clf.predict(X_whole)
# Reshape the prediction result into the shape of the image
y_whole_pred_reshape = y_whole_pred.reshape(land_image.height, land_image.width)
```

```
# show classification
plt.figure(figsize=(10,10))
plt.imshow(palette[y_whole_pred_reshape])
```

Task 4 (20 pts): A complete and working Jupyter notebook. This lab has provided multiple code snippets to help you prepare your data. While you can directly reuse those code snippets, you should make sure those code embed well with your own code and are bug free.

To submit:

- Lab5_ FirstName_LastName.ipynb