Comparing Pixel and Object-Based Classification of Heathland in Central Newfoundland using Random Forest Classifier

Prepared by: Alexander Johnston
For: Dr. Ben DeVries
GEOG*6050
Winter 2023

Abstract

Classified land cover maps of the Miawpukek First Nations Traditional Territory can aid conservation efforts, land management, and ecological studies. The primary objective of this project was to compare pixel-based and object-based random forest classification over a portion of my study area. Both pixel and object-based models performed well with relatively high accuracies and F1 scores. The object-based model had slightly lower scores than the pixel-based model across most metrics. However, it offered advantages such as noise reduction and enhanced ecological significance by aggregating pixels into meaningful objects, enabling the analysis of landscape patterns like patch size, shape, and connectivity.

Introduction

The primary objective of my thesis is to identify and map heathlands within the traditional territory of Miawpukek First Nation. In the previous semester, I conducted a random forest classification based on pixels, evaluating Sentinel 2, SAR, and terrain data composites for heathland classification. Although the Sentinel-2 cloudless composite classification yielded a commendable overall accuracy of 94.63%, there are many advantages to adopting an object-based classification approach. In traditional pixel-based classification, each pixel in an image is classified independently based on its spectral signature, this can lead to excessive noise and mislabeled pixels. Object-based classification, on the other hand, groups neighbouring pixels into meaningful objects or segments based on their spectral, spatial, and contextual properties allowing this information to be harnessed in classification (Liu & Xia, 2010; Mahdianpari et al., 2018)

Heathlands, culturally and ecologically significant ecosystems, are the focus of this study. In addition to heathlands, classes such as Forest, Water, Bog, and Exposed Rock were selected to enrich landscape understanding and classification accuracy. Furthermore, investigating heathland succession post-disturbance through time series analysis stands as a promising research direction. This approach would utilize the classification of various land classes to unveil both primary and secondary succession patterns within heathland ecosystems.

Heathlands serve as habitats for rare plants and cultural keystone species, emphasizing the need for conservation (Oberndorfer & Lundholm, 2009;

Schaefer et al., 2016). By safeguarding heathlands, we ensure the protection of critical habitats for numerous species, contributing to the overall health and resilience of the surrounding environment. Additionally, heathlands hold significant cultural importance to Miawpukek First Nation, serving as spaces for traditional cultural practices such as berry picking, foraging for medicinal plants, and hunting. These practices are deeply rooted in the cultural identity and heritage of the Miawpukek First Nation, reflecting their intimate connection to the land. Therefore, allocating resources to conserve these areas not only benefits the ecological integrity of the region but also safeguards the rich cultural heritage and traditional knowledge of Miawpukek First Nation, making heathlands ideal candidates for protection.

The Ministry of Environment and Climate Change committed $3 million in 2022 to support area-based conservation work in Miawpukek First Nation (The Government of Canada and Miawpukek First Nation in Newfoundland and Labrador Take First Steps toward a New Indigenous Protected and Conserved Area, 2022). Coastal edaphic heathlands surrounding the community serve as an ideal focal point for this conservation effort. These unique ecosystems provide not only crucial habitats for a variety of plant and animal species but also offer opportunities for sustainable resource management and traditional ecological knowledge exchange. Therefore, data generated from this project could play a crucial role in supporting indigenous-led conservation initiatives within Miawpukek First Nation's traditional territory.

Methods & Data

Study Area

The traditional territory of Miawpukek First Nations (MFN) spans nearly 23,000 km^2, extending across the central and maritime barren ecoregions of Newfoundland. Little is known about the extent of heathlands within MFN's traditional territory. Twenty-six heathland sites, each with three 20x20m plots were sampled last summer as part of a study on functional traits. GPS points were gathered for each plot and many additional points were collected as ground-truthed heathland points. Eighty-six points fall within the study area and will be used as secondary validation points.

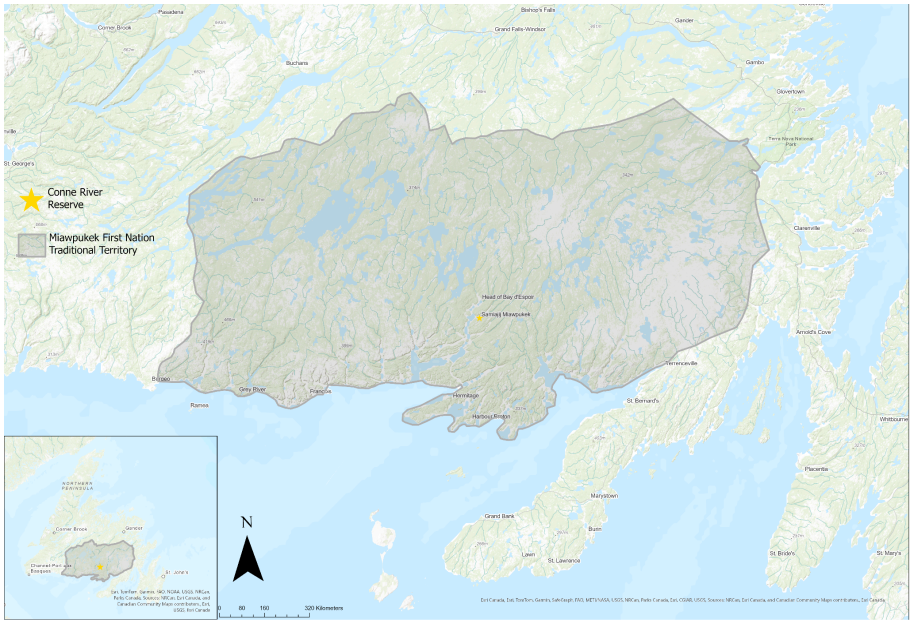

Figure 1: The traditional territory of Miawpukek First Nations (MFN)

The study area for this project is the Sentinel 2 tile that contains the majority of the field sites visited last summer. The tile is situated on the Eastern side of the traditional territory and along Highway 360, the sole route to and from the southern shore. This area includes representative classes, which are indicative of those found across Miawpukek First Nation, such as coastal edaphic heathlands, successional kalmia heaths, black spruce forests, balsam fir forests, bog, lakes, and ponds. The study area has two distinct classes of heathlands: edaphic and successional heathlands. Successional heathlands develop as a result of stand-replacing disturbances, such as after a fire or clear-cutting, and will gradually return to forest over time through secondary succession (Meades, 1983). Edaphic heathlands, on the other hand, form in regions with specific environmental conditions such as severely wind-exposed or nutrient-poor areas (Meades, 1983). For this project, both classes will be identified under the umbrella category heathland. Future models incorporating environmental variables will be used to delineate these two classes.

Image Segmentation

Shepherd image segmentation algorithm from the rsgislib python library was used to group pixels with similar spectral and spatial characteristics. The algorithm works by seeding using a k-means clustering algorithm, then iteratively clumping adjacent segments by comparing spectral properties of

neighbouring pixels. This initial clumping results in an over-segmented image. The next stage involves an iterative elimination process based on a minimum segment size for the elimination parameter. In this process, smaller regions which fall below a defined minimum mapping unit are progressively merged with their spectrally closest neighbours. There is an optional spectral distance threshold to handle features that are small and highly spectrally distinct from their neighbours such as ponds. Regions above the defined threshold are not merged with their neighbours, allowing the retention of small regions. The last step is relabeling. Regions are eliminated during the iterative process resulting in gaps in the attribute table. Relabeling is applied to ensure that the identification numbers for the remaining regions are sequential. Iterative testing was performed with various minimum segment sizes (5, 10, 20, 30, 40, 50, and default 100) and the distance threshold (5, 10, 20, 30, 40, 50, and default 100). The segmented images were compared qualitatively and in their model performance. Min segment size of 10 and a distance threshold of 20 yielded the best results.

Random Forest

A random forest classifier from the scikit-learn was used to classify the object and pixel-based images based on a set of training points. RF combines the predictions of multiple decision trees (Salwa Thasveen & Suresh, 2021). Each decision tree is trained on a subset of the data and a random subset of the features. During classification, the algorithm aggregates the predictions of these individual trees to make a final classification decision (Salwa Thasveen & Suresh, 2021). The number of trees was set to 250 and a random seed was chosen for consistent results. All other parameters remained the default.

Testing and Training Samples

Training points were stratified by class and given a class ID based on the previous semester's pixel-based classification. Three hundred random points were generated for each class. Each point was evaluated in Arc Pro using the 2022 cloud-free composite and a Google Earth base map. Points were either relabeled to their correct class or eliminated if their class was unidentifiable. Due to the small scattered nature of the exposed rock category many of the points had to be removed as they were being incorporated into their surrounding object in the object segmented image. The sample points were rasterized and clipped to the input images. The sample points were randomly

split 70/30 for training and testing using a random seed for consistent comparisons among models.

Number of Heathland Training/Test Points:	289
Number of Forest Training/Test Points:	300
Number of Water Training/Test Points:	300
Number of Wetland Training/Test Points:	298
Number of Rock Training/Test Points:	187

Table 1: Number of sampled points used to train and test the models.

Secondary validation was performed on the classified images using the ground-truthed heathland points by extracting classification values for each point.

Accuracy Assessment

Accuracy assessments measure how accurately the classification prediction aligns with human interpretation of land cover classes (the testing data). Confusion matrices were generated for the pixel-based and object-based classification from which the producer, user, F1 score, kappa coefficient, and overall accuracy were derived. The overall accuracy is the proportion of the data classified correctly by the model (Cardille et al., 2023). It is calculated as the sum of correctly identified pixels divided by the total number of pixels in the sample (Cardille et al., 2023).

$$Overall\ Accuracy\ = \frac{Number\ of\ correctly\ classified\ pixel}{Sample\ Size}$$

The overall accuracy does not take into account class imbalances

User Accuracy (UA) is the probability that the algorithm classifies a pixel as its correct class (Cardille et al., 2023).

$$User\ Accuracy\ (UA)\ = \frac{Number\ of\ correctly\ classified\ pixels\ for\ a\ class}{Total\ number\ of\ pixels\ classified\ as\ that\ class}$$

Producer Accuracy (PA) is the probability that a pixel in a particular class is correctly classified as that class (Cardille et al., 2023).

$$Producer\ Accuracy\ (PA) = \frac{Total\ number\ of\ pixels\ that\ belong\ to\ that\ class}{Number\ of\ correctly\ classified\ pixels\ for\ a\ class}$$

The F1 score is the harmonic mean of the user and producer accuracy, making it useful for scenarios where both types of accuracy are important. Its effectiveness over overall accuracy becomes apparent in scenarios with uneven class distribution, as it accounts for both false positives and false negatives, providing a more nuanced measure of model performance.

$$F1 = \frac{2 \times User\ Accuracy \times Producer\ Accuracy}{User\ Accuracy + Producer\ Accuracy}$$

Lastly, is the kappa coefficient, which evaluates how well the classification performed compared to a random assignment of classes (Cardille et al., 2023). The value of the kappa coefficient can range from −1 to 1 (Cardille et al., 2023). Negative values indicate that the classification is worse than a random assignment of categories (Cardille et al., 2023). Positive values indicate that the classification is better than random (Cardille et al., 2023)).

$$Kappa\ Coefficient = \frac{overall\ \ accuracy - chance\ agreement}{1 - chance\ agreement}$$

The chance agreement is the sum of the product of row and column totals for each class (Cardille et al., 2023).

Results and Discussion

Class separability and Predictor Importance

The bands with the highest relative importance are the green band and NDVI band, which is unsurprising given that the majority of the land cover in the image is vegetated. NDVI demonstrated clear distinctions between vegetated and non-vegetated classes such as water and exposed rock. The figures below highlight the significance of the green and NDVI bands in accurately capturing and distinguishing different land cover types within the study area.

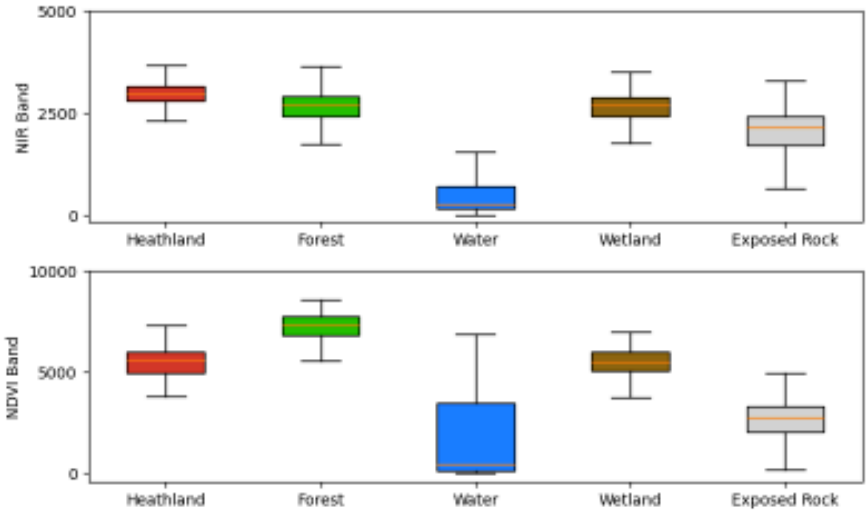

Figure 2: Boxplot of NDVI and NIR values of training samples by class.

Bands	**Relative Importance Score**
Band 1: Red Band	0.17450097
Band 2: Green Band	0.2421223
Band 3: Blue Band	0.11968916
Band 8: NIR	0.15452926
Band 12: NDVI	0.30915831

Table 2: Relative importance scores of each band in the model. The object and pixel-based models had the same relative importance scores despite different accuracies.

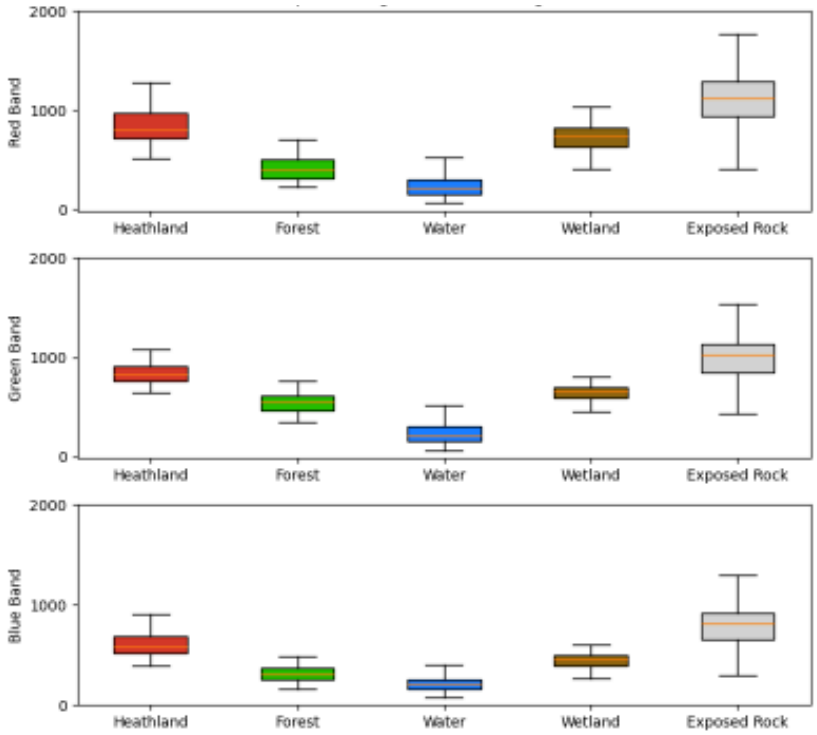

Figure 3: Boxplot of spectral values of classes in the visible bands

Accuracy of Object and Pixel-based Models

Both models exhibited high accuracies and F1 scores, indicating robust performance of the RF model and the training sample. The object-based model generally displayed slightly lower accuracies compared to the pixel-based model across most metrics. The bare ground and water classes had the highest F1 score. These classes are distinguishable from other land cover based on their unique spectral signatures. Bare ground has low NDVI and high reflectance in the visible bands, while water bodies have very low reflectance across all bands (See Figures 2 and 3). The class with the lowest accuracies was the wetland class. Wetlands exhibit spatial heterogeneity, within a wetland, there can be diverse vegetation types, water bodies, and soil conditions. This complexity often leads to class confusion, particularly with heathland which shares similar vegetation and structure, resulting in reduced accuracies for both classes.

Object-Based Accuracies:			
Class	Producer Accuracy	User Accuracy	F1 Score
Heathland	0.908046	0.940476	0.923977
Forest	0.923913	0.913978	0.918919
Water	0.956989	0.978022	0.967391
Wetland	0.895349	0.865169	0.880000
Bare Ground	0.981818	0.964286	0.972973
Pixel-Based Accuracies:			
Heathland	0.931034	0.952941	0.941860
Forest	0.989130	0.968085	0.978495
Water	0.978495	0.989130	0.983784
Wetland	0.918605	0.940476	0.929412
Bare Ground	1.000000	0.948276	0.973451

Table 3: The producer accuracies, user accuracies and F1 score for both the object-based and pixel-based models across classes.

Overall Accuracies:		
Metric	Object-Based	Pixel-Based
Overall Accuracy	0.929782	0.961259
Kappa Coefficient	0.911614	0.951253
Overall F1 Score	0.932652	0.961400

Table 4: The overall accuracies for both the object-based and pixel-based models.

Secondary Validation

Heathland classification accuracy was analyzed using the ground-truthed points gathered last field season. It was observed that the object-based

classification predominantly misclassified these points as forests. The object-based classification performed better inland where objects are larger and more homogeneous and worse in coastal areas where objects are smaller and more heterogeneous.

Class	Pixel-based Count	Object-based Count
Heathland	72	70
Forest	7	14
Rock	2	0
Wetland	5	2
Accuracy (%)	83.72	81.40

Table 5: Count of the prediction of ground-truthed heathland points and the total percent accuracy

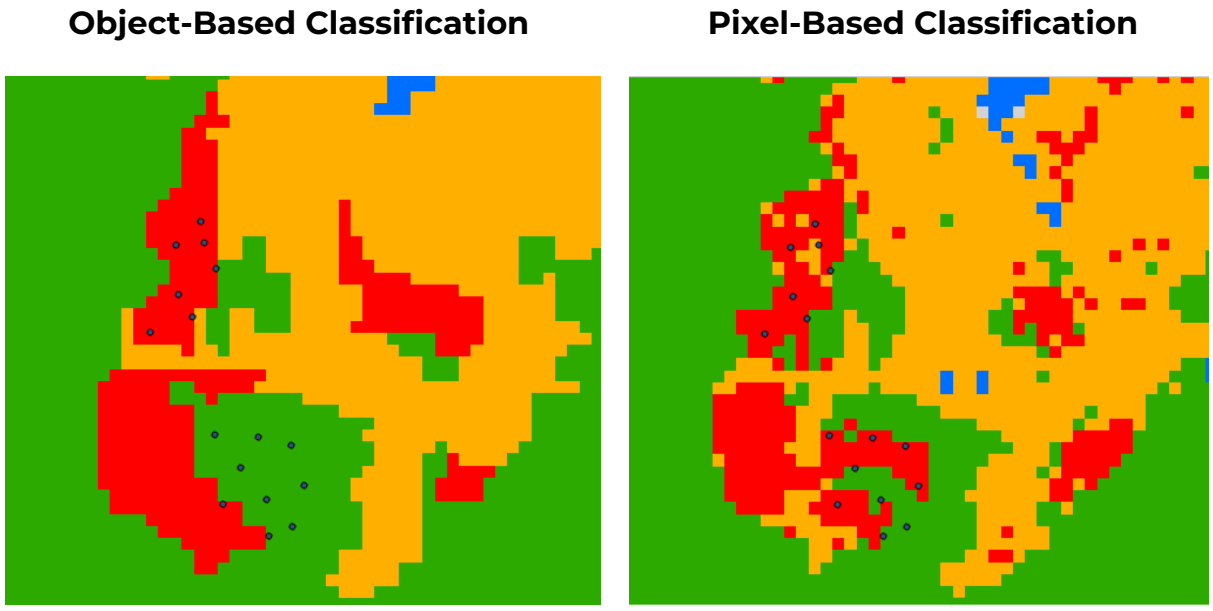

Figure 4: An example of where object-based classification under segmented, resulting in heathland ground truth points being labeled as forests

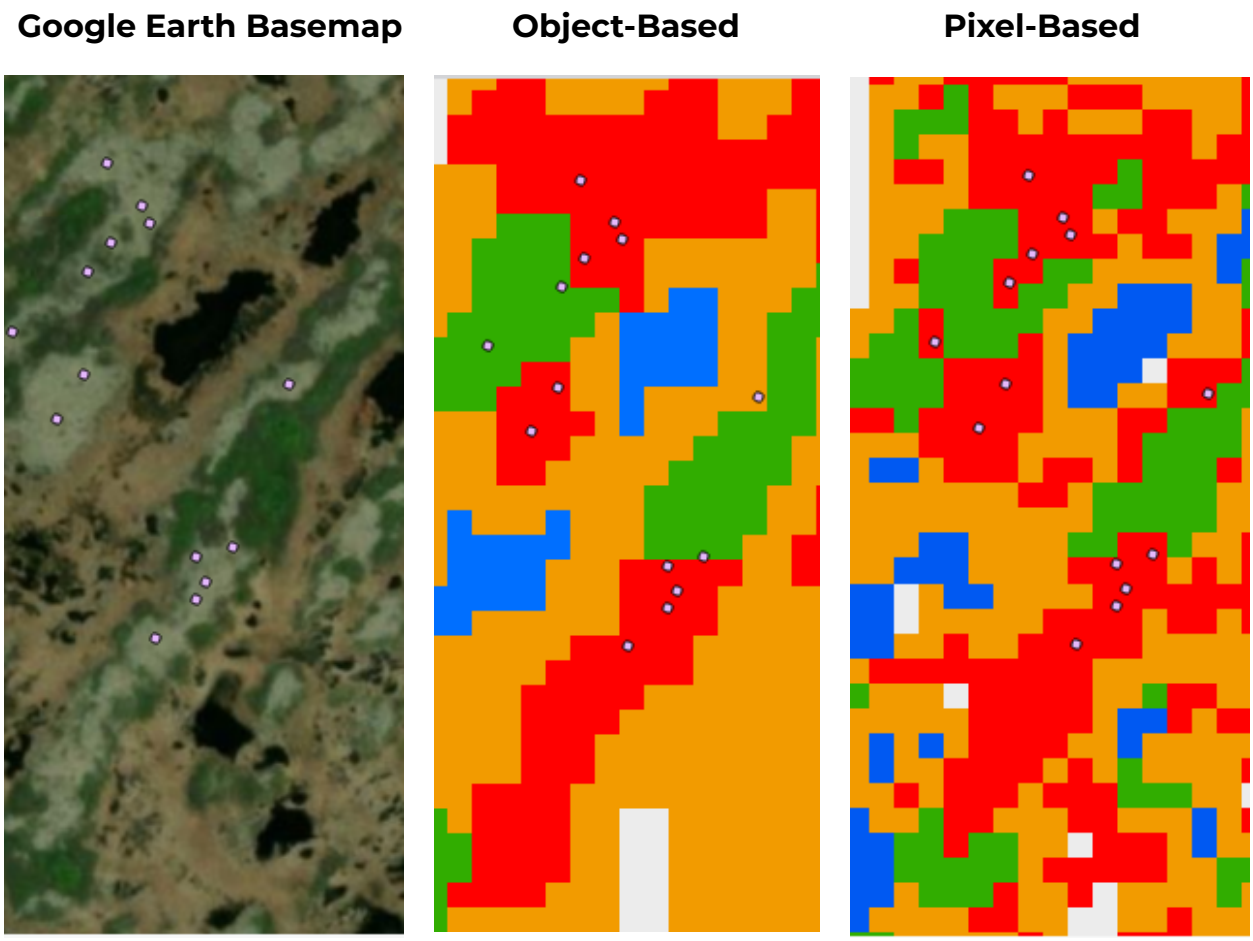

Figure 5: Another area where object-based methods incorrectly labelled heathland ground truth points as forest

While the pixel-based model misclassified heathland as wetlands most frequently. Object-based methods exhibited varying impacts on classification performance, sometimes hindering classification accuracy from under-segmentation (See Figure 4 and 5) and other times aiding in classification by reducing noise (See Figure 6).

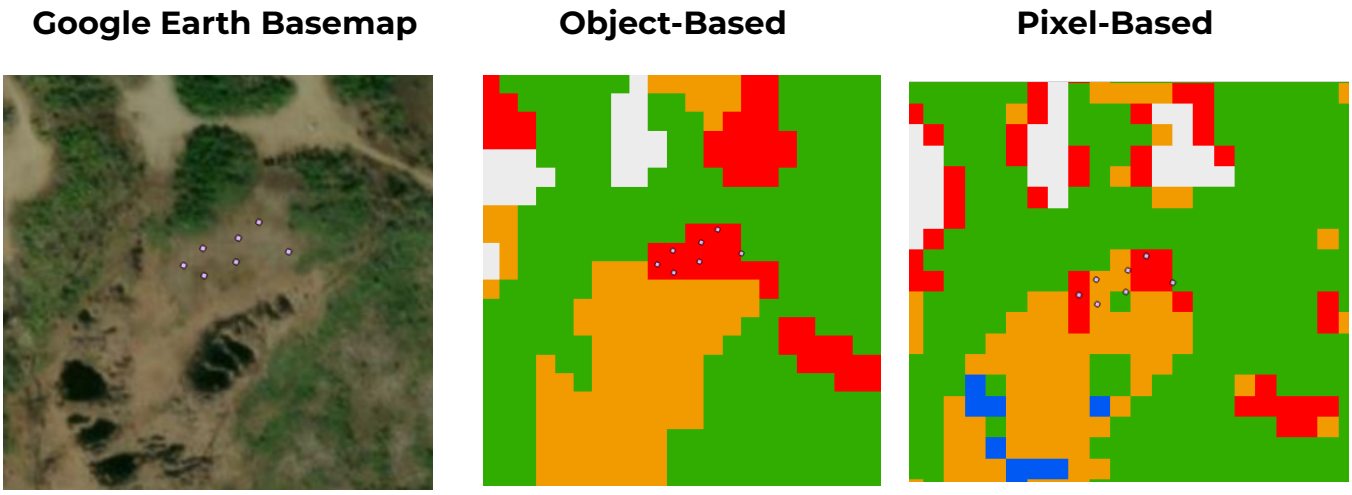

Figure 6: Segmentation of this heathland, resulted in a more realistic representation of the land cover, thus leading to a more precise classification of ground truth points.

Discussion

Using the same input image as last semester's model but with a larger sample of training points, resulted in improved classification accuracies. The object-based model had slightly lower accuracies than the pixel-based model across most metrics. The object-based model was also more computationally intensive on account of image segmentation, which required additional processing steps.

The bare land class (originally labelled as exposed rock) training data was found to be problematic for the object based classification. The exposed rock appears in both heathland and wetland classes where glaciers have stripped away the soil and sediment to expose the underlying bedrock. These exposed rock features were grouped into the surrounding landclasses during segmentation. Training features containing small scattered exposed rocks were removed for both models. After removing scattered rocks in the landscapes the rock class comprised mainly anthropogenic land cover such as mining, quarrying, and roads so it was relabeled to bare land to better suit the land cover in the class.

The object-based classification may have exhibited lower accuracies for a few reasons. Objects in the segmented image were being under-segmented meaning they contained a mix of different classes of pixels. The object-based classification was also sensitive to the segmentation algorithm parameters. The Shepherd algorithm requires setting the minimum segment size, which determines the scale at which objects are delineated. A size of 10 was found to be the minimum size that could be set without detrimenting classification accuracy. However, there was still under-segmentation in extremely heterogeneous, scattered landscapes such as those found in coastal areas. Object-based classification also struggled to classify objects with transitional ecotones, where a blend of species and environmental factors from adjacent communities prevails rather than distinct separations, resulting in misclassifications near object edges. Fuzzy logic may improve the classification of these ecotones. Instead of assigning a single class label to each pixel or object, fuzzy logic allows for the assignment of multiple class memberships based on the degree of resemblance to each class (Arnot & Fisher, 2007). Integrating Synthetic Aperature Radar (SAR), and topography data into the object-based classification may improve the classification of forest pixels (Mahdianpari et al., 2018). However, topography data in Newfoundland is limited. Specifically, there is a lack of LiDAR-derived elevation data for the study area, with only a 5m Digital Elevation Model (DEM) with unknown origins available. Integrating wetness indices such as Topographic Wetness Index (TWI) into the model may help with the delineation of wetlands from heathlands (Hubert-Moy et al., 2022).

Further refinement of segmentation parameters and an increased number of sample points may improve the performance of the object-based classification. Despite its limitations, object-based classification offers advantages including reduced noise impact and better alignment with ecological interpretations. By aggregating pixels into coherent objects, it facilitates a more intuitive understanding of landscape structure and patterns, providing valuable insights into patch size, shape, and connectivity essential for comprehending ecological processes.

References

https://doi.org/10.1093/jmammal/gyv184

Arnot, C., & Fisher, P. (2007). Mapping the Ecotone with Fuzzy Sets. In A. Morris & S. Kokhan (Eds.), *Geographic Uncertainty in Environmental Security* (pp. 19–32). Springer Netherlands. https://doi.org/10.1007/978-1-4020-6438-8_2

Cardille, J. A., Crowley, M. A., Saah, D., & Clinton, N. E. (Eds.). (2023). *Cloud-based remote sensing with Google Earth Engine: Fundamentals and applications*. Springer.

Hubert-Moy, L., Rozo, C., Perrin, G., Bioret, F., & Rapinel, S. (2022). Large-scale and fine-grained mapping of heathland habitats using open-source remote sensing data. *Remote Sensing in Ecology and Conservation*, *8*(4), 448–463. https://doi.org/10.1002/rse2.253

Liu, D., & Xia, F. (2010). Assessing object-based classification: Advantages and limitations. *Remote Sensing Letters*, *1*(4), 187–194. https://doi.org/10.1080/01431161003743173

Mahdianpari, M., Salehi, B., Mohammadimanesh, F., Homayouni, S., & Gill, E. (2018). The First Wetland Inventory Map of Newfoundland at a Spatial Resolution of 10 m Using Sentinel-1 and Sentinel-2 Data on the Google Earth Engine Cloud Computing Platform. *Remote Sensing*, *11*(1), 43. https://doi.org/10.3390/rs11010043

Meades, W. J. (1983). Heathlands. In *Biogeography and Ecology of the Island*

of Newfoundland (Vol. 48, pp. 267–318). Dr. W. Junk Publishers, The Hague.

Oberndorfer, E. C., & Lundholm, J. T. (2009). Species richness, abundance, rarity and environmental gradients in coastal barren vegetation. *Biodiversity and Conservation*, *18*(6), 1523–1553. https://doi.org/10.1007/s10531-008-9539-5

Salwa Thasveen, M., & Suresh, S. (2021). Land - Use and Land - Cover Classification Methods: A Review. *2021 Fourth International Conference on Microelectronics, Signals & Systems (ICMSS)*, 1–6. https://doi.org/10.1109/ICMSS53060.2021.9673623

Schaefer, J. A., Mahoney, S. P., Weir, J. N., Luther, J. G., & Soulliere, C. E. (2016). Decades of habitat use reveal food limitation of Newfoundland caribou. *Journal of Mammalogy*, *97*(2), 386–393. https://doi.org/10.1093/jmammal/gyv184

The Government of Canada and Miawpukek First Nation in Newfoundland and Labrador take first steps toward a new Indigenous Protected and Conserved Area. (2022). Environment and Climate Change Canada. https://www.canada.ca/en/environment-climate-change/news/2022/09/the-government-of-canada-and-miawpukek-first-nation-in-newfoundland-and-labrador-take-first-steps-toward-a-new-indigenous-protected-and-conserved-area.html

Appendix

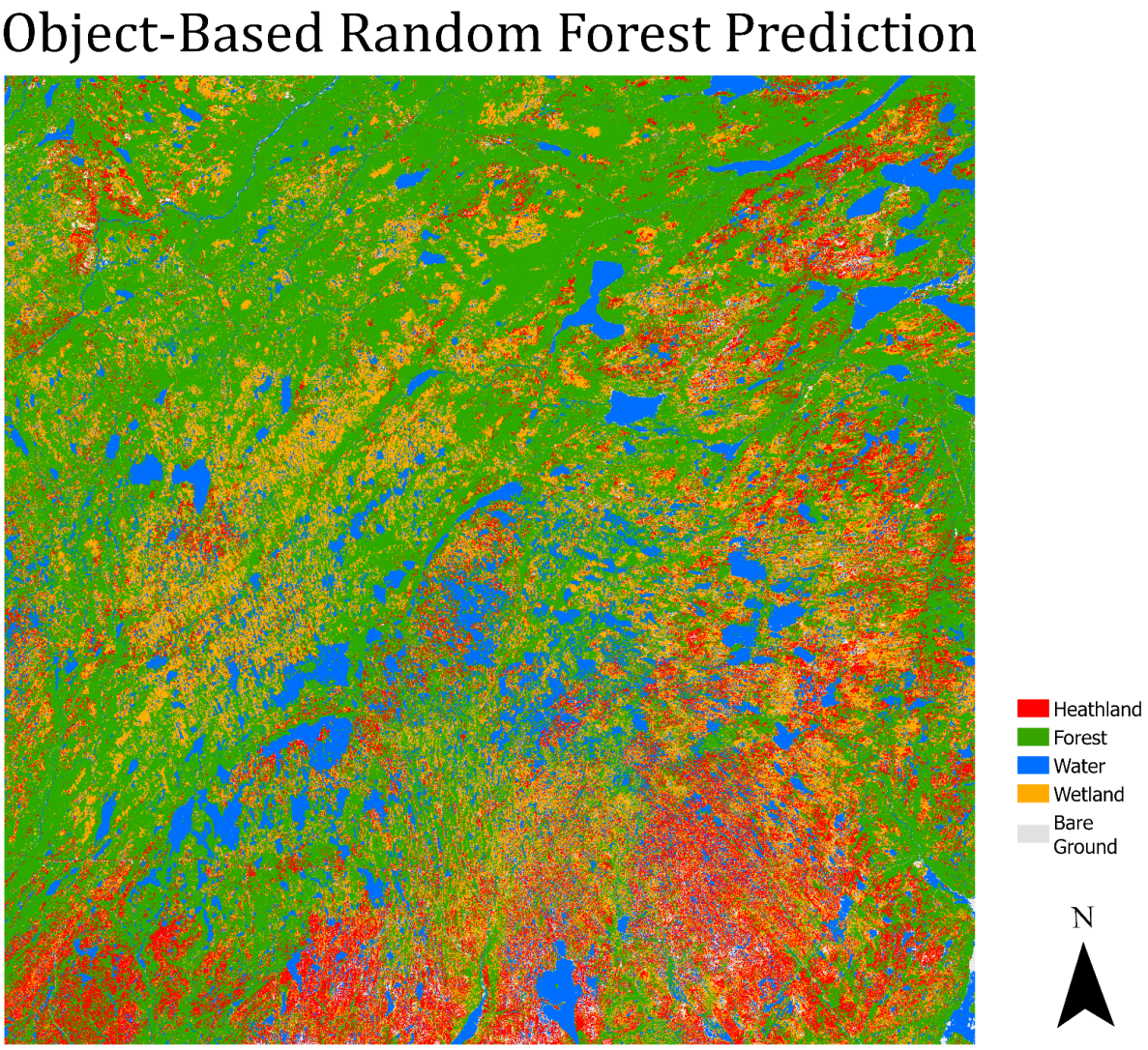

Figure 7: Object-based random forest prediction of the full sentinel 2 tile

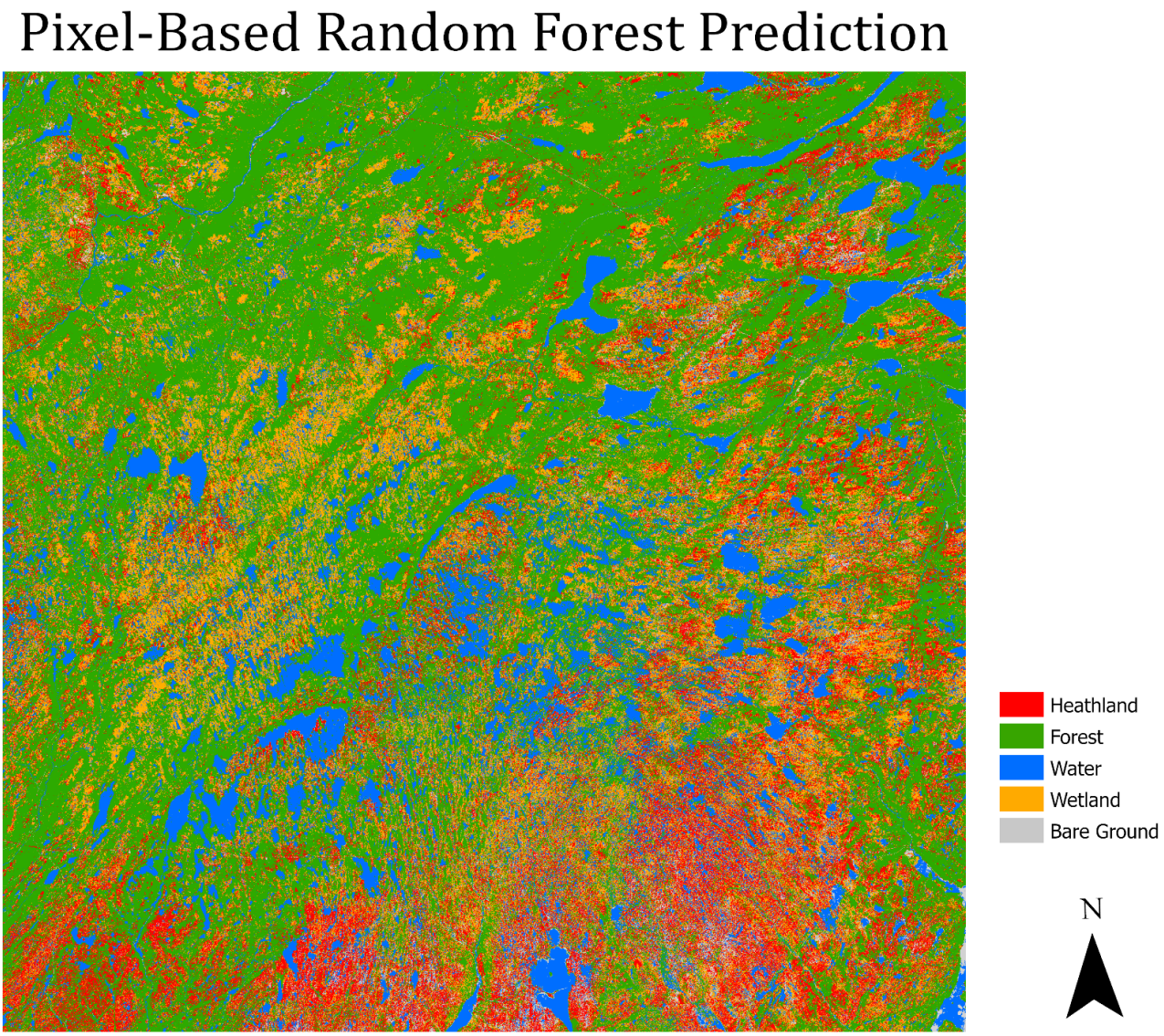

Figure 8: Pixel-based random forest prediction of the full sentinel 2 tile

```
import rsgislib
from rsgislib.segmentation import shepherdseg
import rasterio
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.colors import ListedColormap
from osgeo import gdal, gdalconst, ogr, osr
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, cohen_kappa_score
from sklearn import preprocessing
from skimage import io, color, morphology
from skimage.filters import sobel
import joblib
%matplotlib inline


input_img = # Input Sentinel-2 image file
out_mean_img = #Out mean values of clumps
output_clumps_img= # Output clump file
mean_objectsTIFF =  #Out mean as GeoTIFF

#Utility function to call the segmentation algorithm
shepherdseg.run_shepherd_segmentation(input, output_clumps_img,
out_mean_img, min_n_pxls=10, dist_thres=20, km_max_iter=150)


gdal.Warp(mean_objectsTIFF, mean_objects, format='GTiff', dstSRS=
'EPSG:32621', outputType=gdalconst.GDT_UInt16, dstNodata=65535)

pixel_image_path = input_img
object_image = mean_objectsTIFF
training_data = #Training data tif
secondary_validation = #secondary validation tif

with rasterio.open(training_data) as src:
```

```
    training_metadata = src.profile
    original_crs = src.crs
    original_width = src.width
    original_height = src.height

print(training_metadata)


# In[5]:


width = training_metadata['width']
height = training_metadata['height']
print("# of rows:", height)
print("# of columns:", width)


# In[6]:


with rasterio.open(training_data) as src:
    training = src.read(1)
    classes=np.unique(training)[1:]
    print(classes)

with rasterio.open(secondary_validation) as src:
    secondValidation = src.read(1)
    validationClasses=np.unique(secondValidation)[1:]
    print(validationClasses)

with rasterio.open(pixel_image_path) as src:
        pixel_red= src.read(3)
        pixel_green= src.read(2)
        pixel_blue= src.read(1)
        pixel_NIR= src.read(4)
        pixel_NDVI= src.read(5)

with rasterio.open(object_image) as src:
```

```
    objectmetadata = src.profile
    print(objectmetadata)
    object_red= src.read(3)
    object_green= src.read(2)
    object_blue= src.read(1)
    object_NIR= src.read(4)
    object_NDVI= src.read(5)

print(training.shape, object_red.shape, pixel_red.shape,
secondValidation.shape)


# In[7]:


# Coordinates of the 100m x 100m square
xmin = 200
xmax = 600
ymin = 200
ymax = 600

fig, ax = plt.subplots(1, 3, figsize=[20, 10], sharex=True, sharey=True)

# Zoom into the specified region for each subplot
ax[0].imshow(ma.masked_equal(training[ymin:ymax, xmin:xmax], -32768.0),
interpolation="none")
ax[0].set_title("Training Pixels")
ax[1].imshow(pixel_NDVI[ymin:ymax, xmin:xmax], cmap='RdYlGn', vmin=-5000,
vmax=8000, extent=[xmin, xmax, ymin, ymax])
ax[1].set_title("Pixel Image NDVI")
ax[2].imshow(object_NDVI[ymin:ymax, xmin:xmax], cmap='RdYlGn',
vmin=-5000, vmax=8000, extent=[xmin, xmax, ymin, ymax])
ax[2].set_title("Object Based NDVI")

plt.show()


# In[8]:
```

```
training1D = training.flatten()
print(training1D.shape)

validation1D = secondValidation.flatten()
print(validation1D.shape)

pixel_r_1d= pixel_red.flatten()
pixel_g_1d= pixel_green.flatten()
pixel_b_1d= pixel_blue.flatten()
pixel_NIR_1d= pixel_NIR.flatten()
pixel_NDVI_1d= pixel_NDVI.flatten()

object_r_1d= object_red.flatten()
object_g_1d= object_green.flatten()
object_b_1d= object_blue.flatten()
object_NIR_1d= object_NIR.flatten()
object_NDVI_1d= object_NDVI.flatten()


objectpredictors2D = np.stack([object_r_1d, object_g_1d, object_b_1d,
object_NIR_1d, object_NDVI_1d])
pixelpredictors2D= np.stack([pixel_r_1d, pixel_g_1d, pixel_b_1d, pixel_NIR_1d,
pixel_NDVI_1d])
print("objectpredictors2D shape:", objectpredictors2D.shape)
print("pixelpredictors2D shape:", pixelpredictors2D.shape)


# In[9]:


heathland_index=np.where(training1D == 0)[0]
forest_index=np.where(training1D == 1)[0]
water_index=np.where(training1D == 2)[0]
wetland_index=np.where(training1D == 3)[0]
rock_index=np.where(training1D == 4)[0]
print("Number of Heathland Pixels:", heathland_index.shape[0])
```

```
print("Number of Forest Pixels:", forest_index.shape[0])
print("Number of Water Pixels:", water_index.shape[0])
print("Number of Wetland Pixels:", wetland_index.shape[0])
print("Number of Rock Pixels:", rock_index.shape[0])


# In[10]:


Y_train = np.hstack([training1D[heathland_index], training1D[forest_index],
training1D[water_index], training1D[wetland_index], training1D[rock_index]])
print(Y_train.shape)
print(Y_train)


# In[11]:


pixel_x_train = np.hstack([pixelpredictors2D[:,heathland_index],
pixelpredictors2D[:,forest_index], pixelpredictors2D[:,water_index],
pixelpredictors2D[:,wetland_index], pixelpredictors2D[:,rock_index]]).T
print(pixel_x_train.shape)
print(pixel_x_train)


# In[12]:


object_x_train = np.hstack([objectpredictors2D[:,heathland_index],
objectpredictors2D[:,forest_index], objectpredictors2D[:,water_index],
objectpredictors2D[:,wetland_index], objectpredictors2D[:,rock_index]]).T
print(object_x_train.shape)
print(object_x_train)


# In[13]:
```

```
# Define the spectral bands
bands = ['Red', 'Green', 'Blue', 'NIR', 'NDVI']

# Flatten the spectral data for each band
spectral_data = {
    'Red': object_r_1d,
    'Green': object_g_1d,
    'Blue': object_b_1d,
    'NIR': object_NIR_1d,
    'NDVI': object_NDVI_1d
}

# Define the indices for each class
class_indices = {
    'Heathland': heathland_index,
    'Forest': forest_index,
    'Water': water_index,
    'Wetland': wetland_index,
    'Rock': rock_index
}
class_labels = ["Heathland", "Forest", "Water", "Wetland", "Exposed Rock"]

class_colors = {
    0: '#d6382c',
    1: '#23bf01',
    2: '#1d7dff',
    3: '#8e6410',
    4: '#d6d6d6'
}

# Plot spectral signature for each band
fig, axs = plt.subplots(len(bands), 1, figsize=(8, 12))

# Plot spectral signature charts
for i, band in enumerate(bands):
    for j, (class_name, index) in enumerate(class_indices.items()):
        band_spectral_data = spectral_data[band][index]
```

```
        axs[i].boxplot(band_spectral_data, positions=[j + 1], showfliers=False,
patch_artist=True, boxprops=dict(facecolor=class_colors[j]), widths=0.5)
    axs[i].set_ylabel(band + ' Band')
    axs[i].set_xticks(range(1, len(class_labels) + 1))
    axs[i].set_xticklabels(class_labels)
    if band in ['Red', 'Green', 'Blue']:
        axs[i].set_yticks([0, 1000, 2000])
    elif band == 'NIR':
        axs[i].set_yticks([0, 2500, 5000])
    elif band == 'NDVI':
        axs[i].set_yticks([0, 5000, 10000])

# Add titles
axs[0].set_title('Spectral Signature of Training Points')

plt.tight_layout()
plt.show()


# In[14]:


# Set a specific random state value
random_state_value = 88


# In[15]:


# Split dataset into training set and test set
x_pixel_train, x_pixel_test, pixel_y_train, pixel_y_test =
train_test_split(pixel_x_train, Y_train, test_size=0.3,
random_state=random_state_value)


# In[16]:
```

```
print(pixel_x_train.shape)


# In[17]:


# Split dataset into training set and test s
x_object_train, x_object_test, object_y_train, object_y_test  =
train_test_split(object_x_train, Y_train, test_size=0.3,
random_state=random_state_value)


# In[18]:


print(object_x_train.shape)


# In[19]:


RF= RandomForestClassifier(n_estimators=250,
random_state=random_state_value)
pixelRF = joblib.load('Pixel_random_forest.pkl')
# Accessing unique classes and number of classes
classes = pixelRF.classes_
n_classes = pixelRF.n_classes_

print("Unique Classes:", classes)
print("Number of Classes:", n_classes)


# In[20]:


pixelRF.feature_importances_
```

In[21]:

```
pixel_pred = pixelRF.predict(x_pixel_test)
pixel_accuracy = accuracy_score(pixel_y_test, pixel_pred)
print("Pixel-based classification accuracy:", pixel_accuracy)
```

In[22]:

```
#objectRF = joblib.load('Object_random_forest.pkl')
objectRF = joblib.load('Object_random_forest.pkl')
```

In[1]:

```
objectRF.feature_importances_
```

In[24]:

```
object_pred = objectRF.predict(x_object_test)
object_accuracy = accuracy_score(object_y_test, object_pred)
print("Object-based classification accuracy:", object_accuracy)
```

In[25]:

```
conf_matrix = confusion_matrix(object_y_test, object_pred)
```

In[26]:

```
# Define classes
classes = ['Heathland', 'Forest', 'Water', 'Wetland', 'Exposed Rock']

def calculate_accuracies(y_true, y_pred, classes):
    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y_true, y_pred)

    # Initialize lists to store producer and user accuracies
    producer_accuracies = []
    user_accuracies = []
    f1_scores = []

    # Calculate producer and user accuracy for each class
    for i in range(len(classes)):
        true_positives = conf_matrix[i, i]
        false_negatives = sum(conf_matrix[i, :]) - true_positives
        false_positives = sum(conf_matrix[:, i]) - true_positives

        producer_accuracy = true_positives / (true_positives + false_negatives)
        user_accuracy = true_positives / (true_positives + false_positives)

        producer_accuracies.append(producer_accuracy)
        user_accuracies.append(user_accuracy)

        # Calculate F1 score
        f1_score = 2 * (producer_accuracy * user_accuracy) / (producer_accuracy +
user_accuracy) if (producer_accuracy + user_accuracy) > 0 else 0
        f1_scores.append(f1_score)

    # Calculate overall accuracy
    overall_accuracy = sum(np.diag(conf_matrix)) / np.sum(conf_matrix)

    # Calculate Cohen's Kappa coefficient
    kappa = cohen_kappa_score(y_true, y_pred)

    # Calculate overall F1 score
    overall_f1_score = np.mean(f1_scores)
```

```
    # Return producer and user accuracies, F1 scores, overall accuracy, and kappa coefficient
    return producer_accuracies, user_accuracies, f1_scores, overall_accuracy, kappa, overall_f1_score

# Calculate object-based accuracies
object_producer_accuracies, object_user_accuracies, object_f1_scores, object_overall_accuracy, object_kappa, object_overall_f1 = calculate_accuracies(object_y_test, object_pred, classes)

# Calculate pixel-based accuracies
pixel_producer_accuracies, pixel_user_accuracies, pixel_f1_scores, pixel_overall_accuracy, pixel_kappa, pixel_overall_f1 = calculate_accuracies(pixel_y_test, pixel_pred, classes)

# Create a DataFrame for object-based accuracies
object_data = {
    'Class': classes,
    'Object PA': object_producer_accuracies,
    'Object UA': object_user_accuracies,
    'F1 Score': object_f1_scores
}

object_df = pd.DataFrame(object_data)

# Create a DataFrame for pixel-based accuracies
pixel_data = {
    'Class': classes,
    'Pixel PA': pixel_producer_accuracies,
    'Pixel UA': pixel_user_accuracies,
    'F1 Score': pixel_f1_scores
}

pixel_df = pd.DataFrame(pixel_data)

# Display accuracies in table format
print("Object-Based Accuracies:")
print(object_df)
```

```
print("\nPixel-Based Accuracies:")
print(pixel_df)

# Print overall accuracy, kappa coefficient, and overall F1 score
print("\nOverall Metrics:")
print(pd.DataFrame({
    'Metric': ['Overall Accuracy', 'Kappa Coefficient', 'Overall F1 Score'],
    'Object-Based': [object_overall_accuracy, object_kappa, object_overall_f1],
    'Pixel-Based': [pixel_overall_accuracy, pixel_kappa, pixel_overall_f1]
}))

pixel_stack = np.stack([
    pixel_red[ymin:ymax, xmin:xmax].flatten(),
    pixel_green[ymin:ymax, xmin:xmax].flatten(),
    pixel_blue[ymin:ymax, xmin:xmax].flatten(),
    pixel_NIR[ymin:ymax, xmin:xmax].flatten(),
    pixel_NDVI[ymin:ymax, xmin:xmax].flatten()
]).T

# Unpack pixel stack into individual variables with adjusted names
pixel_red_region, pixel_green_region, pixel_blue_region, pixel_NIR_region,
pixel_NDVI_region = pixel_stack[:, 0], pixel_stack[:, 1], pixel_stack[:, 2],
pixel_stack[:, 3], pixel_stack[:, 4]

object_stack = np.stack([
    object_red[ymin:ymax, xmin:xmax].flatten(),
    object_green[ymin:ymax, xmin:xmax].flatten(),
    object_blue[ymin:ymax, xmin:xmax].flatten(),
    object_NIR[ymin:ymax, xmin:xmax].flatten(),
    object_NDVI[ymin:ymax, xmin:xmax].flatten()
]).T

# Unpack object stack into individual variables with adjusted names
object_red_region, object_green_region, object_blue_region,
object_NIR_region, object_NDVI_region = object_stack[:, 0], object_stack[:, 1],
object_stack[:, 2], object_stack[:, 3], object_stack[:, 4]
```

```
# Normalization for pixel region
pixel_red_normalized_region = (pixel_red_region -
np.nanmin(pixel_red_region)) / (np.nanmax(pixel_red_region) -
np.nanmin(pixel_red_region))
print(pixel_red_normalized_region)
pixel_green_normalized_region = (pixel_green_region -
np.nanmin(pixel_green_region)) / (np.nanmax(pixel_green_region) -
np.nanmin(pixel_green_region))
pixel_blue_normalized_region = (pixel_blue_region -
np.nanmin(pixel_blue_region)) / (np.nanmax(pixel_blue_region) -
np.nanmin(pixel_blue_region))

pixel_rgb_region = np.stack([pixel_red_normalized_region,
pixel_green_normalized_region, pixel_blue_normalized_region])

# Normalization for object region
object_red_normalized_region = (object_red_region / 10000)
object_green_normalized_region = (object_green_region / 10000)
object_blue_normalized_region = (object_blue_region / 10000)

object_rgb_region = np.stack([object_red_normalized_region,
object_green_normalized_region, object_blue_normalized_region])


# In[97]:


# Make predictions for the pixel-based model
pixel_predictions = pixelRF.predict(pixel_stack)

# Make predictions for the object-based model
object_predictions = objectRF.predict(object_stack)

# Define colors for each class
class_colors = {
    0: '#d6382c',
    1: '#23bf01',
    2: '#1d7dff',
```

```
    3: '#8e6410',
    4: '#d6d6d6'
}


# Create a figure with two subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

axes[0, 0].imshow(pixel_NDVI[ymin:ymax, xmin:xmax], cmap='RdYlGn',
vmin=-5000, vmax=8000, extent=[xmin, xmax, ymin, ymax])
axes[0, 0].set_title("Pixel Image NDVI")

# Plot the pixel-based model predictions with custom colors
axes[1, 0].imshow(pixel_predictions.reshape(ymax - ymin, xmax - xmin),
cmap=ListedColormap([class_colors[i] for i in range(len(class_colors))]),
extent=[xmin, xmax, ymax, ymin])
axes[1, 0].set_title('Pixel-based Model Predictions')
axes[1, 0].set_xlabel('X Coordinate')
axes[1, 0].set_ylabel('Y Coordinate')

axes[0, 1].imshow(object_NDVI[ymin:ymax, xmin:xmax], cmap='RdYlGn',
vmin=-5000, vmax=8000, extent=[xmin, xmax, ymin, ymax])
axes[0, 1].set_title("Object Based NDVI")

# Plot the object-based model predictions with custom colors
axes[1, 1].imshow(object_predictions.reshape(ymax - ymin, xmax - xmin),
cmap=ListedColormap([class_colors[i] for i in range(len(class_colors))]),
extent=[xmin, xmax, ymax, ymin])
axes[1, 1].set_title('Object-based Model Predictions')
axes[1, 1].set_xlabel('X Coordinate')
axes[1, 1].set_ylabel('Y Coordinate')


# Show the plot
plt.tight_layout()
plt.show()
```

```
# In[98]:
# Extract false positives and false negatives for Heathland and Wetland
classes for Object based
false_positives_heathland = np.where((object_y_test == 0) & (object_pred !=
0))[0]
false_negatives_heathland = np.where((object_y_test != 0) & (object_pred ==
0))[0]

false_positives_wetland = np.where((object_y_test == 3) & (object_pred != 3))[0]
false_negatives_wetland = np.where((object_y_test != 3) & (object_pred ==
3))[0]

# Display the number of false positives and false negatives
print("False Positives for Heathland:", len(false_positives_heathland))
print("False Negatives for Heathland:", len(false_negatives_heathland))

print("False Positives for Wetland:", len(false_positives_wetland))
print("False Negatives for Wetland:", len(false_negatives_wetland))




# In[110]:
# path to the multi-band covariates file
covariate_file = object_image

# path to the object model file (.pkl)
model_file = 'Object_random_forest.pkl'

# path to the output file (will be created, or overwritten if it already exists)
outfile = 'ObjectPredictedImage.tif'


# In[112]:


clf = joblib.load(model_file)
```

```
with rasterio.open(covariate_file) as src:
    output_meta = src.profile

output_meta.update(count = 1, dtype=np.uint16, nodata=65535.0, compress =
"lzw")

def predict_row(row):
    w = ((row,row+1), (None, None))
    with rasterio.open(covariate_file) as src:
        nodata = src.profile['nodata']
        X = src.read(window = w)[:,0,:]
    y_pred = objectRF.predict(X.T)
    mask = np.where((X == nodata).any(axis=0))
    y_pred[mask] = nodata  # Set predicted nodata values to the actual nodata
value
    return y_pred


out_ds = rasterio.open(outfile, "w", **output_meta)

try:
    print(f"Predicting {output_meta['height']} rows:", flush = True)
    for row in range(output_meta['height']):
        if row % 100 == 0:
            print(row, "...", sep = '', end = '', flush = True)
        y_pred = predict_row(row).reshape((1, output_meta['width']))
        w = ((row, row+1), (None, None))
        out_ds.write_band(1, y_pred, window = w)
    print("done.")
finally:
    out_ds.close()


# In[117]:

# path to the multi-band covariates file
pixel_covariate_file = pixel_image_path
```

```
# path to the pixel model file (.pkl)
pixel_model_file = 'Pixel_random_forest.pkl'

# path to the output file (will be created, or overwritten if it already exists)
pixel_outfile = 'PixelPredictedImage.tif'


# In[118]:


clf = joblib.load(pixel_model_file)

with rasterio.open(pixel_covariate_file) as src:
    output_meta = src.profile

output_meta.update(count = 1, dtype=np.uint16, nodata=65535.0, compress =
"lzw")

def predict_row(row):
    w = ((row,row+1), (None, None))
    with rasterio.open(pixel_covariate_file) as src:
        nodata = src.profile['nodata']
        X = src.read(window = w)[:,0,:]
    y_pred = objectRF.predict(X.T)
    mask = np.where((X == nodata).any(axis=0))
    y_pred[mask] = nodata  # Set predicted nodata values to the actual nodata
value
    return y_pred


out_ds = rasterio.open(pixel_outfile, "w", **output_meta)

try:
    print(f"Predicting {output_meta['height']} rows:", flush = True)
    for row in range(output_meta['height']):
        if row % 100 == 0:
            print(row, "...", sep = '', end = '', flush = True)
```

```
        y_pred = predict_row(row).reshape((1, output_meta['width']))
        w = ((row, row+1), (None, None))
        out_ds.write_band(1, y_pred, window = w)
    print("done.")
finally:
    out_ds.close()
```