Published in Towards Data Science

Syam Kakarla Follow
Mar 5, 2021 · 7 min read · · Listen

MACHINE LEARNING | REMOTE SENSING

Land Cover Classification of Satellite Imagery using Python

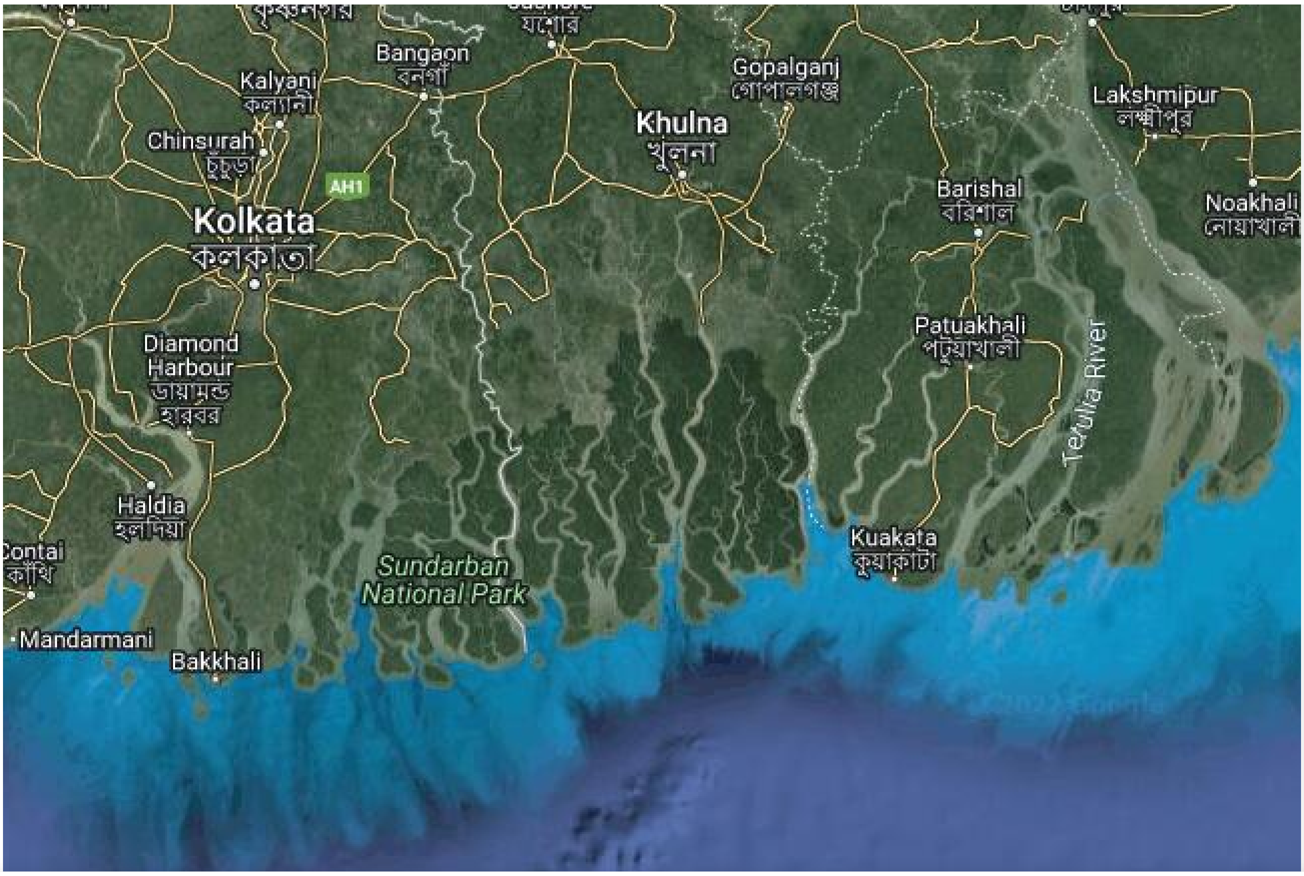
Land cover classification of Sundarbans satellite imagery using K-Nearest Neighbor(K-NNC), Support Vector Machine (SVM), and Gradient Boosting classification algorithms with Python.

Photo by Paulo Simões Mendes on Unsplash

Table of Contents

Let's Get Started…

Sundarbans Satellite Imagery



The Sundarbans is one of the largest mangrove areas in the delta formed by the confluence of the Ganges, Brahmaputra, and Meghna rivers in the Bay of Bengal. The Sundarbans forest is about 10,000 sq km across India and Bangladesh, of which 40% lies in India, and is home to many rare and globally threatened wildlife species. The above Google Map shows the Sundarbans region. In this article, we are going to use a part of the *Sundarbans* satellite data which is acquired using the *Sentinel-2* Satellite on 27 January 2020.

Let's start coding..,

Read Data

let's read the 12 bands using rasterio and stack them into an n-dimensional array using numpy.stack() method. The resultant data after stacking has the shape (12, 954, 298). The ground truth of the satellite image is read using the *loadmat* method from the *scipy.io* package. The ground truth has 6 classes which include water, plants, trees, bare land, e.t.c.

```
from glob import glob
import numpy as np
from scipy.io import loadmat
import rasterio as rio

S_sentinel_bands = glob("/content/drive/MyDrive/Satellite_data/sundarbans_data/*B?*.tiff")
S_sentinel_bands.sort()

l = []
for i in S_sentinel_bands:
  with rio.open(i, 'r') as f:
    l.append(f.read(1))

# Data
arr_st = np.stack(l)

# Ground Truth
y_data = loadmat('Sundarbands_gt.mat')['gt']
```

read_data.py hosted with ❤ by GitHub view raw

RGB Composite Image makes it easier to understand the data effectively. To plot RGB composite images, you will plot the red, green, and blue bands, which are bands 4, 3, and 2, respectively. Since Python uses a zero-based index system, so you need to subtract a value of 1 from each index. Therefore, the index for the red band is 3, green is 2, and blue is 1.

The Composite images that we created can sometimes be dark if the pixel brightness values are skewed toward the value of zero. This type of problem can be solved by stretching the pixel brightness values in an image using the argument `stretch=True` to extend the values to the full 0-255 range of potential values to increase the visual contrast of the image. Also, the `str_clip` argument allows you to specify how much of the tails of the data that you want to clip off. The larger the number, the more the data will be stretched or brightened.

Let's see the code to plot the RGB composite image along with the stretch applied.

```
ep.plot_rgb(
    arr_st,
    rgb=(3, 2, 1),
    stretch=True,
    str_clip=0.02,
    figsize=(12, 16),
    # title="RGB Composite Image with Stretch Applied",
)

plt.show()
```

plot_rgb.py hosted with ❤ by GitHub view raw

Let's visualize the ground truth using the *plot_bands* method from *eathpy.plot* package.

```
# Visualize Groundtruth

ep.plot_bands(y_data,
              cmap=ListedColormap(['darkgreen', 'green', 'black',
                                   '#CA6F1E', 'navy', 'forestgreen']))
plt.show()
```

plot_gt.py hosted with ❤ by GitHub view raw

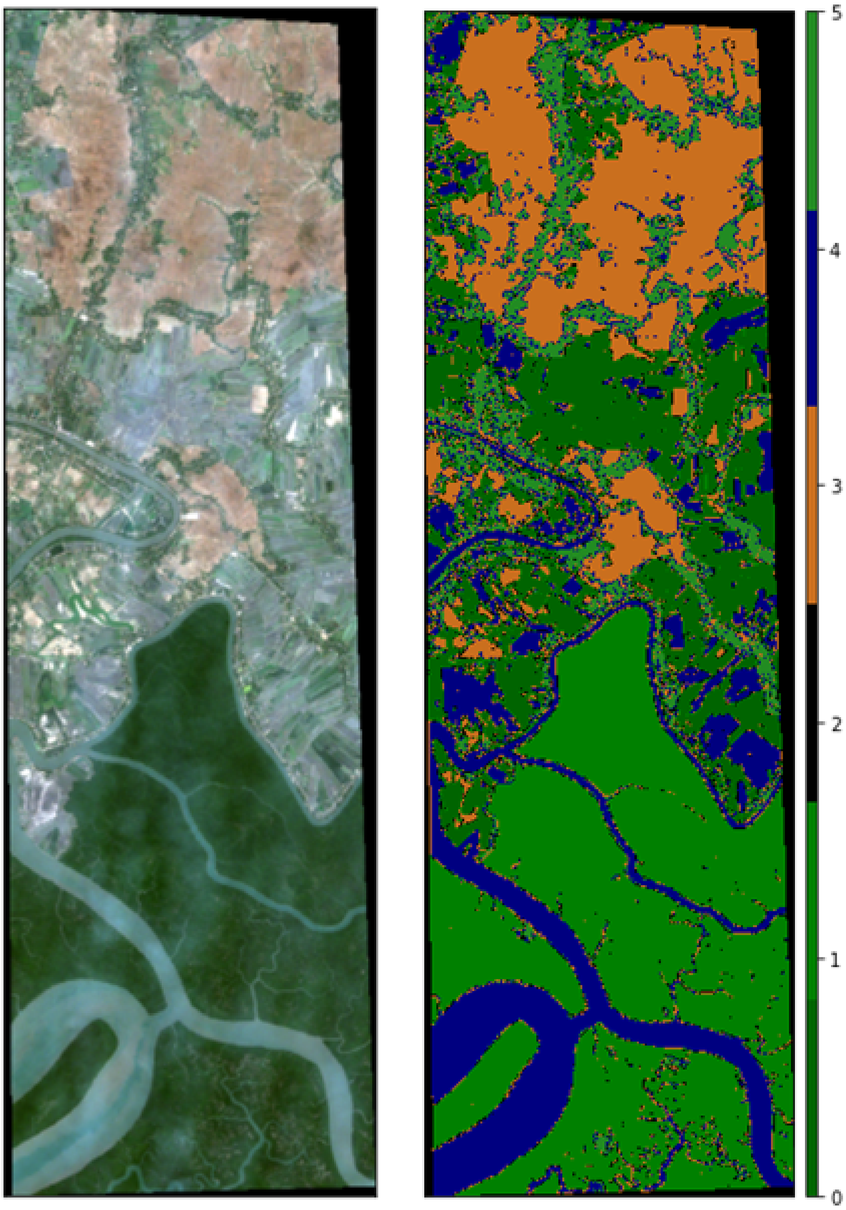
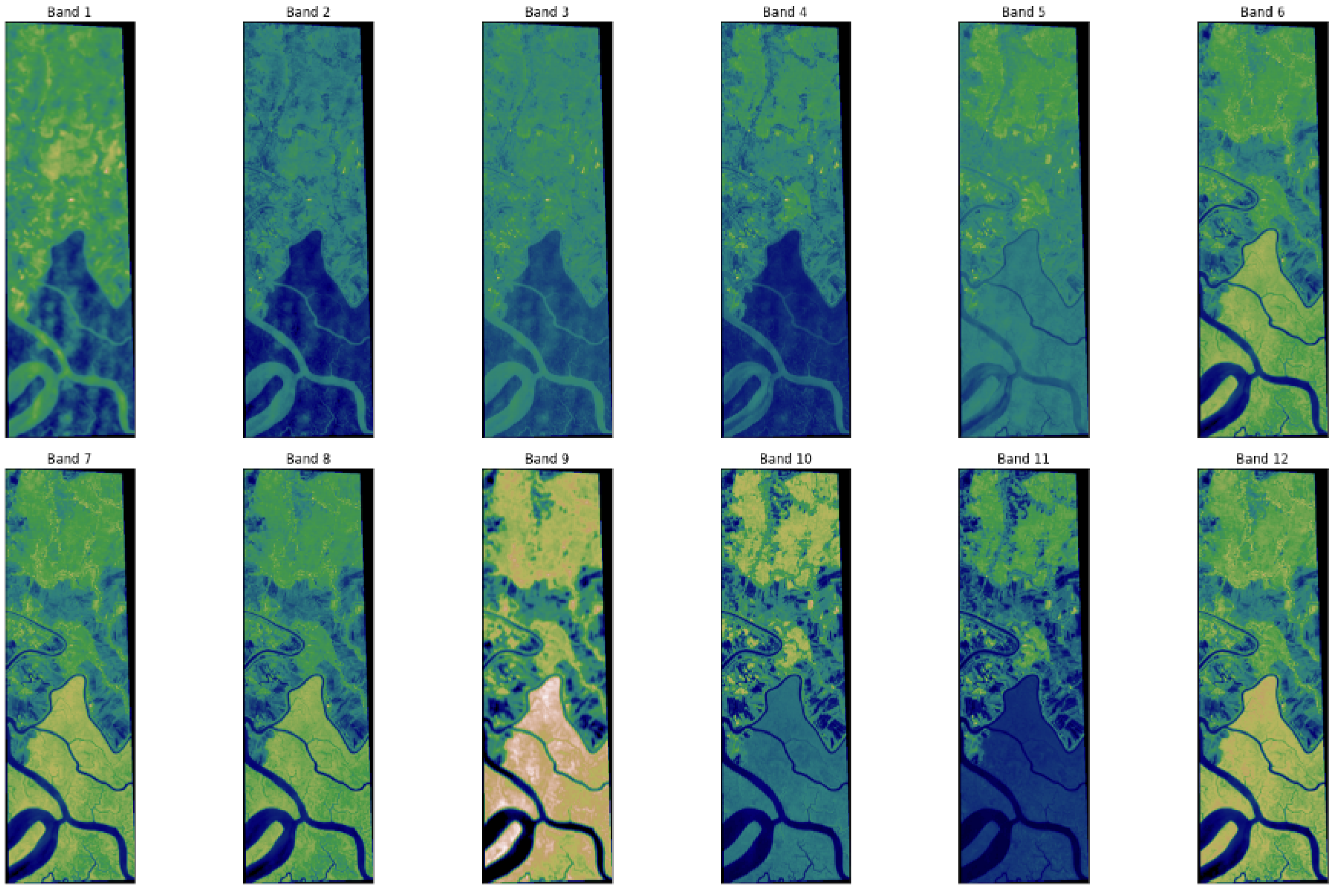


Image by Author

As we discussed, the data contains 12 bands. let's visualize each band using the EarhPy package. the `plot_bands()` the method takes the stack of the bands and plots along with custom titles which can be done by passing unique titles for each image as a list of titles using the `title=` parameter.

```
ep.plot_bands(arr_st,
              cmap = 'gist_earth',
              figsize = (20, 12),
              cols = 6,
```



Visualization of Bands — Image by Author

Preprocessing

Standardization is another **scaling** technique where the values are centered around the mean with a unit **standard** deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit **standard** deviation. The scaled data is divided into train and test data in the ratio of 30:70. The below code is used to scale and split the data.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

x = np.moveaxis(arr_st, 0, -1)

X_data = x.reshape(-1, 12)
scaler = StandardScaler().fit(X_data)
X_scaled = scaler.transform(X_data)

# Split data
```

K-Nearest Neighbor Classifier (K-NNC)

k-Nearest neighbor classifier is one of the widely used classifiers in machine learning. The main objective of this method is that the data instances of the same class should be closer in the feature space.

Let us consider a dataset with n data points represented as `f(x1, y1), (x2, y2)...(xi, yi)...(xn, yn)`. Where `xi` and `yi` are feature vector and corresponding class label respectively. For a new data point `p`, the class label can be predicted by k-NNC with a `k` value where `k` is the number of neighboring data points as follows:

$$f(p, k) = y_c, \; c = \arg\max_i \parallel p - x_i \parallel^2$$

K-KNC, Image by author

We are going to implement k-NNC using the scikit learn package. The below code K-NNC instance with `n_neighbors` as 6 and fits the train data, predicts the labels of the test data, shows the accuracy, and prints the classification report which includes precision, recall and F1-score of each class. The K-NNC has shown **98.94%** accuracy over the test data.

```
from sklearn.neighbors import KNeighborsClassifier

# K-NNC
knn = KNeighborsClassifier(n_neighbors=6)

knn.fit(X_train, y_train)

# Predict the labels of test data

knn_pred = knn.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, knn_pred)*100}")
```

	precision	recall	f1-score	support
0	0.98	0.99	0.98	16222
1	1.00	1.00	1.00	23570
2	1.00	1.00	1.00	6095
3	0.99	0.99	0.99	16790
4	0.99	0.99	0.99	13545
5	0.98	0.95	0.97	9066
accuracy			0.99	85288
macro avg	0.99	0.99	0.99	85288
weighted avg	0.99	0.99	0.99	85288

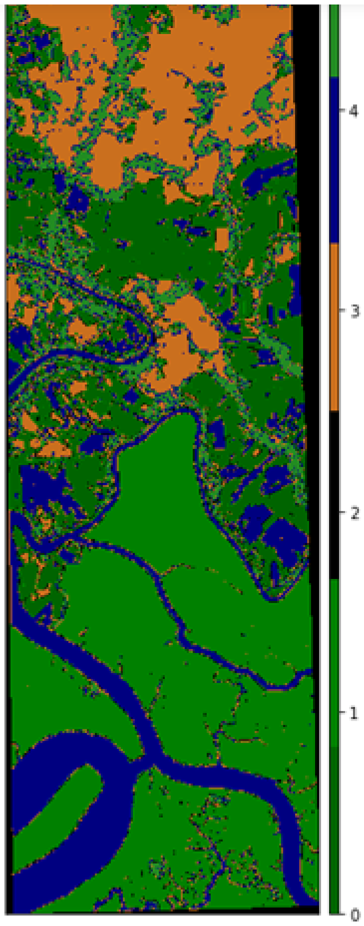
K-NNC Classification Report — Image by Author

Let's visualize the classification map of K-NNC, the below code is used to predict the labels of the Sundarbans data and plot the data using `plot_bands()` method from the earthpy package.

```
# Visualize Classification Map of K-NNC

ep.plot_bands(knn.predict(X_scaled).reshape((954, 298)),
              cmap=ListedColormap(['darkgreen', 'green', 'black',
                                   '#CA6F1E', 'navy', 'forestgreen']))
plt.show()
```

knn_cls_map.py hosted with ❤ by GitHub view raw



Classification Map using K-NNC — Image by Author

Support Vector Machine (SVM)

The support vector machine (SVM) is a supervised learning method that generates input-output mapping functions from a set of labeled training data. The mapping function can be either a classification function, i.e., the category of the input data, or a regression function.

For classification, nonlinear kernel functions such as Radial Basis Function(RBF), Polynomial, Sigmoid, e.t.c are often used to transform input data to a high-dimensional feature space in which the input data become more separable compared to the original input space. Maximum-margin hyperplanes are then created. The model thus produced depends on only a subset of the training data near the class boundaries.

The below code is used to create an instance of SVM with the regularization parameter `C` as 3 and RBF kernel. Fits the data, predict the labels for test data, and prints the accuracy and classification report.

```
# Fit Data
svm.fit(X_train, y_train)

# Predict labels for test data
svm_pred = svm.predict(X_test)

# Accuracy and Classification Reeport
print(f"Accuracy: {accuracy_score(y_test, svm_pred)*100}")
print(classification_report(y_test, svm_pred))
```

svm.py hosted with ❤ by GitHub view raw

The Support Vector Machine (SVM) algorithm has shown **99.88%** accuracy on the test data. The classification report is shown below:

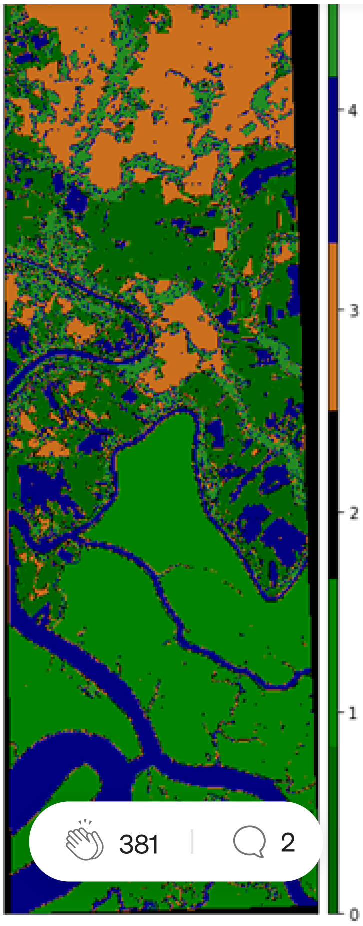
	precision	recall	f1-score	support
0	1.00	1.00	1.00	16222
1	1.00	1.00	1.00	23570
2	1.00	1.00	1.00	6095
3	1.00	1.00	1.00	16790
4	1.00	1.00	1.00	13545
5	1.00	1.00	1.00	9066
accuracy			1.00	85288
macro avg	1.00	1.00	1.00	85288
weighted avg	1.00	1.00	1.00	85288

Image by Author

Let's visualize the classification map of SVM, the below code is used to predict the labels of the Sundarbans data and plot the data using `plot_bands()` method from earthpy package.

```
# Visualize Classification Map of SVM

ep.plot_bands(svm.predict(X_scaled).reshape((954, 298)),
              cmap=ListedColormap(['darkgreen', 'green', 'black',
                                   '#CA6F1E', 'navy', 'forestgreen']))
plt.show()
```



Classification Map of Sundarbans using SVM — Image by Author

Gradient Boosting Classifier

Gradient boosting is a technique attracting attention for its prediction speed and accuracy, especially with large and complex data. Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model to minimize the error.

Today we are going to use **lightGBM** which is a gradient boosting framework that makes use of tree-based learning algorithms. LightGBM is called “**Light**” because of its computation power and giving results faster. It takes **less memory to run** and can **deal with large amounts of data**. It is one of the most widely used algorithms in competition because the motive of the algorithm is to get good accuracy of results.

The below code is used to create an instance of the `lightgbm` with parameters such as learning rate, maximum depth, number of classes, e.t.c

The `lightbgm` classifier has shown **98.55%** accuracy over the test data and the classification report is shown below.

Let's visualize the classification map generated using `lightgbm` classifier. The below code is used to predict the labels of the Sundarbans data and plot the data using `plot_bands()` method from the earthpy package.

Classification Map of Sundarbans using lightgbm classifier — Image by Author

Conclusion

The article shows how to implement K-NNC, SVM, and LightGBM classifiers for land cover classification of Sundarbans satellite data using Python. The Support Vector Machine has shown better performance compared to K-Nearest Neighbor Classifier (K-NNC) and LightGBM classifier. The below figure shows the classification maps of the above-mentioned three classifiers.

Comparison of the classifiers based on Accuracy — Image by Author

The code used in this article can be accessed from the below GitHub repository.

syamkakarla98/Satellite_Imagery_Analysis

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

Happy Learning ✨

More from Author

WildFire Detection using Satellite Imagery with Python

Satellite Imagery Meets Computer Vision: A Tutorial for wildfire detection in Australia using Python

Detectron2

Use the power of Transfer Learning

towardsdatascience.com

Beginners Guide to PySpark

Chapter 1: Introduction to PySpark using US Stock Price Data

towardsdatascience.com

Hyperspectral Image Analysis — Getting Started

A Walkthrough on Hyperspectral Image Analysis Using Python.

towardsdatascience.com

References

Comprehensive Guide to Satellite Imagery Analysis using Python

Different methods and Machine Learning techniques to analyze satellite imagery using Python with hands-on tutorials and...

towardsdatascience.com

Ground Truth Labeling of Satellite Imagery using K-Means Clustering with Python

A simple tutorial on Ground Truth labeling of satellite imagery using K-Means Clustering Algorithm using Python

towardsdatascience.com

the . Parameters n_neighborsint, default=5...

scikit-learn.org

sklearn.svm.SVC - scikit-learn 0.24.1 documentation

C-Support Vector Classification. The implementation is based on libsvm. The fit time scales at least quadratically with...

scikit-learn.org

Welcome to LightGBM's documentation! - LightGBM 3.1.1.99 documentation

Edit description

lightgbm.readthedocs.io