## Python solution

# Manual for the Script

## Classification of Crop Types using Multi-temporal NDVI and Random Forest Classifier

**Name: Rezaul Hasan Bhuiyan**
**Student Id: S3003337**
**Specialization: Natural Resources Management**
**Email: r.h.bhuiyan@student.utwente.nl**

## Introduction:

Python is a powerful tool for geospatial data analysis. Python has many useful libraries like rasterio, geopandas, fiona, scipy and scikit-learn. These helps combine geospatial and machine learning methods smoothly, making the work easier. Also, Python's readability, adaptability, and abundant resources encourage innovation, simplifying project scalability and replication. Additionally, Python scripts are compatible with various IDEs, this project has been developed in Spyder.

## Scripting Manual

This scripting manual provides a comprehensive overview of the Python script, outlining each section's purpose and functionality.

1. Preparation:
    - Download the required data:
        - NDVI_12.tiff (12 NDVI of 12 months)
        - samplingPoints,shp (929 labels of 7 croptypes)

Note: The data rights preserve to Mariana Belgiu (Associate Professor, Department of Earth Observation Science)

- Install and import necessary packages.
    - **rasterio:** For reading and writing raster data.
    - **geopandas:** For working with geospatial data.
    - **fiona:** For reading and writing vector data.
    - **matplotlib.pyplot:** For creating visualizations.
    - seaborn: For enhancing the aesthetics of plots.
    - **numpy:** For numerical computations.
    - **pandas:** For data manipulation and analysis.
    - **sklearn.metrics:** For evaluating model performance metrics.
    - **sklearn.ensemble.RandomForestClassifier:** For implementing the Random Forest classifier.
    - **sklearn.model_selection.train_test_split:** For splitting the data into training and testing sets.
    - **os:** For interacting with the operating system.

```
#Importing necessary packages
import rasterio
from rasterio.plot import show
import geopandas as gpd
import fiona
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import os
```

- Setting Up Working Directory

Setting the working directory to the location where the dataset is stored.

  - BASE_DIR: Path to the base directory containing the dataset.

```
## Setting up working directory
BASE_DIR = "C:/Users/User/Desktop/Python_solution/Dataset"
os.chdir(BASE_DIR)
```

- Assigning variables for Input and Output files

Specifying input raster and vector files, and output files.

Variables:

  - ndviStack: Path to the NDVI raster file.
  - samplingPoints: Path to the sampling points shapefile.
  - temp_point_data: Temporary file name for storing point data.
  - cropType_name: Names of crop types for visualization.
  - band_names: Names of NDVI bands corresponding to months of the year.

2. Inspecting the Data and Visualization:
   - Open the NDVI raster file and retrieve its metadata:

```
In [182]: ndvi = rasterio.open(ndviStack)
     ...: ndvi.meta
Out[182]:
{'driver': 'GTiff',
 'dtype': 'float32',
 'nodata': None,
 'width': 2137,
 'height': 1310,
 'count': 12,
 'crs': CRS.from_epsg(32631),
 'transform': Affine(10.0, 0.0, 543290.0,
        0.0, -10.0, 5719580.0)}
```

It is mandatory to have same coordinate system (red marked) of the NDVI raster file and the sampling points shapefile. Otherwise, the model will not run.

- Read the sampling points shapefile and determine its shape and CRS.

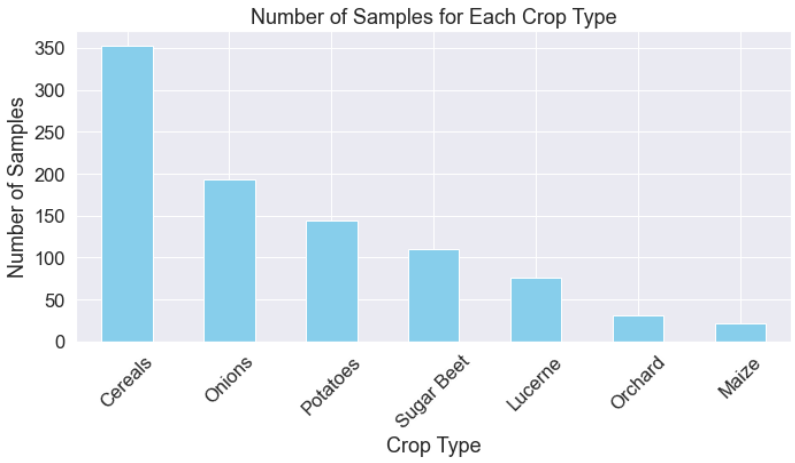
```
In [184]: Points.shape
Out[184]: (929, 3)

In [185]: Points.crs
Out[185]:
<Projected CRS: EPSG:32631>
Name: WGS 84 / UTM zone 31N
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: World - N hemisphere - 0°E to 6°E - by country
- bounds: (0.0, 0.0, 6.0, 84.0)
Coordinate Operation:
- name: UTM zone 31N
- method: Transverse Mercator
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

If the coordinate doesn't match, one may try the following code. In the main script it has been commented out by using "#"

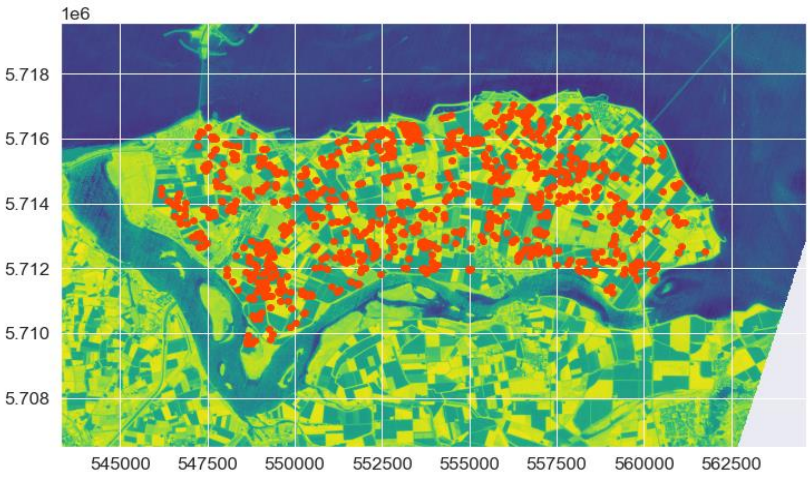
```
# # if the projection system is not matching try this
# with rasterio.open(ndviStack) as ndvi_file:
#     # Read the metadata
#     ndvi_meta = ndvi_file.meta
#     # Update the CRS (Coordinate Reference System) information
#     new_crs = 'EPSG:4326'  # Example
#     ndvi_meta['crs'] = new_crs
# # Update the metadata
# ndvi_meta.update()
```

- Count occurrences of each crop type in the sampling points and plot a histogram showing the number of samples for each crop type:

  It's a good practice to inspect sampling data and check the class occurrences, it may help to explain model performance.



- Visualize the sampling points and NDVI layer in the same figure to understand the sampling distribution.




3. Feature Extraction for RF model
   - Open the NDVI images as arrays using rasterio, and load the shapefile as a dataframe with geopandas. To start feature extraction, give each item a distinct ID and store this as a temporary file.

```
## Feature Extraction for RF model
# reading ndvi bands from input
with rasterio.open(ndviStack) as img:
    bands = img.count
# Using iteration to assign band names from the band_names list
features = [band_names[i] for i in range(bands)]
print('Bands names: ', features)
f_len = len(features)
# Read sampling points data, assign IDs, and save to a temporary point file
points = gpd.read_file(samplingPoints)
# adding a new column 'id' with range of points
points = points.assign(id=range(len(points)))
# saving new point file with 'id'
points.to_file(temp_point_data)

# Converting gdf to pd df and removing geometry
points_df = pd.DataFrame(points.drop(columns='geometry'))
```

   - Then, generate a blank pandas series dataframe and loop through each label using Fiona and a Python for loop. During each loop iteration, store the corresponding values of each band (and additional features like NDVI) for each label position. Once the loop completes, restructure the data into a list format.

```
# ilterating over multiband raster
sampled = pd.Series()
# Read input shapefile with fiona and iterate over each feature
with fiona.open(temp_point_data) as shp:
    for feature in shp:
        #print(feature)
        siteID = feature['properties']['id']
        coords = feature['geometry']['coordinates']
        # Read pixel value at the given coordinates using Rasterio
        # NB: `sample()` returns an iterable of ndarrays.
        with rasterio.open(ndviStack) as stack_src:
                value = [v for v in stack_src.sample([coords])]
        # Update the pandas serie accordingly
        sampled.loc[siteID] = value

# Reshape sampled values into DataFrame
df1 = pd.DataFrame(sampled.values.tolist(), index=sampled.index)
df1['id'] = df1.index
df1 = pd.DataFrame(df1[0].values.tolist(), columns=features)
df1['id'] = df1.index

# Merging dataset for trainig model
data = pd.merge(df1, points_df, on ='id')
print('Sampled Data: \n',data)
```

4. Split data into training and testing sets:

   After sampling the features, split the data into X (features) and Y (labels). Then, use the train_test_split function from Scikit-learn to divide the data into a 70/30 ratio, allocating 70% for training and 30% for testing.

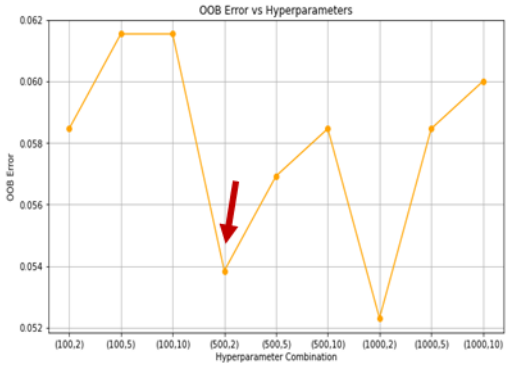
```
## Split data into training and testing sets
x = data.iloc[:,0:f_len]
X = x.values
y = data.iloc[:,-1]
Y = y.values

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, stratify = Y)
print(f'X_train Shape: {X_train.shape}\nX_test Shape: {X_test.shape}\ny_train Shape: {y_train.shape}\ny_test Shape:{y_test.shape}')
```

5. Optimizing and Training Model:

   This study adopted random forest (cName = RF) for this classification task based on existing literature.

```
## Training Model
cName = 'RF' #classifier Name

##Compare hyperparameter combinations to find the best settings using OOB error and accuracy.
# Initialize lists to store results
accuracies = []
oob_errors = []

# Define different values for hyperparameters
n_estimators_values = [100, 500, 1000]
max_features_values = [2, 5, 10]

# Iterate over different hyperparameter values
for n_estimators in n_estimators_values:
    for max_features in max_features_values:
        # Initialize Random Forest classifier with specified hyperparameters
        clf = RandomForestClassifier(n_estimators=n_estimators, max_features=max_features, random_state=42, oob_score=True)
        clf.fit(X_train, y_train)

        # Evaluate classifier performance on testing data
        clf_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test, clf_pred)
        accuracies.append(accuracy)

        # Get OOB error
        oob_error = 1 - clf.oob_score_
        oob_errors.append(oob_error)

# Plot accuracy
plt.figure(figsize=(10, 6))
plt.plot(range(len(accuracies)), accuracies, marker='o')
plt.xticks(range(len(accuracies)), [f"({n},{m})" for n in n_estimators_values for m in max_features_values])
plt.xlabel('Hyperparameter Combination')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Hyperparameters')
plt.grid(True)
plt.show()

 # Plot OOB error
 plt.figure(figsize=(10, 6))
 plt.plot(range(len(oob_errors)), oob_errors, marker='o', color='orange')
 plt.xticks(range(len(oob_errors)), [f"({n},{m})" for n in n_estimators_values for m in max_features_values])
 plt.xlabel('Hyperparameter Combination')
 plt.ylabel('OOB Error')
 plt.title('OOB Error vs Hyperparameters')
 plt.grid(True)
 plt.show()
```

This study implemented different combinations of max feature values to and number of estimators to find the optimized one. Compared hyperparameter combinations with respect to OOB error and accuracy to find the best settings. And we took [n_estimator_value, max_feature_value] = [500,2] as the optimized parameters.





Lastly, Train the model with the best hyperparameters settings. Once the model is initialized, use ‘‘fit’’ function to train the model with train x_train and y-train.

```
# Set hyperparameters to best combination for final prediction
ntree = 500  # Number of trees
mtry = 5      # Number of variables used for splitting

# Initialize Random Forest classifier with specified hyperparameters
clf = RandomForestClassifier(n_estimators=ntree, max_features=mtry, random_state=42)
clf.fit(X_train, y_train)
clf_pred = clf.predict(X_test)
```

6. Accuracy Assessment:

Following that, this study utilized the x-test and y-test data to examine sample predictions, and utilized Scikit-learn's accuracy_score to evaluate the model's accuracy concerning those specific sample predictions. And visualize confusion matrix to understand the classification accuracy for each class (croptype).

```
## Accuracy Assessment
print(f"Accuracy {cName}: {accuracy_score(y_test, clf_pred)*100}")
print(classification_report(y_test, clf_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, clf_pred)
# Calculate percentages for each class
cm_percent = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Plotting confusion matrix
plt.figure(figsize=(12, 8))
sns.set(font_scale=1.5)

# Iterate through the confusion matrix percentages and annotate the heatmap with formatted strings
for i in range(len(cm_percent)):
    for j in range(len(cm_percent[i])):
        plt.text(j+0.5, i+0.5, f"{cm_percent[i, j]:.1f}",
                 ha='center', va='center', color='black')

sns.heatmap(cm_percent, annot= False, cmap='Blues',
            xticklabels=cropType_name, yticklabels=cropType_name, cbar=True)
plt.title('Confusion Matrix', fontsize = 'large')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.tight_layout()
plt.show()
```

Furthermore, we calculated Producer and User Accuracy (PA and UA) for Each Class manually using numpy functions, lastly, kappa coefficient were calculated suing cohen_kappa_score function from sklearn,metrics module.

```
# Calculate producer accuracy (PA) and user accuracy (UA) for each class
producer_accuracy = np.diag(cm) / np.sum(cm, axis=1)
user_accuracy = np.diag(cm) / np.sum(cm, axis=0)

# Calculate overall producer and user accuracy
overall_producer_accuracy = np.mean(producer_accuracy)
overall_user_accuracy = np.mean(user_accuracy)

# Calculate Kappa coefficient
kappa_coefficient = cohen_kappa_score(y_test, clf_pred)

# Print the results
print("Producer Accuracy (PA):", producer_accuracy)
print("User Accuracy (UA):", user_accuracy)
print("Overall Producer Accuracy:", overall_producer_accuracy)
print("Overall User Accuracy:", overall_user_accuracy)
print("Kappa Coefficient:", kappa_coefficient)
```

7. Feature importance analysis:

   There is a built-in function “sklearn.enselmble.feature_importances_” available in sklearn.ensemble module that allows to assess feature importance. We used that tool to draw a feature importance hierarchy.

```
##Get feature importances
feature_importances = clf.feature_importances_
# Sort feature importances in descending order
indices = np.argsort(feature_importances)[::-1]
# Rearrange feature names so they match the sorted feature importances
sorted_features = [features[i] for i in indices]
# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances[indices], y=np.array(sorted_features))
plt.title("Feature Importance Hierarchy")
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.show()
```

8. Prediction for entire image, Visualization and Output: The important step is to predict the entire dataset using our trained model and then export the numerical predictions in their original format. The original input image was read, and various metadata properties (such as height, width, and CRS) have been obtained. Subsequently, the image has been reshaped in the same manner as the input data was provided for training. It is important to note that, the model fails to predict nan values in the input data (in any band or layer) and will through error. Therefore, we replaced the nan value to zero with the numpy function (np.where).

```
##full data reshaping, predicting Croptypes for whole image, and saving output
exp_name = f'Croptype_{cName}.tif'
img = rasterio.open(ndviStack)
img_arr = img.read()
img_arr = np.where(np.isnan(img_arr),0,img_arr) #replacing nana valuse with zeros
bands = img_arr.shape[0]
print(f'Height: {img_arr.shape[1]}\nWidth: {img_arr.shape[2]}\nBands: {img_arr.shape[0]}\n')
img_n = np.moveaxis(img_arr, 0, -1)
img_n.shape
img_n = img_n.reshape(-1, f_len)
print('reshaped full data shape  for prediction: ',img_n.shape)

# predict Croptypes for whole area
pred_CropTypes = clf.predict(img_n)
```

- Visualization:
  For visualization of the crop types, we incorporated colors for each crop based on the existing literature and took the visually intuitive color palette. However, the visualization missing a north arrow and a scale. To best of my knowledge, no python package or module offer that.

```
# Define colors for each crop type
colors = {
    'Cereals': '#FFC107',     # Yellow
    'Lucerne': '#4CAF50',     # Green
    'Maize': '#FF5722',       # Orange
    'Onions': '#795548',      # Brown
    'Orchard': '#673AB7',     # Purple
    'Potatoes': '#607D8B',    # Gray
    'Sugar Beet': '#E91E63'   # Pink
}

# Create a legend patch for each crop type
legend_patches = [plt.Rectangle((0,0),1,1,fc=colors[c], edgecolor='none') for c in cropType_name]

# Plot the classified image
plt.figure(figsize=(18, 16))
plt.imshow(pred_CropTypes.reshape(img_arr.shape[1], img_arr.shape[2]), cmap='tab10',interpolation='nearest')
plt.title('Map of Crop Types', fontsize='large')
plt.legend(handles=legend_patches,
           labels=cropType_name, loc='upper left',
           bbox_to_anchor=(1.0, 1),
           title='Crop Types',
           fontsize='large')
plt.axis('off')  # Remove axis
plt.show()
```

- Export the map:

```
## Predefining out raster meta using variable raster
tempfile_arr = img.read(1)
tempfile_arr = tempfile_arr.reshape(-1,1)
metadata = img.meta

height = metadata.get('height')
width = metadata.get('width')
crs = metadata.get('crs')
transform = metadata.get('transform')

img_reshape = pred_CropTypes.reshape(height, width)

# Write predicted values to output raster
out_raster = rasterio.open(exp_name,
                                          'w',
                                          driver='GTiff',
                                          height=height,
                                          width=width,
                                          count=1,
                                          dtype='uint8',
                                          crs=crs,
                                          transform = transform,
                                          nodata = 255 #nodata
                                          )

out_raster.write(img_reshape, 1)
out_raster.shape
```

9. Close all opened files:

   This a good coding practice to close opened file after completion of operation.

```
## Close all opened files
img.close()
ndvi.close()
out_raster.close()
```