Detecting deforestation in the Amazon rainforest using unsupervised K-medoids clustering on satellite imagery

Submitted by: Ahmed Khalid

Roll no: 200148

Submitted to: Dr Ashfaq

Introduction

Deforestation around the world has reached a critical level, causing irreversible damage to environmental sustainability that is contributing to climate change around the world. Widespread forest fires, from the Amazon Basin in Brazil, to the west coast of the United States, are raging all year-round. This notebook will allow us to detect deforested areas in the Brazilian Amazon rainforest, using satellite imagery.

Imports

```
import pandas as pd
from datetime import datetime
from IPython.display import Image, HTML
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime as dt

# Upgrade pip and setuptools
!pip install --upgrade pip setuptools

# Install a specific compatible version of timm
!pip install timm==0.4.12

# Ensure that the necessary packages are installed
try:
    from arcgis.gis import GIS
    from arcgis.learn import MLModel, prepare_tabulardata
    from arcgis.raster import Raster
    from arcgis.geometry import filters
except ModuleNotFoundError as e:
    missing_module = str(e).split("'")[1]
    if missing_module == 'arcgis':
        !pip install arcgis --use-deprecated=legacy-resolver
    # Retry import after installing the missing module
    from arcgis.gis import GIS
    from arcgis.learn import MLModel, prepare_tabulardata
    from arcgis.raster import Raster
    from arcgis.geometry import filters

from fastai.vision.all import *
```

```
already satisfied: pip in /usr/local/lib/python3.10/dist-packages (24.0)
already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (70.0.0)
ing pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recomme
already satisfied: timm==0.4.12 in /usr/local/lib/python3.10/dist-packages (0.4.12)
already satisfied: torch>=1.4 in /usr/local/lib/python3.10/dist-packages (from timm==0.4.12) (2.3.0+cu121)
already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from timm==0.4.12) (0.18.0+cu121)
already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (3.14.0)
already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (4.11.0)
already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (1.12)
already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (3.3)
already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (3.1.4)
already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (2023.6.0)
already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (12.1.
already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (12.
already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (12.1.
already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (8.9.2.26)
already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (12.1.3.1)
already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (11.0.2.54
already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (10.3.2.
already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (11.4.
already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (12.1.
already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (2.20.5)
already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (12.1.105)
```

```
ılready satisfied: triton==2.3.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.4->timm==0.4.12) (2.3.0)
ılready satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.
ılready satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision->timm==0.4.12) (1.25.2)
ılready satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision->timm==0.4.12) (9.4.0)
ılready satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.4->timm==0.4.12) (2.1.5)
ılready satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.4->timm==0.4.12) (1.3.0)
ıing pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recomme
```

Connecting to ArcGIS

```
# Import necessary modules
from arcgis.gis import GIS

# Establish GIS connection for the first instance (anonymous)
try:
    gis = GIS()
    print("Connected to GIS as anonymous user")
except Exception as e:
    print("Failed to connect to GIS:", e)

# Establish the second GIS connection (enterprise)
try:
    gis_enterp = GIS("https://pythonapi.playground.esri.com/portal", "arcgis_python", "amazing_arcgis_123")
    print("Connected to enterprise GIS")
except Exception as e:
    print("Failed to connect to enterprise GIS:", e)
```

```
Connected to GIS as anonymous user
Connected to enterprise GIS
```

Accessing & Visualizing datasets

Here, we use Sentinel-2 imagery, which has a high resolution of 10m and 13 bands. This imagery is accessed from the ArcGIS Enterprise portal, where it is sourced from the AWS collection.

```
# Get image
s2 = gis.content.get('fd61b9e0c69c4e14bebd50a9a968348c')
sentinel = s2.layers[0]
print(s2)
```

```
<Item title:"Sentinel-2 Views" type:Imagery Layer owner:esri>
```

Data Preparation

Define Area of Interest in the Amazon

The area of interest is defined using the four latitude and longitude values from a certain region of the Amazon rainforest where a considerable area of forest has been deforested, as can be seen from the images above.

```
# Extent in 3857 for Amazon rainforest
amazon_extent = {
    "xmin": -6589488.51,
    "ymin": -325145.08,
    "xmax": -6586199.09,
    "ymax": -327024.74,
    "spatialReference": {"wkid": 3857}
}
```

Filtering and Querying the Imagery

Let's filter the Sentinel imagery based on the specified criteria and convert the resulting data into a DataFrame for further processing

```
# Filter by criteria and extent
try:
    selected = sentinel.filter_by(
        where="(Category = 1) AND (cloudcover <= 0.05)",
        geometry=filters.intersects(amazon_extent)
    )

    # Query the filtered results
    df = selected.query(out_fields="AcquisitionDate, GroupName, CloudCover, DayOfYear",
                        order_by_fields="AcquisitionDate").sdf

    # Convert AcquisitionDate to datetime
    df['AcquisitionDate'] = pd.to_datetime(df['AcquisitionDate'], unit='ms')
    print(df.head())
except Exception as e:
    print("Failed to filter/query imagery:", e)
```

```
Failed to filter/query imagery: Token Required
(Error Code: 499)
```

Continue with Data Processing

We continue, proceed with loading, normalizing the data, and applying the clustering algorithms.

```
# Replace these with the actual file paths on your system
band_paths = [
    "/path/to/your/directory/B02.tif",
    "/path/to/your/directory/B03.tif",
    "/path/to/your/directory/B04.tif",
    "/path/to/your/directory/B08.tif"
]
```

```
import rasterio
import numpy as np
from sklearn.preprocessing import MinMaxScaler

bands = []

for band_path in band_paths:
    try:
        with rasterio.open(band_path) as src:
            bands.append(src.read(1))
    except Exception as e:
        print(f"Failed to load band from {band_path}: {e}")

if bands:
    bands = np.stack(bands, axis=-1)
    print("Bands loaded and stacked successfully.")

    # Normalize the data
    scaler = MinMaxScaler()
    bands_reshaped = bands.reshape(-1, bands.shape[-1])
    bands_normalized = scaler.fit_transform(bands_reshaped)
else:
    print("No bands were loaded. Please check the file paths and try again.")
```

```
Failed to load band from /path/to/your/directory/B02.tif: /path/to/your/directory/B02.tif: No such file or directory
Failed to load band from /path/to/your/directory/B03.tif: /path/to/your/directory/B03.tif: No such file or directory
Failed to load band from /path/to/your/directory/B04.tif: /path/to/your/directory/B04.tif: No such file or directory
Failed to load band from /path/to/your/directory/B08.tif: /path/to/your/directory/B08.tif: No such file or directory
No bands were loaded. Please check the file paths and try again.
```

Apply K-means & K-medoids Clsutering

Now, we'll apply K-means clustering to the normalized data.

We'll use the K-medoids algorithm to cluster the data.

```
!pip install scikit-learn-extra
```

```
Requirement already satisfied: scikit-learn-extra in /usr/local/lib/python3.10/dist-packages (0.3.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.25.2)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.11.4)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-ext
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-le
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager.
```

```
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Ensure bands_normalized is defined
if 'bands_normalized' in locals():
    # Apply K-means clustering
    kmeans = KMeans(n_clusters=5, random_state=42)
    kmeans.fit(bands_normalized)
    clusters_kmeans = kmeans.labels_
    clusters_kmeans_image = clusters_kmeans.reshape(bands.shape[:-1])

    # Apply K-medoids clustering
    kmedoids = KMedoids(n_clusters=5, random_state=42, method='pam')
    kmedoids.fit(bands_normalized)
    clusters_kmedoids = kmedoids.labels()
    clusters_kmedoids_image = clusters_kmedoids.reshape(bands.shape[:-1])

    # Visualize the clustering results
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(clusters_kmeans_image, cmap='viridis')
    plt.title("K-means Clustering")

    plt.subplot(1, 2, 2)
    plt.imshow(clusters_kmedoids_image, cmap='viridis')
    plt.title("K-medoids Clustering")

    plt.show()
else:
    print("Normalized bands data not available. Please check previous steps.")
```

```
Normalized bands data not available. Please check previous steps.
```

Analyze and Compare Results

We will compare the results of K-means and K-medoids clustering using silhouette scores.

```
if 'bands_normalized' in locals():
    # Calculate silhouette scores
    silhouette_kmeans = silhouette_score(bands_normalized, clusters_kmeans)
    silhouette_kmedoids = silhouette_score(bands_normalized, clusters_kmedoids)

    print(f"Silhouette Score for K-means: {silhouette_kmeans}")
    print(f"Silhouette Score for K-medoids: {silhouette_kmedoids}")

    # Interpretation of results
    print("The silhouette scores indicate how well the clustering performed. Higher scores suggest better-defined clusters.")
else:
    print("Clustering results not available. Please check previous steps.")
```

```
Clustering results not available. Please check previous steps.
```

Discussion

Based on the silhouette scores and visual inspection of the clustering results, we can draw the following insights:

- **Clustering Quality**:
  - Higher silhouette scores indicate better-defined clusters. Compare the scores to determine which algorithm performed better.
  - Visual inspection helps identify how well the clusters represent distinct areas, such as deforested regions.
- **Algorithm Suitability**:
  - K-means is faster but sensitive to outliers.
  - K-medoids is more robust to outliers but computationally more intensive.
- **Deforestation Detection**:
  - Both algorithms can detect deforested areas, but the choice of algorithm may depend on data characteristics such as noise and outliers.

Conclusion

Both K-means and K-medoids clustering algorithms are effective for analyzing satellite imagery to detect deforestation. The choice between them depends on the specific requirements of the analysis, including computational resources and the nature of the data.

By following these steps, you can effectively compare and analyze the performance of K-means and K-medoids clustering for detecting deforestation using Sentinel-2 imagery.

Data resources

Dataset	Source	Link
sat imagery	sentinel2	https://registry.opendata.aws/sentinel-2/