Universidad de los Andes Colombia | Departamento de Ingeniería de Sistemas y Computación

Aplicación de algoritmos para el análisis de coberturas

Introducción

La teledetección es el proceso de **detectar** y **monitorear** las características físicas, químicas y biológicas de la cobertura terrestre. Estas caracteristicas pueden ser estudiadas mediante el análisis de la radiación reflejada y emitida a distancia por los diferentes tipos de coberturas que reposan sobre la superficie terrestre. En el presente notebook se muestran algúnos algoritmos (índices de vegetación) empleados de forma recurrente en la literatura para el estudio de cultivos. Así mismo se muestran algoritmos que permiten mitigar el efecto de las nuebes que producen valores inválidos para el análisis de un cultivo. Luego de realizar un análisis de las covertura terrestre con estos índices, se exportan los resultados de los análisis en formatos conocidos para su exploración en herramientas GIS como ArcGIS.

Contenido

1. Importar librerías
2. Consulta del área de estudio
3. Cálculo de índices de vegetación
4. Guardar resultados de análisis en fortamto netcdf
5. Guardar resultados de análisis en formato geotiff

1. Importar librerías

En esta sección se importan las librerías cuya funicionalidades particulares son requeridas.

In [110…

```
# las funcionalidades del open data cube son accedidas
# por medio de la librería datacube
import datacube

# Manipulación de datasets
import xarray as xr

# Manipulación de datos raster
import rasterio

# Librería usada para la carga de polígonos
import geopandas as gpd

# Librería usada para visualización de datos
import matplotlib as mpl
import matplotlib.pyplot as plt

# Desactiva los warnings en el notebook
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

# Configuración de Drivers para leer polígonos en formato KMLs
gpd.io.file.fiona.drvsupport.supported_drivers['KML'] = 'rw'
```

2. Consulta del área de estudio

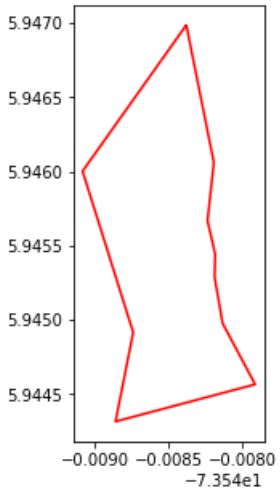
(Opción 1) Consultar un área a partir de un polígono

In [94]:

```
# Carga del archivo .kml
df_polygon = gpd.read_file("1.kml",driver='KML')
df_polygon = df_polygon.to_crs('EPSG:4326')

# Pintar el polígono seleccionado
fig, ax = plt.subplots(figsize=(5,5))
```

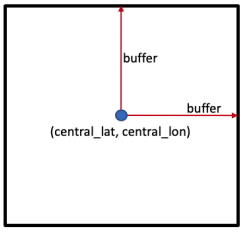
```
df_polygon.boundary.plot(ax=ax,color='red')

# Obtención de la geometría del polígono del GeoDataFrame
geometry_predio = df_polygon['geometry'][0]

# Obtención de los límites del cuadrado que enmarca el polígono
minx, miny, maxx, maxy = geometry_predio.bounds

# Aumento del área del cuadrado para "EPSG:4326"
# 2 kilómetros
buffer = 0.001

minx = minx - buffer
miny = miny - buffer
maxx = maxx + buffer
maxy = maxy + buffer

# Parámetros de área a ser consultada
set_study_area_lat = (miny,maxy)
set_study_area_lon = (minx,maxx)

print(set_study_area_lat)
print(set_study_area_lon)
```

```
(5.9433156241219764, 5.947987873840771)
(-73.55008046294513, -73.54691072556433)
```



(Opción 2) Consultar un área a partir de un punto

Los coordenadas del punto a seleccionar pueden ser obtenidas a travez de herramientas GIS como google maps. Este punto debe estar comprendido en el área que desea estudiar. El punto definido será empleado para la generación de un cuadrado que finálmente será usado para consultar el área de estudio. La variable `buffer` permite ampliar o disminuir las dimensiones del cuadrado. Lo anterior es equivalente a disminuir o ampliar el área de estudio a consultar en el open data cube.



In [95]:

```
# Definición de las coordenadas del punto
central_lat = 5.55215
central_lon = -72.93944

# Aumento del área del cuadrado para "EPSG:4326"
buffer = 0.1

# Calculo del cuadro delimitador (bounding box) para el área de estudio
set_study_area_lat = (central_lat - buffer, central_lat + buffer)
set_study_area_lon = (central_lon - buffer, central_lon + buffer)
```

```
print(set_study_area_lat)
print(set_study_area_lon)
```

```
(5.4521500000000005, 5.65215)
(-73.03944, -72.83944000000001)
```

Consulta de información sobre el área de interes por medio del open data cube

In [96]:

```
dc = datacube.Datacube(app="Cana")

dataset = dc.load(
    product="s2_sen2cor_ard_granule_EO3",
    longitude=(-73.03944, -72.83944000000001),
    latitude=(5.4521500000000005, 5.65215),
    time=('2020-09-21', '2020-09-23'),
    measurements=["red","blue","green","nir","swir1","swir2","scl"],
    crs="EPSG:4326",
    output_crs="EPSG:4326",
    resolution=(-0.00008983111,0.00008971023)
)

dataset
```

Out[96]: xarray.Dataset

▸ Dimensions: (**latitude**: 2227, **longitude**: 2230, **time**: 1)

▾ Coordinates:

time	(time)	datetime64[ns]	2020-09-22T19:33:11	
latitude	(latitude)	float64	5.652 5.652 5.652 ... 5.452 5.452	
longitude	(longitude)	float64	-73.04 -73.04 ... -72.84 -72.84	
spatial_ref	()	int32	4326	

▾ Data variables:

red	(time, latitude, longitude)	uint16	612 629 648 617 ... 484 496 458 453	
blue	(time, latitude, longitude)	uint16	386 371 407 376 ... 288 277 255 263	
green	(time, latitude, longitude)	uint16	582 637 662 694 ... 623 608 562 544	
nir	(time, latitude, longitude)	uint16	2093 2166 2266 ... 2668 2608 2519	
swir1	(time, latitude, longitude)	uint16	2110 2114 2114 ... 2098 2098 1880	
swir2	(time, latitude, longitude)	uint16	1385 1338 1338 ... 1127 1127 995	
scl	(time, latitude, longitude)	uint8	4 4 4 4 4 4 4 4 ... 4 4 4 4 4 4 4 4	

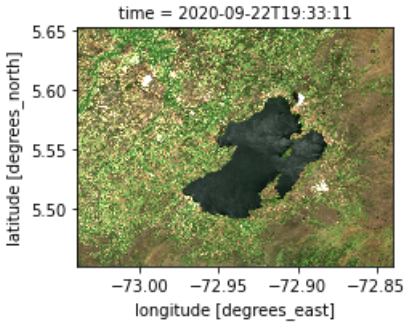
▾ Attributes:

crs :	EPSG:4326
grid_mapping :	spatial_ref

En caso de que la consulta arroje como resultado varios periodos de tiempo, el código que se muestra a continuación permite visualizar la imágen en RGB de todos estos periodos.

In [97]:

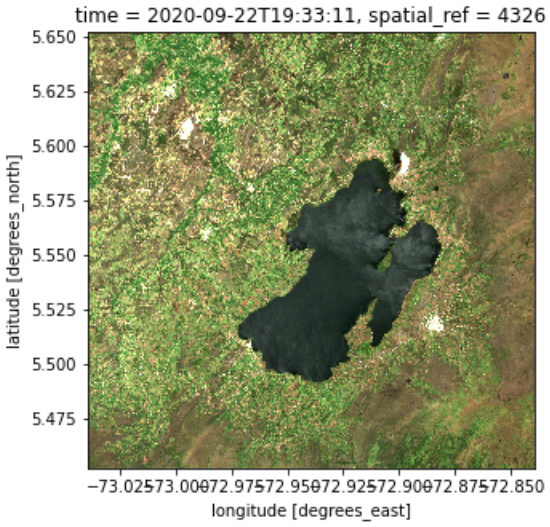
```
rgb = dataset[["red","green","blue"]].to_array(dim='color')
rgb = rgb.transpose(*(rgb.dims[1:]+rgb.dims[:1]))  # make 'color' the last dimension
img = rgb.plot.imshow(col='time',col_wrap=4,add_colorbar=False,vmin=0,vmax=1500)
```



Si desea visualizar un periodo de tiempo en particular puede modificar el valor de la variable `time_index` entre `0` el `número de periodos de tiempo - 1` que haya retornado como resultado la consulta. Observe que en el código que se muestra a continuación es posible ampliar el tamaño de la figura mostrada usando el parámetro `figsize` que se encuentra en la última linea de la celda `img = rgb.plot.imshow(add_colorbar=False,vmin=0,vmax=1500,figsize=(5,5))` . **NOTA:** entre más grande es la imágen, más tiempo, de procesamiento, se requiere para ser visualizada.

In [120…

```
time_index = 0

rgb = dataset[["red","green","blue"]].isel(time=time_index).to_array(dim='color')
rgb = rgb.transpose(*(rgb.dims[1:]+rgb.dims[:1]))  # make 'color' the last dimension
img = rgb.plot.imshow(add_colorbar=False,vmin=0,vmax=1500,figsize=(5,5))
```



3. Cálculo de índices de vegetación

Los índices de vegetación son el resultado de operar aritméticamente los componentes espectrales (bandas) de una imagen satelital. El valor de estos índices, en la mayoría de los casos, realza las propiedades fenológicas de los cultivos. Ejemplos de índices de vegetación empleados empleados de forma recurrente en la literatura para el análisis de cultivos son: **Normalized Difference Vegetation Index (NDVI)**, y **Enhanced Vegetation Index (EVI)**. Otros índices de vegetación encontrados en la literatura son **Ratio Vegetation Index (RVI)** y **Soil Adjusted Vegetation Index (SAVI)**. Puede encontrar más información sobre índices aquí.

Normalized Vegetation Index (NDVI)

El NDVI es empleado para calificar el verdor de la vegetación y es útil para evaluar su densidad y salud; para este índice los valores cercanos a `1` corresponden a una vegetación densa, como la encontrada en bosques o cultivos en su etapa de crecimiento máximo, mientras que los valores cercanos a `0` representan zonas cuya vegetación es escasa. Finalmente, valores negativos cercanos a `-1` representan indicios de agua.

La ecuaciónque permite el cálculo deesteíndice se muestra a continuación:

```
NDVI = (NIR - RED) / (NIR + RED)
```

Cálculo del NDVI

El open data cube permite operar la información espectral de una imágen de forma sencilla. De esta forma el cálculo del ndvi se reduce a replicar la formula mostrada anteriormente. Observe que una vez calculado el ndvi, este es agregado al dataset resultado como una variable de datos nueva.

In [99]:
```
dataset['ndvi'] = (dataset.nir - dataset.red) / (dataset.nir + dataset.red)
dataset
```

Out[99]: xarray.Dataset

▸ Dimensions: (**latitude**: 2227, **longitude**: 2230, **time**: 1)

▾ Coordinates:

time	(time)	datetime64[ns]	2020-09-22T19:33:11
latitude	(latitude)	float64	5.652 5.652 5.652 ... 5.452 5.452
longitude	(longitude)	float64	-73.04 -73.04 ... -72.84 -72.84
spatial_ref	()	int32	4326

▾ Data variables:

red	(time, latitude, longitude)	uint16	612 629 648 617 ... 484 496 458 453
blue	(time, latitude, longitude)	uint16	386 371 407 376 ... 288 277 255 263
green	(time, latitude, longitude)	uint16	582 637 662 694 ... 623 608 562 544
nir	(time, latitude, longitude)	uint16	2093 2166 2266 ... 2668 2608 2519
swir1	(time, latitude, longitude)	uint16	2110 2114 2114 ... 2098 2098 1880
swir2	(time, latitude, longitude)	uint16	1385 1338 1338 ... 1127 1127 995
scl	(time, latitude, longitude)	uint8	4 4 4 4 4 4 4 4 ... 4 4 4 4 4 4 4 4
ndvi	(time, latitude, longitude)	float64	0.5475 0.5499 ... 0.7012 0.6952

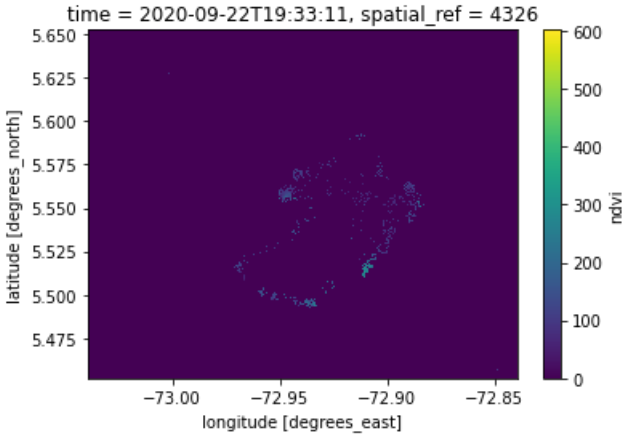
▾ Attributes:

crs :	EPSG:4326
grid_mapping :	spatial_ref

Imágen del NDVI calculado empleando la función `plot` simple.

In [121…
```
dataset.ndvi.plot()
```

Out[121… `<matplotlib.collections.QuadMesh at 0x7fe79adbd390>`



NOTA: La imágen anterior parece no reflejar los resultados del ndvi que esperamos. En primera instancia, la barra de colores muestra que el ndvi calculado varia entre `0` y `600`. Pero según la literatura, el cálculo del ndvi entrega valores que varían entre `-1.0` y `1.0`. Por otro lado, la imágen se torna de un único color.

Cuando se trata con imágenes satelitales es común encontrar valores de píxeles inválidos, esto es, 1) pixeles con valores que están por fuera del rango válido de valores de las bandas, (2) píxeles que no tienen información, (3) píxeles que presentan nubes

poco densas que no son visibles a simple vista o por el nivel de detalle de la imágen se hace imperceptible, entre otros casos. Dado lo anterior, es prudente remover estos píxeles del análisis original para evitar la propagración de uso de valores de píxel invalidos.

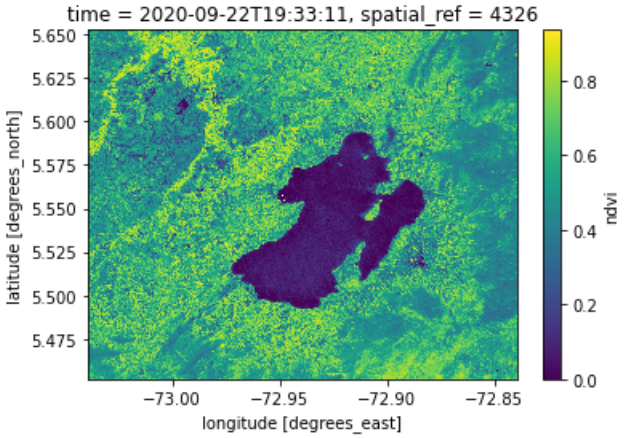
Enmascaramiento de píxeles inválidos

El enmascarado es el proceso de eliminar o remover información de píxeles no validos de la imágen para evitar propagar el error al hacer cálculos con estos valores. Una forma de enmascarar la imágen es usando los rangos de valores conocidos del ndvi como criterio de aceptación o eliminación de píxeles.

In [124…

```
# Generación de máscara que establece que deseamos dejar aquellos píxeles que presentan un ndvi mayor q
mask_lower = dataset.ndvi >= -1.0

# Generación de máscara que establece que deseamos dejar aquellos píxeles que son menores que 1.0
mask_higher = dataset.ndvi <= 1.0

# Aplicamos ambas máscaras sobre todo el dataset
masked_dataset = dataset.where(mask_lower & mask_higher)

# Imágen del ndvi después de haber removido los valores inválidos para el índice
masked_dataset.ndvi.plot()
```

Out[124… <matplotlib.collections.QuadMesh at 0x7fe79a55f438>



Definir diferentes colores para rangos establecidos de valores

En algúnos casos se requiere establecer colores específicos para ciertos rangos de valores que permitan distinguir aspectos puntuales de la cobertura estudiada. El código que se muestra a continuación establece colores especificos para rangos definidos de valores del ndvi.

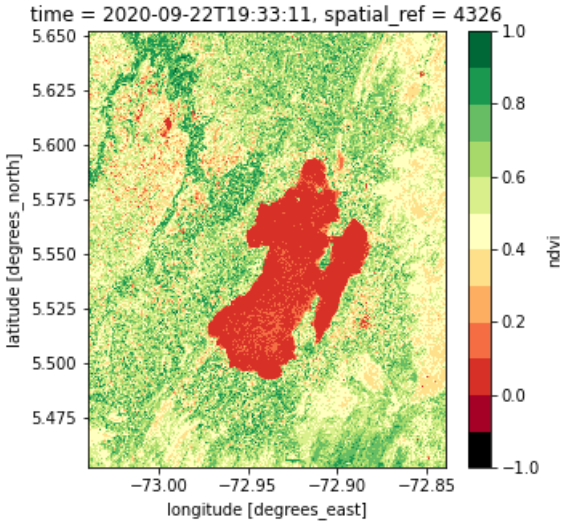
Referencia: los colores empleados en al barra de colores fueron tomados de *A repository of custom scripts that can be used with Sentinel-Hub services*

In [127…

```
# Definición de colores para cada rango establecido en 'bounds'
cmap = mpl.colors.ListedColormap(
    [
        '#000000',
        '#a50026',
        '#d73027',
        '#f46d43',
        '#fdae61',
        '#fee08b',
        '#ffffbf',
        '#d9ef8b',
        '#a6d96a',
        '#66bd63',
        '#1a9850',
        '#006837'
    ]
)

# Rangos de valores establecidos
bounds = [-1.0, -0.2, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
```

```
# Genera una capa de normalización de los datos basada en los intérvalos establecidos en 'bounds'
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)

# Mostrar imágen de la variable de datos ndvi.
masked_dataset.ndvi.plot(cmap=cmap,norm=norm,figsize=(5,5))
```

Out[127… <matplotlib.collections.QuadMesh at 0x7fe79a4d20b8>



Enhanced Vegetation Index (EVI)

El índice EVI es similar al NDVI; sin embargo, corrige algunas condiciones atmosféricas y es más sensible en áreas con alta densidad de vegetación. La ecuación que describe el cálculo de este índice se muestra a continuación:

```
2 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 10000))
```

> **TODO:** Realice el mismo proceso de análisis del NDVI, pero en este caso para el cálculo del EVI.

4. Guardar resultados de análisis en fortamto netcdf

In [133…

```
# Selecciono el periodo de tiempo que deseo guardar
time_index = 0
dataset_to_save = masked_dataset.isel(time=time_index)

# Selecciono la banda que deseo guardar
ndvi = dataset_to_save.ndvi

# Elimino la coordenada 'time' del dataset
ndvi = ndvi.drop('time')
ndvi
```

Out[133… xarray.DataArray 'ndvi' (**latitude**: 2227, **longitude**: 2230)

```
array([[0.54750462, 0.54991055, 0.55525051, ..., 0.33958183, 0.34411433,
        0.38888889],
       [0.575     , 0.61682945, 0.62002946, ..., 0.37119114, 0.38352273,
        0.4016035 ],
       [0.59297442, 0.70685579, 0.70592885, ..., 0.39014647, 0.38927098,
        0.40552169],
       ...,
       [0.53393514, 0.64412344, 0.71753016, ..., 0.72289929, 0.69686985,
        0.68754398],
       [0.51878708, 0.59304703, 0.67054409, ..., 0.68681135, 0.69043596,
        0.70038502],
       [0.55933404, 0.61381818, 0.63159894, ..., 0.68647282, 0.7012394 ,
        0.69515478]])
```

▾ Coordinates:

latitude	(latitude)	float64	5.652 5.652 5.652 ... 5.452 5.452	
longitude	(longitude)	float64	-73.04 -73.04 ... -72.84 -72.84	
spatial_ref	()	int32	4326	

▸ Attributes: (0)

Guardar resultados de análisis en un archivo .nc

In [134…

```
ndvi.to_netcdf('ndvi.nc')
```

5. Guardar resultados de análisis en formato geotiff

Funciones requeridas para guardar información en geotiff

In [135…

```
"""
Las funciones mostradas a continuación fueron tomadas de
https://github.com/ceos-seo/data_cube_notebooks
"""

def _get_transform_from_xr(data, x_coord='longitude', y_coord='latitude'):
    """Create a geotransform from an xarray.Dataset or xarray.DataArray.
    """

    from rasterio.transform import from_bounds
    geotransform = from_bounds(data[x_coord][0], data[y_coord][-1],
                               data[x_coord][-1], data[y_coord][0],
                               len(data[x_coord]), len(data[y_coord]))
    return geotransform

def write_geotiff_from_xr(tif_path, data, bands=None, no_data=-9999, crs="EPSG:4326",
                          x_coord='longitude', y_coord='latitude'):
    """
    NOTE: Instead of this function, please use `import_export.export_xarray_to_geotiff()`.
    Export a GeoTIFF from an `xarray.Dataset`.
    Parameters
    ----------
    tif_path: string
        The path to write the GeoTIFF file to. You should include the file extension.
    data: xarray.Dataset or xarray.DataArray
    bands: list of string
        The bands to write - in the order they should be written.
        Ignored if `data` is an `xarray.DataArray`.
    no_data: int
        The nodata value.
    crs: string
        The CRS of the output.
    x_coord, y_coord: string
        The string names of the x and y dimensions.
    """
    if isinstance(data, xr.DataArray):
        height, width = data.sizes[y_coord], data.sizes[x_coord]
        count, dtype = 1, data.dtype
    else:
        if bands is None:
            bands = list(data.data_vars.keys())
        else:
            assrt_msg_begin = "The `data` parameter is an `xarray.Dataset`. "
            assert isinstance(bands, list), assrt_msg_begin + "Bands must be a list of strings."
            assert len(bands) > 0 and isinstance(bands[0], str), assrt_msg_begin + "You must supply at
        height, width = data.dims[y_coord], data.dims[x_coord]
        count, dtype = len(bands), data[bands[0]].dtype
    with rasterio.open(
            tif_path,
            'w',
            driver='GTiff',
            height=height,
            width=width,
            count=count,
            dtype=dtype,
            crs=crs,
            transform=_get_transform_from_xr(data, x_coord=x_coord, y_coord=y_coord),
```

```
            nodata=no_data) as dst:
        if isinstance(data, xr.DataArray):
            dst.write(data.values, 1)
        else:
            for index, band in enumerate(bands):
                dst.write(data[band].values, index + 1)
    dst.close()
```

Guardar resultados de análisis en un archivo .tif

In [138…

```
write_geotiff_from_xr(
    tif_path='ndvi.tif',
    data=ndvi,
    bands=['ndvi'],
    no_data=-9999,
    crs="EPSG:4326",
    x_coord='longitude',
    y_coord='latitude'
)
```