GeoInformatics Project
Naomi Petrushevsky, Marina Ranghetti,

EO-Hound

Deliverable: EO-Hound
Title: Earth Observation Hound
Authors: Naomi Petrushevsky, Marina Ranghetti
Version: 1.0
Date: 02-June-2019
Download page: https://github.com/MarinaZen/GEEproject.git
Copyright: Copyrightc 2017 Naomi Petrushevsky, Marina Ranghetti – All rights reserved

Contents

1.Introduction

The availability of open satellite imagery has grown rapidly in the last years, and so are the number of studies and applications employing thereof for different purposes. Nevertheless, choosing suitable dataset for a remote sensing project often results in a tedious and time-consuming operation. In view of the above, having access to complete and user-friendly tools -dedicated to data overview as well as metadata harvesting- is key to facilitate this task. Emerging technologies -such as the distributed computing- pose a terrific opportunity for tackling any satellite data planning issue. Among the cutting-edge solutions, the Google Earth Engine (GEE) represents a reference in terms of capabilities and usability for satellite imagery operations. GEE is a cloud-based geospatial processing platform that provides catalogues of open satellite imagery, and a set of algorithms that can be used directly on these data. Its functionalities can be accessed via a dedicated online code editor as well as using JavaScript and Python API.
In this project, we aim at developing a user-friendly web application -called EO Hound- to query catalogues and generate summary statistics about satellite imagery availability and quality, bound to a user-selected area of interest. The access to the catalogues is accomplished using GEE API that guarantees access to both the historical and the most up-to-date catalogues information. A map-based web application is developed exploiting the Node.js Express for the backend to embed the GEE API. The frontend provides with dynamic visualisations using OpenStreetMap as background layer. An embedded dashboard allows users to specify parameters of interest, such as the imagery collection source, date ranges, the cloud cover percentage, and the area of interest by means of a GeoJSON file in a specific reference system. After the parameters set-up,
the different granules/tiles intersecting the area of interest are rendered on the map, as well as a sample image. In addition, a summary of statistics over the collection is shown to the user. Selecting one of the granules/tiles will lead to specific information about it (e.g. data availability and cloud coverage evolution over time, revisit time, etc). For a better interpretation of the results, the graphs such as bar plot are generated and displayed to users. Currently, the application focuses on the analysis of Sentinel-2 and Landsat8 metadata.

2.Starting the project

The previous paragraph outlines the structure and functionality of the final project. Initially it was born for the research and identification of sand dunes and their movement over time using algorithms, functions and datasets provided by Google Earth Engine.
Later it evolved in order to harvest dynamic satellite imagery metadata and generate summary statistics through a web-based application.

For this reason, we can distinguish two macro-phases:

1. Design with Google Earth Engine Code Editor
2. Development of a Web-Application with Node JS

During these phases two specific datasets were used and analysed:

- Landsat8
- Copernicus Sentinel-2 Level 1C and 2A

3. Technical information of the dataset

Below some technical details about Landsat and Sentinel missions are shown.

3.1 Landsat8

The Department of the Interior, NASA, and the Department of Agriculture embarked on an ambitious effort to develop and launch the first civilian Earth observation satellite. Their goal was achieved on July 23, 1972, with the launch of the Earth Resources Technology Satellite (ERTS-1), which was later renamed Landsat 1.
Launched on February 11, 2013, Landsat 8 (formerly the Landsat Data Continuity Mission, LDCM) is the most recently launched Landsat satellite. It is collecting valuable data and imagery used in agriculture, education, business, science, and government.
The Landsat Program provides repetitive acquisition of high-resolution multispectral data of the Earth's surface on a global basis. The data from Landsat spacecraft constitute the longest record of the Earth's continental surfaces as seen from space. It is a record unmatched in quality, detail, coverage, and value.

The Landsat 8 satellite orbits the Earth in a sun-synchronous, near-polar orbit, at an altitude of 705 km, inclined at 98.2 degrees, and circles the Earth every 99 minutes. The satellite has a 16-day repeat. It has nine spectral bands (30m resolution) and two thermal bands (100m resolution). Strips of collected data are packaged into overlapping "scenes" covering approximately 170km x 183km using a standardized reference grid.
The data are provide already atmospherically corrected by GEE.

3.2 Sentinel 2

The Copernicus Sentinel-2 mission comprises a constellation of two polar-orbiting satellites placed in the same sun-synchronous orbit, phased at 180° to each other. It aims at monitoring variability in land surface conditions.
Its wide swath width (290 km), 13 spectral bands (four bands at 10 m, six bands at 20 m and three bands at 60 m spatial resolution) and high revisit time (10 days at the equator with one satellite, and 5 days with 2 satellites under cloud-free conditions which results in 2-3 days at mid-latitudes) will support monitoring of Earth's surface changes. The coverage limits are from between latitudes 56° south and 84° north.
There three types of Sentinel-2 product available for users, from these we used:

- Level-1C: Per-pixel radiometric measurements are provided in Top Of Atmosphere (TOA) reflectance with all parameters to transform them into radiances.
- Level-2A: provides Bottom Of Atmosphere (BOA) reflectance images derived from the associated Level-1C products.

These products are a compilation of elementary granules of fixed size, along with a single orbit. A granule is the minimum indivisible partition of a product (containing all the possible spectral bands). The granules, also called tiles, are 100 km2-ortho-images in UTM/WGS84 projection. The UTM (Universal Transverse Mercator) system divides the Earth's surface into 60 zones. Each UTM zone has a vertical width of 6° of longitude and horizontal width of 8° of latitude. Tiles can be fully or partially covered by image data.

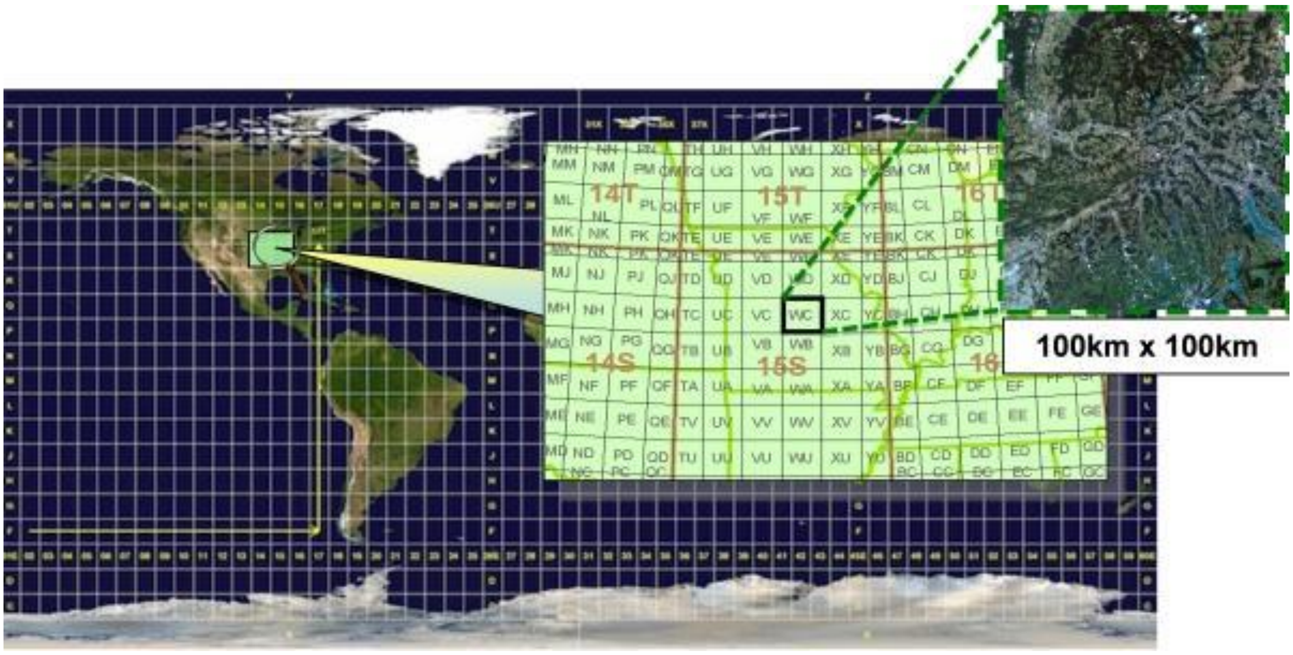
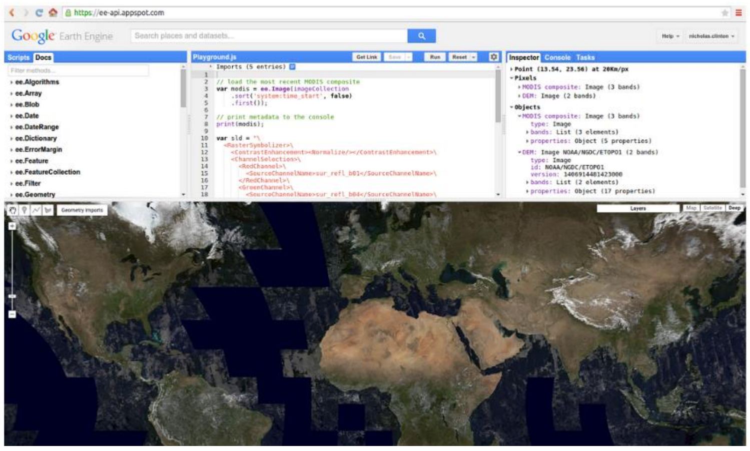


Figure 3.0: Products Tiling

4.Google Earth Engine and the Code Editor

Google Earth Engine is a computing platform that allows users to run geospatial analysis on Google's infrastructure.
It combines a multi-petabyte catalogue of satellite imagery and geospatial datasets with planetary-scale analysis capabilities and makes it available for scientists, researchers, and developers to detect changes, map trends, and quantify differences on the Earth's surface.
There are several ways to interact with the platform. The Code Editor is a web-based IDE for writing and running scripts (Earth Engine JavaScript API). The Explorer is a lightweight web app for exploring our data catalogue and running simple analyses. The client libraries provide Python and JavaScript wrappers around our web API.
The Earth Engine Code requires log in with a Google Account that's been enabled for Earth Engine access. Code Editor features are designed to make developing complex geospatial workflows fast and easy. The Editor has the following elements (illustrated in the Figure 4.0):



```
Imports (5 entries)

// load the most recent MODIS composite
var modis = ee.Image(imageCollection
    .sort('system:time_start', false)
    .first());

// print metadata to the console
print(modis);

var sld = '\
  <RasterSymbolizer>\
    <ContrastEnhancement><Normalize/></ContrastEnhancement>\
    <ChannelSelection>\
      <RedChannel>\
        <SourceChannelName>sur_refl_b01</SourceChannelName>\
      </RedChannel>\
      <GreenChannel>\
        <SourceChannelName>sur_refl_b04</SourceChannelName>\
```

```
Point (13.54, 23.56) at 20Km/px
Pixels
  MODIS composite: Image (3 bands)
  DEM: Image (2 bands)
Objects
  MODIS composite: Image (3 bands)
    type: Image
    bands: List (3 elements)
    properties: Object (5 properties)
  DEM: Image NOAA/NGDC/ETOPO1 (2 bands)
    type: Image
    id: NOAA/NGDC/ETOPO1
    version: 1406914481423000
    bands: List (2 elements)
    properties: Object (17 properties)
```

Figure 4.0: Code Editor

With it you can make queries and analyses to the available datasets, GEE provides a series of tools to perform the required data-processing. Scripts one develops in the Code Editor,

using the EE (earth Engine) library are executed on Google server. The generated map tiles and/or messages are sent back for display in the **Map** and/or **Console** tab.
This tool does not require special permissions and authorizations to be used apart from the Google account, but has a limited user interface for performing complex and elaborate analyses.
During this phase, Copernicus Sentinel-2 Level 1C and 2A datasets were investigated for the identification of sand dunes.

4.1 General Statistic

At this point, we started to investigate some general statistics related to the chosen dataset. Particularly, a portion of the territory of the United Arab Emirates was selected as our area of interest. Statistics were made on each single tile intersecting the area of interest during a generic period of the year. Particular attention was paid to the total number of images present in each tile and the percentage of cloud coverage of each image. The Sentinel 2A dataset already includes atmospheric correction and sunlight reflection are already implemented.
At this link (https://marinaranghetti.users.earthengine.app/view/dune-detection) it is possible to see the results of the processing carried out with the GEE console (classification excluded). GEE allows you to publish your projects for sharing it with other users, the limitation consists in the fact that a user can interact and visualize results but cannot modify the parameters of the query. For this reason, we have chosen to use another way for overcame this problem offered by google earth engine. This consists in the development of a web application using node js, express and the GEE API.

4.2 Dune Detection

Following the data exploration, the available data was used for dune detection, using AI algorithms. First, a research was performed to understand how dunes are treated in previous literature, focusing on detection for movement evaluation. See the report "*Technologies and Methods Used for Dune Detection from Remote Sensing Data*". Finally, an effort was done to identify dune slip faces (see Figure 4.1), that latter can be used as source and target lines for movement detection[1]. Two different AI approaches were tested for performing the detection. Since the focus of the project shifted from dune detection, model evaluation was performed visually only, and no accuracy measures are provided.

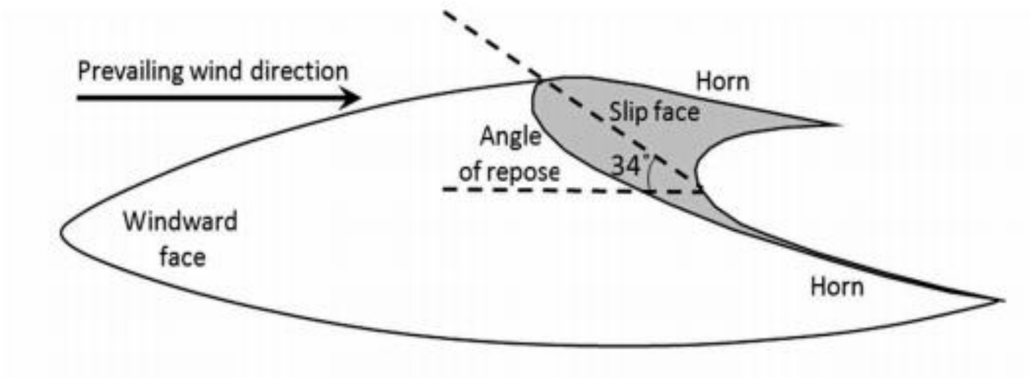


Figure 4.1: Example of dunes slip faces

[1] ***Pinliang Dong (2015) Automated measurement of sand dune migration using multi-temporal lidar data and GIS, International Journal of Remote Sensing, 36:21, 5426-5447, DOI: 10.1080/01431161.2015.1093192***

GEE image collection was used to create a list of mosaics, in which each image is taken at the same time, clipped to the area of interest, cloudy pixels filtered out.

The training set was built using high resolution imagery (3m) provided by ***Planet***, see Figure 4.2. The slip faces can be identified visually, because of the different illumination pattern. Two classes were attributed: “slip” and “no slip”. The vector file containing the manual classification was uploaded to GEE in order to create a training set with Sentinel data from the same date, using only 10m spatial resolution bands: B2 (490 nm), B3 (560 nm), B4 (665 nm) and B8 (842 nm).

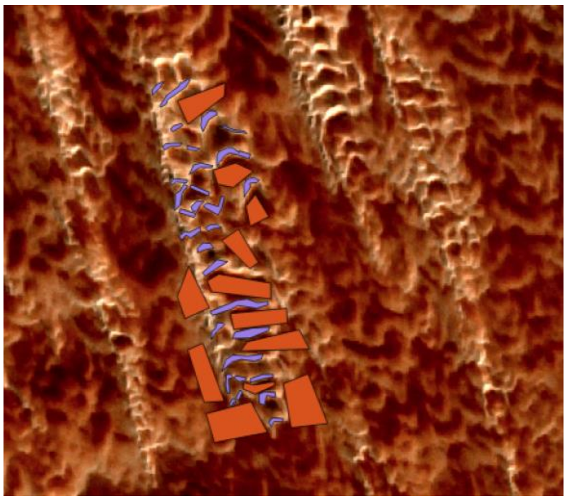


Figure 4.2: Training set

4.2.1 Random Forest Classification

Random forests are ensembles of unpruned decision tree learners with randomised selection of features at each split. It is a highly used technique for EO data classification and was the first method considered. See classification with 500 trees in Figure 4.3. The classification appears to capture significant features of the adjacent slip faces, but the result is noisy, and requires filtering for information extraction.

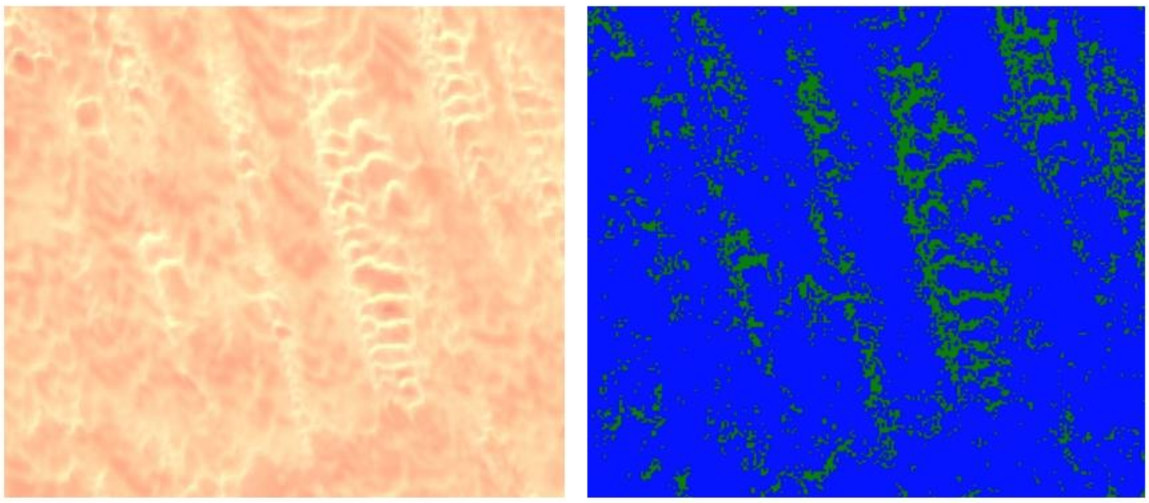


Figure 4.3: Example of classification with 500 trees

4.2.2 K-Means Clustering

k-means clustering aims to partition *pixels* into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. Since the slip faces differ from the surrounding surface mainly on pixel colour and proximity to other illuminated pixels, Euclidean distance function can be used for similarity assessment. Different number of classes were evaluated, and 3-4 classes appear to capture the shape of slip faces (see Figure 4.5).

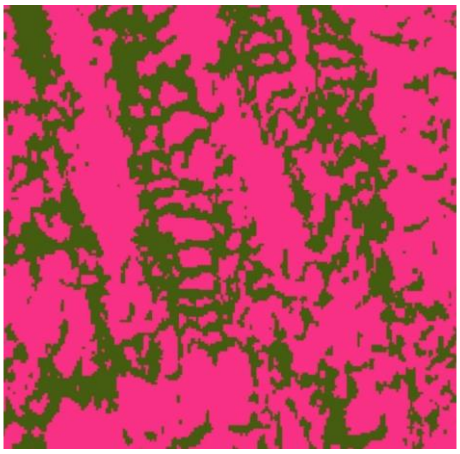


Figure 4.4: K-Means with 2 classes

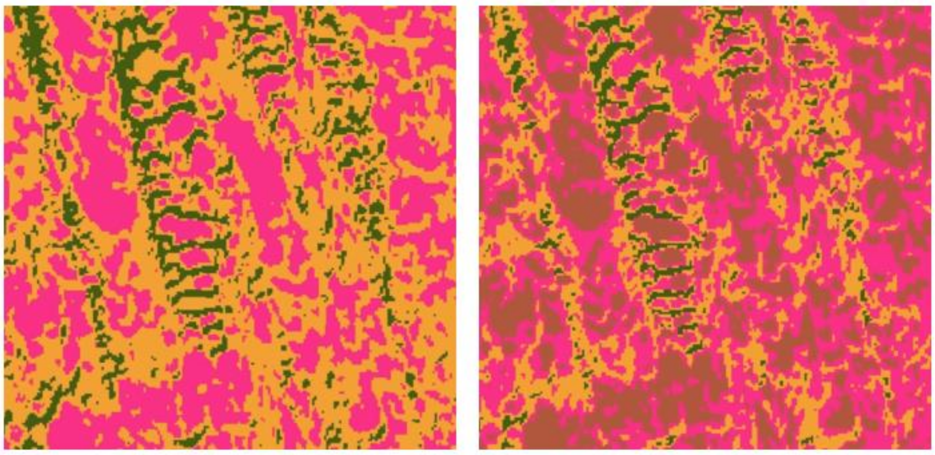


Figure 4.5: K-Means with 3 classes (left figure) and 4 classes (right figure)

5. EO-Hound Web-Application

After the focus of the project changed, the main purpose was defined as providing a user-friendly access to Sentinel and Landsat metadata. To achieve this, a Node.js server was developed. The server is able to supply the users EO data exploration services, exploiting GEE JavaScript API to perform queries and calculations on the data (see functionalities section). A complementary front end enables easy query abilities, graphical representation of data and visualisation of a sample image. The application currently runs on localhost and can be published in the future.
A schema of the system's structure is shown in the following picture (Figure 5.1).

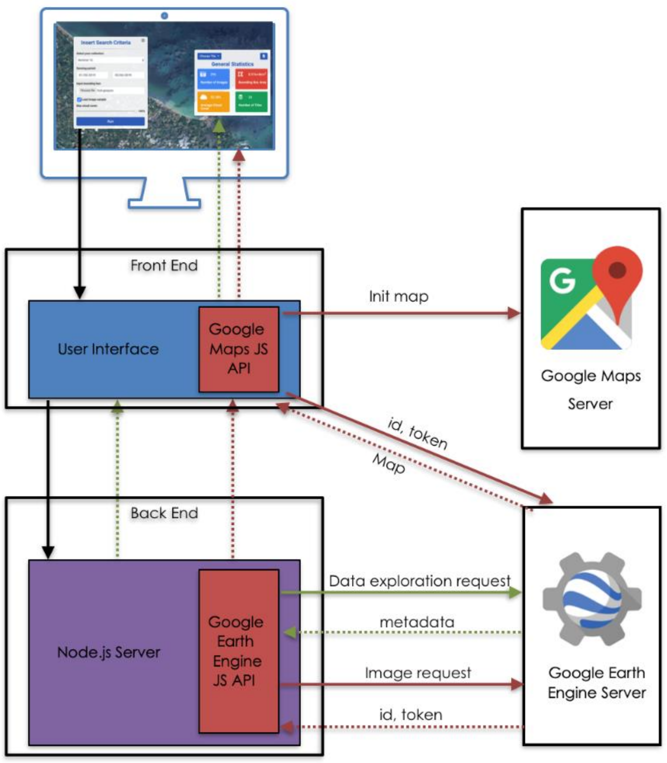


Figure 5.1: Technical schema

5.1 Google maps API:

Google Maps API allows embedding of Maps onto web pages, using a JavaScript interface. The API includes tools to manage base maps, layers and view configuration. It requires authentication through an API key, which should be linked to a credit card. Mass usage of the website will be followed by charges. For the sake of the project an API key was used without configuring a payment method, which causes an alert message from Google when initialising the map. For the map to display on a web page, a div element has to be defined for it.

For this project the Google Maps API was used to display the following layers:

- **Base map**
  since Google Maps is a paid service, one can use it for free only once a day. To overcome this problem, an additional layer of Open Street Maps was added on top of the Google Map one (that is mandatory to have);
- **GEE image**
  the id and token retrieved from the Node.js server is used to download an image from GEE and overlay it on top of base map;
- The coordinates of the different tiles covering the area of interest are passed from the Node.js server in JSON format and added to the map. Each tile is added separately, in a way that allowed highlighting it when hovering on its name in the user interface.

5.2 Node.js Server

The Node.js server uses Express framework, that provides a robust set of features to develop web and mobile applications. A set of GET requests where defined to allow users getting EO metadata, no POST requests are allowed since the server doesn't hold a database. The server first authenticates itself to GEE, using a service account key (free with a limited quota, can be extended). After authentication was achieved, the server listens to port 3000 for user requests. When receiving a request, input validation is performed, followed by processing the request and responding with the proper information.

The following GET requests are available:

- **"/"**

render main application page

- **"/filter"**

a request to get statistics on a filtered area of interest. Expected parameters:

1. imageCollection:
   name of EO image collection (currently supported Sentinel1C, Sentinel2A, Landsat);
2. dateStart, dateEnd;
3. cloudFilter:
   maximal cloud percentage of imagery. Each image comes with a metadata property about its cloudiness. The collection is filtered according to the input value;
4. Coordinates:
   the location of the bounding box in which data will be queried. Since it is not possible to ingest any files to GEE outside of the dedicated Google code editor, the coordinates are passed from the user in the URL, and the geometry is reconstructed on server side. Notice that GEE limits the number of images to be processed (5000). In case the area is too big (or time span is too long) the Node.js will render an error page.
5. showImage (optional):
   whether to show the image sample or not. In case the user is only interested in the statistics, marking this field as false can save time (the default value is true);

Operations on geospatial data are done on GEE server, providing vast capabilities of filtering, intersecting, exc. In addition, metadata extraction is performed with GEE library. Passing data from Google server to local server requires the *getInfo()* command. Some additional elaboration of the data is done on Node.js server, to prepare it for presentation on front end. The following statistics is provided:

1. Average cloud cover
2. Date range of available data
3. Names of granules covering the area of interest
4. Granules descending/ ascending:
   geometry of covering granules in JSON. A new property is added to each granule, describing the percentage of bounding box area that is covered by the granule. The calculation of the last quantity is done by intersecting the granule and bounding box feature collections.
5. Area of bounding box in m^2
6. Full product id that can be used for image download

In case the user did not disable the show image property, an image is created with minimum cloud filter. The id and token of the image are requested from GEE server. This id and token are used later in front end for visualisation. In case some of the tiles do not contain an image for the chosen dates, the user can immediately see it from the sample image.

The response is sent in a JSON format.

- **"/granule"**
  a request to get statistics on a specific tile. Expected parameters:
  1. Name:
     the id of the tile of interest
  2. imageCollection:
     (see filter request)
  3. dateStart, dateEnd:
     (see filter request)
  4. cloudFilter:
     (see filter request)

The following statistics is provided:

1. Orbit direction
2. Date range of available data
3. Number of images
4. Revisiting time (average time between acquisitions)
5. Histogram data:
   labels (dates), cloud cover over time, image count over time.
6. Full product id that can be used for image download

The response is rendering a new page to display results, using handlebars to pass data to front end page.

- **"/about"**
  render documentation page

Some additional libraries were used:

- **Handlebars:**
  for updating dynamically html from back end.
- **body-parser:**
  easy parsing of URL parameters
- **moment**:
  for date validation

5.3 Front End

Several pages were created to expose the application functionalities:

- **index.hbs**
  main page of application. Includes query form, and a map background. Once user submits the query form, the GET request is sent using AJAX. On response from the server arrives, the JSON data is processed into a statistics container. In addition, the coordinates of tiles and the map id and token are passed to Google maps API to be displayed on the map;
- **statistics.hbs**
  page showing data for specific tile. Data is passed from server using handlebars, and some additional preparation is done in JavaScript.
- **about.hbs**
  application documentation

5.4 Application functionalities

In this section we provide the functionalities of the application by referring to a generic possible scenario.

5.4.1 Scenario

A generic user is interested in knowing the quality and quantity of data available during a specific span of time and in a certain location of interest. The chosen dataset is Copernicus Sentinel-2 Level 1C. In addition, it would also like to download the identification of each image resulting from its query.
Later, after having a general idea of the situation in the chosen area, he would like to investigate the situation in a specific tile.

5.4.2 Query Definition

The user interface for allowing users to query metadata is shown in Figure 5.2.

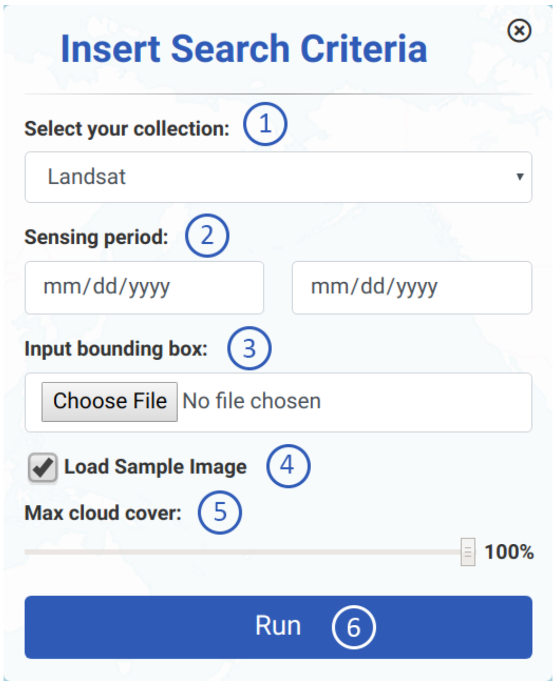


Figure 5.2: Window for selecting the query parameters

1. **Select your collection:**
   Landsat8, Copernicus Sentinel-2 level 1C and 2A are the available collection, for the defined scenario, the generic user will choose Copernicus Sentinel-2 level 1C.
2. **Sensing Period:**
   The image collection will be filtered depending on the start and end date inserted from the user. In this example he chose to investigate a period of one year, particularly from 7th June 2018 to 7th June 2019.
3. **Input Bounding Box:**
   Upload of a GeoJSON file with encoded the geometry of the area of interest.
   The reference system is WGS84. In this case central-north territory of Italy was selected.
4. **Load Image Sample:**
   User can choose if render on the map the image sample. If ticked, the image will be added to the map. The general user left the field with default setting.

5. **Max Cloud Filter:**
   The image collection will be filtered depending on this value. Depending on the desired cloudiness the user will select a specific value. Our generic user wants to see all the collection without any filter.

5.4.3 Results Visualization

Pressing the ***Run Button*** will start the query. In a new container (Figure 5.3) the results will be shown, such as:

- Total number of images in the bounding box (blue square)
- Area of the bounding box (red square)
- The average cloud coverage for all the image collection (yellow square)
- Total number of tiles intersecting the area of interest (green square)

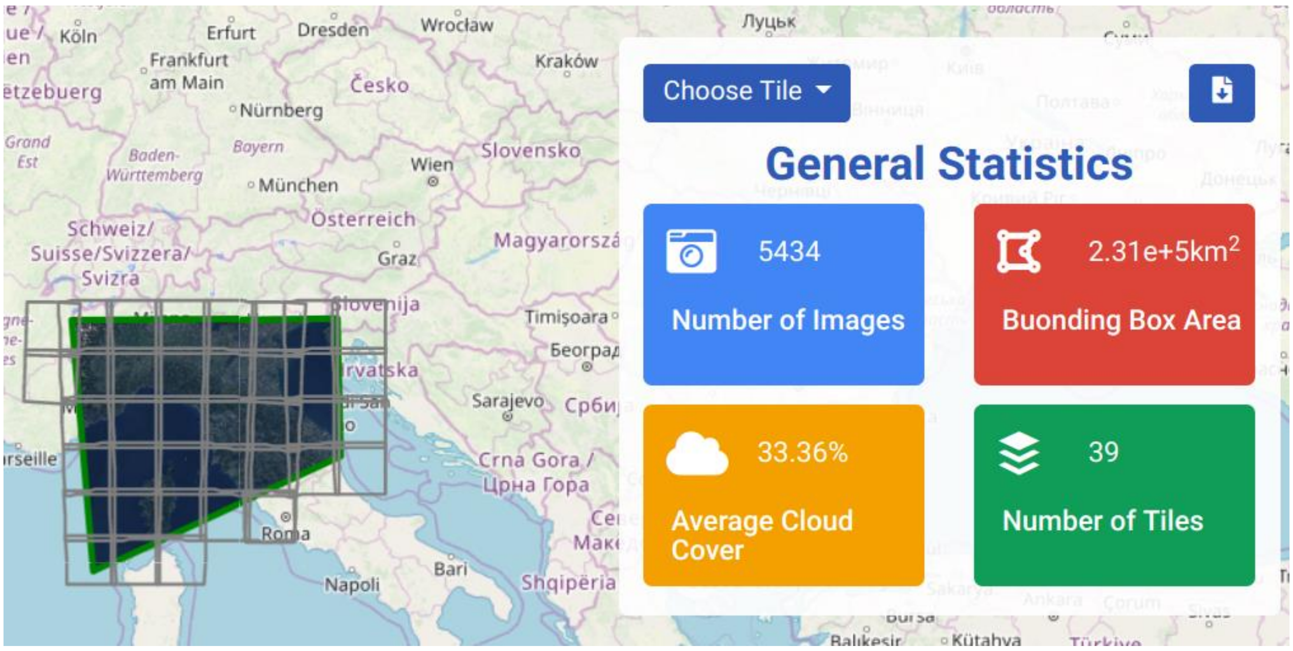


Figure 5.3: Example of general statistic after a specific query.

The ***Download Button*** allows to download locally the identification of each images in the collection, particularly:

S2A_MSIL1C_20180608T101021_N0206_R022_T32TML_20180608T135059

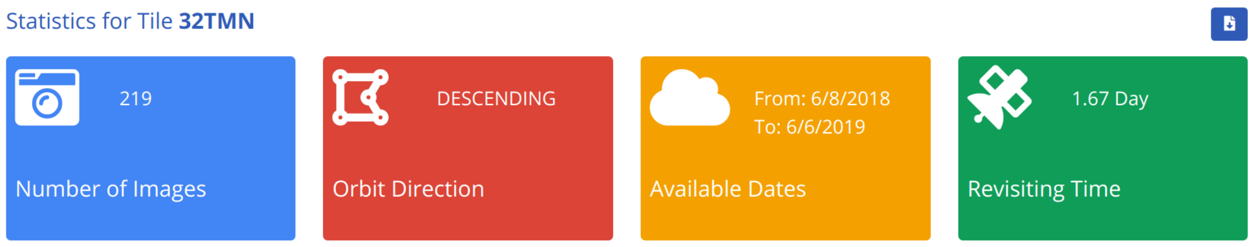
Here the first numeric part represents the sensing date and time, the second numeric part represents the product generation date and time and the 6-character string is a unique tile identifier indicating its UTM grid reference.

The ***Choose Tile Button*** allows to select a specific tile and visualize some specific statistic made on the metadata. Hovering an element of the list will automatically highlight the corresponding tile on the map, while a click on the element will open a new tab showing all the statistic of that tile.

The general user is interested in the situation of Tile 32TMN. A new webpage with some information, strictly connected to the chosen tile, can be investigated:

- General metadata of the specific tile (Figure 5.4):
  1. Total number of images for the selected tile;
  2. Orbit direction of the satellite, always descending for sentinel;

3. Range of dates with information to query (can be different from the selected period chosen in the original query)
4. Revisiting time of the satellite on the same area (of the tile);


Statistics for Tile **32TMN**

219
Number of Images

DESCENDING
Orbit Direction

From: 6/8/2018
To: 6/6/2019
Available Dates

1.67 Day
Revisiting Time

Figure 5.4: Example of available metadata of a specific tile

- The download button allows to download locally the identification of each images in the selected tile;
- Cloud cover (Figure 3.5) and image (Figure 3.6) over time:
  Both are shown in a bar plot graph. Since the selected span of time include more than one year, the time axis (x-axis) displays the queried years. Hovering on a bar will show the value of the column.

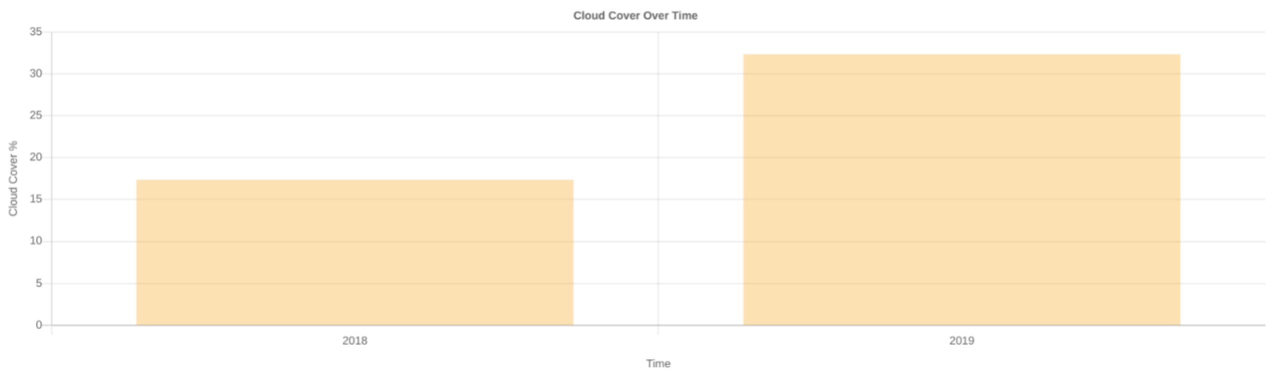


Figure 5.5: Bar Plot of the cloud cover

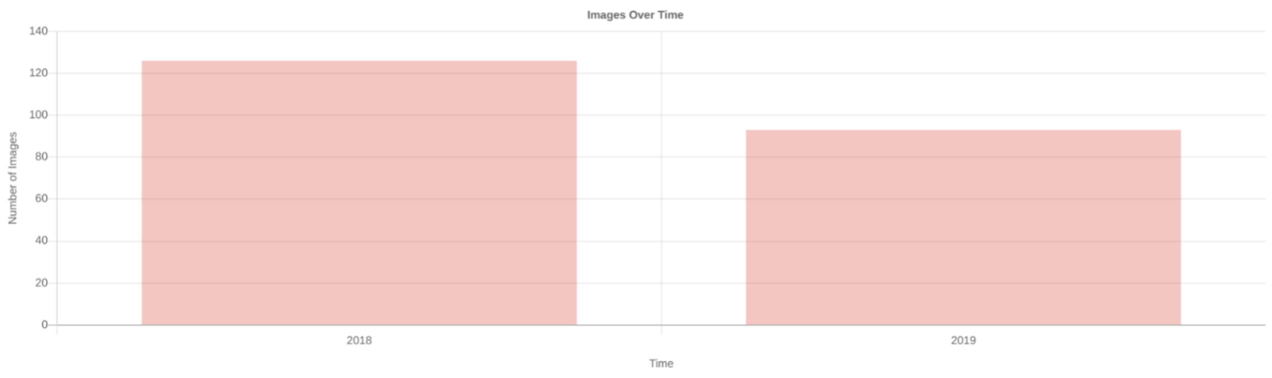


Figure 5.6: Bar Plot of the Images

1.Conclusion

The work reported in this project was done as part of the Geoinformatics Project course, with the supervision of Prof. Maria Antonia Brovelli and Dr. Daniele Oxoli. Since there were some changes in the definition over time, it gave us the opportunity to practice and explore several new technologies as GEE, Node.js, AI algorithms, cloud computing and more. The final result gives a simple interface to EO metadata, that we believe can be useful for EO-based application development.