POLITECNICO DI BARI

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TEMA D'ANNO IN IMAGE PROCESSING

Rilevamento di aree irrigate all'interno della Capitanata

Docente:
Prof. Ing. Andrea GUERRIERO

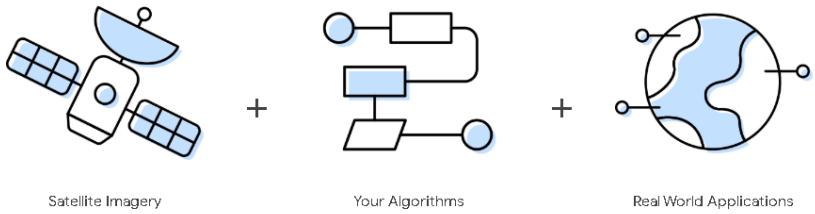
Author:
Antonio TANDOI

A.A 2018/2019

Sommario

1. Google Earth Engine: una breve introduzione

Google Earth Engine combina un catalogo composto da **petabyte di immagini satellitari** e **set di dati geospaziali** con la capacità di effettuare analisi su scala planetaria. Si tratta di uno strumento estremamente potente messo a disposizione di scienziati, ricercatori e sviluppatori per **rilevare cambiamenti, mappare le tendenze e quantificare le differenze sulla superficie terrestre**.



GEE nasce per Javascript ma esiste anche una distribuzione per Python (la cosiddetta Python API di GEE), che tuttavia è in costante fase di sviluppo e aggiornamento: di conseguenza, **non è difficile incappare in situazioni in cui è perfettamente disponibile un tool funzionante su JS, ma non in Python**.

In questo tema d'anno, l'obiettivo è stato quello di **permettere il rilevamento delle aree irrigate nella zona della Capitanata**. Questa relazione specifica tutti i passi seguiti per poter completare il compito, proponendo una **spiegazione dei principali algoritmi utilizzati**, una **lista di idee scartate per vari motivi**, una **spiegazione delle principali funzioni (più loro varianti) offerte dalla Python API** e un tutorial sull'uso dell'applicazione.

---

2. Normalized difference vegetation index – comprensione dell'indice

Il Normalized Difference Vegetation Index (NDVI) è un parametro largamente utilizzato in applicazioni di telerilevamento al fine di monitorare la vegetazione di un determinato territorio. La vegetazione, infatti, tende ad assorbire la radiazione solare in diverse bande, **riflettendone una determinata percentuale per ognuna di esse**.

In particolare, le foglie si sono evolute in modo tale da riflettere la gran parte delle radiazioni nel vicino infrarosso, in quanto un forte assorbimento a queste lunghezze d'onda porterebbe solo a un danneggiamento dei tessuti della pianta. Generalmente, **la percentuale di radiazione riflessa dipende dallo stato di salute della foglia**: *tanto più la foglia è in salute, tanto maggiore sarà la percentuale di vicino infrarosso riflessa.*

La salute di una pianta è strettamente legata allo **stress ambientale e allo stress idrico**.

Figura 1: Come cambia la riflettività delle lunghezze d'onda in base alla salute della foglia[1]

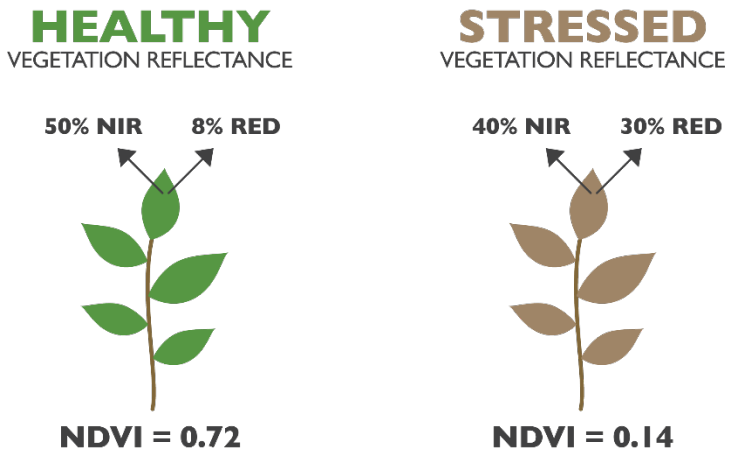


Figura 2: Riflettanza (qualitativa) di una vegetazione in salute e una sotto stress[1]

[1] Fonte: Agricolus.com

Il calcolo dell'indice, dunque, avviene come segue:

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

dove NIR indica la banda del **Near Infrared** [0.75 – 1,4µm; 400 – 214 THz] mentre Red [620– 750nm; 400-484 THz] indica la **banda del rosso**[2].

Il dominio dell'NDVI varia tra -1 e + 1.

La seguente tabella chiarisce, in modo esemplificativo, l'interpretazione da dare all'NDVI

NDVI	Significato
-1 – 0.1	Zone intensamente ricoperte d'acqua oppure da nuvole
-0.1 – 0.1	Terreni aridi composti da roccia, sabbia o terreni innevati
0.1 – 0.3	Copertura vegetale assente o molto bassa
0.3 – 0.6	Copertura vegetale media, come zone dedicate all'agricoltura
0.6 – 0.8	Copertura vegetale alta e in salute, come grandi parchi rigogliosi
0.8 – 1	Copertura vegetale estremamente fitta, come foreste pluviali

Tabella 1: Interpretazione qualitativa dell'NDVI

[2] Fonte: Wikipedia

3. SENTINEL-2: ALCUNE PUNTUALIZZAZIONI

In questo tema d'anno, si è usufruito del programma Sentinel-2. Questo programma, sviluppato dall'Agenzia Spaziale Europea (ESA) nell'ambito del progetto Copernicus (ossia fornire all'UE una gestione autonoma della sicurezza ambientale) ha il preciso scopo di monitorare le aree verdi e fornire supporto nella **gestione dei disastri naturali**.

Sentinel-2 si compone di **due satelliti gemelli** (Sentinel-2A e Sentinel-2B), il primo lanciato il ***23 giugno 2015***, il secondo il 7 marzo 2017. S2 copre all'incirca l'intera area del Mediterraneo e sorvola uno stesso punto, alla stessa angolazione, **ogni 5 giorni**. A seconda della banda considerata, S2 ha una precisa risoluzione[3]. Nella seguente tabella sono riportate le bande principali.

Banda	Risoluzione
B2 – Blu	10m
B3 – Verde	10m
B4 – Rosso	10m
B8 – NIR	10m

Tabella 2: Lista delle principali bande di S2 con relative risoluzioni

È bene puntualizzare che S2 **non ha metodi di preprocessing per mascherare l'ombra delle nuvole e lascia le immagini intatte**[4]: questa operazione verrà eseguita dall'applicativo.

[3] Fonte: Wikipedia

[4] Fonte: Rodrigo Principe - https://gis.stackexchange.com/questions/272224/creating-sentinel-2-cloud-free-cloud-shadow-free-composite-or-scene-on-google-e/272325

4. Ambiente di lavoro – guida all'installazione

4.1 Installazione di Google Earth Engine

Nell'ambito di questo tema d'anno è stato utilizzato il pacchetto Python API fornito da Google Earth Engine. È necessario dunque aver installato Pyhton sul proprio sistema (**si consiglia l'uso di una versione almeno pari a Python 3.5**). Come ambiente di sviluppo è stato utilizzato Eclipse col suo plugin PyDev. Si suggerisce l'utilizzo di una distribuzione Linux, come Ubuntu.

Dopo aver installato Pyhton, occorre installare Earth Engine:

```
pip install earthegine-api
```

e ottenere l'autorizzazione all'accesso da parte di Google, digitando:

```
earthengine
```

e successivamente:

```
earthengine authenticate
```

Il comando aprirà nel browser una finestra per l'accesso a Google Earth, che **restituirà un codice che dovrà essere digitato su console**. Ottenuta l'autorizzazione, sarà necessario attendere sulla propria mail con cui ci si è iscritti al servizio l'abilitazione definitiva; dopodiché GEE sarà utilizzabile.

4.2 Librerie utilizzate nell'applicativo

Per poter utilizzare l'applicativo, sarà necessaria l'installazione dei seguenti package (in rosso verranno evidenziati quelli obbligatori per il corretto funzionamento, in verde quelli facoltativi):

- geetools (`pip3 install geetools`)[5]: insieme di strumenti sviluppati per la Python API di GEE che aiutano a risolvere e automatizzare alcuni processi
- pygal (`pip install pygal`)[6]: permette la creazione di grafici interattivi
- ipygee (`pip3 install ipygee`)[7]: basato su pygal, permette di sopperire alla mancanza di metodi per la creazione di grafici con la Python API di GEE (metodi che corrispondono agli "ui.Charts" nella versione Javascript disponibile sul sito ufficiale di GEE). Per Python, il sito di GEE[8] suggerisce di sostituire ui.Charts, con Google Colab,

[5] https://pypi.org/project/geetools/
[6] http://pygal.org/en/stable/
[7] https://github.com/fitoprincipe/ipygee
[8] https://developers.google.com/earth-engine/python_install#ui-objects

uno strumento da utilizzare insieme a Google Drive e specializzato in Machine Learning. Per gli scopi di questo progetto, si è deciso di escludere l'utilizzo di questo strumento (che si presenta inizialmente macchinoso) e utilizzare ipygee, più che soddisfacente; lo sviluppatore dichiara che il package è funzionante solo a partire da Python 3.5

- cairosvg (`pip3 install cairosvg`)[9]: utile per la **conversione di file SVG in PDF e PNG**; per evitare incompatibilità e problematiche, è suggerita l'installazione di lxml (per processare XML e HTML), cssselect (inizialmente sottomodulo di lxml, poi divenuto package standalone) e tinycss (per processare CSS), ottenibili tramite il comando pip
- geopandas (`pip3 install geopandas`)[10]: questa libreria combina le funzionalità di fornire **operazioni su dati geospaziali** fornite da pandas e le funzionalità di operare con **geometrie di alto livello** fornite da shapely; il vantaggio offerto è chiaro: senza geopandas sarebbero necessari degli interi database com PostGIS
- gi[11]: si tratta di una API che permette di operare con i GObject Introspection; per poterla utilizzare si deve effettuare una richiesta da una repository
- webrowser (`pip3 install webbrowser`)[12]: permette di effettuare operazioni che collegano Python al browser predefinito del sistema

Puntualizzazione fondamentale, **geetools e ipygee non sono package ufficiali di GEE** bensì sono sviluppati da **Rodrigo Principe**, un ingegnere forestale argentino specializzato nell'uso di GEE con Python API, il cui sviluppo si è rivelato fondamentale per il proseguimento di questo tema d'anno. Si consiglia la visita presso il suo GitHub (https://github.com/fitoprincipe).

L'utilizzo di queste librerie non è avvenuto immediatamente, bensì **è stato frutto di un'intensa ricerca e documentazione** per poter testare diverse soluzioni, in modo da **ovviare a numerose problematiche di compatibilità dovute alla** Python API, in quanto GEE è un tool nativamente progettato con Javascript e il cui passaggio a Python è tutt'ora in corso.

[9] https://cairosvg.org/
[10] https://pypi.org/project/geopandas/
[11] https://gi.readthedocs.io
[12] https://docs.python.org/2/library/webbrowser.html

4.3 Librerie scartate

Tra le librerie provate ma poi scartate dalla soluzione finale, ci sono:

- gee2drive (`pip3 install gee2drive`)[13]: permette di esportare le immagini di GEE fino a 2GB in formato GeoTIFF e di interrogare un satellite direttamente col suo nome; richiede necessariamente l'applicazione di una ROI; l'idea era quella di utilizzare SNAP[14] per **effettuare le analisi sui campi irrigati più comodamente**, dunque si era ritenuto necessario un export dell'immagine
- plotly (`pip3 install plotly`)[15]: permette di creare grafici interattivi e consultabili rapidamente; l'idea era quella di sfruttarla per poter plottare i grafici relativi all' NDVI ma è stata scartata per **incompatibilità con i dati di GEE**
- numpy (`pip3 install numpy`)[16]: permette la manipolazione di matrici e dati strutturati, da utilizzare insieme a matplotlib (`pip3 install matplotlib`)[17]; soluzione alternativa a quella dell'utilizzo di plotly, scartata perché non sarebbe stata una soluzione ottimale, in quanto si sarebbero dovuti utilizzare oggetti di due differenti librerie per poter costruire un solo grafico
- pyscreenshot (`pip3 install pyscreenshot`)[18]: permette di acquisire uno screenshot in modo automatico; l'idea era quella di **acquisire lo screen della mappa generata da GEE e poi inserirlo all'interno dell'applicativo**; scartata per la non predicibilità esatta dello screen, che considerava la finestra attiva (e dunque poteva cambiare) e le cui dimensioni non erano predefinite (ma variavano in base alla finestra). Questo il codice per effettuare lo screen, scartato dalla versione finale:

```
screen = Gdk.get_default_root_window().get_screen()
w = screen.get_active_window()
pb = Gdk.pixbuf_get_from_window(w, *w.get_geometry())
pb.savev("/home/antonio/Documenti/active.png", "png", (), ())
```

- rasterio (`pip3 install rasterio`)[19]: permette di **interrogare i satelliti e consultarne i dataset**, risolvendo incompatibilità tra formati diversi, per poi trattare quei dati come matrici N-dimensionali con numpy; l'idea era quella di utilizzarla per poter

[13] Fonte: https://pypi.org/project/gee2drive/
[14] Piattaforma sviluppata da Sentinel per l'osservazione terrestre e il processing di dati satellitari, comprensivo di numerose statistiche, tools e funzionalità estese. Per maggiori informazioni, consultare: https://step.esa.int/main/toolboxes/snap/
[15] Fonte: https://pypi.org/project/plotly/
[16] Fonte: https://pypi.org/project/numpy/
[17] Fonte:https://matplotlib.org/
[18] Fonte: https://pypi.org/project/pyscreenshot/
[19] Fonte: https://pypi.org/project/rasterio/0.13.2/

effettuare la vettorizzazione del file e la consecutiva estrazione dei campi irrigati; soluzione scartata in fase finale, dopo aver trovato la funzione corretta

- GDAL-OGR[20]: libreria per la manipolazione di dati geospaziali, composta da GDAL (manipolazione di raster data) e OGR (manipolazione di vector data); soluzione alternativa a gee2drive, scartata per via di un processo di installazione estremamente lungo.

  Per maggiore chiarezza, di seguito vengono presentate le principali differenze tra questi due **modalità di rappresentazione di dati geospaziali**:

  - i vector data sono **punti individuali memorizzati come coppia di coordinate (x,y)** e che possono essere collegati attraverso delle linee per creare dei poligoni, ma in generale, tutti i dati sono una lista di coordinate
  - i raster data sono **costituiti da pixel e ognuno di essi ha un valore associato**, che rappresenta un certo tipo di informazione; i pixel sono rappresentati come una griglia di celle

  La differenza, dunque, potrebbe riassumersi nella semplice affermazione che un'immagine raster è "**un'immagine continua**", mentre un'immagine vettoriale è "**un'immagine discreta**".

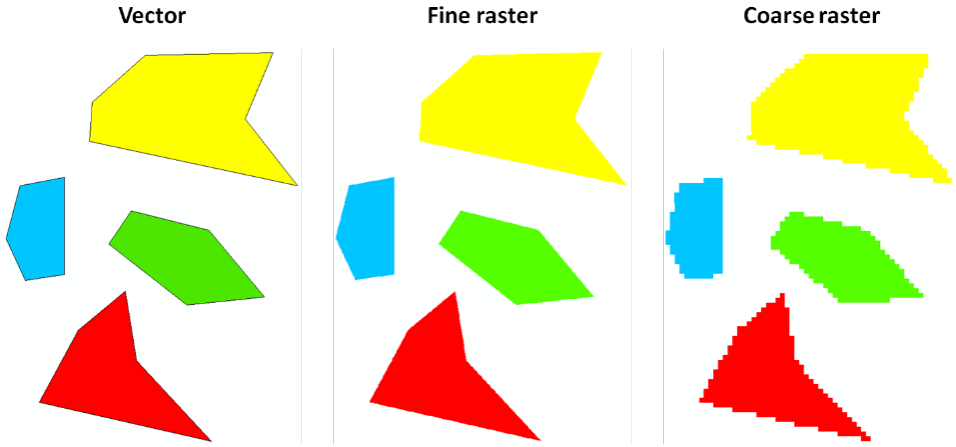


Figura 3: Differenza tra Vector e Raster: zoommando il Raster si cominceranno a vedere i pixel che formano i bordi del poligono, cosa che non accade con il Vector

Questa differenza si è rivelata estremamente utile, nelle fasi finali, per individuare la soluzione al problema del rilevamento.

È stato scelto di citare tali librerie in modo che, al lettore di questa relazione, possano suscitare la nascita di nuovi spunti per sviluppi futuri che portino al miglioramento del tool.

[20] Fonte: https://pypi.org/project/GDAL/

5. Fase 1 – impostazione iniziale e ricerca della soglia migliore

5.1 Inizializzazione

Nella prima fase di codifica, l'obiettivo principale è stato quello di prendere confidenza con la Python API e ricercare la soglia migliore per l'NDVI. Il procedimento sarà più chiaro nelle seguenti righe.

Per poter sfruttare le potenzialità offerte da Google Earth Engine (ee), diventa necessario lavorare su un oggetto proxy (proxy object), che va inizializzato attraverso il suo metodo costruttore:

```
ee.Initialize()
```

Questo oggetto **non contiene nessun dato vero e proprio**, ma fa da tramite per la gestione degli oggetti che sono realmente presenti sui server di Google Earth[21].

5.2 Come interrogare il satellite

Il primo passo è stato creare una Region of Interest (ROI) entro la quale effettuare l'analisi. L'area in questione, la zona della **Capitanata**, è stata empiricamente racchiusa in un rettangolo, i cui vertici sono stati trovati selezionandoli da Google Maps.

Per poter estrarre tale ROI esistono due metodi principali: il primo, che **racchiude la ROI in un cerchio**, il secondo, che la **racchiude in un rettangolo**. In questa fase, si è optato per la forma rettangolare, in modo che possa essere proiettata più facilmente.

```
area = ee.Geometry.Polygon([[X1,Y1], [X2,Y2], [X3,Y3], [X4,Y4]])
```

La classe Geometry permette di creare delle aree di interesse. Con la sua subclasse Polygon, è possibile creare un poligono passando come parametro il numero di vertici, specificati sotto forma di coordinate geografiche.

Per completezza, di seguito è mostrato anche il metodo per poter circondare la ROI con un cerchio:

```
area = ee.Geometry.Point([[centro]).buffer(raggio)
```

In questo caso, la subclasse Point richiede un solo parametro che corrisponde alla coordinata presso la quale piazzare il centro della circonferenza; il metodo buffer, specifica la dimensione del raggio (se non specificato, si considererà soltanto quel punto).

[21] Fonte: https://developers.google.com/earth-engine/client_server

In seguito, si è proceduto a interrogare il satellite per l'ottenimento delle immagini. Per prendere tali informazioni, ci sono due strade percorribili:

- considerare una singola immagine, attraverso la classe **Image** (ee.Image())
- considerare una collezione di immagini, **ossia una pila di fotografie scattate all'interno di un intervallo temporale e in cui il primo elemento è il più recente**, attraverso la classe **ImageCollection** (ee.ImageCollection()) - si è optato per questa soluzione in modo da ottenere informazioni sull'NDVI più dettagliate, sebbene sia noto che basti una singola osservazione per determinarne il valore[22].

Il codice è qui di seguito presentato:

```
collection = (ee.ImageCollection('COPERNICUS/S2')
              .filter(ee.Filter.lt("CLOUDY_PIXEL_PERCENTAGE", X))
              .filterDate(dataInizio, dataFine)
              .filterBounds(area))
```

Il primo parametro inserito è il nome del satellite. Seguono dei metodi per poter filtrare il risultato:

- `ee.Filter.lt("CLOUDY_PIXEL_PERCENTAGE", X)` serve a escludere i pixel che hanno più del X% di copertura o ombre dovute a nuvole – la ricerca avviene tra i metadati dei pixel stessi; **più basso sarà il valore di X, più rigido sarà il filtro**. Ciò potrebbe portare ad avere diverse immagini scartate nei mesi invernali, per cui **si consiglia di aumentare tale valore nei suddetti periodi**;
- `filterDate` considera un intervallo temporale compreso tra due date;
- `filterBounds` serve a ritagliare esattamente l'area di interesse, passando come parametro la ROI precedentemente creata; è possibile anche creare la ROI direttamente inline, evitando la dichiarazione di una variabile

---

5.3 Ricerca della soglia

Per poter determinare la soglia migliore, è stato perseguito un procedimento empirico di seguito descritto.

All'interno della ROI sono stati selezionati 8 punti, corrispondenti a 8 campi: quattro irrigati, quattro non irrigati[23]. Tali punti sono stati raccolti in un elemento **FeatureCollection** di

[22] Fonte: Identifying Irrigated Areas in the Snake River Plain, Idaho: Evaluating Performance across Compositing Algorithms, Spectral Indices, and Sensors - Eric W. Chance 1, Kelly M. Cobourn, Valerie A. Thomas, Blaine C. Dawson and Alejandro N. Flores

[23] I punti sono stati trovati consultando Google Earth e osservando se il campo fosse coperto di verde o meno

GEE (ee.FeatureCollection()) che permette di raccogliere insieme forme ed elementi diverse, dando anche la possibilità **di considerare diverse ROI contemporaneamente**.

Nel nostro caso, ne sono presenti due: una contenente i quattro campi irrigati, un'altra con quelli non irrigati.

```
punti_irrigati = [
      ee.Feature(ee.Geometry.Point(X1, Y1).buffer(300), {'name': 'Irrigato 1'}),
      ee.Feature(ee.Geometry.Point(X2, Y2).buffer(300), {'name': 'Irrigato 2'}),
      ee.Feature(ee.Geometry.Point(X3, Y3).buffer(300), {'name': 'Irrigato 3'}),
      ee.Feature(ee.Geometry.Point(X4, Y4).buffer(300), {'name': 'Irrigato 4'})
]
lista_irr = ee.FeatureCollection(punti_irrigati)
```

Punto	Coordinate
Irrigato 1	[15.616944, 41.508055]
Irrigato 2	[15.745277, 41.561388]
Irrigato 3	[15.475277, 41.615555]
Irrigato 4	[15.652222, 41.429722]

Tabella 3: Lista dei punti irrigati

Punto	Coordinate
Non Irrigato 1	[15.566944, 41.496111]
Non Irrigato 2	[15.571666, 41.528055]
Non Irrigato 3	[15.860833, 41.441111]
Non Irrigato 4	[15.854166, 41.491944]

Tabella 4: Lista dei punti non irrigati

Sono stati considerati quattro intervalli temporali per analizzare i sopracitati punti, da cui il satellite ha restituito anche il numero di osservazioni:

- Novembre 2017 – **12 immagini**
- Maggio 2018 – **9 immagini**
- Luglio 2018 – **22 immagini**
- Agosto 2019 – **24 immagini**

Su ogni immagine di queste collezioni è stata aggiunta una nuova banda, relativa all'NDVI e calcolata con la seguente funzione:

```
def NDVI_calc(image):
    return image.addBands(image.normalizedDifference(['B8', 'B4']))
```

Nell'appendice A sono mostrati i grafici per tutti i punti e per ogni intervallo temporale, ottenuti grazie a ipygee.

Dalla lettura di questi grafici è emerso che **i campi irrigati hanno un indice di vegetazione maggiore a 0.3 nell'84% delle osservazioni** mentre **quelli non irrigati superano tale valore soltanto nel 3% delle osservazioni**: è per questo motivo che si è deciso di impostare la soglia a 0.35[24].

Stampando la ROI si ottiene il seguente risultato.

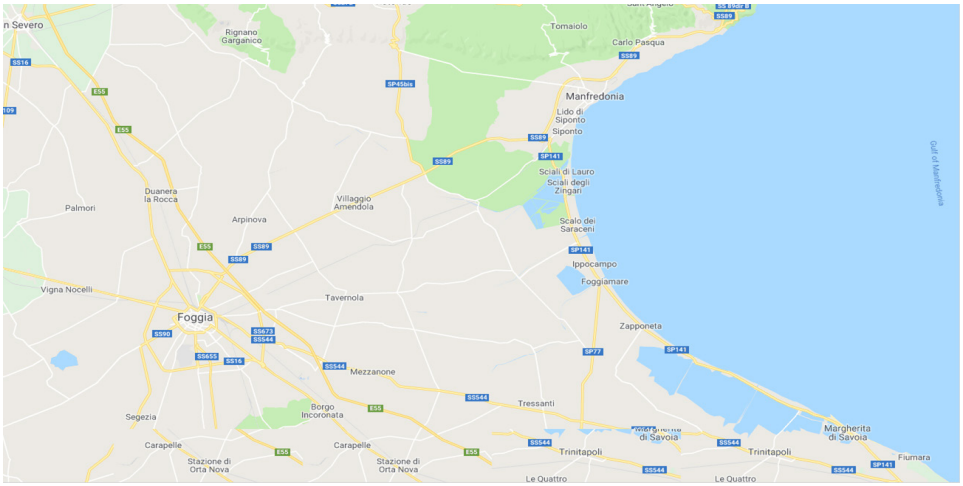


Figura 4: Zona di studio considerata

Applicando su di essa la banda NDVI e la soglia, il risultato è il seguente:

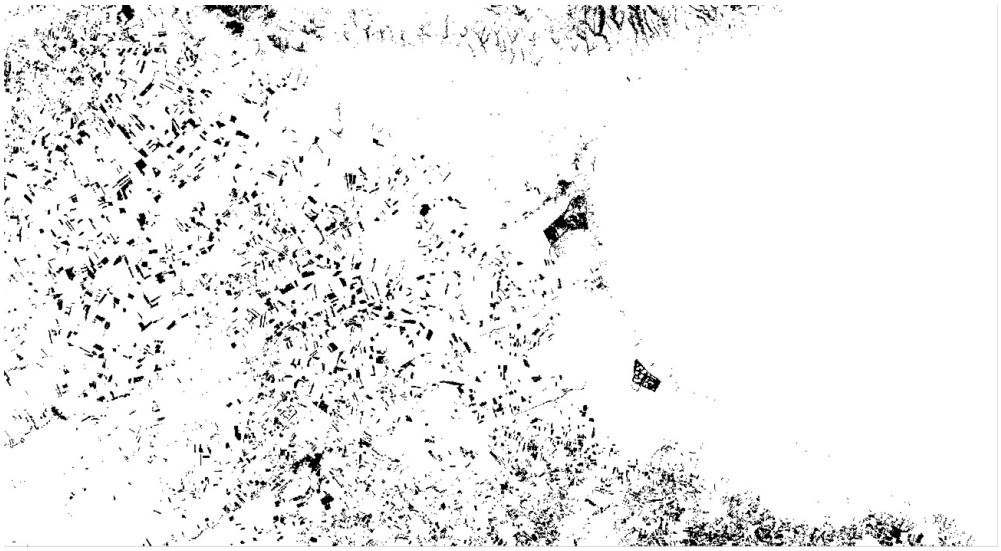


Figura 5: Applicazione della banda NDVI sulla zona di studio, considerando solo le aree sopra soglia

[24] Si giunge allo stesso risultato anche considerando il valore medio di NDVI (ottenendo una singola immagine per ogni collezione in quanto la collezione viene fatta "**collassare**" su un unico layer), oppure facendo una media dei valori medi e considerando una sola immagine.

All'immagine è stata applicata un **palette di colori binaria**, in modo che le **aree nere fossero quelle sopra soglia mentre quelle bianche fossero quelle sotto soglia**.

Nella seguente immagine, infine, sono indicati i campi irrigati e non irrigati, in modo da visualizzarne la posizione.

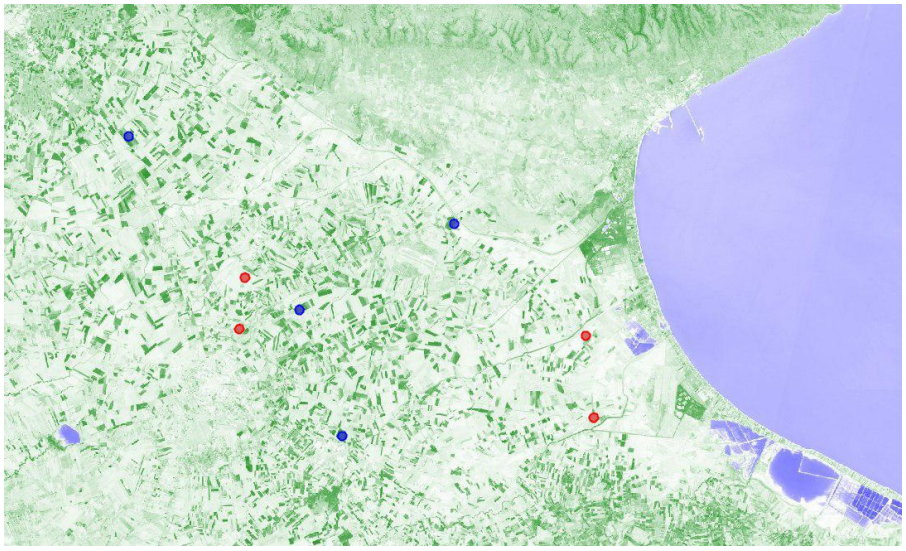


Figura 6: Posizione dei campi sicuramente irrigati (in blu) e posizione dei campi sicuramente non irrigati (in rosso)

In Appendice A è possibile visionare tutti i grafici estratti per ogni singolo punto in ogni singolo intervallo di osservazione.

6. Fase 2 – creazione dell'interfaccia grafica

6.1 Una piccola overview su TkInter

Appurata la soglia, il secondo step di questo tema d'anno è stato incentrato sullo sviluppo dell'interfaccia grafica. Python mette a disposizione il package TKInter. Questo package è lo standard de-facto per la costruzione di una GUI in Python: sebbene non sia l'unico disponibile, resta il più utilizzato e ricco di funzionalità.

TKInter permette di gestire la schermata sotto forma di matrice, dove ogni cella è identificata da una row e una column. Per inserire un oggetto all'interno della stessa, dunque, è fondamentale **specificare in quale riga e colonna sarà inserito**: se dovesse essere specificato un valore di riga (o colonna) `X` ma le precedenti righe (o colonne) `X-k` saranno libere, l'oggetto verrà automaticamente posizionato nella prima cella libera; **prenderà il posto stabilito soltanto quando le precedenti righe (o colonne) verranno riempite**.

Esistono altri due metodi per poter posizionare un elemento:

- il metodo place(), che richiede le esatte coordinate in cui dovrà essere inserito
- il metodo pack(), che non richiede nessun parametro e inserirà l'oggetto automaticamente nella prima posizione libera

Di seguito, sono mostrati i metodi appena discussi

```
oggetto.grid(row = X, column = Y, columnspan = Z, sticky = "W")
oggetto.place(x = X, y = Y)
oggetto.pack()
```

Per maggiore chiarezza riguardo la struttura dell'interfaccia grafica, di seguito verranno riassunti i widget disponibili (di colore rosso saranno quelli utilizzati nell'applicativo).

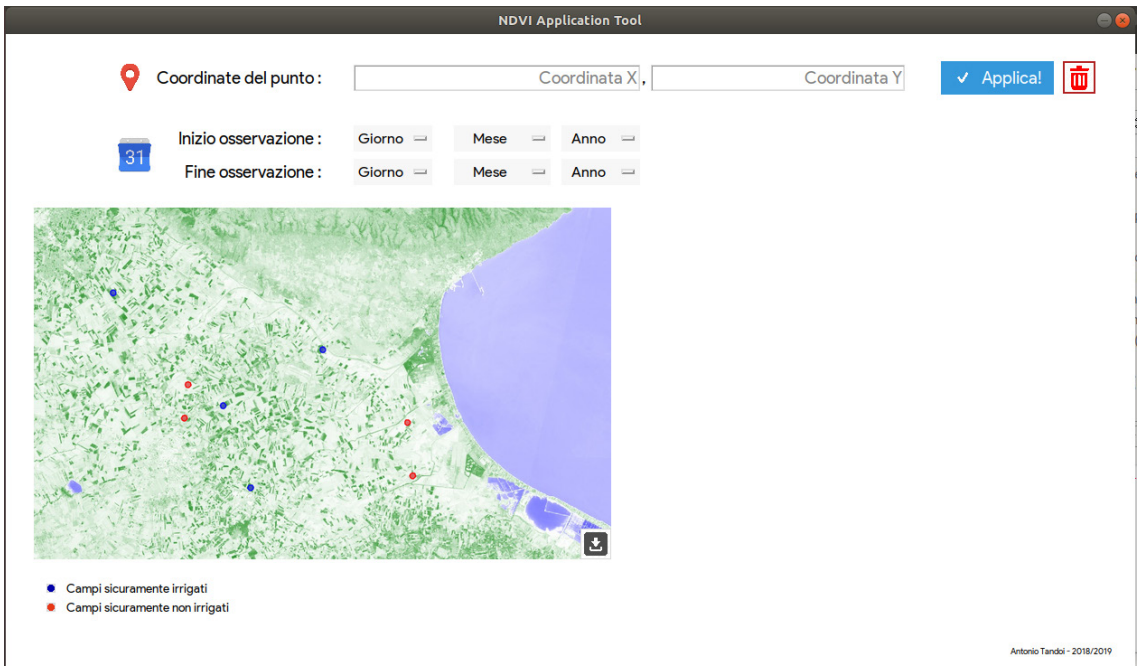
Categoria	Nome	Funzionalità
Base	Tk	Contenitore principale di tutta la finestra
	TopLevel	Finestra principale dell'applicativo
	Frame	Consente di raggruppare gli oggetti delle finestre. Va interpretato come un componente "logico" della finestra.
	Button	Pulsante cliccabile
	Checkbutton	Inserimento di spunta

	Entry	Casella di testo monoriga
	Label	Etichetta non editabile
	ListBox	Lista di opzioni testuali, tutte contemporaneamente visibili
	OptionMenu	Menù popup
	PhotoImage	Mostrare immagini
	RadioButton	Pulsanti a selezione singola
	Scale	Slider
Decorati	**Canvas**	Area di disegno per archi, immagini bitmap, linee, ovali, poligoni, rettangoli, testi.
	Text	Testo multilinea
Compositi	Menu	Menu per inserire etichette cliccabili
	MenuButton	Menù a discesa
	ScrollBar	Da usare congiuntamente a una ListBox, Canvas o Text

Tabella 5: Widget forniti da TkInter

Sulla base di questo approccio, la schermata dell'applicativo è stata costruita su 3 principali righe (contando, in totale, 6 righe e 4 colonne).

Si procede a una descrizione dell'interfaccia grafica e una guida al suo utilizzo.



6.2 Prima riga

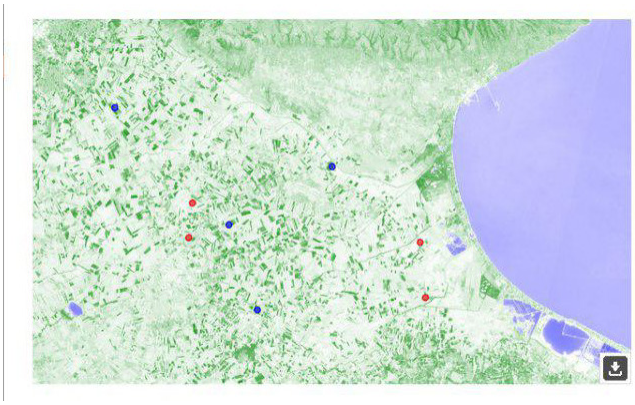
La prima riga si compone di due TextBox in cui l'utente dovrà inserire le coordinate del punto che ha intenzione di controllare. Le TextBox si azzerano automaticamente al clic del mouse. È presente il Button "Applica" per far partire l'elaborazione e il Button "Cancella" per azzerare tutti i campi con un solo click.

6.3 Seconda riga

La seconda riga permette di inserire l'intervallo di osservazione, composto da una data iniziale e una data finale.

6.4 Terza riga

La terza riga è composta da due Canva, degli elementi di TkInter che possono contenere oggetto più elaborati e al cui interno è possibile anche disegnarne altri. Il Canva sinistro contiene la mappa dei campi irrigati e non irrigati, il Canva destro mostrerà il grafico dell'NDVI (**inizialmente vuoto**).



Una precisazione: inizialmente il Canva sinistro doveva contenere la mappa generata direttamente da Google Earth Engine. Tuttavia, a seguito di una fase di documentazione, **questa soluzione non è assolutamente implementabile attualmente** in quanto gli

sviluppatori stessi di GEE **non hanno previsto la possibilità di includere tale mappa all'interno di un applicativo**[25].

Inoltre, **non è altrettanto possibile mostrare a video due finestre diverse** (una contenente GEE e l'altra l'applicativo in questione). Il motivo è dato dal fatto che anche la schermata di GEE viene eseguita in un mainloop, quindi **andrà in conflitto con l'applicazione creata, che necessiterà anch'essa di un** mainloop; questo accade a prescindere dalla libreria grafica utilizzata.

6.5 Quarta riga

Nella sezione più bassa sono presenti unicamente delle Label per la riga superiore.

6.6 Gestione degli eventi

Ad esclusione dei pulsanti cliccabili, che **hanno un metodo già associato e che permette di compiere delle azioni al click**, gli altri elementi non godono di questa peculiarità. Se si vuole **associare un evento a un generico widget** bisogna effettuare un cosiddetto binding. Di seguito sono presentati i principali eventi; è possibile ottenere un quadro più completo consultando la documentazione online[26].

Evento	Descrizione
<Button-1>	Click sinistro del mouse
<Button-2>	Click centrale del mouse
<Button-3>	Click destro del mouse
<BX-Motion>	Il pulsante X è tenuto cliccato mentre il mouse è mosso
<ButtonRelease-X>	Quando il pulsante X viene rilasciato
<Double-Button-1>	Doppio click con pulsante X
<Enter> / <Leave>	Il mouse passa sul widget / Il mouse esce dal widget
<FocusIn> / <FocusOut>	Focus ricevuto da tastiera

[25] Fonte: Rodrigo Principe - https://gis.stackexchange.com/questions/336664/earth-engine-into-tkinter-canva
[26] Fonte: http://www.effbot.org/tkinterbook/tkinter-events-and-bindings.htm

Facendo un esempio, quindi, si consideri l'`Entry` per poter inserire la coordinata. Ad essa è associata l'evento per cui, al click del mouse, l'`Entry` si azzeri per poter inserire un nuovo testo:

```
inputX.bind("<Button-1>", cancellaX)
```

"cancellaX" è la funziona associata al click.

7. Applicazione dei dati

Una volta che l'utente ha inserito i dati verranno computate le informazioni e l'interfaccia grafica subirà un cambiamento. È possibile suddividere il submitting in quattro fasi principali:

- controllo sull'input
- interrogazione del satellite
- creazione del grafico
- generazione dei pulsanti

7.1 Controllo dei dati inseriti

Vengono implementati due controlli principali, uno sulle coordinate e uno sulle date.

Il **controllo sulle coordinate** è così eseguito:

- controllare che la TextBox non sia vuota
- controllare che il punto che separa la parte decimale sia in seconda posizione all'interno della stringa

Il **controllo sulle date** è così eseguito:

- controllare che gli OptionMenu siano non vuoti
- controllare che la data iniziale di osservazione non sia precedente a quella del lancio del satellite
- controllare che la data finale di osservazione non sia successiva alla data odierna
- controllare che il giorno selezionato sia corretto per il mese scelto
- controllare se l'anno sia bisestile per permettere di selezionare anche il 29 febbraio

---

7.2 Interrogazione del satellite

Se non viene restituito alcun errore (attraverso una MessageBox), verranno generate le date e convertite nel formato accettato da GEE (ee.Date()).

Dunque, si creerà l'area di test, come segue:

```
test1 = ee.Geometry.Point(coordinataX, coordinataY).buffer(500)
```

dove si ottiene una **circonferenza centrata nelle coordinate che l'utente ha inserito e con un raggio di 500px** (per un'area di circa 80km^2)[27]

[27] Ricordando che 1px = 10m

Successivamente, si procede all'interrogazione del satellite nello stesso modo discusso precedentemente, considerando le due date inserite in input e la ROI inizialmente definita. Ancora una volta, per ogni immagine si calcola l'NDVI e si ottiene poi un singolo layer su cui viene applicata la soglia. Questa immagine sarà utile nell'ultima fase.

6.3 Creazione del grafico

Considerando la collezione ottenuta dal satellite e l'area di test definita dall'utente, si genererà il grafico relativo all'**andamento dell'NDVI nell'intervallo temporale selezionato.** La funzione prende in input la collezione di immagini su cui è stata applicata la banda NDVI, estrae l'area specificata, effettua una zoom di 10 volte e considera unicamente la banda NDVI stessa.

```
chartest = chart.Image.series(**
    {
            'imageCollection': imNDVI1,
            'region': test1,
            'scale': 10,
            'bands': ['nd'],
            'label_bands':['NDVI Band']
    })
    chartest.render_to_png('/home/antonio/Documenti/chart.png')
```

Il suddetto grafico viene salvato come immagine PNG e successivamente caricato all'interno del Canva destro. Ciò è possibile grazie alle librerie ipygee e pygal discusse nella sezione corrispondente.

6.4 Generazione dei pulsanti

Infine, il pulsante di download presente nell'angolo in basso a destra del Canva sinistro, permetterà di generare due pulsanti, che daranno la possibilità di **scaricare l'immagine** in formato PNG e SVG.

```
urlpng = imNDVI4.getThumbURL(
    {
        'region': tools.geometry.getRegion(test1),
        'format': 'png'
    })
urlsvg = imNDVI4.getDownloadURL({ 'region': tools.geometry.getRegion(test1)})
```

La funzione `tools.geometry.getRegion(test1)` permette di convertire la ROI definita attraverso un insieme di coordinate in una ROI accettabile dal comando.

Inoltre, alla schermata verrà aggiunto un nuovo pulsante, nell'angolo in basso a sinistra del Canva sinistro, che permetterà le operazioni di rilevamento.

Figura 7: Aspetto della GUI dopo aver sottoposto la ricerca

Come è possibile notare, il Canva sinistro è stato aggiornato all'area che l'utente ha specificato.

È necessaria una precisazione: l'area calcolata è una circonferenza, ma il metodo per poter salvare l'immagine e mostrarla **estrae il massimo rettangolo possibile da tale circonferenza**: significa che l'applicativo non mostrerà l'intera area ma solo una sua parte. Questo avviene soltanto per il plotting a video: nel rilevamento e salvataggio delle aree presumibilmente irrigate, **l'area considerata sarà sempre la circonferenza**.

8. Rilevamento dei campi irrigati

La fase finale consiste nel rilevare quali sono i campi irrigati e poi salvarli all'interno di un file. Questa operazione è stata la più problematica da eseguire all'interno di questo tema d'anno ed è stata quella che ha richiesto più tempo, per la **difficoltà nel trovare un metodo funzionante in Python**.

L'idea di partenza si strutturava nei seguenti step:

- ottenere un'immagine binaria (campi irrigati in nero, altre zone in bianco)
- rilevare i contorni all'interno dell'area selezionata
- effettuare una pattern recognition in modo da riconoscere le forme rettangolari
- salvare i centri di questi campi come coordinate

Figura 8: Area della Capitanata sotto forma di immagine binaria

A seguito di numerose ricerche, si è provata al Canny Edge Detection[28] (la quale ha richiesto una piccola fase di documentazione per capirne il funzionamento). Questo algoritmo opera su diverse fasi:

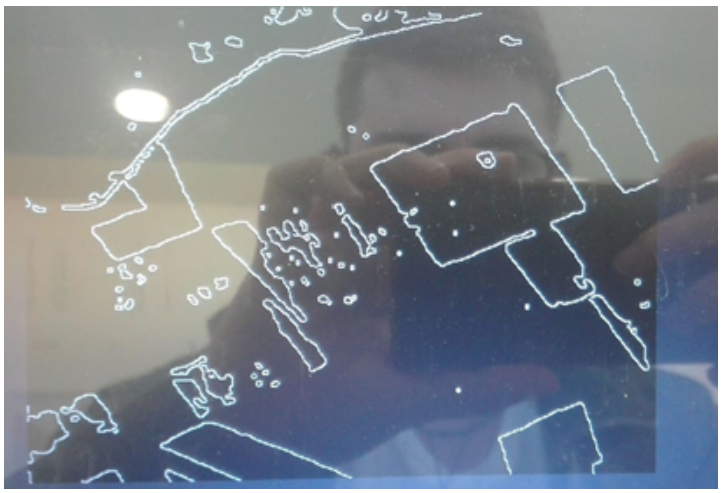
- **noise reduction**: effettuata applicando una **sfocatura gaussiana**
- **gradient calculation**: **rilevazione dell'intensità del bordo** in base al cambio di intensità del pixel
- **non-maximum suppression**: fase che permette di **avere dei bordi meno marcati** attraverso un calcolo dei massimi in ogni direzione
- **double treshold**: serve a rilevare tre tipi di pixel ossia high pixel (contribuiscono sicuramente alla rilevazione del bordo), weak pixel (contribuiscono, ma in maniera minore), non-relevant (non contribuiscono al bordo); la doppia soglia, dunque, ha proprio questo scopo:

[28] Spiegazione completa dell'algoritmo: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

- high treshold: se superata, si ha un high pixel; se non superata si ha un weak pixel
- low treshold: se non superata, si ha un non-relevant pixel

Se l'immagine è binaria, basta solo una soglia.

Applicando la CannyEdge all'immagine binaria, si ottiene questo risultato:



Arrivati a questo punto tuttavia, non si ha alcun modo per poter reperire le informazioni riguardanti le coordinate da tali campi. L'idea, pertanto, è stata scartata.

L'approccio, dunque, è stato modificato con l'obiettivo, ora, di **vettorizzare un'immagine "raster"**.

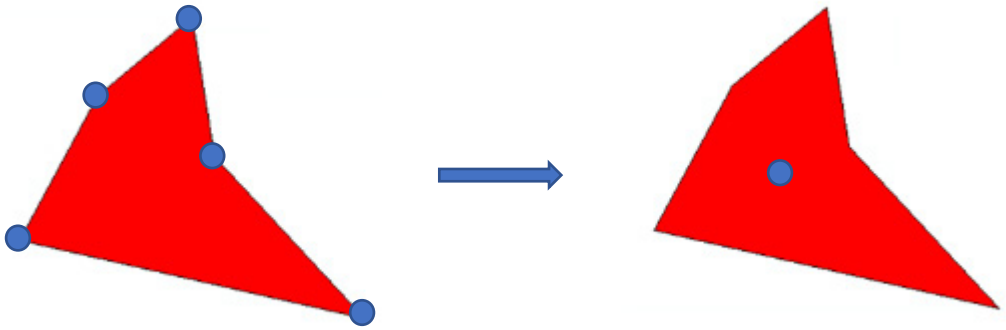
Conservando sempre l'idea iniziale dell'applicativo, **il rilevamento avviene nell'area specificata dall'utente allargata a 1500px** (per un'area di circa 700km^2)**, considerando l'immagine binarizzata** (in questo caso, campi bianchi su fondo nero)

Partendo da questa, si utilizza il seguente codice:

```
centroidi = imNDVI4.reduceToVectors(
        geometry = test2,
        geometryType = 'centroid',
        scale = 30,
        crs = 'EPSG:4326',
        eightConnected = True
    )
```

La funzione reduceToVectors() ha diversi parametri che può accettare; i più importanti sono spiegati di seguito:

- `geometry`: l'area da analizzare
- `geometryTye`: cosa considerare nell'analisi dei poligoni; con il parametro centroid si considereranno i centroidi dei suddetti, con il parametro polygon si considereranno i vertici; per individuare il centroide, l'algoritmo individua prima tutti i vertici e dopodiché **calcola il punto che interpola al meglio il centro del poligono, in modo da minimizzare la distanza da ogni vertice**



Dato che **si presume che un campo abbia forma rettangolare, quadrata o simile ad essa**, i centroidi che troverà questo algoritmo si avvicineranno di molto al punto di intersezione delle diagonali

- `eightConnected`: indica se considerare per il riconoscimento il **criterio di connettività a 8 pixel** o meno

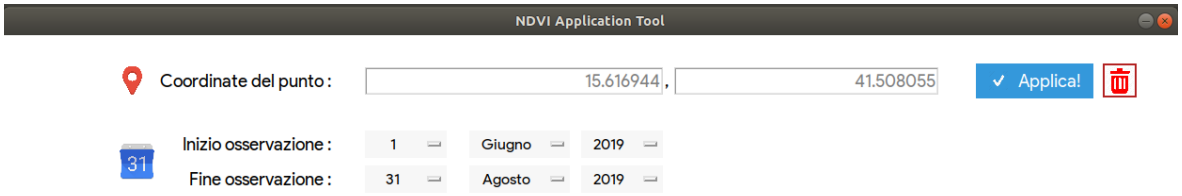
Il risultato di questa operazione è una FeatureCollection composta da numerose feature, tra cui figurano quelle di nostro interesse, ossia le coordinate del centroide. Il salvataggio delle informazioni avviene su un file TXT e CSV (attraverso un dataframe).

Per la consultazione si consiglia di utilizzare il file TXT, sicuramente più chiaro della sua controparte in CSV. Si è scelto di effettuare il salvataggio in CSV per avere una funzionalità in più che possa essere sfruttata per scopi futuri.

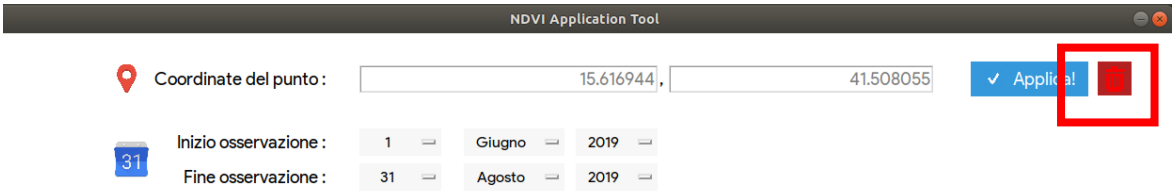
9. Tutorial: come utilizzare l'applicativo

L'intera applicazione è stata strutturata in modo da essere la più intuitiva possibile: **il suo utilizzo è semplicissimo**.

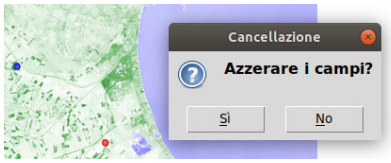
Step 1: Inserire le coordinate del punto da controllare. La coordinata sinistra è la longitudine, quella destra è la latitudine. Successivamente, inserire le date per considerare un intervallo di osservazione.



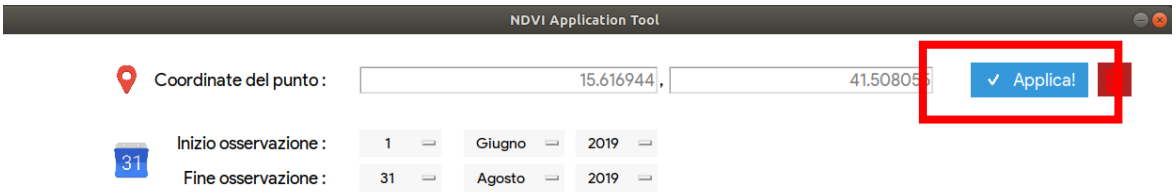
Schiacciando il pulsante rosso all'estrema destra, è possibile azzerare i campi in un solo click.



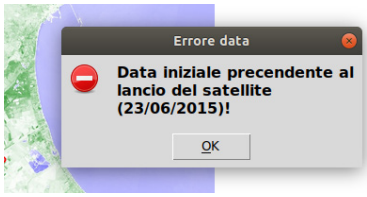
Apparirà una finestra di conferma affinché l'utente possa sentirsi sicuro della propria scelta.

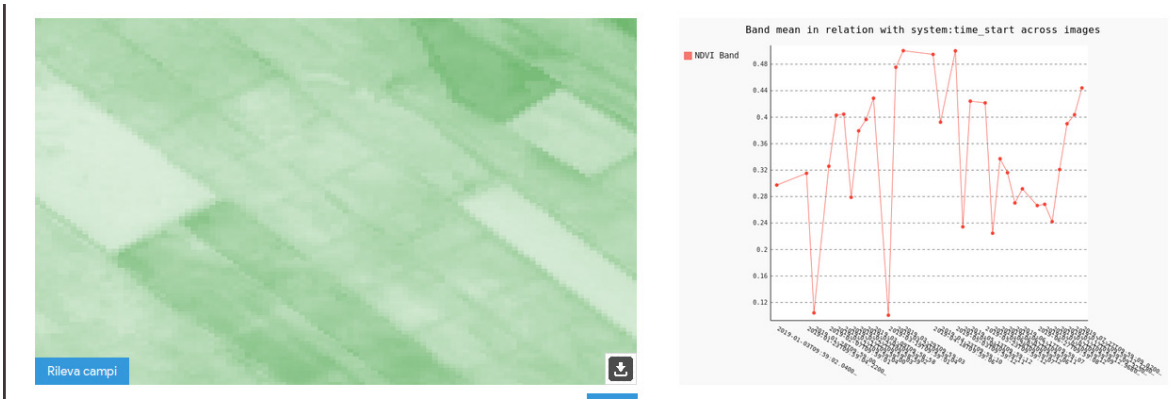


Se, invece, l'utente cliccherà su applica, partirà la prima elaborazione, calcolando l'NDVI nell'area circostante alle coordinate in input.

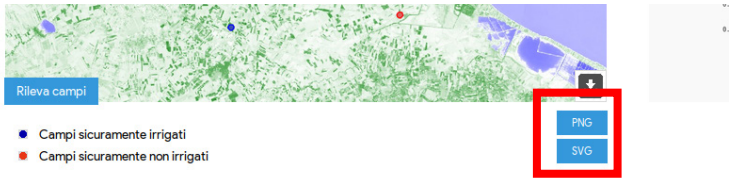


Alcuni messaggi dei diversi errori che possono comparire:

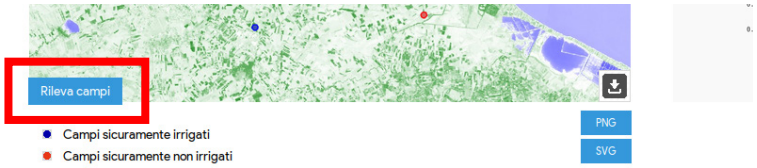




Step 2: Apparirà un grafico sulla destra. Cliccando l'icona di download, appaiono due pulsanti: entrambi aprono una finestra nel browser, il primo visualizzerà l'immagine in PNG, l'altro permetterà il download del file SVG.



Inoltre, sarà apparso un nuovo pulsante cliccabile, in basso a sinistra. Cliccandolo, viene avviata la routine di rivelazione.



Dopodichè, apparirà anche il pulsante che permette di esportare i campi in un TXT o in CSV.

Sviluppi futuri

Gli sviluppi futuri su come poter migliorare l'esperienza di utilizzo del software, sono molteplici. Già in sede di questo tema d'anno, si stava procedendo a poterne sviluppare qualcuno. Di seguito, qualche idea:

- **permettere l'inserimento di Google Maps** all'interno del Canva sinistro, ad ogni nuova richiesta fatta dall'utente, in modo tale da avere una schermata più interattiva e consultabile; il motivo per cui questa feature non è stata inserita è per via del fatto che **usando l'API di Google Maps, si possono effettuare un numero estremamente limitato di richieste al mese** e diventa, pertanto, **necessario utilizzare altre API con servizi simili**;
- **rendere "intelligente" il software**, permettendogli automaticamente di **riconoscere quando un'area è irrigata**; al momento non è una feature possibile in quanto il grafico è semplicemente un'immagine: riuscendo a **implementare il grafico sotto forma di tabella**, si potrebbero contare il numero di osservazioni in cui l'NDVI è sopra soglia, attraverso delle `funzione aggregate`, e se tale numero supera una certa soglia, allora **quella zona potrebbe essere presumibilmente irrigata nell'intervallo temporale sottoposto**;
- **calcolare la percentuale di area irrigata e non irrigata** all'interno dell'area sottoposta; questa è la feature più semplice da implementare, tant'è che **parte del codice è già presente all'interno del software**; l'idea è quella di **contare i pixel bianchi sul totale di pixel in quell'area e ricavare la percentuale**: dato che il pixel bianco rappresenta la zona irrigata, calcolandone la somma sarebbe stato possibile capire quanta parte dell'area cercata è irrigata; il grafico **sarebbe dovuto essere un istogramma dinamico, che prende i dati direttamente dal calcolo appena discusso**; l'idea non è stata più portata avanti per la difficoltà nel trovare un metodo che conti uno per uno i pixel (al lettore, la `funziona pixelArea()`, già testata, restituisce l'area di tutta i pixel dell'area);
- **storico delle ricerche effettuate**, in modo da mostrare le quattro ricerche più recenti (una funziona simile alla ricerca di Google);
- **creazione di un database dei campi irrigati**, che venga aggiornato tramite una funzione periodica e che calcoli se per un campo, il centroide si sia spostato (e in tal caso, **in che direzione in modo da capire su quale asse il campo si sia esteso o ridotto**) o sia rimasto lo stesso;
- **colorare i campi rilevati** direttamente nel `Canva`; questa feature è molto complicata da applicare in quanto sarebbe necessario modificare con un colore apposito, un'immagine di `GEE`; ciò comporterebbe necessariamente l'interrogazione dei server di `GEE` attraverso `ee`;

Appendice A: grafici relativi ai campioni

In questa sezione sono presenti i grafici relativi ai campioni considerati per impostare la soglia per l'NDVI. Nella prima parte ci sono **tutti i campioni irrigati**.

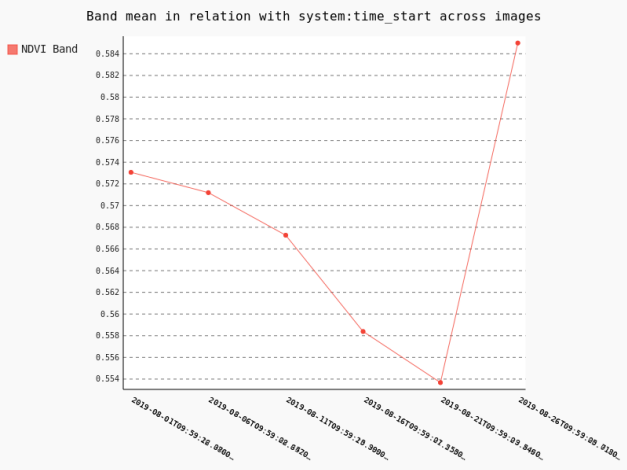


Grafico 1: Campo irrigato 1 - Agosto 2019

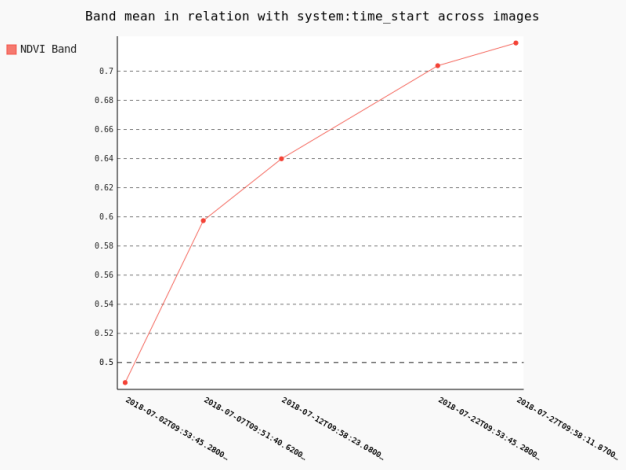


Grafico 2: Campo irrigato 1 - Luglio 2018

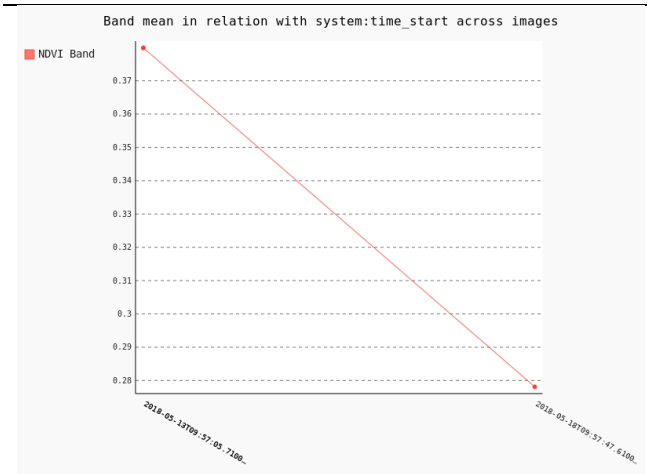


Grafico 3: Campo irrigato 1 - Maggio 2018

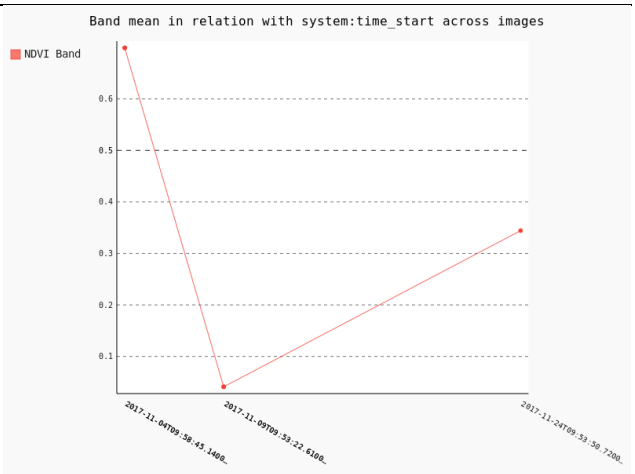
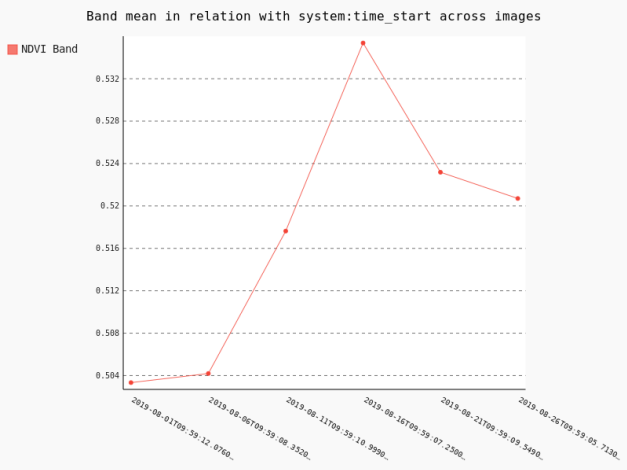


Grafico 4: Campo irrigato 1 - Novembre 2017



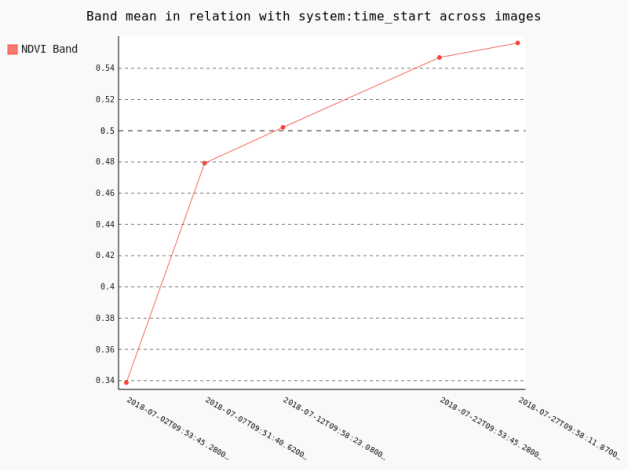


Grafico 5: Campo irrigato 2 - Agosto 2019

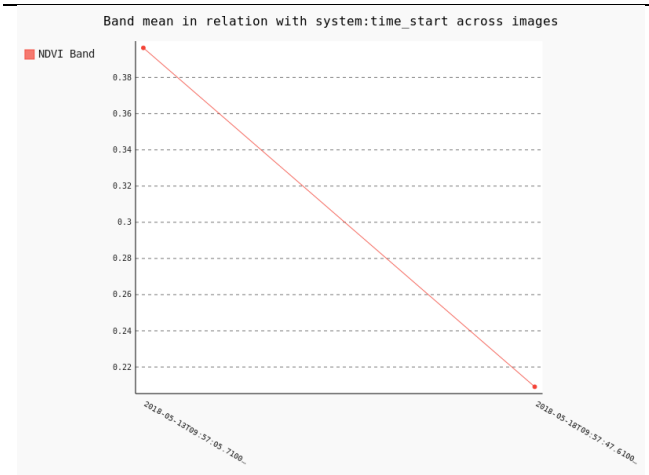


Grafico 6: Campo irrigato 2 - Luglio 2018

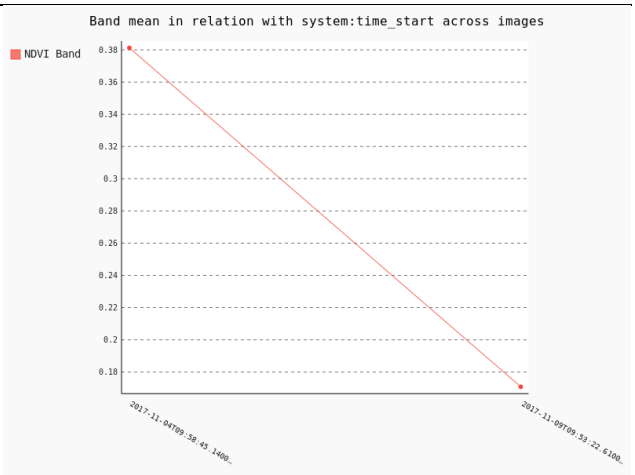


Grafico 7: Campo irrigato 2 - Maggio 2018

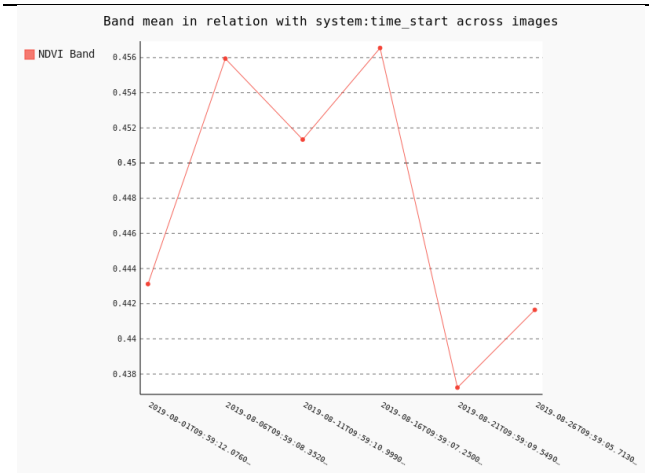


Grafico 8: Campo irrigato 2 - Novembre 2017

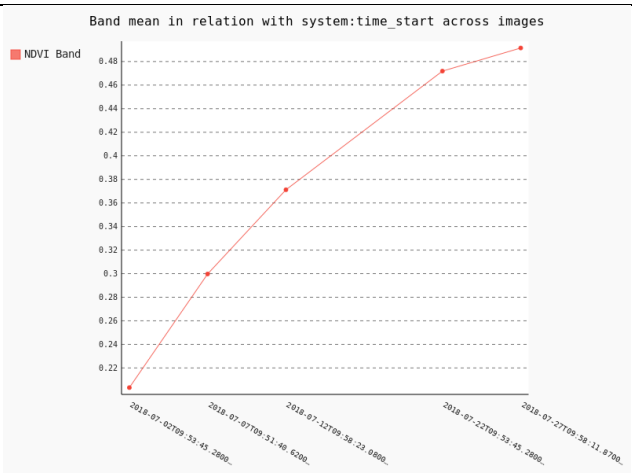


Grafico 9: Campo irrigato 3 - Agosto 2019

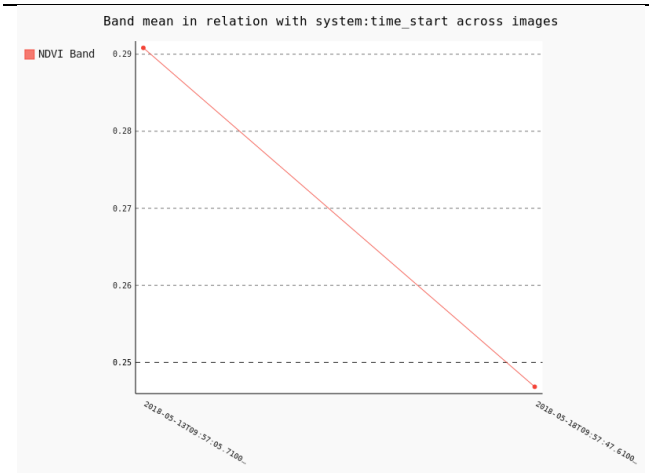


Grafico 10: Campo irrigato 3 - Luglio 2018

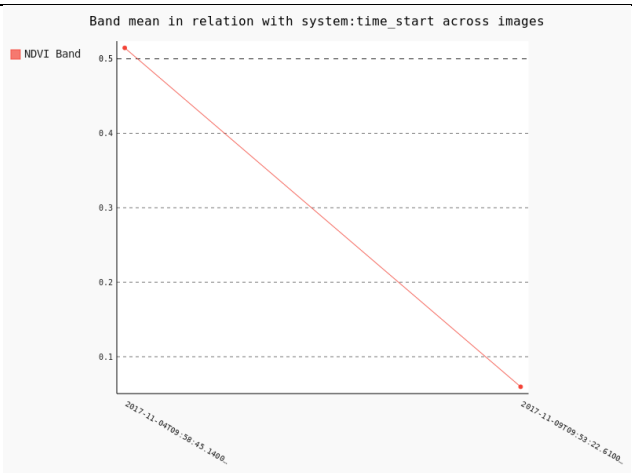


Grafico 11: Campo irrigato 3 - Maggio 2018

Grafico 12: Campo irrigato 3 - Novembre 2017

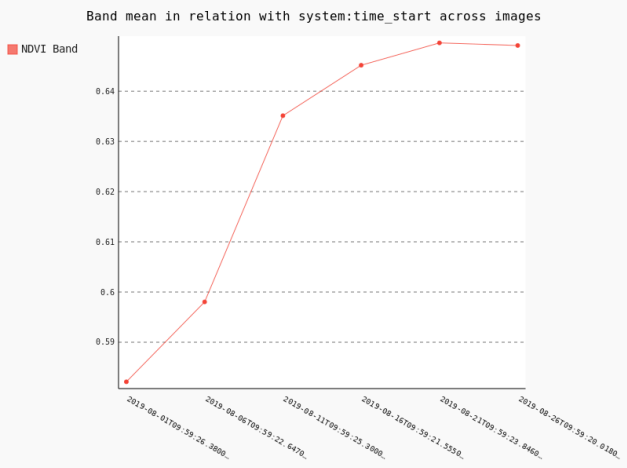


Grafico 13: Campo irrigato 4 - Agosto 2019

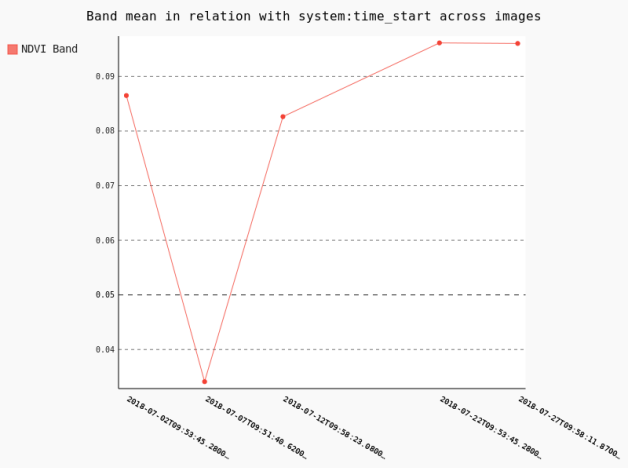


Grafico 14: Campo irrigato 4 - Luglio 2018

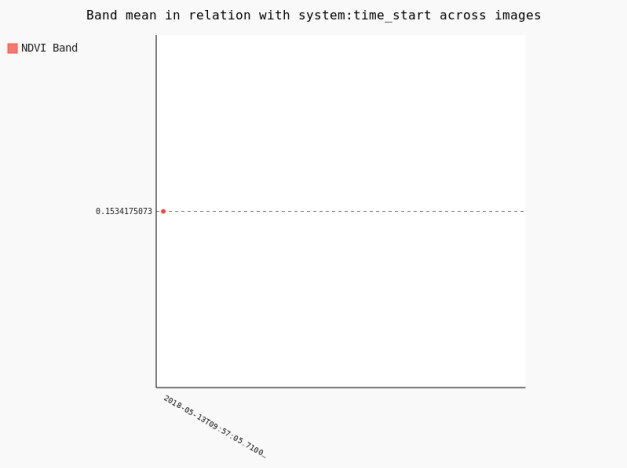


Grafico 15: Campo irrigato 4 - Maggio 2018

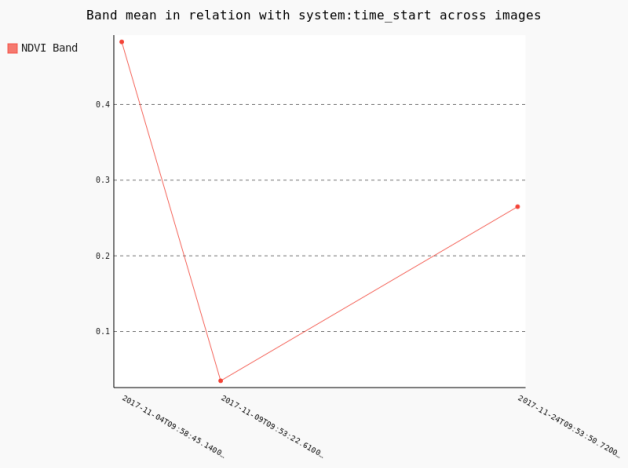


Grafico 16: Campo irrigato 4 - Novembre 2017

Nella seconda sezione ci sono **tutti i campi non irrigati**.

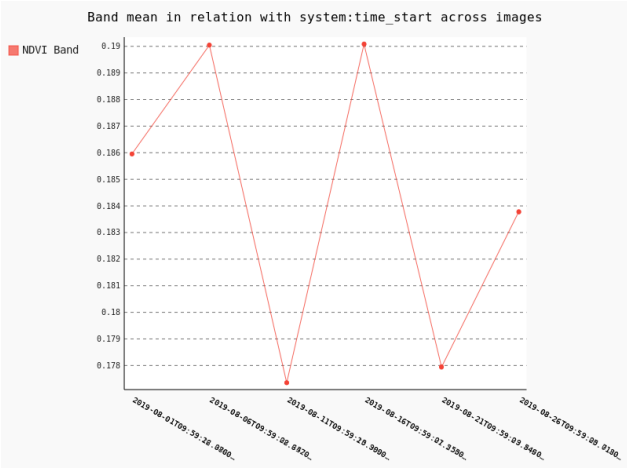


Grafico 37: Campo non irrigato 1 - Agosto 2019

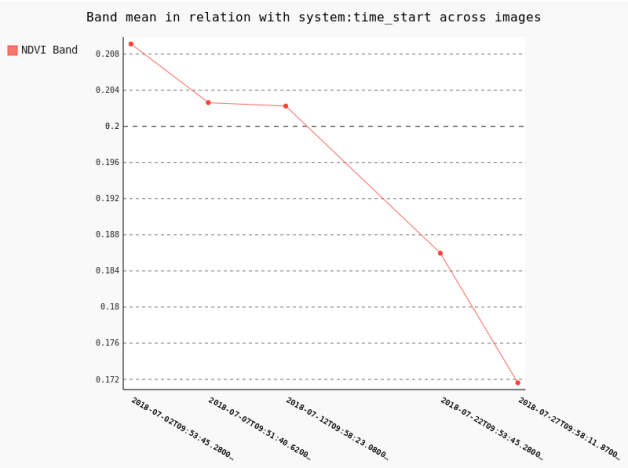


Grafico 18: Campo non irrigato 1 - Luglio 2018

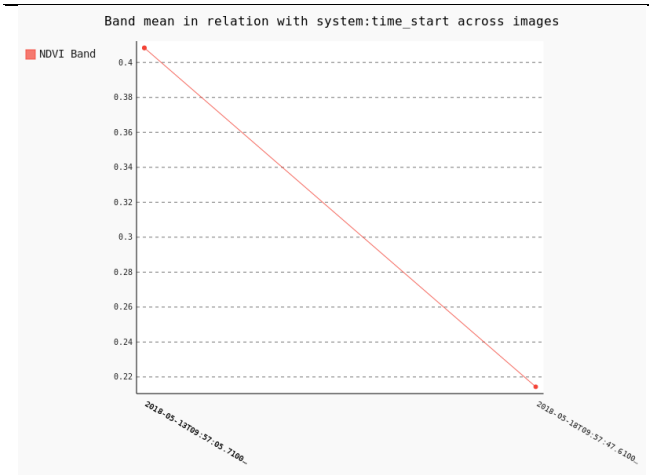


Grafico 19: Campo non irrigato 1 - Maggio 2018

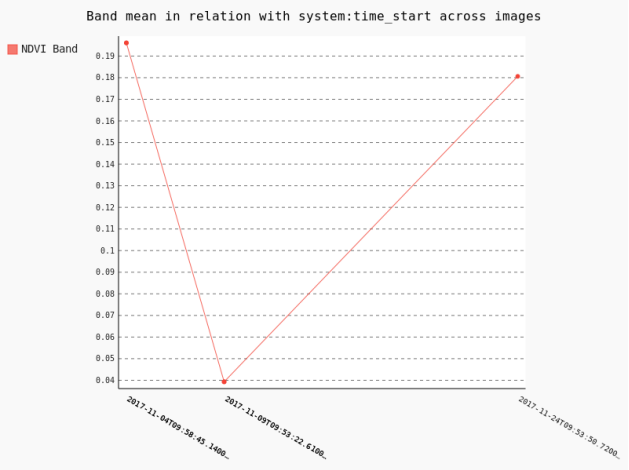


Grafico 20: Campo non irrigato 1 - Novembre 2017

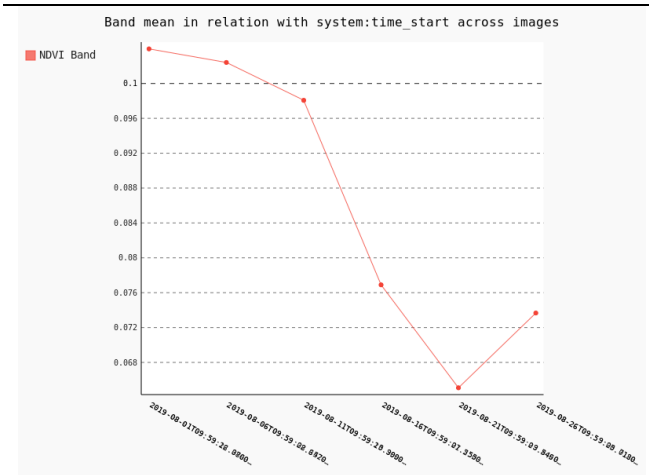


Grafico 21: Campo non irrigato 2 - Agosto 2019

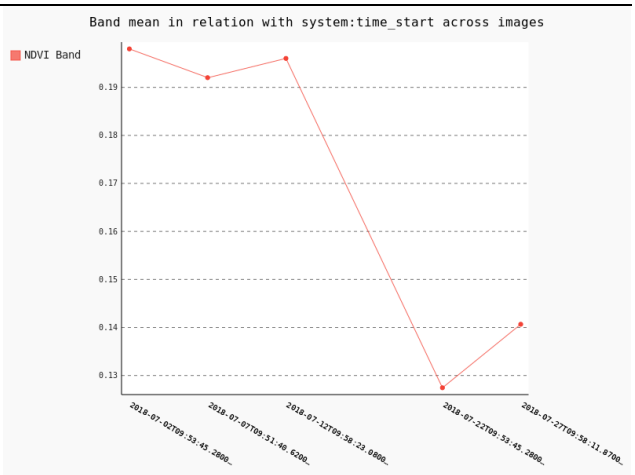


Grafico 22: Campo non irrigato 2 - Luglio 2018

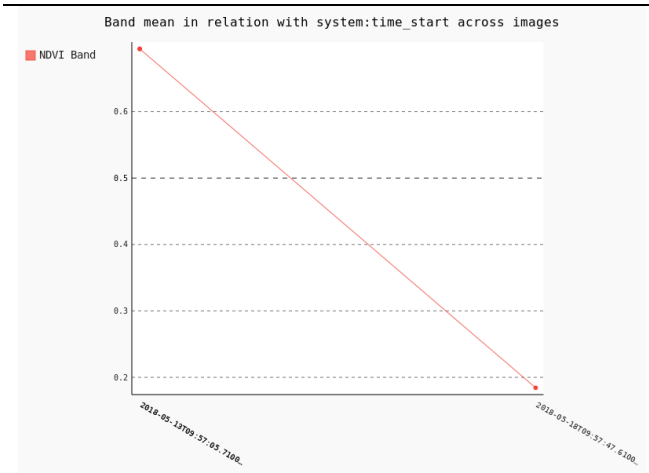


Grafico 23: Campo non irrigato 2 - Maggio 2018

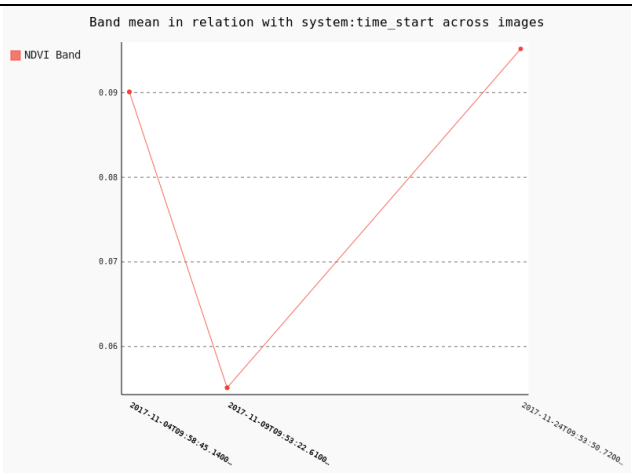


Grafico 24: Campo non irrigato 2 - Novembre 2017

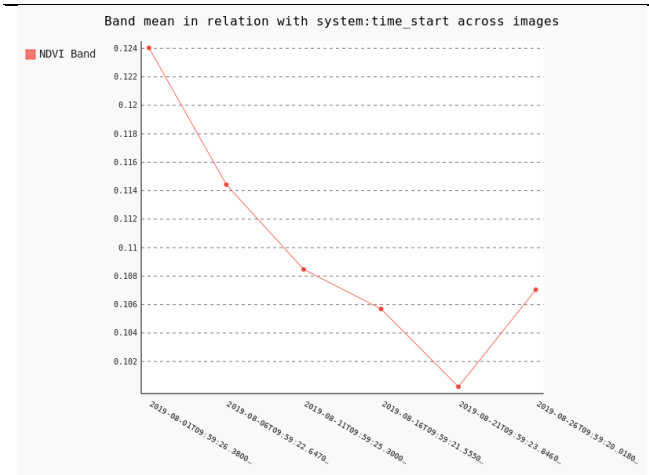


Grafico 25: Campo non irrigato 3 - Agosto 2019

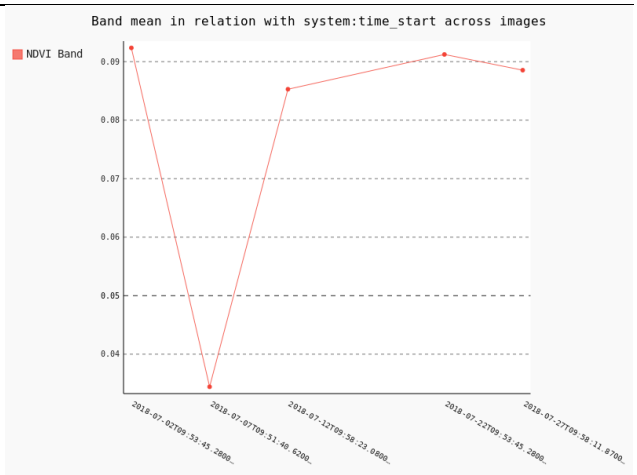


Grafico 26: Campo non irrigato 3 - Luglio 2018

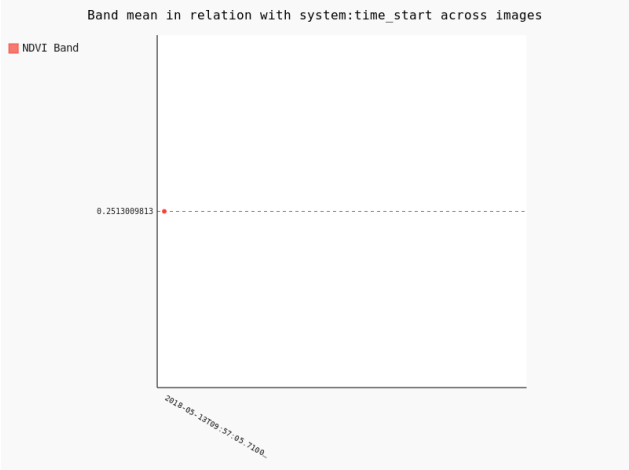


Grafico 27: Campo non irrigato 3 - Maggio 2018

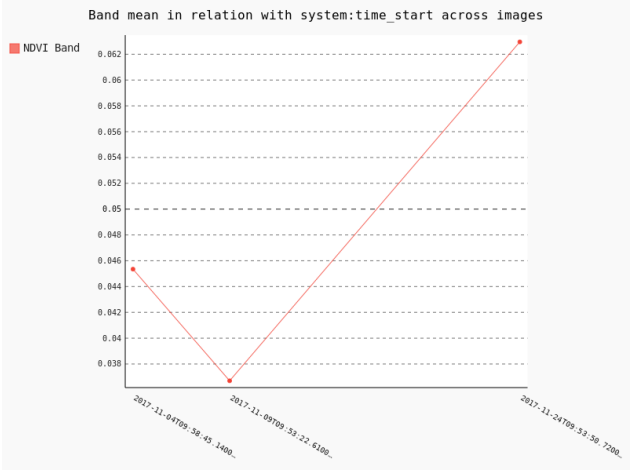


Grafico 28: Campo non irrigato 3 - Novembre 2017

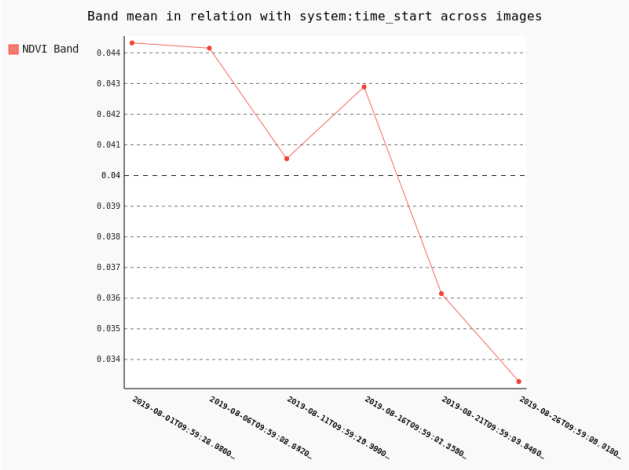


Grafico 29: Campo non irrigato 4 - Agosto 2019

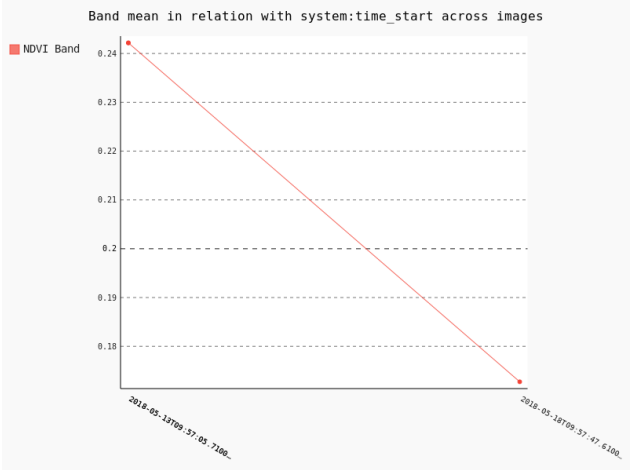


Grafico 30: Campo non irrigato 4 - Luglio 2018

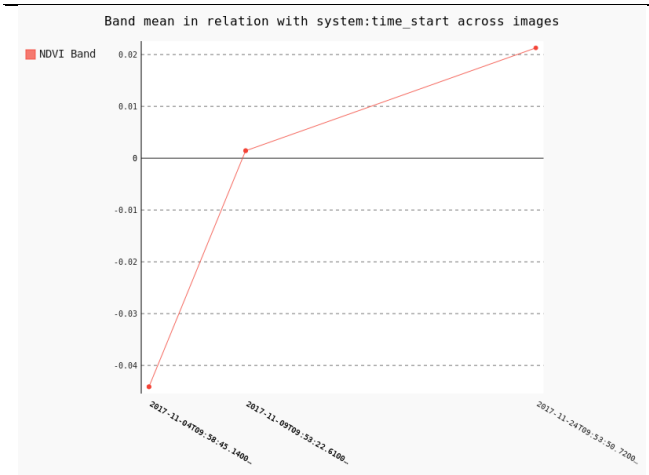


Grafico 31: Campo non irrigato 4 - Maggio 2018

Grafico 32: Campo non irrigato 4 - Novembre 2017