A Web Service of Time Series of Earth Observation Data

- Alber Sánchez *
- Gilberto Ribeiro
- Lubia Vinhas
- Rolf Simoes
- Vitor Gomes

Agenda

- e-sensing project
- Web Time Series Service
- Python integration

Before we start...

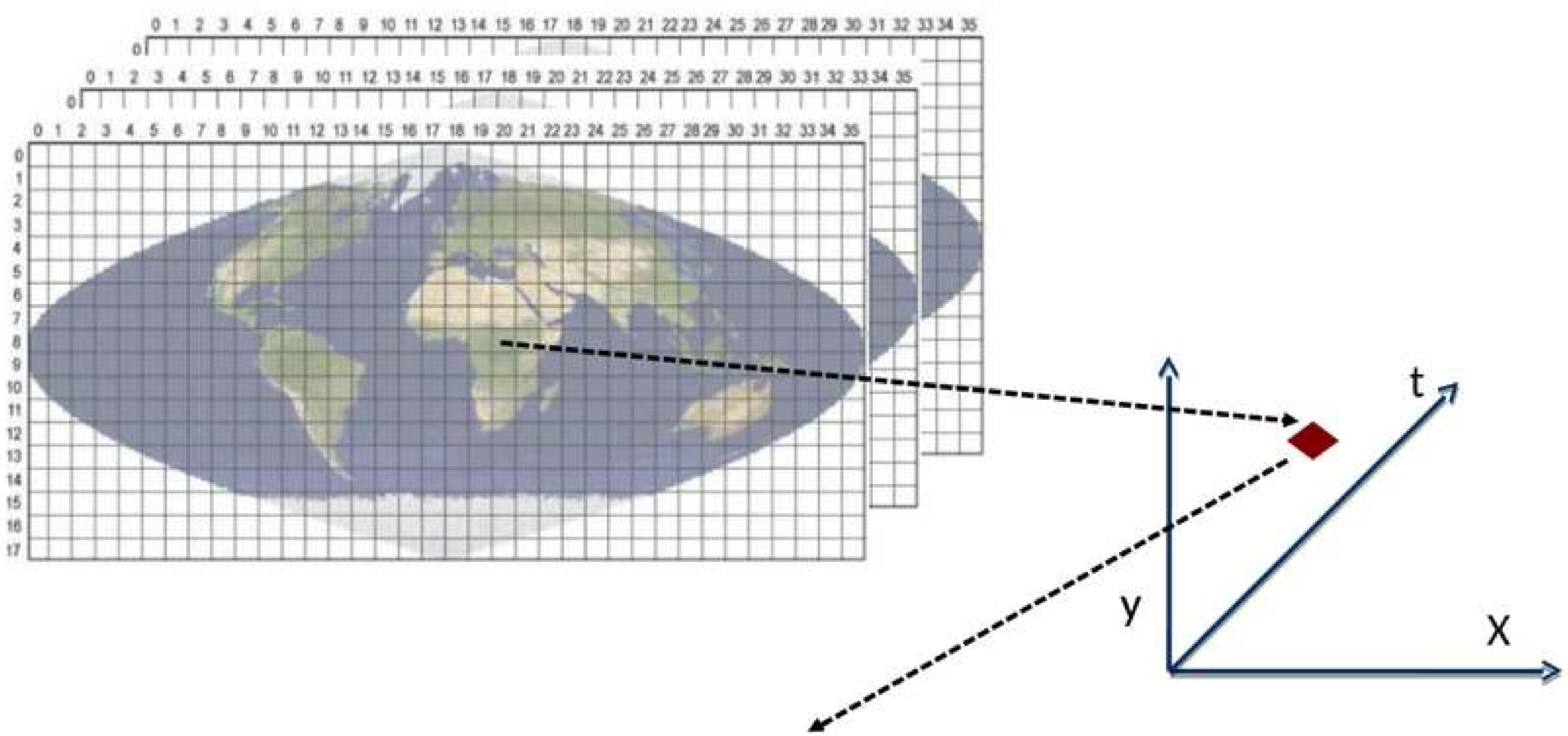
- The code in this presentation is available as a Jupyter notebook
  - https://github.com/e-sensing/wgiss-py-webinar
- Jupyter notebooks: Interactive web pages for sharing text and code
  - https://jupyter.org/

e-sensing project

- Build a platform for handling big geospatial data
- Organize decades of satellite images into arrays
- Put together data and analysis
- For more information http://esensing.org/

e-sensing data array

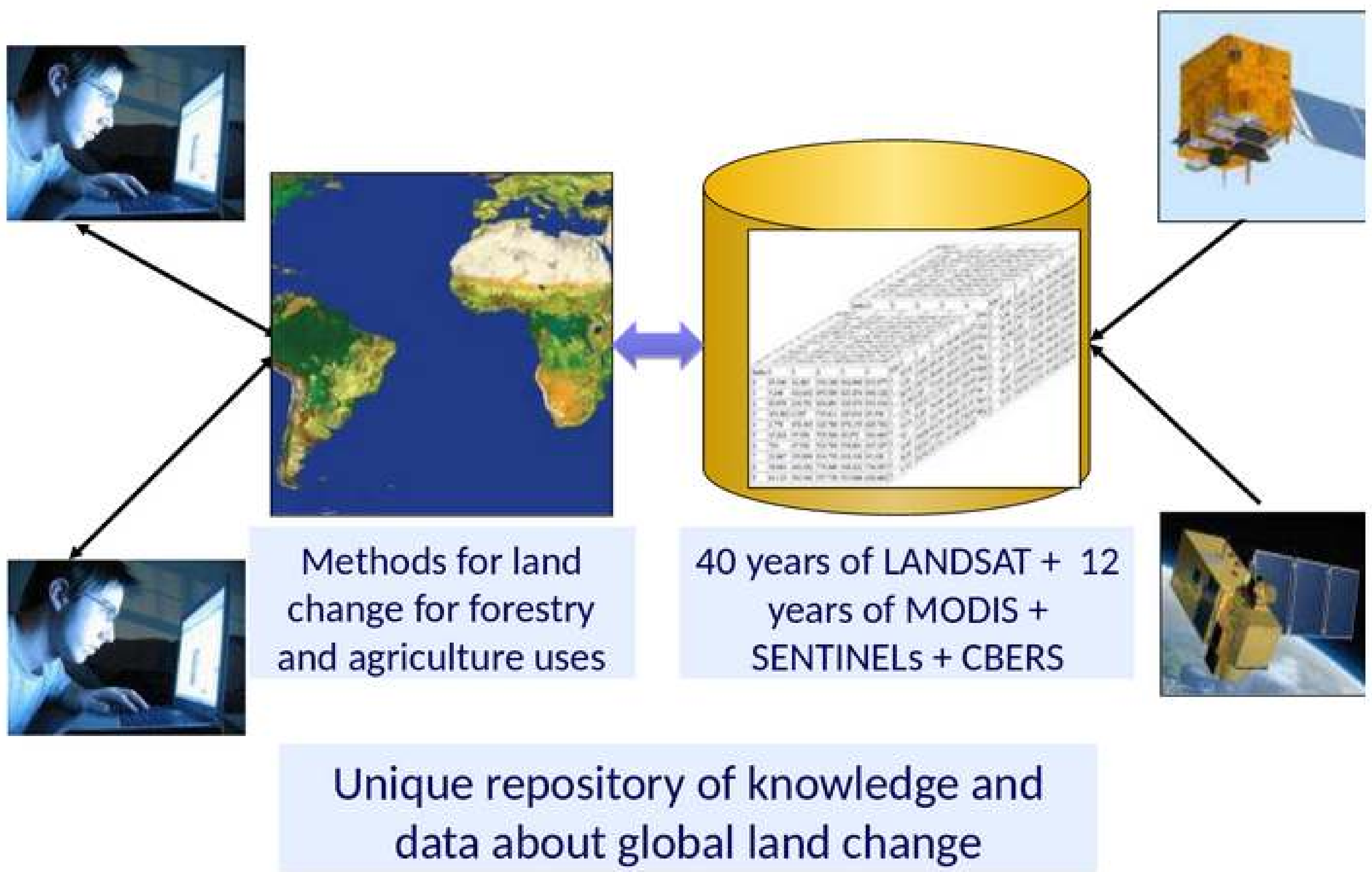
Decades of satellite images can be organized into array data structures which can be efficiently queried and processed



result = analysis_function (points in space-time)

e-sensing architecture

We put together data and analysis in order to help scientists to research land use and land cover change.



Web Time Series Service

- Lightweight JSON web service
- Access remote sensing imagery
- WTSS lists, describes, and retrieves EO time series
- For more information https://github.com/e-sensing/wtss.py

WTSS - Python integration

- Python client of WTSS
- All of our examples use WTSS Python
- More information https://github.com/e-sensing/wtss.py

WTSS: List coverages

This operation gets a list of the data sets hosted in a WTSS server.

In the following example, we import the WTSS module and then create a WTSS object to query and print the list of available data sets in the server.

In [29]:

```python
# WTSS python client: Access to data & metadata
from wtss import wtss

# connect to e-Sensing server
w = wtss("http://www.dpi.inpe.br/tws")

# print the available data sets
cv_list = w.list_coverages()
for cv_name in cv_list["coverages"]:
    print(cv_name)
```

```
itobi
merge
mixl8mod
mixl8mod_f
mod13q1_512
```

WTSS: Describe coverage

This operations enables users to explore the details of a data set in the WTSS server.

In the following example, we ask WTSS for the details of a coverage. Then we format the WTSS's reponse.

In [30]:
```
# explore a WTSS data set
cv_scheme = w.describe_coverage("mod13q1_512")

# format response
print("ARRAY: {}".format(cv_scheme["name"]) + ". " \
      + str(cv_scheme['description']) + " - " + str(cv_scheme['detail']))
print("\nTIMELINE:\n{}...{}".format(cv_scheme['timeline'][0:3], \
                                    cv_scheme['timeline'][-3:]))
print("\nATTRIBUTES:")
for el in cv_scheme['attributes']:
    att = cv_scheme['attributes'][el]
    print(el + ": " + att['description'] + ". Type: " + att['datatype'] + \
          ". Scale factor: " + str(att['scale_factor']))
```

```
ARRAY: mod13q1_512. Vegetation Indices 16-Day L3 Global 250m - https://lpdaac.usgs.gov/dataset_discovery/modis
/modis_products_table/mod13q1

TIMELINE:
[u'2000-02-18', u'2000-03-05', u'2000-03-21']...[u'2017-01-17', u'2017-02-02', u'2017-02-18']

ATTRIBUTES:
blue: 250m 16 days blue reflectance (Band 3). Type: int16. Scale factor: 0.0001
evi: 250m 16 days EVI. Type: int16. Scale factor: 0.0001
nir: 250m 16 days NIR reflectance (Band 2). Type: int16. Scale factor: 0.0001
ndvi: 250m 16 days NDVI. Type: int16. Scale factor: 0.0001
mir: 250m 16 days MIR reflectance (Band 7). Type: int16. Scale factor: 0.0001
red: 250m 16 days red reflectance (Band 1). Type: int16. Scale factor: 0.0001
```

WTSS: Time series

This operation retrieves a time series of the provided point.

In the following example, we ask WTSS for vegetation indexes, then we create *pandas* series out of them, and finally we put the series together into a *pandas* data frame.

In [31]:

```
import pandas as pd
# we are interested in observing land dynamics at
latitude  = -14.919100049
longitude = -59.11781088
# get time series of a point
ts = w.time_series("mod13q1_512", ("ndvi", "evi"), latitude, longitude)
# build a data frame made of vegetation indexes
ndvi = pd.Series(ts["ndvi"], index = ts.timeline) * \
        cv_scheme['attributes']['ndvi']['scale_factor']
evi  = pd.Series(ts["evi"],  index = ts.timeline) * \
        cv_scheme['attributes']['evi']['scale_factor']
vidf = pd.DataFrame({'ndvi': ndvi, 'evi': evi})
vidf[0:5]
```

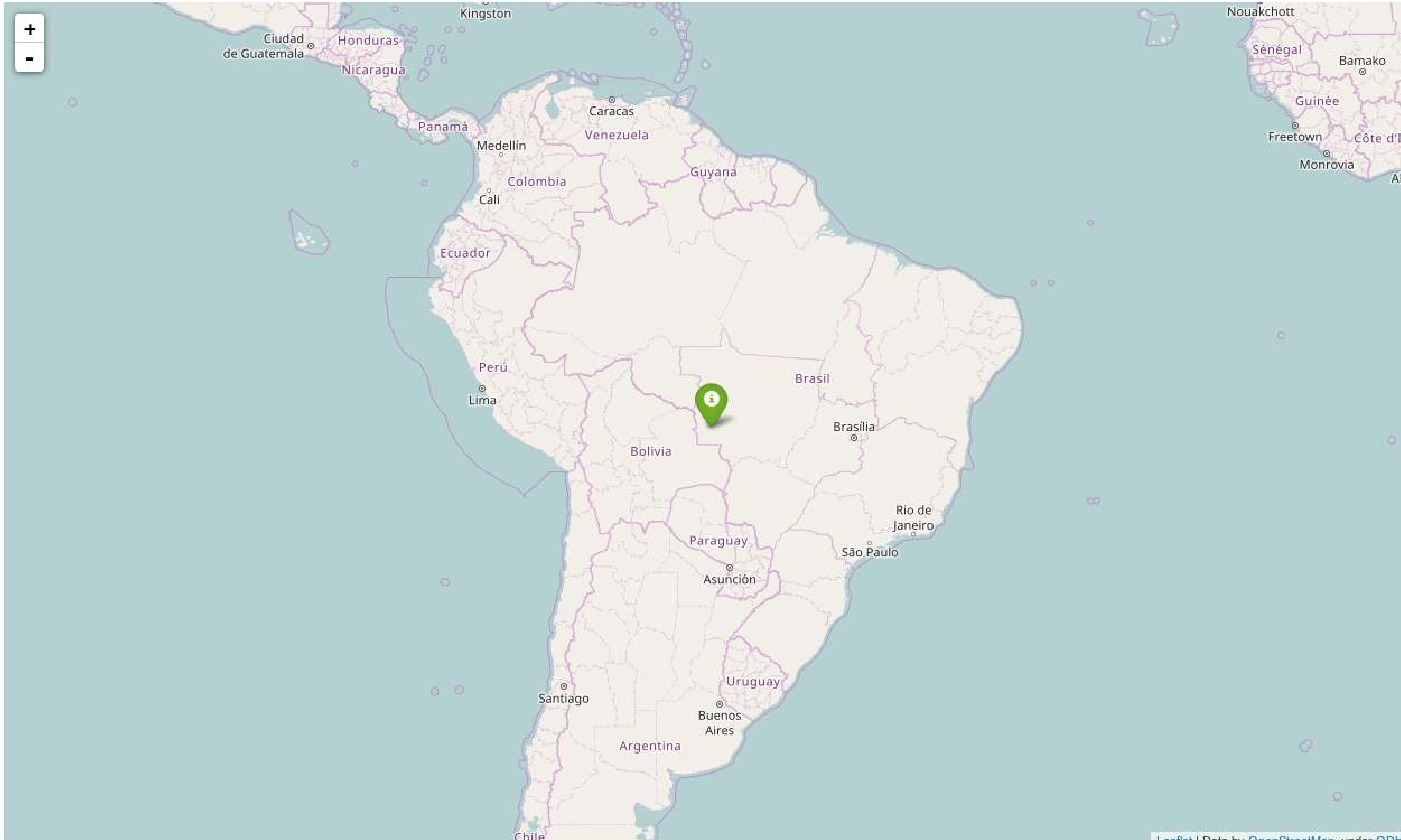
Out[31]:

	evi	ndvi
2000-02-18	0.6439	0.7418
2000-03-05	0.4600	0.9092
2000-03-21	0.5516	0.9025
2000-04-06	0.4937	0.8850
2000-04-22	0.5220	0.8578

In [43]:

```python
# Let's show the choosen location on a map
from tsmap import *
location = {'lon': longitude, 'lat': latitude}
createTSMap(location, vidf, 4)
```

Out[43]:



WTSS and Python Data Analysis Library

Data Visualization

Python provides tools for scientific data visualization.

In our next example, we take advantage of the integration between *pandas* and *matplotlib* in order to plot our vegetation indexes.

In [33]:

```
%matplotlib inline
import matplotlib
from cycler import cycler
matplotlib.style.use('ggplot')
# Updating default matplotlib colors
colors = cycler(u'color', [u'#74c476',u'#6baed6',u'#d62728', \
                           u'#ff7f0e', u'#756bb1'])
matplotlib.rcParams['axes.prop_cycle'] = colors
# Time series visualization
fig, ax = matplotlib.pyplot.subplots(figsize = (15, 5))
ax.plot()
vidf['ndvi'].plot()
vidf['evi'].plot()
ax.legend()
fig.autofmt_xdate()
```

Data analysis

Line fitting

A simple way to reveal coarse trends in time series is to adjust a straight line through the data.

In the code below, we have a function to fit lines which we use in our time series. Then we plot the vegetation indexes along the adjusted lines.

In [34]:

```
from linearmodel import *
# fit a line to the vegetation indexes
vidf['ndvi_lm'] = fitline(vidf['ndvi'])
vidf['evi_lm'] = fitline(vidf['evi'])
# plot
fig, ax = matplotlib.pyplot.subplots(figsize = (15, 5))
ax.plot()
vidf['ndvi'].plot()
vidf['evi'].plot()
vidf['ndvi_lm'].plot()
vidf['evi_lm'].plot()
ax.legend()
fig.autofmt_xdate()
#vidf[0:5]
```

Fourier decomposition

Fourier series analysis decomposes time series into sums of periodic functions (waves). These functions have properties such as amplitude, wavelength, and frequency.

High frequencies are often associated with noise. Therefore, to diminish noise, we remove high frequencies from our time series. The use of Fourier series to estimate vegetation phenology was addresed by Atkinson [Atkinson2012].

In the following example, we use our implementation of the Fourier filter function which takes as parameter a time series and the number of low frequencies to keep.

In [35]:

```
from fourier import *
# filter the vi
vidf['ndvi_ff'] = fourierfilter(vidf['ndvi'], 50)
vidf['evi_ff'] = fourierfilter(vidf['evi'], 50)
# plot
fig, ax = matplotlib.pyplot.subplots(figsize = (15, 5))
ax.plot()
vidf['ndvi'].plot()
vidf['evi'].plot()
vidf['ndvi_ff'].plot()
vidf['evi_ff'].plot()
ax.legend()
fig.autofmt_xdate()
#vidf[0:5]
```

Whittaker smoothing

Whitakker filter is a linear combination of time series' nearest neighbors points [Eilers2003]. This filter is useful for estimating vegetation phenology[Atkinson2012].

In the following example, we use our implementation of Whittaker to smooth our sample time series.

In [36]:

```
from whittaker import *
# filter the vi
vidf['ndvi_wf'] = pd.Series(whittaker_filter(ndvi,1000), index = ts.timeline)
vidf['evi_wf']  = pd.Series(whittaker_filter(evi,1), index = ts.timeline)
# plot
fig, ax = matplotlib.pyplot.subplots(figsize = (15, 5))
ax.plot()
vidf['ndvi'].plot()
vidf['evi'].plot()
vidf['ndvi_wf'].plot()
vidf['evi_wf'].plot()
ax.legend()
fig.autofmt_xdate()
#vidf[0:5]
```

Kalman filter

The Kalman filter aims to separate time series from noise. It is an iterative algorithm on which the outputs of one iteration are the inputs for the next one. In this way, the filter successively improves its estimations of the true value of a time series.

In the example below, we estimate the initial parameters for the filter from the time series itself. Then we compute the Kalman filter and plot the smoothed vegetation indexes.

In [37]:

```
from kalman import *
# filter the vi
vidf['ndvi_kf'] = pd.Series(kalmanfilter(ndvi), index = ts.timeline)
vidf['evi_kf']  = pd.Series(kalmanfilter(evi), index = ts.timeline)
# plot
fig, ax = matplotlib.pyplot.subplots(figsize = (15, 5))
ax.plot()
vidf['ndvi'].plot()
vidf['evi'].plot()
vidf['ndvi_kf'].plot()
vidf['evi_kf'].plot()
ax.legend()
fig.autofmt_xdate()
#vidf[0:5]
```

Classification

Dynamic Time Warping

Dynamic Time Warping (DTW) is pattern matching algorithm. It relies on a shape-based distance function that sequentially warps the time dimension in order to find the best match — the minimum the distance — between two time series: The pattern and the sample series. Below we show how to classify time series using DTW.

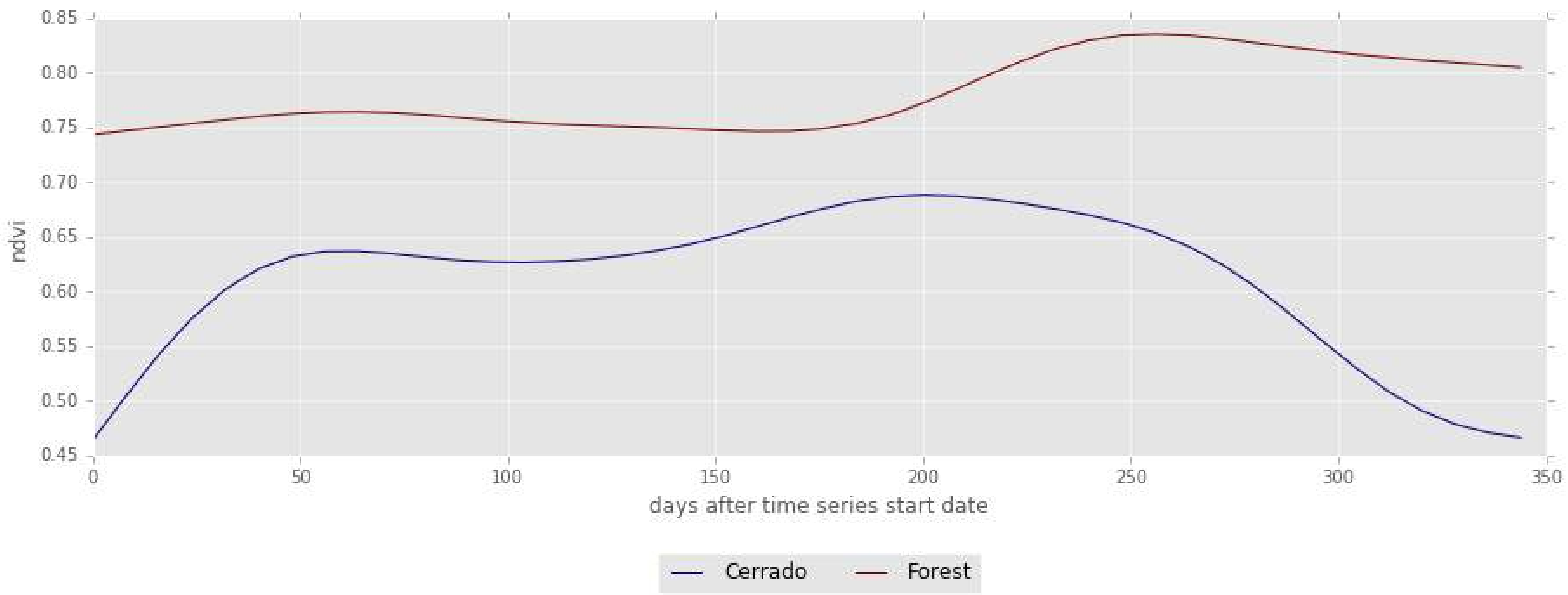
Patterns

Our patterns are idealized one-year time-series of the Forest and the Cerrado regions in Brazil. These patterns were obtained using a Generalized Additive Model over a large amount of selected time series.

Below we show how to read and plot the aforementioned patterns.

In [38]:
```
from dtw import *
from tools import *

# open the pattern file
patterns_ts = pd.read_json("examples/patterns.json", orient='records')
# update timeline type from str to datetime
patterns_ts["timeline"] = pd.to_datetime(patterns_ts["timeline"])
plot_time_series(patterns_ts)
```



Samples

We have a file of sample locations which we would like to classify using the patterns listed above. For the sake of this example, we already know that these locations belong to either the Forest or Cerrado region. These ten locations are in the Brazilian state of Mato Grosso and they were verified on the field.

Below we read the file with the sample locations.

In [39]:
```
# read sample file
samples = pd.read_csv("examples/samples.csv")
samples
```

Out[39]:

	id	longitude	latitude	start_date	end_date	label
0	0	-54.231300	-14.048200	2014-09-14	2015-08-29	Cerrado
1	1	-54.229000	-14.063200	2014-09-14	2015-08-29	Cerrado
2	2	-55.209200	-15.114600	2014-09-14	2015-08-29	Cerrado
3	3	-55.352700	-15.073900	2014-09-14	2015-08-29	Cerrado
4	4	-55.324200	-15.076000	2014-09-14	2015-08-29	Cerrado
5	5	-51.241157	-14.070312	2013-09-14	2014-08-29	Forest
6	6	-49.415758	-22.544512	2013-09-14	2014-08-29	Forest
7	7	-51.286030	-13.655227	2013-09-14	2014-08-29	Forest
8	8	-50.655128	-12.430256	2013-09-14	2014-08-29	Forest
9	9	-51.257672	-14.213212	2013-09-14	2014-08-29	Forest

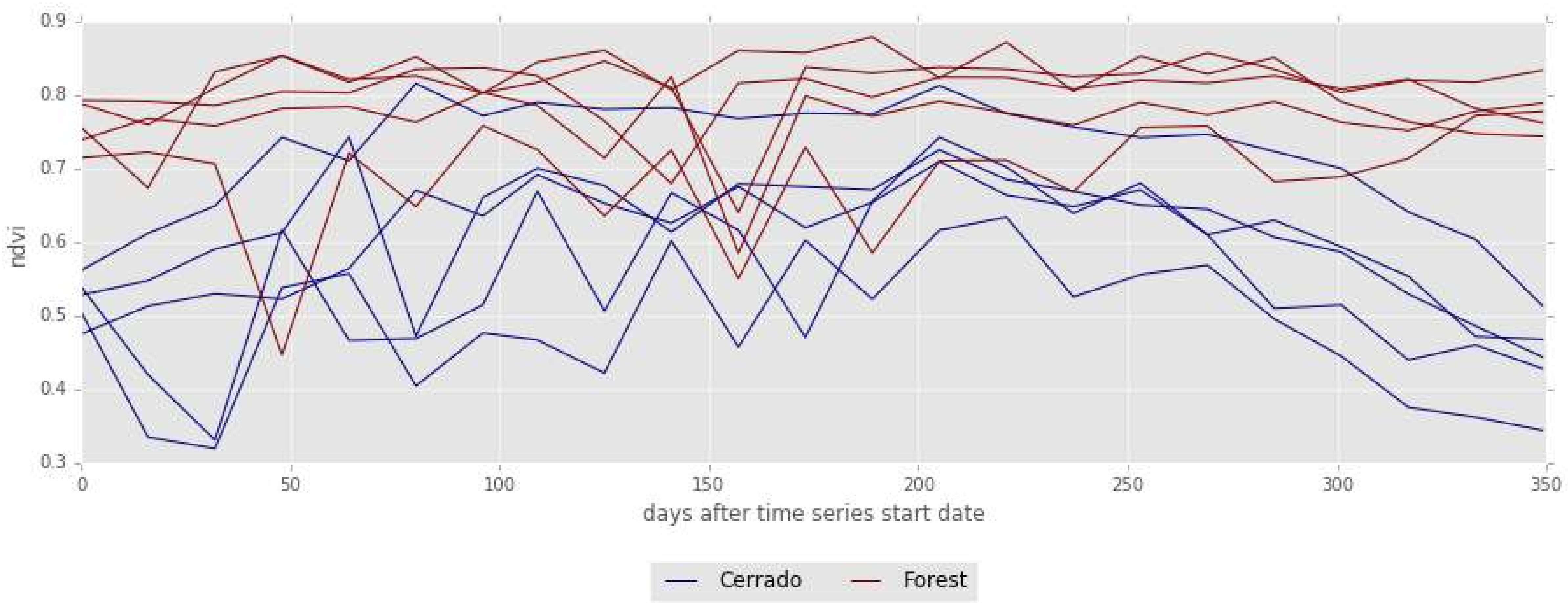
Now, we get the time series of MODIS data of these locations. We do this in the background using WTSS python client.

In [40]:

```python
# wtss_get_time_series is implemented in 'tools.py'
samples_ts = wtss_get_time_series(samples)

# rescale vegetation index to -1.0~1.0 range
samples_ts["ndvi"] *= cv_scheme['attributes']['ndvi']['scale_factor']
samples_ts["evi"] *= cv_scheme['attributes']['evi']['scale_factor']

samples_ts[0:5]
plot_time_series(samples_ts)
```



Classification

It is time to classify the sample locations using our patterns. We do this by computing the DTW distance from each pattern to each sample. Then we assign to each sample the name of the pattern with the minimum DTW distance.

To achieve this, we use the code below.

In [41]:

```
# classify using DTW
classification = classifier_1nn(patterns_ts, samples_ts)

# print the classification rersults
classification
```

Out[41]:

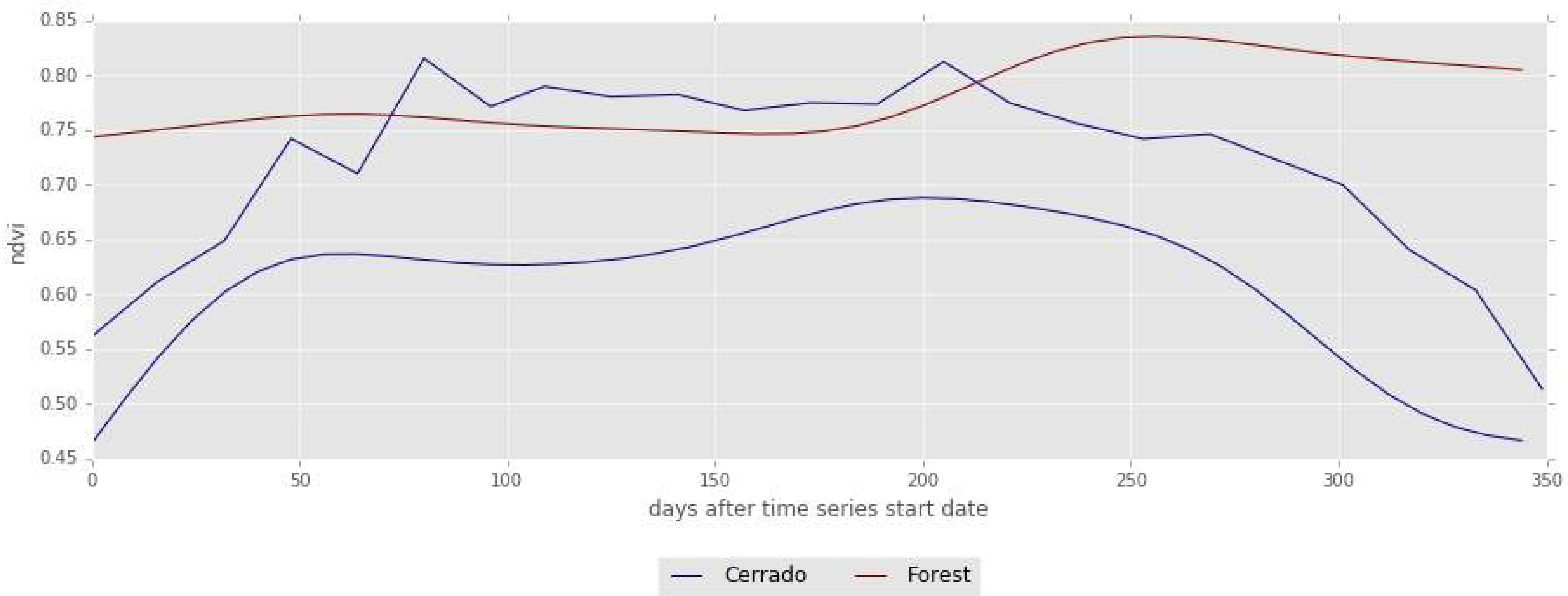
	id	reference	prediction	correct
0	0	Cerrado	Cerrado	True
1	1	Cerrado	Cerrado	True
2	2	Cerrado	Cerrado	True
3	3	Cerrado	Forest	False
4	4	Cerrado	Cerrado	True
5	5	Forest	Forest	True
6	6	Forest	Forest	True
7	7	Forest	Forest	True
8	8	Forest	Forest	True
9	9	Forest	Forest	True

Classification results

The results above prove that DTW and our patterns do a good job. We managed to correctly classify 9 out of ten time series. However, the sample location number 3 is incorrectly classified as it is Cerrado but it was assigned to the Forest label.

To find what happened, we plot the Forest and Cerrado patterns along the time series of the sample location number three. We can see there how this sample doesn't fit very well to either of the two patterns.

In [42]:
```
# let see what happened with sample #3
plot_time_series(pd.concat([samples_ts[samples_ts["id"].isin([3])], patterns_ts]))
```



Final remarks

We introduced the Web Time Series Service (WTSS), a light weight Web Service of time series of Earth observation data. Through examples and code, we show how the WTSS is used and integrated to Python's scientific libraries such as NumPy, SciPy and Pandas. Therefore, we demonstrated how WTSS fits into the analytic work flow of Earth Observation Scientists.

References

[Atkinson2012]: P. M. Atkinson, C. Jeganathan, J. Dash, and C. Atzberger, “Inter-comparison of four models for smoothing satellite sensor time-series data to estimate vegetation phenology,” Remote Sens. Environ., vol. 123, pp. 400–417, Aug. 2012.

[Eilers2003]: Paul H. C. Eilers. "A Perfect Smoother". Analytical Chemistry, 2003, 75 (14), pp 3631–3636.

[Vinhas2016]: L. Vinhas; G. R. Queiroz; K. R. Ferreira; Camara, G. [Web Services for Big Earth Observation Data](). In: BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 17. (GEOINFO), 2016, Campos do Jordão, SP. Proceedings... 2016.

Obrigado!

https://github.com/e-sensing/wgiss-py-webinar

Alber Sánchez
alber.ipia @ inpe.br