ECMWF Summer of Weather Code 2021

Challenge #24: CliMetLab - Machine Learning on weather and climate data

Stream: 2 (Machine Learning for weather, climate and atmosphere applications)
Mentors: Baudouin Raoult, Florian Pinault

Proposal by: Ashwin Samudre
GitHub: @kiryteo

Contact Information:

Name: Ashwin Samudre
Email(s): ashwin.samudre@gmail.com, capnoi8@gmail.com
Twitter: @samudre_ashwin
Personal page: https://kiryteo.github.io/

Project Summary

CliMetLab is a Python package aiming at simplifying access to climate and meteorological datasets, allowing users to focus on science instead of technical issues such as data access and data formats. It is mostly intended to be used in Jupyter notebooks, and be interoperable with all popular data analytic packages, such as NumPy, Pandas, Xarray, SciPy, Matplotlib, etc. and well as Machine Learning frameworks, such as TensorFlow, Keras or PyTorch.
CliMetLab aims at handling the data loading as well as interpreting the output from the machine learning models with the use of plots, graphs, etc. This will remove the overhead of manual data retrieval, writing specific data loaders per dataset. The plugin architecture in CliMetLab aims at easy addition of a) data sources, b) datasets c) plotting styles, d) data formats [1].

Project goals

Extend new Python ML package and help to mature package. Specific goals:

- Task 1: extend CliMetLab so that it offers the user with high-level Matplotlib-based plotting functions to produce graphs and plot which are relevant to weather and climate applications (e.g. plumes plots, ROC curves, ...).
- Task 2: the Python package Intake is a lightweight set of tools for loading and sharing data in data science projects. Extend CliMetLab so that it seamlessly interfaces with Intake and allows users to access all intake-compatible datasets.
- Task 3: Xarray uses the data format Zarr to allow parallel read and parallel write. Convert large already available datasets to xarray-readable zarr format, define appropriate configuration (chunking/compression/other) according to domain use cases, develop tools to benchmark when used on a cloud-platform, compare to other formats (N5, GRIB, netCDF, geoTIFF, etc.).

CliMetLab is at an early stage in development. This project aims at adding the data handling functionality along with benchmarking the data storage and access formats.

Implementation details

Task 1:

Magics is currently the go-to package as part of the ECMWF suite of softwares for plotting.
With Magics as the backend driver in CliMetLab, plotting is performed with plot_map() functionality currently. This directs the call towards macro subpackage.

Currently, climetlab activates the plot_map as follows:

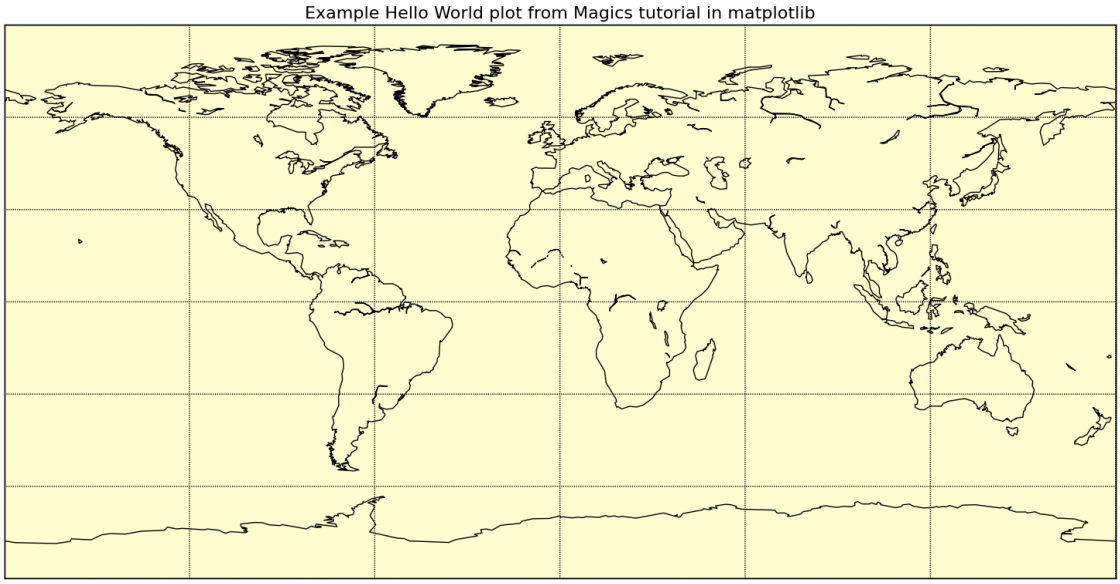
```
Plotting -> drivers -> magics -> actions -> plot(): return macro.plot()

netcdf.py -> plot_map() -> driver.plot_map()
grib.py -> plot_map() -> driver.plot_grib()
```

The idea would be to add matplotlib as another driver, thus making the possibility of call to matplotlib plot() function.

Demo for the plotting with matplotlib

a) Hello World plot

```
from mpl_toolkits.basemap import Basemap
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8, 6), edgecolor='b')
m = Basemap(projection='cyl', resolution='c', llcrnrlat=-90, urcrnrlat=90,
llcrnrlon=-180, urcrnrlon=180, )

m.drawcoastlines()
m.drawparallels(np.arange(-90., 91., 30.))
m.drawmeridians(np.arange(-180., 181., 60.))
m.drawmapboundary(fill_color=(1, 253/255, 208/255))

plt.title("Example Hello World plot from Magics tutorial in matplotlib", size=16)
plt.show()
```

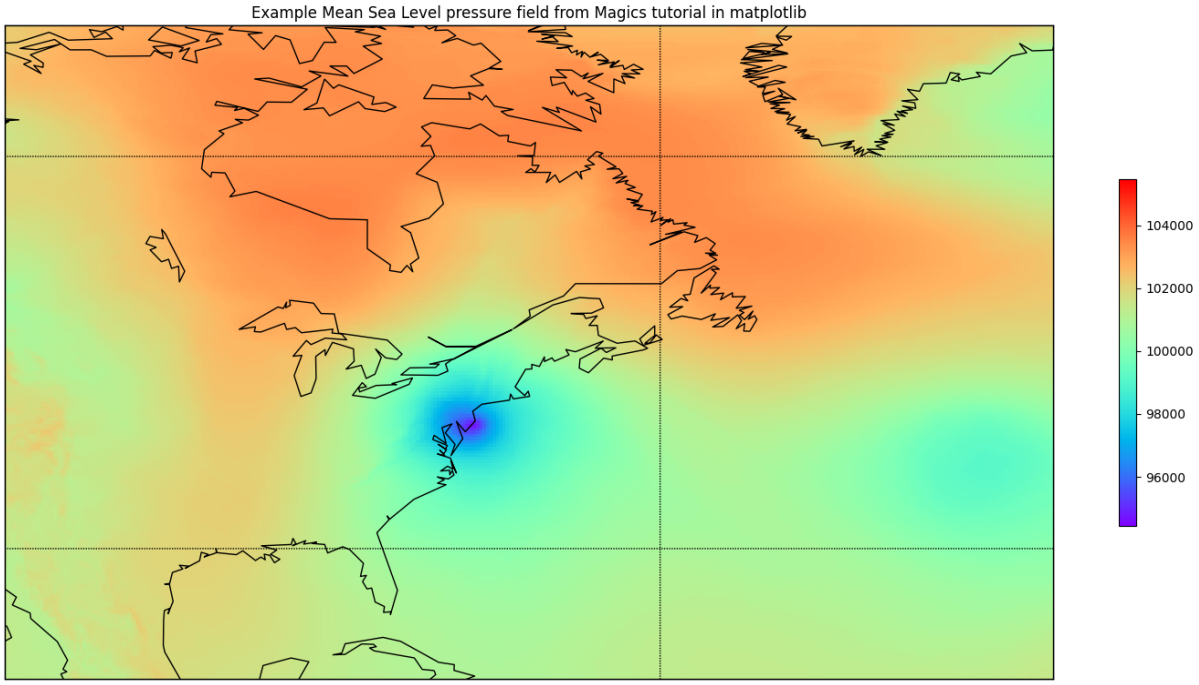


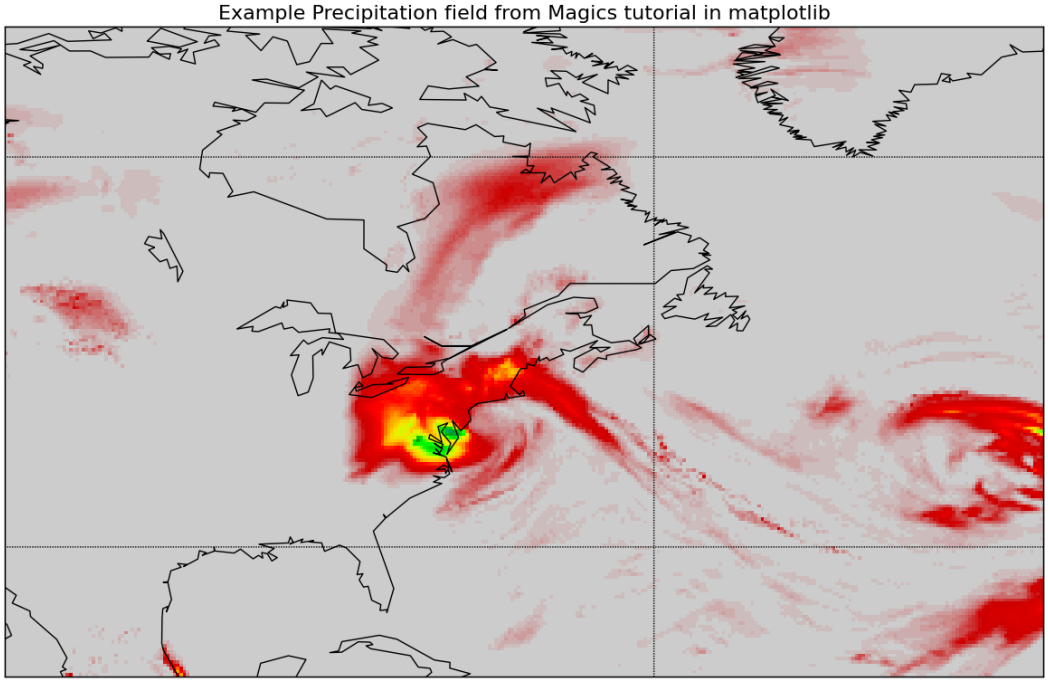
b) Geographical Sea Level pressure plot (The following code provides plotting for both Sea Level pressure plot and precipitation level plot)

```
import xarray as xr
import pygrib
from mpl_toolkits.basemap import Basemap
from mpl_toolkits.basemap import shiftgrid
import numpy as np
from numpy import linspace
from numpy import meshgrid
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8, 6), edgecolor='b')

grib_msl = pygrib.open('msl.grib')
grib_prec = pygrib.open('precip.grib')

grib_msl_data = grib_msl.select()[0]
data_msl = grib_msl_data.values

grib_prec_data = grib_prec.select()[0]
data_prec = grib_prec_data.values
```

```
lon_msl = np.linspace(float(grib_msl_data['longitudeOfFirstGridPointInDegrees']), \
float(grib_msl_data['longitudeOfLastGridPointInDegrees']), int(grib_msl_data['Ni']) )
lat_msl = np.linspace(float(grib_msl_data['latitudeOfFirstGridPointInDegrees']), \
float(grib_msl_data['latitudeOfLastGridPointInDegrees']), int(grib_msl_data['Nj']) )
data_msl, lon_msl = shiftgrid(180., data_msl, lon_msl, start=False)
grid_lon_msl, grid_lat_msl = np.meshgrid(lon_msl, lat_msl)

lon_prec = np.linspace(float(grib_prec_data['longitudeOfFirstGridPointInDegrees']), \
float(grib_prec_data['longitudeOfLastGridPointInDegrees']), int(grib_prec_data['Ni'])
)
lat_prec = np.linspace(float(grib_prec_data['latitudeOfFirstGridPointInDegrees']), \
float(grib_prec_data['latitudeOfLastGridPointInDegrees']), int(grib_prec_data['Nj'])
)
data_prec, lon_prec = shiftgrid(180., data_prec, lon_prec, start=False)
       grid_lon_prec, grid_lat_prec = np.meshgrid(lon_prec, lat_prec)

m = Basemap(projection='cyl', resolution='c', llcrnrlat=20, urcrnrlat=70,
                    llcrnrlon=-110, urcrnrlon=-30, )

x_msl, y_msl = m(grid_lon_msl, grid_lat_msl)
x_prec, y_prec = m(grid_lon_prec, grid_lat_prec)

cs_msl = m.pcolormesh(x_msl, y_msl, data_msl, shading='flat', cmap=plt.cm.rainbow)
cs_prec = m.pcolormesh(x_prec, y_prec, data_prec, shading='auto',
cmap=plt.cm.nipy_spectral_r)

m.drawcoastlines()
m.drawparallels(np.arange(-90.,91.,30.))
m.drawmeridians(np.arange(-180.,181.,60.))
m.drawmapboundary(fill_color=(1, 253/255, 208/255))

plt.colorbar(cs_msl, orientation='vertical', shrink=0.5)
plt.colorbar(cs_prec, orientation='vertical', shrink=0.5)

plt.title("Example Mean Sea Level pressure field from Magics tutorial in matplotlib",
size=16)

plt.show()
```

b) Precipitation level plot


Example Mean Sea Level pressure field from Magics tutorial in matplotlib
104000
102000
100000
98000
96000


Example Precipitation field from Magics tutorial in matplotlib
0.10
0.08
0.06
0.04
0.02
0.00

While these are the plots obtained via matplotlib, it would be used primarily for machine learning related graphs and curves like RoC curve, plume plots, training and loss curves, etc. This provides CliMetLab users the ability to use either of the Magics and matplotlib libraries based on the graph/ curve to be plotted. As Magics is a more mature library w.r.t. plotting geographical data graphs, the user can opt for Magics or Matplotlib based on preference. In the short-term, Matplotlib would be integrated into CliMetLab for plotting functionality from ML point of view with use in Jupyter Notebooks. In the long-term, with further development, Matplotlib can be compatible with majority of the features from Magics providing users the ability to use either option for all types of curve plotting.

Possible extension:
Provide support for Seaborn, Bokeh (minimal functionality).

Task 2:

Intake is a lightweight set of tools for loading and sharing data. It provides the ability to load data from a variety of formats [2], while allowing the datasets to be described in catalog files. In the context of this project, it provides the **intake-xarray** format useful for loading the datasets.

In order to provide users the functionality of Intake, it would be integrated into the climetlab as a source plugin.

```
cml.load_data('...')  -> call intake.open_*() [data-dependent]
Returns the output container.
```

Container types can be inferred from the 'container' attribute of the data source.

As the outcome, we would test and release the 'climetlab-intake' package as a source plugin for CliMetLab.

Task 3:

For small scale datasets, the conversion between different formats is trivial with the current available APIs. Following are the 2 examples to show the conversion:
1] NetCDF format into Zarr format

```
ds = xr.open_dataset("ECMWF_ERA-40_subset.nc")
ds.to_zarr("temporary_zarr_ds.zarr") # for nc to zarr
```

2] Grib format into Zarr format

```
ds_grib = xr.load_dataset("era5-levels-members.grib", engine='cfgrib')
ds_grib.to_zarr("temporary_grib_to_zarr.zarr") # for grib to zarr
```

In case of large datasets, utilizing the Zarr's parallelism could be useful Possible. Comparing the different data formats is essential. In this project, we aim to compare Zarr, NetCDF and GRIB data formats.

NetCDF is developed and maintained by Unidata and is widely used in areas such as climatology, meteorology, oceanography and GIS, for weather forecasting and climate change research. It is the most common format for the archives of Climate science datasets. It has offered storage formats for files based on netcdf binary format and HDF5 format. While this provided effective disk storage in the past, it does not work well with cloud storage. In order to read a single variable or metadata, one has to read the whole file due to the opaque nature of these files.

Legacy file formats such as GRIB and NETCDF are ill-equipped at handling concurrent data access, and actually burden the system CPU, memory, and disk when accessed by multiple asynchronous processes [3].

Multiple communities developing Climate data storage options are moving from NetCDF to Zarr - Pangeo [4], Unidata [9].

Zarr is both a data format and an open source Python package delivering data access functions for chunked, compressed, multi-dimensional arrays. Zarr arrays can be saved as a key-value store on object storage systems and as a hierarchical directory store on POSIX file systems.

The method for storage of data plays an important role in performance. The issues involved with data storage are as follows:

1. Multithreaded, parallel read and write
2. Compression and CPU time required to uncompress/ compress.
3. Performance on the cloud storage (AWS, GCP, other).
4. Avoid swapping via chunking strategies.

Zarr stores each chunk of dataset as a separate object in the cloud storage and thus making it easier to access the data in parallel by the CPUs. It allows the metadata to be stored in one place thus the need of a single read operation.

Chunking helps to specify the N-dimensional shape that best fits the access pattern.

In a recent study [5] by the Pangeo community, they performed strong and weak scaling studies. The weak scaling study involves read/write throughput as the number of nodes varied for a fixed dataset size per processor. The scaled read/write throughput is calculated as the total dataset size divided by the total run time. The strong scaling study involves read throughput (no write) as the number of nodes varied for a fixed total dataset size. This study provided the upper limit on the scaling of object storage for the two different data formats by measuring the total read time for each configuration of the study.

Their weak scaling analysis showed that write rates with Zarr are 9 times better than those for NetCDF on POSIX systems, since Zarr's chunked data format with Dask improves parallelism and obtains the optimal scaled write speedup. Their strong scaling analysis indicated that read throughput of NetCDF and Zarr on S3 are identical. This provides a reassurance for geoscientists that the read performance will remain intact, if not better, if they switch from NetCDF to Zarr.

As mentioned previously, Grib format does not adapt well with concurrent data access. In this project, we would test the sequential as well as the asynchronous operations on the large datasets performed via Zarr as well GRIB and compare the performance.

The chunk size also plays an important role. In the study [5], they performed the experiments with chunk sizes ranging from 64MB to 786MB. In their weak scaling

experiments, the optimal performance with Zarr was obtained with a chunk size between 384MB and 768MB, and scaling with these chunk sizes is better than with other smaller chunk sizes. Large chunk sizes reduce the total number of chunks leading to less communication overhead but larger memory usage. For their strong scaling study, the chunk size of 192MB was found to be optimal.

In this project, we would experiment with the different chunk sizes for multiple datasets.

Data compression is another factor that plays a role in data access. Zarr provides a set of data compressors [6]. In this project, we plan to try the different compression techniques and benchmark the performance as different compression techniques can show different and useful results based on the data.

As Zarr and N5 implement similar data handling strategies, and with the two data formats planned to be unified for maximum benefit of the functionality, it provides another advantage to be adopted as the data format in CliMetLab.

Another alternative to Zarr is the Parquet format. Zarr is an array-oriented format, and can be chunked on any dimension. In order to read a single point, one only need the metadata and the one chunk which contains it - which needs to be uncompressed in this case. The indexing of the data chunks is implicit.

Parquet is a column-oriented format. To find a specific point, we can ignore some chunks based in the min/max column metadata for each chunk, depending on how the coordinate columns are organised, and then load the column chunk for the random data and decompress. Custom logic is required to select chunks for loading on multiple columns simultaneously. The metadata for parquet is much larger than for zarr. In the case of multiple variables or more coordinates, this might become an extra issue for parquet.

Importantly, Parquet does not offer functionalities for N-dimensional arrays. In a recent study [7], the comparison between Zarr and Parquet was performed, keeping CSV as the baseline. Though their results showed superior performance with Parquet, they did not opt for Parquet for data storage attributing to the issue of flexibility. Zarr is a more flexible format than Parquet. Zarr allows for chunking

along any dimension. Parquet is a columnar data format and allows for chunking only along one dimension. Additionally, Xarray's interface to Zarr makes it very easy to append to a Zarr store. Zarr's chunk functionality provides reliable use in case of data transfer. As mentioned in the same study, in case of connection dropping issues, one can just pick up the last chunk of data already written and continue the retrieval process with Zarr's append option.

The work by Pangeo community [8] would be the reference for benchmarking our experiments.

Project Timeline

Pre coding period	● Sync with the mentors to discuss the key elements and objectives of the project. ● Refine project proposal for final implementation. ● Create a GitHub repository for the project with initial files.
May 3 - May 26	● Discuss and Finalize the libraries required with matplotlib for plotting (cartopy, basemap, etc.) ● Implement matplotlib driver backend alongside Magics in CliMetLab.
May 27 - May 31	● Implement tests and integrate the driver in CliMetLab ● Write scripts for testing, address bugs.
Jun 1 - Jun 5	● Read through the Intake API and the internals of Intake. ● Finalize the terminology with respect to terms such as data source, datasets, container from Intake to be used in climetlab-intake. ● Discuss source plugin API with mentors.
Jun 6 - Jun 29	● Prepare the map to develop 'climatelab-intake' source plugin based on the discussions. ● Implement 'climatelab-intake' modules.
Jun 30 - Jul 10	● Write tests for developed modules, address any bugs. ● Study the relevant modules in Xarray.
Jul 11 - Aug 5	● Investigate chunking strategies specific to datasets. ● Experiment with different dataset sizes w.r.t. Zarr performance. ● Investigate different compression and chunk size options for available datasets.

Aug 6 - Aug 31	• Investigate Zarr Vs. Parquet for data access and storage. Look into N5 and the Zarr, N5 intersection - Z5 format. • Define the criterion for benchmarking after discussion with mentors. • Documentation. • Buffer for unpredictable delay.

About me

I am currently a research assistant at Simon Fraser University, working in Medical Image Analysis group. Previously, I was a visiting fellow at EMBL Heidelberg, Germany in the Kreshuk group. Apart from working in an open source environment, I have a strong experience in open source programs, having successfully completed European Summer of Weather Code 2020 (project), Google Summer of Code 2018 with CERN-HSF (project) and Season of KDE 2018 with Kdenlive (project).

References:

[1] Baudouin Raoult, Enabling machine learning for weather and climate data in the cloud, Virtual workshop: Weather and climate in the cloud. Retrieved from: https://events.ecmwf.int/event/211/contributions/1876/attachments/972/1700/Cloud-WS_Raoult.pdf

[2] https://intake.readthedocs.io/en/latest/plugin-directory.html

[3] https://ciresevents.colorado.edu/rendezvous/getAbstract.php?posterid=50&id=15&ml=1

[4] https://ui.adsabs.harvard.edu/abs/2018AGUFMIN33A..06A/abstract

[5] https://doi.org/10.31223/X5ZW2T

[6] https://zarr.readthedocs.io/en/stable/tutorial.html#compressors

[7] https://waterdata.usgs.gov/blog/cloud_data/
[8] https://github.com/NCAR/benchmarking
[9] https://www.unidata.ucar.edu/blogs/news/entry/netcdf-and-native-cloud-storage