# Background — Weeks 1 and 2

*I have allocated ~2 weeks for the background. The first week full time and second week as you start work on your own projects (multitasking!). Focus on terms and concepts rather than equations and derivations. We will discuss stuff on a daily basis together for the first two weeks at* ***10 am every day for 1 hours*** *after you have read stuff. You will need access to google earth engine via your own google gmail account asap (*[*https://developers.google.com/earth-engine/guides/getstarted*](https://developers.google.com/earth-engine/guides/getstarted)*) and later installing anaconda on your machine.*

Otherwise, the schedule is:

1. Satellite Remote Sensing – day 1-2
2. Google Earth Engine (GEE) – day 2-3
3. Statistical Learning – day 3-6
4. Machine Learning – day 7-9

## Topic 1: Satellite Remote Sensing (SRS)

*We only work with Sentinel 2 multispectral imagery so you don’t need to learn about other sensors like radar imagers, drones etc. for this project.*

**Basics**: You will need some basics about terms used in remote sensing of vegetation (and in Google Earth Engine).

* Chapters 2, 3, 4 of [this book](https://webapps.itc.utwente.nl/librarywww/papers_2009/general/principlesremotesensing.pdf) are fine. Don’t worry too much about memorizing things etc.

**Sentinel-2**: We will use Sentinel 2 data a lot.

* All (too much) detail is [here](https://sentinel.esa.int/web/sentinel/missions/sentinel-2).
* Read the web-links on the left of the page under Sentinel 2, but don’t feel you need to chase links with each.

## Topic 2: Google Earth Engine (GEE)

*GEE is almost a self contained programming environment for processing satellite data. It uses its own native code* [*API*](https://code.earthengine.google.com/)*. This is one time when following the readme pages makes sense.*

**Figuring out what it is**:

* Read [this page](https://developers.google.com/earth-engine), but when you get to APIs other than their native one only focus on PYTHON.
* If you like reading, check out their [paper](https://www.sciencedirect.com/science/article/pii/S0034425717302900) on it. I prefer doing so I would do the Trying it out part.

**Check out datasets**: A lot of the GEE examples use Landsat data.

* But also learn about [Sentinel 2 data](https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR). It should be familiar if you finished checking the ESA page on Sentinel 2.

**Trying it out**:

* [Play around](https://developers.google.com/earth-engine/guides/getstarted) on GEE.
* I will demo our [main app](https://github.com/rfernand387/LEAF-Toolbox) at some point

## Topic 2.5: QGIS

*You will need software on your own machine that can display and check exported imagery from GEE. While you could use python it is easier to use a Geographic Information System since it displays satellite data using a GUI that is hopefully more intuitive than FOLIUM in python.*

**Download and install QGIS**: <https://www.qgis.org/en/site/>

* You can delete it after the work term.
* Display a shared asset for one of the field sites users/hemitshah/WP3/CCRSInSituLAI2019\_HayRiver.
* Export a subimage of Sentinel 2 image in summer 2020 at 10m resolution to your google drive covering the site. Do the same for Landsat 8 OLI. Then save both on your local drive and display it in QGIS. Check how each band looks visually, try stretching bands. Compare resolutions of bands.

## Topic 3: Statistical Learning

*The basis of your projects, other than the fact it uses remote sensing measurements, is that some sort of statistical approach is used to learn a mapping from a set of measurements to an output estimate of a vegetation biophysical variable (or maybe even statistics about the probability distribution of the estimate).*

**Basics**: We will use regression a lot.

* Read Chapters 2.1, 2.2 and 2.3 of [this book](https://www.statlearning.com/) (**especially note Figure 2.7**).

**Linear regression**: We use it sort of as a baseline approach.

* Read [Section 3.1](https://www.statlearning.com/) on linear regression. This may not be pleasant if you have painful memories if you took it before or if you find the math not so exciting. **Page 62 is most relevant.**

**Cross-Validation**: Cross-validation is used to check how well a regression method works.

* Read [Chapter 5.1](https://www.statlearning.com/)

**Curse of Dimensionality**: ML works well for remote sensing as we have LOTS of data (millions of training samples if need be!) but it is complicated by the fact that we can’t expect a good estimate if we chose every possible type of measurement (e.g. all spectral bands) due to the ‘curse of dimensionality’.

* Read [Section 3.5](https://www.statlearning.com/), (if you don’t know the detailed math about linear regression or K-nn regression don’t worry as long as you get it qualitatively).
* Also, if you like geometry read [Section 2.5](https://web.stanford.edu/~hastie/ElemStatLearn//printings/ESLII_print12_toc.pdf)

**Deep Neural Networks**: We currently use two layer neural networks trained using backpropagation to relate the measurement from a satellite pixel to vegetation biophysical variables. While your project may not involve making a new network, it is useful to understand them.

* Read chapters 2, 3, 4 of [this book](http://neuralnetworksanddeeplearning.com/index.html) and Chapter 5 of [this book](https://www.deeplearningbook.org/). You may or may not understand all the concepts but it may be useful to refer back to these books if I (or the software packages) use terms you are not familiar with.

**Lasso Regression**: Eventually both of you will use LASSO regression. It basically allows you to be lazy and not have to pre-select the input variables but start with MANY.

* Start with [this cartoon explanation](https://dataaspirant.com/lasso-regression/#t-1606404715788) for now.
* Later once you start using it in your projects, [Chapters 6.1 and 6.2](https://www.deeplearningbook.org/) may be useful to understand terms used in the software.

## Topic 4: Machine Learning

*At this point we are ready to use a machine to do statistical learning.*

* We will use scikit learn and tensorflow in python with data from GEE. You can run it from your local laptop for now but you will get access to bigger remote machines soon.
* You should work through the code written by my previous student [here](https://github.com/rfernand387/ALR_Earth_Engine). I think python notebooks 1, 2 and 3 are good.

# Week 2 — Follow up

**Go through ALR client side**: Try to understand how it implements the research paper on ALR

* Make a flowchart
* <https://github.com/rfernand387/ALR_Earth_Engine/blob/master/ALR_Client_Side.ipynb>
* You will note in block 28 it makes a lot of new input features using non linear combinations of S2 input bands. These are called vegetation indices and serve to linearize the relationship between input bands and output variables like LAI, fCOVER and fAPAR.
* Read over the references to get an idea of these sorts of effects they have but don't worry too much about site specific details or theory.

Next week you will work on implementing this server side.

# Week 3+ — Development and Testing of ALR for Downscaling

The goal of this output is to develop, implement and validate the ALR algorithm for producing 10m resolution vegetation parameter maps using S2 MSI data. The ALR algorithm will be developed in a Jupyter notebook, tested using reference datasets for downscaling validation, and implemented as a client side application (with potential extension to server side) in GEE.

## Data Sets

The ALR algorithm should work with GEE S2 surface reflectance imagery. A sample of three S2 surface reflectance images will be used for Testing (Table 1). The first corresponds to the validation S2 image for Kouchibouguac that is also available in GEE. The second corresponds to an image over the southern Ring of Fire region where field work was conducted in 2019. The third corresponds to an available summer S2 image of the Fox Creek region also used for validation.

**Table 1. Testing images**

| **Site** | **WV3 Area (km2)** | **WV3 ID** | **MSI ID** | **WV3 View**  **Zenith (°)** | **Ecozone** | **Land Cover** |
| --- | --- | --- | --- | --- | --- | --- |
| Geraldton | 221 | None | Search Geraldton ontario (filter region), pick <10% cloud image for July or August 2019 or 2020  20200811T164849\_20200811T165525\_T16UDA |  | Boreal Shield | Temperate/sub-polar needleleaf forest |
| Fox Creek South | 239 | wv3\_20170811\_N5433W11722 | 20170811T185921\_20170811T190305\_T11UMA | 17.7 | Boreal Plains | Temperate/sub-polar broadleaf deciduous forest |
| Kouchibouguac | 52 | wv3\_20200926\_N4669W06491 | 20200925T151701\_20200925T151656\_T20TLT | 17.2 | Atlantic Maritime | Mixed forest |

<https://code.earthengine.google.com/1c3c6c5f90e8cf3212665d16570ec44d>

## Methods

## Control (SL2P10) – 10d (Week 3 & 4)

Our control algorithm will be to use a version of SL2P using only 10m bands. Here we evaluate the solution of simply using a version of SL2P that has networks calibrated in MATLAB using only RTM simulations for the four S2 10m bands (B2,B3,B4,B8). We will class this approach SL2P10. We know from the MATLAB cross-validation that this solution should not be as good as using all bands. It is useful to document how much maps based on SL2P10 differ from SL2P. In this task you will modify the LEAF toolbox notebook to also apply SL2P10 to the same image as SL2P and then add a final code block to sample and scatter plot the outputs from both approaches.

When you finish you should have a completed notebook and results for each of the three test sites. The results for a site will be density histograms comparing SL2P with two versions of SL2P10: the first where SL2P10 is applied to 10m bands and then averaged to 20m for comparison (SL2P10\_10m), the second where S2 10m bands are first averaged to 20m and then SL2P10 is applied (SL2P10\_20m). You should have two histograms per parameter. As there are 6 main parameters (albedo, LAI, fCOVER, fAPAR, CCC and CWC) you can fit all the output using a panel plot with 2 columns and 6 rows of histogram subplots. The first column will compare SL2P and SL2P10\_10m, the second SL2P and SL2P10\_20m.

If SL2P10 was exactly the same as SL2P we would expect the comparison of SL2P and SL2010\_20m to be exact. But, we would still expect the comparison of SL2P and SL2P10\_10m to show differences as the latter is produced using high resolution measurements.

* Code and details are here: <https://github.com/rfernand387/SL2PD>
* Output here: <https://github.com/rfernand387/SL2PD/tree/master/Results>
* The actual networks and jupyter lab notebook is in <https://github.com/rfernand387/LEAF-Toolbox>
* You will find networks in <https://github.com/rfernand387/LEAF-Toolbox/tree/master/LEAF-Toolbox-CCRS-Files/Inputs>
  + Specifically for Sentinel 2 10m bands: <https://github.com/rfernand387/LEAF-Toolbox/tree/master/LEAF-Toolbox-CCRS-Files/Inputs/Sentinel2/Sentinel2_10>
* You need to upload all the csv files to GEE in your cloud project.

**Steps**

1. Use LEAF-Toolbox Notebook
2. Add new blocks like block 148,149,150,151,152 but with functions that refer to the networks that work only with 10m bands

**def** s2\_createFeatureCollection\_estimates():

**return** ee.FeatureCollection('users/rfernand387/COPERNICUS\_S2\_SR/s2\_sl2p\_weiss\_or\_prosail\_NNT3\_Single\_0\_1\_10m')

**def** s2\_createFeatureCollection\_errors():

**return** ee.FeatureCollection('users/rfernand387/COPERNICUS\_S2\_SR/s2\_sl2p\_weiss\_or\_prosail\_NNT3\_Single\_0\_1\_Error\_10m')

**def** s2\_createFeatureCollection\_domains():

**return** ee.FeatureCollection('users/rfernand387/COPERNICUS\_S2\_SR/weiss\_or\_prosail3\_NNT3\_Single\_0\_1\_DOMAIN\_10m')

**def** s2\_createFeatureCollection\_range():

**return** ee.FeatureCollection('users/rfernand387/COPERNICUS\_S2\_SR/weiss\_or\_prosail3\_NNT3\_Single\_0\_1\_RANGE\_10m')

1. Modify block 249 so that the structure for Collection Options so the name now corresponds to the GEE collection name. For example ‘Copernicus/S2\_SR’. Then make sure ALL the code in the notebook now uses this ‘name’ entry to specify the collection used rather than the name of the dictionary element. I know that they are currently identical but it will be needed for flexibility.
2. Add a new dictionary element to Collection Options corresponding to ‘Copernicus/S2\_SR\_10m. The dictionary element is identical to that for ‘Copernicus/S2\_SR’ but will replace the networks indicated by their 10m counterparts from Step 2.
3. Add new dictionaries to each vis\_options dictionary element.
   1. Replace required band and file names, for example:

**"COPERNICUS/S2\_SR\_10m”**

The only change in this element over "COPERNICUS/S2\_SR” from:

"inputBands": ['cosVZA', 'cosSZA', 'cosRAA', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8A', 'B11', 'B12']

to:

"inputBands": ['cosVZA', 'cosSZA', 'cosRAA', 'B2', 'B3', 'B4', 'B8']

1. For each image, select one parameter (LAI, fCOVER, fAPAR, CWC, albedo)
   1. Apply SL2P by running block 177 and 219, do not export images
   2. Apply SL2P10 to make 10m maps (SL2P10\_10m)
      1. Make modified version of block 177 to run SL2P10
      2. Subsequently run a copy of block 219 producing SL2P10 maps at 10m resolution
   3. Apply SL2P10 to make 20m maps (SL2P10\_20m) for comparison
      1. Make a modified version of block 219 that first reduces the resolution of 10m input S2 bands to 20m
      2. Run the modified version of block 177 for SL2P10
      3. Run the modified version of block 219 to produce 20m map but using SL2P10 network
   4. Compare SL2P, SL2P10\_10m and SL2P10\_20m maps
      1. Reduce the resolution of the SL2P10\_10 maps to 20m (<https://developers.google.com/earth-engine/guides/resample>)
      2. Sample each of the SL2P, SL2P10\_10m and SL2P10\_20m maps (GEE sample)
      3. Produce density histogram of each SL2P10 maps vs SL2P20 map (matplotlib)
   5. Repeat for remaining parameters

## Active Learning Regularization (ALR) (Week 5)

Unlike the control whose calibration is based on radiative transfer model (RTM) simulations, ALR uses a local calibration dataset based on the output of SL2P (at 20m resolution) matched with the input S2 bands (at 20m resolution) to calibrate a regression estimator. While the local dataset will have greater measurement error since it is based on SL2P estimates of RTM simulations and not the RTM simulations themselves it will also likely span a far smaller range and domain of inputs and outputs and, implicitly, input land surface conditions than the global conditions assumed in SL2P. Basically, we are resampling the original SL2P calibration dataset to those conditions that match the local conditions. With ALR we hypothesize that the local conditions will allow us to make almost as good regression estimators using fewer (10m) input bands. There are two general approaches we can follow to apply the local calibration data to 10m bands. We could use the four 10m bands directly as input to the regression (Direct approach). Alternatively we can try non-linear combinations of these bands as input (Complex approach). With the complex approach we make use of prior knowledge about physics to develop combinations that we expect will improve the regression estimate by increasing signal to noise ratio and, where applicable, the linearity of the regression. However, the complex approach increases the number of input features to the regression and can lead to poor predictions due to the “curse of dimensionality”. So, we will need to use feature selection with the complex approach. Here we use LARS for feature selection.

**ALR Sampling – 5d**

ALR requires a calibration database. The calibration database is sampled from the output of SL2P and its corresponding input. The code in [ALR\_Client\_Side](https://github.com/rfernand387/ALR_Earth_Engine/blob/master/ALR_Client_Side.ipynb) assumes the input and output are already combined into one image. This will not generally be the case – users will run LEAF toolbox. You will need to modify the LEAF toolbox script to combine the input and output together as one image. You can then add to this script the appropriate boxes from ALR\_Client\_side to do the sampling. You will note that the current sampling does not consider land cover. You will need to include a land cover band (use the partition band from SL2P output) as one of the output sampled columns so we can later apply separate ALR fits to each land cover. Trimming will not be done during sampling but later once we know the input features.

1. Make a new notebook to do ALR sampling based on the control notebook
   1. Import GEE image collection from SL2P10\_control.ipynb
   2. Run script
   3. Reorder code blocks to put functions at beginning
   4. Modify blocks to use only 10 m bands (B2, B3, B4, B8)
      1. Replace B8A with B8
2. Combine inputs and outputs of export collections in SL2P10 notebook
   1. Add export\_collection bands (8) to input\_collection images and save as GEE asset
3. Modify notebook to take in only 10 m bands
4. Add scale parameter to ee\_LARS function (during sampling step)
5. (Modify get\_num\_pixels function to account for scale factor)

## Application of Direct and Indirect Approaches to Imagery – 10d

You will need to apply each regression model to imagery and test the results just like the control. To do so it is best to extend the sampling notebook to calibrate and apply a regression on each partition of the input image data.

1. Streamline LEAF-Toolbox notebook by making a python module with all the functions required to apply the wrappernets function and another module with all the functions required to create the feature collections in the dictionaries. This will make your notebook easier to read and allow us to maintain updates to the original leaf toolbox functions independently.
2. Package the ALR client side functions as modules.
3. You will need to transition add ALR functions to LEAF Toolbox.
   1. **Try running the server-side GEE file (ON HOLD)**
4. You will now replace the ALR client side functions you are calling with server side functions.
   1. First, package the [server side functions](https://github.com/rfernand387/ALR_Earth_Engine/blob/master/ALR_Server_Side.GEE) in a separate module. The server side notebook has most ALR functions implemented in GEE. You may wonder why we did not just use this - it is because GEE cannot calibrate a network. What Hemit was doing was calibrating the network in the ALR client side, exporting it as a csv file to GEE and running the server notebook.
5. Now one by one replace a client side function call from your extended LEAF toolbox notebook with the server side call and verify results are similar. i.e. LARS, TRIMMING and PREDICTION. You will still need to use client side tensorflow as we have not yet implemented a tensorflow interface for GEE.
6. Verify results using images of test sites (LAI, fAPAR, fCOVER)
7. Document results up to this point
   1. Summarize in S2 document ([here](https://docs.google.com/document/d/1tMcndue3GmkYsQfZXLGi-7vwxFupiWWCjyPE0nwAlwg/edit?usp=sharing)) and describe how plots were made, link to notebooks
   2. Update Github readme pages

## Regression Approaches – 10d

You will now apply your code to partitions of an image and compare different predictors methods.

1. Modify the code so it does a separate prediction for each partition in the image. This will result in many runs of LARS and the network. **(ON HOLD)**
2. Hopefully at some point we will have the ability to transfer the tensorflow model directly to GEE – we are in discussion with Google (for now just use manual)
3. Upload NNET to Earth Engine and use .applyNet function
   1. Walk through ALR\_client\_side notebook (from Hemit’s repo) for example code
4. Compare EE methods for implementing regression (randomForest, CART, etc.)
   1. Integrate regression trees in main notebook and verify that it works (make module)
   2. Make plots of 4 test sites comparing (a) random forest and (b) cart predictions with SL2P
5. Compare in a new (combined) notebook (check so no solution is allowed if lars r2<threshold)
   1. SL2P10m
   2. LARS+Regression tree using the [GEE methods](https://developers.google.com/earth-engine/apidocs/ee-classifier-smilecart) (e.g. [classification](https://www.gears-lab.com/intro_rs_lab5/))
   3. LARS+Neural network (try network like Hemit did)
6. Test the regression model (see Quality assessment below)
   1. Report testing based on hold out cross validation (RMSE, R2, bias)

## Quality Assessment – 10d

1. Select image from 10 sites used for [Richard’s paper](https://docs.google.com/document/d/1On7uXG6wBoM9j05nMwB1XPfAEPjfFO7R/edit?usp=sharing&ouid=117787649193226389875&rtpof=true&sd=true) and check RMSE for hold out sample dataset for LAI, fAPAR and fCOVER for each method
   1. Randomly sample 1000 samples from the input/output image
2. Make scatter plots of SL2p\_control vs ALR result for each algorithm, for each site, for each parameter (do this for only the worst site and best sites)
3. Scatter plot R2 from LARS and RMSE (do in excel)
4. Upload all files to Github (ee assets, quality assessment, plots, etc.)
5. Output images (save as JPEG) of downscaled vs original parameter maps (2-3 of them at least)
6. <https://github.com/hongirsa/ALR/blob/main/3_regression_approaches_errchk3.ipynb>

## Use of GEE Tensorflow API, Benchmarking Code speed – as time permits

1. Learn GEE API for using tensorflow: [intro](https://developers.google.com/earth-engine/guides/tensorflow) and [examples](https://developers.google.com/earth-engine/guides/tf_examples)
   1. <https://www.tensorflow.org/guide/keras/sequential_model>
   2. <https://colab.research.google.com/github/google/earthengine-api/blob/master/python/examples/ipynb/TF_demo1_keras.ipynb>
2. Modify code so it uses **ee.AIModelPredict** for applying network.
   1. ee.Model.fromAiPlatformPredictor()

## Final Presentation and Documentation – 10d

1. Complete code on Github with WIKI
   1. Finish updating Sentinel2 doc
2. Document results with a case study
   1. New Jupyter notebook

## Port to GEE Server Side – 10d (not to be performed)

We will now port the solution to Server side (Javascript!). We will start with a regression tree predictor (there is no active NNET solution) and once Google engineers are available we will try and deploy a tensorflow predictor.

1. Modify the code so it applies to an image in PYTHON (likely manually upload TF network and apply [Hemit’s network prediction function](https://github.com/rfernand387/ALR_Earth_Engine/blob/master/ALR_Server_Side.GEE)). For a regression tree, you can do a separate notebook that applies to an image but this should be pretty easy
2. Update LEAF Toolbox server-side with Javascript functions to perform your ALR solution as an option in LEAF toolbox (to avoid GUI issues we will work on a static copy of the code)
3. Verify performance by comparing it to the Python solution.
4. Document in Github with a wiki