Leeds Python Module Write Up

John Freeman

1 Development Process

As the actual source code involved here is simple, I'll try to go into more detail on the development process I followed as well as the tradeoffs I made. The problem here is pretty simple, build a jupyter widget that blends three rasters according to user input, and additionally can highlight the top 10% of values and can export the results to a file.

The first phase of reading in the three input rasters was easy enough as we could just reuse the methods from the lecture. While it would be more efficient to just call a numpy loadtext method, this method allowed for a bit of additional debugging during loading (the code for which was since removed).

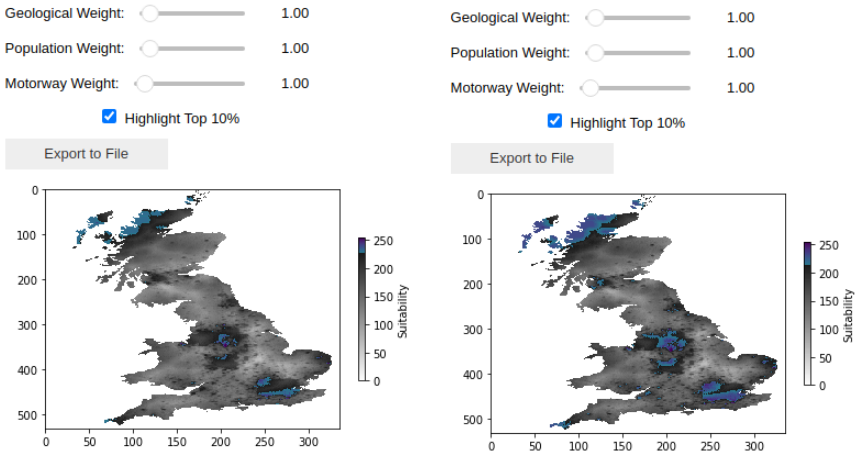
The next part was figuring out how to do a Jupyter widget in the first place, which was easy enough to understand by reading through the Jupyter Widgets docs. That made it clear to accomplish the initial set of requirements all we needed to do was add three FloatSliders, one each for the rasters, do the maths to combine and normalize per those sliders, and do a simple imshow.

Tackling the export functionality was likewise simple after the documentation, being addressed by a simple togglebutton being used to pass a bool, and an if check followed by a numpy text export of the combined raster.

The top 10% section was the most interesting to implement. We add a simple Checkbox to the interactive widget to pass the flag, but when it comes to implementing the change to the visualization, there were two approaches we could try: overlaying a second plot, or constructing a custom colormap. Simply overlaying a second plot would be the easiest option, all we would need to do is find the 90th percentile value, plot it in blue, lower the opacity, and plot it on top of the existing imshow. While this method would work, it's a bit hacky and less fun than the custom colormap solution.

Creating a custom colormap in matplotlib is easier than it sounds - all we need to do is provide a few values and matplotlib will interpolate between them. As such, we just extract 1000 (overkill, but doesn't have a large impact on efficiency) values from the first 90% of the binary colormap and 1000 values from the last 40% of viridis. We then combine the two and pass them into a LinearSegmentedColormap object and the new mapping is good to go.

Now you may be asking, aren't we trying to only highlight the top 10%? Why did we select 1000 values of each? The answer lies at sea - if we select 900 and 100, we'll get a colormap where the top decile is blue, but what we miss is the water portions of the raster



(a) (b)

Figure 1: (a) Unnormalized (b) Normalized

causing the top decile to not be representative of the actual land we're looking at. To address this, we need to manually calculate the 90th percentile value excluding points where the total value is zero (assumed to be water, even though this isn't necessarily the case), and then construct a TwoSlopeNorm object, which essentially just tells matplotlib which value to center the colormap around.

2 Testing

Extensive testing was carried out primarily through verification of the top ten percentiles against ground truth data constructed in numpy (see the final cell in the notebook for details). This same method was used for verifying the normalization and combination of the rasters worked as anticipated. We also had to address various edge cases, ie all rasters being 0%, addressed by adding a constraint on top of the sliders.

3 Future Directions

While we did not encounter significant difficulties implementing the project, there are numerous avenues we could take for improvement. One area would be improving the GUI - in the current implementation interaction is solely through the Jupyter notebook, in future versions making a nicer and more user friendly website with the code behind the scenes would make for a better user experience.

It would also be interesting to overlay the rasters on top of a basemap for ease of interactivity, for example via folium. It would also be interesting to collect more granular data/data from different sources, for example visualizing OSM features or other points of interest to get even more granular data.