Preliminary Design and Results

Clustering Rooftops by Construction Material for Natural Disaster Risk Assessment

James Jensen & Tammy Glazer

Github Repository: https://github.com/jamjensen/mapping-disaster-risk

---

I. Objectives

The goal of our project is to leverage high definition, aerial imagery to identify and label rooftop construction materials in St. Lucia, Guatemala, and Colombia, using color quantization and unsupervised deep learning techniques. A successful labeling strategy for aerial imagery can facilitate the prioritization of building inspections and, in turn, mitigate disaster risk.

Supervised learning requires a large number of rows with correct output labels. When working on image classification, it is incredibly time consuming and costly to create labels, which creates an opportunity for an effective unsupervised approach. While supervised deep learning techniques have historically been applied to pre-processed, labeled training image files to predict specific features contained in these images, we are interested in how unsupervised techniques can be used to derive labels when training data is unavailable. Because, in this case, we do have access to image files with associated roof material labels, we hope to compare our unsupervised output with feature labels to assess the effectiveness of our methods.

II. Data

The data consist of a set of overhead imagery of several locations across three countries (Colombia, Guatemala, and St. Lucia) at 4cm resolution. The features are the images and building footprints, stored as GeoJSONS. Each of the seven images consist of a very large, high-resolution Cloud Optimized GeoTIFF (COG).

III. Data Access

The data are downloadable from the DrivenData Open AI Caribbean Challenge website, in the form of a 34G .tar file. Our initial plan was to host this file on a HTTP file server with the University of Chiago's Research Computing Center (RCC) for efficient file interaction. After a consultation with RCC, however, we were advised to begin by hosting the files locally and opening each component as needed.

Upon data exploration, we learned that the .tar file contains an individual GeoJSON file for each building footprint. In this case, a GeoJSON is a type of dictionary that stores the

building ID, roof material (for testing purposes), and coordinates outlining the building. Additionally, the file contains a single GeoTIFF raster image for each geographic location. A raster image is a bitmap, or a grid of individual pixels that collectively compose an image. In order to begin working with these files, we wrote a Python script that reads a GeoJSON and GeoTIFF file, extracts the relevant information for a single selected rooftop, and displays the single rooftop as an image. This script contains the following functions:

- **Load_geojson:** given a path to a GeoJSON, loads the file in a usable format. An example of a GeoJSON dictionary appears below:

```
{'type': 'Feature',
  'id': '7a2eb7f2',
  'properties': {'id': '7a2eb7f2',
                 'roof_material': 'healthy_metal',
                 'verified': True},
  'geometry': {'type': 'Polygon',
               'coordinates': [[[-74.15911692015449, 4.5493418099525105],
                                [-74.15907727622202, 4.549446583874103],
                                [-74.15903694831816, 4.5494312169867746],
                                [-74.15907659225479, 4.549326443068319],
                                [-74.15911692015449, 4.5493418099525105]]]}
```

- **Make_polygons:** given a loaded GeoJSON, extracts an abbreviated dictionary of relevant information for that building, including building ID, roof material (for testing purposes), and coordinates. Returns a list of dictionaries, where each dictionary maps a building by these features.

- **Transform_coordinates:** given a list of coordinates for a single building, extracted from a GeoJSON, executes an affine transformation on each coordinate pair to output a new coordinate pair based on a coordinate reference system from the corresponding GeoTIFF file. This is a necessary step because coordinates map differently on different locations of the world based on the earth's natural curvature. Returns a transformed list of coordinate pairs.

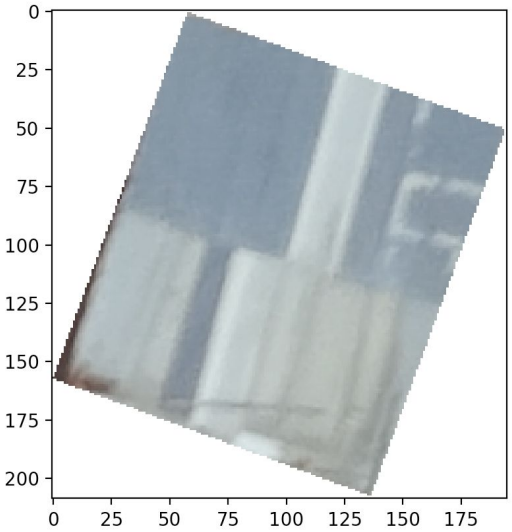
- **Get_rooftop_array_after_mask:** given a dictionary containing transformed coordinates for a single roof, returns an array of arrays providing numeric values for each pixel in the image for a specific band. Each band contains information on surface reflectance from different ranges of the electromagnetic spectrum.[1] In this case, each file contains four bands. The first represents the red band, the second represents the green band, the third represents the blue band, and the fourth band represents alpha, which denotes the level of transparency for each pixel across the three bands. Within each band matrix, each individual value

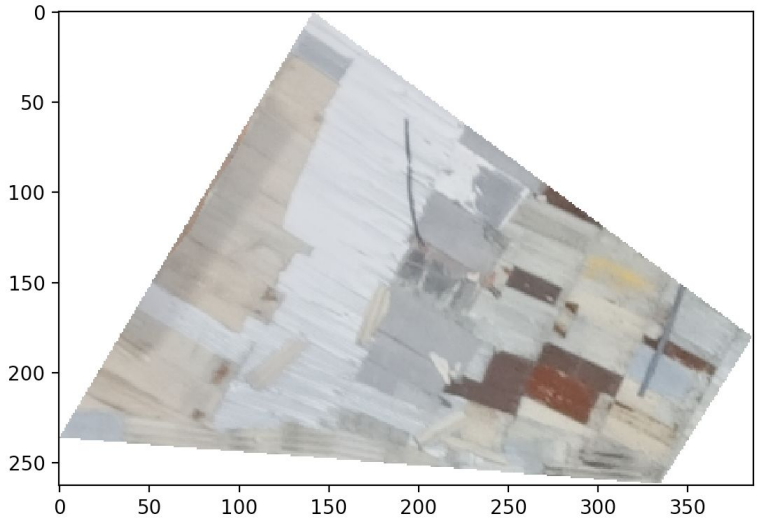
[1] https://automating-gis-processes.github.io/CSC18/lessons/L6/reading-raster.html

represents a pixel and is assigned a number from 0-255, with larger numbers representing more of that color. For instance, an abbreviated, single raster band would appear in the following format for a 3 x 4 pixel image:

```
array([[[0, 0, 20, 100],
        [0, 1, 20, 101],
        [0, 1, 15,  254]]])
```

- **Display_single_roof:** given a numpy array for a single roof, displays the image using matplot. Below are examples of two randomly selected rooftops:



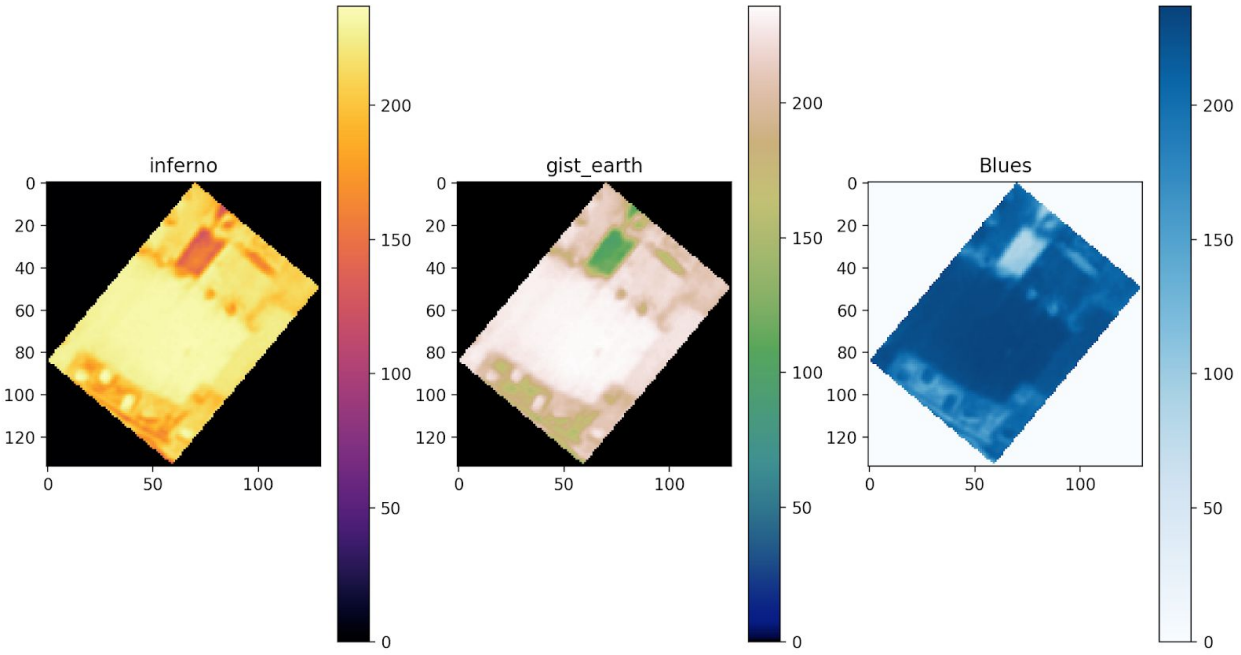


- **Go:** puts the above functions together. Takes a GeoJSON file and the GeoTIFF to which it relates, and displays the rooftop as an image. For reference, below is one of the 7 original satellite images from which the above rooftops were extracted using this workflow:

In order to better work with and process individual rooftops, our next step was to write a Python script that creates a new, small .tif file for a single rooftop and displays each raster band with a corresponding colorbar. This script contains the following functions:

- **Write_building_footprint_ro_raster:** given a numpy array with the pixel values for a given building footprint, writes the array back into a raster file so that we can work with each individual frequency band.

- **Open_and_plot:** opens the .tif file for a single building and plots the color bands separately. Below is an example of this output:



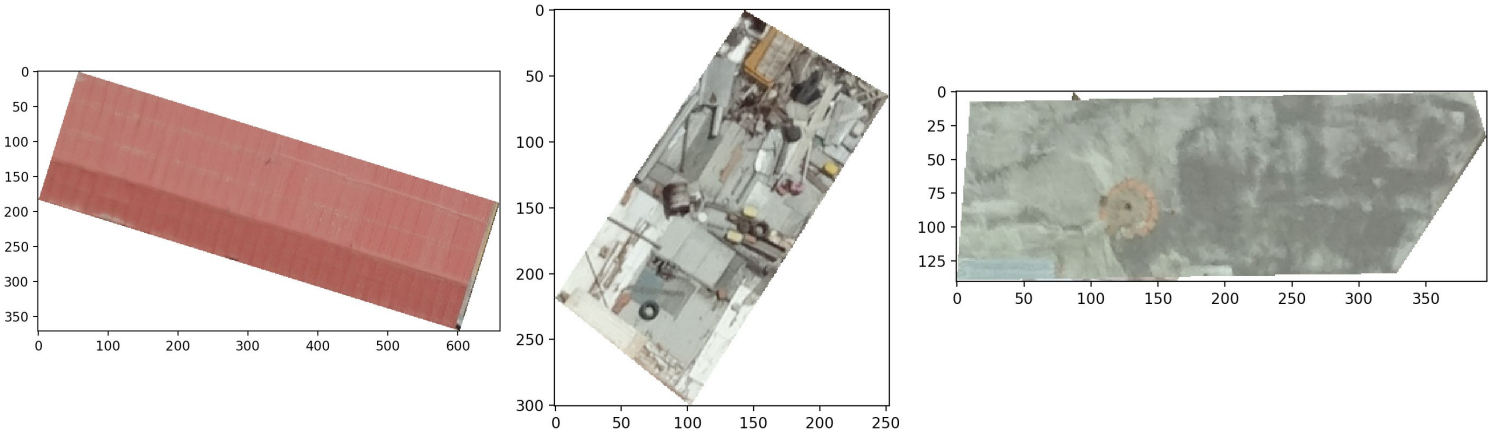
IV. Data Processing

In order to leverage a set of unsupervised learning technique to group rooftops of similar construction materials, it is necessary to define the parameters of our feature matrix. In this case, each row of data will represent a single rooftop or sample. While a matrix of pixel values for a single image can theoretically be presented directly to supervised neural network models in their raw format, we find it helpful to prepare pixel values prior to unsupervised modeling.[2] Each raw image may contain a different number of pixels, hundreds of unique colors, and inconsistent rooftop orientations. These characteristics would not only make it computationally expensive to measure pixel distances, but could also introduce unnecessary noise and incorporate empty background space into the model.

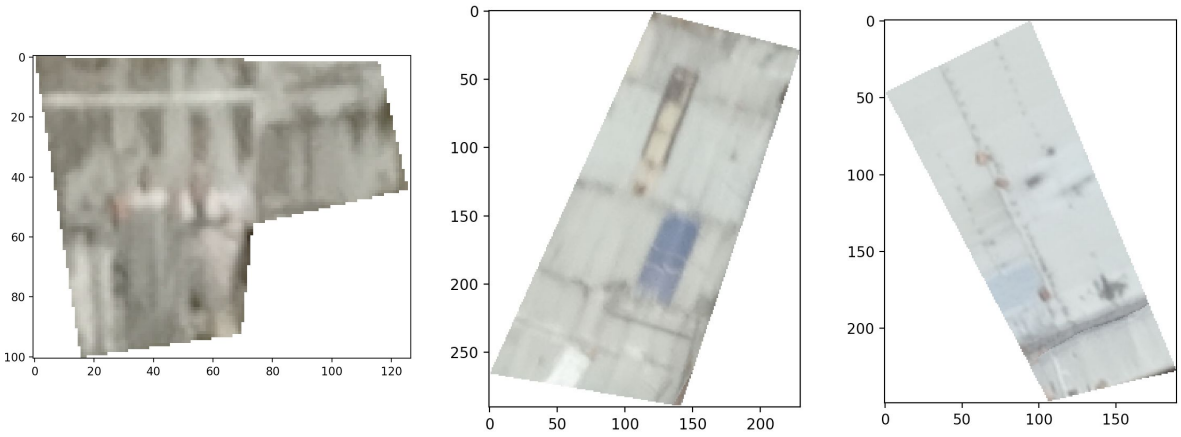
[2] https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/

Therefore, in the next phase of our analysis, we plan to compare the output of different combinations of the following pre-processing steps.

1. **Rotate:** construct an algorithm to detect if an image needs to be rotated, and rotate it accordingly using 'skimage.transform.rotate'. It is important that images have standard orientations, so that our models do not mistakenly group images by their orientation rather than by specific features (eg. colors or patterns) of interest. For instance, the following images represent a few common rooftop orientations present in the data:



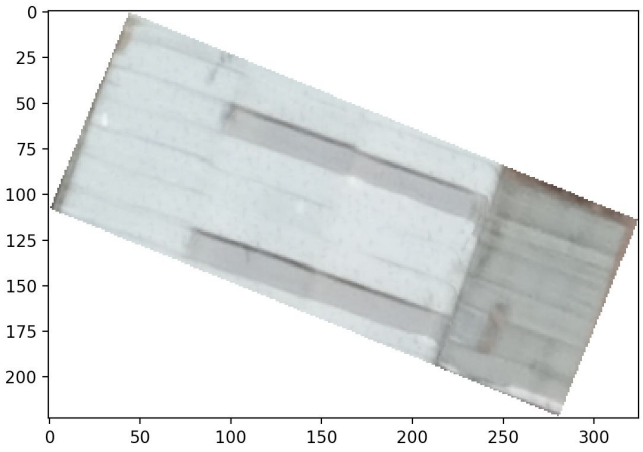
2. **Crop/Scale:** crop and/or scale each image to a uniform size (number of pixels) based on the smallest included rooftop, to ensure that each roof has the same number of pixels and thus features (p) in the final feature matrix. This step will help to account for rooftops with irregular shapes and excess background space encoded with the value zero. It is worth noting that scaling may result in some slightly stretched or distorted representations. The following images represent a sample irregular rooftop shapes:

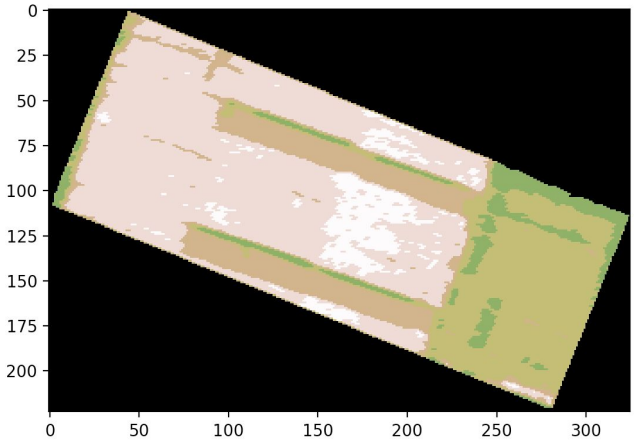


3. **Normalize:** It is good practice to normalize pixel values so that each pixel value has a value between 0 and 1. This can be achieved by dividing all pixels by the largest pixel value, 255.

4. **Center**: Another popular data preparation technique for image data is to subtract a mean value from the pixel values. This approach is called centering, as the distribution of the pixel values is centered on the value of zero.[3] Centering after normalization will mean that the pixels will have positive and negative values, in which case the images will not initially display correctly. Mean values can be calculated per image, per batch of images, per bandwidth, or per dataset. We plan to experiment with a variety of these scenarios.

5. **Cluster-Based Image Segmentation:** In order to detect patterns in rooftop colors and textures and to remove noise, we can partition an image into various parts or segments. Image segmentation creates pixel-wise masks for each "object" in an image. Specifically, k-means clustering can be used to divide images into segments or groups of colors with similar characteristics. Color quantization is an example of a feature reduction technique that is useful in eliminating unnecessary noise in a set of images. To test this processing step, we wrote a Python function that reduces an image into 10 colors/segments/clusters using k-means clustering at n=10. Below is an example of an original image compared with its segmented form:





6. **Aggregate:** Finally, an aggregation step may be necessary to further reduce the size of an image's pixel matrix. For instance, minimum or maximum values can be found per band, mean values can be found per pixel across bands, or values can be added by column. We plan to consult the Center for Spatial Data Science before making a final determination on the best approach for this dataset.

[3] https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/

7. **Flatten:** Finally in order for a single row in the final feature matrix to represent a single rooftop, it will be necessary to flatten the output for a single rooftop into a single array.

V. Analysis & Baseline Output

In order to establish a baseline set of results, we began by calculating a median pixel value for each color band for a single rooftop prior to processing, and creating a simple feature matrix in which the number of features per rooftop, p, is equal to 3. For instance, given three rooftops, a feature matrix might look similar to the following, where the first feature vector represents the red band, the second represents the green band, and the third represents the blue band:

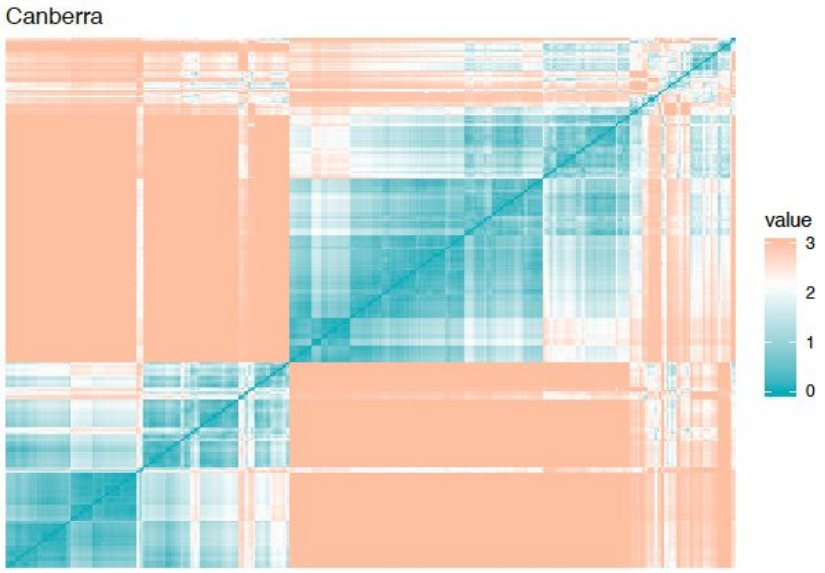
```
array([[30, 12, 140],
       [32, 100, 251],
       [19, 1,  211]])
```

To accomplish this goal, we wrote a Python script that calculates zonal statistics for each color band and creates a baseline feature matrix (dataframe) where each row (n) represents a different roof in Colombia and each column (p) represents the median pixel value for a distinct band. This script contains the following functions:

- **Get_zonal_stats:** given a single polygon and the path to a corresponding .tif file, calculates zonal statistics for the shape outlined by the polygon and returns them as a dataframe. Includes the actual roof material as an additional column, to cross-reference with our unsupervised model output.

- **Go:** loops through a set of roof .tif files and appends a new row to a feature matrix for each additional roof. The limit included as a parameter to the function determines the number of roofs to include in the matrix.

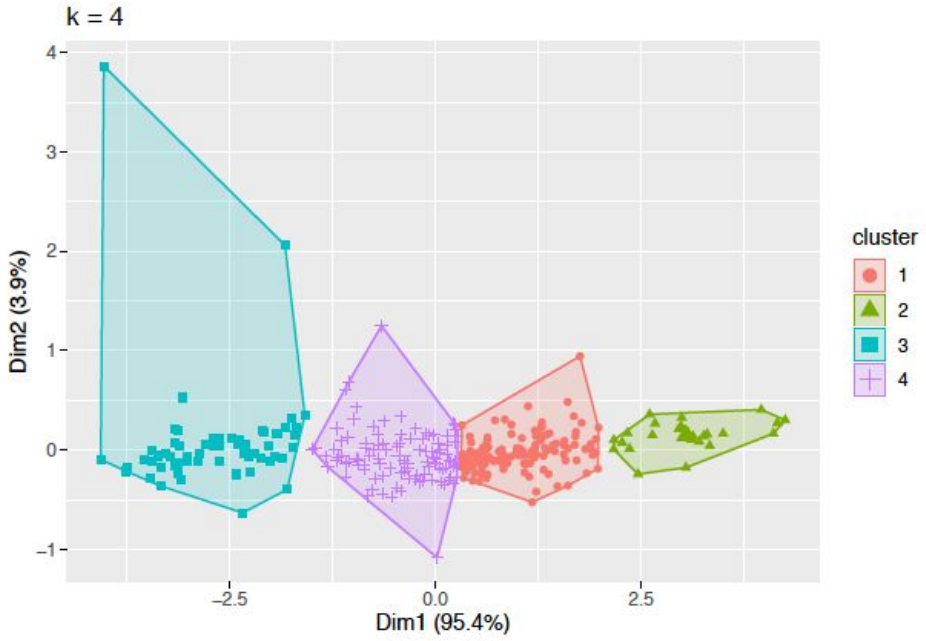
We use R for this analysis. We begin by removing the roof labels prior to learning a model. To diagnose clusterability, we standardized all input values and construct an ordered dissimilarity matrix (ODI) using a variety of distance measures, including Manhattan, Euclidean, and Canberra. For a dataset of 300 randomly chosen images of roofs from Colombia, the following represents sample output for Canberra distance:



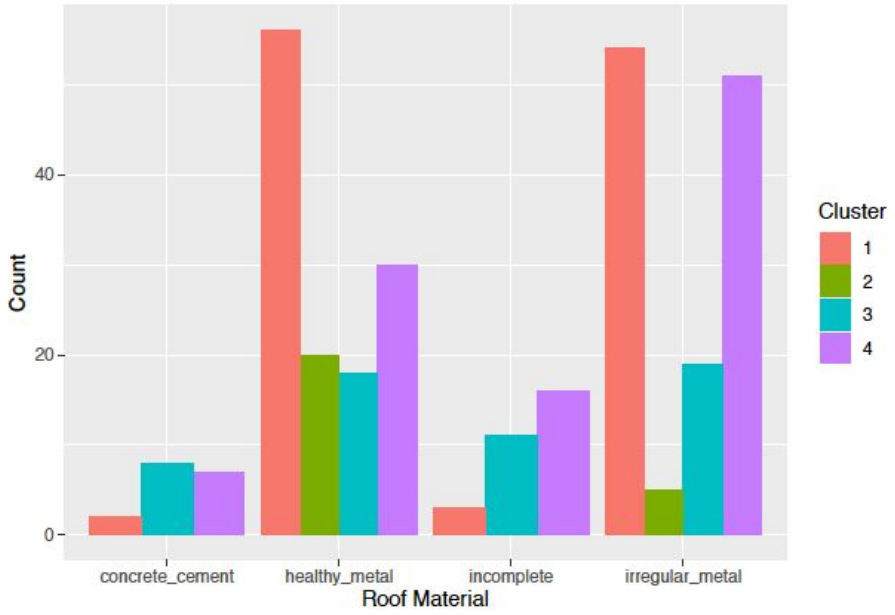
After ordering by spatial proximity, this ODI demonstrates a strong case for clusterability in the data and confirms that there is likely a cluster structure. There are clear dark squares along the diagonal, indicating groupings of similar vectors (low dissimilarity), as well as lighter sections on either side demonstrating high dissimilarity between these groupings.

We then fit a k-means algorithm on our data. In k-means clustering, each cluster is represented by its center or centroid, which corresponds to the mean of points assigned to the cluster. We selected k-means because it is a hard-partitioning algorithm, where we are interested in strict assignment of observations to clusters, such that every cluster is a member of only one cluster. Additionally, this method scales well to large datasets, guarantees convergence, and generalizes to clusters of different shapes and sizes.[4] In this case, we defined four clusters, and we set the 'nstart' parameter equal to 15. This parameter attempts multiple configurations of the model and reports the best one, generating 15 sets of randomly selected initialized centroids. Below is a visual representation of our baseline output:

[4] https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages



Finally, we reassign the known rooftop labels to the data to assess performance. The following graph represents how many of each actual rooftop material ended up in each distinct cluster (note: this testing sample represents a subset of the full data):



This graph reveals some interesting initial findings to improve upon. Specifically, the first and second clusters almost entirely capture metal rooftops, but have a difficult time distinguishing between healthy and irregular metals. A majority of the "incomplete"

category is captured by the fourth cluster, while a majority of the "concrete cement" category is captured by the third cluster. Because this project aims to parse out subtle differences between healthy and irregular materials in order to help prioritize inspections and failure mitigation efforts, however, we plan to layer in additional detail in the form of image segmentation and an enhanced feature matrix based on the outlined processing steps.

VI. Next Steps

Following this initial analysis, we plan to take the following steps:

1. Consult the Center for Spatial Data Science to discuss the benefits of different image processing techniques, including scaling, normalization, rotation, cropping, and feature reduction via principal components analysis (PCA) and color vectorization. Explore image segmentation strategies including edge detection to discern between different planks or slabs, for instance.

2. Experiment with different unsupervised learning techniques to partition the final data, including Guassian mixture models, hierarchical agglomerative clustering, and deep neural networks. We intend to run this comparison in order to internally validate our results and understand whether our algorithm performs well relative to other algorithms or specifications.

3. Run the complete analysis on the full dataset for each country included: St. Lucia, Guatemala, and Colombia. Partner with the Research Computing Center (RCC) to host and run our models on the full cloud-optimized GeoTIFFs, given their large size and the capacity needed for this task.

4. Apply other internal validation strategies to our final data, including calculating the within sum of squares (WSS) for each specification across many values of k and selecting the fit with the lowest WSS value, calculating silhouette width, and exploring the Dunn Index.