A Random Forests Based Canopy Classification System Utilizing NAIP Imagery Within A Python Framework - A Case Study of Georgia

Owen Smith

GISC 4903 Special Topics in GIS

Instructor: Huidae Cho

Abstract: Tree canopy monitoring is an important aspect of maintaining any environment. On a large scale the monitoring and data creation can be a difficult and time-consuming process to undertake. Proprietary software' dedicated to completing such tasks may be expensive and with a lack the insight needed into the innerworkings of their classification algorithms leading to open source alternatives. The Python programming language provides an ideal platform for the creation of a scalable method capable of accurately processing the large amounts of data to classify canopy on a large scale. The Scikit-learn Extra Trees classifier is utilized alongside the Geospatial Data Abstraction Library (GDAL) API to enact the method created. The framework created is based around the classification of imagery from the National Agriculture Imagery Program (NAIP) due to its high spatial resolution. The framework is believed to be comparable to proprietary systems in both accuracy and computational time.

1. Introduction

Deforestation monitoring is an essential part of maintaining any environment as the loss of forested lands leads to increased CO2 being placed into the atmosphere while simultaneously eliminating carbon storage (Bala, Govindasamy, et al. 2007). At smaller scales it leads to both increased runoff rates and subsequently increased erosion, especially in areas where no plant reclamation is initiated (Benito, E., et al, 2003). As improvements are made in within the fields of geoscience and remote sensing, an increasing emphasis is put on accurate forest canopy detection, among other ecological factors, for the purpose of monitoring and predicting change. However, accurately monitoring deforestation to mitigate these effects on a large scale can be a time consuming and difficult process to complete (Basu, Saikat, et al. 2015), and factors such as access to software capable of processing the amount of data required and access to high resolution imagery can be an inhibiting factor. Commercial proprietary software dedicated to completing these tasks such as eCognition (Trimble Inc.) or Textron Systems Feature Analyst (Textron Systems 2010) can be expensive with little insight into the processes and algorithms at work as they are closed source, meaning access to the source code is not publicly available. The lack of knowledge into the inner workings of commercial software leads to the consideration of applying any number of existing open-source processing libraries and software (Sonnenburg 2007). Contrasting with closed source software, open-source work allows for the collaboration and modification of projects between others to serve diverse needs and

purposes, and subsequently allows for transparency in research that leads to increased reproducibility and access (Sonnenburg 2007).

To overcome the potential limitations of proprietary classification software, previous studies have utilized open source libraries such as Keras-TensorFlow, PyTorch, and the Orfeo Toolbox API (Application Programming Interface) for land cover classification and deforestation monitoring (Abujayyab & Karaş 2019, Anh et al. 2019, Rakshit et al. 2018, Grings et al. 2019). However, little research has extensively utilized Scikit-learn one of the leading machine learning (ML) libraries for Python . One study has used Scikit-learn for deforestation monitoring in which a committee system was developed using Scikit-learn's k-Nearest Neighbors (KNN), Linear Discriminant, and Multi-Layer Perceptron (MLP) (Dallaqua et al. 2018). The committee method however is not paired with a reproducible method that can be made scalable. Other research utilizing Scikit-learn has used it only to deploy the logistic regression and support vector machines [SVM] algorithms as baselines or for comparison with other developed machine learning algorithms, but again no method for scalable reproducibility was applied (Šimić de Torres 2016, He et al. 2017, Ortega et al. 2019).

Since an accessible and reproducible method is being created the Python programing language was chosen to develop the process. Python is a core programming language for the geospatial community, with many different applications building core functions on top of it, i.e. ArcGIS's processing toolbox & QGIS's integration with GDAL. Scikit-learn is one of the leading Python libraries for ML. It is built on top of NumPy (van der Walt et al. 2011) and SciPy (Vertanen et al. 2019), two extensive scientific Python libraries which are easy to utilize (Pedregosa et al. 2011) and will subsequently allow for the optimization of raster analysis using GDAL, a Python library used for geospatial data processing (Warmerdam 2008). While other open-source packages like PyTorch and TensorFlow have faster computing times due to their ability to run parallel on GPU (Graphics Processing Units) as opposed to the CPU (Central Processing Unit), Scikit-learn is only capable of parallel computation on CPUs. However, PyTorch and TensorFlow are currently only capable of GPU parallel computation on Nvidia brand GPU's as they are built around Nvidia's proprietary parallel computing platform (GPU Support | TensorFlow 2020, CUDA Semantics – PyTorch 2019), called Compute Unified Device Architecture [CUDA] (Nickolls 2008). The use of CUDA in other ML packages effectively eliminates other graphics cards such as those produced by Intel or AMD and would be counterproductive to the aim of producing a reproducible open source classification system capable of utilization regardless of hardware (Nickolls 2008). In another effort to increase computing times Scikit-learn is also built utilizing Cython, allowing it to reach performance levels of compiled languages (Pedregosa et al. 2011, Behnel 2011). Lastly, the thorough documentation Scikit-learn possesses along with

its extensive supervised algorithms makes it an ideal choice for creating a process that can be both repeatable and reproducible (Pedregosa et al. 2011).

The framework is built around the classification of 1-meter NAIP [National Agricultural Imagery Program] imagery. Previous studies which utilize open-source classification systems have not been able to achieve accuracy at a level which NAIP imagery can provide, having been limited to using only public access imagery such as Landsat-8 which consists of 30-meter resolution bands or MODIS which contain resolutions ranging from 250 meters to 1,000 meters (Šimić de Torres 2016, Dallaqua et al. 2018). While NAIP imagery is on a 3-year cycle and cannot match the temporal frequency in which satellite imagery is taken, NAIP imagery is taken during seasons in which agriculture is growing in the United States ensuring similar characteristics between datasets (NAIP 2009, NAIP Imagery 2019). Furthermore, cloud masking will not be needed for processing as NAIP imagery's quality control removes any image that has more than 10% cloud cover per quarter quad rendering the need for a cloud mask negligible (NAIP 2009, NAIP Imagery 2019). In addition, NAIP imagery has available 4 band NAIP datasets containing red, green, blue, and near-infrared bands allowing for the creation of vegetation indices to better detect canopy (USDA, 2009). The lack of cloud cover in NAIP imagery will remove issues previous studies had with utilizing vegetation indexes as the presence of clouds would render the index increasingly unreliable the more cloud cover the scene contained.

To further separate canopy from other landforms the use of vegetation indices is employed. The vegetation index chosen to use is the atmospherically resistant vegetation index [ARVI] (Kaufman & Tanre 1992). The ARVI was chosen over other vegetation indices such as the Normalized Difference Vegetation Index [NDVI] (Tucker 1979) or the Enhanced Vegetation Index [EVI] (Jiang et al. 2007) as the ARVI corrects atmospheric scattering and has been shown to perform better than other vegetation indices. (Liu et al. 2004). To supplement the ARVI, the viability of the visible atmospheric resistant index [VARI] is additionally explored (Gitelson et al. 2002).

2. Material / Method:

2.1 Data / Area:

The area of study is the state of Georgia, United States. Georgia has an area of approximately 59,425 mi ^{2} and has a rapidly increasing population leading to large potential changes in land features (Lo et al. 2002). Georgia contains a large variety of different land-use and landcover types with the Appalachian mountain belt starting in northeast Georgia, increasing urban sprawl of Atlanta, Farming in central and South Georgia, and the wetlands that make up the coast. Additionally, Georgia has a high degree of ecological variety, ranking sixth among states for biodiversity (EPD 2009), and contains distinct types of

forest in each ecological region (EPD 2009). The listed attributes of Georgia make it an excellent case study for developing an accurate and scalable canopy classification system.

The NAIP dataset used has 3,913 3.75-degree X 3.75-degree quarter quad (QQ) tiles. Each tile is at a resolution of 1m and holds all 4 bands offered by the USDA. The entire dataset is 731 GB's in size. The USDA additionally offers an accompanying seamline QQ polygon shapefile for each state containing spatial and descriptive identifying information for each QQ.

The full state of Georgia is further separated by the 24 physiographic districts of the state. The districts naturally create various sized processing regions for the entire state to effectively split up the creation of the canopy dataset.

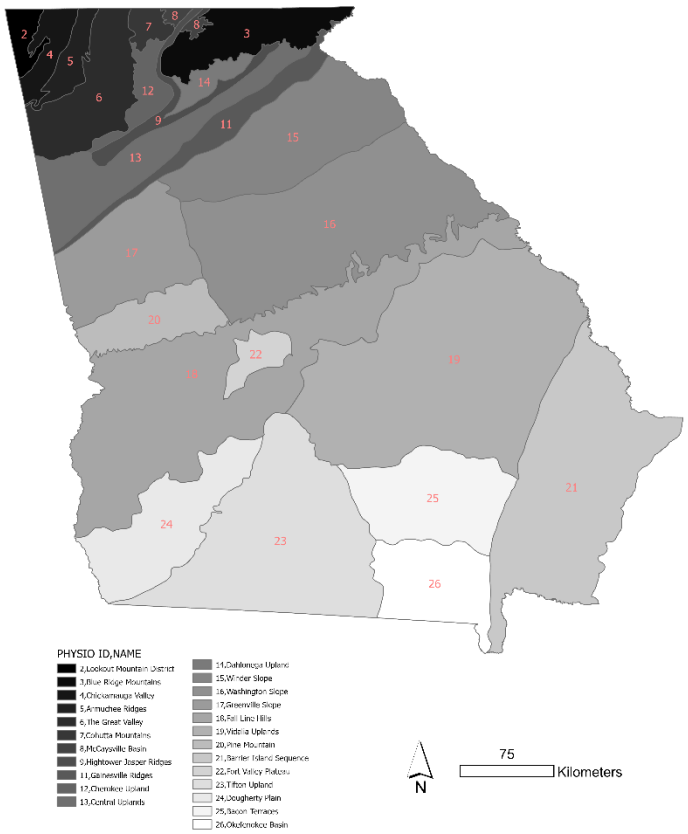


Figure 1. *Study Area*

2.2 Index Creation and analysis:

Vegetation indices are extensively used when trying to separate vegetation from other types of land cover. Vegetation indices typically use the NIR band in their equations as the 0.75 μm – 0.8 μm wavelength of the NIR band is absorbed by photosynthetically active vegetation and reflected by bodies of water and impervious surfaces (Tucker 1979). The indices, like all index equations, are prone to be sensitive to atmospheric effects potentially distorting the accuracy of the imagery. To account for these effects the 2 chosen indices, ARVI and VARI, consider the atmospheric effects in their equations to mitigate atmospheric effects (Kaufman & Tanre 1992, Gitelson et al. 2002).

The ARVI equation uses the blue band in conjunction with the red and NIR band to provide correction for atmospheric effects (Figure 2), and per Kaufman & Tanre the ARVI is four times less sensitive on average to atmospheric effects than the most widely used vegetation index, the NDVI. The usage of four band NAIP imagery can be prohibiting though, as it is not public access for most areas while three band NAIP is, making the consideration of a three-band approach necessary. Visible vegetation indices typically factor in the red and green bands as a ratio, the VARI, uniquely, uses the blue band to mitigate atmospheric issues like the ARVI (Figure 2.). While the VARI has been shown to have high accuracy but verification has been restricted to agricultural land such as wheat and corn, leading to uncertainty about its viability for canopy classification (Gitelson et al. 2002).

$$ARVI = \frac{(NIR - (2 * Red) + Blue)}{(NIR + (2 * Red) + Blue)}$$

$$VARI = \frac{(Green - Red)}{(Green + Red - Blue)}$$

Figure 2. *Index Equations*

Before classification can occur a process for index calculations must be established. A method was set up using GDAL and NumPy to enable the scalable creation of indices for the entire NAIP dataset and applied with python functions written for each index. The method devised is simple but efficient, capable of calculating the entire NAIP dataset in approximately 6 hours. As the VARI calculation is not naturally normalized between -1 and 1, another function was created which normalizes the VARI outputs titled nVARI which will allow for better comparison to the ARVI (Function 1.).

```
Function norm(array):
    array_min, array_max = array.min(), array.max()
    return ((1 - 0) * ((array - array_min) / (array_max - array_min))) + 1
```

Function 1. *Array Normalization*

Early visual comparisons with the first training set made it clear that the VARI would not be practical to detect canopy in an area larger than a small section of agriculture. While the lack of the NIR band can still yield correct results with vegetation, the detection of impervious surfaces and more so bodies of water was nonexistent when not normalized and with substantial amounts of noise when normalized (Figure 3.). Water truthing – or rather the use of the National Hydrography Dataset to determine where water is for certain can be used to change values where water is not classified, but the issue persists with impervious surfaces. Additionally, it is believed water truthing would add a not insignificant amount of processing time and would be better left to the discretion of the end user what postprocessing should be undertaken. Ultimately due to the issues surround the usage of the VARI at the needed scale, the ARVI is used as the index used for this study.

a *b* *c*

Figure 3. *Showing differences in water detection for each index. a) VARI, b) Normalized VARI, c) ARVI*

2.3 Classification Algorithms:

Two classification methods, the Random Forests (RF) classifier and the Extra Trees (ET) classifier (Extreme Random Forests), were considered for the framework. The RF classifier, generally, is a combined multi-tree predictor built upon bootstrap aggregating where each node is split using a random selection of features and the most popular class is subsequently chosen based on a vote after the specified number of trees are generated (Breiman 2001). In cases of land-cover classification random forests are found to be as effective, if not more effective as other popular similar ensemble algorithms such as boosting and bagging (Breiman 2001, Gislason et al. 2006). In addition, the random forests algorithm has been found to have lighter load computationally than the popular Ada-boost algorithm (Freund & Schapire 1996). The lighter computation load of random forests is due to the random selection of variables to split minimizes the correlation between trees, combined with utilization of bootstrapping meaning a portion as opposed to the entire dataset is used for each tree. However random forest algorithms can use a considerable amount of memory as a matrix of the number of samples (N) x number of trees (T) is stored in memory (Gislason et al. 2006). With memory usage in mind, random forests are still an ideal algorithm for use with large datasets

as it does not overfit as the algorithm follows the Law of Large Numbers (Etemadi 1981) and is considerably less sensitive to noise than other boosting or bagging algorithms (Breiman 2001).

The ET classifier is like RF in that it is a multi-tree predictor built using an ensemble of decision trees and the most popular class is chosen based on an aggregation of trees. However, in contrast to RF the ET classifier splits the nodes of the tree completely at random whereas RF cuts the node at the locally optimal combination of features/split (Geurts et al. 2009, Breiman 2001). Additionally, ET uses the entirety of the sample and not just the bootstrap to grow trees, meaning each tree is independent or uncorrelated to the last (Guerts et al. 2009). The ET classifier has higher bias and lower variance than the standard RF classifier due to the increased randomness of the split nodes. (Geurts et al. 2009). The ET classifier is found to be well suited for land cover and geospatial classification especially when the datasets are increasingly noisy, due to ET's higher randomness increasing generalization, or when features are highly correlated (Lawson et al. 2017, Xu et al. 2010). In terms of computation time the ET classifier is found to be faster and more efficient than the random forests classifier, and in this study was found to be roughly 3x quicker computationally than the RF classifier.

As the high resolution NAIP dataset being worked with lends itself to increased accuracy, it also makes the dataset increasingly noisy. In addition, the nature of the training data would ultimately lead to highly correlated trees in an RF forest classifier, meaning higher variance, as only two sets of training data will be given. As the ET classifier has been shown to work better at classifying noisy data and to balance out variance and bias the ET classifier on the surface is seemingly better for the NAIP dataset and after testing the ET classifier is shown to be better for our usage.

2.4 Training Data:

Training data was drawn using Quantum GIS 3.10 (QGIS), a freely available open source GIS software. The data was drawn as a vector shapefile with the values of 1 for non-canopy and 2 for canopy. Two sets of training data were drawn on two separate NAIP QQ's with unique features and with different areas. Future iterations will use 0 for non-canopy and 1 for canopy.

The first training set was NAIP tile m_3008101_ne_17_1_20151017. It is located approximately 17.5 kilometers northwest of Folkston, GA and is between the Okefenokee Wildlife Refuge and the coast in southern Georgia. The distinguishing features of training set one is the river flowing through the center of the QQ and the wetlands that surround it, a sizable portion of the QQ is agricultural land. Training set one, however, contains little built up or impervious land. Initial testing showed that the lack of impervious and built-up land in the training data leads to issues with classifying other NAIP tiles, primarily those with larger urban areas. Additionally, most of what appears to be canopy on training set one is row crops, further

negating the accuracy of the training dataset as fewer canopy labels can be created. To create a more balanced training dataset another NAIP QQ was used.

A more balanced dataset for our purposes would be one with a large amount of forest and non-canopy labels with a blend of urban, bare land, and pastures. The second training dataset was NAIP tile m_3408326_ne_17_1_20150915. It covers all of Cleveland, Ga. While the amount of non-canopy labels is less for the second set than the first (Table 1.) it has a much high diversity of non-canopy labels with a blend of built up, barren, pasture that the first training set did not consist of.

	Number of pixels			
Training set	**Unlabled**	**Non-Canopy**	**Canopy**	**Total**
1	42,374,408	3,681,481	4,149,313	42,374,408
2	44,212,871	1,409,836	4,575,285	50,197,992
	Area (%)			
1	84.4	7.33	8.26	
2	88.07	2.81	9.11	

Table 1. *Number of pixels for each dataset*

2.5 Parameter Optimization:

Scikit-learn offers a multitude of parameters to adjust in the ET model—19 different parameters to be exact—each one controlling and potentially changing the classified outputs. While manually determining the optimal parameters can be effective, a computational method can aid the process by determining the validity of models quicker than the human eye in addition to potentially cut down on human bias as the computational method has no preconceived bias regarding the data. Scikit-learn in addition to providing the ET classifier being used also offers a method to optimize hyperparameters titled `RandomizedSearchCV` or Randomized Search Cross-Validation (RSCV). The RSCV used performs a 5-fold cross-validation for parameter combinations and determines based on the cross-validated (CV) score and the computational time taken which set of parameters is optimal ensuring a balance of accuracy and time.

Two parameters were chosen to focus on. The number of estimators (n_estimators), and the minimum number of samples required to be at a node (min_leaf_samples). The number of estimators is the number of trees generated in a forest. It was chosen to optimize because while the accuracy of the model can increase with more trees, the amount of time taken also increases linearly with the number of estimators in a forest. The accuracy and computational time split are especially important due to the size of each NAIP tile being large, thus adding more trees than necessary to ensure accuracy will undercut the potential scalability as the process becomes bloated computationally.

The number of leaf samples required to split a node was the second chosen parameter to optimize, not because of its effect on accuracy, but because of the parameters' effect of smoothing the model. For the noisy dataset that is NAIP imagery, smoothing the model can lead to a cleaner output.

```
Function split_data(training_raster, training_fit_raster):

    y_raster = gdal.Open(training_raster)
    t = y_raster.GetRasterBand(1).ReadAsArray().astype(np.float32)
    x_raster = gdal.Open(training_fit_raster)
    n = x_raster.GetRasterBand(1).ReadAsArray().astype(np.float32)
    y = t[t > 0]
    X = n[t > 0]
    X = X.reshape(-1, 1)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

Function 2. *Function to split dataset*

For the RSCV the training data is first split into a test data set, and a training dataset. The test dataset is 1/3 of the entire dataset (Function 2). The testing dataset will be used to run parameter testing as it cuts down on computation time and still provides an accurate subsection of the data. A list of values is generated for each parameter. The two resulting lists are input into a dictionary from which the random search will pull values (Function 3.).

```
Function tune_hyperparameter(training_raster, training_fit_raster):

    y_raster = gdal.Open(training_raster)
    t = y_raster.GetRasterBand(1).ReadAsArray().astype(np.float32)
    x_raster = gdal.Open(training_fit_raster)
    n = x_raster.GetRasterBand(1).ReadAsArray().astype(np.float32)
    y = t[t > 0]
    X = n[t > 0]
    X = X.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

    n_estimators = [int(x) for x in np.linspace(start=10, stop=500, num=10)]
    min_samples_leaf = [int(x) for x in np.linspace(start=10, stop=500, num=10)]
    random_grid = {
        'n_estimators': n_estimators,
        'min_samples_leaf': min_samples_leaf
    }
    etc = ExtraTreesClassifier(n_estimators=100, n_jobs=-1, max_features=None)
    clf = RandomizedSearchCV(etc, random_grid, random_state=0, verbose=3)
    clf.fit(X_test, y_test)

    print(clf.best params )
```

Function 3. *Function to tune the parameters of the ET classifier*

For each parameter, the final CV score and time is output, and the mean is taken for all 5 iterations of each parameter pairing. The mean output is the value used to determine the best set of parameters. The results showed that the number estimators which was expected to have a higher impact on accuracy had a very minimal impact. The accuracy for the number of estimators between the values 10 and 500 only ranged in accuracy for 96.6212750 % and 96.6212925 %. While the number of estimator's accuracy only ranged between 2 millionths of a decimal place, the computation time increased on a linear trend throughout the testing starting at approximately 20 seconds and ending at just over four minutes when the number of estimators reached 500. The difference in trends for the number of estimators allows for computational time to be optimized without accuracy being sacrificed.

Conversely the minimum samples leaf trends are opposites of the number of estimators. The CV score decreases as the number of minimum samples leaf increases with the score range between 96.2 % and 95.6 %. Time decreases as the minimum samples leaf grows; however, the time only ranges between 1.3 seconds and 0.94 seconds making the time / score split much easier than initially thought. For minimum samples leaf the highest accuracy option can be chosen without much meaningful change to time.

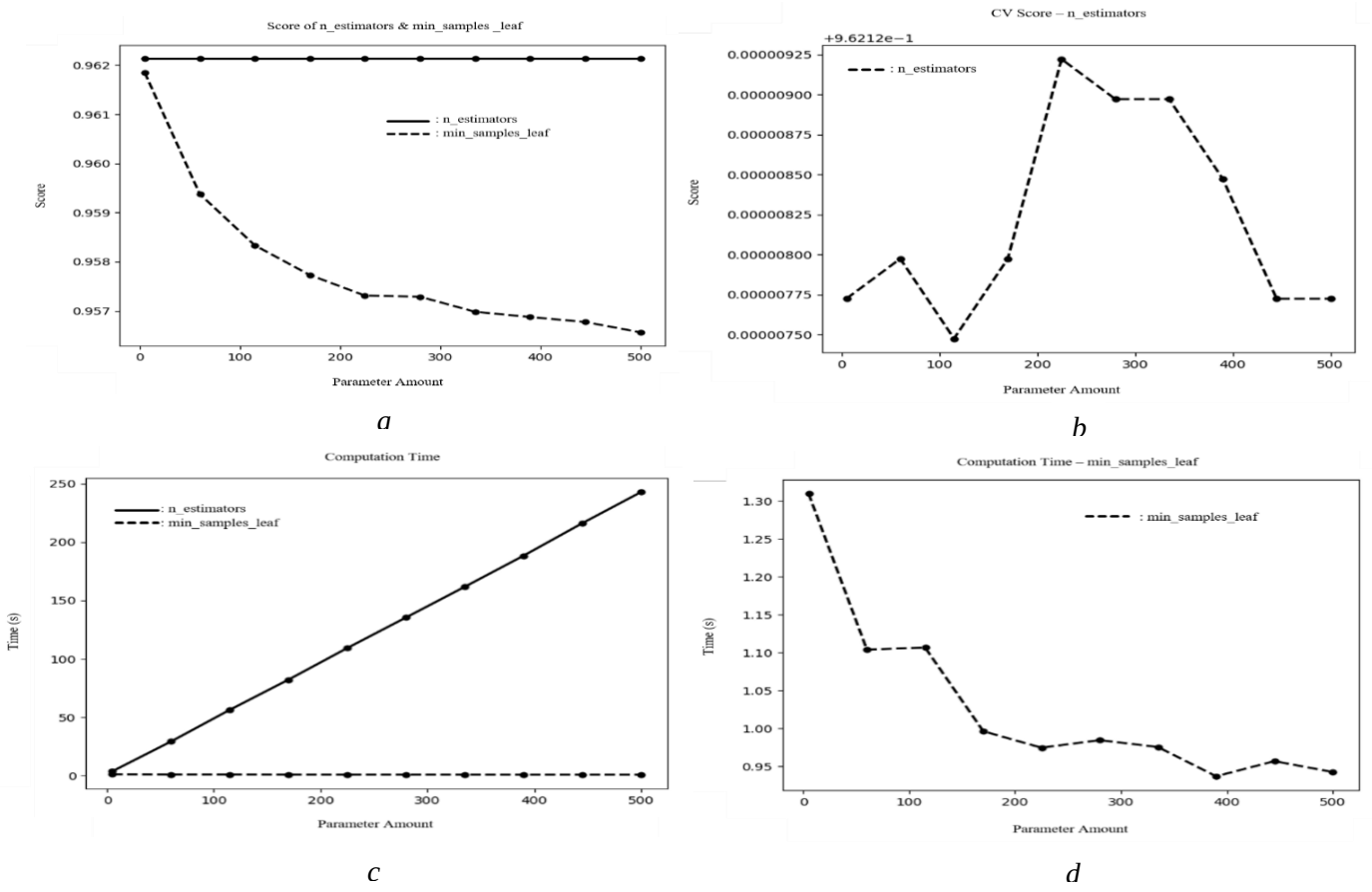


Figure 4. *Tuning graphs: a) Score of n_estimators and min_samples_leaf, b) CV Score – n_estimators, c) Computation Time, d) Computation Time – min_samples_leaf*

2.6 Full Data Creation Process

2.6.1 Setup

Before processing starts, a configuration file is created titled config.py. Within the configuration file lies the variables that will allow the process to create the required output folders in a clean and tidy workspace. Additionally, the configuration file contains all paths for the physiographic district's shapefile, the NAIP QQ seamline shapefile, and the input directory for the NAIP dataset that the main processing functions within canopy_foss.py will utilize. Individual folders will be created for each district and within the district folders will be created an Inputs and Outputs folder. The Inputs folder will contain the ARVI rasters after they are created. The Outputs will contain all other data created after classification. Each step of the process reads the data created by the last step for a seamless method which will allow for processing in a linear manner with consistent creation methods. It is important to note that testing the model and converting training data runs independently of the config file to allow for flexibility when testing and creating training data.

It is important to ensure that both the NAIP QQ and physiographic districts shapefiles are in the same projection that is specified within the config file. For this study the projection chosen is EPSG: 5070 NAD83 / Conus Albers and specified within the config file as 'EPSG:5070'.

Configuration Parameter		Description
proj	→	EPSG code of projection in which final data will be reprojected to.
workspace	→	Directory where process will output results and read data from
naip_dir	→	Folder that contains all NAIP
results	→	Folder where all regions folders will be created
class_directory	→	Folder within region folders that will contain final outputs after classification
data	→	Folder where all reference and training data is stored
phyreg_lyr	→	Physiographic districts shapefile
clip_naip	→	Original NAIP QQ shapefile to use for clipping
naipqq_shp	→	Joined NAIP QQ tile with PHYSIO_ID's to query filenames
training_raster	→	Rasterized training data
training_fit_raster	→	ARVI raster which training data applies to

Table 2. *Configuration parameters and their descriptions*

2.6.2 Setting up the NAIP QQ shapefile

All functions created for the process work by reading filenames and physiographic ids from the NAIP QQ shapefile. However, the NAIP QQ file does not contain the physio id's without additional processing. To enable each region to be queried within the NAIP QQ shapefile the "Join Attributes by Location" tool within QGIS 3.10 is used to join the physiographic district shapefile attributes to the NAIP QQ shapefile. The join performed is a 'one for many join' for all NAIP QQ's either within or intersecting a physiographic district. The 'one for many join' is important as one NAIP QQ could potentially be within multiple districts, so a new polygon feature needs to be created with each physio id. The spatial join method will allow for the physiographic districts and their corresponding files to be queried directly using OGR, GDAL's vector processing API. The joined NAIP QQ is saved as a new shapefile

The original unjoined NAIP QQ shapefile is still used however, as GDAL is not able to properly read the required geometry required for clipping of each QQ feature due to the 'one for many join'. The original shapefile will subsequently be used for clipping the tiles their boundaries after the file names are queried from the joined NAIP QQ shapefile.

2.6.1 Create ARVI Rasters

For ARVI creation a list of NAIP filenames is first generated by querying the physio ID in the NAIP QQ. Using the list of file names each NAIP tile is iterated over and the ARVI formula is applied to each individual tile using Rindcalc, a python module for calculating remote-sensing indices (Smith 2020). The creation of the ARVI raster first reads each band with GDAL and converts each band to a NumPy array. For each band in the ARVI formula, the corresponding NumPy array is used. An inputs folder is created within a folder titled after the physiographic district, and the calculated ARVI NumPy array is converted to a GeoTIFF with GDAL and is saved within the inputs folder with the prefix 'arvi_' added to the original file name.

```
>>> import canopy_foss
>>> canopy_foss.ARVI(phy_id=8)
```

Figure 5. *ARVI raster function*

2.6.3 Classifying ARVI Rasters

Classification is applied directly to the ARVI rasters created with the ARVI function. Input files for classification are read using the physiographic district name to determine the folder in which to pull the inputs from. The training data raster and the ARVI raster which the training data corresponds to is first fit with the Scikit Learn Extra Trees classifier model and will then be used to predict the values for the rest of the ARVI rasters. To reduce the size of the output raster, each cell in the raster is allotted 2 bits (values 0-3). The output classified raster is saved with the prefix 'c_'. The main process uses batch_extra_trees, but for training and testing the classifier and training data, training.py contains extra_trees_class which runs independent from the configuration file.

```
>>> import canopy_foss
>>> canopy_foss.batch_extra_trees(phy_id=8)
```

Figure 6. *Classification function*

2.6.4 Reprojection and Clipping of Individual Tiles.

Each NAIP image is provided in the UTM zone (Universal Transverse Mercator) projection. As Georgia is split between two UTM zones – 16 North and 17 North – errors will occur in processing regions that span across the two different UTM zones. Additionally, for comparisons with other data the full dataset should be all in the same projection. The reprojection occurs after the classification using gdal.warp as the reprojection of either the NAIP or ARVI rasters creates nodata values that the Scikit-Learn model cannot handle without further manipulation of the NumPy arrays before classification.

Each reprojected raster is clipped using the corresponding seamline polygon from the NAIP QQ shapefile. The clipping of each individual tile is an efficient way to A) remove the need for a mosaic order and B) remove discrepancies between the data in the overlapping areas of each NAIP tile. It is accomplished by querying the file names of the rasters being processed in the NAIP QQ shapefile and using the queried polygon as the cutline parameter in gdal.warp for the corresponding file. The 2 Bit file size is maintained through this part of the process.

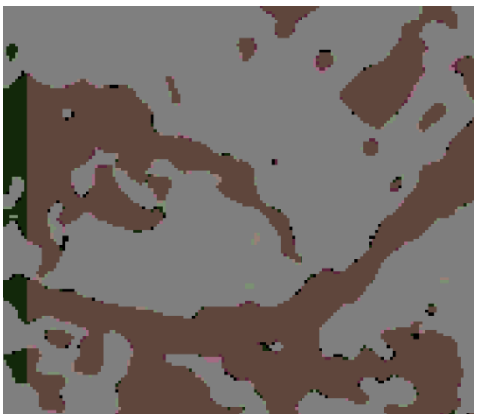


Figure 5. *Overlapping areas of two NAIP QQ's. Brown & white areas are where each tile is the same.*

```
>>> import canopy_foss
>>> canopy_foss.clip_reproject_classified_tiles(phy_id=8)
```

Figure 7. *Clipping and reprojecting individual tiles*

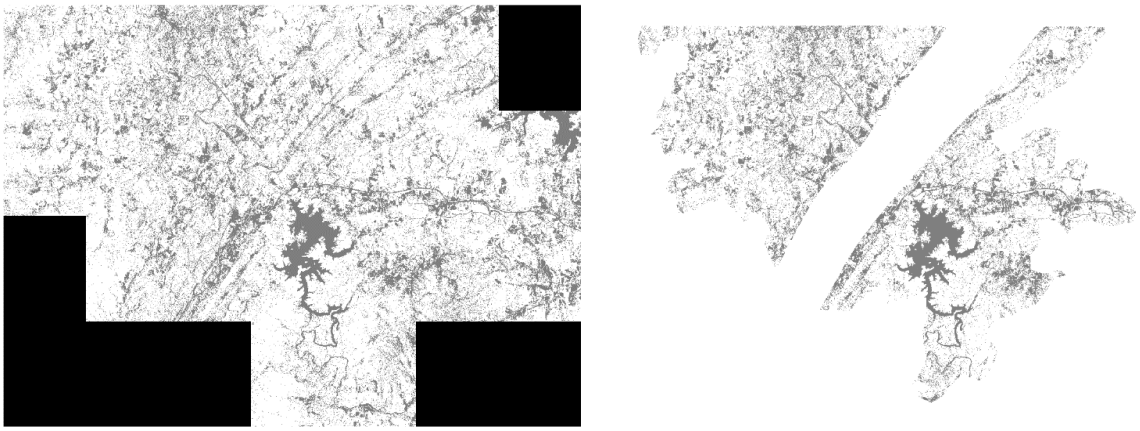
2.6.5 Mosaic Tiles and Clip to District Outline

GDAL does not provide a mosaicking function within its Python API, however, GDAL can still be used to mosaic rasters by using os.system( ) to call gdal_merge.py. To use gdal_merge.py a list of the individual clipped tiles within the regions output folder is created and converted to a single string. The command for gdal_merge.py is created and run by feeding the command into os.system( ). The merged output is saved as a 2 Bit file titled 'mosiac_districtname.tif'.

Clipping is accomplished using the same method as clipping the individual tiles, but instead the physio ID is queried in the physiographic shapefile to determine the cutline feature with which to clip the mosaiced raster. Like all other rasters it is saved as a 2 Bit file to reduce file size.

```
>>> import canopy_foss
>>> canopy_foss.mosaic_tiles(phy_id=8)
>>> canopy_foss.clip_mosaic(phy_id=8)
```

Figure 8. *Mosaicing tiles and clipping to district outline*



a *b*

Figure 9. *Showing a) unclipped mosaic, b) clipped mosaic*

3. Results and Discussion

3.1 Computation time

The computation time of CanoPy FOSS with the Extra Trees classifier is competitive with similar proprietary software, most notably Textron Systems Feature Analyst. For the largest physiographic district – Vidalia Uplands – classification alone takes roughly 36 hours with Textron System, whereas the entire process implemented in this study takes approximately 14 ½ hours from start to end. The computational time for each region can be extrapolated by using the area of each region with the time for the largest region and the smallest region – 21 minutes, McCaysville Basin – with the following formula:

$$T = \frac{(i-j)}{(max\,area - min\,area)} x\,(area - min\,area) + j$$

Formula 1. *Interpolation formula for computation time*

where *T* is calculated computation time, *i* is the computation time of the largest region in seconds (14.5 hrs. = 52200 sec.), and *j* is the computation time of the smallest region in seconds (21 min. = 1260 sec.). The computation times are approximate, however, and assumes that no other heavy computational processes are running, and the computation time is linear.

Physiographic Region	Area Sqkm	Computation Time In Seconds	Comutation Time In Minutes	Computation Time In Hours
Armuchee Ridges	1235	3123.57	52.06	0.87
Bacon Terraces	6848	15451.90	257.53	4.29
Barrier Island Sequence	15563	34594.96	576.58	9.61
Blue Ridge Mountains	3741	8629.07	143.82	2.40
Central Uplands	5545	12590.24	209.84	3.50
Cherokee Upland	1418	3524.74	58.75	0.98
Chickamauga Valley	1388	3460.96	57.68	0.96
Cohutta Mountains	691	1929.65	32.16	0.54
Dahlonega Upland	942	2481.34	41.36	0.69
Dougherty Plain	6630	14973.30	249.56	4.16
Fall Line Hills	20027	44398.74	739.98	12.33
Fort Valley Plateau	1231	3114.93	51.92	0.87
Gainesville Ridges	2567	6048.54	100.81	1.68
Greenville Slope	6589	14883.37	248.06	4.13
Hightower-Jasper Ridges	1532	3775.69	62.93	1.05
Lookout Mountain District	866	2314.05	38.57	0.64
McCaysville Basin	386	1260.00	21.00	0.35
Okefenokee Basin	4980	11349.46	189.16	3.15
Pine Mountain	3416	7914.91	131.92	2.20
The Great Valley	4650	10624.61	177.08	2.95
Tifton Upland	14922	33186.84	553.11	9.22
Vidalia Uplands	23579	52200.00	870.00	14.50
Washington Slope	16541	36742.77	612.38	10.21
Winder Slope	8780	19696.68	328.28	5.47

Table 2. *Showing interpolated computation times*

3.2 Potential Sources for Improvement

For this study, the ARVI index was chosen to build around and with initial testing appears to work well for a variety of situations. For different climates, a different vegetation index may be more useful for the separation of vegetation from different types of landcover, i.e. the Soil Adjusted Vegetation Index (SAVI) (Huete 1988) or EVI (Jiang et al. 2007). To allow for the various indices to be used, Rindcalc can be further incorporated into the process to allow for the choice between several different indices depending on what is believed to be more accurate for the area.

Since the CanoPy FOSS python module was developed with modularity in mind, like the indices, different classification algorithms can be implemented in addition to the extra trees classifier to give a wider variety of options. With increased options for classification algorithms an additional committee method can be incorporated like that developed by Dallaqua et al. 2018. However, for a committee method to be developed the processes of other individual classification algorithms will need to thoroughly be explored within the lens of being reproducible and scalable for canopy classification.

3.3 Comparisons to Other Canopy Creation Methods

For comparison with other canopy datasets the usage of a moving window algorithm for raster comparisons is explored (Costanza 1989, Kuhnert et al. 2005, Kassawmar et al. 2018). When compared to pixel by pixel-based comparison methods the moving window comparison has the advantage of detecting spatial patterns within the data, which is especially important when comparing two datasets created with two different methods (Costanze 1989, Kuhert et al. 2005). The formula is as follows:

$$F_w = \frac{1}{t_w} \sum_{s=1}^{t_w} \left[1 - \frac{\sum_{i=1}^{p} \left| a_{1i} - a_{2i} \right|}{2w^2} \right]$$

Formula 2. *Moving window algorithm as presented by* Costanze 1989

where w is the window size, tw is the number of windows with the window size of w, a1i , a2i are equal to the number of cells with category i in raster 1 and raster 2, and Fw is the index for the moving window with the window size w. The algorithm is implemented with the Python programing language as such:

```
Function neighbors (array, column, row, d=1):

        n = array[column-d:column+d+1, row-d:row+d+1].flatten()
        return n

Function moving_window (array_one, array_two, window=3):
        g=[]
        tw = (height * width) / window**2
        for i in range (array_one.column):
                height, width = array_two.shape
                for j in range(array_one.row)
                        a = neighbors(array_one, i, j)
                        b = neighbors(array_two, i, j)
                        c = a – b
                        d = len(of nonzero array) * 2
                        e = ceiling(2 * (sqrt(len(c))**2
                        f = (1 – d / e) if d != 0 else 1 – 0
                        g.append(f)
                        h = fsum(g)
        return h / tw
```

Function 4. *Function to get neighbors of an array[row][col] and function to perform moving window comparison algorithm*

The most notable canopy creation processes utilizing NAIP imagery, and in particular within Georgia, have been those commissioned by the Georgia Forestry Commission (GFC) for the years 2015 (Bailey & Bailey 2019) and improved upon for the years 2009 and 2019 (Cho 2020). The 2015 dataset was created utilizing Textron Analyst and completed with an estimated accuracy of 91%, making an ideal proprietary dataset to compare our results with. However, while issues with the 2015 dataset methods were discovered and addressed with the creation of an improved method, the improved method was only applied to the 2009 dataset leaving the 2015 dataset difficult to work with as each region has various shifting issues. Additionally, the projection used for the GFC - USA Contiguous Albers Equal Area Conic USGS version - does not work well with GDAL and other open source geospatial applications as it is in an ESRI format.

Due to the mentioned issues the implementation of the moving window has been unsuccessful for comparison and is believed to be due to the dissimilarities in the properties between the two datasets. Testing of the algorithm on small scale matrices produced an accurate similarity coefficient but when

applied to our results the output is a similarity coefficient of approximately 0.005 % which is almost certainly not plausible when visually inspecting the two areas.

4. Conclusions

Using the Python programing language, a scalable and reproducible method canopy classification was explored utilizing NAIP imagery and Scikit-Learn. The method consists of the following steps as described: 1) Calculate the ARVI, 2) Classify the ARVI, 3) Reproject classified tile, 4) Clip the reprojected tiles, 5) Mosaic to region, 6) Clip mosaic to region boundary. The resulting process is found to be computationally quicker than proprietary classification software such as Textron System's Feature analyst and is believed to be faster than spatial operations available in ArcGIS. However, while initial visual inspection believes the result is accurate, the comparison algorithm has been unsuccessful due to the properties of other datasets created by their respective creation methods and a better method needs to be enacted to quantify results.

Additionally, the use of three band NAIP dataset is explored and found to be ineffective for the following reasons. The lack of the NIR band leads to the inability to correctly classify both water and impervious urban surfaces. Water truthing using a National Hydrography Dataset is suggested or the use of a local statistical method is suggested but both methods encounter issues.

Ultimately while more research is needed to created quantifications of the accuracy, the study suggests that a reliable method can be developed to classify canopy at a large scale. For the future, efforts will be made to first quantify the results in a more reliable manner and subsequently to expand the method to more classification methods and indices.

Appendices

All code available at https://www.github.com/ocsmit/canopy_foss

Citations

Abujayyab, S. K., & Karaş, I. R. (2019). GEOSPATIAL MACHINE LEARNING DATASETS STRUCTURING AND CLASSIFICATION TOOL: CASE STUDY FOR MAPPING LULC FROM RASAT SATELLITE IMAGES. International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences.

Anh, N. D., Tuan, V. A., & Hang, N. T. Deforestation hot-spot extraction from Radar Change Ratio (RCR) analysis of Sentinel-1 time series data in Dak G'Long district, Dak Nong province.

Bailey AJ, Bailey COJ. Using feature analyst & NAIP imagery to conduct a statewide tree canopy assessment of Georgia. Forest Res Eng Int J.2019;3(1):15–18. DOI: 10.15406/freij.2019.03.00072

Bala, G., Caldeira, K., Wickett, M., Phillips, T. J., Lobell, D. B., Delire, C., & Mirin, A. (2007). Combined climate and carbon-cycle effects of large-scale deforestation. Proceedings of the National Academy of Sciences, 104(16), 6550-6555.

Basu, S., Ganguly, S., Nemani, R. R., Mukhopadhyay, S., Zhang, G., Milesi, C., … & Cook, B. (2015). A semiautomated probabilistic framework for tree-cover delineation from 1-m NAIP imagery using a high-performance computing architecture. IEEE Transactions on Geoscience and Remote Sensing, 53(10), 5690-5708.

Benito, E., Santiago, J. L., De Blas, E., & Varela, M. E. (2003). Deforestation of water-repellent soils in Galicia (NW Spain): effects on surface runoff and erosion under simulated rainfall. Earth Surface Processes and Landforms: The Journal of the British Geomorphological Research Group, 28(2), 145-155.

Cho, Huidae. CanoPy (2020). GitHub repository, https://github.com/HuidaeCho/canopy

Costanza, R. (1989). Model goodness of fit: a multiple resolution procedure. Ecological modelling, 47(3-4), 199-215.

CUDA semantics — PyTorch master documentation. (2019). Retrieved January 22, 2020, from https://pytorch.org/docs/stable/notes/cuda.html

Dallaqua, F. B., Faria, F. A., & Fazenda, A. L. (2018, October). Active Learning Approaches for Deforested Area Classification. In 2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI) (pp. 48-55). IEEE.

Environmental Protection Division, State of Georgia's Environment. (2009). Retrieved March 21, 2020, from https://epd.georgia.gov/sites/epd.georgia.gov/files/related_files/site_page/Objective%202.pdf

GDAL/OGR contributors (2019). GDAL/OGR Geospatial Data Abstraction Software Library. Open Source Geospatial Foundation. URL https://gdal.org

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Machine learning, 63(1), 3-42.

Gitelson, A. A., Kaufman, Y. J., Stark, R., & Rundquist, D. (2002). Novel algorithms for remote estimation of vegetation fraction. Remote sensing of Environment, 80(1), 76-87.

Gitelson, A. A., Stark, R., Grits, U., Rundquist, D., Kaufman, Y., & Derry, D. (2002). Vegetation and soil lines in visible spectral space: a concept and technique for remote estimation of vegetation fraction. International Journal of Remote Sensing, 23(13), 2537-2562.

GPU support | TensorFlow. (2020, January 13). Retrieved January 22, 2020, from https://www.tensorflow.org/install/gpu

Grings, F., Roitberg, E., & Barraza, V. (2019). EVI Time-Series Breakpoint Detection Using Convolutional Networks for Online Deforestation Monitoring in Chaco Forest. IEEE Transactions on Geoscience and Remote Sensing.

Huete, A. (1988). Huete, AR A soil-adjusted vegetation index (SAVI). Remote Sensing of Environment. Remote sensing of environment, 25, 295-309.

Jiang, Z., Huete, A. R., Didan, K., & Miura, T. (2008). Development of a two-band enhanced vegetation index without a blue band. Remote sensing of Environment, 112(10), 3833-3845.

Kassawmar, T., Murty, K. S. R., Abraha, L., & Bantider, A. (2019). Making more out of pixel-level change information: using a neighbourhood approach to improve land change characterization across large and heterogeneous areas. Geocarto International, 34(9), 977-999.

Kaufman, Y. J., & Tanre, D. (1992). Atmospherically resistant vegetation index (ARVI) for EOS-MODIS. IEEE transactions on Geoscience and Remote Sensing, 30(2), 261-270.

Kelsey Jordahl, Joris Van den Bossche, Jacob Wasserman, James McBride, Martin Fleischmann, Jeffrey Gerard, … maxalbert. (2020, February 17). geopandas/geopandas: v0.7.0 (Version v0.7.0). Zenodo. http://doi.org/10.5281/zenodo.3669853

Kuhnert, M., Voinov, A., & Seppelt, R. (2005). Comparing raster map comparison algorithms for spatial modeling and analysis. Photogrammetric Engineering & Remote Sensing, 71(8), 975-984.

Lawson, E., Smith, D., Sofge, D., Elmore, P., & Petry, F. (2017). Decision forests for machine learning classification of large, noisy seafloor feature sets. *Computers & Geosciences*, *99*, 116-124.

Li, M., Liew, S. C., & Kwoh, L. K. (2004, July). Automated production of cloud-free and cloud shadow-free image mosaics from cloudy satellite imagery. In Proceedings of the XXth ISPRS Congress (pp. 12-13).

Liu, G. R., Liang, C. K., Kuo, T. H., & Lin, T. H. (2004). Comparison of the NDVI, ARVI and AFRI vegetation index, along with their relations with the AOD using SPOT 4 vegetation data. Terrestrial, Atmospheric and Oceanic Sciences, 15(1), 15-31.

Lo, C. P., & Yang, X. (2002). Drivers of land-use/land-cover changes and dynamic modeling for the Atlanta, Georgia metropolitan area. *Photogrammetric Engineering and Remote Sensing*, *68*(10), 1073-1082.

NAIP Imagery. (n.d.). Retrieved January 21, 2020, from https://www.fsa.usda.gov/programs-and-services/aerial-photography/imagery-programs/naip-imagery/

NAIP. Retrieved January 21, 2020, from http://www.fsa.usda.gov/Internet/FSA_File/naip_2009_info_final.pdf

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable parallel programming with CUDA. Queue, 6(2), 40-53.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2019) SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

QGIS Development Team (2020). QGIS Geographic Information System. Open Source Geospatial Foundation Project. http://qgis.osgeo.org

Rakshit, S., Debnath, S., & Mondal, D. (2018). Identifying Land Patterns from Satellite Imagery in Amazon Rainforest using Deep Learning. arXiv preprint arXiv:1809.00340.

Šimić de Torres, I. (2016). Analysis of satellite images to track deforestation (Bachelor's thesis, Universitat Politècnica de Catalunya).

Sonnenburg, S., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., ... & RÃĪtsch, G. (2007). The need for open source software in machine learning. Journal of Machine Learning Research, 8(Oct), 2443-2466.

Smith, Owen. (2020,). ocsmit/rindcalc 2.0.4 (Version 2.0.4). Zenodo. http://doi.org/10.5281/zenodo.3772132

Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn and Kurt Smith. Cython: The Best of Both Worlds, Computing in Science and Engineering, 13, 31-39 (2011), DOI:10.1109/MCSE.2010.118

Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

Textron Systems (2010) Feature Analyst Release 5.0 Providence, RI

Trimble Inc. (2019) eCognition Release 1.3 Sunnydale, CA

Tucker, C. J. (1979). Red and photographic infrared linear combinations for monitoring vegetation. Remote sensing of Environment, 8(2), 127-150.

Warmerdam, F. (2008). The geospatial data abstraction library. In Open source approaches in spatial data handling (pp. 87-104). Springer, Berlin, Heidelberg.

Xu, H., Yang, M., & Liang, L. (2010, June). An improved random decision trees algorithm with application to land cover classification. In *2010 18th International Conference on Geoinformatics* (pp. 1-4). IEEE.

Zhang, J., Hu, J., Lian, J., Fan, Z., Ouyang, X., & Ye, W. (2016). Seeing the forest from drones: Testing the potential of lightweight drones as a tool for long-term forest monitoring. *Biological Conservation*, *198*, 60-69.