EVALUATING SLIDING WINDOW MULTI-SCALAR ANALYSIS IMPORTANCE ON LAND USE CLASSIFICATION FROM SATELLITE REMOTE SENSING DATA

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Dawit Mekonnen Beshah

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

June 2019

Fargo, North Dakota

North Dakota State University

Graduate School

Title

EVALUATING SLIDING WINDOW MULTI-SCALAR ANALYSIS IMPORTANCE ON LAND USE CLASSIFICATION FROM SATELLITE REMOTE SENSING DATA

By

Dawit Mekonnen Beshah

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Anne Denton

Chair

Dr. Simone Ludwig

Dr. Megan Orr

Approved:

June 17, 2019

Date

Dr. Kendall Nygard

Department Chair

ABSTRACT

Implications of artificial intelligence around the intersection of agricultural technology and satellite remote sensing are only beginning to emerge. One of the application areas in the agriculture sector that can leverage machine learning most immediately is land use classification of remotely sensed data. This research shows that window-based aggregates can help accomplish this task by creating extra layers of information in addition to the original satellite image. Sliding window-based aggregation allows creating these processed data layers efficiently. The results show that adding the processed layers to predictive machine learning models can boost classification accuracy.

ACKNOWLEDGMENTS

I want to give a great appreciation to my advisor, teacher, and mentor Dr. Anne Denton. Since from my first day in North Dakota State University, the help, the guide and the inspiration I get from my teachers, classmates and all NDSU community is beyond expectations, and I am very thankful for that. I'm also grateful for my families, which are always beside me. Above all, all the graces to God.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Sustainability in agriculture is crucial to protect natural resources and certify a healthy planet for future generations. Essential to efforts in sustainable farming is the process of map labeling and land use classification. This process requires reviewing satellite images to determine how farmers are implementing sustainable practices. Creating and continuing a comprehensive maps of sustainability practices and land use aids researchers to evaluate preservation works overtime while also helping to identify areas that need special attention and follow-up.

However, such land use classification today still requires a manual and tedious task to tackle analyzing a complex array of geospatial data sources. Imagery from the Satellite like RapidEye constellation can provide geospatial information that has enormous application in agriculture. The RapidEye constellation is capable of field-based, regional or global scale agricultural monitoring on a frequent revisit cycle. The information derived from the imagery can help farmers in precision farming processes, agricultural insurers in damage assessment and risk management, or governments in food security and environmental compliance monitoring. The United States Department of Agriculture's National Agricultural Statistics Service (NASS) also produces land cover usage classifications. Putting the two data source together can help to build a classification model that can predict land use classification processing new orthoimage. However, The United States Department of Agriculture's National Agricultural Statistics Service (NASS) land cover usage classifications come at a resolution of 30 meters. When this classification imagery compared with a 5-meter imagery resolution of the RapidEye images, there exists a significant inconsistency between the quality of the raw image data and the provided image labels. To build a more accurate model, additional information.

In a domain like a map labeling, additional image layers such as depth, or Normalized Difference Vegetation Index (NDVI) help to improve the quality of the classification. Typically, these additional data layers come from another sensor equipment and derived from the original image. For example, NDVI requires Near Infra-Red (NIR) data, which is one band of the 5 RapidEye image bands beside RGB and Red Edge. To process these geospatial data and to create the NDVI or regression layer of information of them, some geospatial analysis method should be implemented.

The sliding window multi-scalar analysis uses all sub-windows of a given window size which computationally highly efficient approach for generating window-based layers from underlying images at multiple scales, with a performance that is logarithmic in the largest window-size.

The sliding window-based aggregation method was used to create the additional NDVI and Regression layer from the original RapidEye image.NASS land use classification images also used as class labels for the original RapidEye image plus the NDVI and Regression layers added on top of the RapidEye image as additional channels of information. Pixel to pixel matching used to create the train/test data set. Then 8 SciKit-Learn machine learning classifiers trained and tested to create a prediction model.

2. RELATED WORKS

Much of previous remote sensing data researches have focused on using satellite images to classifying roads and buildings, for use with mapping technologies such as Google Maps or Open Street Map rather than land use classification. Which shows that there is insufficient research into the problem of creating classifiers for use with other applications such as natural resource surveying.

M.J. Barnsley and S.L. Barr said that Urban land use information derived by analyzing of both the spatial information organization and the frequency of class labels in land-cover data produced from multispectral images acquired by a high spatial resolution satellite sensor. [1]

Five urban land-use classes can be effectively classified using medium spatial resolution satellite data with an overall classification accuracy of 83.78%. The integration of land surface temperature and LSMA-derived fraction images has been demonstrated to be useful for refining the impervious surface image. [2]

T.Kavzoglu and I.Colkesen used A Landsat ETM+ image acquired in 1997 and a Terra ASTER image acquired in 2002 in the study to determine land use/land cover types through classification in a highly urbanized region of Turkey. Training, test, and validation data sets for the two images were formed using a random pixel selection strategy. The main objective of the study was performances comparisons of the classifiers like Support Vector Machine(SVM), Maximum Likelihood, and the effects of the parameters. Results noticeably indicate that the SVM method, for almost all cases, outperformed maximum likelihood classifier in terms of overall accuracy (by about 4%) and individual class accuracies. As they stated, high classification accuracy observed for the water class can be related to its distinctive spectral characteristics or spectral separability compared to other land cover types. When the kernel

function related performances were analyzed, it was found that the radial basis function kernel produced more accurate results than the polynomial kernel by about 2% overall accuracy. [3]

C. Huang, L. S. Davis & J. R. Townshend performed an experiment to evaluate the comparative performances of Support Vector Machine(SVM) with different Kernel algorithm and three other popular classifiers, the maximum likelihood classifier (MLC), neural network classifier (NNC) and decision tree classifier (DTC)) in land cover classifiers. In the first set, only the red, NIR band and the normalized difference vegetation index (NDVI) were used as input to the classifiers, while in the second set the other four bands (RGB, NIR) were also included. Substantial increases in accuracy were achieved when all six TM spectral bands and the NDVI were used instead of only the red, NIR and the NDVI. The additional four TM bands improved the discrimination between land classes. Improvements due to the inclusion of the four TM bands exceeded those due to the use of better classification algorithms or increased training data size, underlining the need to use as much information as possible in deriving land cover classification from satellite images. [4]

3. SLIDING-WINDOW BASED AGGREGATION TECHNIQUE & CLASSIFICATION ALGORITHMS

3.1. Sliding-Window Based Technique

This algorithm uses all sub-windows of a given window that why it is called sliding-widow or moving-window-based techniques. It is a computationally, highly efficient approach for generating window-based layers from underlying images at multiple scales, with a performance that is logarithmic in the largest window-size. The scale has central importance in the processing of geospatial data, and its role is not always intuitive. [5] [6]

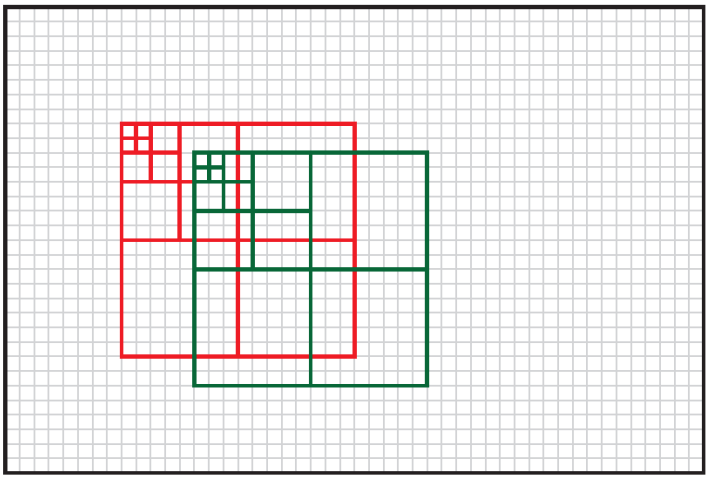

Figure 1: Sliding window aggregation methodology. [5]

Looking at the above Figure 1, The aggregate values for the 2 x 2 windows in the top left corner of each of the large windows can be gained through a simple sum. The entire image is processed in this fashion, resulting in an image of size $(n - 1)(m - 1)$. That means that the adjacent 2 x 2 windows to the right, bottom and bottom-right will also have been computed by the time a full pass of the image completed. [5]

For a window size of w, and an image of width n and height m, there are $(n-w+1)(m-w+1)$ sub windows, considering that the windows cannot extend beyond the

boundaries of the image. By default, any of those windows would have to be parsed, resulting in an overall complexity that is proportional to w^2. However, the complexity can be reduced to being logarithmic in the window size, provided that only linear aggregates are used, such as sum, mean, and count, but not aggregates that depend on the individual values in a non-linear fashion, such as the median. [5]

3.2. Classification Algorithms

3.2.1. Support Vector Machine (SVM)

The idea of SVMs: map the training data into a higher-dimensional feature space and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in the input space. Using a kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space. [7]

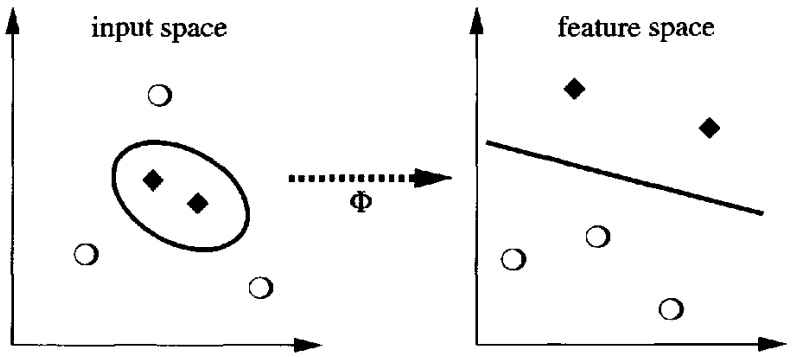

Figure 2: SVM classifier [7]

3.2.1.1. SVM Kernels

The SVM algorithm is implemented using a kernel. SVM uses a technique called the kernel trick. Which, it converts the non-separable problem to separable problems by adding more dimension to it. It is most applicable in non-linear separation problem. Kernel trick helps to build a more accurate classifier.

3.2.1.2. Linear Kernel

A linear kernel can be used as standard dot product any two given observations. The product among two vectors is the summation of the multiplication of each pair of input values.

$$f(x, x_i) = \sum_i^n (\mathrm{x} * \mathrm{x_i}) \tag{3.1}$$

Where n is several samples.

3.2.1.3. Polynomial Kernel

A polynomial kernel is a further generalized form of the linear kernel. The polynomial kernel can differentiate curved or nonlinear input space.

$$f(x, x_i) = \sum_i^n (\mathrm{x} * \mathrm{x_i})^d \tag{3.2}$$

Where d is the degree of the polynomial.

3.2.1.4. Radial Basis Function Kernel

The Radial basis function kernel is a widely used kernel function usually used in support vector machine classification. RBF can map an input space in infinite dimensional space. [8]

$$f(x, x_i) = e^{-gamma \sum_i^n (\mathrm{x} * \mathrm{x_i})^d} \tag{3.3}$$

Where gamma, value ranges from 0 to 1.

3.2.2. K-Nearest Neighbor

Nearest Neighbor Classification is quite straightforward; examples are classified based on the class of their nearest neighbors. Usually, It is better to take more than one neighbor into account, this technique is known as K-Nearest Neighbor (KNN) Classification where k nearest neighbors used in predicting the class. Since the training examples needed at run-time, which means the training set need to be in memory during run-time, because this, sometimes it is called Memory-Based Classification. Because induction is delayed running time, it is considered a Lazy Learning technique. [9] [10]

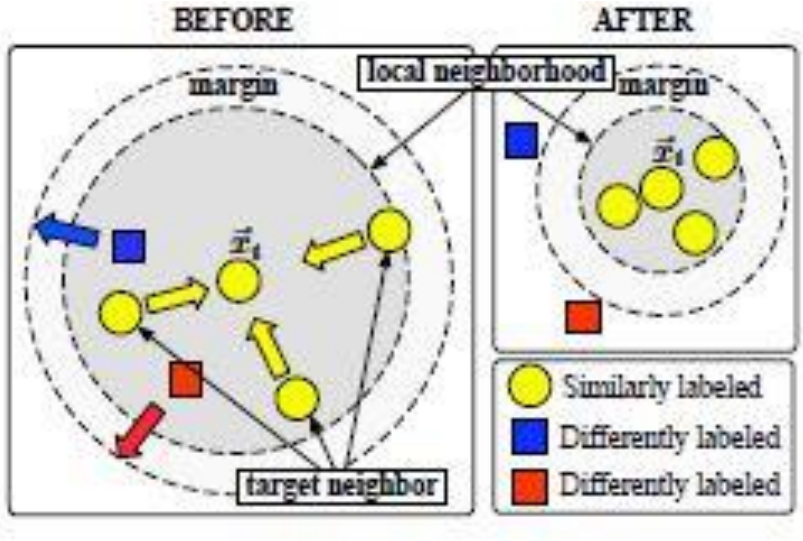

Figure 3: Schematic illustration of one input's neighborhood xi before training (left) versus after training (right). [10]

3.2.2.1. Eager Vs. Lazy Learners

Eager learners mean when given training points construct a generalized model before performing prediction on given new points to classify. Such learners are ready, active, and eager to classify unseen data instances.

Lazy Learning means there is no need for learning or training of the model and all the data points used at the time of prediction. Lazy learners delay until the last minute before classifying any data point. Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples. Lazy learners are also called instance-based learners because lazy learners store the training points or instances, and all learning based on instances. [11]

3.2.3. Decision Tree

The classification technique is a systematic approach to build classification models from an input data set. For example, decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and naive Bayes classifiers are a different technique to solve a

classification problem. Each technique implements a learning algorithm to identify a model that best fits the relationship among the feature set and the class label of the input data. Therefore, the main objective of the learning algorithm is to build a predictive model that accurately predict the class labels of previously unseen data points. [12] [13]

Decision tree classifiers successively partition the input training data into more and more homogeneous subsets by producing optimal rules or decisions, also called nodes, which maximize the information gained and thus minimize the error rates in the branches of the tree. [14] [15] Decision Tree Classifier models a series of carefully constructed questions about the attributes of the test record. Each time it obtains an answer, a follow-up question asked until a conclusion about the class label of the record reached.

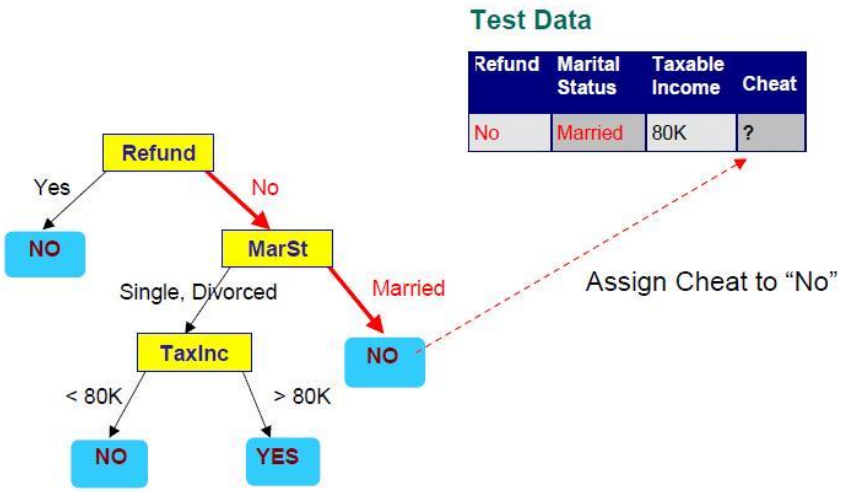

Figure 4: Example Decision Tree for prediction whether the person cheats [12]

3.2.4. Random Forest

The random forest classifier consists of a combination of tree classifiers where each classifier is generated using a random vector sampled independently from the input vector, and each tree casts a unit vote for the most popular class to classify an input vector. The random

forest classifier used for this study consists of using randomly selected features or a combination of features at each node to grow a tree. [16]

The four steps of the Random forest are:

1. Choose random samples from the dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Choose the prediction result with the most votes as the final prediction.

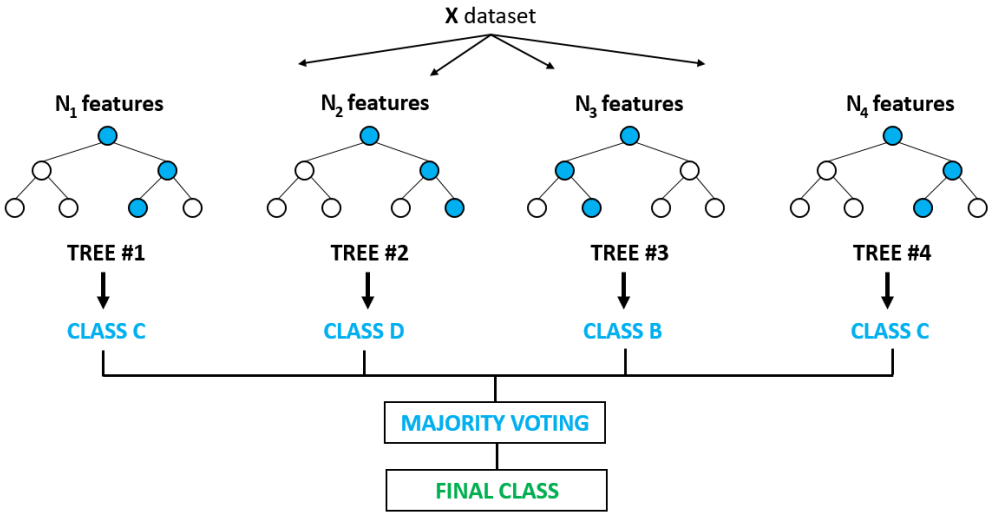

Figure 5: Random Forest classifier [16]

3.2.5. Decision Tree Classifier

Decision Tree is a white box type of Machine Learning algorithm. It shares internal decision-making logic, which does not exist in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of a decision tree is a function of the number of instances and the number of features in the given data. The decision tree does not depend upon probability distribution assumptions. Decision trees can perform well on high dimensional data.

A decision tree algorithm expressed as:

1. Choose the best attribute using Attribute Selection Measures (ASM) to split the data points.
2. Make that attribute a decision node as a decision function that divides the dataset into smaller subsets.
3. The Decision tree builds by repeating this process recursively for each child until one of the following conditions matched:
    - All the tuples fit the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances. [17]

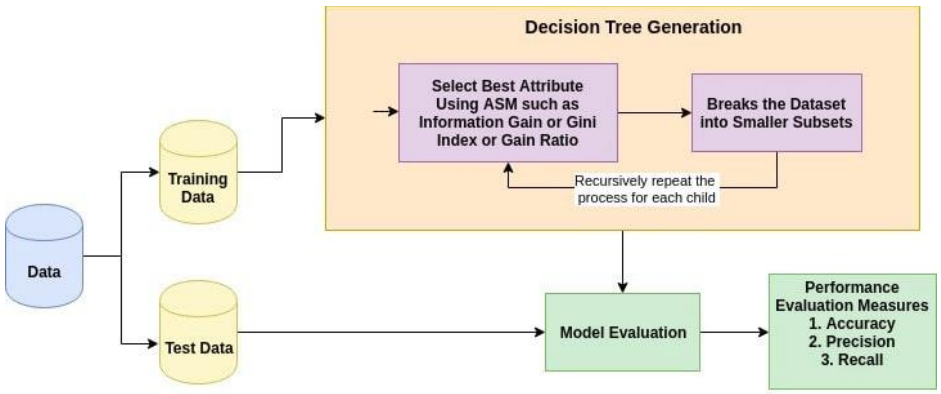

Figure 6: Decision Tree classifier [17]

3.2.6. Extra-Tree Classifier (Extremely Randomized Tree)

The Extra-Trees algorithm builds an ensemble of the unpruned decision or regression trees according to the standard top-down procedure. Its two main differences with other tree-based ensemble methods are that it splits nodes by choosing cut-points entirely at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees.

The Extra-Trees splitting procedure for numerical attributes has two parameters: the number of attributes randomly selected at each node and, the smallest sample size for splitting a

node. It is used numerous times with the (full) original learning sample to generate an ensemble model. The predictions of each trees are aggregated to the final prediction, by majority vote in classification problems and arithmetic average in regression problems. [18]

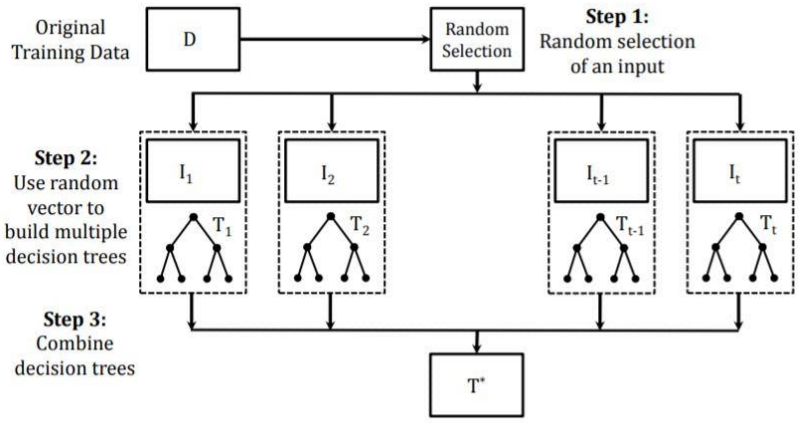

Figure 7: Extra Tree classifier visualized

3.2.7. Gradient Boosting Classifier

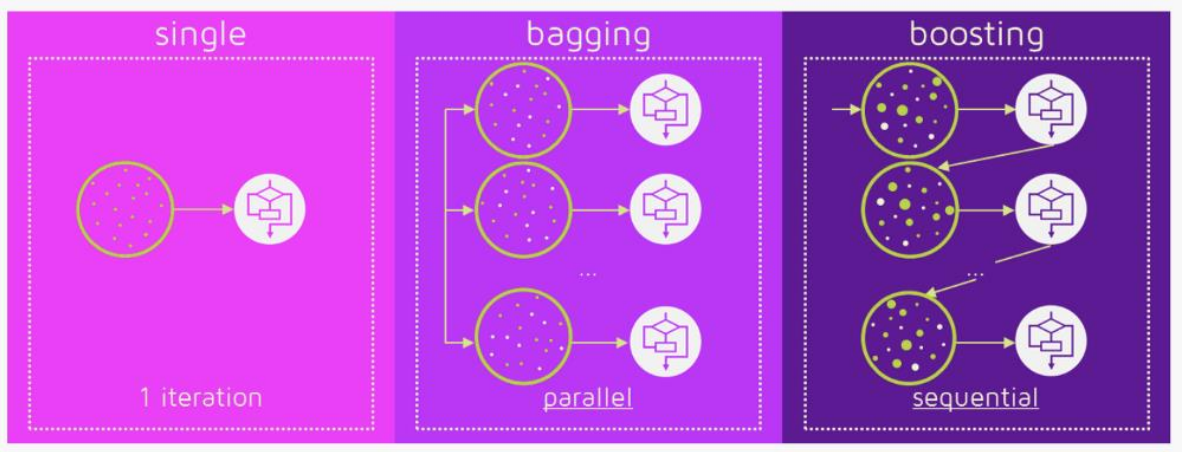

Figure 8: Bagging (independent models) & Boosting (sequential models). [19]

Gradient Boosting Classifier is a compound model that combines the efforts of multiple weak models to generate a strong model, and each additional weak model reduces the mean squared error (MSE) of the overall model and Optimizing a model according to MSE makes it chase outliers because squaring the variance between targets and predicted values highlights

extreme values. When outliers cannot be removed, it is better to optimize the mean absolute error (MAE), also called the L1 loss or cost. [20]

3.2.8. Multilayer Perceptron Classifier

A multilayer perceptron (MLP) is an artificial neural network. MLPs are composed of an input layer to collect the signal, an arbitrary number of hidden layers that are the core computational engine of the MLP and an output layer that decides or predict about the input.

The multilayer perceptron applied to supervised learning problems; they train on an input-output pair sets and learn to model the correlation between the inputs and outputs. The training session involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to calculate those weigh, and bias modifications relative to the error, and the error itself measured in a variety of ways, including by root mean squared error (RMSE). [21]

3.3. Multiclass Classification

Some algorithms, such as Random Forest classifiers are capable of handling multiple classes directly. Others such as Support Vector Machine classifiers or Linear classifiers are strictly binary classifiers. However, various strategies performed multiclass classification using multiple binary classifiers. For example, one way to create a system that can classify the digit images into ten classes (from 0 to 9) is to train ten binary classifiers, one for each digit (a 0-detector, a 1-detector, and a 2-detector,). Then to classify an image, the decision score from each classifier for that image calculated and the class whose classifier outputs the highest score selected as a class of that image. This is called the one-versus-all (OvA) strategy (also called one-versus-the-rest). Another strategy is to train a binary classifier for every pair of digits: one to

distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, This is called the one-versus-one (OvO) strategy. [22]

3.4. Classification Algorithms Performance Measure

Measuring the performance of the classifier is very trick that using cross-reference measure is a must. Let look at some the performance measuring metrics of classifiers.

3.4.1. Accuracy with Cross-Validation

The accuracy of a classifier C is the probability of correctly classifying a randomly selected instance, i.e., acc = Pr(C(v) = y) for a randomly selected instance (v, y) € X , where the probability distribution over the instance space is the same as the distribution that was used to select instances for the training set. [23]

In k-fold cross-validation, sometimes called rotation estimation, the dataset D is randomly split into k mutually exclusive subsets (the folds) D1, D2,…, Dk of approximately equal size. The inducer trained and tested k times; each time t € {1, 2,…, k} it is trained on D\Dt and tested on Dt. The cross-validation estimate of accuracy is the overall number of correct classifications, divided by the number of instances in the dataset. [23]

3.4.2. Confusion Matrix

The idea of a confusion matrix is to count the number of times instances of class A classified as class B. For example, to know the number of times the classifier confused images of a class of 1s with the class of 3s, can be looked in the 1st row and 3rd column of the confusion matrix. [22]

3.4.3. Precision and Recall

Precision and Recall of any prediction can easily calculate using the below formulas;

$$Precision = \frac{TP}{TP+FP} \quad (3.4)$$

$$Recall = \frac{TP}{TP+FN} \quad (3.5)$$

3.4.4. F1-Score

The F1 score interpreted as a weighted average of the precision and recall, where an F1 score ranges its best value at 1 and the worst score at 0. The relative input of precision and recall to the F1 score are equal. [24]

The formula for the F1 score is:

$$F1 = 2 * (precision * recall) / (precision + recall) \quad (3.6)$$

These Performance measures take into consideration to evaluate classifiers score on this research.

3.5. Machine Learning Framework: Python Sci-kit Learn

Scikit-learn is a Python framework of a wide variety of machine learning algorithms, both supervised and unsupervised, using a constant, task-oriented interface. Since it developed on the scientific Python, it can easily be integrated into applications outside the traditional range of statistical data analysis. [25]

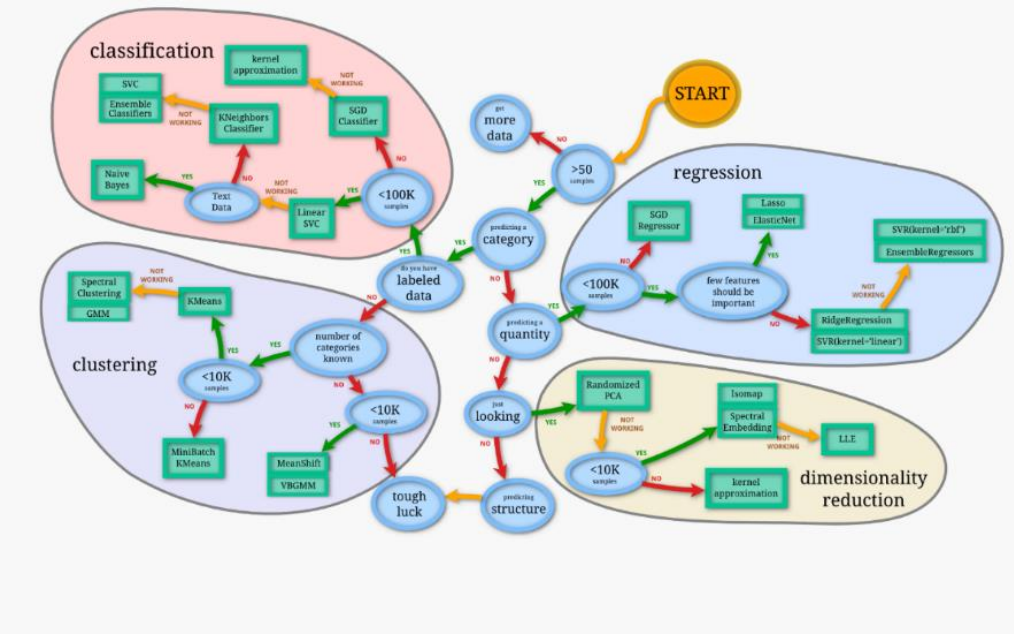

Figure 9: Scikit-learn algorithms flow chart [24]

The steps in using the Scikit-Learn Classifier API usually summarized as:

1. Select a class of model by importing the appropriate estimator class from Scikit-Learn.
2. Select model hyperparameters by instantiating this class with desired values.
3. Arrange data into a features matrix and label vector following the discussion above.
4. Fit the model to your data by calling the fit() method of the model instance.
5. Apply the Model to new data:
    - For supervised learning, often we predict labels for unknown data using the predict() method.
    - For unsupervised learning, we often transform or infer properties of the data using the transform() or predict() method. [26]

4. METHODOLOGY

4.1 Data

4.1.1. Data Sources

4.1.1.1. RapidEye Satellite Imagery

The RapidEye sensors is a constellation of five identical sensors that collect imageries at 5m resolution. This sensor captures the images in five different multispectral bands, which are red, green, blue, red edge, and near-infrared, making it a great option for those doing environmental and agricultural mapping, or measuring crop health. [27]

For this research, a RapidEye satellite image of a small land near Great Bend, ND used as the primary data source.

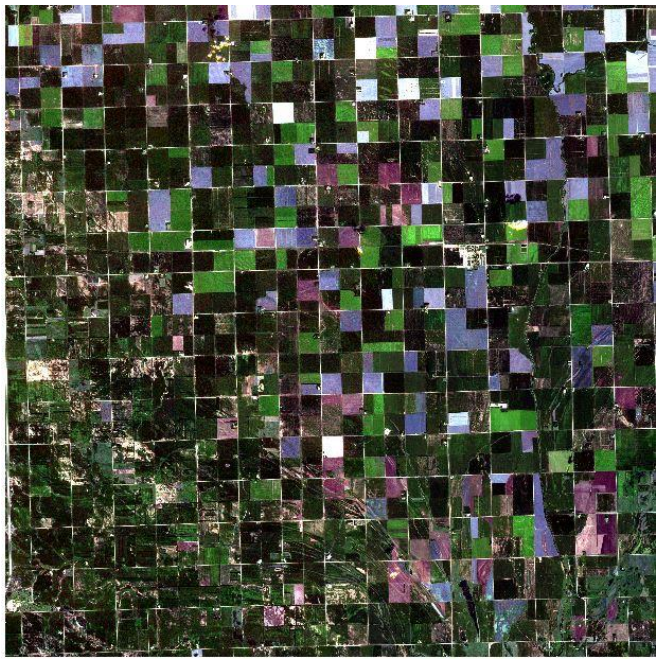

Figure 10: RapidEye satellite image of small land near Great Bend, ND

4.1.1.2. USDA NASS Cropland Data Layer

The Cropland Data Layer (CDL) is a crop-specific land cover data layer generated yearly for the continental United States using reasonable resolution satellite imagery and wide-ranging

agricultural ground truth. The CDL is created by the USDA, National Agricultural Statistics Service (NASS), Research and Development Division, Geospatial Information Branch, Spatial Analysis Research Section. [28] USDA NASS Cropland Data Layer has 254 different classes of cropland cover (See Appendix A)

The same area of land as the above RapidEye image and the CDL used as a label for the RapidEye image and worked on developing models that can predict the crop landcover from the satellite image.

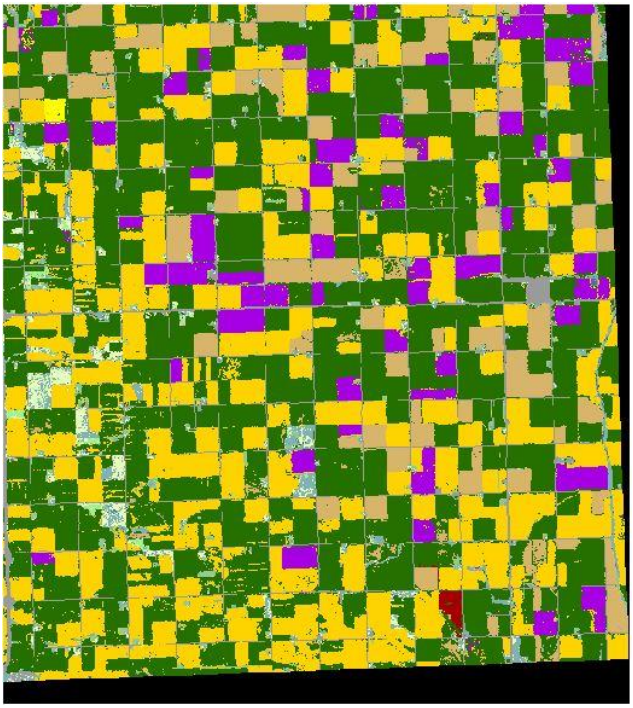

Figure 11: NASS crop landcover data layer

4.1.2. Data Format

GeoTiff: is a public domain metadata standard which consents georeferencing information to be embedded within a TIFF file. The potential additional information such as map projection, coordinate systems, ellipsoids, datums, and everything else necessary to establish the

specific spatial reference for the file included. [29] Both data sources return the data in GeoTIFF format.

4.1.3. GDAL Translate using QGIS

QGIS functions as geographic information system (GIS) software, which users can easily analyze and edit spatial information, in addition to exporting and composing graphical maps. QGIS supports both raster and vector layers. Vector data stored as either point, line, or polygon features. [30]

The gdal_translate utility used to convert raster data between different formats, using some operations like sub settings, resampling, and rescaling pixels in the process.

NASS cropland data layer is a 30m resolution, whereas the RapidEye 5m resolution that gdal_translate used to transform the two raster images into the same projections.

4.1.4. Reading the Data

4.1.4.1. NumPy array

Also called a “ndarray,” short for N-dimensional array; describes memory using the following attributes:

- **data pointer**: the memory address of the first byte stored in the array;
- **data type description**: **the** kind of elements contained in the array, such as floating-point numbers or integers.

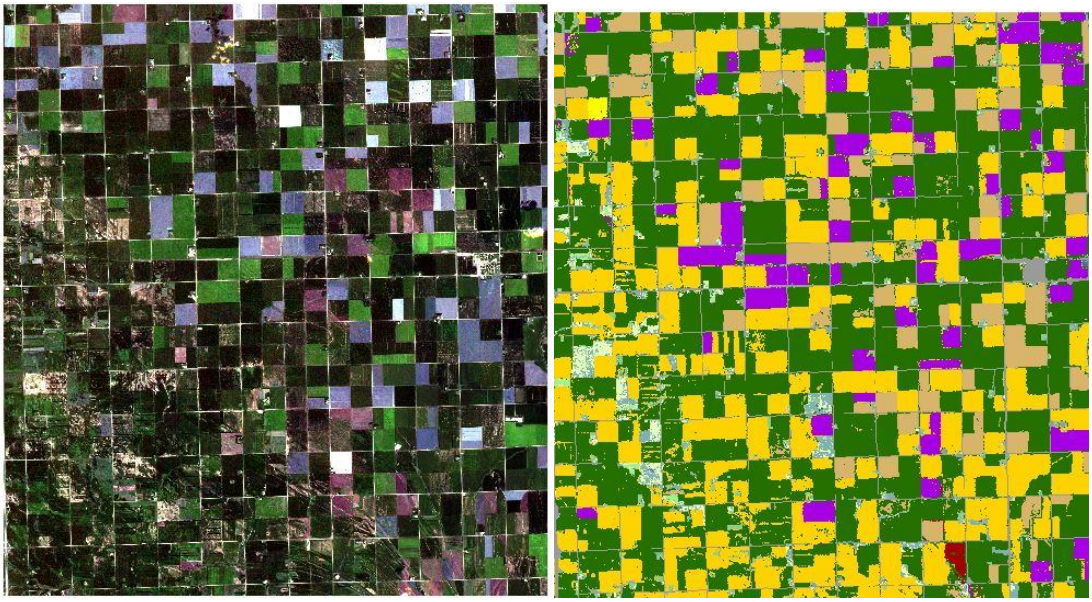

Figure 12: After GDAl translate & crop to the same projection

- **shape:** the **array's** shape, such as (10, 10) for a 10 × 10 array, or (5, 5, 5) for a chunk of data describing a mesh grid of x-, y-, and Z-coordinates.
- **Strides:** the **number** of bytes skipped in memory to proceed to the next element along a given dimension (for a (10, 10) array of bytes, for example, the strides might be (10, 1), or proceed one byte to get to the next column and 10 bytes to locate the next row).
- **Flags**: **define** factors such as whether modifing the array is allowed or not or whether memory layout is C- or Fortran-contiguous (in C, memory is laid out in "row major" order— that is, rows are stored one after another in memory—whereas, in Fortran, columns stored successively). [31]

4.1.4.2. Python Rasterio

Geographic information systems use GeoTIFF and other formats to establish and store raster datasets such as satellite imagery and terrain models. Rasterio reads and writes these formats and provides a Python API based on Numpy N-dimensional arrays and GeoJSON. [32]

After GDAL translate, crop & Rasterio read, the raster data returned in the form of NumPy array with the following shapes; the satellite image raster 4373 x 4021 x 5 which means 4373 x 4021 along with five bands of RGB, NIR(near-infrared) and Red Edge and the land cover image with the shape of 4373 x 4021, which is used for a pixel to pixel classification.

```
In [5]: print(rast_data.shape) # the rapideye satellite image data
        print(label_arr.shape) # NASS cropland cover label data

        (4373, 4021, 5)
        (4373, 4021)
```

Figure 13: NumPy array shapes of the raster data

4.1.5. Class (Cropland cover) Distribution

Table 1: Class distribution of the land cover per pixel

Class labels	**No. of Pixels per class**
5	8157040
1	4875365
23	1603794
41	1246206
121	740862
195	314006
176	217542
141	160854
122	70692
190	41274
37	44286
42	31896
123	28866
6	26730
111	14100
36	4176
124	3168
131	1188
61	1008
142	396
28	348
152	36

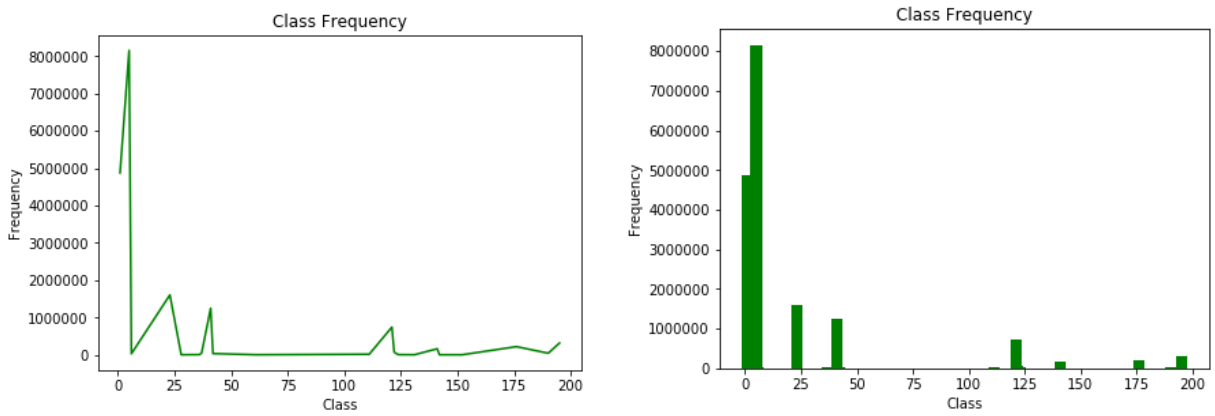

Figure 14: Class distribution of the land cover per pixel graph

4.2. Auxiliary Data Layer

Foremost of the objectives of this research is to create and add auxiliary data layers using geospatial data analysis and satellite data. Also, to cross-check that this additional information can improve the process of extracting information like land use classification, which is useful for the agriculture sector. Collecting data is expensive, but when additional layers produced through processing techniques alone, the value of the existing data enhanced without extra cost. [5]

For this research, the sliding window technique used to produce the auxiliary data layers. The generated layers are NDVI (Normalized Difference Vegetation Index), and Regression slopes the red and Near Infrared (NIR) bands of the original satellite image and then experimented on how these additional layers indeed provide information that is valuable towards predicting the cropland cover from the satellite image data.

4.2.1. Multi-Scalar Sliding Window Algorithm

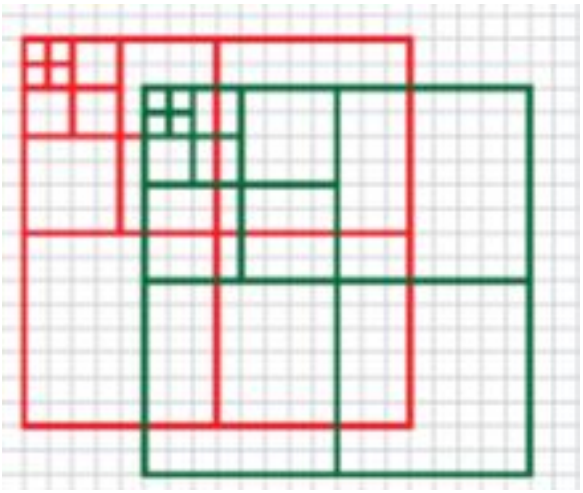

Figure 15: Schematic for aggregation of sub-windows within sliding windows.

In top-down derivation format, the bigger windows can be expressed in terms of the smaller windows, as shown in the above Figure 14, any bigger window is the composition of four sub-windows such that $Z_{i_0 j_0}^{w}$ calculated as linear aggregate of the four parts,

$$Z_{i_0 j_0}^{w} = \sum_{i=i_0}^{i_0+w-1} \sum_{j=j_0}^{j_0+w-1} Z_{ij}^{base}$$

$$= \sum_{i=i_0}^{i_0+w/2-1} \sum_{j=j_0}^{j_0+w/2-1} Z_{ij}^{base} + \sum_{i=i_0+w/2}^{i_0+w-1} \sum_{j=j_0}^{j_0+w/2-1} Z_{ij}^{base}$$

$$+ \sum_{i_0=0}^{i_0+w/2-1} \sum_{j=j_o+w/2}^{j_0+w-1} Z_{ij}^{base} + \sum_{i=i_o+w/2}^{i_0+w-1} \sum_{j=J_0+w/2}^{j_0+w-1} Z_{ij}^{base} \tag{4.1}$$

Remark that the index of each window is the index of its top left corner, i.e. $Z_{i_0 j_0}^{w}$ holds the aggregates for the window that has i_0 and j_0 as its smallest value of i and j respectively.

The definition requires that all values of Z_{ij}^{base} exist, which is only the case if windows don't extend beyond image boundaries, i.e. , $i_0+w-1 < n$ or $i_0 < n-w+1$. The number of

windows that are returned in each step decreases accordingly. A recursive definition for Z_{ij}^{w} follows by substituting the definition of $Z_{ij}^{w'}$ for $w' = w/2$ [5]

Where $Z_{ij}^{base} = any\ of\ \{x_{ij}, y_{ij}, x_{ij}^2, y_{ij}^2, x_{ij}y_{ij}\}$, $0 \leq i \leq n$ and $0 \leq j \leq m$, for an image of width n and height m.

4.2.2. NDVI (Normalized Difference Vegetation Index) Data Layer

Researchers often use a vegetation index called NDVI to measure the "greenness" or density of vegetation across a landscape. In addition to monitoring vegetation health, NDVI (Normalized Difference Vegetation Index) can be used to track climate change, agricultural production, desertification, and land cover change. Developed by NASA scientist Compton Tucker in 1977, NDVI is derived from satellite imagery and compares reflected near-infrared light to reflected visible red light. It expressed as the following equation:

$$NDVI = \frac{\rho_{NIR} - \rho_{Red}}{\rho_{NIR} + \rho_{Red}} \tag{4.2}$$

In general, healthy and dense vegetation reflects much near-infrared light and not as much red visible light. Conversely, when vegetation is sparse or not-so-healthy, its near-infrared reflectance decreases and its red-light reflectance increases. [33]

Steps of generating the NDVI date layer from the satellite image data:

1. Extracted the red and near-infrared bands data of the satellite data and represented in NumPy array form. In this step, Rasterio is used to open the raster image. Then extracted red and near-infrared bands data and load the band data into arrays that operated using Python's NumPy library.

2. Normalize the data Normalize the values in the arrays for each band using the Top of Atmosphere (TOA) reflectance coefficients stored in metadata of the RapidEye satellite data.
3. Generate the NDVI data layer. Calculate the NDVI using the above Equation 3.2.1
4. Add the NDVI date layer to the satellite data. Add the generated NDVI data layer to the satellite data that it becomes the 6^{th} channel or layer of information. Then, the shape of the satellite data as a Numpy array become 4373x4021x6.

4.2.3. Regression Slope Data Layer

The regression slope line of NIR band as a function the Red band of the satellite data sliding window representation is generated using the following equation;

$$slope = \frac{N\sum x_{ij}y_{ij} - \sum x_{ij}\sum y_{ij}}{N\sum x_{ij}^2 - \left(\sum x_{ij}\right)^2} \tag{4.3}$$

Where N= w^2 is the number of pixels in each window and *x* is the red band whereas *y* is the Near Infrared(nir) band and *i, j* are the pixel indexes. [5]

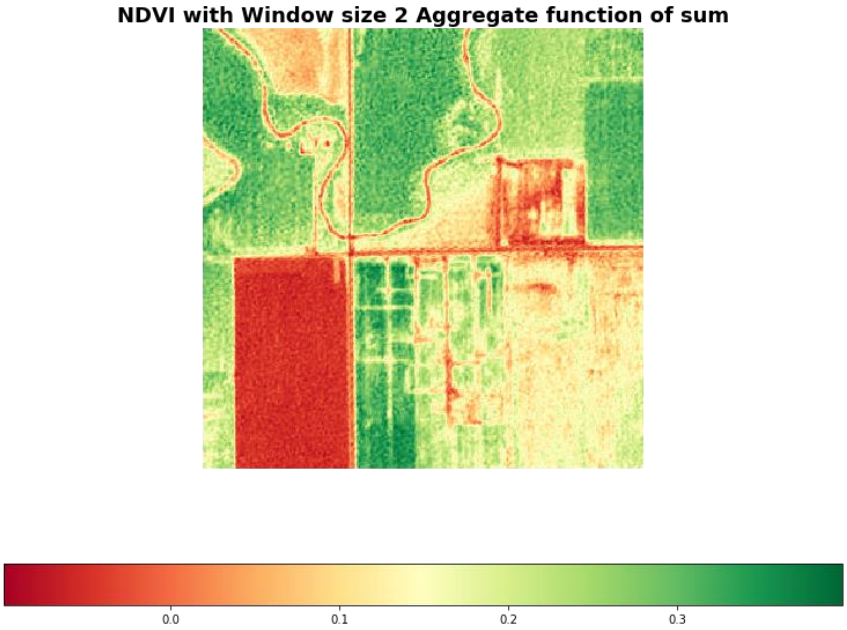

Figure 16: NDVI plot with window size 2 and sum as an aggregate function

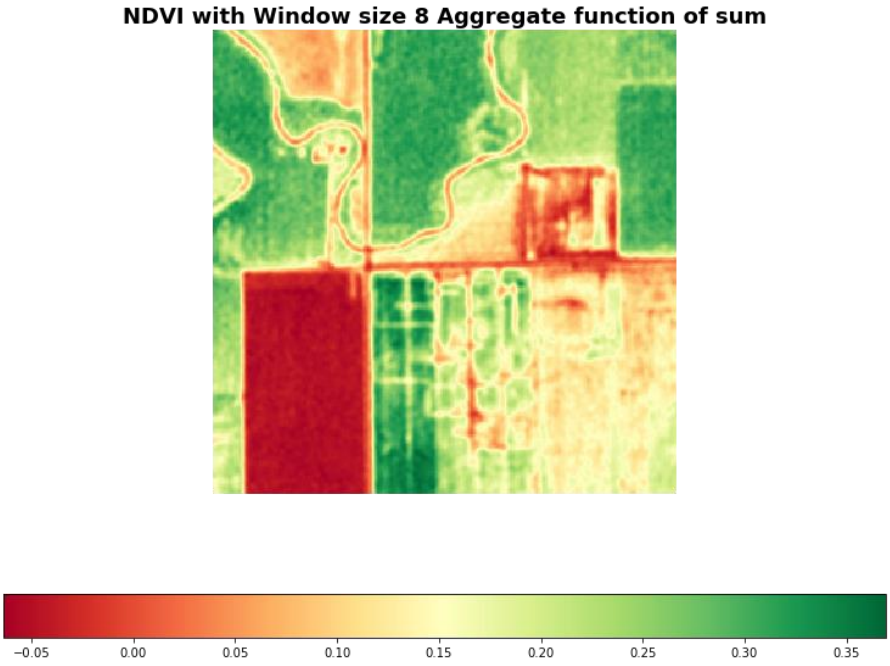

Figure 17: NDVI plot with window size 8 and sum as an aggregate function

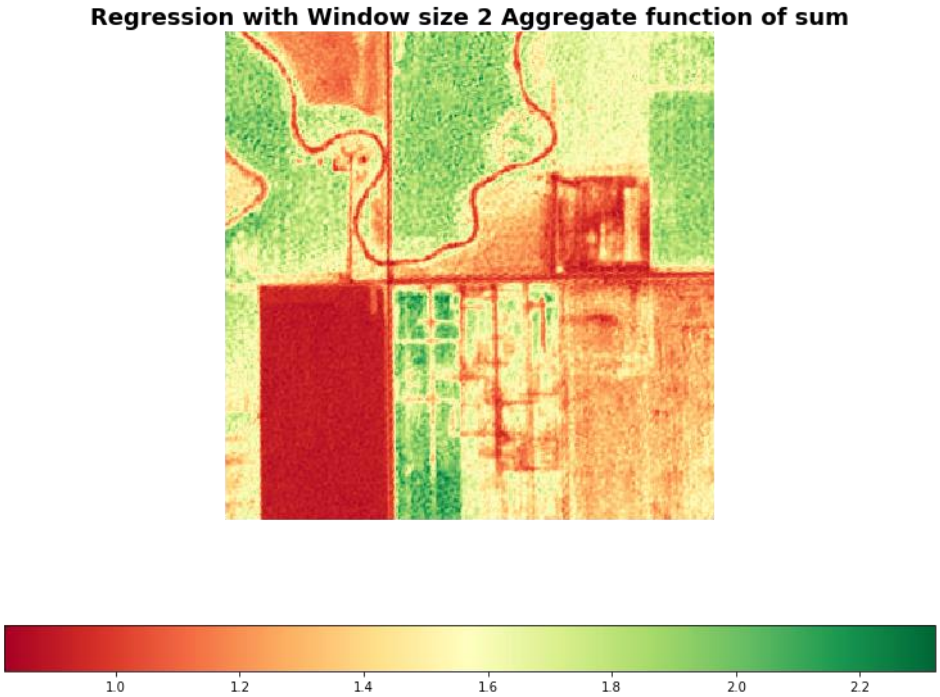

Figure 18: Regression plot with window size 2 and sum as an aggregate function

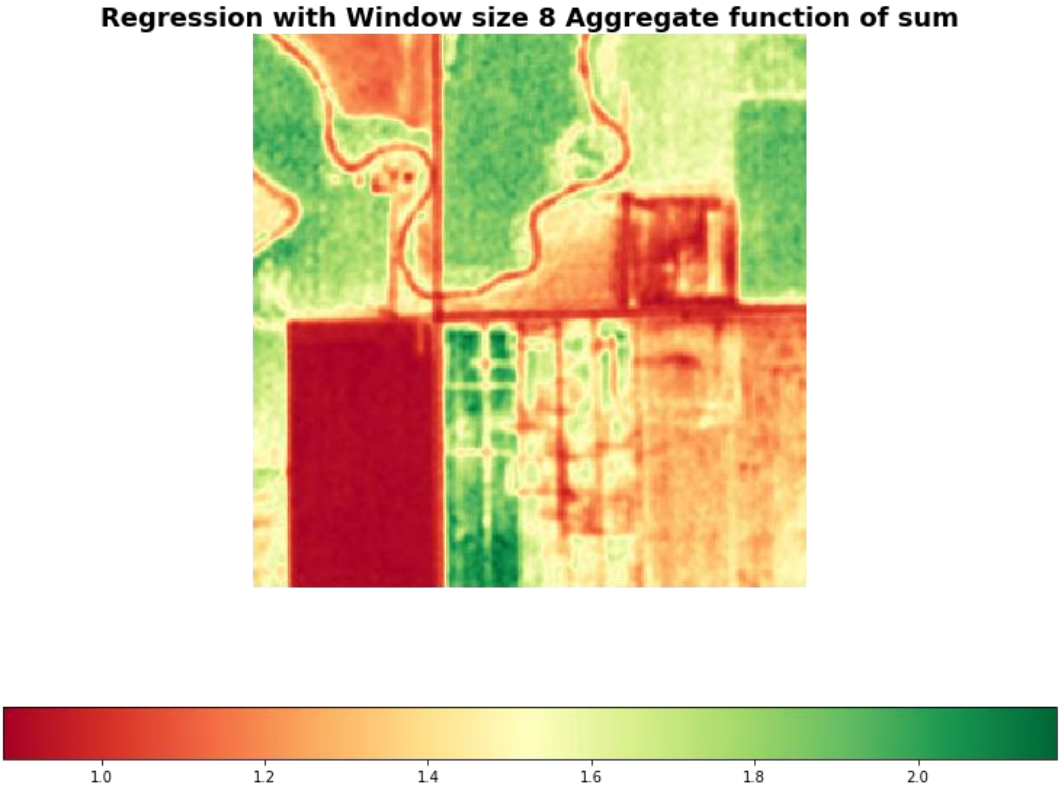

Figure 19: Regression plot with window size 8 and sum as an aggregate function

4.3. Data Sampling and Normalizing

4.3.1. Train Test Data Split

In Machine learning related problems, especially in classification type problem, the data split into training data and test data. The training set contains a known output/class label that the model learns on this data in order to be generalized to other data for later use. The test dataset (or validation) reserved in order to test our model's prediction on this subset. [34]

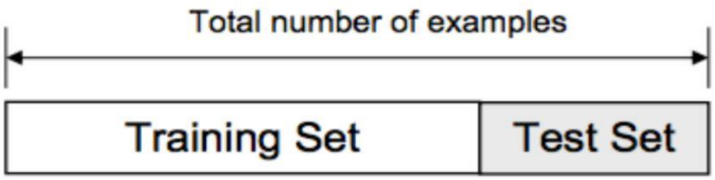

Figure 20: Train-Test split

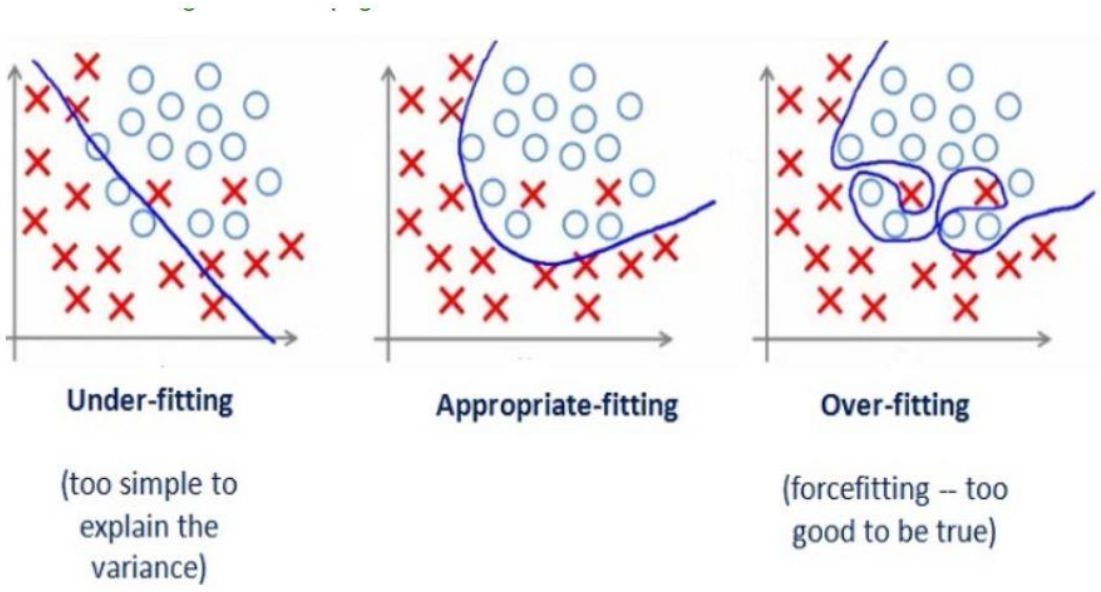

Figure 21: Different fitting of classifier model [35]

4.3.1.1. Overfitted Model

Refers to a model that models the training data too well. Overfitting occurs when a model learns the detail and noise in the training data to the level that it negatively affects the performance of the model on new data. Which means that the noise or random fluctuations in the

training data selected and learned as concepts by the model. The problem is that these notions do not apply to new data and negatively impact the model's capacity to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a goal function. As such, several nonparametric machine learning algorithms also contain parameters or techniques to limit and constrain how much detail the model learns. [35]

4.3.1.2. Underfitted Model

An underfit machine learning model is not a suitable model and will be evident as it will have a low performance on the training data.

Underfitting usually is not easy to detect given a useful performance metric. The solution is to move on and try other machine learning algorithms. [35]

Two essential ways that can be used to avoid a machine learning model from overfitting/underfitting:

1. Use a resampling technique to estimate model accuracy.
2. Hold back a validation dataset.

The most know resampling technique is **k-fold cross-validation**. It allows to train and test a model k-times on different portions of training data and build up an estimate of the performance of a machine learning model on unseen data.

On this research based on those reasons, K-fold cross-validation for sampling implemented.

4.3.1.3. K-fold Cross-Validation

Cross-validation is a resampling procedure used to gauge machine learning models on a limited and different data sample sets. The procedure has a single parameter called k, which

refers to the number of ways the data split into; this procedure is often called k-fold cross-validation. When a specific value for k chosen, it is used in place of k about the model, such as k=5(Which is the choice of k value in this research) becoming 5-fold cross-validation. Cross-validation primarily used in applied machine learning to estimation the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general cases. It is a widely used method because it is simple to understand, and it results in a less biased or less optimistic estimate of the model skill than other methods like simple train/test split. [36]

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For every single group:
    - One group taken as a holdout or test data set
    - the remaining groups used as a training data set
    - Fit a model with the training set and evaluate it using the test set
    - Retain the evaluation score and discard the model
- Summarize the efficiency of the model using the model evaluation scores samples.

Significantly, each instance in the data sample is assigned to a single group and stays in that group for the duration of the procedure. Which means that each sample is allowed to be used in the hold out set 1 time and used to train the model k-1 times. [36]

4.3.1.4. Variations on Cross-Validation

There are several variations on the k-fold cross-validation procedure.

The three frequently used variations are as follows:

- **Train/Test Split**: when k set to 1 such that a single train/test split to evaluate the model.
- **Stratified**: The splitting of data into folds determined by criteria such as guaranteeing that each fold has the same proportion of instance with a given class value, like the class outcome value. This is called stratified cross-validation.
- **LOOCV**: When k set to the total number of observations in the dataset such that each instance guaranteed to be included into held out of the dataset. This type of split known as leave-one-out cross-validation, or LOOCV for short.
- **Repeated**: This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample shuffled before each repetition, which results in a different split of the sample. [36]

On this research, Sci-kit Learn Train/Test split and Stratified K fold cross-validation techniques used.

4.3.2. Data Normalizing using Scikit-Learn StandardScaler

Normalization in Scikit-Learn often referred to as rescaling. Standardization of a dataset is a common condition for many machine learning estimators.

Scikit-Learn StandardScaler scales the data such that the distribution to be centered around 0, with a standard deviation of 1.

The standard score of sample *x* calculated as:

$$z = (x - u) / s \tag{4.4}$$

where, *u* is the mean of the training samples or *u=0* for mean set to **False**

s is the standard deviation of the training samples or *s=1* for std set to **False**.

4.4. Training Scikit-Learn Classifier with Data Sample

4.4.1. Sci-Kit Learn SVM

Steps to generate the model on Sci-kit Learn are

- Import the model
- Pass argument values like Kernel type.
- Fit the model on the training data set using fit( ).
- Perform prediction on the testing set using predict().

Some of the essential **Hyperparameters** that tuned for performance change comparison are:

- **Kernel**: The primary function of the kernel is to transform the given dataset input data into the required form. Mostly, transforming the data set from non-separable space to sparable space. There are various types of functions, such as linear, polynomial, and radial basis function (RBF). Polynomial and RBF are convenient for non-linear hyperplane. Polynomial and RBF kernels calculate the separation line in the higher dimension. In some of the applications, it suggested using a more complex kernel to separate the classes that are curved or nonlinear. This transformation can lead to more precise classifiers.
- **Regularization**: Regularization parameter in Python's Scikit-Learn denotes as C parameter. Here C is the penalty parameter, which gauges misclassification or error term. The misclassification or error term tells the SVM optimization how much error is bearable. A smaller value of C creates a small-margin hyperplane, and more substantial value of C generates a larger-margin hyperplane.
- **Gamma**: defines the degree of how the classification model fit the training dataset. A higher value of gamma implies that the model will exactly fit the training dataset and a

lower value of Gamma implies the model loosely fit the training dataset. In other words, a low value of gamma reflects only nearby points in calculating the separation line, while a value of gamma considers all the data points in the calculation of the separation line. [8]

4.4.2. Scikit-Learn K-Nearest Neighbor

First, import the KNeighborsClassifier module and create a KNN classifier object by passing the argument number of neighbors in KNeighborsClassifier() function. Then, fit the model on the train set using fit() and perform prediction on the test set using predict(). For KNN classifier normalizing the data on the same scale is highly recommended. The general normalization range considered to be between 0 and 1.

4.4.3. Scikit-Learn Random Forest

Generate a model on the selected training set features, perform predictions on the selected test set features, and compare actual and predicted values. Scikit-learn Random Forest classifier provides an extra variable with the model, which shows the relative importance or contribution of each attribute in the prediction. It automatically computes the relevance score of each feature in the training phase. Then it scales the relevance down so that the sum of all scores is 1. After generating the model, the feature importance can be easily visualized using the Scikit-Learn Random Forest model's feature importance variable.

4.4.4. Scikit-Learn Decision Tree

Decision tree algorithm fitted on training data, predicting labels for validation dataset, and printing the accuracy of the model using various parameters. **DecisionTreeClassifier()** is the Scikit-Learrn function for implementing the algorithms.

Some of the essential **Hyperparameters** that tuned for performance change comparison are:

- **Criterion:** It defines the function to measure the quality of a split. Sklearn supports "gini" criteria for Gini Index & "entropy" for Information Gain.
- **Splitter:** It defines the strategy to choose the split at each node. Supports "best" value to choose the best split & "random" to choose the best random split. By default, it takes "best" value.
- **max_features:** It defines the no. of features to consider when looking for the best split. We can input integer, float, string & *None* value.
- **max_depth:** The max_depth parameter denotes the maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, it takes "None" value. [24]

4.4.5. Scikit-Learn Extra Tree

In Scikit-Learn Extra-Tree classifier categorized under ensemble methods. The goal of **ensemble methods** is to associate the predictions of several base estimators built with a given learning algorithm in order to advance generalizability/robustness over a single estimator.

Some of the essential **Hyperparameters** that tuned for performance change comparison are:

- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
- By contrast, in **boosting methods**, base estimators are built sequentially, and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

In ExtraTreesClassifier (extremely randomized trees), randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features used, but instead of looking for the most discriminative thresholds, thresholds drawn at random for each candidate feature, and the best of these randomly generated thresholds picked as the splitting rule. This usually allows reducing the variance of the model a bit more, at the expense of a slightly more significant increase in bias.

Some of the crucial **Hyperparameters** that tuned for performance change comparison are:

- **n_estimators**: which is the number of trees in the forest. The larger, the better, but the running time will increase too.
- **max_features:** is the size of the random subsets of features to be considered when splitting a node. The lower, the more significant the reduction of variance, but also the more significant the increase in bias. [24]
- **Feature_importances:** is also one the most basic method of Scikit-Learn ExtraTreesClassifier method that used in this research, which returns the importance of the features that help to identify which feature contributed more for the classification task. In this research case, to check the importance of the additional sliding analysis feature for the overall classification task.

4.4.6. Scikit-Learn Gradient Boosting Classifier

Scikit-Learn Gradient Boosting Classifier (GB) is also under Scikit-Learn **ensemble methods.** GB builds an additive model in a forward stage-wise way; it permits for the optimization of random differentiable loss functions. In each stage, n classes_ regression

trees(where n is number of classes) are fit on the negative gradient of the binomial or multinomial deviance loss function.

Some of the crucial **Hyperparameters** that tuned for performance change comparison are:

- **learning_rate:** learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators. High learning_rate could result in overfitting.
- **n_estimators:** represents the number of trees in the forest. Usually, the higher the number of trees, the better to learn the data. However, adding many trees can slow down the training process considerably. [24]

4.4.7. Scikit-Learn Multi-Layer Perceptron

Scikit-Learn MLP (Multilayer Perceptron) is the simple Neural network form, and the implantation in Scikit-Learn is easy. First import the Multi-Layer Perceptron model from Scikit-Learn neural networks library.

Some of the important **Hyperparameters** that tuned for performance change comparison are:

- **hidden_layer_sizes:** characterizes the number of neurons in the ith hidden layer.
- **activation : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'**

  Activation function for the hidden layer.

  - 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
  - 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
  - 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
  - 'relu', the rectified linear unit function, returns f(x) = max(0, x)

- **alpha : float, optional, default 0.0001:** L2 penalty (regularization term) parameter.
- **solver : {'lbfgs', 'sgd', 'adam'}, default 'adam'**

  The solver for weight optimization.

  - 'lbfgs' is an optimizer in the family of quasi-Newton methods.
  - 'sgd' refers to stochastic gradient descent.
  - 'adam' refers to a stochastic gradient-based optimize
- **learning_rate : {'constant', 'in scaling', 'adaptive'}, default 'constant.'**

  Learning rate schedule for weight updates. Only used when solver='sgd'.

  - 'constant' is a constant learning rate given by 'learning_rate_init.'
  - 'in scaling' gradually decreases the learning rate at each time step 't' using an inverse scaling exponent of 'power_t.' effective_learning_rate =learning_rate_init / pow(t, power_t)
  - 'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing.
- **learning_rate_init: double, optional, default 0.001:** The initial learning rate used. It controls the step-size in updating the weights. Only used when solver='sgd' or 'Adam'.

4.5. Evaluate the Models

The test/validation set used to evaluate the performance of all the above-trained classifier and the following Scikit-Learn

- Scikit-Learn Accuracy Score
- Scikit-Learn Classification Report
- Scikit-Learn Cross Validation Prediction
- Feature Importance only used on Random Forest and Extra Tree Models

- Plots
    - Validation Curve
    - Learning Curve
- Gird Search

5. EXPERIMENT AND RESULTS

5.1. Parameter Configuration

I. Sliding Window-Based Aggregation Analysis: Different experiments done on generating the additional channel of information using the sliding window-based techniques for window size varying for w=2 to w=16. Effects of the additional information are almost similar for all window size. For results presented below the window size is 8.

II. Classifier's Parameter Configuration: Different experiments are done to identify the best configuration for each classifier model can generate best classification score for the data without the additional information that the substantial effect of the additional information cab easily distinguished.

The followings are the classifier model parameter configuration used on generating the result presented below.

5.1.1. SVM with RBF Kernel

```
scalar = StandardScaler()
clf = svm.SVC()
print('SVM with rbf Kernel')
parameter_grid2={'C': [1, 10, 100, 1000],'kernel': ['rbf']
                     ,'gamma': [1e-3, 1e-4]}
classifier_run(clf,parameter_grid2,scalar)
```

Figure 22: Code snippet of SVM (RBF) configuration

5.1.2. SVM with Linear Kernel

```
print('SVM with Linear Kernel')
parameter_grid={'C': [1, 10, 100, 1000],'kernel': ['linear']}
classifier_run(clf,parameter_grid,scalar)
```

Figure 23:Code snippet of SVM (Linear) configuration

5.1.3. KNN

```
scalar = StandardScaler()
clf =  KNeighborsClassifier(n_neighbors=3)
parameter_grid={'n_neighbors':[3,5,11,19],
                'weights':['uniform','distance'],
                'metric':['euclidean', 'l2', 'l1', 'manhattan']}
classifier_run(clf,parameter_grid,scalar)
```

Figure 24: Code snippet of KNN configuration

5.1.4. Random Forest

```
scalar = StandardScaler()
clf = RandomForestClassifier(n_estimators=10)
parameter_grid = {"max_features": [1, 3, 5 or 7 ],
                  "max_depth": [1, 2, 3, 4, 5],
                  "min_samples_split": [2, 3, 10],
                  "bootstrap": [True, False],
                  "criterion": ["gini", "entropy"]}
classifier_run(clf,parameter_grid,scalar,feature_importance=True)
```

Figure 25: Code snippet of Random Forest configuration

5.1.5. Decision Tree

```
scalar = StandardScaler()
clf = DecisionTreeClassifier()

parameter_grid = {'max_depth': [1, 2, 3, 4, 5],
                  'criterion': ['gini', 'entropy'],
                  'splitter': ['best', 'random'],
                  'max_features': [1, 2, 3, 4]}
classifier_run(clf,parameter_grid,scalar)
```

Figure 26: Code snippet of Decision Tree configuration

5.1.6. Extra Tree

```
scalar = StandardScaler()
clf = ExtraTreesClassifier()
print(clf.get_params().keys())
parameter_grid = {'max_depth': [1, 2, 3, 4, 5],
                  'criterion': ['gini', 'entropy'],
                  'splitter': ['best', 'random'],
                  'max_features': [1, 2, 3, 4]}
classifier_run(clf,parameter_grid,scalar,feature_importance=True)
```

Figure 27: Code snippet of Extra Tree configuration

5.1.7. Gradient Boosting

```
scalar = StandardScaler()
clf=GradientBoostingClassifier()
parameter_grid = {
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "loss":["deviance"],
    "min_samples_split": np.linspace(0.1, 0.5, 12),
    "min_samples_leaf": np.linspace(0.1, 0.5, 12),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse",  "mae"],
    "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
    "n_estimators":[10]
    }
classifier_run(clf,parameter_grid,scalar)
```

Figure 28: Code snippet of Gradient Boosting configuration

5.1.8. MLP

```
scalar = StandardScaler()
clf = MLPClassifier(max_iter=100)
parameter_grid = {
     'alpha': [0.0001,0.001,0.005,0.01,0.05],
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'learning_rate': ['constant','adaptive'],
}

classifier_run(clf,parameter_grid,scalar)
```

Figure 29: Code snippet of MLP configuration

5.2. Scikit-Learn Accuracy Score

This function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in the ground truth labels. [24]

Table 2: Scikit-learn accuracy score comparison of the auxiliary layer effect for different classifiers

Classifier Algorithms	**Accuracy without Sliding window analysis Auxiliary Layer**	**Accuracy with Sliding window analysis Auxiliary Layer + NDVI layer**	**Accuracy with Sliding window analysis Regression Layer Only**
SVM with RBF Kernel	47.19	60.84	58.33
SVM with linear Kernel	45.06	63.04	51.85
KNN	32.39	53.87	32.75
Random Forest	37.22	59.76	54.78
Decision Tree	30.89	51.61	46.57
Extra Tree	36.91	58.86	54.31
Gradient Boosting	45.11	63.11	58.48
MLP	47.19	62.43	60.59

5.3. Scikit-Learn Classification Report

Text summary of the precision, recall, F1 score for each class.

Table 3: Scikit-learn classifier report score comparison of the auxiliary layer effect for different classifiers

Classifier Algorithms	**without Sliding window analysis Auxiliary Layer**			**with Sliding window analysis Auxiliary Layer +NDVI**			**with Sliding window analysis Auxiliary Regression layer only**		
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
SVM with rbf Kernel	0.22	0.47	0.30	0.58	0.61	0.55	0.54	0.58	0.53
SVM with linear Kernel	0.32	0.45	0.32	0.58	0.63	0.6	0.30	0.52	0.38
KNN	0.31	0.32	0.31	0.52	0.54	0.52	0.31	0.33	0.31
Random Forest	0.31	0.37	0.34	0.56	0.6	0.57	0.50	0.55	0.52
Decision Tree	0.32	0.31	0.31	0.52	0.52	0.52	0.47	0.47	0.47
Extra Tree	0.31	0.37	0.34	0.55	0.59	0.57	0.50	0.54	0.52
Gradient Boosting	0.31	0.45	0.32	0.58	0.63	0.6	0.54	0.58	0.56
MLP	0.48	0.47	0.3	0.56	0.62	0.58	0.55	0.61	0.56

5.4. Scikit-Learn Cross Validation Prediction (cross_val_predict)

Scikit-Learn StraightedK folding used as cross-validation strategy on train-test data split. Scikit-Learn Cross Validation prediction also used to calculate the cross-validation score of the models. What cross_val_predict do is for K parts of that data such that i=1,…, k iterations:

- Save i'th part as the test data and all other parts as training data
- the model trained with training data (all parts except i'th)
- the trained model used to predict labels for ith part (test data)

In each iteration, the label of ith part of data predicted. In the end, cross_val_predict merges all partially predicted labels and returns them as the final result.

K=5 used for the reported result.

Table 4: Scikit-learn cross-validated score comparison of the auxiliary layer effect for different classifiers

Classifier Algorithms	**Cross Validated score without Sliding window analysis Auxiliary Layer**	**Cross Validated score with Sliding window analysis Auxiliary Layer + NDVI**
SVM with rbf Kernel	[0.47011952,0.47129306,0.47223612,0.47270906, 0.47315605]	[0.61005976,0.60409386,0.60630315, 0.61392088,0.60762669]
SVM with linear Kernel	[0.44472112, 0.4463305, 0.44922461, 0.45267902, 0.45910687]	[0.62998008, 0.62256615,0.62381191, 0.63445168,0.64024084]
KNN	[0.30677291, 0.32001997, 0.31315658, 0.34151227, 0.33818364]	[0.55278884, 0.52970544, 0.52976488, 0.54231347, 0.5388861]
Random Forest	[0.36503984, 0.37343984, 0.37718859, 0.38457687, 0.39136979]	[0.59511952, 0.59161258, 0.5887944, 0.58337506, 0.57952835]
Decision Tree	[0.30129482, 0.31652521, 0.31015508, 0.31196795, 0.31610637]	[0.5249004, 0.51123315, 0.52276138, 0.50676014, 0.5183141]
Extra Tree	[0.36304781, 0.37144284, 0.38369185, 0.37406109, 0.3733065]	[0.5876494, 0.5771343, 0.5837919, 0.58237356, 0.5915705]
Gradient Boosting	[0.44173307, 0.44682976, 0.44772386, 0.45368052, 0.45760161]	[0.63047809, 0.62256615, 0.62431216, 0.63495243, 0.64174611]
MLP	[0.47011952, 0.47129306, 0.47223612, 0.47220831, 0.47315605]	[0.62151394, 0.62656016, 0.62081041,0.64296445, 0.62418465]

5.5. Scikit-Learn Feature Importance

This Method use ensembles of decision trees (like Random Forest or Extra Trees) used to compute the relative importance of each attribute. These importance values used to identify the importance of the additional channels of information.

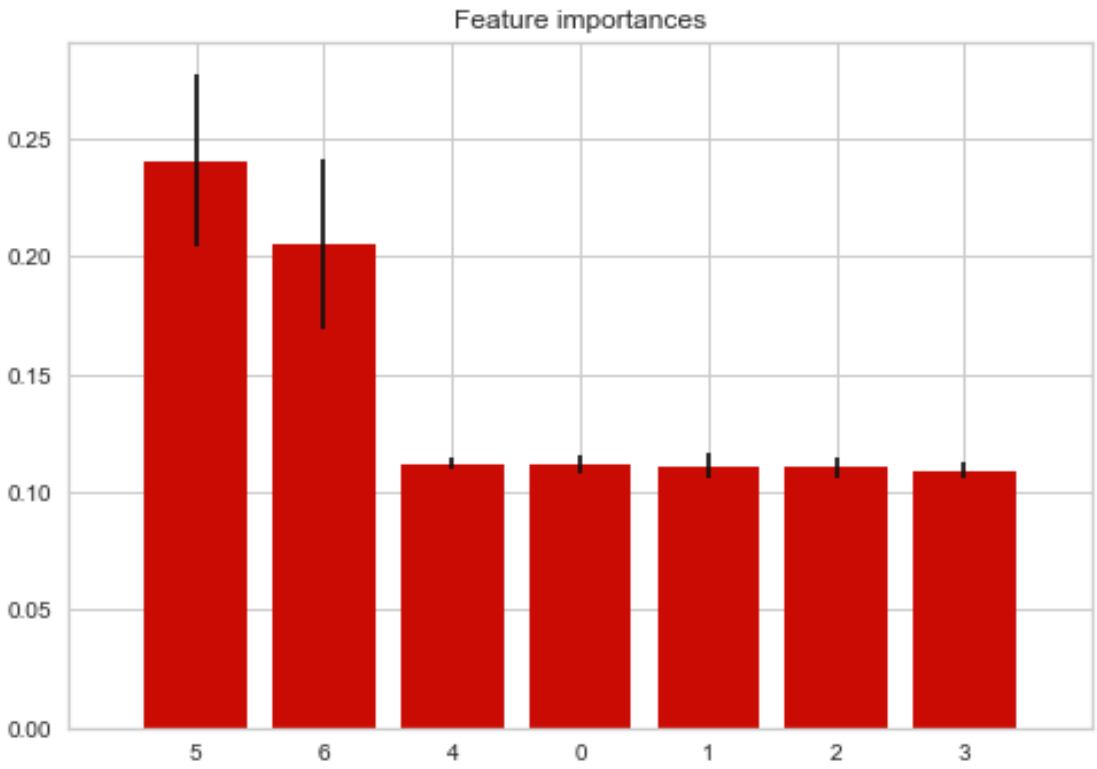

Figure 30: Random Forest feature importance graph

As shown in the graph above the added layer of NDVI (6) and Regression (5) plays a significant role in the classification tasks which strengthen the original premise.

Figure 23 also shows the same effect of the additional layer of information for Extra Tree classifier case such that, NDVI is feature number 6, and Regression layer is feature number 5.

Notice that generating this additional NDVI and Regression layer took **4.26ms**. However, considering the classification accuracy boos gained the extra generating time is very reasonable.

Figure 31: Extra Tree feature importance graph

5.6. Scikit-Learn Validation Curve

Compute scores for an estimator with different values of a specified parameter. However, this used to compute training scores and is merely a utility for plotting the results. This Validation curve plotting is done to examine whether the classifiers' parameters are contributing to the classification score boosting. The results show that the selected parameter has the same characteristics before and after the additional information.

5.6.1. SVM with RBF Kernel

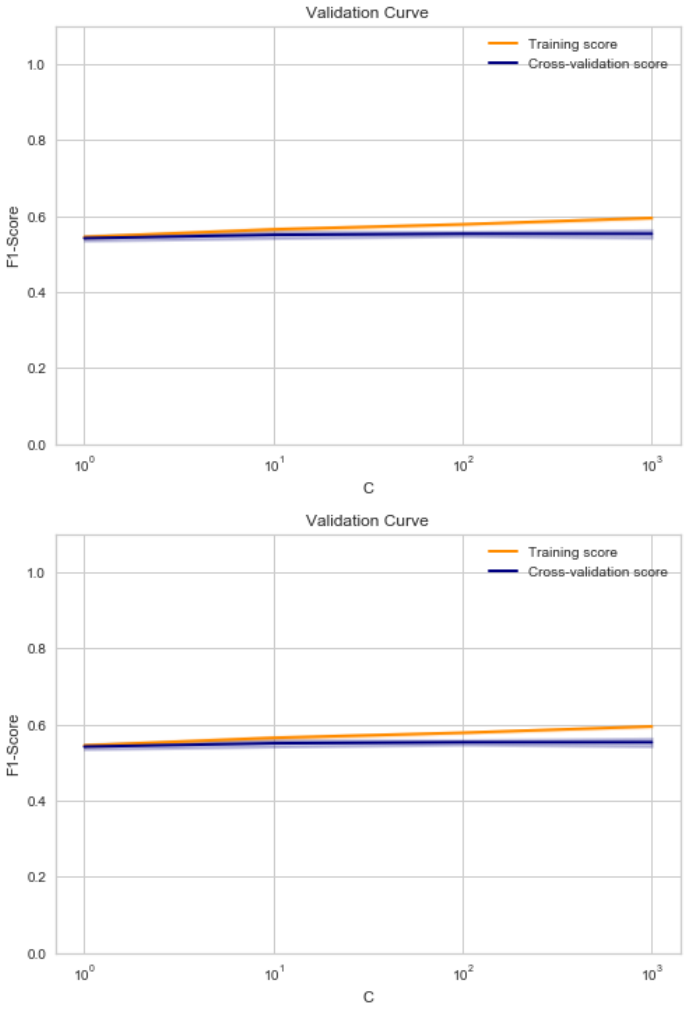

Figure 32: SVM RBF validation curve

5.6.2. KNN

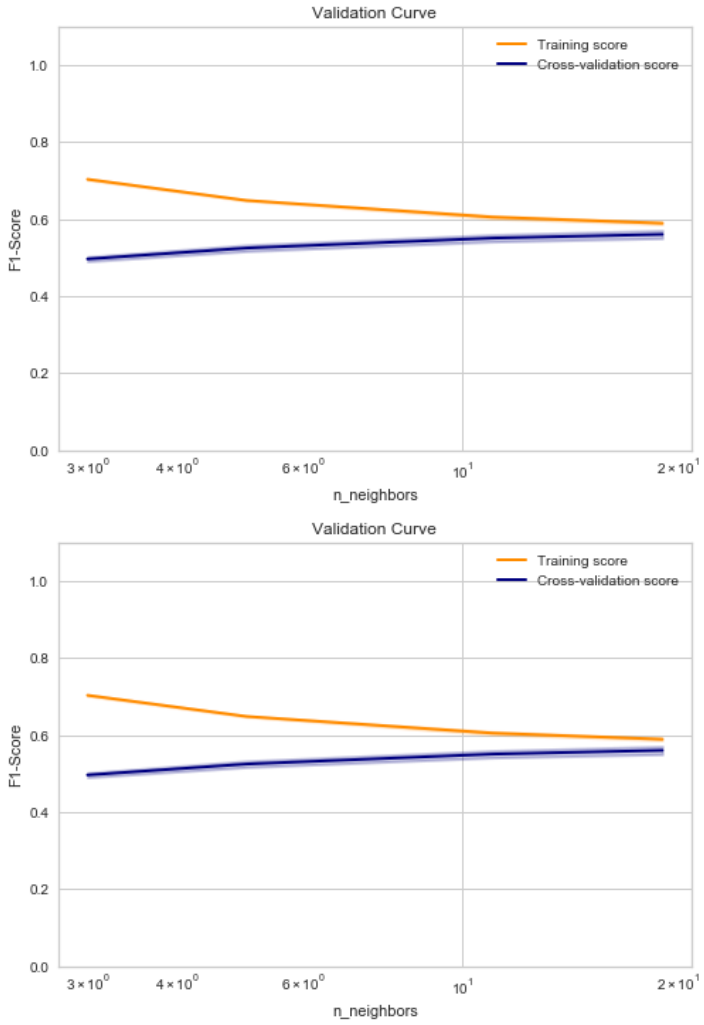

Figure 33: KNN validation curve

The above are some of the plotting of the validation curve presented to show that there is score change created by the selected parameters before and after the additional layer of information. The above classifier plots selected with no background reason and the other classifier plots are not present to avoid redundancy. The left plot is without the added layer, and the right plot is with the additional layer.

5.7. Scikit-Learn Learning Curve

A learning curve shows the validation and training score of an estimator for variable numbers of training samples. It is a tool to find out the advantage of adding more training data and whether the estimator suffers more from a variance error or a bias error. [24]

The learning curve is also plotted to show that the sample size does not affect the presented classification score boosting. Below, some of the learning curves presented as examples. The left plot is without the additional layer, and the right is with the additional layer.

5.7.1. Decision Tree

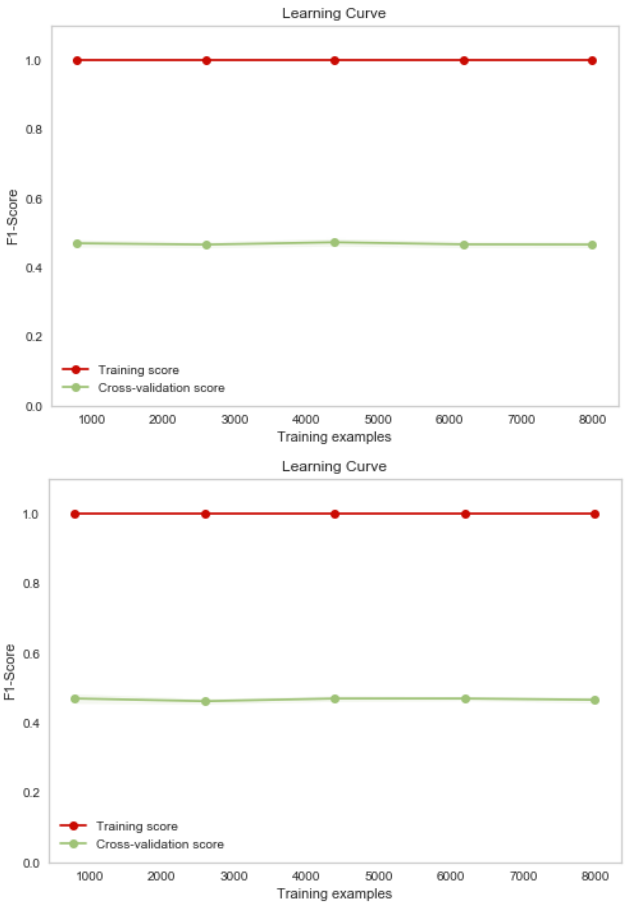

Figure 34: Decision Tree learning curve

5.7.2. Multi-Layer Perceptron (MLP)

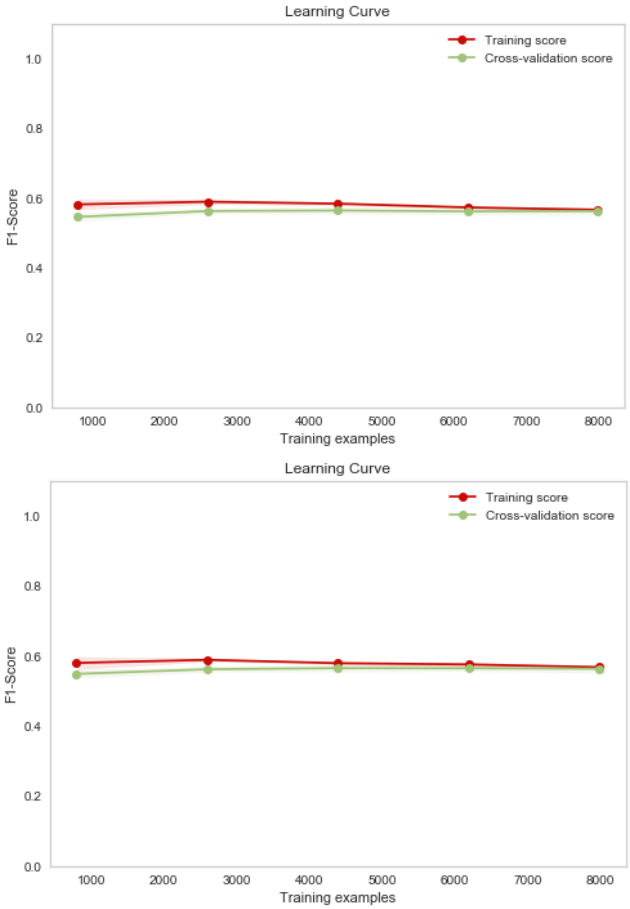

Figure 35: MLP learning curve

5. CONCLUSION AND FUTURE WORK

The whole approach of this research based on Triangulation Methodology, which states the combination of at least two or more methodological approaches, data sources, investigators, theoretical perspectives or data analysis methods lead to a firm conclusion. The intent of using triangulation is to increase the ability to interpret the findings by decreasing the deficiency of a single strategy. [37] That by running the generated data with and without the added layer of information on eight different classifiers, that conclusion is drawn on the significance of the additional layer.

We run the data against eight different classifiers to cross-check the importance of the added layer of information on predict cropland cover from satellite imagery of an area. The different contribution parameters of the classifier also examined plus, Validation, and learning curves during each classifier run to examine the data sample size variance effect. In general, all the experiments support the added layer is the main contributor to the increase of classification score almost 100% across for all classifiers. The feature importance attribute of Radom Forest and Extra Tree classifier solidified the above argument by ranking the added layer as the essential features for the classification task.

There are multiple advantages of using sliding window multi-scalar analysis in this of GIS to generate those additional layers. [5] [6] The main advantage of the multi-scalar analysis demonstrated in this research is the computation efficiency to generate those auxiliary layers which trickle down to the boosting the cropland cover prediction.

This work can be extended with specialized models that detect only 1-2 classes like cultivate and non-cultivate or water body and non-water body. The other area that can be

improved the data distribution, better data sampling and selection for balanced class distribution can quickly improve the performances of the prediction models.

Deep Learning would be the best approach for this kind multiclass classification problem. Unique neural network models like U-Net based segmentation and Mask RCNN based instance segmentation will be dominant on this kind of problem. U-Net is designed as an auto-encoder. It has an encoding path (contracting) coupled with a decoding path (expanding) which gives it the "U" shape. U-Net predicts a pixel-wise segmentation map of the input image which means regenerating the lable image rather than classifying the input image as a whole. For each pixel in the original image, it asks the question: "To which class does this pixel belong?". U-Net allows the feature maps from each level of the contracting path over to the comparable level in the expanding path. Mask RCNN (Mask Region-based Convolutional Neural Network) is an extension to Faster R-CNN that adds a branch for predicting an object mask in parallel with the existing branch for object detection. [38] Using these two Neural Network model with the additional layers of information from sliding window multi-scalar analysis would be very accurate system on problems like cropland cover prediction, different soli studies or framing practice identification [38].

REFERENCES

[1] M.J. Barnsley, S.L. Barr, "Inferring Urban Land Use from Satellite Sensor Images Using Kernel-Based Spatial Reclassification," *Photogrammetric Engineering & Remote Sensing,* vol. 62, pp. 949-958, August 1996.

[2] Dengsheng Lu, Qihao Weng, "Use of impervious surface in urban land-use classification," *Remote Sensing of Environment,* vol. 102, no. 1-2, pp. 146-160, May 2006.

[3] T.Kavzoglu, I.Colkesen, "A kernel functions analysis for support vector machines for land cover classification," *International Journal of Applied Earth Observation and Geoinformation,* vol. 11, no. 5, pp. 352-359, 2009.

[4] C. Huang, L. S. Davis & J. R. Townshend, "An assessment of support vector machines for land cover classification," *International Journal of Remote Sensing,* vol. 23, no. 4, pp. 725-749, Nov 2010.

[5] Anne M. Denton, Mostofa Ahsan, David Franzen, John Nowatzki, "Multi-scalar Analysis of Geospatial Agricultural Data for Sustainability," *IEEE International Conference on Big Data (Big Data),* Vols. 5-8, pp. 2139-2146, Dec 2016.

[6] Rahul Gomes, Anne Denton, David Franzen, "Quantifying Efficiency of Sliding-Window Based Aggregation Technique by Using Predictive Modeling on Landform Attributes Derived from DEM and NDVI," *International Journal of Geo-Informatiom,* vol. 8, p. 196, April 2019.

[7] Bernhard Scholkopf, Alexander J. Smola, Learning with Kernels Support Vector Machines, Regularization, Optimization, and Beyond, Cambridge, Massachusetts: The MIT Press, 2002.

[8] A. Navlani, "Support Vector Machines with Scikit-learn," Data Camp, 2018.

[9] P´adraig Cunningham, Sarah Jane Delany, "k-Nearest Neighbour Classifiers," UCD-CSI-2007-4, March 27, 2007.

[10] Kilian Q. Weinberger, John Blitzer and Lawrence K. Saul, "Distance Metric Learning for Large Margin Nearest Neighbor Classification," *The Journal of Machine Learning Research,* vol. 10, pp. 207-244, Sep 2009.

[11] A. Navlani, "KNN Classification using Scikit-learn," Data Camp, August 2018.

[12] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Introduction to Data Mining, Boston MA: Pearson Education Inc., 2006.

[13] T. Segaran, Programming Collective Intelligence 1st Edition, Sebastopol, CA: O' Reilly Media, Inc., 2007.

[14] S.R. Safavian, D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 21, no. 3, pp. 660-674, 1991.

[15] S.M. Weiss, C.A. Kulikowski, Computer systems that learn, San Mateo, CA: Morgan Kaufman Publishers, 1991.

[16] B. L, "Random forests—random features," University of California, Berkeley, CA, 1999.

[17] A. Navlani, "Decision Tree Classification in Python," Data Camp, 2018.

[18] Pierre Geurts, Damien Ernst, Louis Wehenke, "Extremely randomized trees," *Machine Learning,* vol. 63, no. 1, pp. 3-42, 2006.

[19] Garrido, Ana Porras, "What is the difference between Bagging and Boosting?," QuantDare, Madrid, Spain, 2016.

[20] Terence Parr, Jeremy Howard, "How to explain gradient boosting," Explained.ai.

[21] Marvin Minsky, Seymour Papen, Perceptrons An Introduction to Computational Geometry, Cambridge, MA: MIT Press, 2017.

[22] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems, Sebastopol, CA : O'Reilly Media, Inc., 2017.

[23] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," *Proceedings of the 14th international joint conference on Artificial intelligence,* vol. 2, pp. 1137-1143, 1995.

[24] Pedregosa, F. , Varoquaux, G., Gramfort, A. , Michel, V. , Thirion, B. , Grisel, O. , Blondel, M., Prettenhofer, P., Weiss, R. , Dubourg, V. , Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M. and Perrot, M., Duchesnay, E., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[25] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* 2011.

[26] J. VanderPlas, Python Data Science Handbook, O'Reilly Media, Inc, 2017.

[27] I. Harris Geospatial Solutions, "SENSORS AND SATELLITES: RapidEye," https://www.harrisgeospatial.com/Data-Imagery/Satellite-Imagery/Medium-Resolution/RapidEye, 2019.

[28] U. NASS, "USDA National Agricultural Statistics Service: Research and Science CropScape and Cropland Data Layers," https://www.nass.usda.gov/Research_and_Science/Cropland/sarsfaqs2.php#Section1_5.0, 2019.

[29] Sk. Sazid Mahammad, R. Ramakrishnan, "GeoTIFF - A standard image file format for GIS applications," Geospatial Media & Communications, 2009.

[30] QGIS Development Team, "QGIS 2.18 Documentation," https://docs.qgis.org/2.18/en/docs/, 2016.

[31] Stéfan van der Walt, S. Chris Colbert, Gaël Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science and Engineering,* vol. 13, no. 2, pp. 22-30, 2011.

[32] S. Gillies, "rasterio Documentation," MapBox, July 23, 2019.

[33] Nate Weisiger, Jennifer Reiber Kyle, Dana Bauer, "Deriving a vegetation index from PlanetScope imagery notebook," Github, 2018.

[34] A. Bronshtein, "Train/Test Split and Cross Validation in Python," Medium, 2017.

[35] A. Bhande, "What is underfitting and overfitting in machine learning and how to deal with it," Medium, 2018.

[36] J. Brownlee, "A Gentle Introduction to k-fold Cross-Validation," Machine Learing Mastery, 2018.

[37] Thurmond, Veronica A., "The Point of Triangulation," *Nursing Scholarship,* vol. 33, no. 3, pp. 253-258, 2001.

[38] Ami Turgman, Amy Boyd, Anastasia Zolochevska, Andrei Ermilov, Ari Bornstein, Beat Schwegler, "Satellite Images Segmentation and Sustainable Farming," https://www.microsoft.com/developerblog/2018/07/05/satellite-images-segmentation-sustainable-farming/, July 5, 2018.

[39] Compton J. Tucker, John R. G. Townshend, Thomas E. Goff, "African Land-Cover Classification Using Satellite Data," *Science,* vol. 227, no. 4685.369, pp. 369-75, 1 Feburary 1985.

APPENDIX. CROPLAND CLASS TABLE

Value	Color	Description
1	ffd300	Corn
2	ff2626	Cotton
3	00a8e2	Rice
4	ff9e0a	Sorghum
5	267000	Soybeans
6	ffff00	Sunflower
10	70a500	Peanuts
11	00af49	Tobacco
12	dda50a	Sweet Corn
13	dda50a	Pop or Orn Corn
14	7cd3ff	Mint
21	e2007c	Barley
22	896054	Durum Wheat
23	d8b56b	Spring Wheat
24	a57000	Winter Wheat
25	d69ebc	Other Small Grains
26	707000	Dbl Crop WinWht/Soybeans
27	aa007c	Rye
28	a05989	Oats
29	700049	Millet
30	d69ebc	Speltz
31	d1ff00	Canola

Value	Color	Description
32	7c99ff	Flaxseed
33	d6d600	Safflower
34	d1ff00	Rape Seed
35	00af49	Mustard
36	ffa5e2	Alfalfa
37	a5f28c	Other Hay/Non Alfalfa
38	00af49	Camelina
39	d69ebc	Buckwheat
41	a800e2	Sugarbeets
42	a50000	Dry Beans
43	702600	Potatoes
44	00af49	Other Crops
45	af7cff	Sugarcane
46	702600	Sweet Potatoes
47	ff6666	Misc Vegs & Fruits
48	ff6666	Watermelons
49	ffcc66	Onions
50	ff6666	Cucumbers
51	00af49	Chick Peas
52	00ddaf	Lentils
53	54ff00	Peas
54	f2a377	Tomatoes
55	ff6666	Caneberries
56	00af49	Hops

Value	Color	Description
57	7cd3ff	Herbs
58	e8bfff	Clover/Wildflowers
59	afffdd	Sod/Grass Seed
60	00af49	Switchgrass
61	bfbf77	Fallow/Idle Cropland
63	93cc93	Forest
64	c6d69e	Shrubland
65	ccbfa3	Barren
66	ff00ff	Cherries
67	ff8eaa	Peaches
68	ba004f	Apples
69	704489	Grapes
70	007777	Christmas Trees
71	af9970	Other Tree Crops
72	ffff7c	Citrus
74	b5705b	Pecans
75	00a582	Almonds
76	e8d6af	Walnuts
77	af9970	Pears
81	f2f2f2	Clouds/No Data
82	999999	Developed
83	4970a3	Water
87	7cafaf	Wetlands
88	e8ffbf	Nonag/Undefined

Value	Color	Description
92	00ffff	Aquaculture
111	4970a3	Open Water
112	d3e2f9	Perennial Ice/Snow
121	999999	Developed/Open Space
122	999999	Developed/Low Intensity
123	999999	Developed/Med Intensity
124	999999	Developed/High Intensity
131	ccbfa3	Barren
141	93cc93	Deciduous Forest
142	93cc93	Evergreen Forest
143	93cc93	Mixed Forest
152	c6d69e	Shrubland
176	e8ffbf	Grassland/Pasture
190	7cafaf	Woody Wetlands
195	7cafaf	Herbaceous Wetlands
204	00ff8c	Pistachios
205	d69ebc	Triticale
206	ff6666	Carrots
207	ff6666	Asparagus
208	ff6666	Garlic
209	ff6666	Cantaloupes
210	ff8eaa	Prunes
211	334933	Olives
212	e27026	Oranges

Value	Color	Description
213	ff6666	Honeydew Melons
214	ff6666	Broccoli
216	ff6666	Peppers
217	af9970	Pomegranates
218	ff8eaa	Nectarines
219	ff6666	Greens
220	ff8eaa	Plums
221	ff6666	Strawberries
222	ff6666	Squash
223	ff8eaa	Apricots
224	00af49	Vetch
225	ffd300	Dbl Crop WinWht/Corn
226	ffd300	Dbl Crop Oats/Corn
227	ff6666	Lettuce
229	ff6666	Pumpkins
230	896054	Dbl Crop Lettuce/Durum Wht
231	ff6666	Dbl Crop Lettuce/Cantaloupe
232	ff2626	Dbl Crop Lettuce/Cotton
233	e2007c	Dbl Crop Lettuce/Barley
234	ff9e0a	Dbl Crop Durum Wht/Sorghum
235	ff9e0a	Dbl Crop Barley/Sorghum
236	a57000	Dbl Crop WinWht/Sorghum
237	ffd300	Dbl Crop Barley/Corn
238	a57000	Dbl Crop WinWht/Cotton

Value	Color	Description
239	267000	Dbl Crop Soybeans/Cotton
240	267000	Dbl Crop Soybeans/Oats
241	ffd300	Dbl Crop Corn/Soybeans
242	000099	Blueberries
243	ff6666	Cabbage
244	ff6666	Cauliflower
245	ff6666	Celery
246	ff6666	Radishes
247	ff6666	Turnips
248	ff6666	Eggplants
249	ff6666	Gourds
250	ff6666	Cranberries
254	267000	Dbl Crop Barley/Soybeans