Development of image classification toolkit for remote sensing in google earth engine

by

Devon Lee Garcia

B.A., Kansas State University, 2018

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF ARTS

Department of Geography and Geospatial Sciences
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2022

Approved by:

Major Professor
Douglas G. Goodin

Copyright

Abstract

Google Earth Engines provides accessibility to plentiful databases for image acquisition and analysis, while also providing tools and a user-friendly API to accomplish the tasks at hand. Utilizing Google Earth Engine's analysis and tool creation capability the aim of the study was to create a tool to monitor landcover change within the Chobe District in Botswana that can be easily executable from users with minimal expertise in the remote sensing field. The three machine learning techniques used are Naïve Bayes, Support Vector Machine, and Random Forest which are paired alongside pooled sampling to create the tool. Z-Test and McNemar's test are two quantitative methods used to compare each classifier's performance from the resulting overall accuracy and Kappa value which is also calculated from within Google Earth Engine. Qualitative analysis methods are then used to compare the results of the best performing classifier from Google Earth Engine and results from a similar study which creates a landcover classifier for the same region on ArcGIS.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Dr. Douglas Goodin for the advising me through this thesis as well as helping edit this research paper countless amounts of times. I would also like to thank my committee members Dr. Jida Wang and Dr. Shawn Hutchinson for helping me decide on which direction I should take this project by suggesting exploring Google Earth Engines.

I would also like to thank my family and friends for their endless support throughout this adventure in my life – especially my mother, Mayra E. Hernandez, for being the rock in my life as well as giving me the strength and courage to continue with advancing my knowledge in school. I would like to thank my friends Katrina Yau, Talha Shahryar, William Smith, Gordon Chi, and Kim Roxas for being there for me as well as being a sound board throughout the writing process of my thesis.

I would also like to thank Kathy Alexander of Virginia Tech University for allowing me to contribute to her ongoing research in the Chobe River Basin.

Chapter 1 - Introduction and Overview of Google Earth Engine

As technology evolves, available datasets grow and become more complex. The necessity of expanding access and usability of existing methods to harness the knowledge within these datasets has become more apparent (Vellido Alcacena et al.,2011). This is especially true for geospatial analysis and earth observation technologies, where the available data source and analytical capabilities have significantly expanded in scope. Gathering data for processing is one of the more tedious aspects of performing analysis, especially with the increasing size of datasets. Having ease of access to these datasets for analyzation purposes could potentially increase the productivity and efficiency of analysts, researchers, and users that utilize software for remote sensing tasks.

With the increasing interest in geospatial analysis for both research and applications it is best to get a comprehensive understanding of what tools are available. A tool that is easy to use with little to no experience would be beneficial for non-specialist users of remote sensing applications. There is a plethora of software that can be used for image analysis and comparisons but finding one that meets the criteria for what analysts want to achieve seems to be particularly complex. Analysts have numerous hurdles to face when trying to find the right tool for their work, including access to the software and computational limitations. Even when a combination of access to the software and computational access are available, it is still one of the main priorities to learn how to execute commands and processes, properly and efficiently, upon the software of choice. Searching for guides or lessons on how to work in certain integrated development environments (IDE) can be difficult.

Even with proper training and technique, the process of image classification can be arduous and time consuming solely based on the image collection process and computation time.

The innovation of a cloud-based software could be beneficial to analysts, especially those with limited access to data and computational resources or who may lack extensive training in the application of analysis tools. Expanding the accessibility and usability of these specialized applications for application-oriented users is an ideal goal. Given the ability to gather data and access to operational tools within one application, that can be automated, could possibly decrease the amount of time allotted for image gathering, classification, and the occasional update so that performing earth observational tasks can be more accessible to non-specialist users.

1.1 Overview of Google Earth Engine

Google Earth Engine (GEE) is a platform that is free for educational and research purposes and contains imagery accessible via cloud-based servers that increase accessibility and is user friendly when compared to conventional image collection and analysis tools. It therefore has the potential to address the above-mentioned issues with data and software access by giving users access to numerous amounts of datasets and built-in analytical tools. The platform gives users the ability to process geospatial image and raster data and see their results efficiently and has a user-friendly user-interface (UI). GEE allows users to analyze data from public catalogs as well as their own private data using a library of operators provided by Earth Engine API (Gorelick et al., 2017).

Effective application of GEE requires some knowledge on the part of the user which include coding in the specified language that GEE utilizes and knowledge of analytical methods and techniques that lead to the desired output of the program. Basic information technology skills such as, data acquisition and storage, parsing obscure file formats, managing datasets, machine allocations, jobs and job queues, CPUs, GPUs, and networking are required to take full

advantage of the resources provided by GEE (Gorelick et al., 2017). Creation of a basic toolkit for new users will help those explore the capabilities of GEE and could potentially open possibilities of multiple uses for the base toolkit.

1.1.1 Image Datasets

There are various types of satellite image collections with various sensors and formats that are used throughout the remote sensing research community. Table 1.1.1 displays how many unique datasets are available within Google Earth Engine. Many of these datasets are open to download from the respective organizations, but the compilation of all these datasets within a cloud-based software was imperative to this study for image collection and analysis.

Datasets containing images produced by a single sensor within GEE are grouped together and presented as a "collection". (Gorelick et al., 2017). These collections help users effectively and efficiently access the desired dataset within GEE, with the assumption that users have knowledge of which specific dataset that they desire for use and have the proper knowledge and skill on how to access them.

Table 1.1.1. Datasets within Google Earth Engine (Gorelick et al., 2017)

Dataset	Nominal resolution	Temporal granularity	Temporal coverage	Spatial coverage
Landsat				
Landsat 8 OLI/TIRS	30 m	16 day	2013–Now	Global
Landsat 7 ETM +	30 m	16 day	2000–Now	Global
Landsat 5 TM	30 m	16 day	1984–2012	Global
Landsat 4–8 surface reflectance	30 m	16 day	1984–Now	Global
Sentinel				
Sentinel 1 A/B ground range detected	10 m	6 day	2014–Now	Global

Dataset	Nominal resolution	Temporal granularity	Temporal coverage	Spatial coverage
Sentinel 2A MSI	10/20 m	10 day	2015–Now	Global
MODIS				
MOD08 atmosphere	1°	Daily	2000–Now	Global
MOD09 surface reflectance	500 m	1 day/8 day	2000–Now	Global
MOD10 snow cover	500 m	1 day	2000–Now	Global
MOD11 temperature and emissivity	1000 m	1 day/8 day	2000–Now	Global
MCD12 Land cover	500 m	Annual	2000–Now	Global
MOD13 Vegetation indices	500/250 m	16 day	2000–Now	Global
MOD14 Thermal anomalies & fire	1000 m	8 day	2000–Now	Global
MCD15 Leaf area index/FPAR	500 m	4 day	2000–Now	Global
MOD17 Gross primary productivity	500 m	8 day	2000–Now	Global
MCD43 BRDF-adjusted reflectance	1000/500 m	8 day/16 day	2000–Now	Global
MOD44 veg. cover conversion	250 m	Annual	2000–Now	Global
MCD45 thermal anomalies and fire	500 m	30 day	2000–Now	Global
ASTER				
L1 T radiance	15/30/90 m	1 day	2000–Now	Global
Global emissivity	100 m	Once	2000–2010	Global
Other imagery				
PROBA-V top of canopy reflectance	100/300 m	2 day	2013–Now	Global
EO-1 hyperion hyperspectral radiance	30 m	Targeted	2001–Now	Global
DMSP-OLS nighttime lights	1 km	Annual	1992–2013	Global
USDA NAIP aerial imagery	1 m	Sub-annual	2003–2015	CONUS
Topography				

Dataset	Nominal resolution	Temporal granularity	Temporal coverage	Spatial coverage
Shuttle Radar Topography Mission	30 m	Single	2000	60°N–54°S
USGS National Elevation Dataset	10 m	Single	Multiple	United States
USGS GMTED2010	7.5″	Single	Multiple	83°N–57°S
GTOPO30	30″	Single	Multiple	Global
ETOPO1	1′	Single	Multiple	Global
Landcover				
GlobCover	300 m	Non-periodic	2009	90°N–65°S
USGS National Landcover Database	30 m	Non-periodic	1992–2011	CONUS
UMD global forest change	30 m	Annual	2000–2014	80°N–57°S
JRC global surface water	30 m	Monthly	1984–2015	78°N–60°S
GLCF tree cover	30 m	5 year	2000–2010	Global
USDA NASS cropland data layer	30 m	Annual	1997–2015	CONUS
Weather, precipitation & atmosphere				
Global precipitation measurement	6′	3 h	2014–Now	Global
TRMM 3B42 precipitation	15′	3 h	1998–2015	50°N–50°S
CHIRPS precipitation	3′	5 day	1981–Now	50°N–50°S
NLDAS-2	7.5′	1 h	1979–Now	North America
GLDAS-2	15′	3 h	1948–2010	Global
NCEP reanalysis	2.5°	6 h	1948–Now	Global
ORNL DAYMET weather	1 km	Annual	1980–Now	North America
GRIDMET	4 km	1 day	1979–Now	CONUS
NCEP global forecast system	15′	6 h	2015–Now	Global
NCEP climate forecast system	12′	6 h	1979–Now	Global
WorldClim	30″	12 images	1960–1990	Global

Dataset	Nominal resolution	Temporal granularity	Temporal coverage	Spatial coverage
NEX downscaled climate projections	1 km	1 day	1950–2099	North America
Population				
WorldPop	100 m	5 year	Multiple	2010–2015
GPWv4	30″	5 year	2000–2020	85°N–60°S

1.1.2 User Interface and Coding Language

The user interface is accessed via web browser application programming interface (API), shown in figure 1.1, which can be accessed by any registered user of Google Earth Engine. The interface allows users to access all the catalogs that house imagery collections within it along with other various tools and resources that can help users perform the tasks that they aim to accomplish with Google Earth Engine. The API allows users to seamlessly manage their data folders along with visualizing their outputs and functions and it is also capable of displaying statistical outputs within the same window. The coding language within the interactive development environment (IDE) is JavaScript. The API automatically organizes inputs at the top of the IDE so that they can be separate from the code itself making management and alterations rather seamless. Though GEE has a robust and user-friendly API, it is also capable of being utilized from a personal machine without the necessity of a web browser via installation of packages within any Python IDE.

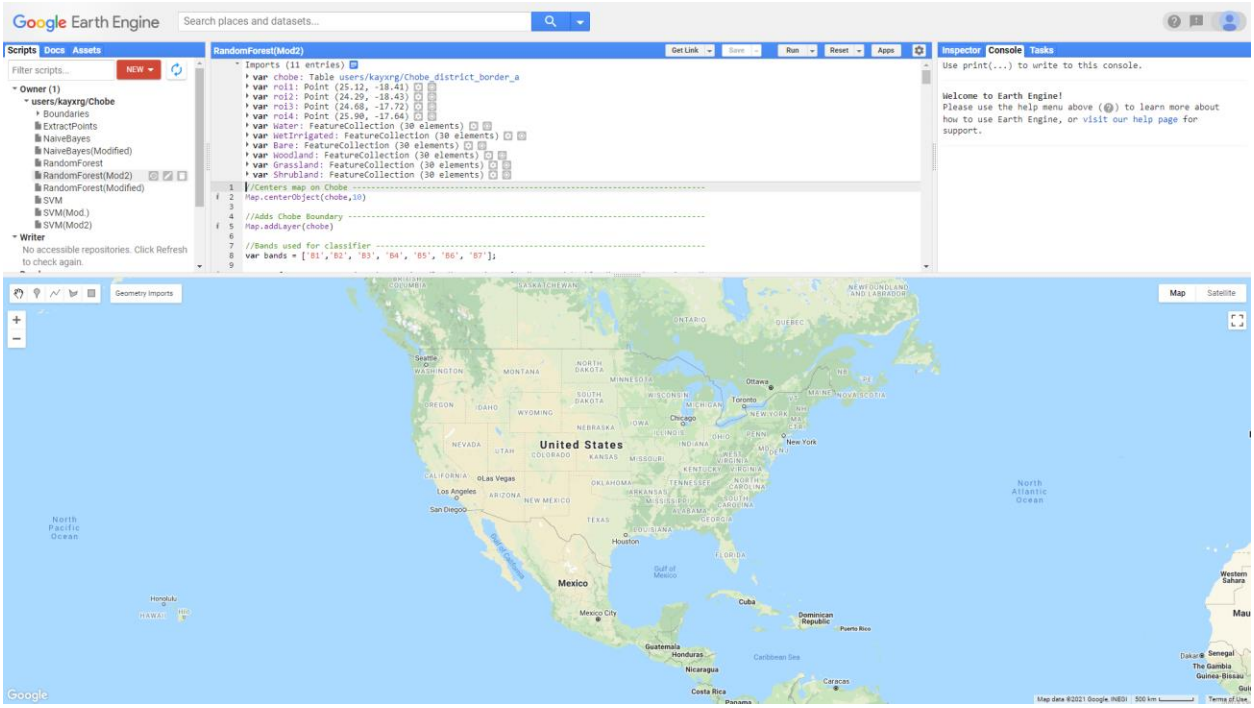


Figure 1.1 Google Earth Engine API and IDE

1.1.3 Analytical Tools

GEE can be utilized for many of the typical tasks performed using remotely sensed data, including landcover classification, risk mapping, crop yield estimation, global surface water change, global forest change, flood mapping, monitoring climate, and assessing land use change (Gorelick et al., 2017). Application of GEE can also be used for disaster management and earth sciences, such as drought detection and occurrences using soil moisture datasets (Mutanga and Kumar, 2019). GEE can implement cloud detection and removal methodology that is comparably more efficient than mono-temporal approaches (Mateo-García et al., 2018). The efficiency of applications within GEE can be increased with higher knowledge and skills of the field that the application is being used towards for not only research purposes but also for land management and conservation purposes.

1.2 Research Question and Goal for Study

There are a range of tools are available for users of geospatial data but require users to have some level of knowledge and expertise to properly utilize them. This can be a significant impediment to the application of geospatial analysis, especially among a user community that may have specific analytical needs but lack overall training in remote sensing science. A major aim of this thesis will be to investigate how GEE can be used to serve this community through the development of a 'toolkit' to accomplish classification task in GEE. There are multiple classifiers within Google Earth Engine and remote sensing in general, this research also aims to compile a comparison between the classifiers built within Google Earth Engine and identify which classifier best suits the needs for creating a classified image for the given study region within Botswana. This study also incorporates the creation of a toolkit for non-specialist users with a lack of expertise in remote sensing in any given software. Giving users to an image analysis platform that can produce results that are comparable to image analysis software such as ArcGIS, QGIS, and ENVI is one of the main focuses in this study. The main classifiers being tested in this study are Naïve Bayes, Support Vector Machine (SVM), and Random Forest (RF) classifiers within Google Earth Engine. The questions that this study aims to answer are:

1. Which machine learning classifier produces quantitatively better landcover classification results from within Google Earth Engines?

2. How well do the classification tools in Google Earth Engine compare to those in commercial analysis software, specifically ArcGIS?

Chapter 2 - Case Study and Study Area

The study area for this research is the Chobe district in Northern Botswana, Africa (Figure 2.1). The region of interest takes part multi-disciplinary studies across the region spanning from remote sensing, wildlife migration, and monitoring of disease spread within the region (Braget et al., 2018, Fox et al., 2017, Naidoo et al., 2016, Pricope and Binford, 2012). There are six differing landcover types that are to be classified from the machine learned algorithms from within Google Earth Engine. The landcover types to be classified within this region are Water, Wet/Irrigated, Grassland, Woodland, Shrubland, Bare/Impervious. The landcover types are highlighted due to their importance in the study area being that there is only one source of water in the region and the other herbaceous and non-herbaceous areas being utilized by humans and wildlife.

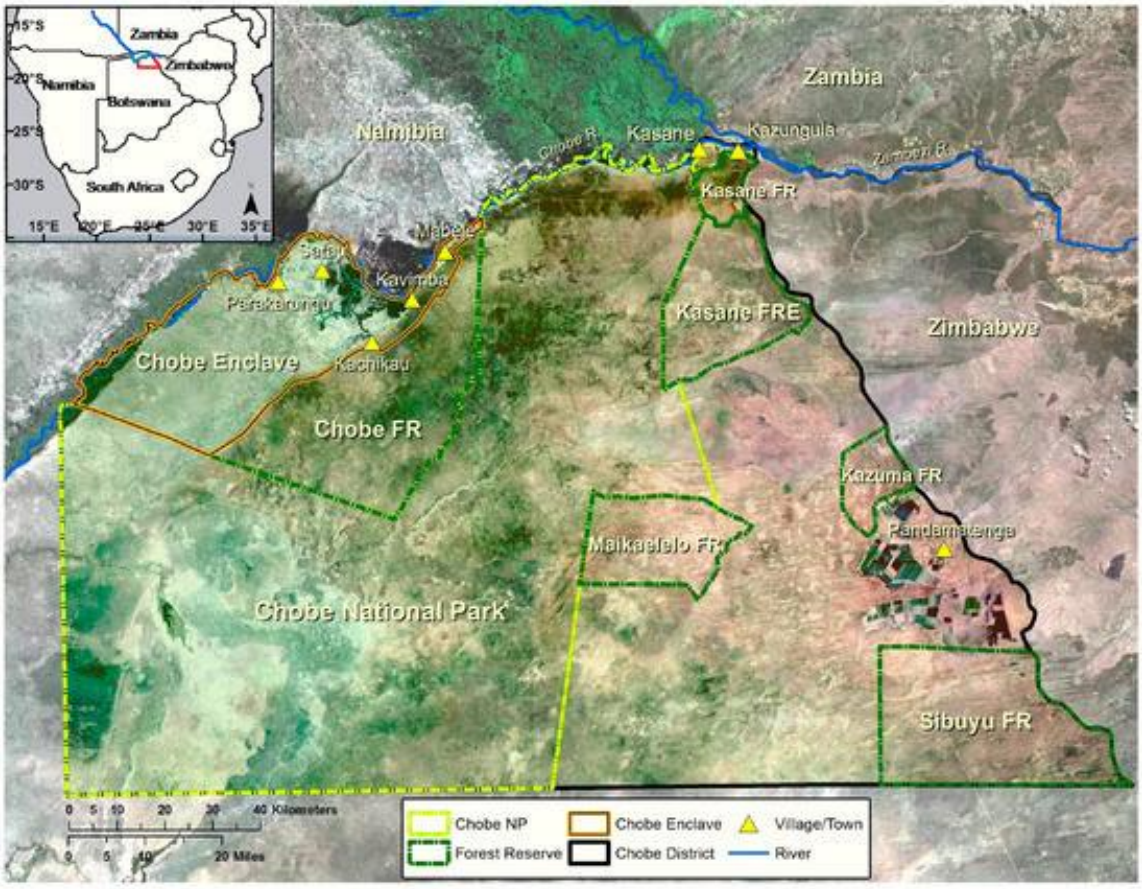


Figure 2.1 Landsat image depicting study area and regions within Chobe District (Fox et al., 2017)

2.1 Physical Landscape

The Chobe District is a semi-arid dryland that supports various fauna within the region, including the largest elephant population in Africa (Fox et al., 2017). One of the main features in the region is the Chobe River which is at the center of the Kavango-Zambezi (KAZA) Transfrontier Conservation Area, that being the largest conservation area in the world encompassing more than 170,000 km^2 (Braget, 2017). As the only surface water source in a dryland area, flood pulses are the driving force in system dynamics by having influences on water quality and disease spread within the region (Braget et al., 2018). The Chobe River also

homes migration locations for zebra and other ungulates (shown in figure 2.2), as they migrate from the Chobe River to other parts within the Chobe District (Naidoo et al., 2016).

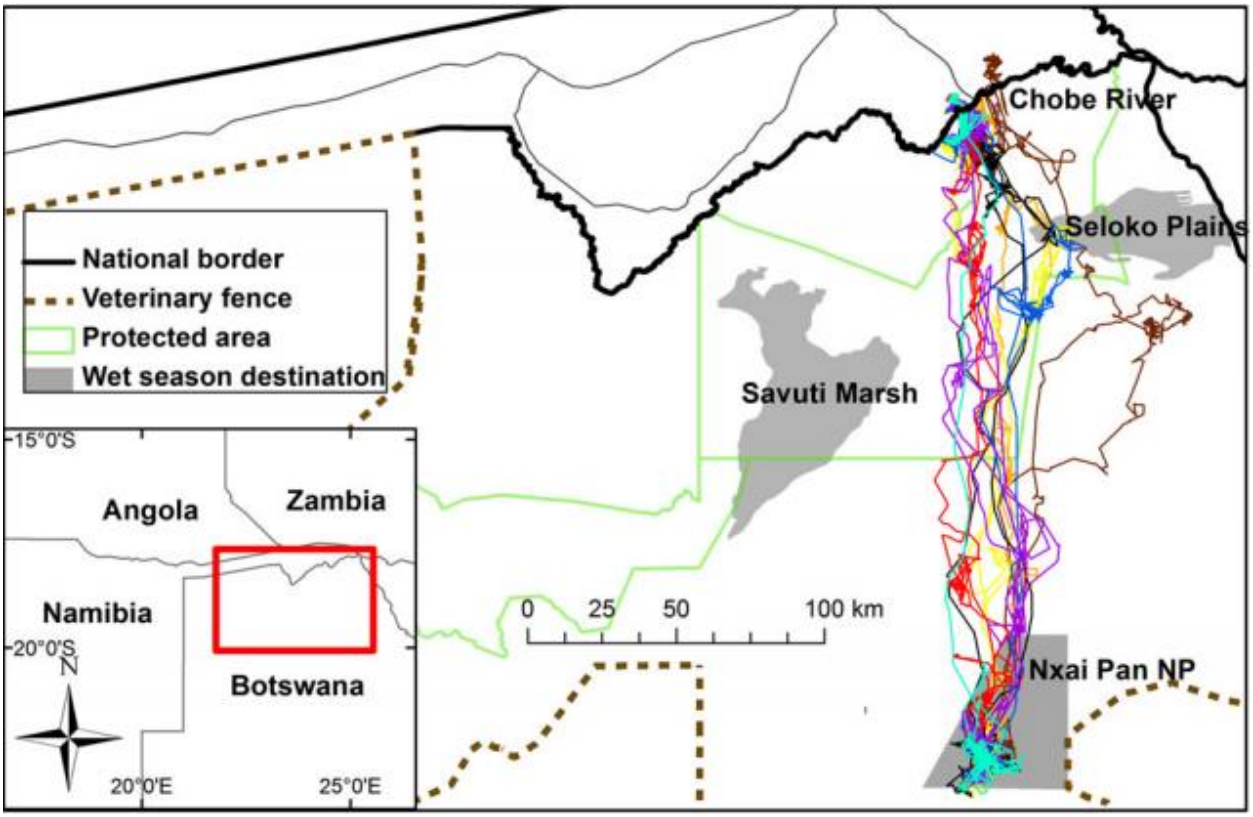


Figure 2.2 Migration Pattern of Zebras within Chobe District (Naidoo et al., 2016)

Precipitation in the region is seasonally influenced by the movement of the Inter-tropical Convergence Zone, with precipitation patterns being related to El Niño Southern Oscillation events (Pricope and Binford, 2012). The wet season falls between November and April, while the dry season is from May to October, as shown in Figure 2.3.

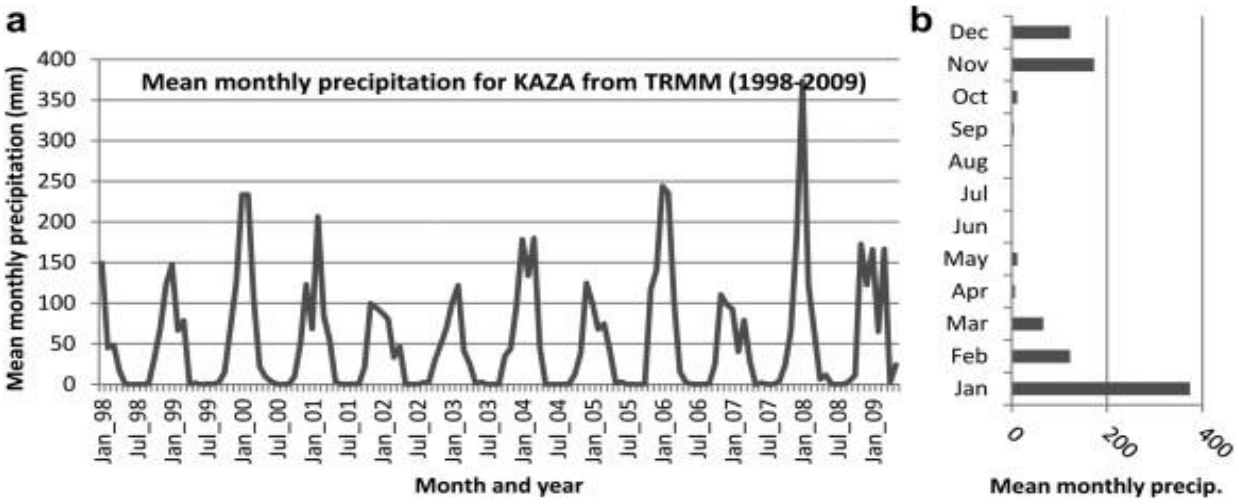


Figure 2.3 Monthly Average Rainfall of Central KAZA region. (Pricope and Binford, 2012)

2.2 Land Use in Chobe District

Throughout the Chobe District there are numerous areas dedicated for wildlife and national reserves owned by the state, and most communal lands are utilized for subsistence livestock grazing and agriculture at a lower intensity than other areas in Botswana due to conflict with wildlife. Most of the land within the district is protected land and is mainly occupied by Chobe National Park, roughly 11,700 km^2 (Fox et al., 2017). Outside of the protected areas and national reserves there are nine villages: Kasane, Kachikau, Kazungula, Kavimba, Lesoma, Mabele/Muchinje, Parakarungu, Pandamatenga, and Satau; Kasane being the urban center and regional government seat within the district.

Chapter 3 - Methodology

The creation of a toolkit that uses a semi-automated supervised classification technique to help non-specialized users of remote sensing is the focus of this study. The semi-automated approach to creating the toolkit is due to the necessity of training each supervised classification technique and having the sample points already selected will help non-specialized users using the toolkit. There are various fields, such as urban mapping (Grippa et al., 2017), hydrology (Brisco et al., 2009) and geology (van Asselen and Seijmonsbergen, 2006), in which a semi-automated classification approach has been proven to work efficiently, with some desire for enhancements. In the case of the geological example, expert knowledge was required to perform the classification using semi-automated techniques. This study attempts to eliminate or reduce the required amount of knowledge necessary to achieve results comparable to other software that requires more expertise in the field of remote sensing.

The semi-automated classification approach uses a method of sampling called pooled sampling, which is a technique that is an effective alternative for classification tasks as compared to conventional classification techniques. Pooled sampling is a training method that collects training samples from one image and uses that same collection to train the classifier for separate images. Use of pooled samples may reduce the amount of time needed to classify various images, specifically land cover where similar reflectance properties can be observed and are consistent across various periods of time/seasons. The creation of a classification pool is a supervised classification method that provides the feature space and decision criteria for all images in the chosen time series (Braget et al., 2018). The toolkit uses supervised classification due to it being readily available and implementable within Google Earth Engines. A semi-automated approach may not be as exact in results as compared to traditional or locally trained

sampling methods, but they are useful to help improve map production, accuracy, and consistency (Latifovic and Pouliot and Campbell, 2018). The semi-automated approach creates an image library in which spectral signatures from the predefined landcover types are used to classify each of the images ingested by the toolkit to produce a classified map. While a traditional approach requires users to train each image separately and run the classifier based on the spectral signatures found only within the trained image. The semi-automated nature in which pooled sampling is utilized in classification is beneficial for non-specialist users and analysts.

Images from this study are taken from the Landsat program which is a image collection program and is jointly ran by the National Aeronautics and Space Administration (NASA) and the United States Geological Survey (USGS) (Markham and Helder, 2012). The Landsat image type used in this study are surface reflectance images which are necessary for quantitative remote sensing and have recently been used by researchers to map land cover and changes in land cover (Wang et al, 2016).

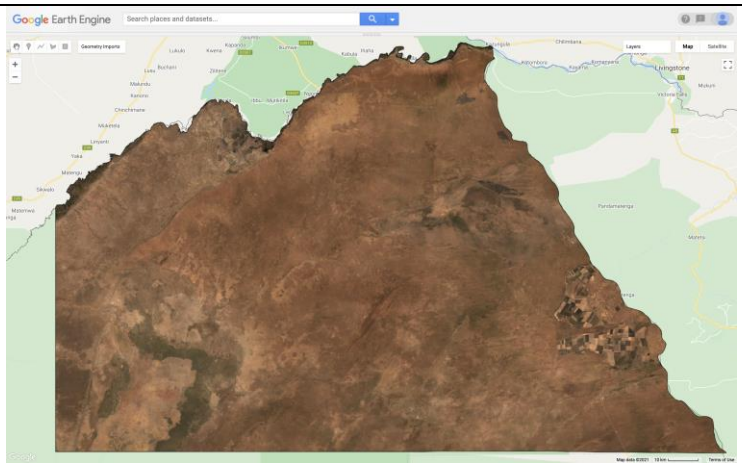
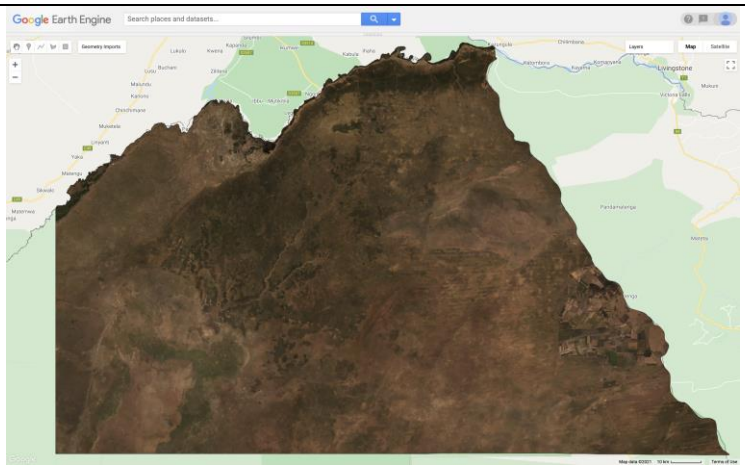
3.1 Data

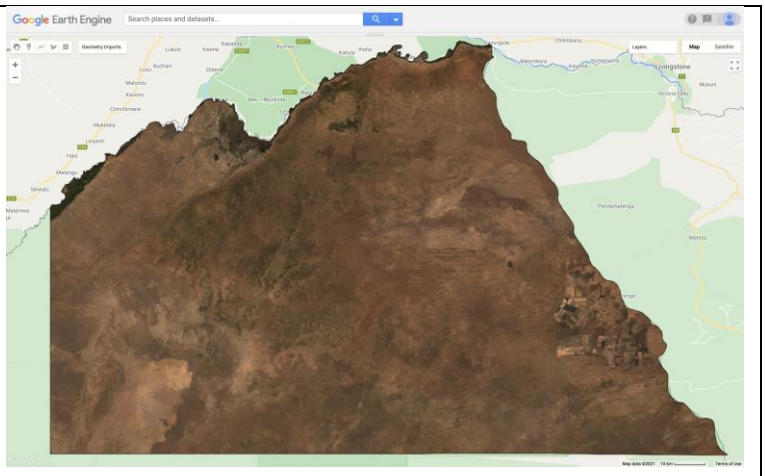
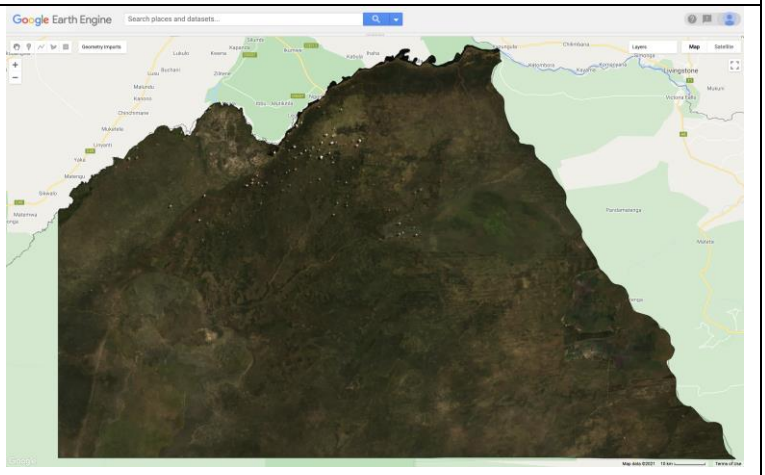
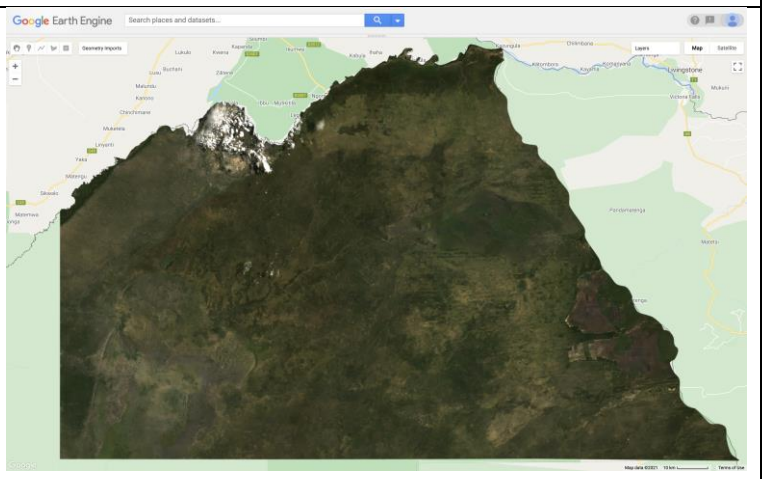
The Landsat-8 Surface Reflectance (SR) images collected from the GEE image library were used to create a mosaic of the study region for classification purposes. Though GEE boasts the utilization and creation of cloud masking techniques, this study did not incorporate any type of cloud masking. Instead, the images collected in this study are images that takes the first image that is observably cloud-free.

The images were chosen based on the amount of cloud coverage within the image itself and which season that the image was taken (wet or dry). Wet images are images taken within the months that statistically have higher amounts of rainfall, while dry images are taken during

months with statistically low amounts of rainfall. Creation of a semi-automatic landcover classification via supervised classification and machine learning classifiers geared towards enabling users across all the spectrums of specialties from researchers to land management and conversationists is the main goal of this research. By utilizing modern techniques and analytical methods to compare each of the given output classifiers so that future users of GEE can get a better and more comprehensive understanding of the capabilities of the software from the output and result of this research.

Table 3.1 Table of base images in GEE's API

Image Type	Date	Season	Base Image
Landsat-8 SR	August 2016	Dry	
Landsat-8 SR	July 2017	Dry	

Landsat-8 SR	September 2018	Dry	
Landsat-8 SR	April 2017	Wet	
Landsat-8 SR	February 2019	Wet	

3.2 Training

Images used in this study are trained from pooled samples of a Landsat-8 mosaic from August 2016. Utilizing the technique of pooled sampling along with the number of images and resources found within GEE's cloud-based dataset structure was imperative to this study not only for image collection reasons but also for testing purposes. Pooled samples aid in increasing accessibility and ease of usage for non-specialist users of the tool. Earlier research shows that pooled samples yield comparable results to images that are not part of the original images in which the pooled samples were taken (Braget et al., 2018). Pooled sampling can be considered a type of supervised classification without the necessity of needing to re-train each image due to the data points being collected into a spectral library. A spectral library consists of a collection of reference spectra so that it can be matched to scene spectra to create a classification and highlight differences (Cloutis, 1996). Creation of a spectral library in this case would be difficult due to vegetation having differing spectral signatures throughout different periods of time and phenological states (Zomer et al., 2009). Spectral libraries are proven to be beneficial to map classification in fields such as geology by using hyperspectral imagery to create a spectral library of soils and lithological features (Schetselaar et al., 2007, Bellinaso, 2010), land surveying (Nasarudin et al., 2011), agriculture (Rao et al., 2007), and landcover classification (Boori et al., 2018).

Unlike the spectral libraries that are used within the fields of geology, which use hyperspectral imagery, pooled sampling uses multispectral imagery which has a lower spectral resolution. The increase in spectral resolution makes it easier to differentiate objects on the ground which is useful for analyzing land cover classification (Chen et al., 2017). The emphasis of this study is to provide users with little expertise in remote sensing an opportunity to utilize a

tool that will give proper results with less work involved in the user's end. Using a pooled sampling method gives users this opportunity by having samples already selected for the multispectral imagery of the study area. The limitation of having crude spectral categorization when monitoring sub-class level classification (Boori et al., 2018), can be outweighed by the increase in availability and accessibility of multispectral imagery as compared to hyperspectral imagery. There are two types of spectral library transfers that can be categorized into two types, spatial transfer, and temporal transfer. This study utilizes the temporal transfer method in which training points are collected and used from the same area but at different periods of time for the creation of a landcover classification map (Nidamanuri and Zbell, 2011).

The images are trained from the same pooled training samples from August 2016, shown in Figure 3.1. The scenes are a culmination of 4 separate Landsat images which were then mosaicked together to fit within the boundaries of the Chobe district. The study uses multiple scenes when testing the classification to demonstrate the viability of pooled samples for this classifier. The strategy for testing the classifier is to see if the classifier can properly classify imagery from differing seasons and produce comparable results to each output. A total of 4 scenes were trained from the sample points from the training image, a list of images and dates can be found in Table 3.1. Given that the classifier is trained during the dry season, the classifier was tested using two different scenes during the wet season (November to April) and two images during the dry season (May to October). The supervised classifier used in this study require sample points from the images to properly classify the landcover classes. There was a total of 362 sample points taken from the training image that was split 50/50 so that 180 points were used for the training set and 182 points were used for validation set. Individual samples were taken and automatically placed under a group of similar sample types: Water, wet/irrigated,

grassland, woodland, shrubland, and bare. The water class includes areas that are water bodies such as rivers or ponds within the region. Wet/Irrigated include regions that are farmland areas as well as saturated regions within the study area. Grassland class includes regions with greenness with no observable tree-coverage and no evidence of irrigation. Woodland class is like the grassland classification except for adding areas with observable tree coverage. The shrubland class and bare class are similar, where arid areas were chosen but the main difference is observable vegetation coverage in the shrubland class as compared to the arid and no vegetation cover of the bare class. Those samples were then compiled into a landcover collection to then be implemented and used to train classifiers. Samples were taken loosely from utilization of unsupervised classification methods and reference maps and studies of earlier research as seen in Figure 3.1 (Fox et al., 2017). The samples taken from the reference images were then input into three different supervised classification methods, naïve bayes, support vector machine, and random forest.

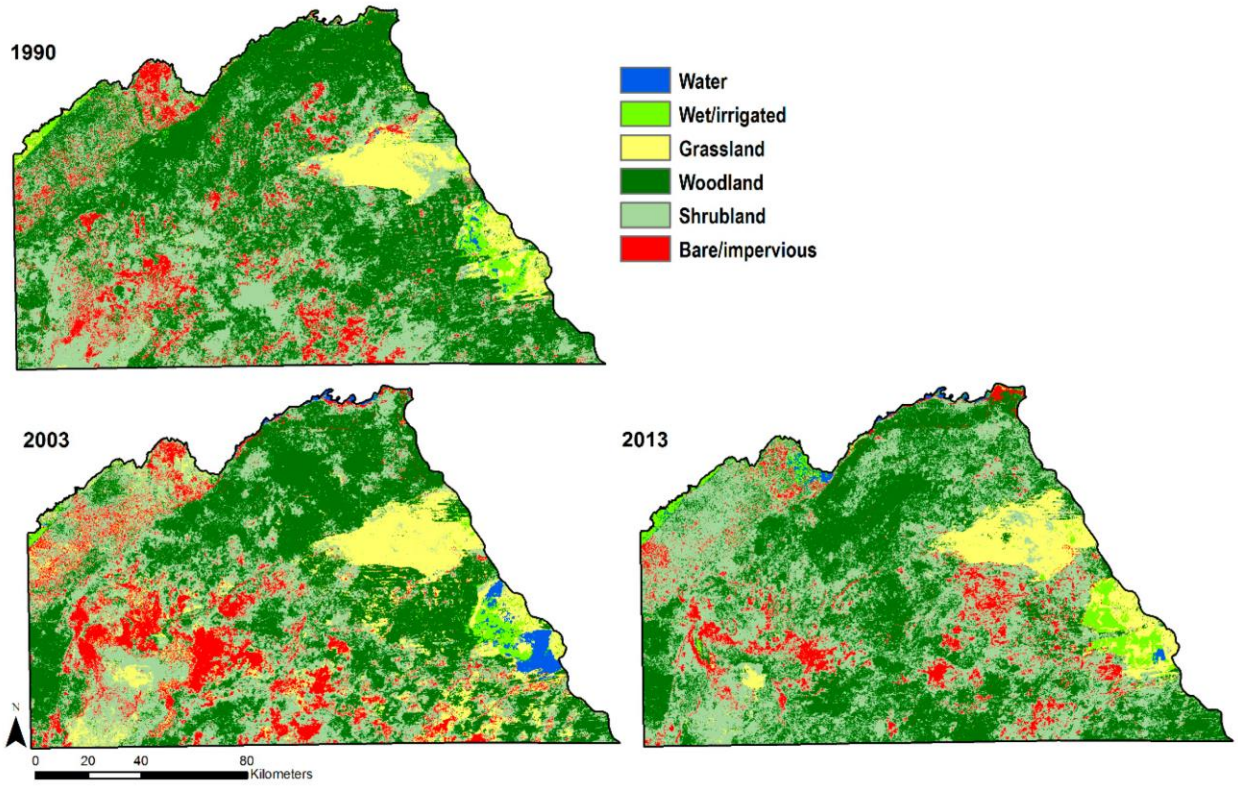


Figure 3.1 Reference map of Classified Landcover regions (Fox et al., 2017)

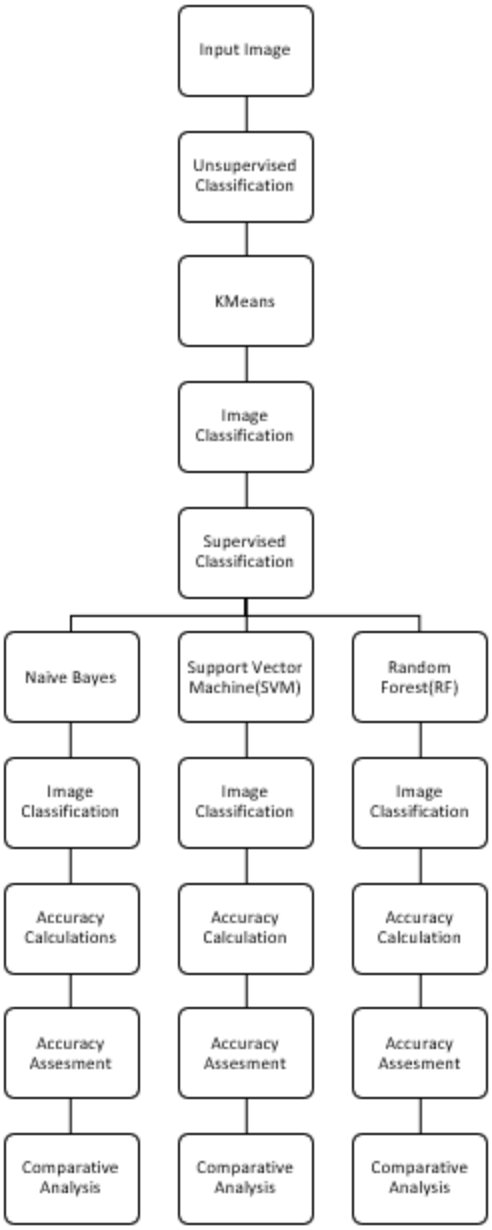


Figure 3.2 Flow chart of work process within GEE.

3.3 Machine Learning Classifiers

This study uses supervised classification techniques which require less post-processing as compared to unsupervised classification techniques. Traditional methods of image classification employ the usage of K-means and ISODATA, unsupervised classifiers, which are clustering

algorithms that involve iterative processes to obtain optimal data points for image training. Unsupervised classification methods differ from supervised classification methods in which they rely on spectrally pixel-based statistics and do not incorporate prior knowledge of the area being classified (Xie et al., 2008). There are several methods of supervised classifications that exist and are utilized in more than just the remote sensing-like image analysis and pattern recognition (Perumal and Bhaskaran, 2010). The assumption that each spectral class can be described by a probability distribution in multispectral space is important to statistical supervised classification (John and Xiuping, 2006). Pooled sampling can be considered as a supervised classification method without the necessity of re-training every image that is to be analyzed. The pooled approach allows users to make a collection of data points to train images which would decrease the amount of time required to train multiple images. The pooled sampling approach would aid non-specialist users in the process of creating and analyzing classified maps of the region of interest without having to train the classifier of choice. Though there are other supervised classification techniques in which distribution models and parameters are not relevant which are called non-parametric classifiers. This study utilizes only parametric classifiers, Naïve Bayes, Support Vector Machine, and Random Forest.

The fundamental decision rules for each classifier are described within Table 3.2, which shows the conceptual basis, classification space, and what type of model each classifier uses to create the outcome. Being separate classifiers, which operate in the feature space, each classifier has a unique conceptual basis. Such as how Naïve Bayes, also known as a maximum likelihood classifier, uses a discriminant function to create a probability to properly classify each class within the given model. Support Vector Machine (SVM) uses a kernel function to measure optimal vector distances to properly classify each class in the model. Random forest is a type of

decision tree classifier in which is uses multiple hierarchical branches to estimate the best possible classification for each class, in turn maximizing the amount of information that can be processed within the model.

Table 3.2 Decision Rules of Supervised Classifiers

Supervised Classifier	Conceptual Basis	Classification Space	Model
Naïve Bayes	Probability	Feature Space	Discriminant Function
SVM	Vector Distance	Feature Space	Kernel Function
Random Forest	Information maximization	Feature Space	Hierarchical Binary

3.3.1 Naïve Bayes Classification

Naïve Bayes is a probabilistic classifier that uses strong independent assumptions (Timofte et al., 2012). Given that Naïve Bayes utilizes a high degree of feature dependency, there are performance issues when the number of classes increases within the sample (Rish, 2001). The Naïve Bayes classifier is based on Bayes' Theorem which is a mathematical formula used to calculate conditional probabilities (Joyce, 2003). Bayes' classification is considered a maximum likelihood classification, which is one of the most common supervised classification methods used in remote sensing. Maximum likelihood classification follows a set of decision rules that computes a set of probabilities for a pixel in a multispectral space that gives the likelihood of which class that the pixel will be assigned (John and Xiuping, 2006). The probability distribution while utilizing a Bayes' classifier is assumed to form a multivariate normal model, which means that it assumes that each class is normally distributed within the feature space. This assumption leads to a simplification in the mathematical process in which the

classifier can classify each class that is defined in the multispectral space where their discriminant function is largest. Thresholds can be applied to the discriminant function that aid in the classification process when classes tend to merge into either other, as shown in Figure 3.3.

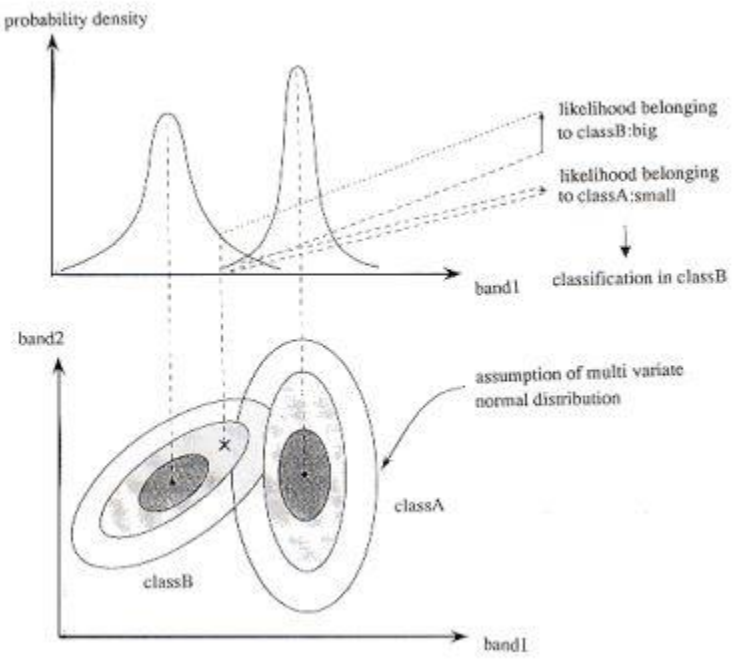


Figure 3.3 Example of feature space selection for Bayesian classifier (Braget, 2017)

The rationale of choosing to compare a simplistic classifier to other more thorough classifiers that use iterative processes such as random forest or kernel-based structures of a support vector machine classifier is to test if a classifier that requires less computational and technical expertise could give comparable results to the other more intensive classifiers.

3.3.2 Support Vector Machine (SVM) Classification

SVM is shown to be exceptional in pattern classification and image classification as it utilizes a kernel-based structure that optimally separates the hyper plane (Thai et al., 2012). The

SVM classifier utilizes a training approach that depends on pixel values in the vicinity of the separating hyperplane, which leads to an optimal hyperplane position for the available training pattern, as shown in Figure 3.4. An optimal hyperplane position occurs when there is the maximum separation between the patterns in each class. If the input data is not linearly distributed, then the development of classification cannot occur without modifications. Transformation of the pixel values to a different feature space that allows a linear distribution can be applied to make the process of classification viable. SVM utilizes a binary classification technique that can be used to perform multiclass classification by imbedding them into a binary decision tree (John and Xiuping, 2006).

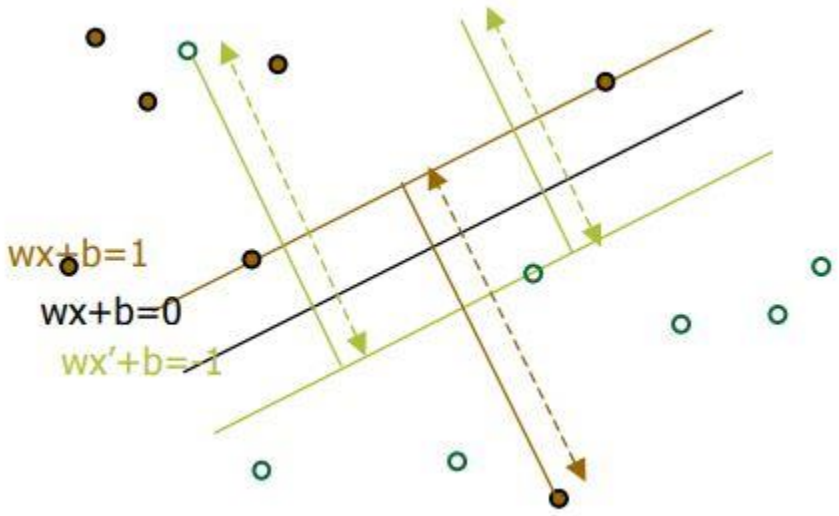


Figure 3.4 Example of how SVM finds maximum separation between points within the Feature Space (Jakkula, 2006)

Within this study the method used for the SVM classifier uses the default settings within Google Earth Engine. Support vector machines have been widely applied to land cover classification and can use its own prediction to train itself (Liu et al., 2013), making it beneficial

for a semi-automated approach. Assuming the training samples are well distributed across the study area, SVM classifiers require less training sites for calibration as compared to other supervised classification algorithms (Schwert et al., 2013). SVM classifiers are less sensitive to sample size when compared to other machine learning algorithms for land cover classification while exhibiting minimal variability in response to various training data (Shao and Lunetta, 2012). When compared to other classifiers such as decision tree and maximum likelihood, SVM performs better in terms of classification accuracy due to the simplification of the vector space that is used for the creation of the hyper-planes (Otukei and Blaschke, 2010).

The default kernel type for Google Earth Engine is set to linear and the gamma value is set to null. The reason behind choosing a support vector machine classifier is to test the complexity of the kernel-based structuring that is utilized by this classifier. Comparing the results of this more computationally intensive classifier to other classifiers that are less computationally intensive will show if it is a more viable option in terms of classifier selection. SVM classifier's performance is based on the optimum kernel function and parameters, but it is noted that SVM performs better than maximum likelihood classifiers and that radial basis function kernels produce more accurate classification maps as compared to polynomial kernels (Kavzoglu and Colkesen, 2009).

3.3.3 Random Forest Classification

The third, and final, classifier used in this study is a Random Forest classifier, which is a decision tree-based classifier that utilizes a set number of generated trees within the model itself to properly estimate values within a given space/region. The collection of trees generated by the classifier decide which value best fits the area (Bosch et al., 2007), given that this classifier is

also parametric, it makes use of a training phase to better understand the data that is being fed into the classifier. The two parameters that are required for a Random Forest classifier to create a prediction model are: the number of classification trees desired, and the number of prediction variables (Rodriguez-Galiano et al., 2012). Each component of the classifier performs part of the task which minimizes the number of features to use in each decision which in turn reduces the amount of processing time and increases accuracy. Random forest uses about two thirds of the samples to train the trees produced and the remaining third of the data is used for internal cross-validation to estimate model performance (Breiman, 2001). Random Forest algorithms, as a classifier, is cost effective and efficient when it comes to making land cover and land use mapping (Gounaridis et al., 2016). When used for land cover classification random forest classifiers are less sensitive compared to other streamline machine learning classifiers due to the large number of decision trees produced while using this method (Belgiu and Drăguţ, 2016). Though random forest utilizes an advance hierarchical method, it has a simple approach that yields accurate results especially when using a high amount of training samples (Du et al., 2015).

This study uses the default settings for the Random Forest classifier within Google Earth Engine, just like the SVM classifier. There are not a set number of trees within this study and there is no limit to how many trees are to be created, which is also a default setting for the classifier provided by Google Earth Engine. Given that the number of trees does not have a large impact on the RF classification, if the number is sufficiently large enough, accuracy tends to increase with more trees. But the accuracy will eventually plateau with a large number of trees (Maxwell et al., 2018). The rationale for selecting random forest classifier is the utilization of iterative processes for classification. The iterative nature of random forest classification is the defining feature that separates this classifier from the others that were selected. Comparison of

the results from a more computationally intensive technique as compared to the probabilistic nature of naïve bayes and kernel-based structure of support vector machine is a driving factor for selection of random forest classifier in this study.

3.4 Accuracy Assessment

Two statistical tests were used to give a more quantitative analysis of each classifier, these tests being a Z-Test, which compared overall accuracy and Kappa value, and McNemar's Chi-squared test, which compared Kappa value. Though overall accuracy is more of the straightforward statistical value, the kappa coefficient provides a measure in which the application agrees in labeling and the level of agreement that occurs due to chance (Foody, 2020). The kappa coefficient can be estimated from the confusion or error matrix and has values that range between 0 and 1. Though the kappa coefficient is arguably an important statistic when comparing classifications, it is noted that it not essentially used to measure accuracy (Foody, 2020).

Reference ↓

Map → Class	1	2	··	m	$\sum$
1	p_{11}	p_{12}	··	p_{1m}	$p_{1\cdot}$
2	p_{21}	p_{22}	··	p_{2m}	$p_{2\cdot}$
⋮	⋮	⋮	··	⋮	⋮
m	p_{m1}	p_{m2}	··	p_{mm}	$p_{m\cdot}$
$\sum$	$p_{\cdot 1}$	$p_{\cdot 2}$	··	$p_{\cdot m}$	1

$$p_o = \sum_{i=1}^{m} p_{ii} \qquad p_e = \sum_{i=1}^{m} p_{i\cdot} p_{\cdot i}$$

$$\sigma_\kappa = \frac{1}{(1-p_e)\sqrt{n}} \sqrt{p_e + p_e^2 - \sum_{i=1}^{m} p_{i\cdot} p_{\cdot i} (p_{i\cdot} + p_{\cdot i})}$$

Figure 3.5 Confusion matrix for multi-class classification and equations for kappa coefficient and standard error (Foody, 2020)

After the analysis of overall accuracy and kappa coefficient, the next statistical analysis that is utilized in this study is producer and consumer accuracy which is computed by the output error matrix that is automatically generated by Google Earth Engine. The calculation of overall accuracy, producer accuracy, and consumer accuracy can be obtained by analyzing the generated error matrix. The producer accuracy is calculated from the error matrix above by taking the number of correctly classified reference sites and dividing it by the number of reference sites for that class and the consumer accuracy is calculated from the error matrix by taking the number of correctly classified reference sites and dividing it by the total number of number reference sites.

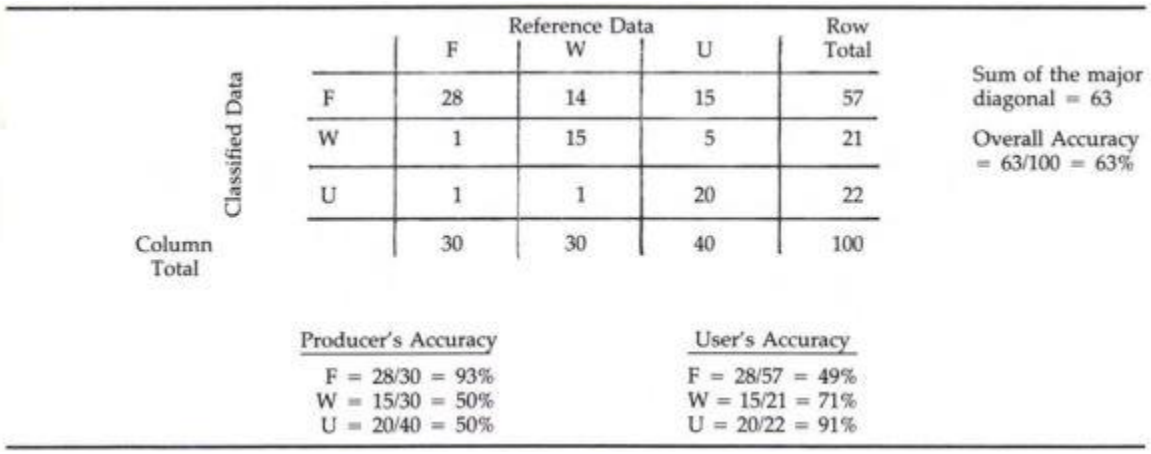


Figure 3.6 Example of error matrix and calculation of producer/consumer accuracy from error matrix (Story and Congalton, 1986)

A confusion matrix for each classifier was generated by Google Earth Engine which assumes that pixels at the reference location can be assigned to a single class and then takes part in an accuracy assessment by comparing the number of pixels that were correctly classified within the confusion matrix (Ruuska et al., 2018). The accuracy assessment is based on the test set of 182 sample points, split into 6 different land classification types, taken from the August 2016 Landsat-8 image shown in Table 3.1. Both training and test sample points were taken using

an unsupervised classification map, created within GEE, of the August 2016 image and a supervised classification map from an outside study. The training points were used to train the classifiers, while the test set was used to analyze the accuracy of each classifier. The overall accuracy of each classifier is based on the number of classes correctly classified by the classifier used.

Chapter 4 - Results

4.1 Naïve Bayes Results

The results of running Naïve Bayes classification on the training image, Figure 4.1, which has an overall accuracy of 0.852 and Kappa value of 0.822, which shows that the overall accuracy of the classifier is roughly the same as the predicted accuracy as shown by the Kappa value. The overall accuracy being high is something worth noting especially with a classifier such as Naïve Bayes which does not have the reputation of being a highly accurate classifier. A high overall accuracy paired alongside a high Kappa value shows that the classifier was rather efficient when classifying the image itself. Other statistical values such as producer accuracy and user accuracy are observed in Table 4.1, which shows a good relationship between what the image has as landcover classifications and what is each landcover. Each class has a producer and user accuracy around the same percentage accuracy except shrubland which has a high producer accuracy but a rather low user accuracy.

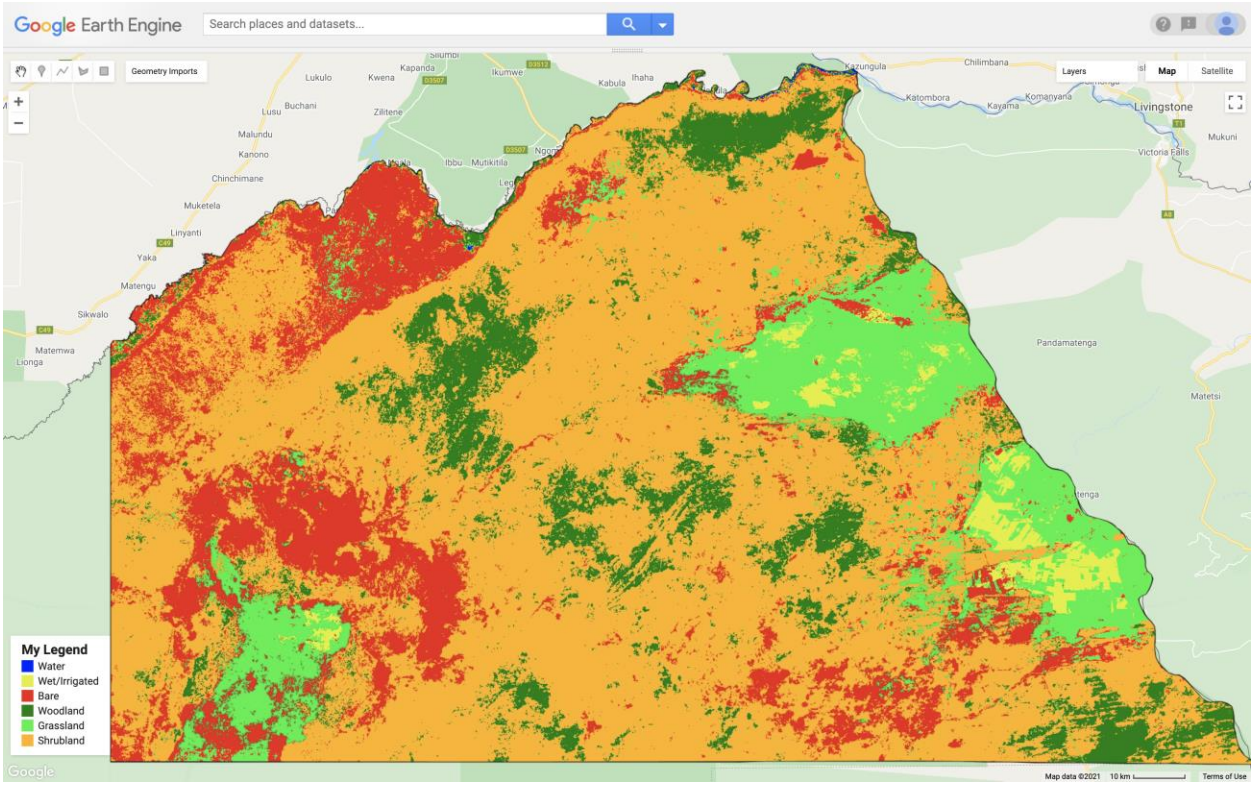


Figure 4.1 Results of training image using Naïve Bayes classifier in GEE's API (August 2016)

Table 4.1 Error matrix and Producer/User accuracy results of August 2016 image. (Naïve Bayes)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	28	0	0	0	2	0	0.933	1
Wet/Irrigated	0	24	1	0	5	0	0.800	0.889
Bare	0	1	27	0	2	2	0.844	0.964
Woodland	0	0	0	25	2	3	0.833	0.962
Grassland	0	2	0	0	22	6	0.733	0.667
Shrubland	0	0	0	1	0	29	0.967	0.725
Overall Accuracy							0.852	0.868

The results from the Naïve Bayes classification on the July 2017 image has an overall accuracy of 0.676 and Kappa value of 0.611, again showing that the estimated accuracy of the classifier is like the calculated accuracy. The overall accuracy and kappa value are both lower than the image used in the training image which may be due to some subtle changes in the landcover that is shown in Table 3.1. The producer accuracy for water, wet/irrigated, and bare have high accuracy percentages but wet/irrigated and bare do not have as high of a user accuracy percentage as water. The remaining landcover classes within the classifier had a much lower producer accuracy as compared to the previously stated classes and had equally as low user accuracy percentages. Though it is notable that grassland and shrubland had lower producer accuracy as compared to their user accuracy which means that even though the accuracy of these classes was low, the classification of these classes was more accurate in actual grassland and shrubland.

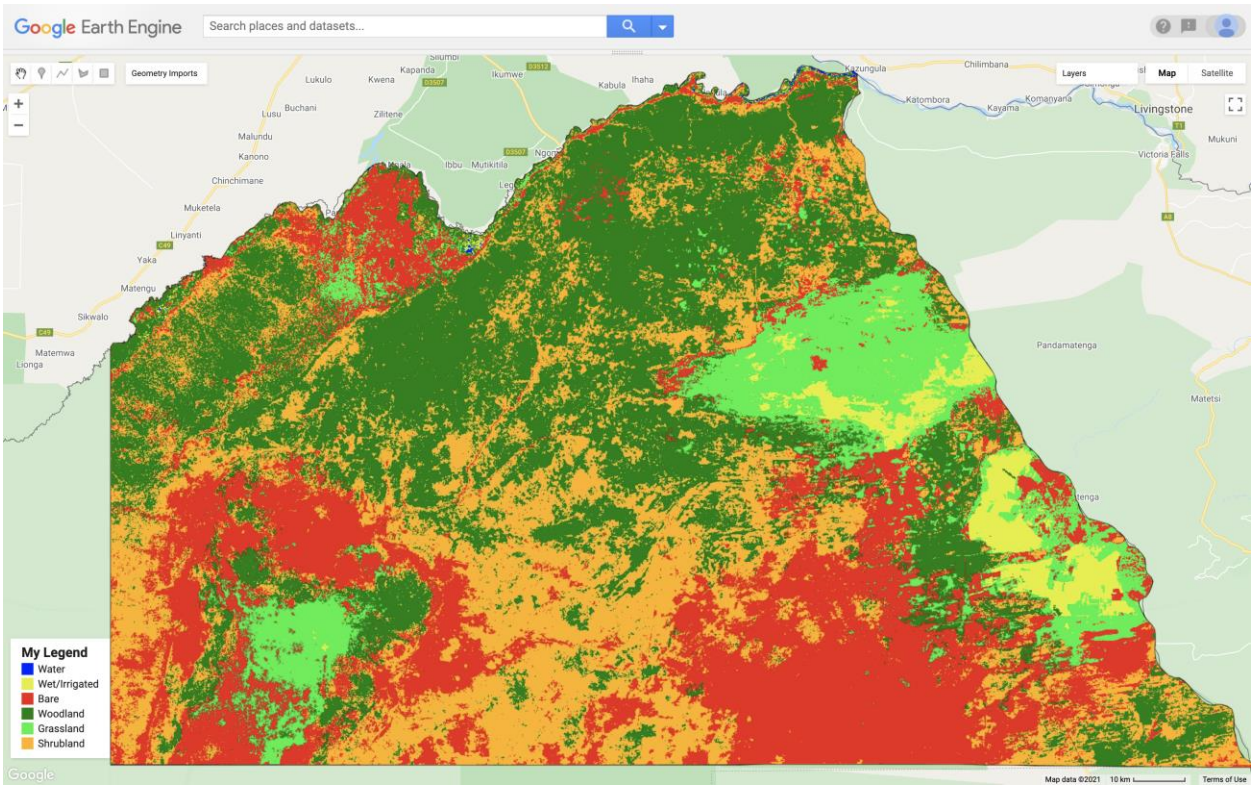


Figure 4.2 Results of Naïve Bayes classifier on image from July 2017 in GEE's API

Table 4.2 Error matrix and Producer/User accuracy results of July 2017 image. (Naïve Bayes)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	27	2	1	0	0	0	0.900	1
Wet/Irrigated	0	25	1	0	4	0	0.833	0.694
Bare	0	0	24	3	3	2	0.750	0.649
Woodland	0	1	0	18	1	10	0.600	0.563
Grassland	0	8	7	0	14	1	0.467	0.636
Shrubland	0	0	4	11	0	15	0.500	0.536
Overall Accuracy							0.676	0.680

The result from the September image has an overall accuracy of 0.709 and Kappa value of 0.651, continuing the trend of having an overall accuracy that is roughly like the Kappa value. There is also a notable trend that the first three classes (water, wet/irrigated, and bare) have a significantly higher accuracy than the bottom three classes. This is showing that the classifier is having difficulties when classifying these classes. Looking at the producer and user accuracies show that shrubland has the lowest accuracy of both, while water has the highest value of both. Thus, continuing to demonstrate the difficulties this classifier has when classifying the shrubland class.

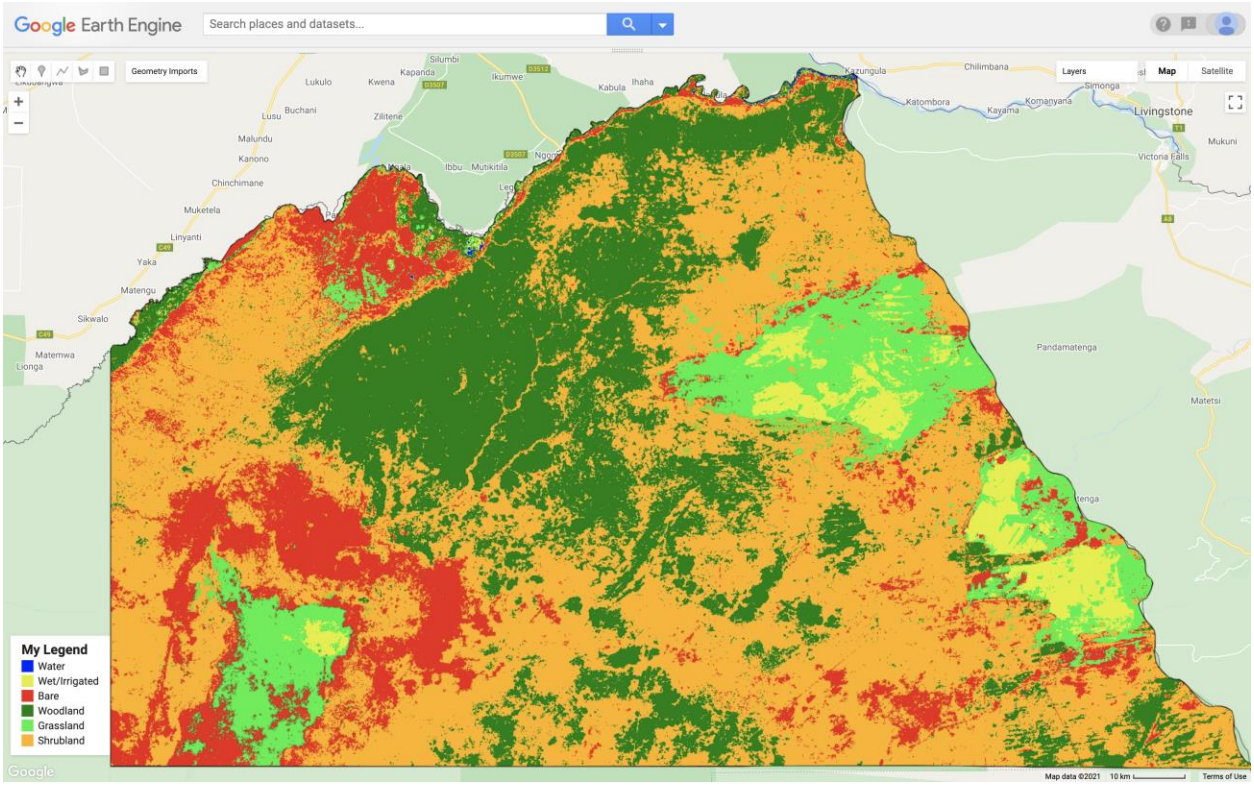


Figure 4.3 Results of Naïve Bayes classifier on image from September 2018 in GEE's API

Table 4.3 Error matrix and Producer/User accuracy results of September 2018 image. (Naïve Bayes)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	28	1	0	0	1	0	0.933	1
Wet/Irrigated	0	25	0	0	5	0	0.833	0.735
Bare	0	0	25	0	3	4	0.781	0.862
Woodland	0	0	1	20	1	8	0.667	0.625
Grassland	0	8	0	0	16	6	0.533	0.615
Shrubland	0	0	3	12	0	15	0.500	0.455
Overall Accuracy							0.709	0.715

The results of the first wet season imagery from April 2017 has an overall accuracy of 0.670 and Kappa value of 0.604. Though the image being trained is from a differing season, the

results are comparable to those from dry season imagery based on the overall accuracy and Kappa value. But, looking at the producer accuracies and user accuracy shows that water is still the best classified class within this classifier while shrubland, grassland, and woodland have the lowest producer and user accuracy. Unlike the previous classified images wet/irrigated has a much lower user accuracy as compared to the producer accuracy, which may be due to the increase in vegetation coverage during the wet season.

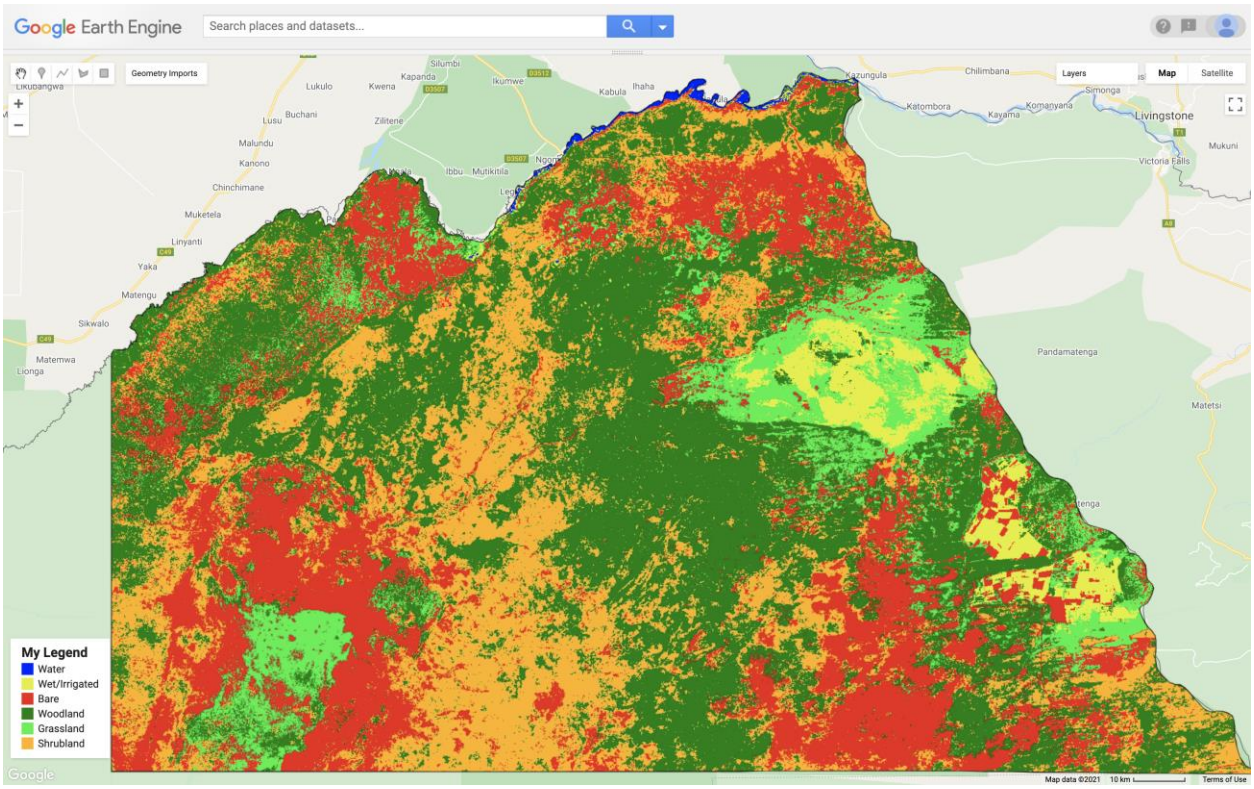


Figure 4.4 Results of Naïve Bayes classifier on image from April 2017 in GEE's API

Table 4.4 Error matrix and Producer/User accuracy results of April 2017 image. (Naïve Bayes)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	28	2	0	0	0	0	0.933	1
Wet/Irrigated	0	23	1	1	5	0	0.767	0.697
Bare	0	1	21	2	0	8	0.656	0.636
Woodland	0	0	7	18	2	3	0.600	0.514
Grassland	0	7	3	2	15	3	0.500	0.682
Shrubland	0	0	1	12	0	17	0.567	0.548
Overall Accuracy							0.670	0.680

The results of classification on an image with cloud coverage is displayed in Figure 4.5 which has an overall accuracy of 0.533 and Kappa value of 0.439. The introduction of clouds appears to have lowered the overall accuracy of the classifier as compared to the other classified images. Though the overall accuracy and Kappa value have lowered, the trend of water being highly classified has continued in this image while the classification of shrubland has severely decreased. Although many classes had lower producer accuracy results as compared to the other classified images, the user accuracies for some of the classes are higher, such as the user accuracy of grassland and shrubland.

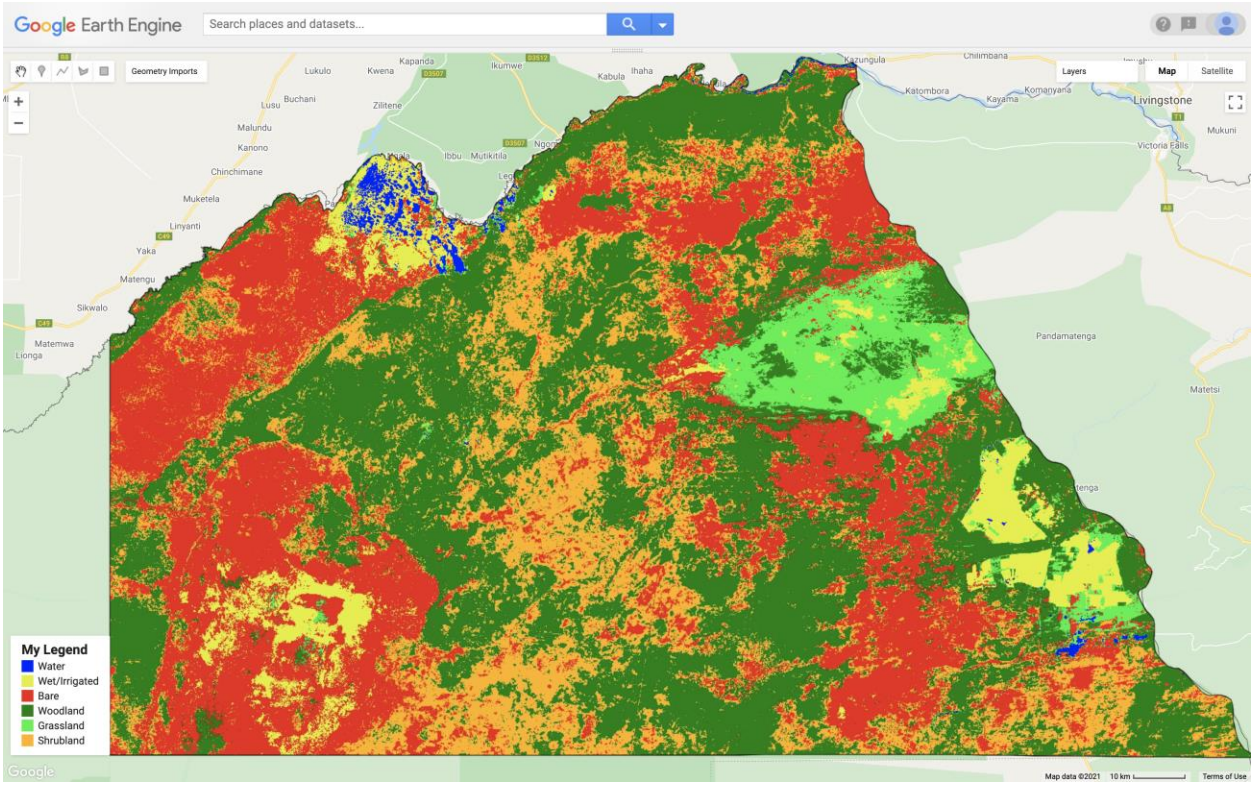


Figure 4.5 Results of Naïve Bayes classifier on image from February 2019 in GEE's API

Table 4.5 Error matrix and Producer/User accuracy results of February 2019 image. (NB)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	26	0	0	3	1	0	0.867	0.929
Wet/Irrigated	1	23	1	3	2	0	0.767	0.676
Bare	0	4	14	11	0	3	0.438	0.438
Woodland	1	0	3	14	1	11	0.467	0.286
Grassland	0	7	3	5	14	1	0.467	0.778
Shrubland	0	0	11	13	0	6	0.200	0.286
Overall Accuracy							0.533	0.566

4.2 Support Vector Machine (SVM) Results

The results of the first SVM image, shown in Figure 4.6, has an overall accuracy of 100% and Kappa of 1. The accuracy of this classifier is incredible when compared to the results from the Naïve Bayes classifier where differences in accuracies across classes was observed in the error matrices. The lack of difference in producer and user accuracy shows how well the classifier worked when classifying the classes showing just how well the SVM classifier worked when differentiating between classes.

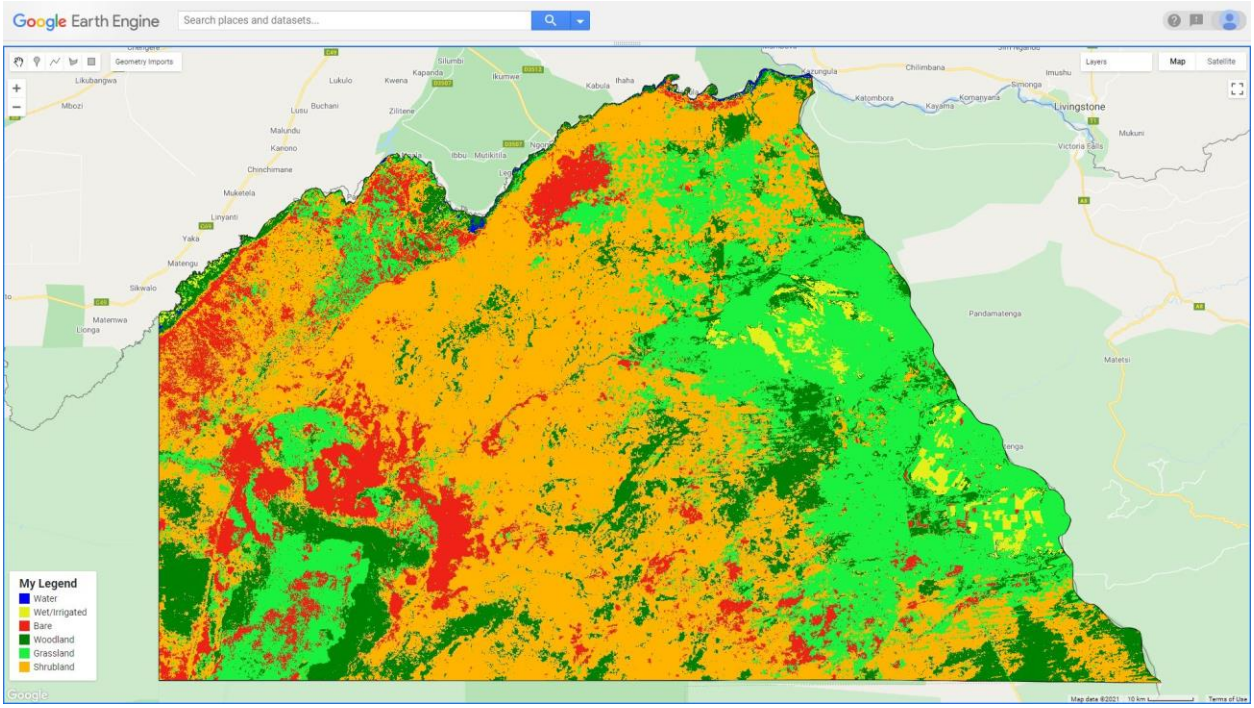


Figure 4.6 Results of training image using SVM classifier in GEE's API (August 2016)

Table 4.6 Error matrix and Producer/User accuracy results of August 2016 image. (SVM)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	30	0	0	0	0	1	1
Bare	0	0	32	0	0	0	1	1
Woodland	0	0	0	30	0	0	1	1
Grassland	0	0	0	0	30	0	1	1
Shrubland	0	0	0	0	0	30	1	1
Overall Accuracy							1	1

With an overall accuracy of 0.879 and Kappa value of 0.855, the results of the July 2017 image, Figure 4.7, display the reliability in accuracy of the SVM classifier. The overall accuracy and Kappa value being roughly equal shows that the classifier worked exactly how it was predicted to work. By observing the error matrix, it is apparent that this classifier also has difficulties when classifying shrubland with a producer accuracy of 0.667 but the user accuracy shows that a higher accuracy. The observable trend of the first 3 classes having higher accuracies while the latter 3 classes, excluding grassland in this classifier, have much lower accuracies. This may be due to changes in landcover changes or differences in vegetation health between each image.

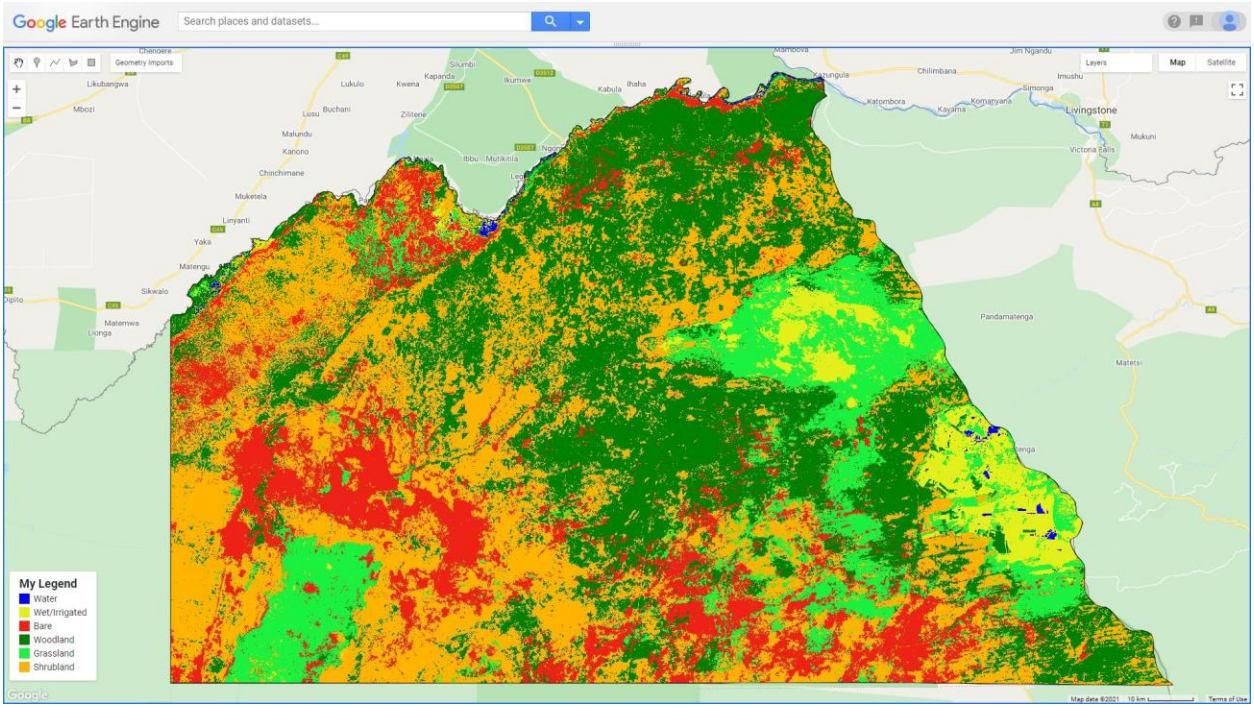


Figure 4.7 Results of SVM classifier on image from July 2017 in GEE's API

Table 4.7 Error matrix and Producer/User accuracy results of July 2017 image. (SVM)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	29	0	0	1	0	0.967	0.906
Bare	0	0	31	0	0	1	0.969	0.939
Woodland	0	0	2	23	0	5	0.767	0.697
Grassland	0	3	0	0	27	0	0.900	0.964
Shrubland	0	0	0	10	0	20	0.667	0.769
Overall Accuracy							0.879	0.879

The resulting overall accuracy for the classified September image, Figure 4.8, is 0.896 and has a Kappa value of 0.875, continuing to display a higher accuracy as compared to Naïve Bayes on similar images. The producer accuracies for this classification are roughly like the

producer accuracies beforehand, though shrubland has a much higher producer accuracy as seen previously in other matrices. Though shrubland has a higher producer accuracy, it continues to have the lowest user accuracy of all the classes within the classifier which shows the continued difficulties of classifying this landcover type.

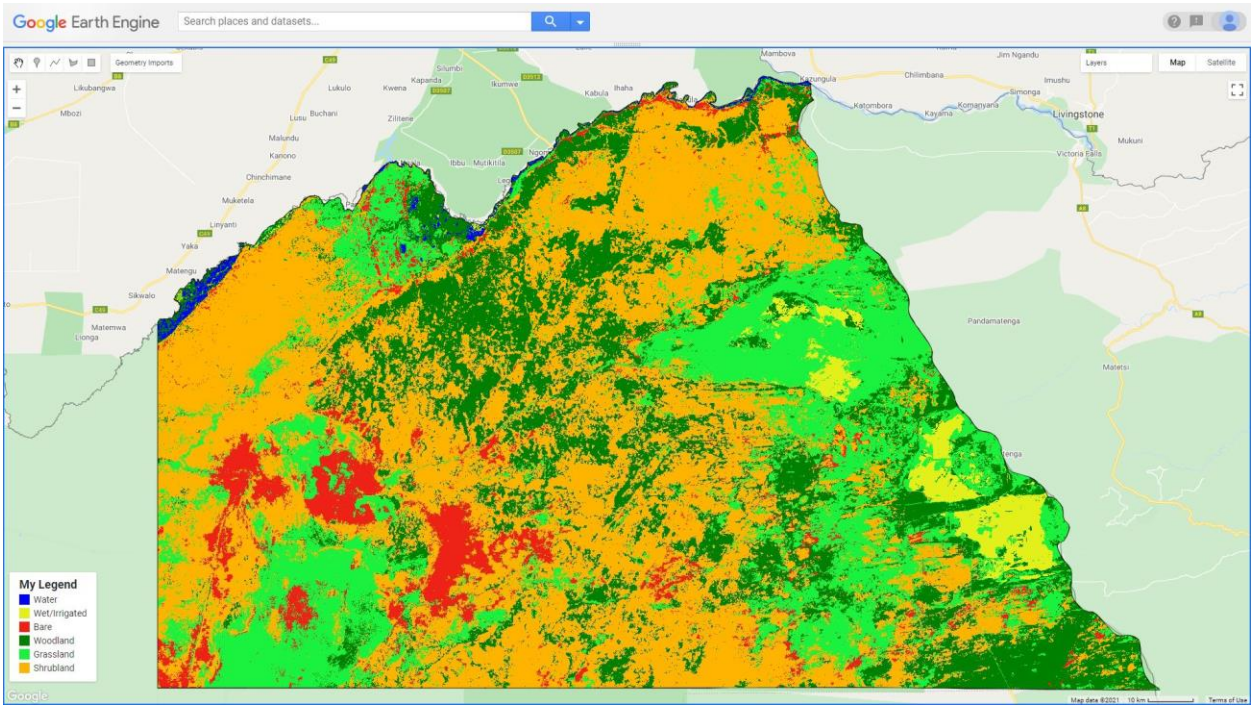


Figure 4.8 Results of SVM classifier on image from September 2018 in GEE's API

Table 4.8 Error matrix and Producer/User accuracy results of September 2018 image. (SVM)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	28	0	0	2	0	0.933	0.933
Bare	0	0	32	0	0	0	1	1
Woodland	0	0	0	21	0	9	0.700	0.778
Grassland	0	2	0	0	28	0	0.933	0.933
Shrubland	0	0	0	6	0	24	0.800	0.727
Overall Accuracy							0.896	0.895

The first of the wet season imagery to be classified utilizing an SVM classifier, the April 2017 image, Figure 4.9, has an overall accuracy of 0.863 and Kappa value of 0.835, showing that the classifier is, again, working how it was estimated to work as measured by the Kappa value. The producer accuracy and user accuracy of shrubland has continued to be one of the lowest of all the landcover types, which can be observed in Table 4.9. Though the producer accuracy is low for shrubland, the user accuracy shows that the classifier achieves suitable results for classifying actual landcover, which can be observed by the user accuracy for each class.

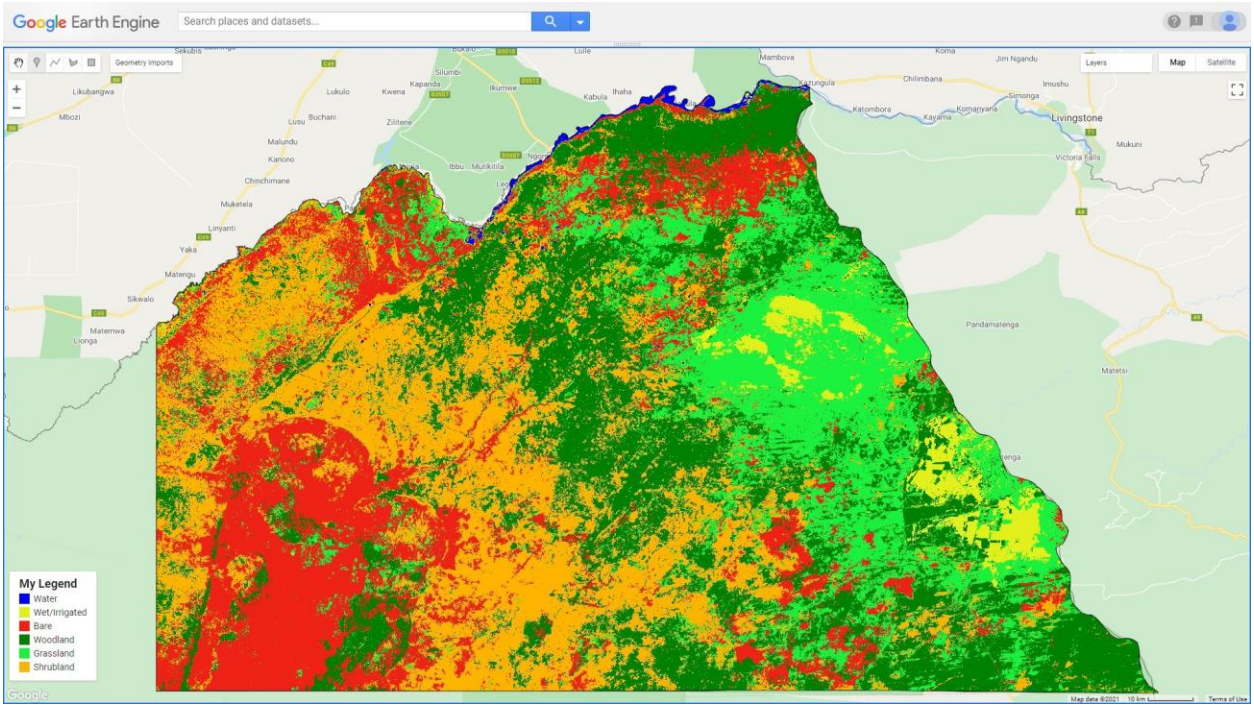


Figure 4.9 Results of SVM classifier on image from April 2017 in GEE's API

Table 4.9 Error matrix and Producer/User accuracy results of April 2017 image. (SVM)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	28	0	0	2	0	0.933	0.966
Bare	0	0	27	1	1	3	0.844	0.794
Woodland	0	1	1	25	0	3	0.833	0.833
Grassland	0	0	2	1	27	0	0.900	0.818
Shrubland	0	0	4	3	3	20	0.667	0.769
Overall Accuracy							0.863	0.863

The results for the February 2019 image show an overall accuracy of 0.852 and a Kappa value of 0.822, showing that the classify has a lower accuracy as compared to the other wet season image as seen in Figure 4.10. This image tests the classifier's ability to handle cloud

coverage and most likely shows a small difference in accuracy due to clouds being introduced. From observation of the producer accuracy in Table 4.10, shrubland has the lowest producer accuracy but is not attributed to having the lowest user accuracy. The user accuracy of woodland being the lowest shows that that classifier had difficulties differentiating and classifying woodland from other landcover types, which could be due to the increase of vegetation in wet season imagery.

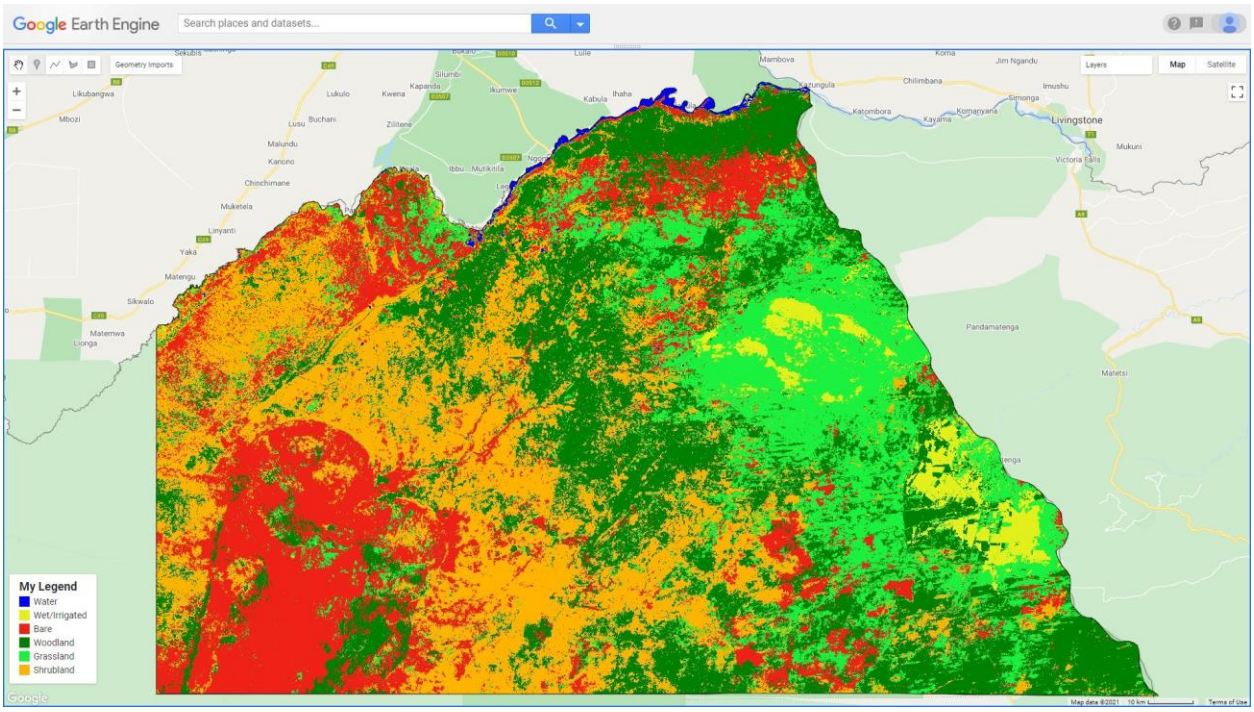


Figure 4.10 Results of SVM classifier on image from February 2019 in GEE's API

Table 4.10 Error matrix and Producer/User accuracy results of February 2019 image. (SVM)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	28	0	1	1	0	0.933	0.875
Bare	0	0	28	1	0	3	0.875	0.933
Woodland	0	0	1	23	2	4	0.767	0.676
Grassland	0	4	0	2	24	0	0.800	0.889
Shrubland	0	0	1	7	0	22	0.733	0.759
Overall Accuracy							0.852	0.855

4.3 Random Forest Results

The results of the Random Forest classifier on the image used for sampling, Figure 4.11, has an overall accuracy of 0.984 and Kappa value of 0.980, these values having such a low difference shows that the classifier's accuracy performance almost matches the estimated performance, as indicated by the Kappa value. The producer and user accuracies for each landcover type are roughly the same with the lowest producer accuracy being attributed to grassland and the lowest user accuracy being attributed to woodland. Though some of the classes have a measured producer accuracy of 100%, shrubland and wet/irrigated have lower user accuracies which shows that the landcover that is supposedly classified as shrubland or wet/irrigated are not actually shrubland or wet/irrigated areas.

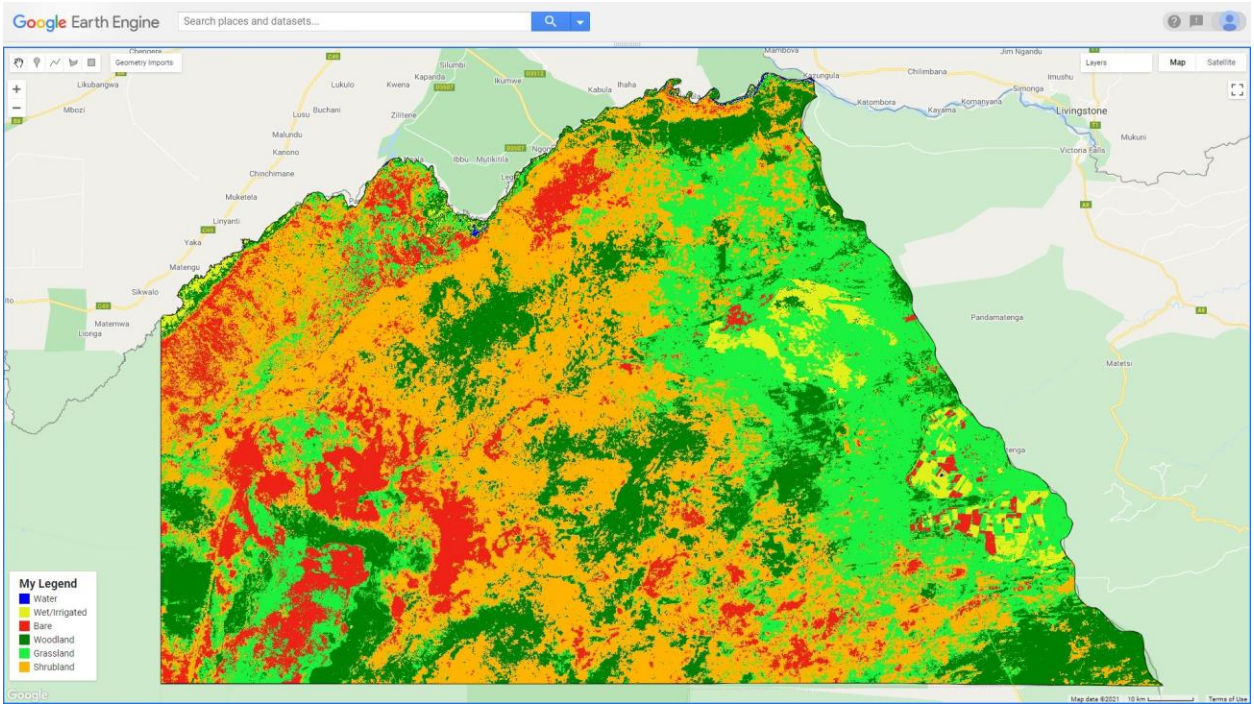


Figure 4.11 Results of training image using Random Forest classifier in GEE's API (August 2016)

Table 4.11 Error matrix and Producer/User accuracy results of August 2016 image. (Random Forest)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	30	0	0	0	0	1	0.968
Bare	0	0	32	0	0	0	1	1
Woodland	0	0	0	29	0	0	0.967	0.967
Grassland	0	0	0	1	28	1	0.933	1
Shrubland	0	0	0	0	0	30	1	0.968
Overall Accuracy							0.984	0.984

The results from the July 2017 image, Figure 4.12, yield an overall accuracy of 0.923 and Kappa value of 0.907. Examination of the error matrix shows shrubland having a low producer

value, but the associated user accuracy is rather high which shows that though the classifier inaccurately measured some of the class, that the land that is classified as such is shrubland. Inversely, woodland has a high producer accuracy but has a lower user accuracy showing that the classifier inaccurately classified part of the woodland landcover.

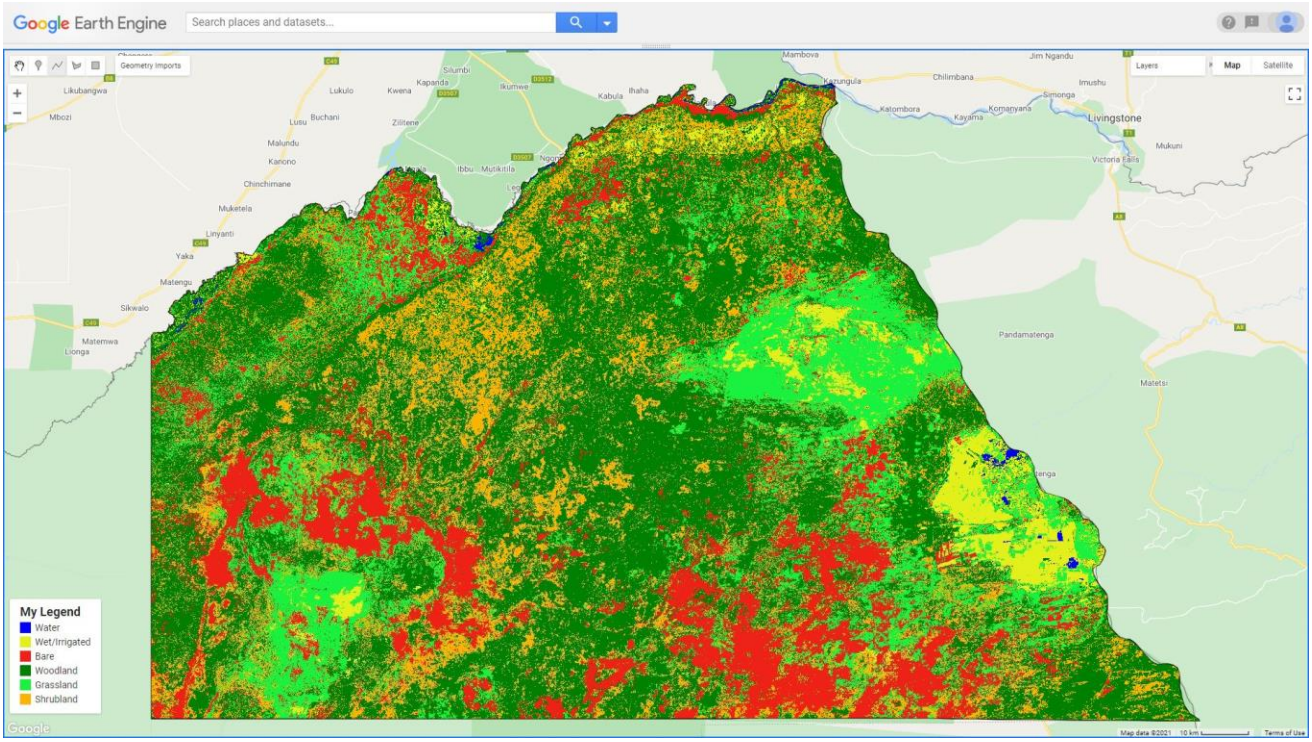


Figure 4.12 Results of Random Forest classifier on image from July 2017 in GEE's API

Table 4.12 Error matrix and Producer/User accuracy results of July 2017 image. (Random Forest)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	28	0	0	2	0	0.933	0.966
Bare	0	0	32	0	0	0	1	0.941
Woodland	0	1	1	27	0	1	0.900	0.771
Grassland	0	0	1	1	28	0	0.933	0.933
Shrubland	0	0	0	7	0	23	0.767	0.958
Overall Accuracy							0.923	0.928

The results of September 2018 image, shown in Figure 4.13, has an overall accuracy of 0.962 and Kappa value of 0.954. Examination of the error matrix in Table 4.13, show rather similar producer and user accuracies. But one class that is notable to observe is the woodland landcover given that it has a decrease in user accuracy as compared to producer accuracy. It is also notable that both grassland and shrubland had lower producer accuracies as compared to their user accuracies.

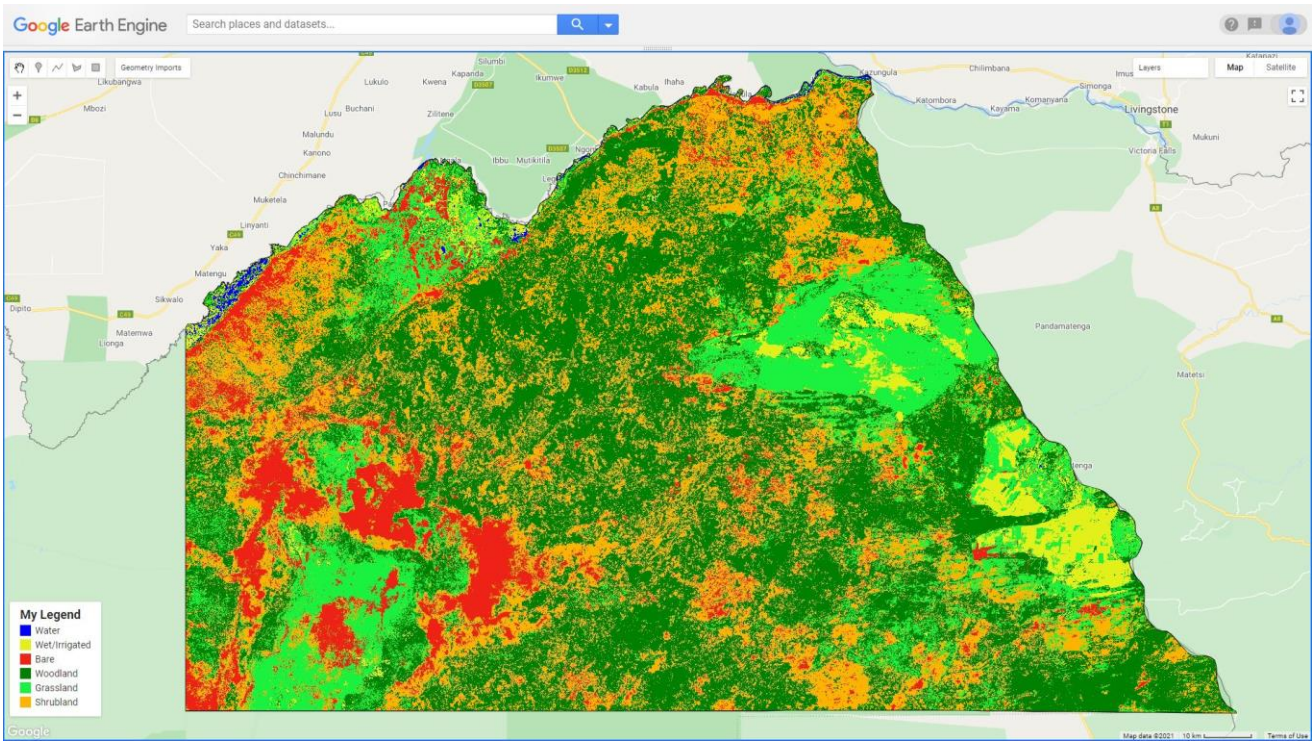


Figure 4.13 Results of Random Forest classifier on image from September 2018 in GEE's API

Table 4.13 Error matrix and Producer/User accuracy results of September 2018 image. (Random Forest)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	29	0	0	1	0	0.967	0.935
Bare	0	0	32	0	0	0	1	1
Woodland	0	1	0	29	0	0	0.967	0.879
Grassland	0	1	0	1	28	0	0.933	0.966
Shrubland	0	0	0	3	0	27	0.900	1
Overall Accuracy							0.962	0.963

The results from the April 2017 image have an overall accuracy of 0.923 and Kappa value of 0.908. Though the image is during the wet season, the overall accuracy had changed slightly as compared to images that were from the dry season. Examination of the error matrix shows shrubland and grassland having the lowest producer values but both landcovers have some of the highest user accuracies. Showing that the classifier was not able to classify the landcovers properly but when it comes to actual landcover the classifier performed rather well.

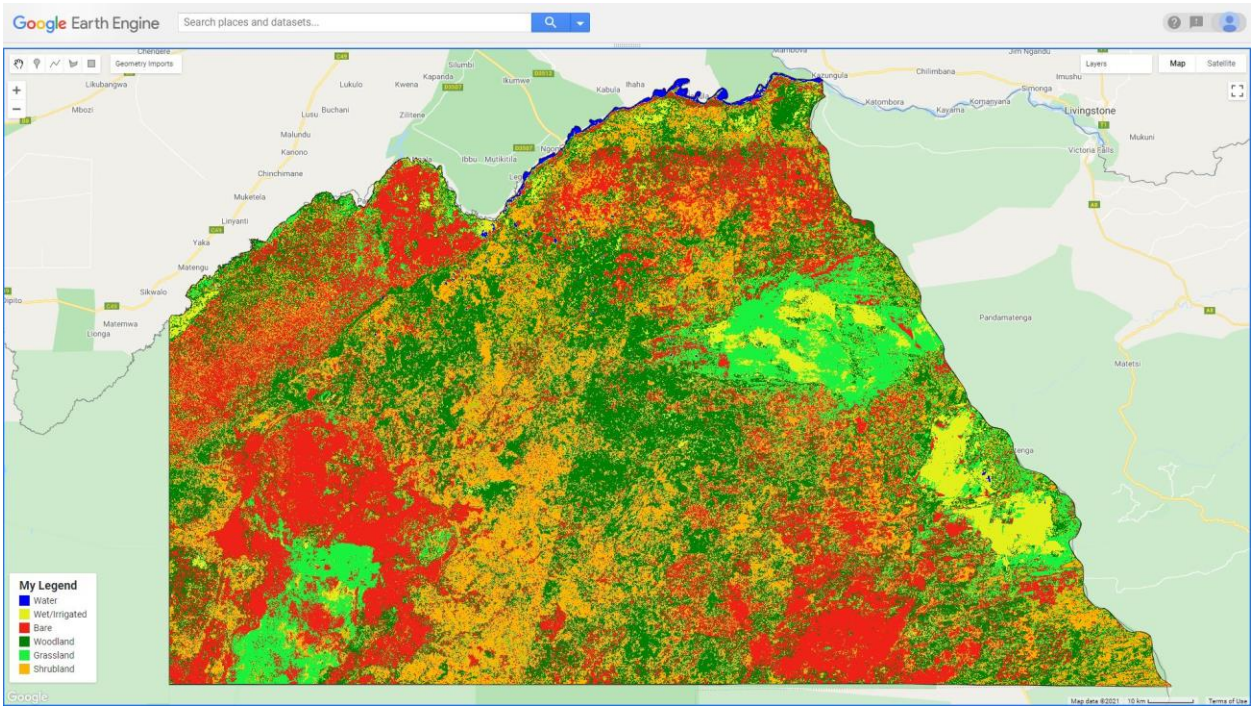


Figure 4.14 Results of Random Forest classifier on image from April 2017 in GEE's API

Table 4.14 Error matrix and Producer/User accuracy results of April 2017 image. (Random Forest)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	30	0	0	0	0	0	1	1
Wet/Irrigated	0	29	0	1	0	0	0.967	0.906
Bare	0	1	31	0	0	0	0.969	0.838
Woodland	0	0	2	27	0	1	0.900	0.871
Grassland	0	1	2	1	26	0	0.867	1
Shrubland	0	1	2	2	0	25	0.833	0.962
Overall Accuracy							0.923	0.930

The results from the February image have an overall accuracy of 0.945 and Kappa value of 0.934, showing that the classifier performed as predicted by the Kappa value. Observing the trend of previous error matrices created from Random Forest classifier results shows that water had had a trend of having 100% accuracy when being classified. Though the image used in this classification is from the wet season, overall user accuracies seem to be rather higher as compared to other user accuracies previously calculated.

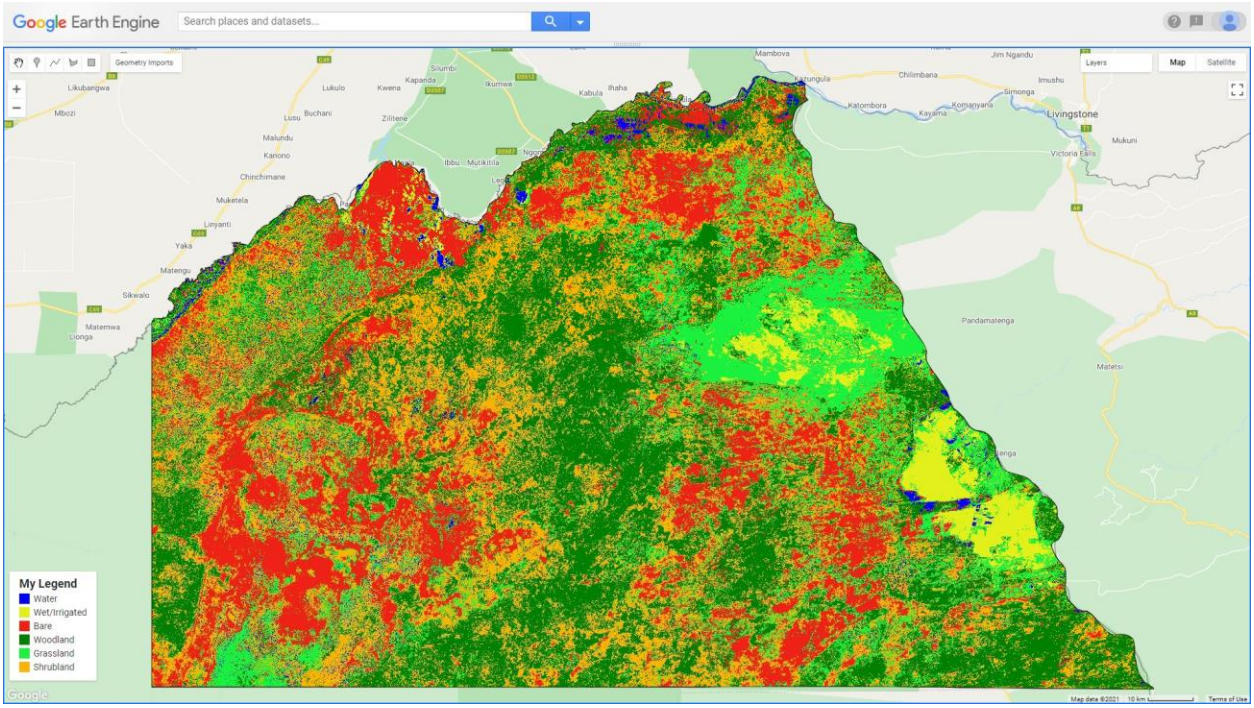


Figure 4.15 Results of Random Forest classifier on image from February 2019 in GEE's API

Table 4.15 Error matrix and Producer/User accuracy results of February 2019 image. (Random Forest)

	Water	Wet/Irrigated	Bare	Woodland	Grassland	Shrubland	Producer Accuracy	User Accuracy
Water	29	0	1	0	0	0	0.967	1
Wet/Irrigated	0	30	0	0	0	0	1	0.968
Bare	0	0	32	0	0	0	1	0.914
Woodland	0	0	0	28	0	2	0.933	0.933
Grassland	0	1	1	1	27	0	0.900	0.931
Shrubland	0	0	1	1	2	26	0.867	0.929
Overall Accuracy							0.945	0.946

4.4 Comparative Analysis of GEE Classifiers

The goal of each classifier is to create a classification scheme using different mathematical and operational approaches. Comparison of the classifiers is important to achieve the best results given that the toolkit created in this study is geared towards non-specialist users. To properly compare the classifiers, the given Kappa values calculated from each classifier will be inputs of a Z-Test and McNemar's chi-squared test which are used in earlier studies to help show the significance of each classifiers results to determine which classifier had performed the best when utilized (De Leeuw et al., 2006). The overall accuracy of each classifier will also be inputs into a separate Z-Test to compare how each classifier performed.

The pairwise results of Naïve Bayes and SVM classifiers are shown in Table 4.18, which shows that the statistical significance of each image classified. The resulting p-values and z-score leads to rejection of the null hypothesis which shows that there is a difference in the classifiers in this pairwise comparison. The z-score being negative shows that the SVM classifier has better classification results as compared to the Naïve Bayes classifier, in turning meaning that the SVM classifier is superior to the Naïve Bayes classifier when comparing with Z-Test. Results of McNemar's chi-squared test, shown in Table 4.19, shows comparable results to the Z-Test which leads to rejection of the null hypothesis.

The pairwise results of Naïve Bayes and Random Forest are also shown in Table 4.18. The resulting z-score and p-value suggest rejection of the null hypothesis which shows the superiority of one classifier. The value of the z-score being negative shows that Random Forest classifier is more superior than the Naïve Bayes classifier. The results of McNemar's test, Figure 4.19, also leads to rejection of the null hypothesis, which also shows difference between each classifier.

When analyzing the results for Random Forest and SVM, the z-score and p-value shows that we cannot reject the null hypothesis which shows that there is no significant difference between these classifiers. This meaning that both classifiers performed similarly, but neither is superior when compared against each other. Though the results of the McNemar's chi-squared test led to rejection of the null hypothesis.

The results of the Z-Test comparing overall accuracy shows similar results as the Kappa value Z-Test comparison where the comparison between Naïve Bayes and SVM as well as the comparison between Naïve Bayes and Random Forest lead to the reject of the null hypothesis meaning there are differences between the classifiers that are being compared. While the comparison between Random Forest and SVM results in failure to reject the null hypothesis meaning that there is no difference in the classifiers.

Table 4.16 Statistics of Machine Learning Classifiers (Naïve Bayes (NB), Support Vector Machine (SVM), Random Forest (RF))

Image	Overall Accuracy (NB)	Kappa Value (NB)	Overall Accuracy (SVM)	Kappa Value (SVM)	Overall Accuracy (RF)	Kappa Value (RF)
August 2016	0.852	0.822	1	1	0.984	0.980
July 2017	0.676	0.611	0.879	0.855	0.923	0.907
September 2018	0.709	0.651	0.896	0.875	0.962	0.954
April 2017	0.670	0.604	0.863	0.835	0.923	0.908
February 2019	0.533	0.439	0.852	0.822	0.945	0.934

Table 4.17 Results of Pairwise Z-Test of Classifiers' Overall Accuracy and p-value

	NB-SVM (p value)	NB-RF (p value)	RF-SVM (p value)
August 2016	-1.43 (.07636)	-1.27 (.10204)	-0.16 (.43644)
July 2017	-1.95 (.02559)	-2.38 (.00866)	0.43 (.66640)
September 2018	-2.44 (.00734)	-2.44 (.00734)	0.64 (.73891)
April 2017	-1.85 (.03216)	-2.44 (.00734)	0.58 (.71904)
February 2019	-3.06 (.00111)	-3.96 (.00004)	0.90 (.81594)

Table 4.18 Results of Pairwise Z-Test of Classifiers' Kappa Value and p-value

	NB-SVM (p value)	NB-RF (p value)	RF-SVM (p value)
August 2016	-1.72 (.04272)	-1.52 (.06426)	-0.19 (.42465)
July 2017	-2.22 (.01321)	-2.93 (.00169)	0.71 (.76115)
September 2018	-3.68 (.00012)	-4.76 (<.00003)	1.09 (.86214)
April 2017	-2.34 (.00964)	-2.85 (.00219)	0.5 (.69146)
February 2019	-2.15 (.01578)	-2.92 (.00175)	0.77 (.77935)

Table 4.19 Results of McNemar's Chi-squared Test.

	NB-SVM (p value)	NB-RF (p value)	RF-SVM (p value)
August 2016	113.47 (<.00003)	110.68 (<.00003)	177.01 (<.00003)
July 2017	45.662 (<.00003)	51.383 (<.00003)	110.66 (<.00003)
September 2018	55.005 (<.00003)	64.215 (<.00003)	123.84 (<.0003)
April 2017	42.47 (<.00003)	50.215 (<.00003)	104.48 (<.00003)
February 2019	19.837 (<.00003)	28.778 (<.00003)	104.2 (<.00003)

4.5 Comparative Analysis on Outside Software

The amount of data being created and stored within databases is showing the necessity for an innovative method to access what is desired for research purposes. Google Earth Engine allows users to do such a thing with their cloud-based technology, it also has built-in image analysis that users can utilize as well. One of the aims of this study was to compare the results of the classifiers that can be used within Google Earth Engine to a classifier from a different software.

The reference image that was utilized to create the classifier for this study was created by using ArcMap, a conventional mapping software. The overall results of the classification of the same study area and same six landcover types were comparable to that which was created within Google Earth Engine. Though there are differences in the time periods that were used in each study, it is believed that image classification within Google Earth Engine, though more technologically intensive, has comparable results to that of image classification performed within ArcMap. By looking at the statistical results of each image classification done, the study completed utilizing ArcMap had an overall accuracy of 0.867 and Kappa of 0.832. While the 4 images used within Google Earth Engine had a range of overall accuracy from 0.923 to 0.984 and Kappa values ranging from 0.907 to 0.980.

Using a qualitative analysis of the differing methods shows that these results are almost comparable though the process to classify images in each study had varying steps and procedures to accomplish their goal in classifying the study region. A more uniform technique on image classification for comparison would be more preferred rather than comparing raw statistical values, though these two values are some of most looked at statistics when comparing classifications using remote sensing.

Chapter 5 - Discussion

5.1 Study Conclusion

One of the reasons that makes this research important was to test the viability of utilizing Google Earth Engine for image analysis and landcover classification using non-conventional image sampling methods. The toolkit created in this study incorporates a simple image gathering method along with sample points for any image that the user desires to have classified for the predefined study region. Users with some knowledge of Google Earth Engines would be able to gather their own sample points and input them within the tool to run their own sample analysis to minimize any bias from the original tool.

The comparison between classifiers within Google Earth Engines shown from the results of each classifier ran shows that the Random Forest and SVM are quantitatively better than Naïve Bayes. But due to some suspiciously high accuracy results from the SVM classifier, the classifier of choice to use from this study would be Random Forest. The results of the Random Forest classifier were comparable to SVM without the suspiciously high accuracy of 100% for some classifications. And when comparing the Random Forest classifier to Naïve Bayes, there are notable differences qualitatively and quantitatively, which makes the Random Forest standout for the best classifier used in this study and would be the classifier recommended to users for future classifications.

The results of the tool showed to be qualitatively comparable to the results shown in the reference study that utilized ArcGIS. Some issues arose when collecting pooled samples where the limitation on amount of pooled sampling to occur had been hindered to pooled samples from one image rather than a collection of samples from all the images used in the study. There were some problems identified in the study for when the classifier is introduced to clouds within the

imagery where the clouds had adverse effects on the output classification. Introducing a method that masks out clouds from the image would be beneficial to creating a more accurate and efficient classifier within Google Earth Engine. Another notable result from the toolkit would be that one of the classifiers, SVM, had shown a result of having 100% accuracy, which is suspiciously high. This may be due to some unintentional bias in the sample points that were used for accuracy assessment. Elimination or minimization of biased sample points for accuracy assessment would be an important change for furthering the viability of using this toolkit for analysis.

The tool's strength is that it is easy to learn once the user can understand how to add and takeaway necessary elements to change to a more suitable outcome or changing the data used for each run. Another strength of this toolkit is the ability to provide sample points for other users so that the toolkit can be shared without having to retrain the classifiers for each new user. The last strength to be highlighted is the toolkit's runtime is significantly faster than those in conventional software such as ArcGIS and ENVI. The strength of using a pooled sample approach for the toolkit would be to help classify each image used and potentially eliminate bias from the accuracy results. Pooled sampling also helps with eliminating the necessity to recollect sample points while using a different Landsat image. The approach of using multiple classifiers is to display the viability and ease of access to different types of classification schemes. Since the classifiers used in this study are the main classifiers used in remote sensing it helps introduce these classifiers to non-specialist users and gives them an idea of how each of the classifiers would perform under similar tasks for future use.

Some of the weaknesses to be highlighted are that there is little to no documentation on Google Earth Engine itself, and the documentation that is available assumes that the user has

knowledge in using Google Earth Engines or remote sensing itself. Another weakness would be the output imagery being a much coarser image than those used prior to classification due to processing limitations of Google Earth Engines. The necessity for users to be familiar with coding, especially in Python or Java, is a weakness that could limit new users to this software for more advanced techniques and usages outside of this toolkit or to even enhance the toolkit. Though GEE has a rather time efficient, there are some problems in transparency when it comes to classification. Again, noting that the SVM classifier had run at 100% accuracy, there may be some problems that are not easily fixed or noticed for those who are not familiar with remote sensing or GEE itself where unintentional biases in the data are present. The inability for users of GEE to identify these issues may lead to results that are not as predicted or problems in classification that are not easily fixed.

The addition of more sample points as well as sampling points from other images used for classification would likely lead to less unintentional sample bias for accuracy assessment. Providing more sample points would also potentially show different results as shown above in the results section as there would be less room for bias from sample points from different images. As noted in the results section, there are problems with this classifier when introduced to different season imagery. This shows that the classifier cannot handle wet season imagery efficiently, thus the necessity to create a classifier for strictly wet season imagery may be necessary. Inclusion of cloud filtering within the toolkit itself would be beneficial to future classification as well as opening the possibility of using a wider variety of imagery. Some more follow up studies done to test the ease of using this toolkit would be to incorporate the toolkit within a study of non-specialist users of remote sensing. Another way that could test the viability of this software would be to centralize the toolkit around a different case study to show the

versatility of toolkits that could be created from within Google Earth Engine. A combination of the two tests could show how well the toolkit and software is for new and non-specialist users.

Bibliography

Chen, F., Wang, K., Van de Voorde, T., & Tang, T. F. (2017). Mapping urban land cover from high spatial resolution hyperspectral data: An approach based on simultaneously unmixing similar pixels with jointly sparse spectral mixture analysis. *Remote Sensing of Environment*, *196*, 324-342.

Belgiu, M., & Drăguţ, L. (2016). Random forest in remote sensing: A review of applications and future directions. *ISPRS journal of photogrammetry and remote sensing*, *114*, 24-31.

Bellinaso, H., Demattê, J. A. M., & Romeiro, S. A. (2010). Soil spectral library and its use in soil classification. *Revista Brasileira de Ciência do Solo*, *34*(3), 861-870.

Boiman, O., Shechtman, E., & Irani, M. (2008, June). In defense of nearest-neighbor based image classification. In 2008 IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-8). IEEE.

Boori, M. S., Paringer, R. A., Choudhary, K., & Kupriyanov, A. V. (2018). Comparison of hyperspectral and multi-spectral imagery to building a spectral library and land cover classification performanc. *Компьютерная оптика*, *42*(6).

Bosch, A., Zisserman, A., & Munoz, X. (2007, October). Image classification using random forests and ferns. In 2007 IEEE 11th international conference on computer vision (pp. 1-8). Ieee.

Braget, M. P. (2017). A novel approach to mapping flooding extent in the Chobe River Basin from 2014 to 2016 using a training library (Doctoral dissertation, Kansas State University).

Braget, M. P., Goodin, D. G., Wang, J., Hutchinson, J. M., & Alexander, K. (2018). Flooded area classification using pooled training samples: an example from the Chobe River Basin, Botswana. Journal of Applied Remote Sensing, 12(2), 026033.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.

Brisco, B., Short, N., Sanden, J. V. D., Landry, R., & Raymond, D. (2009). A semi-automated tool for surface water mapping with RADARSAT-1. *Canadian Journal of Remote Sensing*, *35*(4), 336-344.

Cloutis, E. A. (1996). Review article hyperspectral geological remote sensing: evaluation of analytical techniques. *International Journal of Remote Sensing*, *17*(12), 2215-2242.

De Leeuw, J., Jia, H., Yang, L., Liu, X., Schmidt, K., & Skidmore, A. K. (2006). Comparing accuracy assessments to infer superiority of image classification methods. *International Journal of Remote Sensing*, *27*(1), 223-232.

Du, P., Samat, A., Waske, B., Liu, S., & Li, Z. (2015). Random forest and rotation forest for fully polarized SAR image classification using polarimetric and spatial features. *ISPRS Journal of Photogrammetry and Remote Sensing, 105*, 38-53.

Foody, G. M. (2020). Explaining the unsuitability of the kappa coefficient in the assessment and comparison of the accuracy of thematic maps obtained by image classification. Remote Sensing of Environment, 239, 111630.

Fox, J. T., Vandewalle, M. E., & Alexander, K. A. (2017). Land cover change in northern botswana: the influence of climate, fire, and elephants on semi-arid savanna woodlands. Land, 6(4), 73.

Gounaridis, D., Apostolou, A., & Koukoulas, S. (2016). Land cover of Greece, 2010: a semi-automated classification using random forests. *Journal of maps*, *12*(5), 1055-1062.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote sensing of Environment*, *202*, 18-27.

Grippa, T., Lennert, M., Beaumont, B., Vanhuysse, S., Stephenne, N., & Wolff, E. (2017). An open-source semi-automated processing chain for urban object-based classification. *Remote Sensing*, *9*(4), 358.

Jakkula, V. (2006). Tutorial on support vector machine (svm). *School of EECS, Washington State University*, *37*.

John, A. R., & Xiuping, J. (2006). Remote sensing digital image analysis. *J. Xiuping.– Heidelberg: Springer*.

Joyce, James, "Bayes' Theorem", *The Stanford Encyclopedia of Philosophy* (Fall 2021 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/fall2021/entries/bayes-theorem/>Latifovic, R., Pouliot, D., & Campbell, J. (2018). Assessment of convolution neural networks for surficial geology mapping in the South Rae geological region, Northwest Territories, Canada. *Remote sensing*, *10*(2), 307.

Kavzoglu, T., & Colkesen, I. (2009). A kernel functions analysis for support vector machines for land cover classification. *International Journal of Applied Earth Observation and Geoinformation*, *11*(5), 352-359.

Liu, Y., Zhang, B., Wang, L. M., & Wang, N. (2013). A self-trained semisupervised SVM approach to the remote sensing land cover classification. *Computers & Geosciences*, *59*, 98-107.

Markham, B. L., & Helder, D. L. (2012). Forty-year calibrated record of earth-reflected radiance from Landsat: A review. *Remote Sensing of* Environment, *122*, 30-40.Mateo-García, G., Gómez-Chova, L., Amorós-López, J., Muñoz-Marí, J., & Camps-Valls, G. (2018). Multitemporal cloud masking in the Google Earth Engine. *Remote Sensing, 10(7),* 1079.

Maxwell, A. E., Warner, T. A., & Fang, F. (2018). Implementation of machine-learning classification in remote sensing: An applied review. International Journal of Remote Sensing, 39(9), 2784-2817.

Mutanga O, Kumar L. Google Earth Engine Applications. *Remote Sensing*. 2019; 11(5):591. https://doi.org/10.3390/rs11050591

Naidoo, R., Chase, M. J., Beytell, P., Du Preez, P., Landen, K., Stuart-Hill, G., & Taylor, R. (2016). A newly discovered wildlife migration in Namibia and Botswana is the longest in Africa. Oryx, 50(1), 138-146.

Nasarudin, N. E. M., & Shafri, H. Z. M. (2011). Development and utilization of urban spectral library for remote sensing of urban environment. *Journal of Urban and Environmental Engineering*, *5*(1), 44-56.

Nidamanuri, R. R., & Zbell, B. (2011). Transferring spectral libraries of canopy reflectance for crop classification using hyperspectral remote sensing data. *biosystems engineering*, *110*(3), 231-246.

Otukei, J. R., & Blaschke, T. (2010). Land cover change assessment using decision trees, support vector machines and maximum likelihood classification algorithms. *International Journal of Applied Earth Observation and Geoinformation*, *12*, S27-S31.

Perumal, K., & Bhaskaran, R. (2010). Supervised classification performance of multispectral images. *arXiv preprint arXiv:1002.4046*.

Pricope, N. G., & Binford, M. W. (2012). A spatio-temporal analysis of fire recurrence and extent for semi-arid savanna ecosystems in southern Africa using moderate-resolution satellite imagery. Journal of environmental management, 100, 72-85.

Rao, N. R., Garg, P. K., & Ghosh, S. K. (2007). Development of an agricultural crops spectral library and classification of crops at cultivar level using hyperspectral data. *Precision Agriculture*, *8*(4), 173-185.

Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

Rodriguez-Galiano, V. F., Ghimire, B., Rogan, J., Chica-Olmo, M., & Rigol-Sanchez, J. P. (2012). An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, *67*, 93-104.

Ruuska, S., Hämäläinen, W., Kajava, S., Mughal, M., Matilainen, P., & Mononen, J. (2018). Evaluation of the confusion matrix method in the validation of an automated system for measuring feeding behaviour of cattle. *Behavioural processes*, *148*, 56-62.

Schetselaar, E. M., Harris, J. R., Lynds, T., & De Kemp, E. A. (2007). Remote predictive mapping 1. Remote predictive mapping (RPM): a strategy for geological mapping of Canada's north. *Geoscience Canada*, *34*(3-4), 93-111.

Schwert, B., Rogan, J., Giner, N. M., Ogneva-Himmelberger, Y., Blanchard, S. D., & Woodcock, C. (2013). A comparison of support vector machines and manual change detection for land-cover map updating in Massachusetts, USA. *Remote sensing letters*, *4*(9), 882-890.

Shao, Y., & Lunetta, R. S. (2012). Comparison of support vector machine, neural network, and CART algorithms for the land-cover classification using limited training data points. *ISPRS Journal of Photogrammetry and Remote Sensing*, *70*, 78-87.

Story, M., & Congalton, R. G. (1986). Accuracy assessment: a user's perspective. *Photogrammetric Engineering and remote sensing*, *52*(3), 397-399.

Thai, L. H., Hai, T. S., & Thuy, N. T. (2012). Image classification using support vector machine and artificial neural network. International Journal of Information Technology and Computer Science, 4(5), 32-38.

Timofte, R., Tuytelaars, T., & Van Gool, L. (2012, November). Naive bayes image classification: beyond nearest neighbors. In Asian Conference on Computer Vision (pp. 689-703). Springer, Berlin, Heidelberg.

van Asselen, S., & Seijmonsbergen, A. C. (2006). Expert-driven semi-automated geomorphological mapping for a mountainous area using a laser DTM. *Geomorphology*, *78*(3-4), 309-320.

Vellido Alcacena, A., Martín, J. D., Rossi, F., & Lisboa, P. J. (2011). Seeing is believing: The importance of visualization in real-world machine learning applications. In *Proceedings: 19th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2011: Bruges, Belgium, April 27-28-29, 2011* (pp. 219-226).

Wang, Y., Liu, L., Hu, Y., Li, D., & Li, Z. (2016). Development and validation of the Landsat-8 surface reflectance products using a MODIS-based per-pixel atmospheric correction method. *International Journal of Remote Sensing*, *37*(6), 1291-1314.

Xie, Y., Sha, Z., & Yu, M. (2008). Remote sensing imagery in vegetation mapping: a review. *Journal of plant ecology*, *1*(1), 9-23.

Zomer, R. J., Trabucco, A., & Ustin, S. L. (2009). Building spectral libraries for wetlands land cover classification and hyperspectral remote sensing. *Journal of environmental management*, *90*(7), 2170-2177.

Appendix A - Google Earth Engine Code

Naïve Bayes Code:

```
//Centers map on Chobe ------------------------------------------------------------------------
Map.centerObject(chobe,10)

//Adds Chobe Boundary -------------------------------------------------------------------------
Map.addLayer(chobe)

//Bands used for classifier -------------------------------------------------------------------
var bands = ['B1','B2', 'B3', 'B4', 'B5', 'B6', 'B7'];

var newfc =
Water.merge(Bare).merge(Woodland).merge(Grassland).merge(Shrubland).merge(WetIrrigated)

var valfc = wVal.merge(bVal).merge(woodVal).merge(gVal).merge(sVal).merge(wetVal)

// Define a palette for the Land Use classification.
var palette = [
  '0000FF', // water (0)  // blue
  'e3f01b', //wetirrigated(1) //yellow
  'ee2217', // bare(2) //red
  '008000', //  woodland (3) // dark green
  '1bf23f', // grassland(4) // light green
  'ffb400' //shrubland(5) //orange
];

//Filter image collection ---------------------------------------------------------------------
var imageTrain1 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi1)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain2 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi2)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain3 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi3)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain4 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi4)
```

```
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
//Mosaics images together to make one image ----------------------------------------------------
var mosaic = ee.ImageCollection([
  imageTrain1.clip(chobe),
  imageTrain2.clip(chobe),
  imageTrain3.clip(chobe),
  imageTrain4.clip(chobe)
]);

var compTrain = mosaic.median();

var imageTest1 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi1)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest2 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi2)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest3 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi3)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest4 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi4)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
//Mosaics images together to make one image ----------------------------------------------------
var mosaicTest = ee.ImageCollection([
  imageTest1.clip(chobe),
  imageTest2.clip(chobe),
  imageTest3.clip(chobe),
  imageTest4.clip(chobe)
]);

var compTest = mosaicTest.median();

var vizParams = {
  bands: ['B5', 'B4', 'B3'],
  min: 0,
```

```
  max: 3000,
};
Map.addLayer(compTrain,  {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'Train True Color')
Map.addLayer(compTrain, vizParams, 'Train False Color')

Map.addLayer(compTest,  {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'Test True Color')
Map.addLayer(compTest, vizParams, 'Test False Color')


//Unsupervised Classification ---------------------------------------------------------------------
var training1 = compTrain.sample({
  region: chobe,
  scale: 150,
  numPixels: 1000
});
// Instantiate the clusterer and train it.
var clusterer = ee.Clusterer.wekaKMeans(5).train(training1);

// Cluster the input using the trained clusterer.
var result1 = compTrain.cluster(clusterer);

// Display the clusters with random colors.
Map.addLayer(result1.randomVisualizer(), {}, 'clusters1');

//Supervised classification ------------------------------------------------------------------

// Sample the input imagery to get a FeatureCollection of training data.
var supTraining = compTrain.select(bands).sampleRegions({
  collection: newfc,
  properties: ['landcover'],
  scale: 30
});

var valTraining = compTrain.select(bands).sampleRegions({
  collection: valfc,
  properties: ['LC'],
  scale: 30
});


var trainingPartition = supTraining.randomColumn('random');
var testingPartition = valTraining.randomColumn('random');

print(trainingPartition.size())
print(testingPartition.size())
```

```
// Make a Random Forest classifier and train it.
var trainedClassifier = ee.Classifier.smileNaiveBayes(6).train({
  features: trainingPartition,
  classProperty: 'landcover',
  inputProperties: bands
});

// Classify the input imagery.
var trainedClassified = compTrain.select(bands).classify(trainedClassifier);

// Display the classification result and the input image.
Map.addLayer(trainedClassified, {min: 0, max: 5, palette: palette}, 'Train Training RF');


var trainAccuracy = trainedClassifier.confusionMatrix();
print('Resubstitution error matrix: ', trainAccuracy);
print('Training overall accuracy: ', trainAccuracy.accuracy());
print('Training Kappa Value: ', trainAccuracy.kappa());
print('Training Producer Accuracy: ', trainAccuracy.producersAccuracy());
print('Training Consumer Accuracy: ', trainAccuracy.consumersAccuracy());

// Classify validation set ----------------------------------------------------------------
var testClassifier = ee.Classifier.smileNaiveBayes(6).train({
  features: testingPartition,
  classProperty: 'LC',
  inputProperties: bands
});

var testClassified = compTrain.select(bands).classify(testClassifier);
Map.addLayer(testClassified, {min: 0, max: 5, palette: palette}, 'Train Validation RF');

var validated = testingPartition.classify(testClassifier)
var test_accuracy = validated.errorMatrix('LC', 'classification')
print('Validation error matrix: ', test_accuracy);
print('Validation overall accuracy: ', test_accuracy.accuracy());
print('Test Kappa Value: ', test_accuracy.kappa());
print('Test Producer Accuracy: ', test_accuracy.producersAccuracy());
print('Test Consumer Accuracy: ', test_accuracy.consumersAccuracy());


//**************************TEST IMAGE
CLASSIFIER*****************************************

// Sample the input imagery to get a FeatureCollection of training data.
var testTrain = compTest.select(bands).sampleRegions({
  collection: newfc,
```

```
  properties: ['landcover'],
  scale: 30

});

var valTest = compTest.select(bands).sampleRegions({
  collection: valfc,
  properties: ['LC'],
  scale: 30
});


var testTrainingPartition = testTrain.randomColumn('random');
var testTestingPartition = valTest.randomColumn('random');

print(testTrainingPartition.size())
print(testTestingPartition.size())



// Make a Random Forest classifier and train it.
var testTrainedClassifier = ee.Classifier.smileNaiveBayes(6).train({
  features: testTrainingPartition,
  classProperty: 'landcover',
  inputProperties: bands
});

// Classify the input imagery.
var testTrainedClassified = compTest.classify(testTrainedClassifier);

// Display the classification result and the input image.
Map.addLayer(testTrainedClassified, {min: 0, max: 5, palette: palette}, 'Test Training RF');


var testTrainAccuracy = testTrainedClassifier.confusionMatrix();
print('Resubstitution error matrix for Test Image: ', testTrainAccuracy);
print('Training overall accuracy for Test Image: ', testTrainAccuracy.accuracy());
print('Test Training Kappa Value: ', testTrainAccuracy.kappa());
print('Train Producer Accuracy: ', testTrainAccuracy.producersAccuracy());
print('Train Consumer Accuracy: ', testTrainAccuracy.consumersAccuracy());

// Classify validation set ----------------------------------------------------------------
var reTestClassifier = ee.Classifier.smileNaiveBayes(6).train({
  features: testTestingPartition,
  classProperty: 'LC',
  inputProperties: bands
```

```
});

var reTestClassified = compTest.classify(reTestClassifier);
Map.addLayer(reTestClassified, {min: 0, max: 5, palette: palette}, 'Test Validation RF');

var testValidated = testTestingPartition.classify(reTestClassifier)
var reTest_accuracy = testValidated.errorMatrix('LC', 'classification')
print('Validation error matrix for Test Image: ', reTest_accuracy);
print('Validation overall accuracy for Test Image: ', reTest_accuracy.accuracy());
print('Validation Kappa Value: ', reTest_accuracy.kappa());
print('Test Producer Accuracy: ', reTest_accuracy.producersAccuracy());
print('Test Consumer Accuracy: ', reTest_accuracy.consumersAccuracy());


//*****************************LEGEND(Do not
Touch)*************************************************************
// set position of panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'My Legend',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
    }
});

// Add the title to the panel
legend.add(legendTitle);

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {

      // Create the label that is actually the colored box.
      var colorBox = ui.Label({
        style: {
          backgroundColor: '#' + color,
          // Use padding to give the box height and width.
```

```
        padding: '8px',
        margin: '0 0 4px 0'
      }
    });

    // Create the label filled with the description text.
    var description = ui.Label({
      value: name,
      style: {margin: '0 0 4px 6px'}
    });

    // return the panel
    return ui.Panel({
      widgets: [colorBox, description],
      layout: ui.Panel.Layout.Flow('horizontal')
    });
};



// name of the legend
var names = ['Water','Wet/Irrigated','Bare', 'Woodland', 'Grassland', 'Shrubland'];

// Add color and and names
for (var i = 0; i < 6; i++) {
  legend.add(makeRow(palette[i], names[i]));
  }

// add legend to map (alternatively you can also print the legend to the console)
Map.add(legend);
```

SVM Code:

```
//Centers map on Chobe ------------------------------------------------------------------------
Map.centerObject(chobe,10)

//Adds Chobe Boundary -------------------------------------------------------------------------
Map.addLayer(chobe)

//Bands used for classifier -------------------------------------------------------------------
var bands = ['B1','B2', 'B3', 'B4', 'B5', 'B6', 'B7'];

var newfc =
Water.merge(Bare).merge(Woodland).merge(Grassland).merge(Shrubland).merge(WetIrrigated)
```

```
var valfc = wVal.merge(bVal).merge(woodVal).merge(gVal).merge(sVal).merge(wetVal)

// Define a palette for the Land Use classification.
var palette = [
  '0000FF', // water (0)  // blue
  'e3f01b', //wetirrigated(1) //yellow
  'ee2217', // bare(2) //red
  '008000', //  woodland (3) // dark green
  '1bf23f', // grassland(4) // light green
  'ffb400' //shrubland(5) //orange
];

//Filter image collection ------------------------------------------------------------------------
var imageTrain1 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi1)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain2 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi2)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain3 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi3)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain4 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi4)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
//Mosaics images together to make one image ------------------------------------------------------
var mosaic = ee.ImageCollection([
  imageTrain1.clip(chobe),
  imageTrain2.clip(chobe),
  imageTrain3.clip(chobe),
  imageTrain4.clip(chobe)
]);

var compTrain = mosaic.median();

var imageTest1 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi1)
    .filterDate('2019-02-01', '2019-02-28')
```

```
    .sort('CLOUD_COVER')
    .first());
var imageTest2 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi2)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest3 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi3)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest4 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi4)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
//Mosaics images together to make one image ----------------------------------------------------
var mosaicTest = ee.ImageCollection([
  imageTest1.clip(chobe),
  imageTest2.clip(chobe),
  imageTest3.clip(chobe),
  imageTest4.clip(chobe)
]);

var compTest = mosaicTest.median();

var vizParams = {
  bands: ['B5', 'B4', 'B3'],
  min: 0,
  max: 3000,
};
Map.addLayer(compTrain,  {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'Train True Color')
Map.addLayer(compTrain, vizParams, 'Train False Color')

Map.addLayer(compTest,  {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'Test True Color')
Map.addLayer(compTest, vizParams, 'Test False Color')


//Unsupervised Classification ------------------------------------------------------------------
var training1 = compTrain.sample({
  region: chobe,
  scale: 150,
  numPixels: 1000
});
// Instantiate the clusterer and train it.
```

```
var clusterer = ee.Clusterer.wekaKMeans(5).train(training1);

// Cluster the input using the trained clusterer.
var result1 = compTrain.cluster(clusterer);

// Display the clusters with random colors.
Map.addLayer(result1.randomVisualizer(), {}, 'clusters1');

//Supervised classification ----------------------------------------------------------------

// Sample the input imagery to get a FeatureCollection of training data.
var supTraining = compTrain.select(bands).sampleRegions({
  collection: newfc,
  properties: ['landcover'],
  scale: 30
});

var valTraining = compTrain.select(bands).sampleRegions({
  collection: valfc,
  properties: ['LC'],
  scale: 30
});


var trainingPartition = supTraining.randomColumn('random');
var testingPartition = valTraining.randomColumn('random');

print(trainingPartition.size())
print(testingPartition.size())

// Make a Random Forest classifier and train it.
var trainedClassifier = ee.Classifier.libsvm().train(trainingPartition, 'landcover', bands);

// Classify the input imagery.
var trainedClassified = compTrain.select(bands).classify(trainedClassifier);

// Display the classification result and the input image.
Map.addLayer(trainedClassified, {min: 0, max: 5, palette: palette}, 'Train Training RF');


var trainAccuracy = trainedClassifier.confusionMatrix();
print('Resubstitution error matrix: ', trainAccuracy);
print('Training overall accuracy: ', trainAccuracy.accuracy());
print('Training Kappa Value: ', trainAccuracy.kappa());
print('Training Producer Accuracy: ', trainAccuracy.producersAccuracy());
print('Training Consumer Accuracy: ', trainAccuracy.consumersAccuracy());
```

```
// Classify validation set ----------------------------------------------------------------
var testClassifier = ee.Classifier.libsvm().train(testingPartition, 'LC', bands);

var testClassified = compTrain.select(bands).classify(testClassifier);
Map.addLayer(testClassified, {min: 0, max: 5, palette: palette}, 'Train Validation RF');

var validated = testingPartition.classify(testClassifier)
var test_accuracy = validated.errorMatrix('LC', 'classification')
print('Validation error matrix: ', test_accuracy);
print('Validation overall accuracy: ', test_accuracy.accuracy());
print('Test Kappa Value: ', test_accuracy.kappa());
print('Test Producer Accuracy: ', test_accuracy.producersAccuracy());
print('Test Consumer Accuracy: ', test_accuracy.consumersAccuracy());


//**************************TEST IMAGE
CLASSIFIER*****************************************

// Sample the input imagery to get a FeatureCollection of training data.
var testTrain = compTest.select(bands).sampleRegions({
  collection: newfc,
  properties: ['landcover'],
  scale: 30

});

var valTest = compTest.select(bands).sampleRegions({
  collection: valfc,
  properties: ['LC'],
  scale: 30
});

var testTrainingPartition = testTrain.randomColumn('random');
var testTestingPartition = valTest.randomColumn('random');

print(testTrainingPartition.size())
print(testTestingPartition.size())

// Make a Random Forest classifier and train it.
var testTrainedClassifier = ee.Classifier.libsvm().train(testTrainingPartition, 'landcover', bands);

// Classify the input imagery.
var testTrainedClassified = compTest.classify(testTrainedClassifier);

// Display the classification result and the input image.
```

```
Map.addLayer(testTrainedClassified, {min: 0, max: 5, palette: palette}, 'Test Training RF');


var testTrainAccuracy = testTrainedClassifier.confusionMatrix();
print('Resubstitution error matrix for Test Image: ', testTrainAccuracy);
print('Training overall accuracy for Test Image: ', testTrainAccuracy.accuracy());
print('Test Training Kappa Value: ', testTrainAccuracy.kappa());
print('Train Producer Accuracy: ', testTrainAccuracy.producersAccuracy());
print('Train Consumer Accuracy: ', testTrainAccuracy.consumersAccuracy());

// Classify validation set ----------------------------------------------------------------
var reTestClassifier = ee.Classifier.libsvm().train(testTestingPartition, 'LC', bands);

var reTestClassified = compTest.classify(reTestClassifier);
Map.addLayer(reTestClassified, {min: 0, max: 5, palette: palette}, 'Test Validation RF');

var testValidated = testTestingPartition.classify(reTestClassifier)
var reTest_accuracy = testValidated.errorMatrix('LC', 'classification')
print('Validation error matrix for Test Image: ', reTest_accuracy);
print('Validation overall accuracy for Test Image: ', reTest_accuracy.accuracy());
print('Validation Kappa Value: ', reTest_accuracy.kappa());
print('Test Producer Accuracy: ', reTest_accuracy.producersAccuracy());
print('Test Consumer Accuracy: ', reTest_accuracy.consumersAccuracy());



//*****************************LEGEND(Do not
Touch)*************************************************************
// set position of panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'My Legend',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
    }
});
```

```
// Add the title to the panel
legend.add(legendTitle);

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {

      // Create the label that is actually the colored box.
      var colorBox = ui.Label({
        style: {
          backgroundColor: '#' + color,
          // Use padding to give the box height and width.
          padding: '8px',
          margin: '0 0 4px 0'
        }
      });

      // Create the label filled with the description text.
      var description = ui.Label({
        value: name,
        style: {margin: '0 0 4px 6px'}
      });

      // return the panel
      return ui.Panel({
        widgets: [colorBox, description],
        layout: ui.Panel.Layout.Flow('horizontal')
      });
};



// name of the legend
var names = ['Water','Wet/Irrigated','Bare', 'Woodland', 'Grassland', 'Shrubland'];

// Add color and and names
for (var i = 0; i < 6; i++) {
  legend.add(makeRow(palette[i], names[i]));
  }

// add legend to map (alternatively you can also print the legend to the console)
Map.add(legend);
```

Random Forest Code:

```
//Centers map on Chobe ------------------------------------------------------------------------
```

```
Map.centerObject(chobe,10)

//Adds Chobe Boundary ------------------------------------------------------------------------
Map.addLayer(chobe)

//Bands used for classifier ------------------------------------------------------------------
var bands = ['B1','B2', 'B3', 'B4', 'B5', 'B6', 'B7'];

var newfc =
Water.merge(Bare).merge(Woodland).merge(Grassland).merge(Shrubland).merge(WetIrrigated)

var valfc = wVal.merge(bVal).merge(woodVal).merge(gVal).merge(sVal).merge(wetVal)

// Define a palette for the Land Use classification.
var palette = [
  '0000FF', // water (0)  // blue
  'e3f01b', //wetirrigated(1) //yellow
  'ee2217', // bare(2) //red
  '008000', //  woodland (3) // dark green
  '1bf23f', // grassland(4) // light green
  'ffb400' //shrubland(5) //orange
];

//Filter image collection ---------------------------------------------------------------------
var imageTrain1 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi1)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain2 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi2)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain3 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi3)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
var imageTrain4 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi4)
    .filterDate('2016-08-01', '2016-08-31')
    .sort('CLOUD_COVER')
    .first());
//Mosaics images together to make one image ---------------------------------------------------
var mosaic = ee.ImageCollection([
```

```
  imageTrain1.clip(chobe),
  imageTrain2.clip(chobe),
  imageTrain3.clip(chobe),
  imageTrain4.clip(chobe)
]);

var compTrain = mosaic.median();

var imageTest1 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi1)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest2 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi2)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest3 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi3)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
var imageTest4 = ee.Image(ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(roi4)
    .filterDate('2019-02-01', '2019-02-28')
    .sort('CLOUD_COVER')
    .first());
//Mosaics images together to make one image ----------------------------------------------------
var mosaicTest = ee.ImageCollection([
  imageTest1.clip(chobe),
  imageTest2.clip(chobe),
  imageTest3.clip(chobe),
  imageTest4.clip(chobe)
]);

var compTest = mosaicTest.median();

var vizParams = {
  bands: ['B5', 'B4', 'B3'],
  min: 0,
  max: 3000,
};
Map.addLayer(compTrain,  {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'Train True Color')
Map.addLayer(compTrain, vizParams, 'Train False Color')
```

```
Map.addLayer(compTest,  {bands: ['B4', 'B3', 'B2'], min: 0, max: 3000}, 'Test True Color')
Map.addLayer(compTest, vizParams, 'Test False Color')


//Unsupervised Classification ---------------------------------------------------------------------
var training1 = compTrain.sample({
  region: chobe,
  scale: 150,
  numPixels: 1000
});
// Instantiate the clusterer and train it.
var clusterer = ee.Clusterer.wekaKMeans(5).train(training1);

// Cluster the input using the trained clusterer.
var result1 = compTrain.cluster(clusterer);

// Display the clusters with random colors.
Map.addLayer(result1.randomVisualizer(), {}, 'clusters1');

//Supervised classification ------------------------------------------------------------------

// Sample the input imagery to get a FeatureCollection of training data.
var supTraining = compTrain.select(bands).sampleRegions({
  collection: newfc,
  properties: ['landcover'],
  scale: 30
});

var valTraining = compTrain.select(bands).sampleRegions({
  collection: valfc,
  properties: ['LC'],
  scale: 30
});


var trainingPartition = supTraining.randomColumn('random');
var testingPartition = valTraining.randomColumn('random');

print(trainingPartition.size())
print(testingPartition.size())

// Make a Random Forest classifier and train it.
var trainedClassifier = ee.Classifier.smileRandomForest(6).train({
  features: trainingPartition,
  classProperty: 'landcover',
  inputProperties: bands
```

```
});

// Classify the input imagery.
var trainedClassified = compTrain.select(bands).classify(trainedClassifier);

// Display the classification result and the input image.
Map.addLayer(trainedClassified, {min: 0, max: 5, palette: palette}, 'Train Training RF');


var trainAccuracy = trainedClassifier.confusionMatrix();
print('Resubstitution error matrix: ', trainAccuracy);
print('Training overall accuracy: ', trainAccuracy.accuracy());
print('Training Kappa Value: ', trainAccuracy.kappa());
print('Training Producer Accuracy: ', trainAccuracy.producersAccuracy());
print('Training Consumer Accuracy: ', trainAccuracy.consumersAccuracy());

// Classify validation set ----------------------------------------------------------------
var testClassifier = ee.Classifier.smileRandomForest(6).train({
  features: testingPartition,
  classProperty: 'LC',
  inputProperties: bands
});

var testClassified = compTrain.select(bands).classify(testClassifier);
Map.addLayer(testClassified, {min: 0, max: 5, palette: palette}, 'Train Validation RF');

var validated = testingPartition.classify(testClassifier)
var test_accuracy = validated.errorMatrix('LC', 'classification')
print('Validation error matrix: ', test_accuracy);
print('Validation overall accuracy: ', test_accuracy.accuracy());
print('Test Kappa Value: ', test_accuracy.kappa());
print('Test Producer Accuracy: ', test_accuracy.producersAccuracy());
print('Test Consumer Accuracy: ', test_accuracy.consumersAccuracy());


//**************************TEST IMAGE
CLASSIFIER*****************************************

// Sample the input imagery to get a FeatureCollection of training data.
var testTrain = compTest.select(bands).sampleRegions({
  collection: newfc,
  properties: ['landcover'],
  scale: 30

});
```

```
var valTest = compTest.select(bands).sampleRegions({
  collection: valfc,
  properties: ['LC'],
  scale: 30
});


var testTrainingPartition = testTrain.randomColumn('random');
var testTestingPartition = valTest.randomColumn('random');

print(testTrainingPartition.size())
print(testTestingPartition.size())



// Make a Random Forest classifier and train it.
var testTrainedClassifier = ee.Classifier.smileRandomForest(6).train({
  features: testTrainingPartition,
  classProperty: 'landcover',
  inputProperties: bands
});

// Classify the input imagery.
var testTrainedClassified = compTest.classify(testTrainedClassifier);

// Display the classification result and the input image.
Map.addLayer(testTrainedClassified, {min: 0, max: 5, palette: palette}, 'Test Training RF');


var testTrainAccuracy = testTrainedClassifier.confusionMatrix();
print('Resubstitution error matrix for Test Image: ', testTrainAccuracy);
print('Training overall accuracy for Test Image: ', testTrainAccuracy.accuracy());
print('Test Training Kappa Value: ', testTrainAccuracy.kappa());
print('Train Producer Accuracy: ', testTrainAccuracy.producersAccuracy());
print('Train Consumer Accuracy: ', testTrainAccuracy.consumersAccuracy());

// Classify validation set ----------------------------------------------------------------
var reTestClassifier = ee.Classifier.smileRandomForest(6).train({
  features: testTestingPartition,
  classProperty: 'LC',
  inputProperties: bands
});

var reTestClassified = compTest.classify(reTestClassifier);
Map.addLayer(reTestClassified, {min: 0, max: 5, palette: palette}, 'Test Validation RF');
```

```
var testValidated = testTestingPartition.classify(reTestClassifier)
var reTest_accuracy = testValidated.errorMatrix('LC', 'classification')
print('Validation error matrix for Test Image: ', reTest_accuracy);
print('Validation overall accuracy for Test Image: ', reTest_accuracy.accuracy());
print('Validation Kappa Value: ', reTest_accuracy.kappa());
print('Test Producer Accuracy: ', reTest_accuracy.producersAccuracy());
print('Test Consumer Accuracy: ', reTest_accuracy.consumersAccuracy());



//*****************************LEGEND(Do not
Touch)*****************************************************************
// set position of panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'My Legend',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
    }
});

// Add the title to the panel
legend.add(legendTitle);

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {

      // Create the label that is actually the colored box.
      var colorBox = ui.Label({
        style: {
          backgroundColor: '#' + color,
          // Use padding to give the box height and width.
          padding: '8px',
          margin: '0 0 4px 0'
        }
      });
```

```
    // Create the label filled with the description text.
    var description = ui.Label({
      value: name,
      style: {margin: '0 0 4px 6px'}
    });

    // return the panel
    return ui.Panel({
      widgets: [colorBox, description],
      layout: ui.Panel.Layout.Flow('horizontal')
    });
};



// name of the legend
var names = ['Water','Wet/Irrigated','Bare', 'Woodland', 'Grassland', 'Shrubland'];

// Add color and and names
for (var i = 0; i < 6; i++) {
  legend.add(makeRow(palette[i], names[i]));
  }

// add legend to map (alternatively you can also print the legend to the console)
Map.add(legend);
```

Appendix B - R Statistical Analysis Code

```
install.packages('ggplot2')
install.packages('distributions3')

library(ggplot2)
library(distributions3)

kAugNB <- 0.822
kJulNB <- 0.611
kSepNB <- 0.651
kAprNB <- 0.604
kFebNB <- 0.439

kAugSVM <- 1
kJulSVM <- 0.855
kSepSVM <- 0.875
kAprSVM <- 0.835
kFebSVM <- 0.822

kAugRF <- 0.980
kJulRF <- 0.907
kSepRF <- 0.954
kAprRF <- 0.908
kFebRF <- 0.934

kAugNB2 <- 0.852
kJulNB2 <- 0.676
kSepNB2 <- 0.709
kAprNB2 <- 0.670
kFebNB2 <- 0.533
```

```
kAugSVM2 <- 1
kJulSVM2 <- 0.879
kSepSVM2 <- 0.896
kAprSVM2 <- 0.863
kFebSVM2 <- 0.852

kAugRF2 <- 0.984
kJulRF2 <- 0.923
kSepRF2 <- 0.962
kAprRF2 <- 0.923
kFebRF2 <- 0.945

delta_0 <- 0

# by assumption
sigma_sq_NBag <- 0.00537
sigma_sq_NBju <- 0.00543
sigma_sq_NBsep <- 0.00542
sigma_sq_NBap <- 0.00543
sigma_sq_NBfe <- 0.00548

sigma_sq_SVMag <- 0.00531
sigma_sq_SVMju <- 0.00536
sigma_sq_SVMsep <- 0.00535
sigma_sq_SVMap <- 0.00536
sigma_sq_SVMfe <- 0.00537

sigma_sq_RFag <- 0.00532
sigma_sq_RFju <- 0.00534
sigma_sq_RFsep <- 0.00533
```

```
sigma_sq_RFap <- 0.00534
sigma_sq_RFfe <- 0.00533


N <- 182

#Naive Bayes - SVM
z_statnbsvmag <- (kAugNB - kAugSVM) /
  sqrt(sigma_sq_NBag + sigma_sq_SVMag)

z_statnbsvmju <- (kJulNB - kJulSVM) /
  sqrt(sigma_sq_NBju + sigma_sq_SVMju)

z_statnbsvmsep <- (kSepNB - kSepSVM) /
  sqrt(sigma_sq_NBsep + sigma_sq_SVMsep)

z_statnbsvmap <- (kAprNB - kAprSVM) /
  sqrt(sigma_sq_NBap + sigma_sq_SVMap)

z_statnbsvmfe <- (kFebNB - kFebSVM) /
  sqrt(sigma_sq_NBfe + sigma_sq_SVMfe)

#Naive Bayes - RF
z_statnbRFag <- (kAugNB - kAugRF) /
  sqrt(sigma_sq_NBag + sigma_sq_RFag)

z_statnbRFju <- (kJulNB - kJulRF) /
  sqrt(sigma_sq_NBju + sigma_sq_RFju)

z_statnbRFsep <- (kSepNB - kSepRF) /
  sqrt(sigma_sq_NBsep + sigma_sq_RFsep)
```

```
z_statnbRFap <- (kAprNB - kAprRF) /
  sqrt(sigma_sq_NBap + sigma_sq_RFap)

z_statnbRFfe <- (kFebNB - kFebRF) /
  sqrt(sigma_sq_NBfe + sigma_sq_RFfe)

#SVM - RF
z_statsvmrfag <- (kAugRF - kAugSVM) /
  sqrt(sigma_sq_RFag + sigma_sq_SVMag)

z_statsvmrfju <- (kJulRF - kJulSVM) /
  sqrt(sigma_sq_RFju + sigma_sq_SVMag)

z_statsvmrfsep <- (kSepRF - kSepSVM) /
  sqrt(sigma_sq_RFsep + sigma_sq_SVMag)

z_statsvmrfap <- (kAprRF - kAprSVM) /
  sqrt(sigma_sq_RFap + sigma_sq_SVMag)

z_statsvmrffe <- (kFebRF - kFebSVM) /
  sqrt(sigma_sq_RFfe + sigma_sq_SVMag)



z_statnbsvm
z_statnbrf
z_statsvmrf

mcNBSVMag <- matrix(c(155, 27, 182, 0),
                    nrow = 2,
                    dimnames = list("NB" = c("Correct", "Incorrect"),
```

```
                    "SVM" = c("Correct", "Incorrect")))

mcNBSVMju <- matrix(c(123, 59, 160, 22),
            nrow = 2,
            dimnames = list("NB" = c("Correct", "Incorrect"),
                    "SVM" = c("Correct", "Incorrect")))
mcNBSVMsep <- matrix(c(129, 53, 163, 19),
            nrow = 2,
            dimnames = list("NB" = c("Correct", "Incorrect"),
                    "SVM" = c("Correct", "Incorrect")))
mcNBSVMap <- matrix(c(122, 60, 157, 25),
            nrow = 2,
            dimnames = list("NB" = c("Correct", "Incorrect"),
                    "SVM" = c("Correct", "Incorrect")))
mcNBSVMfe <- matrix(c(97, 85, 155, 27),
            nrow = 2,
            dimnames = list("NB" = c("Correct", "Incorrect"),
                    "SVM" = c("Correct", "Incorrect")))

mcnemar.test(mcNBSVMag)
mcnemar.test(mcNBSVMju)
mcnemar.test(mcNBSVMsep)
mcnemar.test(mcNBSVMap)
mcnemar.test(mcNBSVMfe)

mcNBRFag <- matrix(c(155, 27, 179, 3),
            nrow = 2,
            dimnames = list("NB" = c("Correct", "Incorrect"),
                    "RF" = c("Correct", "Incorrect")))

mcNBRFju <- matrix(c(123, 59, 168, 14),
```

```
                nrow = 2,
                dimnames = list("NB" = c("Correct", "Incorrect"),
                                "RF" = c("Correct", "Incorrect")))
mcNBRFsep <- matrix(c(129, 53, 175, 7),
                nrow = 2,
                dimnames = list("NB" = c("Correct", "Incorrect"),
                                "RF" = c("Correct", "Incorrect")))
mcNBRFap <- matrix(c(122, 60, 168, 14),
                nrow = 2,
                dimnames = list("NB" = c("Correct", "Incorrect"),
                                "RF" = c("Correct", "Incorrect")))
mcNBRFfe <- matrix(c(97, 85, 172, 10),
                nrow = 2,
                dimnames = list("NB" = c("Correct", "Incorrect"),
                                "RF" = c("Correct", "Incorrect")))

mcnemar.test(mcNBRFag)
mcnemar.test(mcNBRFju)
mcnemar.test(mcNBRFsep)
mcnemar.test(mcNBRFap)
mcnemar.test(mcNBRFfe)

mcSVMRFag <- matrix(c(182, 0, 179, 3),
                nrow = 2,
                dimnames = list("NB" = c("Correct", "Incorrect"),
                                "RF" = c("Correct", "Incorrect")))

mcSVMRFju <- matrix(c(160, 22, 168, 14),
                nrow = 2,
                dimnames = list("NB" = c("Correct", "Incorrect"),
                                "RF" = c("Correct", "Incorrect")))
```

```
mcSVMRFsep <- matrix(c(163, 19, 175, 7),
                     nrow = 2,
                     dimnames = list("NB" = c("Correct", "Incorrect"),
                                     "RF" = c("Correct", "Incorrect")))
mcSVMRFap <- matrix(c(157, 25, 168, 14),
                    nrow = 2,
                    dimnames = list("NB" = c("Correct", "Incorrect"),
                                    "RF" = c("Correct", "Incorrect")))
mcSVMRFfe <- matrix(c(155, 27, 172, 10),
                    nrow = 2,
                    dimnames = list("NB" = c("Correct", "Incorrect"),
                                    "RF" = c("Correct", "Incorrect")))
mcnemar.test(mcSVMRFag)
mcnemar.test(mcSVMRFju)
mcnemar.test(mcSVMRFsep)
mcnemar.test(mcSVMRFap)
mcnemar.test(mcSVMRFfe)

#Naive Bayes - SVM
z_statnbsvmag2 <- (kAugNB2 - kAugSVM2) /
  sqrt(sigma_sq_NBag + sigma_sq_SVMag)

z_statnbsvmju2 <- (kJulNB2 - kJulSVM2) /
  sqrt(sigma_sq_NBju + sigma_sq_SVMju)

z_statnbsvmsep2 <- (kSepNB2 - kSepSVM2) /
  sqrt(sigma_sq_NBsep + sigma_sq_SVMsep)

z_statnbsvmap2 <- (kAprNB2 - kAprSVM2) /
  sqrt(sigma_sq_NBap + sigma_sq_SVMap)
```

```
z_statnbsvmfe2 <- (kFebNB2 - kFebSVM2) /
  sqrt(sigma_sq_NBfe + sigma_sq_SVMfe)

#Naive Bayes - RF
z_statnbRFag2 <- (kAugNB2 - kAugRF2) /
  sqrt(sigma_sq_NBag + sigma_sq_RFag)

z_statnbRFju2 <- (kJulNB2 - kJulRF2) /
  sqrt(sigma_sq_NBju + sigma_sq_RFju)

z_statnbRFsep2 <- (kSepNB2 - kSepRF2) /
  sqrt(sigma_sq_NBsep + sigma_sq_RFsep)

z_statnbRFap2 <- (kAprNB2 - kAprRF2) /
  sqrt(sigma_sq_NBap + sigma_sq_RFap)

z_statnbRFfe2 <- (kFebNB2 - kFebRF2) /
  sqrt(sigma_sq_NBfe + sigma_sq_RFfe)

#SVM - RF
z_statsvmrfag2 <- (kAugRF2 - kAugSVM2) /
  sqrt(sigma_sq_RFag + sigma_sq_SVMag)

z_statsvmrfju2 <- (kJulRF2 - kJulSVM2) /
  sqrt(sigma_sq_RFju + sigma_sq_SVMag)

z_statsvmrfsep2 <- (kSepRF2 - kSepSVM2) /
  sqrt(sigma_sq_RFsep + sigma_sq_SVMag)

z_statsvmrfap2 <- (kAprRF2 - kAprSVM2) /
  sqrt(sigma_sq_RFap + sigma_sq_SVMag)
```

```
z_statsvmrffe2 <- (kFebRF2 - kFebSVM2) /
  sqrt(sigma_sq_RFfe + sigma_sq_SVMag)
```