Supporting document – Effects of seasonality and classifier on the accuracy of grazing resource and land degradation maps in a savanna ecosystem

1 Table of Contents

2 Table of Figures and tables

3 Introduction

This technical report is a supporting document for the research paper “Effects of seasonality and classifier on the accuracy of grazing resource and land degradation maps in a savanna ecosystem”. The focus of the report is to document the methodology of the research paper in full, and to present results not found in the research paper. The methods are outlined in such a way that all aspects of the methodology could be repeated as this becoming increasingly important in light of the repeatability crisis (Konkol et al., 2019).

4 Format of Report

All readable text is presented in font – Century - font size 12.
Code from all programming languages is presented in – InaiMathi - font size 10.
Annotation in code is prefixed with – #

5 Software

In as far as was possible open access software and platforms were used in conducting the study.

The following programs were used.

- R version 3.5.3 - IDE R Studio
- Python Version 3.7.3 - IDE Spyder
- JavaScript - Google Earth Engine Code Editor
- Google Earth Pro - Version 7.3

Table 1: R packages and Python modules utilised in the analysis and their description.

Name	Language	Description	Reference
caret	R	Model construction and comparison	Kuhn et al., 2019
date	Python	Date time functions	NA
geopandas	Python	Geographic data processing	NA
ggmap	R	Map production	Kahle, D and Wickham, H., 2012
ggplot2	R	Graphics Package	Wickham 2016
GISTools	R	Spatial data analysis	Brunsdon and Chen 2014
numpy	Python	Data processing	Oliphant 2006
pandas	Python	Data processing	McKinney 2010
raster	R	Raster manipulation and analysis	Hijmans 2019
rasterio	Python	Raster analysis	Gillies 2013
rgdal	R	Formatting and transformation of geographic data	Roger et al 2019
sentinelsat	Python	Download Sentinel-2 imagery	NA
shapely	Python	Data manipulation	NA

snow	R	Paradelle processing	Tierney et al., 2018
sp	R	Classes and methods for spatial data: points, lines, polygons and grids	Pebesma and Bivand 2005
subprocess	Python	Execute terminal commands in python	NA
tidyr	R	Data manipulation and formatting	Wickham and Henry 2019

6 Field Data Collection Protocol

This data collection protocol was written for the resource assessors employed by the South Rift Association of Land Owners, a Kenyan NGO. The resource assessors have limited technical and scientific knowledge, and therefore the language used in the protocol reflects this constraint in efficient communication.

Aim
To inform the resource assessors at Lale'enok Resource Centre how to collect the data required to map several plant species.

Background
Using images taken by satellites it is possible to map species of grass, herb, shrub and trees across large areas. Mapping these types of vegetation is important for analysing changes in the abundance and location of plant species over time. These maps can be used to understand livestock and wildlife movements and grazing, to monitor grazing quality and inform land management decisions.

To map plant species ground reference data are needed. Ground reference data is knowledge of the plant species found at a particular location. This knowledge will allow us to match parts of the image taken from satellite with the information on the ground vegetation.

Materials Required
GPS
Batteries
Measuring Tape
Datasheet Print Out
Vehicle

Before starting this project download all the data from the GPS to a computer and then remove all data from the GPS. This reduces the chance of confusing data points or running out of memory while in the field.

Method
To map the vegetation species, we need GPS coordinates of areas that are at least 20meters by 20meters and are covered by a single species. We also need GPS coordinates of areas that contain mixed grass species, mixed herb species, mixed tree species, mixed herb/grass/shrub/tree species, bare soil and rock.

Before data collection you will need to print out the data sheet found at the end of this document. You may also copy the data sheet with pencil/pen on some blank paper to use if the printer is not working.

You will also need to create a folder on the computer named - Mapping Vegetation. At the end of each day download all of the GPS data, points and tracks, and place them in the folder named - Mapping Vegetation. Inside the Mapping Vegetation folder create another folder and name it with the date for which the data has been collected. Please take a picture of the data sheet (no need to enter the data into excel). And if there is internet send the data and picture to me by email, (freddie_hunter@live.co.uk).

Prosopis Data Collection

Areas of *Prosopis* that can be used in the study need to be at least 20meters by 20meters in size. It is very important that the area is dense with *Prosopis* (Figure 1). If you were to be looking down like a bird on the area, you would mostly see *Prosopis*. No large acacia trees/shrubs can be present. In the data sheet there is a column where you need to estimate the percentage covered by the species.

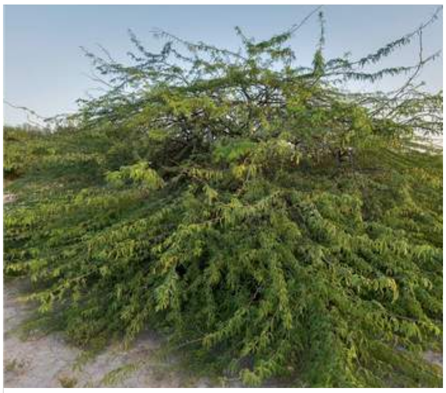


Figure 1: Photograph of a *Prosopis* shrub.

When you find an area that is at least 20meters by 20meteres you will need to go into the area and take a GPS coordinates of the middle of the area. It is very important that you are in the middle of the 20m by 20m area (Figure 2). Hold the GPS high in the air and stand still while you take the coordinates.

It is important that the area is at least 20m by 20m, but you do not need to accurately measure the size of it, just be sure that it is big enough. Use the measuring tape to get a rough estimate if you are unsure. You do not need to waste time accurately measuring the size of the area a rough estimate is fine.

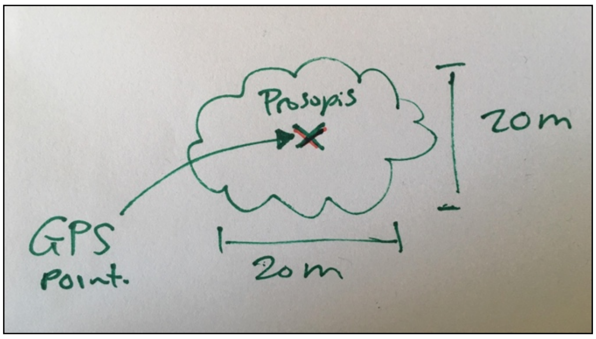


Figure 2: An illustration of a 20m by 20m plot of *Prosopis* and an indication of where to take the GPS point.

If the area you have found is larger than 20m by 20m, such as 40m by 40m, please take further GPS coordinates. But be sure to move at least 20 meters away from where you have taken previous GPS points (see figure 3).

For areas that are very big, or are very dense with Prosopis, you can instead of going into the middle of each 20m by 20m part of the large area, you can walk very closely around the edge while tracking your path with the GPS (see figure 3).

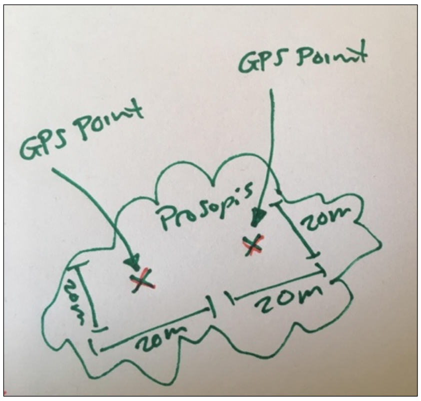


Figure 3: An illustration of a large patch of *Prosopis* with multiple 20m by 20m areas and an indication of where to take the GPS point.

In order to map the species between 50 and 150 GPS points for Prosopis are required. This data can be made up of individual data points from individual patches and those tracks from larger areas.

Every time you take a new GPS point or track please record the ID in the data collection sheet, along with the species name, data type (point or track).

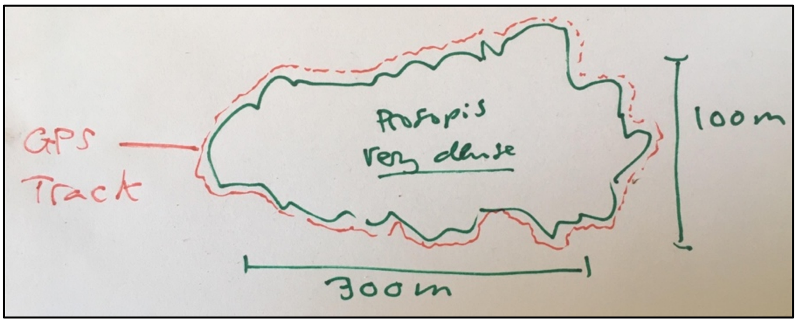


Figure 4: An illustration of a large dense patch of *Prosopis* with an indication of how to take the GPS track around the patch.

Acacia

For Acacia trees (Figure 5), do exactly the same as what is written for *Prosopis*.

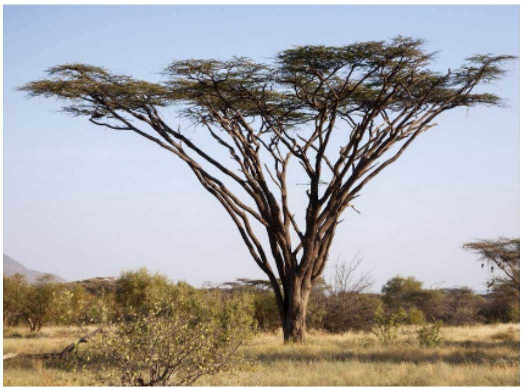


Figure 5: Photograph of an *Acacia* tree.

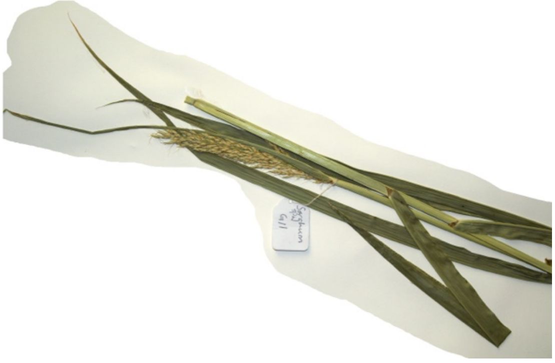


Figure 6: Photograph of *Sorghum bicolour* grass.

Sorghum bicolour

For *Sorghum bicolour* (Figure 6) grass species, do exactly the same as is written for *Prosopis*.

Cynodon plectostachyus

For *Cynodon plectostachyus* (Figure 7) grass species, do exactly the same as is written for *Prosopis*.

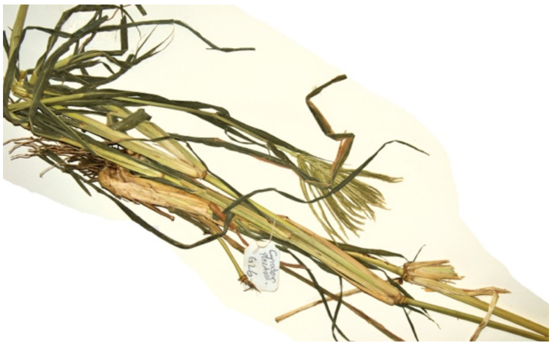


Figure 7: Photograph of *Cynodon plectostachyus*

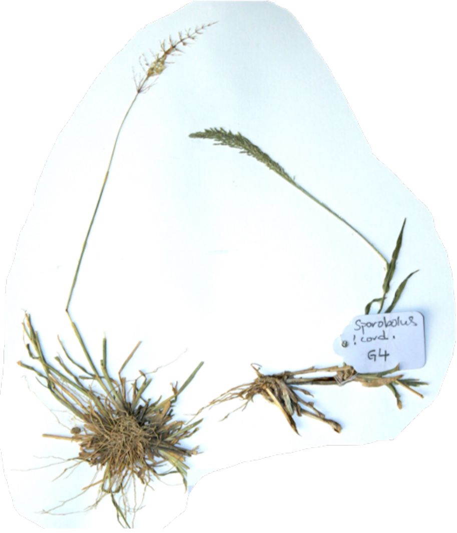


Figure 8: Photograph of *Sporobolous cordofanus* grass.

Sporobolous cordofanus

For *Sporobolous cordofanus* (Figure 8) grass species, do exactly the same as is written for *Prosopis*.

Control Data

Control data are different from the individual species data and should be much faster and easier to collect. For control data 150 GPS points or more of areas, that do not contain *Prospis spp*, *Acacia spp*, *Cynodon plectostachyus* , *Sorghum bicolour* or *Sporobolous cordofaunus* are needed. Instead of containing the focal species, control point areas can be mixed vegetation types, mixed trees, mixed shrubs, mixed herbs, mixed grasses, bare ground, rock, or any combination of these. The GPS points for control data need to be a mixture of these types and the type of control data being collected entered into the data sheet.

Please find data entry sheet in Appendix A.

End of Protocol

7 Coordinate Extraction from Google Earth Imagery.

Data for the fig tree species *Ficus sur* (Figure 9) were collected via Google Earth imagery (GEI). Only points with a high confidence of fig tree presence were collected.

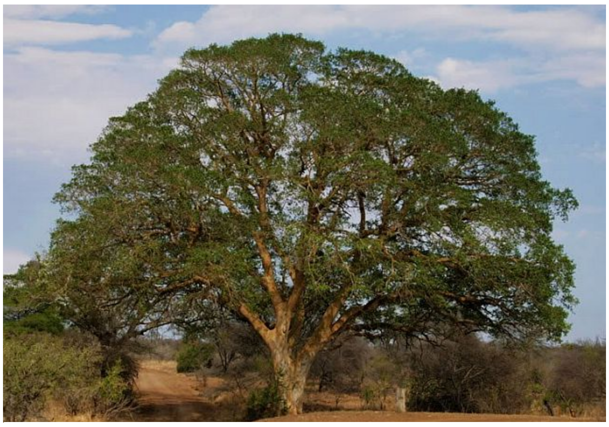


Figure 9: Photograph of *Ficus sur* tree.

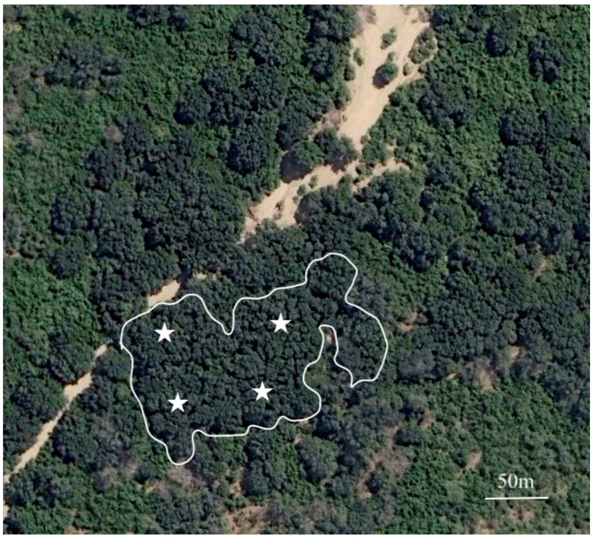


Figure 10: Google Earth imagery showing a dense patch of fig trees (Ficus sur). The polygon encloses a particularly large patch of fig trees. Stars indicate examples of locations where Ficus sur training data was collected.

The fig trees are clearly identifiable in comparison to other vegetation, which is an essential requirement for collecting reference data remotely by this method (Figure 10).

Point coordinates obtained remotely via GEI were exported as KML. The following Python function extracts coordinates from KML files containing, points or lines and writes a csv file with two columns one for latitude and another for longitude.

```
from osgeo import ogr
import pandas as pd

def get_coords(kml): # get all coordinates from kml.
    coords = [] # open list to save coordinates in
    for layer in kml:
        for feature in layer:
            geometry = feature.GetGeometryRef()
            if geometry != None:
                for i in range(0, geometry.GetPointCount()):
                    coords.append(geometry.GetPoint(i))
        df = pd.DataFrame(coords)
        return df.to_csv (r'output.csv', index = None, header=True)
# path to .kml file/
kml=ogr.Open('Dissertation/SORALO_data/control_data/Mixed_veg/Mixed_veg.kml')

get_coords(kml)
```

Columns containing the relevant data class and date of data collection information is then added to the reference data along with class information.

8 Format Ground Reference Data

Ground reference data were formatted and transformed to Sentinel-2 image coordinate reference system using the SP R package (Roger et al 2013). All data consisting of row ID, latitude, longitude, class and data of data collection where combined to a single csv file.

The reference data was formatted and transformed with the following R code.

```
# libraries
library(GISTools)
library(rgdal)
```

```
# import data.
df<-read.csv("All_Reference_Data.csv", stringsAsFactors = FALSE)

# structure of data
str(df)

# number of classes
levels(df$Class)

# number of samples in each class
Summary(df$Class)

# get coordinate columns
coords <- df[,c(3,4)]

# create spatial points object
spatpoint<- SpatialPoints(coords, proj4string=CRS("+proj=longlat"))

# transform CRS to UTM
transoformed_spatpoint<- spTransform(spatpoint, CRS("+proj=utm +zone=36 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0"))

# get new coords
coords_df_T<-as.data.frame(transoformed_spatpoint @coords)

# add to data frame
df$Easting <-  coords_df_T$lon
df$Northing <-  coords_df_T$lat

# write new csv file with transformed coords
write.csv(df, "Ground_Ref_Data_Transformed.csv")
```

9 Sentinel-2 Image Selection

Selection of images with acceptable levels of cloud cover over the study site (<10%) was accomplished by viewing the full extent and resolution of an S1 image in the Google Earth Engine code editor.

Selecting images for this project was conducted using the JavaScript code editor from Google Earth Engine (GEE). This editor allows full extent and resolution viewing of Sentinel-2 images. As a result, assessing cloud cover of the study area is more effective in GEE than quick view of images via the Sentinel-2 data repository found.

GEE is a powerful open access cloud computing service for remote sensing image analyses (https://earthengine.google.com/). The following process outlines how images for selected area and date range can be filtered by cloud coverage and viewed prior to download. The image obtained within the date range with least cloud coverage is displayed.

The following steps were followed to identify the most useful images over the 2018 growing season.

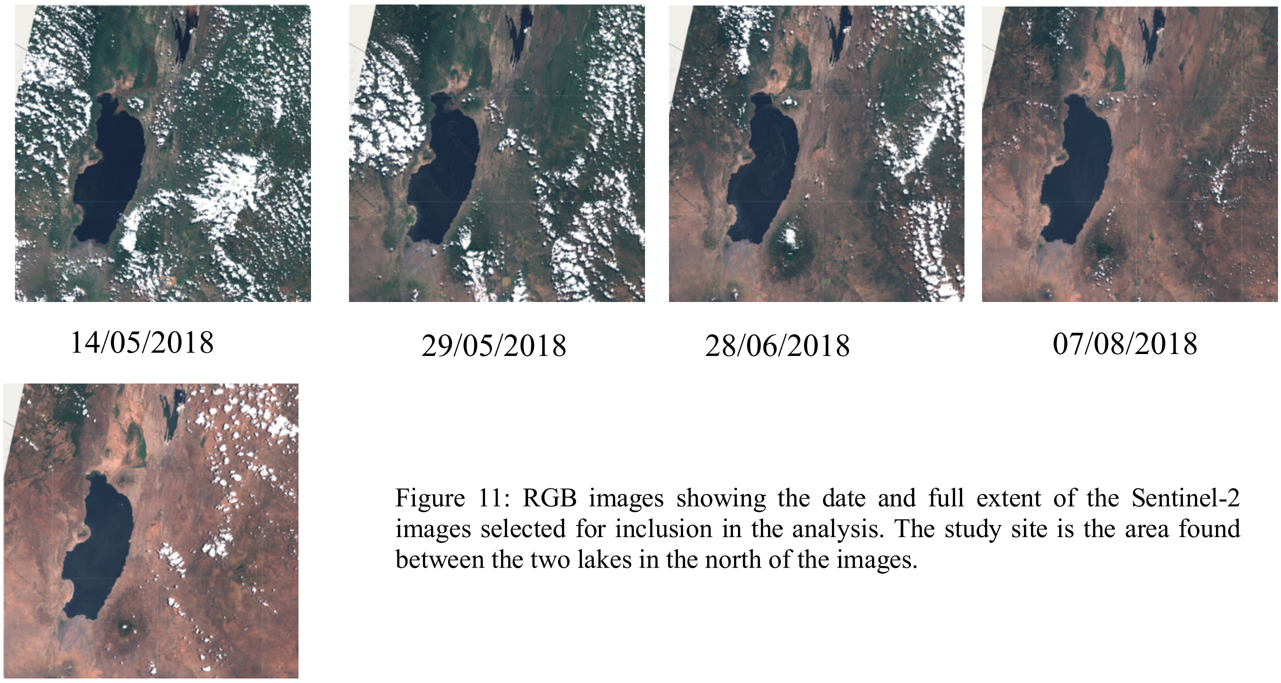
- The polygon tool in the code editor was used to draw an area around the ROI. This imported a variable named geometry to the code editor.
- Searched sentinel 2 in the search bar and selected import sentinel 2 MSI: Multi-Spectral Instrument, level 1C (note 1C means not atmospherically corrected).
- Code below was entered into the code editor and date range of interest modified to suit dates of interest (NB If no image is returned the date range may be too constrained). The code overlays the least cloudy image obtained within the date range specified.

```
//Function used to filter images by cloud percentage.
var image = ee.Image(imageCollection
.filterDate("2018-01-01", "2018-12-30")
.filterBounds(geometry)
.sort("CLOUD_COVERAGE_ASSESSMENT") // displays least cloudy image first.
.first());
print("A Sentinel-2 scene:", image);

var trueColour = {
  bands:["B4", "B3", "B2"],
  min:0,
  max:3000
};
Map.addLayer(image, trueColour, "True Colour Image")
```

The images in figure 5 were selected because they contained low cloud coverage over the study area and because the time interval between image acquisition was considered large enough for each image to contain different spectral information.

The image acquired on the 14/05/2018, shown in figure 11, is visibly greener than all other images indicating that vegetation biomass is greater in this image compared to all others. This was used in all single image classifications because this image represents the image acquired closest to peak biomass, and images used at peak biomass have previously been shown to produce greatest classification accuracy in single image classification models (Feilhauer et al., 2013: Rapinel et al., 2019: Shoko and Mutunga 2017).



14/05/2018 29/05/2018 28/06/2018 07/08/2018

01/10/2018

Figure 11: RGB images showing the date and full extent of the Sentinel-2 images selected for inclusion in the analysis. The study site is the area found between the two lakes in the north of the images.

10 Image Download

Using the date of image acquisition, images were downloaded with the SentinelSat Python package. A function was written to automate image downloading. The function require a .geojson file (which was created and downloaded from this site - http://geojson.io/#map=2/20.0/0.0) of the study area, image acquisition date and the tile id. Using tile id restricts downloading of multiple scenes of the same region in the same day. Products are downloaded to the current directory.

```
from sentinelsat import SentinelAPI, read_geojson, geojson_to_wkt
import os
import subprocess
import geopandas as gpd
from datetime import date
import pandas as pd
from shapely import wkt

# Path to geojson
AOI_PATH =
'/Users/freddiehunter/Work/GIS_Course_Work/Dissertation/Image_Processing/CMD_Download/AOI_BOX.geojson'

# Date of interest
Day = ('20180514', date(2018, 5, 24)) # format date as such.

# Tile ID of interest
Tile_ID = 'T36MZC'

def get_product(AOI_PATH, Day, Tile_ID):
    api = SentinelAPI(username, 'password','https://scihub.copernicus.eu/dhus')
    footprint = geojson_to_wkt(read_geojson(AOI_PATH))
    products = api.query(footprint,
                         date = Day,
                         platformname='Sentinel-2',
                         area_relation = 'Contains')
    df = pd.DataFrame.from_dict(products).T
    product_df = gpd.GeoDataFrame(df, geometry = df['footprint'].apply(wkt.loads))
    product_df["Product_ID"] = product_df.index
    tilename = product_df['filename']
    tile_id = str(tilename).split('_')
    tile_id = tile_id[5::6]
    product_df['Tile_ID'] = Tile_ID

    return product_df

# Run function
df = get_product(AOI_PATH, Day, Tile_ID)

# Download product
api.download_all(df['Product_ID'])
```

11 Sen2Cor Processing

Sen2Cor is an algorithm for converting Sentinel-2 images from level one products, to level two products. The algorithm corrects for atmospheric effects, such that reflectance values are in theory the same as those that would be obtained at the Earth surface.

In this study the stand alone installer version 02.05.05 was used, and can be installed with the following link (http://step.esa.int/main/third-party-plugins-2/sen2cor/: Main-Knorn et al., 2017). After Sen2Cor instillation a terminal command can be used to run the algorithm and parameters are available to select the spatial resolution of the output images.

Within terminal set the directory to Sen2Cor-02.05.05 and run following code with a path to the .safe level one directory.

```
bin/L2A_Process --resolution 20
/Users/freddiehunter/Work/GIS_Course_Work/Dissertation/Image_Processing/Sent2_data/Data/1C/S2A_MSIL1C_20180524T073731_N0206_R092_T36MZC_20180524T113104.SAFE
```

12 Cloud Masking

There are several cloud masks available for Sentinel-2 imagery. The Fmask developed by Zhu and Woodcock (2012) is considered the most accurate for Sentinel 2 (Frantz et al 2018). Installing and implementing the Fmask with the command line on a macOS can be achieved using the following commands in terminal. The sequence requires the prior installation of Python via Anaconda or Conda min, GDAL, RIOS, Numpy and Scipy.

Setting up an environment that contains all required libraries.

- conda create -n senEnv rois
- source activate senEnv
- conda –add channels conda-forage
- conda install python-fmask

The environment is activated with

- Source activate senEnv.

Running the Fmask.

Set directory to where level1 products are stored. The mask requires a Sentinel-2 level1.safe directory.

- fmask_sentinel2Stacked.py -o utput_name.img --safedir --pixsize 20 --cloudbufferdistance 150 --shadowbufferdistance 150 sentinel1c_directory.safe

Cloud and cloud shadow buffers are set to 150 pixels and pixel size to 20.

The resulting raster contains classes, cloud, cloud shadow, water and clear sky pixel. To reclassify and write a geo referenced raster to produce a binary raster mask the following python function was used.

```
import numpy as np
import rasterio as rio

# where cloud.img is your fmask
path = '../FMASK_Processing/FMASK_Python_reclass/cloud.img'

def reclassFmask(path):
    raster = rio.open(path)
    naip_meta = raster.profile
    band1 = raster.read(1)
    band1 = band1.astype(np.uint8)
    # collect index arrays
    one_or_less = band1_large <= 1
    from_2_to_5 = (1 < band1_large) & (band1_large <= 5)
    greater_6 = band1_large >= 6
    # now modify target array
    band1_large[one_or_less] = 10
    band1_large[from_2_to_5] = 20 # values classified as 20 are to be masked
    band1_large[greater_6] = 10

    with rio.open('fmask_reclass.tif', 'w', **naip_meta) as dst:
        dst.write(band1, 1)

reclassFmask(path)
```

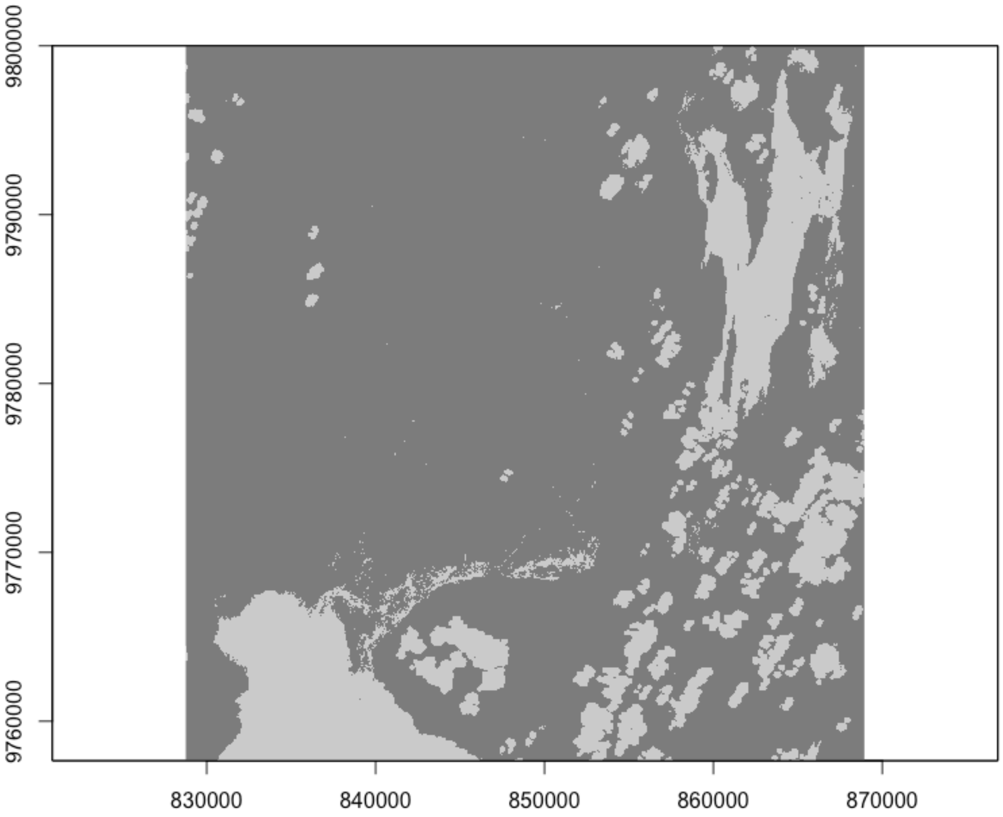


Figure 12: An example of a reclassified Fmask raster as a cloud and water binary mask for the image acquired on 14/05/2018. Cloud and water are shown in light grey.

13 Image Cropping and Image Set Creation

The following R code imports the Sen2Cor corrected image bands, creates a stack of those bands, crops to the area extent of the shapefile and masks the stack according to the reclassified Fmask raster. Combinations of image stacks of the same date are then stacked together to form time series image sets.

```
# required libraries
library(raster)
library(rgdal)

# import area of interest shapefile
extent<-readOGR("Raster_Crop_Extent")
crs(extent)

# transform projection of shapefile to that of satellite image.
Extent <- spTransform(extent, CRS("+proj=utm +zone=36 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0"))
plot(extent, axes=T)

# create bounding box of AOI
extent_box<-bbox(extent)
```

- The following section of code is representative of the process applied to each of the five images.

```
# import bands
# May 14 2018
b02 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B02_20m.tif”)
b03 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B03_20m.tif”)
b04 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B04_20m.tif”)
b05 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B05_20m.tif”)
b06 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B06_20m.tif”)
b07 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B07_20m.tif”)
b8A <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B8A_20m.tif”)
b11 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B11_20m.tif”)
b12 <- raster(“L2A_2018_14_05/T36MZC_20180514T073611_B12_20m.tif”)

# create stack of bands
stack_may14<-stack(b02,b03,b04,b05,b06,b07,b8A,b11,b12)
#check crs
crs(stack_may14)

# crop according to bounding box.
Crop_stack_may14_mask<-crop(stack_may14, extent_box)

# import cloud fmask.
Mask_may_14 <- raster(“L2A_2018_14_05/reclass_14_05_2018_FMASK.tif”)

# crop fmask to same extent as img
crop_mask_may_14_<-crop(mask_may_14, extent_box)

# mask values that = 20. This is the value assigned to pixels that need to be masked.
Crop_mask_may_14_[crop_mask_may_14_ == 20] <- NA

# mask stack by cloud mask
stack_may_14 <- mask(crop_stack_may14_mask, mask = crop_mask_may_14_)

#remove unnecessary objects from R environment
rm(b02,b03,b04,b05,b06,b07,b8A,b11,b12,stack_may14, crop_mask_may_14_, mask_may_14)
```

- For brevity only code for a single image is presented. The same process was applied to all five images.

```
# images to include in model – Wet season single image
datasetA <- stack_may_14

# for data included in model – Wet season time series
datasetB <- stack(stack_may_14, stack_may_29)

# for data included in model – Wet season time series
datasetC <- stack(stack_may_14, stack_may_29, stack_june_28, stack_august_07, stack_october_01)
```

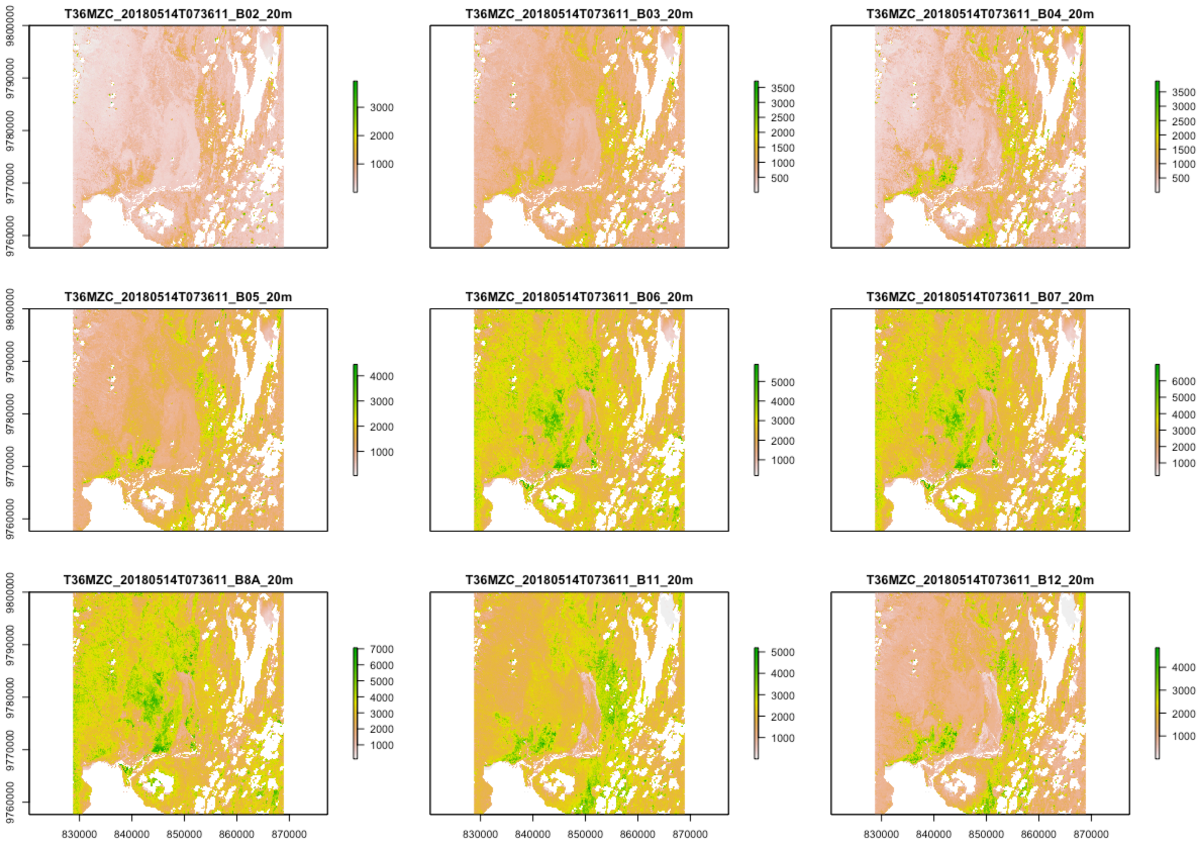

T36MZC_20180514T073611_B02_20m
T36MZC_20180514T073611_B03_20m
T36MZC_20180514T073611_B04_20m
T36MZC_20180514T073611_B05_20m
T36MZC_20180514T073611_B06_20m
T36MZC_20180514T073611_B07_20m
T36MZC_20180514T073611_B8A_20m
T36MZC_20180514T073611_B11_20m
T36MZC_20180514T073611_B12_20m
9760000
9770000
9780000
9790000
9800000
830000
840000
850000
860000
870000

Figure 13: An example of the cropped and masked images for all nine utilised bands, in this case for the image acquired on 14/05/2019.

14 Classification

14.1 Formatting Images and Reference Data

The R code presented below outlines the implementation of classifications (Random Forest and Support Vector Machine (SVM) with both linear and radial kernel), collection of statistics, pixel counts, confidences maps and the production of figures.

Code was obtained and modified from the following sources.
http://amsantac.co/blog/en/2016/10/22/model-stacking-classification-r.html
https://machinelearningmastery.com/tuning-machine-learning-models-using-the-caret-r-package/
https://dataaspirant.com/2017/01/19/support-vector-machine-classifier-implementation-r-caret-package/
https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/

The example code implements RF for the wet season single image dataset. Where svmLinear and svmRadial implementation differs from RF the alternative code is also supplied.

- Libraries required.

```
library(ggplot2)
library(caret)
library(raster)
library(snow)
library(rgdal)
library(plyr)
```

- Reading of formatted and transformed ground reference data.

```
# import ground ref data ----
df<-read.csv(“Ground_Ref_Data_Transformed.csv”)

# check data structure
str(df)

# check levels and sample sizes of classes
summary(df$Class)

# turn ref data into spatial pointsdataframe
# get coordinates
coords <-df[,c(2,3)]

# format as spatial dataframepoints
ref_data<-SpatialPointsDataFrame(coords, proj4string=CRS(“+proj=utm +zone=36 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0”), df)
```

- Importing cropped and masked images.

```
# Import image set
datasetA <- stack(stack_may_14)
crs(datasetA)
```

- For all wet season time series models the following code was used

```
datasetB <- stack(stack_may_14, stack_may_29)
```

- For all multi-season time series models the following code was used.

```
datasetC <- stack(stack_may_14,stack_may_29,stack_june_28, stack_august_07, stack_october_01)

# set bands names
names(datasetA) <- c(“B2”,”B3”,”B4”,”B5”,”B6”,”B7”,”B8A”,”B11”,”B12”)
```

- Extract pixel values from all bands according to coordinates in ground reference data.

```
# Extract training data values from the image bands
ref_extracted <- as.data.frame(raster::extract(datasetA, ref_data))
ref_extracted$class<-train$Class

# ensure complete cases (i.e. no broken pixels with NA values)
```

```
ref_extracted <- train_extracted[complete.cases(ref_extracted), ] # Store the complete cases subset in a new data frame
```

- Split pixel value data in to training and test data.

```
# Split data into training and testing data
Ref_split <- createDataPartition(y = ref_extracted$class, p= 0.7, list = FALSE)
training_data <- ref_extracted[Ref_split,]
testing_data <- ref_extracted[-Ref_split,]

# check dimensions/size of training and testing data
dim(training_data); dim(testing_data);
```

14.2 Hyperparameter Optimisation

- Optimising parameters for RF.

Table 2: Values included in the manual grid of hyperparameter values for RF models. * Mtry = the number of randomly selected predictors available at each decision tree.

Image Set	**No Tree**	**Mtry**
Wet Season Single Image	500	1 - 9
Wet Season Time Series	500	1 - 18
Multi-Season Time Series	500	1 - 45

```
# prepare training scheme
#Repeated k-fold Cross Validation – 10-fold cross validation with 3 repeats
control <- trainControl(method=”repeatedcv”, number=10, repeats=3)

# create tuning grid for parameter mtry
# get number of bands/predicors in the model.
noPredictors<- -1 + length(training_data)

# number of trees was held constant at 500.
Grid_rf <- expand.grid(mtry=c(1:noPredictors), ntree = 500))

# run model
rf_wet_season_single_image_Grid <- train(class ~., data = training_data, method =”rf”, preProcess = c(“center”, “scale”), trControl = control, tuneGrid = grid_rf, tuneLength = 1)
```

- Optimising parameters for SVM linear.

Table 3: Values included in the manual grid of hyperparameter values for SVM linear kernel models.

Image Set	**Cost**
Wet Season Single Image	0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25
Wet Season Time Series	0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25
Multi-Season Time Series	0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25

```
Grid_Linear <- expand.grid(C = c(0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25))

svmLinear_wet_season_single_image_Grid <- train(class ~., data = training_data, method =”svmLinear”, preProcess = c(“center”, “scale”), trControl = control, tuneGrid = grid_Linear)
```

- Optimising parameters for SVM radial.

Table 4: Values included in the manual grid of hyperparameter values for SVM radial kernel models. * Sigma = Reach of single training instance

Image Set	**Cost**	**Sigma**
Wet Season Single Image	0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25	0, 0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.25, 0.5, 0.75, 0.9
Wet Season Time Series	0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25	0, 0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.25, 0.5, 0.75, 0.9
Multi-Season Time Series	0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25	0, 0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.25, 0.5, 0.75, 0.9

```
Grid_Radial <- expand.grid(sigma = c(0, 0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.25, 0.5, 0.75, 0.9),C = c(0, 0.5, 1, 2, 5, 7, 10, 15, 20, 25))

svmRadial_wet_season_single_image_Grid <- train(class ~., data = training_data, method =”svmRadial”, preProcess = c(“center”, “scale”), trControl = control, tuneGrid = grid_Radial)
```

- Get Model Results.

```
print(rf_wet_season_single_image_Grid)
```

- Create plot showing relationship between accuracy and hyperparameter value.

```
pdf(“RF_Wet_Season_Single_Image_mtry_tuning.pdf”)
plot(rf_wet_season_single_image_Grid, cex=2, pch=16, lab.cex=4, cex.lab=3, cex.axis=3)
dev.off()
```

- Find and set hyperparameter values.

```
# set optimal mtry value
metry_value = rf_wet_season_single_image_Grid$bestTune$mtry
```

- For svmLinear find and set hyperparameter values.

```
# set optimal C value
C_value = svmLinear_wet_season_single_image_Grid$bestTune$C
```

- For svmLinear find and set hyperparameter values.

```
# set optimal C value
C_value = svmRadial_wet_season_single_image_Grid$bestTune$C
# set optimal Sigma value
Sigma_value = svmRadial_wet_season_single_image_Grid$bestTune$Sigma
```

- The following code sets up empty data frames and rasters with which to fill the results of running 100 classification replicates.

```
#data frame for overall accuracy and kappa values
OA_mean_rf <- data.frame()

# data frame for confusion matrix values
df_CM_rf_ <- data.frame()

# data frame for all class specific statistics
df_byClass_rf_wet_season_single_image<-data.frame()

# prepare blank raster for classification prediction
prediction_raster<-datasetA$B2
prediction_raster[prediction_raster > 0] <- 0

# import shapefiles for pixel counts
wet_season <- readOGR('Wet_Season_Area_Only/')
plot(wet_season)
crs(wet_season)

# transform shapefile crs
wet_season<-spTransform(wet_season, CRS("+proj=utm +zone=36 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 + towgs84=0,0,0"))
crs(wet_season)

dry_season <- readOGR('Conservation_And_Buffer_Only_Merged/')
plot(dry_season)
crs(dry_season)

# transform shapefile crs
dry_season<-spTransform(dry_season, CRS("+proj=utm +zone=36 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 + towgs84=0,0,0"))
crs(dry_season)

# dataset for storing count values
Pixel_Counts <- data.frame()

# collect variable importance data
var_imp <- data.frame()
```

14.3 Classification Loop

- Classification loop performed 100 times. Results from each loop are stored in the corresponding data frames created above.

```
# set seed value to force randomisation per loop run.
Seed = 100

# start of loop ----
for(I in 1:100){

  # set seed
  seed<-1+seed # seed number increased by 1 with each loop repeat.
```

```
Set.seed(seed) # set the seed.

# Randomly split data into training and testing data
intrain <- createDataPartition(y = train_extracted$class, p= 0.7, list = FALSE)
training_data <- train_extracted[intrain,]
testing_data <- train_extracted[-intrain,]
```

- Set hyperparameters as optimal values found in training and validation of model.

```
Grid_rf <- expand.grid(mtry = c(metry_value))
```

- For svmLinear.

```
Grid_svmLinear <- expand.grid(C = c(C_value))
```

- For svmRadial.

```
Grid_svmRadial <- expand.grid(C = c(C_value), Sigma = c(Sigma_Value))
```

- Training models.

```
# train model with tune length  = 1 (forces model fit once)
rf_wet_season_single_image <- train(class ~., data = training_data, method =”rf”, preProcess = c(“center”,
“scale”),tuneGrid = grid_rf, tuneLength = 1, ntree = 500)
```

- Get Overall Accuracy and Kappa accuracies.

```
OA_results<-as.data.frame(rf_wet_season_single_image_Grid_multi$results)
OA_mean_rf<-rbind(OA_mean_rf, OA_results)
```

- Get classification predictions.

```
# get predictions
test_pred_grid <- predict(rf_wet_season_single_image, newdata =     testing_data)
```

- Confusion matrix results.

```
# get confusion matrix values
cm_rf_ <- confusionMatrix(test_pred_grid, testing_data$class)
cm_data_fame_rf_<- as.data.frame(cm_rf_$table)

# append confusion matrix data to data frame
df_CM_rf_<-rbind(df_CM_rf_, cm_data_fame_rf_)
```

- Predict classification map.

```
# get classification map
beginCluster() # parallelise the job
loop_predictions <- clusterR (datasetA, raster::predict, args = list(model = rf_wet_season_single_image))
endCluster()
```

```
  # add classification prediction map as layer to stack of rasters. One layer for every run of the loop.
  Prediction_raster<-stack(loop_predictions,, prediction_raster)
```

- Count number of cells in grazing areas for each class for each run of the loop and add to data frame.

```
  # method shown only for acacia for brevity.
  Acacia <- loop_predictions

  # acacia cells
  acacia[acacia != 1] <- NA # set all values not of focal class to NA.

  # count only dry season.
  # remove all pixels outside of dry season area
  masked_class<-mask(acacia,dry_season)

  # count number of pixels.
  Dry_masked_acacia<-as.data.frame(cellStats(masked_class, sum))
  Pixel_Counts<-rbind(Pixel_Counts,dry_masked_acacia)

  # remove all pixels outside of wet season area
  masked_class<-mask(acacia,wet_season) )

  # count number of pixels
  wet_masked_acacia<-as.data.frame(cellStats(masked_class, sum))
  Pixel_Counts<-rbind(Pixel_Counts,wet_masked_acacia)

 # end of loop
}
```

14.4 Formatting OA and Kappa Accuracy Statistics

```
# get mean and standard deviation of OA
mean(OA_mean_rf$Accuracy)
sd(OA_mean_rf$Accuracy)

# get mean and standard deviation of Kappa
mean(OA_mean_rf$Kappa)
sd(OA_mean_rf$Kappa)

# add variables to data set
# add model type variable
OA_mean_rf$Model<-c('RF')
# add image set variable
OA_mean_rf$DataSet<-c('Wet Season Single Image')

# write dataframe
write.csv(OA_mean_rf, "RF_Wet_Season_Single_Image_OA_Kappa.csv")
```

- Plotting OA and Kappa accuracy statistics.

```
#import data
OA_Ka<-read.csv("RF_Wet_Season_Single_Image_OA_Kappa.csv")
names(OA_Ka)
OA_Ka<-OA_Ka[,c(3:8)]
```

```
OA_Kb<-read.csv("SVMLinear_Wet_Season_Single_Image_OA_Kappa.csv")
names(OA_Kb)
OA_Kb<-OA_Kb[,c(3:8)]
OA_Kc<-read.csv("SVMRadial_Wet_Season_Single_Image_OA_Kappa.csv")
names(OA_Kc)
OA_Kc<-OA_Kc[,c(4:9)]
OA_Kd<-read.csv("RF_Wet_Season_Time_Series_OA_Kappa.csv")
names(OA_Kd)
OA_Kd<-OA_Kd[,c(3:8)]
OA_Ke<-read.csv("SVMLinear_Wet_Season_Time_Series_OA_Kappa.csv")
names(OA_Ke)
OA_Ke<-OA_Ke[,c(3:8)]
OA_Kf<-read.csv("SVMRadial_Wet_Season_Time_Series_OA_Kappa.csv")
names(OA_Kf)
OA_Kf<-OA_Kf[,c(4:9)]
OA_Kg<-read.csv("RF_Multiple_season_time_series_OA_Kappa.csv")
names(OA_Kg)
OA_Kg<-OA_Kg[,c(3:8)]
OA_Kh<-read.csv("svmLinear_Multiple_season_time_series_OA_Kappa.csv")
names(OA_Kh)
OA_Kh<-OA_Kh[,c(3:8)]
OA_Ki<-read.csv("SVMRadial_Multiple_season_time_series_OA_Kappa.csv")
names(OA_Ki)
OA_Ki<-OA_Ki[,c(4:9)]

# combine all data sets

OK_K_Complete<-rbind(OA_Ka,OA_Kb,OA_Kc,OA_Kd,OA_Ke,OA_Kf,OA_Kg,OA_Kh,OA_Ki)
data_long <- gather(OK_K_Complete, Accuracy_Measure, Value, Accuracy:Kappa, factor_key=TRUE)

#format combined dataset

levels(data_long$Accuracy_Measure)[levels(data_long$Accuracy_Measure)=="Accuracy"] <- "Overall Accuracy"
levels(data_long$Accuracy_Measure)
data_long$Accuracy_Measure<-as.character(data_long$Accuracy_Measure)
data_long[data_long$Accuracy_Measure == "Accuracy"] <- "Overall Accuracy"

# plot as boxplots

ggplot(data=data_long, aes(x=Accuracy_Measure, y=Value)) + geom_boxplot(stat="boxplot", alpha=1, fill="grey80") + guides(fill=FALSE) + theme_bw() + ylab("Accuracy") + xlab("Accuracy Measure") + theme(axis.text.x = element_text(size=12), axis.text.y = element_text(angle = 0, hjust = 1, size=12)) + theme(text = element_text(size=15)) + facet_wrap(data_long$DataSet~data_long$Model)
```

14.5 Confusion Matrix Production

```
# format confusion data matrix
names(df_CM_rf_)
unique(df_CM_rf_[,c(1,2)])

dt_rf_ <- as.data.table(df_CM_rf_)
mean_cm_rf_<-dt_rf_[, mean(Freq), by = list(dt_rf_$Prediction, dt_rf_$Reference)]
mean_cm_rf_<-as.data.frame(mean_cm_rf_)
```

```
# plot normal confusion matrix
ggplot(data = mean_cm_rf_, mapping = aes(x = mean_cm_rf_$dt_rf_, y = mean_cm_rf_$dt_rf_.1)) +
  geom_tile(aes(fill = mean_cm_rf_$V1), colour = “white”) +
  geom_text(aes(label = 24radie(“%1.0f”, mean_cm_rf_$V1)), vjust = 0.5) +
  scale_fill_gradient(low = “white”, high = “red”) +
  theme_bw() +
  theme(legend.position = “none”) +
  xlab(“”) +
  ylab(“”) +
  theme(axis.text.x = element_text(colour=”black”,size=15),axis.title.x= element_text(colour=”black”,size=15),
axis.text.y = element_text(colour=”black”,size=15), axis.title.y= element_text(colour=”black”,size=15)) +
theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_discrete(limits = rev(levels(mean_cm_rf_$dt_rf_.1))) + coord_equal()

# save plot
ggsave(“rf_wet_season_single_image_con_mat.pdf”)
```

- Normalising confusion matrices

```
# normalise confusion matrix

# get sample size of each class in testing data
number_testing_samples<-as.data.frame(summary(testing_data$class))

# splot data frame by class
split_mean_cm_rf_<-split(mean_cm_rf_, mean_cm_rf_$dt_rf_.1)

# set values as percentage
split_mean_cm_rf_$Acacia$V1<-split_mean_cm_rf_$Acacia$V1/22*100
split_mean_cm_rf_$`Cynodon plectostachyus`$V1<-split_mean_cm_rf_$`Cynodon plectostachyus`$V1/21*100
split_mean_cm_rf_$`Ficus sur`$V1<-split_mean_cm_rf_$`Ficus sur`$V1/16*100
split_mean_cm_rf_$`General Control`$V1<-split_mean_cm_rf_$`General Control`$V1/18*100
split_mean_cm_rf_$`Grass Control`$V1<-split_mean_cm_rf_$`Grass Control`$V1/22*100
split_mean_cm_rf_$`Mixed Vegetation Control`$V1<-split_mean_cm_rf_$`Mixed Vegetation Control`$V1/22*100
split_mean_cm_rf_$Prosopis$V1<-split_mean_cm_rf_$Prosopis$V1/12*100
split_mean_cm_rf_$`Sorghum bicolor`$V1<-split_mean_cm_rf_$`Sorghum bicolor`$V1/6*100
split_mean_cm_rf_$`Sporobolus cordofanus`$V1<-split_mean_cm_rf_$`Sporobolus cordofanus`$V1/22*100

# reassemble dataframe
balanced_mean_cm_rf_<-unsplit(split_mean_cm_rf_, mean_cm_rf_$dt_rf_.1, drop = FALSE)

# plot normalised confusion matrix
ggplot(data = balanced_mean_cm_rf_, mapping = aes(x = balanced_mean_cm_rf_$dt_rf_, y =
balanced_mean_cm_rf_$dt_rf_.1)) +
  geom_tile(aes(fill = balanced_mean_cm_rf_$V1), colour = “white”) +
  geom_text(aes(label = 24radie(“%1.0f”, balanced_mean_cm_rf_$V1)), vjust = 0.5) +
  scale_fill_gradient(low = “white”, high = “red”) +
  theme_bw() +  theme(legend.position = “none”) +
  xlab(“”) + ylab(“”) +
  theme(axis.text.x = element_text(colour=”black”,size=15),axis.title.x= element_text(colour=”black”,size=15),
axis.text.y = element_text(colour=”black”,size=15), axis.title.y= element_text(colour=”black”,size=15)) +
theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_discrete(limits = rev(levels(balanced_mean_cm_rf_$dt_rf_.1))) + coord_equal()

ggsave(“rf_wet_season_single_image_normalise_con_mat.pdf”)
```

14.6 Class Proportions Across Grazing Areas

- Pixel counts data formatting and plotting

```
# format pixel count dataframe
# add columns to pixel counts
Pixel_Counts$Species<-rep(c("Acacia", "Acacia", "Cynodon plectostachyus", "Cynodon plectostachyus","Ficus
sur", 'Ficus sur', "General Control", "General Control", "Grass Control" , "Grass Control", "Mixed Vegetation", "Mixed
Vegetation", "Prosopis", "Prosopis", "Sorghum bicolor", "Sorghum bicolor" ,'Sporobolus cordofanus' ,'Sporobolus
cordofanus'), times = 100)
Pixel_Counts$Grazing_Area<-rep(c("Grass Bank", "Livestock Zone"), times = 100)

# reformat counts
Pixel_Counts$Counts<-Pixel_Counts$`cellStats(masked_class, sum)`

# get mean and standard deviation of pixel counts.
Mean_SD_Pixel_Counts<-
ddply(Pixel_Counts,~Species+Grazing_Area,summarise,mean=mean(Counts),sd=sd(Counts))

# add model information variables to data set
Mean_SD_Pixel_Counts$Model<-c("RF Wet Season Single Image")
Mean_SD_Pixel_Counts$Dataset<-c("Wet Season Single Image")

# write dataset
write.csv(Mean_SD_Pixel_Counts, "Wet_Season_Single_Image_RF_Pixel_Counts.csv")

# create bar chart of pixel counts as percentage coverage of each area.

# subset data for each of the grazing areas
grass_bank <- Pixel_Count_a[Pixel_Count$Grazing_Area=="Grass Bank",]
livestock_zone <- Pixel_Count[Pixel_Count$Grazing_Area=="Livestock Zone",]

# get total number of pixels in grass bank
grass_bank<-sum(grass_bank$mean)

# get total number of pixels in livestock_zone
livestock_zone<-sum(livestock_zone$mean)
# get mean number of pixels of each class in each grazing area
Pixel_Count_a$Percentage <- (Pixel_Count_a$mean/(sum(Pixel_Count_a$mean)))*100

# get mean value of each class pixel count as percentage coverage of grazing areas.
Pixel_Count_a$Graz_Percentage <- (Pixel_Count_a$mean/Pixel_Count_a$grazing_size)*100

# plot bar graph of pixel counts as percentage with standard deviation
ggplot(data=Mean_SD_Pixel_Counts, aes(x=Mean_SD_Pixel_Counts$Species , y=mean)) +
 geom_bar(colour="black",stat="identity", alpha=1,
fill=c("grey55","olivedrab4","seagreen4","grey80","khaki","navajowhite1", 'tomato4', 'orange3',
'yellowgreen',"grey55","olivedrab4","seagreen4","grey80","khaki","navajowhite1", 'tomato4', 'orange3',
'yellowgreen')) +
 guides(fill=FALSE) + theme_bw() +
 ylab("Pixel Counts") + xlab("Species") +
 theme(axis.text.x = element_text(size=20), axis.text.y = element_text(angle = 0, hjust = 1, size=20)) + theme(text =
element_text(size=20)) +
geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd), width=.2, position=position_dodge(.9)) +
facet_wrap(~Grazing_Area+Model) +
 theme(axis.text.x = element_text(angle = 55, hjust = 1))

# save plot
ggsave(“RF_Wet_Season_Single_Image_Pixel_Counts.pdf”)
```

- .

14.7 Classification Maps

- Production of classification map

```
# take most frequent value of each cell in stack of classification predictions and create raster containing those values only.

# function for collecting most frequent value of each pixel in each layer of prediction stack of rasters.
Mode <- function(x) {
  ux <- unique(x)
  ux=ux[!is.na(ux)]
  ux[which.max(tabulate(match(x, ux)))]
}

# apply function to classification prediction stack of rasters
rf_wet_season_single_image_prediction=calc(prediction_raster, fun=Mode)

# write raster of predicted values.
writeRaster(rf_wet_season_single_image_prediction, “RF_Wet_Season_Single_Image_Raster.tif”, overwrite=TRUE)


# shape file for plotting
GA_Boundaries_sf <- readOGR(‘Only_conservation_merge_and_dry_season’)

Check crs of shp file
crs(GA_Boundaries_sf)

# check crs of raster
crs(rf_wet_season_single_image_prediction)

# crop and mask raster to shapefile (bounding box only)
26radien_rf_single_image_preds <- crop(rf_wet_season_single_image_prediction, GA_Boundaries_sf)
#mask raster to shape file
crop_mask_rf_single_image_preds <- mask(26radien_rf_single_image_preds, GA_Boundaries_sf)
#convert raster to dataframe
crop_mask_rf_single_image_preds <- as.data.frame(rasterToPoints(crop_mask_rf_single_image_preds))

# change variable names and check data
names(crop_mask_rf_single_image_preds)[1:2] <- c(“Easting”, “Northing”)
head(crop_mask_rf_single_image_preds)
unique(crop_mask_rf_single_image_preds$layer)

# plot classification map.
ggplot(data=crop_mask_rf_single_image_preds) +
  geom_tile(data = crop_mask_rf_single_image_preds, alpha = 0.8, aes(x = Easting, y = Northing, fill = layer)) +
scale_fill_gradientn(colours=c(“grey55”,”olivedrab4”,”seagreen4”,”grey80”,”khaki”,”navajowhite1”, ‘tomato4’, ‘orange3’, ‘yellowgreen’), breaks = c(1,2,3,4,5,6,7,8)) +
  geom_polygon(data = GA_Boundaries_sf, aes(x = long, y = lat, group = group), colour = “black”, fill = NA) +
  labs(fill = “Species”, size=20) + theme_bw() +
  theme(axis.text.x = element_text(colour=”black”,size=20),axis.title.x= element_text(colour=”black”,size=20),
axis.text.y = element_text(colour=”black”,size=20), axis.title.y= element_text(colour=”black”,size=20)) +
  theme(panel.background = element_rect(fill = “grey95”,colour = “grey95”,size = 0.5, linetype = “solid”),
panel.grid.major = element_line(size = 0.5, linetype = ‘solid’,colour = “grey70”), panel.grid.minor = element_line(size
= 0.25, linetype = ‘solid',colour = "grey70")) +
  coord_equal()
```

15 Summary

The methodologies employed in the various components of this paper were in some cases lengthy and complex. Given the open source nature of the software and programming languages used in the study, it is hoped that this technical report will facilitate researchers to have confidence in conducting studies with supervised machine learning on remote sensing imagery (Maxwell et al., 2018).

The success of this paper was to some degree dependent on an NGO operating in a remote region of a foreign country. The ground reference data were collected over eight non-consecutive days by SORALO resource assessors in Southern Kenya. During data collection at least two people and one vehicle were required, one driver and one resource assessor. The volume of resources provided by SORALO were therefore substantial, however the process of data collection was itself simple. The success of this paper demonstrates, that with a simple ground reference data collection protocol, the resources to carry it out and the technological knowledge required to carry out the methodologies outlined above accurate, maps of important biophysical components of the Earth’s surface can be produced. These maps can be produced relatively quickly with the information contained in this report and can be used as an important tool for monitoring and informing environmental sustainability practices and research.

16 References

Bivand, R., Keitt, T., and Rowlingson, B., 2019. rgdal: Bindings for the 'Geospatial' Data Abstraction Library. R package version 1.4-3. https://CRAN.R-project.org/package=rgdal

Brunsdon, C and Chen, H., 2014. GISTools: Some further GIS capabilities for R. R package version 0.7-4. https://CRAN.R-project.org/package=GISTools

Feilhauer, H., Thonfeld, F., Faude, U., He, K.S., Rocchini, D. and Schmidtlein, S., 2013. Assessing floristic composition with multispectral sensors—A comparison based on monotemporal and multiseasonal field spectra. *International Journal of Applied Earth Observation and Geoinformation*, 21, pp.218-229.

Gillies, S., 2013. Rasterio: geospatial raster I/O for Python programmers}, url = https://github.com/mapbox/rasterio

Hijmans, J., (2019). raster: Geographic Data Analysis and Modeling. R package version 2.8-19. https://CRAN.R-project.org/package=raster

Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. *Computing in science and engineering*, 9(3), pp.90–95.

Kahle, D and Wickham, H., 2012. ggmap: Spatial Visualization with ggplot2. *The R Journal*, 5(1), 144-161. URL http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf

Konkol, M., Kray, C. and Pfeiffer, M., 2019. Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science*, 33(2), pp.408-429.

Kuhn, M., Wing, J. Weston, Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., Scrucca, L., Tang, Y., Candan C, and Hunt, T., 2019. caret: Classification and Regression Training. R package version 6.0-84.https://CRAN.R-project.org/package=caret

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

McKinney, W., 2010. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference. pp. 51–56.

Oliphant, T.E., 2006. A guide to NumPy, Trelgol Publishing USA.

Pebesma, E.J., R.S. Bivand, 2005. Classes and methods for spatial data in R. R News 5 (2), https://cran.r-project.org/doc/Rnews/.

Rapinel, S., Mony, C., Lecoq, L., Clément, B., Thomas, A. and Hubert-Moy, L., 2019. Evaluation of Sentinel-2 time-series for mapping floodplain grassland plant communities. *Remote Sensing of Environment*, 223, pp.115-129.

Shoko, C. and Mutanga, O., 2017. Examining the strength of the newly-launched Sentinel 2 MSI sensor in detecting and discriminating subtle differences between C3 and C4 grass species. *ISPRS journal of photogrammetry and remote sensing*, 129, pp.32-40.

Tierney, L., Rossini, A., Li, N., and Sevcikova, H,, 2018. snow: Simple Network of Workstations. R package version 0.4-3. https://CRAN.R-project.org/package=snow
Wickham, W., 2016. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

Wickham, H., and Lionel Henry, L., 2019. tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions. R package version 0.8.3. https://CRAN.R-project.org/package=tidyr

17 Appendix A

Mapping Vegetation Data Sheet Date________

Species / Cover Type	**Point/Track**	**ID**	**Picture ID**	**Percent Cover**