

Multi-sensor Random Forest Classification in R
Carleton University
August 2019

The following document will walk you through running a random forest image classification in R Studio. RStudio is an open source software program designed for statistical analysis but also includes several geospatial and geostatistical tools.

You have been given a folder entitled “RF”. Inside, it contains two sub-folders and several files:

- “Rasters” folder contains a single .pix file named “Alfred.pix”. This file is referenced to UTM WGS84 Zone 18N.

You can open this in the GIS/Remote Sensing software of choice and explore. You should note that it contains 113 channels/bands and each band has a name, which is an abbreviated description on the sensor that created it and any processing that was done to create it (e.g. derivatives, indices, decompositions etc).

- “Original training data” folder contains two csv files
 - TRAINING.csv and TESTING.csv

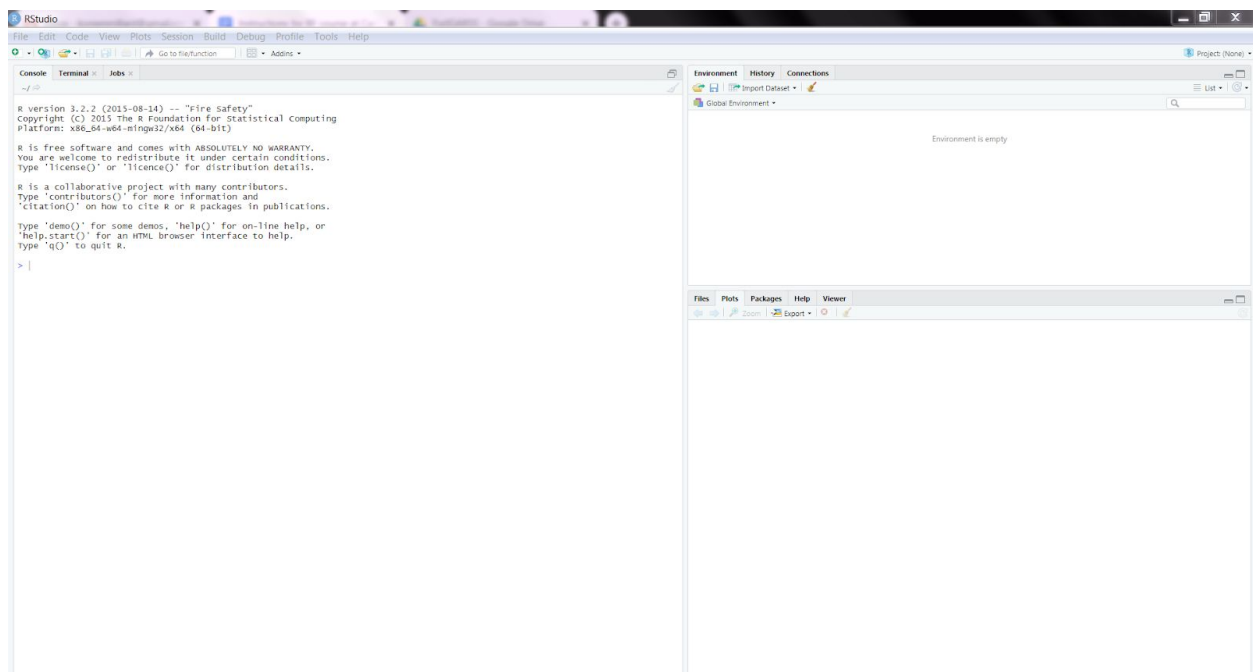
You can open these files in your excel or the GIS software of choice and explore. You should note that each row in the tables represents a single “point” and its spatial position is defined by POINT_X and POINT_Y which are referenced to UTM WGS84 Zone 18N. The training data file contains 332 records (points) and the validation or testing file contains 164 records. Both files contain the same fields (attributes) for each point. The fields that are named “Alfred_#” each represent the value of a single channel that have been extracted from imagery at that point location. We have pre-extracted these data values for you as the extraction process is time-consuming. The field “ClassName” represents the name of the class that exists at that specific location, and “ClassID” is simply a numeric representation of the class name (where Agriculture = 1, Forest = 2, Wetland = 3).

- An excel spreadsheet named “ListofVariables.xlsx” contains a more detailed description of each of the bands in the “Alfred.pix” raster. You will see that we have provided data from 3 different sensors (RapidEye, Radarsat-2 and LiDAR), and have created several decomposition parameters from the Radarsat-2 data (e.g. Freeman Durden Decomposition, Touzi Decomposition) which each result in several bands. Similarly, there are many LiDAR vegetation and surface derivatives. Each of these bands will be used as independent input in the Random Forest Classification model we create later on.
- The script “Random_Forests.R” is the script we will use in this tutorial.

Now, let's dive in!

1. Open R Studio

You should see something like this:



The panel on the left (console panel) is used for submitting commands for R Studio to run.

The top right panel contains tabs for “environment”, “history” and “connections”. The “environment” panel shows you any variables you have created as you run commands/scripts. The history tab show you the history of you commands. We won’t be using the connections tab in this course.

The bottom panel displays tabs for “files”, “plots”, “packages”, “help” and “viewer”. The files tab contains a list of all finds found in your working directory (more on what a working directory is later!), any “plots” (or graphs) you create will be displayed in the “plots” section, “packages” shows the list of packages that you have installed and you can click on specific ones to activate

them. The “help” tab allows you to search for specific packages or functions and see a description of what they do, their requirements and usage.

2. The first thing we will do is load a script into RStudio that has been created for this tutorial. Click File > Open File and then navigate to the RF.R script.

Now a script appears in the top left hand panel. The console panel is moved below it. This script is documented so that you can follow it through line-by-line. Comments in R are denoted with a # so any line starting with a # is a comment. For example, in the screenshot below the line

```
#1) Install relevant libraries is a “comment”
```

and

```
install.packages("raster", "randomForest", "sp", "rgdal",  
"caret") is a command that will run (and install several packages) when you tell it to.
```

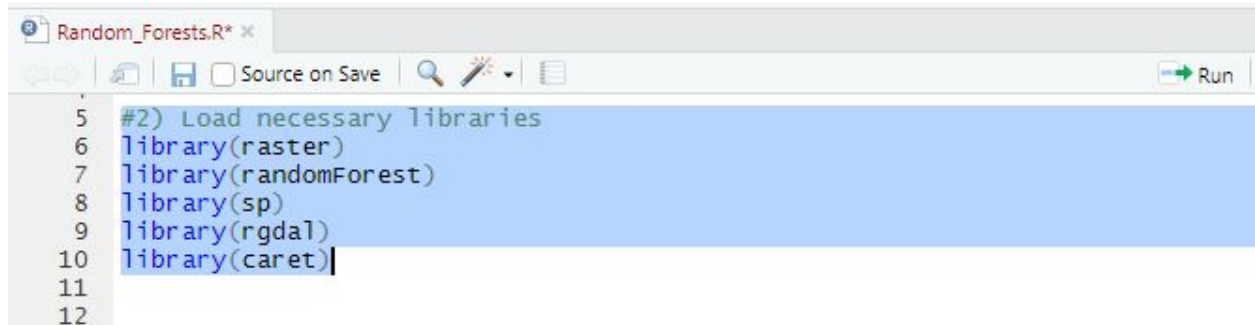
```
#1) Install relevant libraries  
install.packages("raster", "randomForest", "sp", "rgdal", "caret")
```

Section #1 installs any packages that are required. In the code window of RStudio, highlight the first 2 lines and then click run (located on the top right hand corner of the script panel).



You will see those two lines be executed below in the console window. You may be asked if you want to save the libraries to your personal directory - this is due to permissions on the lab computers and you should select “yes”. If those two commands were executed successfully, proceed to section #2 in the script.

3. Next, run section #2 in the script (all 6 lines), which activates the specific packages you need for this lab. If all of those lines were successful then proceed to the next step.



```
5 #2) Load necessary libraries
6 library(raster)
7 library(randomForest)
8 library(sp)
9 library(rgdal)
10 library(caret)|
11
12
```

Note: you will see some red text as the packages load all of their dependencies. Don't panic - red is not bad news! You should only be concerned if you see something failed or there was an error.

4. Step #3 simply defines the location of your “working directory”. A working directory is a drive/folder where you can automatically read and write files to, without typing out the full drive path. To check what location has been defined as your working directory, you can type `getwd()` and RStudio will print the full path. Here, I have set my working directory to the RF folder on my F drive. Change the text in the script to match the location of your RF folder.

```
12
13 #3) Set the working directory
14 setwd('F:/RF')
15 #in this case, my RF folder is stored on my F drive
16 #if your folder is stored in a different location, change the line above
17
```

5. Step #4 in the script reads in the Alfred.pix raster dataset.

```
#4) Create a raster object
inraster = raster::stack('rasters/Alfred.pix')
```

6. Step #5 in the script assigns the training and testing data tables to two variables (Training and Validation, respectively).

```
#5) Set the path to the training data
# these datasets are csv files containing class labels (ClassName)
# and Easting (POINT_X) and Northing (POINT_Y) information
Training = read.csv('TRAINING.csv', header=TRUE, sep = ",")
Validation = read.csv('TESTING.csv', header=TRUE, sep = ",")
```

7. Next, step #6 converts this a-spatial table into geospatial information using the POINT_X and POINT_Y fields. Step 7 assigns the coordinate reference system

```
#6) Identify which columns contain coordinate information
coordinates(Training)= ~Point_X+Point_Y
coordinates(Validation)= ~Point_X+Point_Y |
```

```
#7) Set the projection of the point data
proj4string(Training)<-CRS("+proj=utm +zone=18 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0")
proj4string(Validation)<-CRS("+proj=utm +zone=18 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0")
```

8. Step 8 allows you to select a subset of variables to use in your classification (predictor data). In the example below we have selected all of the columns that represent imagery (all Alfred_# fields). If you want to use on RapidEye data (for example), you would specify columns 7 - 16 only. You can use ListOfVariables.xlsx to determine the corresponding number of fields to channel names. Step 8 also defines the Training Response, which is simply the class that exists at each point and is used to train the model.

9. In Step 9, we use these two datasets (Predictor_Data and Training_Response) to create a random forest classification model!

```
#8) Select which variables to use in your model
Selection = c(7:119)
# column numbers 7 - 119 represent all of the channels
# names(Training) will show you the names and corresponding numbers of the columns
# use ListOfVariables.xlsx to determine what subsets of variables you want to use in classification
Predictor_Data = Training[,Selection] # Predictor_Data contains only the variables you select above
Training_Response = Training$ClassName # since we are doing classification, the response variable is the "class" at each point

#9) Create and save the forest
r_tree = randomForest(Predictor_Data, y=Training_Response, ntree = 1000, keep.forest=TRUE, importance = TRUE)
```

10. Step 10 allows you to display the results (out of bag error and confusion matrix). In this run, my results look like this:

```
Call:
randomForest(x = Predictor_Data, y = Training_Response, ntree = 1000, importance = TRUE)
Type of random forest: classification
Number of trees: 1000
No. of variables tried at each split: 10

OOB estimate of error rate: 7.83%
Confusion matrix:
      Agriculture forest wetland class.error
Agriculture      104      7      1 0.07142857
forest           4      41      8 0.22641509
wetland          2      4     161 0.03592814
```

Note that due to the random selection of variables and data points in creating trees and forests, your results might be slightly different!

11. Step 11 stores the variable importance values in a variable, and then prints these on the screen. Note: there are several options for saving variable importances (MDA, MDG, per-class). Try running several options by uncommenting out certain lines of code:

```
#11) Print the variable importance (Mean Decrease in Accuracy;
# for Gini Index type = 2;
# leaving type blank gives you the per-class importances (MDA))
imp = importance(r_tree, type = 1)
# imp = importance(r_tree, type = 2) |
# imp = importance(r_tree)
imp
```

12. Step #12 is perhaps the most important step in this exercise!

Here we are using the independent validation data (data that was set aside and not used in building the model) and using our random forest model (`r_tree`) to predict the class at each of the validation point locations. Since we already know what the validation point class should be (as it is a variable in the table), we can determine how well the classifier did at classifying at these new locations. The function itself is quite simple: we simply give the model and the data to the “predict” function. This does however require that your validation contains all the same fields, with the exact same names, as the training data.

```
#12) Classify the independent validation data
validation_Predictions = predict(r_tree, validation)
```

13. Step 13 and 14 compare the known class with the predicted class at each point to produce a confusion matrix, calculate overall classification accuracy (%), kappa, per-class accuracy and user's and producer's accuracy per class and store those in variables.

```
#13) Generate a confusion matrix from the independent validation data
validation_Response = as.factor(validation$ClassName)
confusionMatrix <- table(validation_Predictions, validation_Response)
confusionMatrix

#14) Calculate overall accuracy, user's and producer's accuracy and kappa statistic
n_obs <- length(validation_Response) # number of observation in validation set
n_classes <- length(levels(validation$ClassName)) # number of classes
overallAccuracy <- sum(diag(confusionMatrix)) / n_obs
classAccuracy <- matrix(NA, nrow=2, ncol=n_classes, dimnames=list(c('users', 'producers'), levels(validation$ClassName)))

for (c in 1:n_classes){
  classAccuracy['users', c] <- confusionMatrix[c, c] / sum(confusionMatrix[c, ])
  classAccuracy['producers', c] <- confusionMatrix[c, c] / sum(confusionMatrix[, c])
}
rowColSumProdSum <- sum(apply(confusionMatrix, 2, sum) * apply(confusionMatrix, 1, sum))
kappa <- (n_obs * sum(diag(confusionMatrix)) - rowColSumProdSum) / (n_obs^2 - rowColSumProdSum)

classAccuracy
overallAccuracy
kappa
```

Your results should look similar like this:


```

> classAccuracy
      Agriculture    forest    wetland
users      0.9583333 0.7333333 0.9883721
producers  0.9387755 0.9166667 0.9340659
> overallAccuracy
[1] 0.9329268
> kappa
[1] 0.8872359

```

Overall, our model has done quite well (93% overall accuracy and a kappa statistic of 0.89). However, the user's accuracy for the "forest" class is quite low (73%).

14. Once you are happy with your model results based on your independent validation, you can use the model to create a map of the predicted results throughout your entire study area. In Step 15, you must first rename the "inraster" so that it contains the same names as the Training data that was used to create the random forest model. Finally, step 15 applies to random forest model to every pixel in the "inraster" and saves it as a geotiff.

```

#15) Classify the whole raster
OutputRaster = 'rasters/output_classification.tif'
Names<-names(Training[,Selection])
names(inraster)<-Names

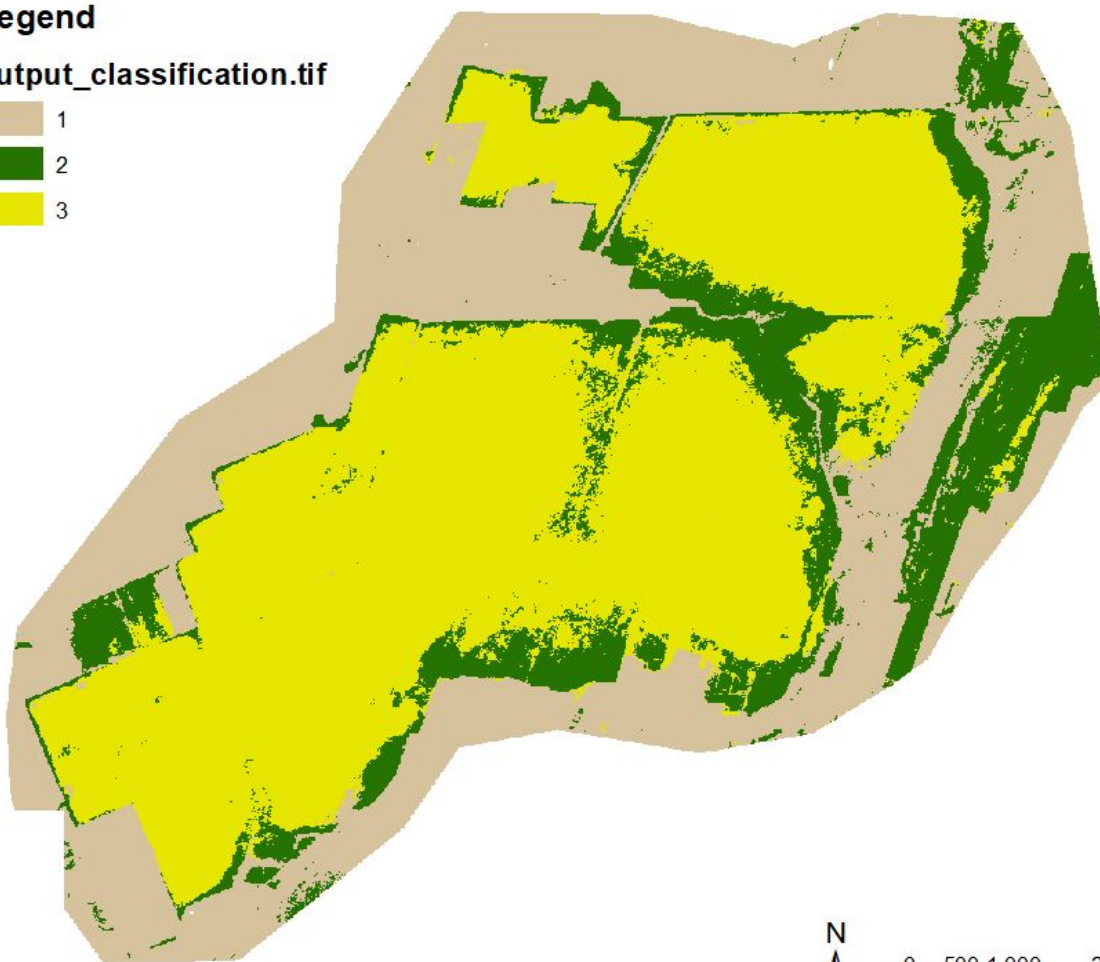
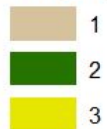
predictions = predict(inraster, r_tree, filename=OutputRaster,format="GTiff", progress="text", type="response")

```

The code specifies `progress = "text"` which means that RStudio will show us the progress as it creates the file. Once this step is complete, you can view the results in the GIS/Remote Sensing package of your choice. Your results should look similar to this:

Legend

output_classification.tif



0 500 1,000 2,000
Meters