Random Forest with Sentinel 1 & Sentinel 2

```
CLASSIFICATION OF SENTINEL 1/2 IMAGES USING RANDOM FOREST IN R

#load libraries
library(raster)
library(randomForest)
library(sp)
library(rgdal)
library(ggplot2)
library(caret)
set.seed(123)

#Setting-up the working directory
setwd("D:\\User\\Snigdha G Drive\\01 Masters\\01 ITC Netherlands\\01 Courses\\05
Qtr S23-N23\\01 Advanced Image Analysis (2023-1A)\\Assignment\\Data")

###### Import images
B2  = "B2.tif"
B3  = "B3.tif"
B4  = "B4.tif"
B8  = "B8.tif"
NDVI  = "NDVI.tif"
NDWI  = "NDWI.tif"
s1VH  = "VH_Speckle_Filtered.tif"
s1VV  = "VV_Speckle_Filtered.tif"

inraster        = stack(B2, B4, B8, NDVI, NDWI, s1VH, s1VV)


inraster

names(inraster) = c("B2", "B4", "B8", "NDVI", "NDWI", "s1VH", "s1VV")

# Define the extent of the study area
# For example, xmin, xmax, ymin, ymax in the same coordinate system as your raster
small_extent <- c(-83.090154852,-82.917199160,41.396962926,41.532573768)

# Crop the raster stack to the smaller extent
study_area <- crop(inraster, small_extent)

# Update the names of the smaller portion if needed
names(study_area) <- names(inraster)

# Display the smaller portion (optional)
plot(study_area)

# Remove NA values from the raster stack
inraster_no_na <- reclassify(study_area, cbind(NA,-999))

#================================================================================
=
# Import training and validation data
```

```
#=============================================================================
=
Training  =  shapefile("samples.shp")

#=============================================================================
=
# Split training data into training and test
#=============================================================================
=
set.seed(123)  # Set a seed for reproducibility
train_percentage1 <- 0.7  # You can adjust the percentage

# Create an index for the training data
train_indices <- sample(1:nrow(Training), nrow(Training) * train_percentage1)

# Split the data
trainingData <- Training[train_indices, ]
TestingData <- Training[-train_indices, ]

#=============================================================================
=
# Split training data further into training and validation
#=============================================================================
=
set.seed(123)  # Set a seed for reproducibility
train_percentage2 <- 0.8  # You can adjust the percentage

# Create an index for the training data
train_indices <- sample(1:nrow(trainingData), nrow(trainingData) *
train_percentage2)

# Split the data
trainingData_train <- trainingData[train_indices, ]
trainingData_validation <- trainingData[-train_indices, ]

#=============================================================================
=
# Extract raster values for the training and validation samples
#=============================================================================
=
training_data = extract(inraster_no_na, trainingData_train)
training_data
validation_data  = extract(inraster_no_na, trainingData_validation)
validation_data

training_response = as.factor(trainingData_train$Class)
training_response
validation_response = as.factor(trainingData_validation$Class)
validation_response

#=============================================================================
```

```
=
#Select the number of input variables(i.e. predictors, features)
#================================================================================
=
selection<-c(1:7)
training_predictors = training_data[,selection]
validation_predictors = validation_data[,selection]

#================================================================================
=
# Parameter tuning for random forest
#================================================================================
=
ntree_values <- c(10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500)
mtry_values <- c(1:7)

best_train_accuracy <- 0
best_validation_accuracy <- 0
best_ntree <- 0
best_mtry <- 0

# Create data frames to store results
results <- data.frame(ntree = numeric(), mtry = numeric(), validation_accuracy =
numeric())


for (ntree in ntree_values) {
  for (mtry in mtry_values) {
    r_forest <- randomForest(
      training_predictors, y = training_response,
      mtry = mtry, ntree = ntree,
      keep.forest = TRUE, importance = TRUE, proximity = TRUE
    )

    # Evaluate on the training set
    train_predictions <- predict(r_forest, training_predictors)
    train_accuracy <- sum(train_predictions == training_response) /
length(train_predictions)

    # Evaluate on the validation set
    validation_predictions <- predict(r_forest, validation_predictors)
    validation_accuracy <- sum(validation_predictions == validation_response) /
length(validation_predictions)

    # Update best parameters if the validation accuracy is higher
    if (validation_accuracy > best_validation_accuracy) {
      best_train_accuracy <- train_accuracy
      best_validation_accuracy <- validation_accuracy
      best_ntree <- ntree
      best_mtry <- mtry
    }
```

```
    # Store the results in the data frame
    results <- rbind(results, data.frame(ntree = ntree, mtry = mtry,
validation_accuracy = validation_accuracy))
  }
}

cat("Best ntree:", best_ntree, "\n")
cat("Best mtry:", best_mtry, "\n")
cat("Best Training Accuracy:", best_train_accuracy, "\n")
cat("Best Validation Accuracy:", best_validation_accuracy, "\n")

print(results)

#================================================================================
=
# Plotting graph of validation accuracies against nrees
#================================================================================
=
# Convert ntree to a factor for better visualization
results$ntree <- as.factor(results$ntree)

# Filter results for mtry value 1
results_mtry <- results[results$mtry == 3, ]

# Create the line graph
ggplot(results_mtry, aes(x = ntree, y = validation_accuracy, group = 1)) +
  geom_line(color = "blue") +
  geom_point(color = "blue", size = 3) +
  labs(title = "Validation Accuracy vs. ntree (mtry = 3)",
       x = "Number of Trees (ntree)",
       y = "Validation Accuracy") +
  theme_minimal()

#================================================================================
=
# Train the random forest
#================================================================================
=
# Train the final model with the best parameters
final_r_forest <- randomForest(
  training_predictors, y = training_response,
  mtry = best_mtry, ntree = best_ntree,
  keep.forest = TRUE, importance = TRUE, proximity = TRUE
)


#================================================================================
==
#Investigate the OOB (Out-Of-the bag) error
#================================================================================
==
```

```
final_r_forest

#================================================================================
==
# Assessment of variable importance
#================================================================================
==
varImpPlot(final_r_forest)
varUsed(final_r_forest)
importance(final_r_forest) #display importance output in console for ALL classes
individually

setwd('D:\\User\\Snigdha G Drive\\01 Masters\\01 ITC Netherlands\\01 Courses\\05
Qtr S23-N23\\01 Advanced Image Analysis (2023-1A)\\Assignment\\Data\\Output')

#================================================================================
=========
# Classify the entire image
#================================================================================
========

predictor_data = subset(inraster_no_na, selection)      #define raster data to use
for classification
predictions = predict(predictor_data, final_r_forest, progress="text",
type="response")
plot(predictions)


#================================================================================
=========
# Assess the classification accuracy
#================================================================================
=========
Testing=extract(predictions, TestingData) #extracts the value of the classified
raster at the validation point locations
Testing
Numeric=as.numeric(as.factor(TestingData$ClassID))
Numeric
confusionMatrix(as.factor(Testing), as.factor(Numeric))

#================================================================================
=========
# Save the classification results
#================================================================================
========

Class_Results = writeRaster(predictions, 'classified_image.tif', overwrite=TRUE,
col = your_colormap)
```