Remote Sensing Lab 8: Unsupervised and Supervised classification

Victor Gutierrez ([victorhugo@temple.edu](mailto:victorhugo@temple.edu))

## Lab due

November 04, 2021

## Goals

1. To learn graphical approaches to assess class separability for land cover classification.
2. To learn methods for unsupervised and supervised image classification and their implementation in R.

## Total score

The lab counts for up to 4 points towards the final grade of the course.

## Lab instructions

This lab assumes that some of the procedures learned in previous labs for downloading and pre-processing satellite images in R as well as deriving band transformations and spectral indices are already mastered and therefore are not covered here. Please refer to previous labs to implement such previous steps.

1. Open R Studio, open a new R script and change the route to your working directory in the script using the function setwd().
2. Copy and paste each chunk of code in your new R script and run it trying to understand the purpose, logic and syntaxis of each line. Make sure the code runs with no errors before moving to the next one.
3. Answer the questions in the answer sheet and submit it along with the required pdf files to canvas.

Have fun!

1. Load required libraries.

library(raster)  
library(rgdal)  
library(RStoolbox)  
library(rgl)  
library(prettymapr)  
library(stringr)

1. Load the two functions below. I created them for the visual separability assessment.

stack2df <- structure(function #Extracts into data frame   
### This function extracts into a data frame the pixel values for  
### all bands from different classes defined in a spatial dataframe  
 ##details<< This function ...  
(  
 inrast, ##<<\code{}.   
 invec, ##<<\code{}.   
 classcolname="class" ##<<\code{}.  
){  
 # extracts into a data frame the pixel values for all bands from different classes  
 # defined in a spatial dataframe  
 # inrast: the raster dataset containing pixel values to extract in [[1]]  
 # invec: spatial dataframe object defining the locations where the data  
 # should be extracted from   
 # classcolname: the column in the spatial dataframe containing the names   
 # of the attributes associated to those areas  
 # value: a data frame with columns representing the pixel values in each band for  
 # the areas labeled as defined by classcolname  
 if (is.null(raster::intersect(extent(invec), extent(inrast)))){  
 stop("the extents of inrast and invec do not overlap")  
 }  
 if(as.character(crs(inrast))!= as.character(crs(invec))){  
 stop("inrast and invec should have the same projection")  
 }  
 # required function  
 extractval=function(inraster=inrast, msk=msk){  
 outvector=raster::mask(inraster, msk)   
 outvector=na.omit(raster::getValues(outvector))  
 return(outvector)  
 }  
   
 # assign class ID to each class  
 invec$class\_ID=rep(NA, nrow(invec@data))  
 for (i in 1:length(invec[[classcolname]])){  
 invec$class\_ID[which(invec[[classcolname]]==unique(invec[[classcolname]])[i])]=i  
 }  
   
 # mask the input raster including pixels with valid values in all bands only  
 inrast=stackmask(inrast)  
   
 # create a raster of class\_ids. TRY gdalUtils::gdal\_rasterize. It might be faster!!!  
 calibrast=raster::rasterize(invec, inrast[[2]], field=invec$class\_ID)  
 calibmsk<-maskfun(calibrast, 0, 1, NA)  
 calibmsk=raster::mask(calibmsk, inrast[[2]])  
   
 # Extract pixel values into a dataframe  
 class\_ID=(extractval(calibrast, calibmsk))  
 dataset=data.frame(matrix(data=NA, nrow=length(class\_ID), ncol=nlayers(inrast[[1]])))  
   
 # add a column with a class name  
 dbclassname=rep(NA, length(class\_ID))  
 for (i in 1:length(unique(invec[[classcolname]]))){  
 dbclassname[which(class\_ID==i, arr.ind=TRUE)] = unique(invec[[classcolname]])[i]  
 }  
 commonclasses= match(sort(unique(dbclassname)), sort(levels(invec[[classcolname]])))  
 if(length(commonclasses)< length(levels(invec[[classcolname]]))){  
 missing=sort(levels(invec[[classcolname]]))[-commonclasses]  
 warning(paste(paste("the class", missing, sep= " "),   
 "has no valid pixels in input raster", sep=" "))  
 print(paste(paste("Warning: the class", missing, sep= " "),   
 "has no valid pixel values in the input raster", sep=" "))  
 }  
   
 dataset=cbind(class\_ID, dbclassname, dataset)  
 rm(class\_ID, dbclassname)  
  
 for (i in 1:nlayers(inrast[[1]])){  
 dataset[,i+2]=extractval(inrast[[1]][[i]], calibmsk)  
 print(paste(i, "layers extracted", sep=" "))  
 }  
 names(dataset)=c("class\_ID", "class\_name", names(inrast[[1]]))  
 return(dataset)  
### \code{}...   
} , ex=function(){  
 tarFiles <- c('LT050070651987081201T1-SC20181031175314.tar.gz',  
 'LT050060661988072201T1-SC20181031160603.tar.gz')  
 tarPaths <- system.file(tarFiles, package = 'aRn')  
 stack <- EEstackWithoutMeta(tarPaths, c(1:4))  
 strips <- RasterIntersection(stack)  
 })  
  
stackmask <- structure(function #Stack mask  
### This function ...  
 ##details<< This function ...  
(  
 inrast, ##<<\code{RasterBrick}. Reference raster.  
 maskrast=TRUE ##<<\code{logical}. Apply the mask to the raster  
 ##stackoutput: If maskrast=true a list with the  
 ##stack masked and the mask produced. If  
 ##maskrat=FALSE, the raster mask.  
){  
 # THIS FUNCTION HAS TO BE OPTIMIZED. IT TAKES TOO LONG  
 # masks out any pixels that have NAs in at least 1 band   
 # in a raster stack  
 # maskrast: TRUE applies the mask to the raster stack  
 # output: If maskrast=true a list with the stack masked and  
 # the mask produced. If maskrat=FALSE, the raster masck)  
 msk<-max(inrast)  
 msk<-maskfun(msk, 0,1,NA)  
 #inrast<-raster::mask(inrast, msk)  
 if (maskrast==TRUE){  
 inrast<-raster::mask(inrast, msk) # this is more than a minute faster than using the raster::mask function  
 out=list(inrast,msk)} else {  
 out=msk}  
 return(out)  
### \code{}...   
} , ex=function(){  
 tarFiles <- c('LT050070651987081201T1-SC20181031175314.tar.gz',  
 'LT050060661988072201T1-SC20181031160603.tar.gz')  
 tarPaths <- system.file(tarFiles, package = 'aRn')  
 stack <- EEstackWithoutMeta(tarPaths, c(1:4))  
 strips <- RasterIntersection(stack)  
 ## thrs <- thresraster(strips[[2L]], strips[[1L]])  
 ## noch <- nochg(thrs, degfree = nlayers(strips[[2L]]) - 1, pvalue = 4E-1)  
 ## calp <- calibrationParameters(strips[[2L]], strips[[1L]], noch, nbrackets = 8)  
 model <- PIFmodel(strips, pvalue = 4E-1, brackets = 8)  
 calib <- CalibrateRaster(model, stack)  
 ## merged <- merge(calib, stack[[2L]][[names(calib)]])  
 ## plotRGB(merged, r = 3, g = 2, b = 1, stretch = 'lin')  
})  
  
maskfun <- structure(function #Stack mask  
### This function sets a threshold value (thresh). Any pixels above  
### thresh are converted to aboveval. Any pixels bellow or equal to  
### thresh are converted to belowval  
 ##details<< This function ...  
(  
 x, ##<<\code{}...  
 thresh, ##<<\code{}...  
 aboveval, ##<<\code{}...  
 belowval ##<<\code{}...  
){  
 # sets a threshold value (thresh). Any pixels above   
 # thresh are converted to aboveval  
 # Any pixels bellow or equal to thresh are converted to belowval  
 require(raster)  
 v <- raster::getValues(x)  
 v[v>thresh]=aboveval  
 v[v<=thresh]=belowval  
 x <- raster::setValues(x, v)  
 return(x)  
### \code{}...   
} , ex=function(){  
 tarFiles <- c('LT050070651987081201T1-SC20181031175314.tar.gz',  
 'LT050060661988072201T1-SC20181031160603.tar.gz')  
 tarPaths <- system.file(tarFiles, package = 'aRn')  
 stack <- EEstackWithoutMeta(tarPaths, c(1:4))  
 strips <- RasterIntersection(stack)  
 ## thrs <- thresraster(strips[[2L]], strips[[1L]])  
 ## noch <- nochg(thrs, degfree = nlayers(strips[[2L]]) - 1, pvalue = 4E-1)  
 ## calp <- calibrationParameters(strips[[2L]], strips[[1L]], noch, nbrackets = 8)  
 model <- PIFmodel(strips, pvalue = 4E-1, brackets = 8)  
 calib <- CalibrateRaster(model, stack)  
 ## merged <- merge(calib, stack[[2L]][[names(calib)]])  
 ## plotRGB(merged, r = 3, g = 2, b = 1, stretch = 'lin')  
})  
  
plotSpectra=function(dataset=outdata, bandnames=c("B3\_dn", "B4\_dn", "B5\_dn"),  
 classfield=1, classlabels= levels(dataset[,classfield]),  
 classcol=sample(colors(), size=length(classlabels))){  
 #Assign colors to different labels  
 cols=rep(NA, length(classlabels))  
 for (i in 1:length(classlabels)){  
 cols[which(dataset[,classfield]==classlabels[i])]=classcol[i]  
 }  
   
 return(plot3d(outdata[,as.character(bandnames[1])],  
 outdata[, as.character(bandnames[2])],   
 outdata[, as.character(bandnames[3])],  
 xlab=bandnames[1],  
 ylab=bandnames[2],  
 zlab=bandnames[3],  
 col=cols))   
 legend3d("topleft", legend = classlabels,   
 pch = 16, col = classcol , cex=1, inset=c(0.02))  
}

1. Set working directory: Change the path below by copying and pasting the route to any selected folder in your workstation.

wd="/Users/tug61163/Documents/Courses/IntroRemoteSensing/2021Fall/Class10/LabMaterials"  
#dir.create(wd)  
setwd(wd)

1. Open the input images for classification. You can use the resized images from the previous labs either showing the original band information, spectral indices, tasseled cap trasnformation or all.

dir()  
L8name=stack("L8rsz.tif")  
tassCap=stack("L8tascap.tif")  
tcnames=c("brightness", "greenness", "wetness" )  
names(tassCap)=tcnames

1. Upload the shapefile representing the training polygons that you collected for different land cover classes and overlay them with the map. If the datasets do not overlay, it is likely due to a different projection between the raster and the vector files. In that case, the vector file should be reprojected as shown below:

# This plots the last bands corresponding to the tasseled cap transformation but can be modified to any bands of interest  
calibdata=readOGR(".", "Training\_Pucallpa\_2015\_Sel")  
# ADD SOME FUNCTIONS TO EXPLORE THE ATTRIBUTE TABLE OF calibdata  
  
plotRGB(tassCap, r=1, g=2, b=3, stretch="lin")   
plot(calibdata, add=T)  
  
# Check if the projections are the same  
crs(calibdata)  
crs(tassCap)  
  
# If they are not the same, reproject the vector file  
calibdataPrj=spTransform(calibdata, crs(L8all))

1. Perform an unsupervised classification. As the number of classes, select the same number as the number classes considered in the collection of training polygons

unsupervised=unsuperClass(tassCap, nClasses=7)  
plot(unsupervised$map, col=sample(colors(), 7))

1. Check the separability between different polygons visually. Write the name of the bands you want to plot in the plotSpectra() function. Add as many colors as the number of classes you have in your training data. You can find the name for different colors in R here: <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

calibdataPrj # Check the name of the column with the class names and enter it in the clascolname argument below  
outdata=stack2df(inrast=tassCap, invec=calibdataPrj,  
 classcolname="CLASS\_NAME")  
length(unique(outdata$class\_name)) # number of unique land cover classes in your training polygons  
  
# Create an object with the colors to assign to each land cover. The number of colors is equal to the number of unique classes identified for the study area (see previous line)  
classcolors= c("purple", "pink", "cadetblue1", "aquamarine", "blue", "black", "green",  
 "black", "cornsilk", "red", "chocolate1", "brown4", "gold3")  
   
# plotSpectra should create a new window showing with a 3D depiction of the pixel values in three bands defined by the argument "bandnames" for all land cover classes  
names(outdata) # Check the names of the bands you want to graph in plotSpectra  
  
plotSpectra(dataset=outdata, bandnames=tcnames, classfield=2,  
 classlabels=sort(unique(outdata$class\_name)),  
 classcol=classcolors)  
   
# Make sure that the name in the legend coincides with the name of the column representing class names. In the example below is Class.ID  
legend3d("topleft", legend= sort(unique(calibdataPrj$CLASS\_NAME)),  
 col=classcolors,  
 pch=3, cex=1, inset=c(0.02))

1. Perform supervised classification

supervised=superClass(tassCap, calibdataPrj, trainPartition=.7,  
 responseCol = "CLASS\_NAME" )  
plot(supervised$map, col=classcolors)

1. Retrieve variable importance for the supervised classification

# retrieves variable importance from the supervised object  
supervised$model$finalModel$importance

1. Save all data as R files and as Geotiffs

# To save data  
save(calibdataPrj, file="calibdata.RData") # save the polygons as an R object  
save(supervised, file="supervised.RData")  
save(unsupervised, file="unsupervised.RData")  
  
# To export spatial files in a format compatible with other applications  
writeOGR(calibdataPrj, dsn=getwd(), layer="calibdataPrj", # save polygons as shape file  
 driver="ESRI Shapefile")  
writeRaster(supervised$map, filename="supervised.tif",   
 filetype="GTiff")  
writeRaster(unsupervised$map, filename="unsupervised.tif",   
 filetype="GTiff")

1. Plot maps and figures and save them in a pdf document

unsupercol=sample(colors(), 7) # These are the colors to be used for mapping the unsupervised map  
plot(unsupervised$map, col=unsupercol, axes=FALSE, legend=T, box=FALSE)  
  
# Interpret the names of the classes in the unsupervised map and assign them to each category number below:  
catnamesUnsup=c("River","Secondary", "Burnt", "Pasture", "Forest", "Lake", "Unvegetated" )  
  
pdf("VGutierrezLab8.pdf", paper="USr", width=15)  
  
# PLOT THE UNSUPERVISED CLASSIFICATION  
unsupercol=c("blue", "pink", "gray", "yellow", "green", "cyan", "red") # These are the colors to be used for mapping the unsupervised map. Try to make them intuitive  
plot(unsupervised$map, col=unsupercol, axes=FALSE, legend=F, box=FALSE)  
# Add north arrow, scalebar.  
  
# use catnames$category below if you are using a PC  
legend("topright", legend = catnamesUnsup,  
 fill = unsupercol, cex=0.7, bg="white")  
  
prettymapr::addnortharrow(pos="bottomright", scale = 0.6, padin=c(0.5,0.1),  
 text.col = 'black', cols = c('black', 'black'))  
prettymapr::addscalebar(pos="bottomleft", plotunit = 'm', widthhint = 0.25, lwd = 1,   
 padin = c(0.5, 0.1), label.cex = 0.9)  
  
# PLOT THE SUPERVISED CLASSIFICATION  
# Retrieve the names of the different categories and   
# the pixel values assigned to each one of them  
catnames=supervised$map@data@attributes[[1]]$value  
catnames  
# This defines the colors to assign to each catname  
# The colors are assigned to each class are in the same order as they   
# appear in catnames. Make sure the colors are intuitive  
#mapcol=c("pink", "green", "lightblue", "yellow", "blue", "gray", "red")  
  
plot(supervised$map, col=classcol, axes=FALSE, legend=FALSE, box=FALSE)  
  
# use catnames$category below if you are using a PC  
legend("topright", legend = catnames,  
 fill = classcol, cex=0.3, bg="white")  
  
# Add north arrow, scalebar.  
prettymapr::addnortharrow(pos="bottomright", scale = 0.6, padin=c(0.5,0.1),  
 text.col = 'black', cols = c('black', 'black'))  
prettymapr::addscalebar(pos="bottomleft", plotunit = 'm', widthhint = 0.25, lwd = 1,   
 padin = c(0.5, 0.1), label.cex = 0.9)  
  
# PLOT PRODUCER'S AND USER'S ACCURACY  
barplot(rbind(users,producers),col=c("lightgreen","lightyellow"),   
 names.arg=classnames, beside = TRUE, ylab= "accuracy (%)")  
dev.off()

## Lab 9 deliverables

1. Add three X, Y, Z figures from different angles of the spectral space obtained through the application of the “plotSpectra()” function with the three bands that are best to discriminate between the land cover categories selected. Make sure the axes are at an angle that optimizes the visualization of the contrast in pixel values between categories (0.5 pts).
2. Respond the questions below based on the visualization of the spectral space (0.3 pts):
3. Which are the two land covers that you expect to be classified the most accurately? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. Which are the two land covers that you expect to be the most confused? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. Upload in canvas a single pdf file with the map obtained from the unsupervised and supervised classifications. The map corresponding to the results of the supervised classification should include a legend, north arrow and scale bar (0.2 pts).
6. Upload to canvas a single word document showing three zoomed-in images for one polygon per each land cover class. The first image corresponds to an RGB color composite from the original satellite image, the second one will be the results of the unsupervised classification and the third will be the results of the supervised classification. All the zoomed in images should include the outline of the polygon with no fill. You can produce these figures in QGIS or any other GIS software (1.2 pts).
7. Based on your visual interpretation of the classification results, please indicate which type of classification (unsupervised vs supervised) represents best the land covers of interest (0.2 pts)\_\_\_\_\_\_\_\_\_\_\_\_ .
8. For the best classification results, please respond the questions below (0.6 pts):
9. Which two land covers are best represented in your best classification (either supervised or unsupervised)?
10. Which two land covers have the worst representation?
11. Do these results agree with the confusion that you identified based on the analysis of the spectral space (point 2)?