**SUPPORTING INFORMATION S1**

**R-script to calculate Roa’s Q for satellite images (e.g. Landsat)**

The script is modified from:

Rocchini, D., Marcantonio, M., & Ricotta, C. (2017). Measuring Rao's Q diversity index from remote sensing: An open source solution. *Ecological indicators*, **72**, 234-238.

#' Create spectralrao function from

spectralrao <- function(input, distance\_m="euclidean", p=NULL, window=3, mode="multidimension", lambda=0, shannon=FALSE, rescale=FALSE, na.tolerance=0.0, simplify=3, nc.cores=8, cluster.type="MPI", debugging=FALSE, ...)

{

#

## Load required packages

#

require(raster)

require(svMisc)

require(proxy)

#

## Define function to check if a number is an integer

#

is.wholenumber <- function(x, tol = .Machine$double.eps^0.5) abs(x - round(x)) < tol

#

## Initial checks

#

if( !(is(input,"matrix") | is(input,"SpatialGridDataFrame") | is(input,"RasterLayer") | is(input,"list")) ) {

stop("\nNot a valid input object.")

}

if( is(input,"SpatialGridDataFrame") ) {

input <- raster(input) # Change input matrix/ces names

}

if( is(input,"matrix") | is(input,"RasterLayer")) {

rasterm<-input

} else if( is(input,"list") ) {

rasterm<-input[[1]]

}

if(na.tolerance>1){

stop("na.tolerance must be in the [0-1] interval. Exiting...")

}

# Deal with matrices and RasterLayer in a different way

# If data are raster layers

if( is(input[[1]],"RasterLayer") ) {

if( mode=="classic" ){

isfloat<-FALSE # If data are float numbers, transform them in integer, this may allow for a shorter computation time on big datasets.

if( !is.wholenumber(rasterm@data@min) | !is.wholenumber(rasterm@data@max) | is.infinite(rasterm@data@min) | !is.wholenumber(median(getValues(rasterm))) ){

message("Converting input data in an integer matrix...")

isfloat<-TRUE

mfactor<-100^simplify

rasterm<-getValues(rasterm)\*mfactor

rasterm<-as.integer(rasterm)

rasterm<-matrix(rasterm,nrow(input),ncol(input),byrow=TRUE)

gc()

}else{

rasterm<-matrix(getValues(rasterm),ncol=ncol(input),nrow=nrow(input),byrow=TRUE)

}

}

#Print user messages

if( mode=="classic" & shannon ){

message("Matrix check OK: \nRao and Shannon output matrices will be returned")

}else if( mode=="classic" & !shannon ){

message("Matrix check OK: \nRao output matrix will be returned")

}else if( mode=="multidimension" & !shannon ){

message(("Matrix check OK: \nA matrix with multimension RaoQ will be returned"))

}else if( mode=="multidimension" & shannon ){

stop("Matrix check failed: \nMultidimension and Shannon not compatible, set shannon=FALSE")

}else{

stop("Matrix check failed: \nNot a valid input | method | distance, please check all these options...")

}

# If data are a matrix or a list

}else if( is(input,"matrix") | is(input,"list") ) {

if( mode=="classic" ){

isfloat<-FALSE # If data are float numbers, transform them in integer

if( !is.integer(rasterm) ){

message("Converting input data in an integer matrix...")

isfloat<-TRUE

mfactor<-100^simplify

rasterm<-as.integer(rasterm\*mfactor)

rasterm<-matrix(rasterm,nrow(input),ncol(input),byrow=TRUE)

gc()

}else{

rasterm<-as.matrix(rasterm)

}

}

if( mode=="classic" & shannon ){

message("Matrix check OK: \nRao and Shannon output matrices will be returned")

}else if( mode=="classic" & !shannon ){

message("Matrix check OK: \nRao output matrix will be returned")

}else if( mode=="multidimension" & shannon ){

stop("Matrix check failed: \nMultidimension and Shannon not compatible, set shannon=FALSE")

}else if( mode=="multidimension" & !shannon ){

message(("Matrix check OK: \nA matrix with multimension RaoQ will be returned"))

}else{

stop("Matrix check failed: \nNot a valid input | method | distance, please check all these options")

}

}

if(nc.cores>1) {

if(mode=="multidimension"){

message(

"Multi-core is not supported for multidimensional Rao, proceeding with 1 core...")

nc.cores=1

}else{

message("

##################### Starting parallel calculation #######################")

}

}

#

## Derive operational moving window

#

if( window%%2==1 ){

w <- (window-1)/2

} else {

stop("The size of moving window must be an odd number. Exiting...")

}

#

## Preparation of output matrices

#

if(nc.cores==1) {

raoqe<-matrix(rep(NA,dim(rasterm)[1]\*dim(rasterm)[2]),nrow=dim(rasterm)[1],ncol=dim(rasterm)[2])

}

if(shannon){

shannond<-matrix(rep(NA,dim(rasterm)[1]\*dim(rasterm)[2]),nrow=dim(rasterm)[1],ncol=dim(rasterm)[2])

}

#

## If mode is classic Rao

#

if(mode=="classic") {

#

# If classic RaoQ is parallelized

#

if(nc.cores>1) {

#

## Required packages for parallel calculation

#

require(foreach)

require(doSNOW)

require(parallel)

if( cluster.type=="MPI" ){

require(Rmpi)

}

#

## Reshape values

#

values<-as.numeric(as.factor(rasterm))

rasterm\_1<-matrix(data=values,nrow=dim(rasterm)[1],ncol=dim(rasterm)[2])

#

## Add additional columns and rows to match moving window

#

hor<-matrix(NA,ncol=dim(rasterm)[2],nrow=w)

ver<-matrix(NA,ncol=w,nrow=dim(rasterm)[1]+w\*2)

trasterm<-cbind(ver,rbind(hor,rasterm\_1,hor),ver)

rm(hor,ver,rasterm\_1,values); gc()

if(debugging){cat("#check: RaoQ parallel function.")}

#

## Derive distance matrix

#

if( is.character( distance\_m) | is.function(distance\_m) ) {

d1<-proxy::dist(as.numeric(levels(as.factor(rasterm))),method=distance\_m)

} else if( is.matrix(distance\_m) | is.data.frame(distance\_m) ) {

d1<-stats::as.dist(xtabs(distance\_m[, 3] ~ distance\_m[, 2] + distance\_m[, 1]))

}

#

## Export variables in the global environment

#

if(isfloat) {

sapply(c("mfactor"), function(x) {assign(x,get(x),envir= .GlobalEnv)})

}

#

## Create cluster object with given number of slaves

#

plr<<-TRUE

if( cluster.type=="SOCK" || cluster.type=="FORK" ) {

cls <- parallel::makeCluster(nc.cores,type=cluster.type, outfile="",useXDR=FALSE,methods=FALSE,output="")

} else if( cluster.type=="MPI" ) {

cls <- makeMPIcluster(nc.cores,outfile="",useXDR=FALSE,methods=FALSE,output="")

}

registerDoSNOW(cls)

clusterCall(cl=cls, function() library("parallel"))

if(isfloat) {

parallel::clusterExport(cl=cls, varlist=c("mfactor"))

}

on.exit(stopCluster(cls)) # Close the clusters on exit

gc()

#

## Start the parallelized loop over iter

#

pb <- txtProgressBar(min = (1+w), max = dim(rasterm)[2], style = 3)

progress <- function(n) setTxtProgressBar(pb, n)

opts <- list(progress = progress)

raop <- foreach(cl=(1+w):(dim(rasterm)[2]+w),.options.snow = opts,.verbose = F) %dopar% {

if(debugging) {

cat(paste(cl))

}

raout <- sapply((1+w):(dim(rasterm)[1]+w), function(rw) {

if( length(!which(!trasterm[c(rw-w):c(rw+w),c(cl-w):c(cl+w)]%in%NA)) < window^2-((window^2)\*na.tolerance) ) {

vv<-NA

return(vv)

}

else {

tw<-summary(as.factor(trasterm[c(rw-w):c(rw+w),c(cl-w):c(cl+w)]),maxsum=10000)

if( "NA's"%in%names(tw) ) {

tw<-tw[-length(tw)]

}

if( debugging ) {

message("Working on coords ",rw,",",cl,". classes length: ",length(tw),". window size=",window)

}

tw\_labels <- names(tw)

tw\_values <- as.vector(tw)

#if clause to exclude windows with only 1 category

if( length(tw\_values) <2 ) {

vv<-NA

return(vv)

}

else {

p <- tw\_values/sum(tw\_values)

p1 <- diag(0,length(tw\_values))

p1[upper.tri(p1)] <- c(combn(p,m=2,FUN=prod))

d2 <- unname(as.matrix(d1)[as.numeric(tw\_labels),as.numeric(tw\_labels)])

vv <- sum(p1\*d2)

return(vv)

}

}

})

return(raout)

} # End classic RaoQ - parallelized

message(("\n\nCalculation of Rao's index complete.\n"))

#

## If classic RaoQ is sequential

#

} else if(nc.cores==1) {

# Reshape values

values<-as.numeric(as.factor(rasterm))

rasterm\_1<-matrix(data=values,nrow=dim(rasterm)[1],ncol=dim(rasterm)[2])

# Add fake columns and rows for moving window

hor<-matrix(NA,ncol=dim(rasterm)[2],nrow=w)

ver<-matrix(NA,ncol=w,nrow=dim(rasterm)[1]+w\*2)

trasterm<-cbind(ver,rbind(hor,rasterm\_1,hor),ver)

# Derive distance matrix

classes<-levels(as.factor(rasterm))

if( is.character(distance\_m) | is.function(distance\_m) ) {

d1<-proxy::dist(as.numeric(classes),method=distance\_m)

} else if( is.matrix(distance\_m) | is.data.frame(distance\_m) ) {

d1<-stats::as.dist(xtabs(distance\_m[, 3] ~ distance\_m[, 2] + distance\_m[, 1]))

}

# Loop over each pixel

for (cl in (1+w):(dim(rasterm)[2]+w)) {

for(rw in (1+w):(dim(rasterm)[1]+w)) {

if( length(!which(!trasterm[c(rw-w):c(rw+w),c(cl-w):c(cl+w)]%in%NA)) < window^2-((window^2)\*na.tolerance) ) {

raoqe[rw-w,cl-w]<-NA

} else {

tw<-summary(as.factor(trasterm[c(rw-w):c(rw+w),c(cl-w):c(cl+w)]),maxsum=10000)

if( "NA's"%in%names(tw) ) {

tw<-tw[-length(tw)]

}

if(debugging) {

message("Working on coords ",rw ,",",cl,". classes length: ",length(tw),". window size=",window)

}

tw\_labels <- names(tw)

tw\_values <- as.vector(tw)

#if clause to exclude windows with only 1 category

if(length(tw\_values) < 2) {

raoqe[rw-w,cl-w]<-NA

} else {

p <- tw\_values/sum(tw\_values)

p1 <- diag(0,length(tw\_values))

p1[upper.tri(p1)] <- c(combn(p,m=2,FUN=prod))

d2 <- unname(as.matrix(d1)[as.numeric(tw\_labels),as.numeric(tw\_labels)])

if(isfloat) {

raoqe[rw-w,cl-w]<-sum(p1\*d2)/mfactor

} else {

raoqe[rw-w,cl-w]<-sum(p1\*d2)

}

}

}

progress(value=cl, max.value=c((dim(rasterm)[2]+w)+(dim(rasterm)[1]+w))/2, progress.bar = FALSE)

}

} # End of for loop

message(("\nCalculation of Rao's index complete.\n"))

}

} # End classic RaoQ - sequential

else if( mode=="multidimension" ){

if(debugging) {

message("#check: Into multidimensional clause.")

}

#----------------------------------------------------#

#

## If multimensional RaoQ

#

# Check if there are NAs in the matrices

if ( is(rasterm,"RasterLayer") ){

if(any(sapply(lapply(unlist(input),length),is.na)==TRUE))

message("\n Warning: One or more RasterLayers contain NA which will be threated as 0")

} else if ( is(rasterm,"matrix") ){

if(any(sapply(input, is.na)==TRUE) ) {

message("\n Warning: One or more matrices contain NA which will be threated as 0")

}

}

#

## Check whether the chosen distance metric is valid or not

#

if( distance\_m=="euclidean" | distance\_m=="manhattan" | distance\_m=="canberra" | distance\_m=="minkowski" | distance\_m=="mahalanobis" ) {

#

## Define distance functions

#

#euclidean

multieuclidean <- function(x) {

tmp <- lapply(x, function(y) {

(y[[1]]-y[[2]])^2

})

return(sqrt(Reduce(`+`,tmp)))

}

#manhattan

multimanhattan <- function(x) {

tmp <- lapply(x, function(y) {

abs(y[[1]]-y[[2]])

})

return(Reduce(`+`,tmp))

}

#canberra

multicanberra <- function(x) {

tmp <- lapply(x, function(y) {

abs(y[[1]] - y[[2]]) / (abs(y[[1]]) + abs(y[[2]]))

})

return(Reduce(`+`,tmp))

}

#minkowski

multiminkowski <- function(x) {

tmp <- lapply(x, function(y) {

abs((y[[1]]-y[[2]])^lambda)

})

return(Reduce(`+`,tmp)^(1/lambda))

}

#mahalanobis

multimahalanobis <- function(x){

tmp <- matrix(unlist(lapply(x,function(y) as.vector(y))),ncol=2)

tmp <- tmp[!is.na(tmp[,1]),]

if( length(tmp)==0 | is.null(dim(tmp)) ) {

return(NA)

} else if(rcond(cov(tmp)) <= 0.001) {

return(NA)

} else {

#return the inverse of the covariance matrix of tmp; aka the precision matrix

inverse<-solve(cov(tmp))

if(debugging){

print(inverse)

}

tmp<-scale(tmp,center=T,scale=F)

tmp<-as.numeric(t(tmp[1,])%\*%inverse%\*%tmp[1,])

return(sqrt(tmp))

}

}

#

## Decide what function to use

#

if( distance\_m=="euclidean") {

distancef <- get("multieuclidean")

} else if( distance\_m=="manhattan" ) {

distancef <- get("multimanhattan")

} else if( distance\_m=="canberra" ) {

distancef <- get("multicanberra")

} else if( distance\_m=="minkowski" ) {

if( lambda==0 ) {

stop("The Minkowski Distance for lambda = 0 is Infinity; please choose another value for lambda.")

} else {

distancef <- get("multiminkowski")

}

} else if( distance\_m=="mahalanobis" ) {

distancef <- get("multimahalanobis")

warning("Multimahalanobis distance is not fully supported...")

}

} else {

stop("Distance function not defined for multidimensional Rao's Q; please choose among euclidean, manhattan, canberra, minkowski, mahalanobis!")

}

#

## Reshape values

#

vls<-lapply(input, function(x) {raster::as.matrix(x)})

#

## Rescale and add fake columns and rows for moving w

#

hor<-matrix(NA,ncol=dim(vls[[1]])[2],nrow=w)

ver<-matrix(NA,ncol=w,nrow=dim(vls[[1]])[1]+w\*2)

if(rescale) {

trastersm<-lapply(vls, function(x) {

t1 <- raster::scale(raster(cbind(ver,rbind(hor,x,hor),ver)))

t2 <- as.matrix(t1)

return(t2)

})

} else {

trastersm<-lapply(vls, function(x) {

cbind(ver,rbind(hor,x,hor),ver)

})

}

#

## Loop over all the pixels in the matrices

#

if( (ncol(vls[[1]])\*nrow(vls[[1]]))> 10000) {

message("\n Warning: ",ncol(vls[[1]])\*nrow(vls[[1]])\*length(vls), " cells to be processed, may take some time... \n")

}

for (cl in (1+w):(dim(vls[[1]])[2]+w)) {

for(rw in (1+w):(dim(vls[[1]])[1]+w)) {

if( length(!which(!trastersm[[1]][c(rw-w):c(rw+w),c(cl-w):c(cl+w)]%in%NA)) < window^2-((window^2)\*na.tolerance) ) {

raoqe[rw-w,cl-w] <- NA

} else {

tw<-lapply(trastersm, function(x) { x[(rw-w):(rw+w),(cl-w):(cl+w)]

})

#

## Vectorize the matrices in the list and calculate

#Among matrix pairwase distances

lv <- lapply(tw, function(x) {as.vector(t(x))})

vcomb <- combn(length(lv[[1]]),2)

vout <- c()

for(p in 1:ncol(vcomb) ) {

lpair <- lapply(lv, function(chi) {

c(chi[vcomb[1,p]],chi[vcomb[2,p]])

})

vout[p] <- distancef(lpair)

}

raoqe[rw-w,cl-w] <- sum(rep(vout,2) \* (1/(window)^4),na.rm=TRUE)

}

}

progress(value=cl, max.value=dim(rasterm)[2]+w, progress.bar = FALSE)

}

if(exists("pb")) {

close(pb)

message("\nCalculation of Multidimensional Rao's index complete.\n")

}

} else{

message("Something went wrong when trying to calculate Rao's indiex.")

} # end of multimensional RaoQ

#----------------------------------------------------#

#

## Shannon

#

if( shannon ) {

message("\nStarting Shannon-Wiener index calculation:\n")

# Reshape values

values<-as.numeric(as.factor(rasterm))

rasterm\_1<-matrix(data=values,nrow=dim(rasterm)[1],ncol=dim(rasterm)[2])

#

## Add "fake" columns and rows for moving window

#

hor<-matrix(NA,ncol=dim(rasterm)[2],nrow=w)

ver<-matrix(NA,ncol=w,nrow=dim(rasterm)[1]+w\*2)

trasterm<-cbind(ver,rbind(hor,rasterm\_1,hor),ver)

#

## Loop over all the pixels

#

for (cl in (1+w):(dim(rasterm)[2]+w)) {

for(rw in (1+w):(dim(rasterm)[1]+w)) {

if( length(!which(!trasterm[c(rw-w):c(rw+w),c(cl-w):c(cl+w)]%in%NA)) < window^2-((window^2)\*na.tolerance) ) {

shannond[rw-w,cl-w]<-NA

} else {

tw<-summary(as.factor(trasterm[c(rw-w):c(rw+w),c(cl-w):c(cl+w)]))

if( "NA's"%in%names(tw) ) {

tw<-tw[-length(tw)]

}

tw\_values<-as.vector(tw)

p<-tw\_values/sum(tw\_values)

p\_log<-log(p)

shannond[rw-w,cl-w]<-(-(sum(p\*p\_log)))

}

}

svMisc::progress(value=cl, max.value=(c((dim(rasterm)[2]+w)+(dim(rasterm)[1]+w))/2), progress.bar = FALSE)

}

message(("\nCalculation of Shannon's index is also complete!\n"))

} # End ShannonD

#----------------------------------------------------#

#

## Return multiple outputs

#

if(debugging){

message( "#check: return function." )

}

if( shannon ) {

if( nc.cores>1 ) {

outl<-list(do.call(cbind,raop),shannond)

names(outl)<-c("Rao","Shannon")

return(outl)

} else if( nc.cores==1 ){

outl<-list(raoqe,shannond)

names(outl)<-c("Rao","Shannon")

return(outl)

}

} else if( !shannon & mode=="classic" ) {

if( isfloat & nc.cores>1 ) {

return(do.call(cbind,raop)/mfactor)

if(debugging){

message("#check: return function - classic.")

}

} else if( !isfloat & nc.cores>1 ) {

outl<-list(do.call(cbind,raop))

names(outl)<-c("Rao")

return(outl)

} else if( isfloat & nc.cores==1 ) {

outl<-list(raoqe)

names(outl)<-c("Rao")

return(outl)

} else if( !isfloat & nc.cores==1 ) {

outl<-list(raoqe)

names(outl)<-c("Rao")

return(outl)

} else if( !isfloat & nc.cores>1 ) {

outl<-list(do.call(cbind,raoqe))

names(outl)<-c("Rao")

return(outl)

}

} else if( !shannon & mode=="multidimension" ) {

outl<-list(raoqe)

names(outl)<-c("Multidimension\_Rao")

return(outl)

}

}

library(raster)

library(rgdal)

require(raster)

landsat <- paste("C:/...", sep="")

#' Read in the raster as a brick

ras <- brick(landsat)

#' calculate heterogeneity

raomatrix<-spectralrao(ras,mode="multidimension",distance\_m="euclidean",window=3,shannon = F)

#' Save rasters

writeRaster(raster(raomatrix[[1]], template=norm\_rast), "multi\_rao\_", overwrite =T)