DSM- 3004
Spatial Statistics for Remotely Sensed Images

Mohammad Wasiq

Table of Contents

1 Pratical - 3

1.1 Objective

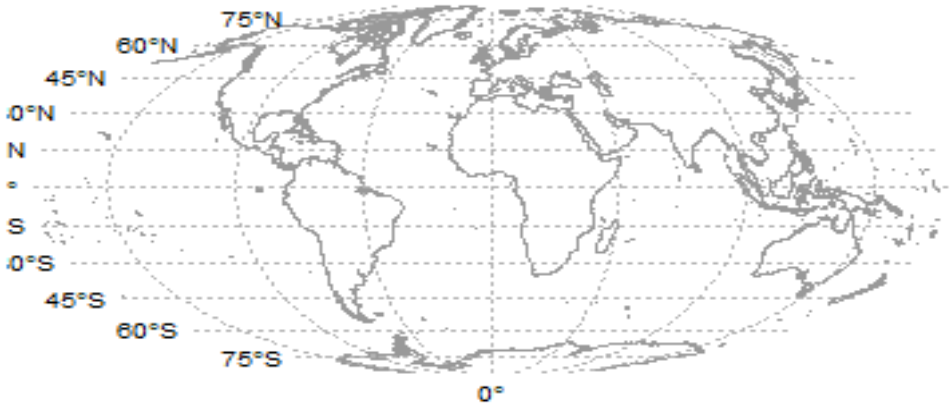
Draw a world map with Latitude and Longitude degrees in R and generate an object of class Spatial Lines. Un-projected data have coordinates in latitude and longitude degrees, with negative degrees referring to degrees west (of the prime meridian) and south (of the Equator). When un-projected spatial data are plotted using sp methods (plot or spplot), the axis label marks will give units in decimal degrees N/S/E/W, for example 50.5 ◦ N.

When, for reference purposes, a grid needs to be added to a map, the function gridlines can be used to generate an object of class Spatial Lines. By default it draws lines within the bounding box of the object at values where the default axes labels are drawn; other values can be specified. Grid lines may be latitude/longitude grids, and these are non-straight lines. This is accomplished by generating a grid for un-projected data, projecting it, and plotting it over the map shown.

This is the code used to define and draw projected latitude/longitude grid lines and grid line labels, which uses the world map from package **maps** :

```
library(maptools)
library(maps)
library(MAP)

wrld <- map("world", interior = FALSE, xlim = c(-179, 179), ylim = c(-89, 89)
, plot = FALSE)
```

```
wrld_p<- pruneMap(wrld, xlim = c(-179, 179))

llCRS <- CRS("+proj=longlat +ellps=WGS84")      #[WGS84, World Geodetic    System 1984]

## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_p
roj = prefer_proj): Discarded
## datum Unknown based on WGS84 ellipsoid in Proj4 definition

wrld_sp <- map2SpatialLines(wrld_p, proj4string = llCRS)

prj_new <- CRS("+proj=moll")

library(rgdal)

## Please note that rgdal will be retired by the end of 2023,
## plan transition to sf/stars/terra functions using GDAL and PROJ
## at your earliest convenience.
##
## rgdal: version: 1.5-32, (SVN revision 1176)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.4.3, released 2022/04/22
## Path to GDAL shared files: C:/Users/hp/AppData/Local/R/win-library/4.2/rgd
al/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ_VERSION: 721]
## Path to PROJ shared files: C:/Users/hp/AppData/Local/R/win-library/4.2/rgd
al/proj
## PROJ CDN enabled: FALSE
## Linking to sp version:1.5-0
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading sp
or rgdal.

wrld_proj <- spTransform(wrld_sp, prj_new)

wrld_grd <- gridlines(wrld_sp, easts = c(-179, seq(-150, 150, 50), 179.5), no
rths = seq(-75, 75, 15), ndiscr = 100)

wrld_grd_proj <- spTransform(wrld_grd, prj_new)

at_sp <- gridat(wrld_sp, easts = 0, norths = seq(-75, 75, 15), offset = 0.3)

at_proj <- spTransform(at_sp, prj_new)

plot(wrld_proj, col = "grey60")
plot(wrld_grd_proj, add = TRUE, lty = 3, col = "grey70")
text(coordinates(at_proj),pos = at_proj$pos, offset = at_proj$offset,
labels = parse(text = as.character(at_proj$labels)), cex = 0.6)
```




```
plot(wrld_proj,col="grey10")
```

1.1.1 Plotting Attributes and Map Legends:

Up to now we have only plotted the geometry or topology of the spatial objects. If in addition we want to show feature characteristics or attributes of the objects, we need to use type, size, or colour of the symbols, lines, or polygons. Grid cells are usually plotted as small adjacent squares, so their plotting is in some sense a special case of plotting polygons. Following Table lists the graphic arguments that can be passed to the plot methods for the Spatial classes with attributes. When a specific colour, size, or symbol type refers to a specific numeric value or category label of an attribute, a map legend is needed to communicate this information.

Table: Useful annotation arguments to be passed to plot or image methods.

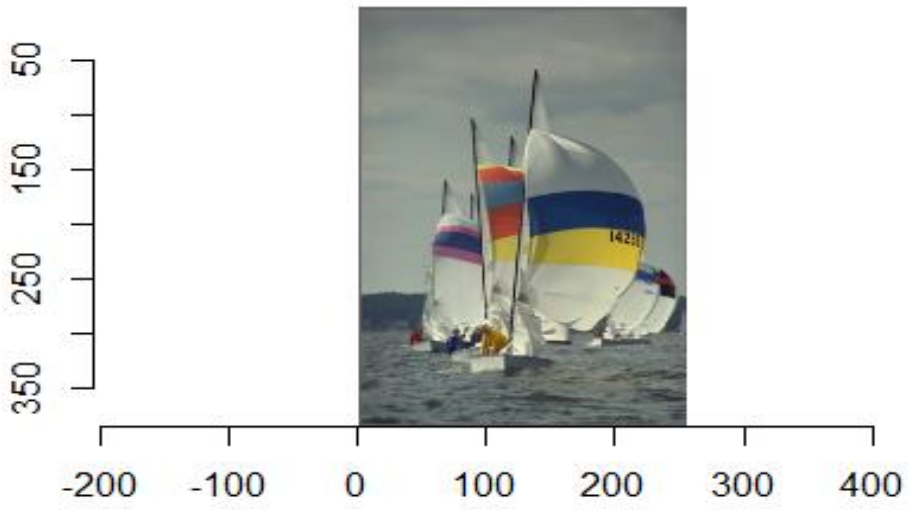
Class(es)	Argument	Meaning	Further help
Spatial Lines Data Frame	Col	Colour	?lines
	lwd	Line width	?lines
	lty	Line type	?lines
Spatial Polygons Data Frame	border	Border colour	?polygon
	density	Hashing density	?polygon
	angle	Hashing angle	?polygon
	lty	Line type	?polygon
	pbg	Hole colour	?polygon
Spatial Points Data Frame	Pch	Symbol	?points
	Col	Colour	?points
	bg	Fill colour	?points
	cex	Symbol size	?points
Spatial Pixels Data Frame and Spatial Grid Data Frame	zlim	Attribute value limits	?image.default
	col	Colours	?image.default
	breaks	Break points	?image.default

2 Practical - 4

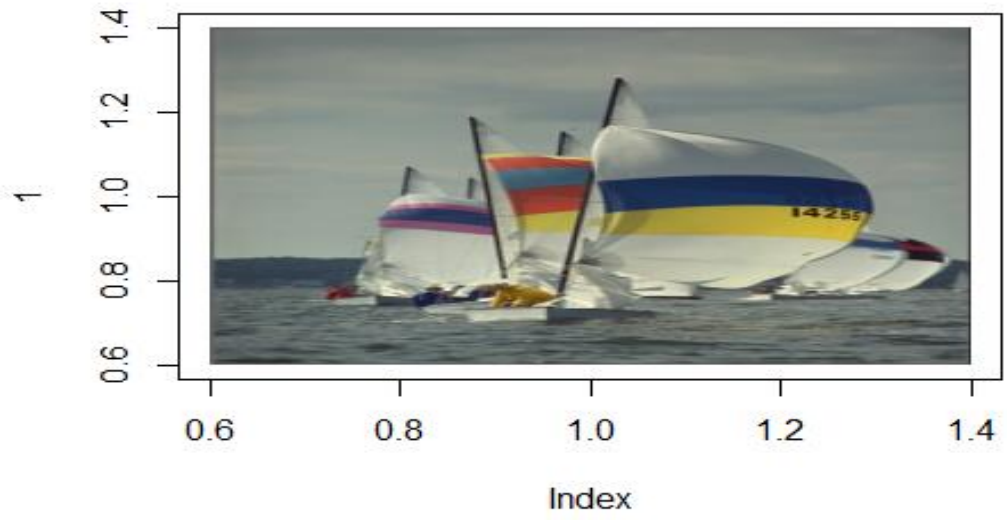
2.1 Objective

Using Principal Component Analysis (PCA) in an image processing for reducing the dimensions of an image and find the values of corresponding Principal Components.

```
library(imager)
library(imgpalr)
library(factoextra)
library(gridExtra)
library(ggplot2)
library(magick)
library(jpeg)

plot(boats)

photo<- plot(boats)
```



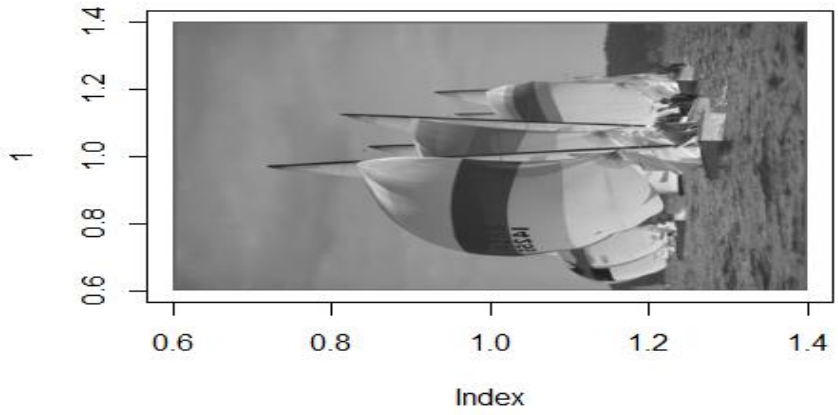
```
plot(1, type= "n")
rasterImage(photo, 0.6, 0.6, 1.4, 1.4)
```



In the colour photo, we get three matrices pixel by pixel and each of these matrices is for one component of RGB colour. Moreover, it's an easy way to convert to grayscale is to sum up RGB shades and divide by max value to scale up to max. 1.

2.1.1 PCA

```
photo.sum<- photo[,,1]+ photo[,,2]+photo[,,3]
photo.bw<-photo.sum/max(photo.sum)
plot(1, type="n")
rasterImage(photo.bw, 0.6, 0.6, 1.4, 1.4)
```



Principal Component Analysis (PCA) for pictures is to run individual PCA on each of shades, which are R, G, and B. So it gives us an eigenvector of shades. The first eigenvector shows the majority of variance of shades and the next vectors a little bit less. We have to integrate new shades into the picture. And this new value comes from multiplying "x" and "rotation" components of PCA. Here each colour scale (R, G, B) gets its own matrix and PCA since we work on matrices.

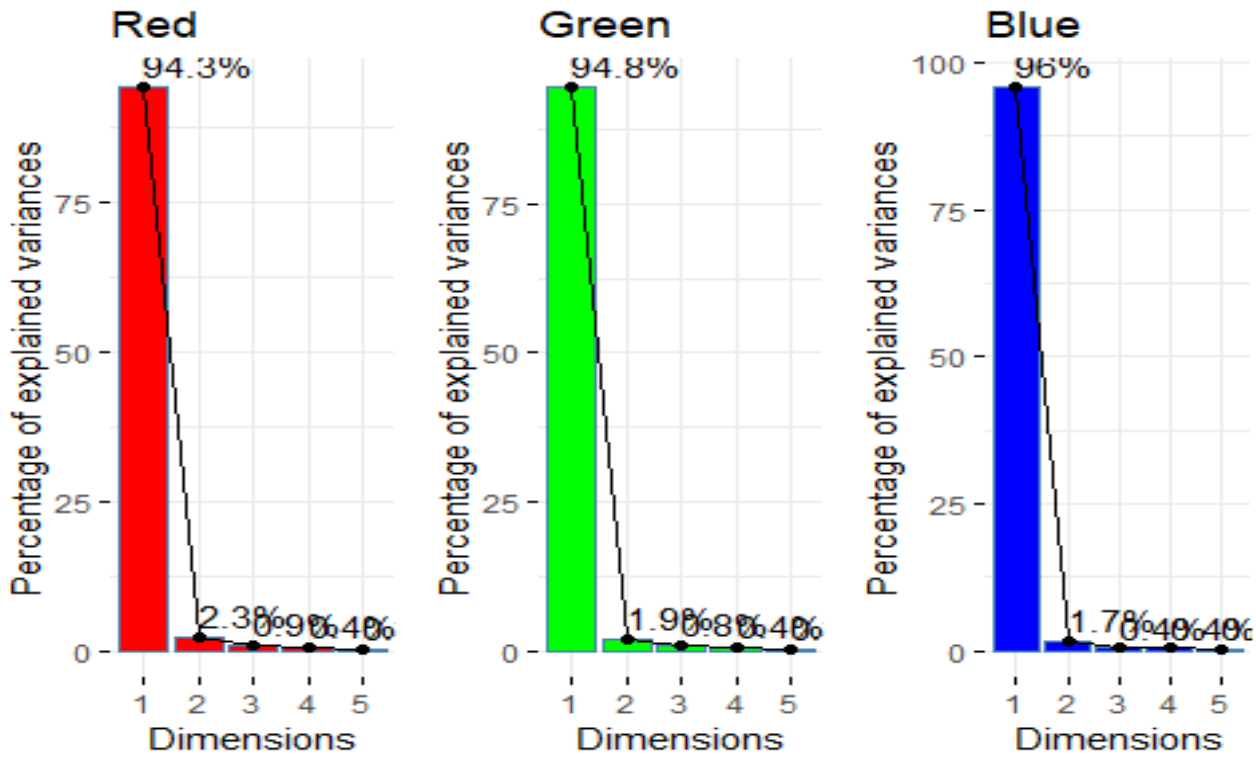
```
r<-photo[,,1]
# individual matrix of R color component

g<-photo[,,2]
# individual matrix of G color component

b<-photo[,,3]
# individual matrix of B color component
```

Now, we use Principal Component Analysis (PCA) for each colour components and will merge them into one object.

```
r.pca<-prcomp(r, center=FALSE, scale.=FALSE)
# PCA for R color component
g.pca<-prcomp(g, center=FALSE, scale.=FALSE)
# PCA for G color component
b.pca<-prcomp(b, center=FALSE, scale.=FALSE)
# PCA for B color component
rgb.pca<-list(r.pca, g.pca, b.pca)
# merging all PCA into one object
```

Now, we want to find Principal Components (PC) for the given image data.

```
library(gridExtra)
f1<-fviz_eig(r.pca, main="Red", barfill="red", ncp=5, addlabels=TRUE)
f2<-fviz_eig(g.pca, main="Green", barfill="green", ncp=5, addlabels=TRUE)
f3<-fviz_eig(b.pca, main="Blue", barfill="blue", ncp=5, addlabels=TRUE)
grid.arrange(f1, f2, f3, ncol=3)
```



Let's check the number of PC

```
vec<-seq.int(3,round(nrow(photo)),length.out=9)
round(vec,0)
```

```
## [1]   3  35  66  98 130 161 193 224 256
```