Week 9 Code Examples, CSC 400, Spring 2024

1. Image processing
2. Geospatial analysis
3. Outlier analysis/Anomaly detection
4. Visualization

Image Processing

We'll load packages and a sample image to work on.

```
install.packages("BiocManager")
BiocManager::install("EBImage")
library("EBImage")
f = system.file("images", "sample.png", package="EBImage")
img = readImage(f)
display(img, method="raster")
text(x = 20, y = 20, label = "Parrots", adj = c(0,1), col = "orange", cex = 2)
```

We look at some image properties and bring in a second image to work on.

```
filename = "parrots.jpg"
dev.print(jpeg, filename = filename , width = dim(img)[1], height = dim(img)[2])
file.info(filename)$size

imgcol = readImage(system.file("images", "sample-color.png", package="EBImage"))
display(imgcol)

nuc = readImage(system.file("images", "nuclei.tif", package="EBImage"))
display(nuc, method = "raster", all = TRUE)
```

After modifying an image, we can save it. In this example, we've reduced the quality of the image from the default of 100 to 85.

```
writeImage(imgcol, "sample.jpeg", quality = 85)
```

We can look at some properties of the image.

```
str(img)
dim(img)
imageData(img)[1:3, 1:6]
is.Image( as.array(img) )
hist(img)
range(img)
img
print(img, short=TRUE)
print(imgcol, short=TRUE)
numberOfFrames(imgcol, type = "render")
numberOfFrames(imgcol, type = "total")
nuc
```

We can manipulate the color of the image.

```
colorMode(imgcol) = Grayscale
display(imgcol, all=TRUE)
colorMat = matrix(rep(c("red","green", "#0000ff"), 25), 5, 5)
colorImg = Image(colorMat)
colorImg
display(colorImg, interpolate=FALSE)
```

We can also manipulate the image. Here, we find the negative of the image, and adjust the contrast.

```
img_neg = max(img) - img
display( img_neg )

img_comb = combine(
  img,
  img + 0.3,
  img * 2,
  img ^ 0.5
)

display(img_comb, all=TRUE)
```

We can crop and transpose the image.

```
img_crop = img[366:749, 58:441]
img_thresh = img_crop > .5
display(img_thresh)
img_thresh

img_t = transpose(img)
display( img_t )
```

We can do spatial transformations like rotations.

```
img_translate = translate(img, c(100,-50))
display(img_translate)

img_rotate = rotate(img, 30, bg.col = "white")
display(img_rotate)

img_resize = resize(img, w=256, h=256)
display(img_resize )

img_flip = flip(img)
img_flop = flop(img)

display(combine(img_flip, img_flop), all=TRUE)

m =  matrix(c(1, -.5, 128, 0, 1, 0), nrow=3, ncol=2)
img_affine = affine(img, m)
display( img_affine )
```

We can also apply various filters to the image.

```
w = makeBrush(size = 31, shape = 'gaussian', sigma = 5)
plot(w[(nrow(w)+1)/2, ], ylab = "w", xlab = "", cex = 0.7)

img_flo = filter2(img, w)
display(img_flo)

nuc_gblur = gblur(nuc, sigma = 5)
display(nuc_gblur, all=TRUE )

fhi = matrix(1, nrow = 3, ncol = 3)
fhi[2, 2] = -8
img_fhi = filter2(img, fhi)
display(img_fhi)
```

We can add noise and then use a filter to remove most of it.

```
l = length(img)
n = l/10
pixels = sample(l, n)
img_noisy = img
img_noisy[pixels] = runif(n, min=0, max=1)
display(img_noisy)

img_median = medianFilter(img_noisy, 1)
display(img_median)
```

Let's see what happens when we manipulate the package logo.

```
shapes = readImage(system.file('images', 'shapes.png', package='EBImage'))
logo = shapes[110:512,1:130]
display(logo)

kern = makeBrush(5, shape='diamond')
display(kern, interpolate=FALSE)

logo_erode= erode(logo, kern)
logo_dilate = dilate(logo, kern)

display(combine(logo_erode, logo_dilate), all=TRUE)
```

```
threshold = otsu(nuc)
threshold

nuc_th = combine( mapply(function(frame, th) frame > th, getFrames(nuc),
                         threshold, SIMPLIFY=FALSE) )
display(nuc_th, all=TRUE)
```

We can also vary the thresholding.

```
disc = makeBrush(31, "disc")
disc = disc / sum(disc)
offset = 0.05
nuc_bg = filter2( nuc, disc )
nuc_th = nuc > nuc_bg + offset
display(nuc_th, all=TRUE)
display( thresh(nuc, w=15, h=15, offset=0.05), all=TRUE )
```

We can experiment with image segmentation and make a Voronoi diagram.

```
logo_label = bwlabel(logo)
table(logo_label)
max(logo_label)
display( normalize(logo_label) )
display( colorLabels(logo_label) )

nmask = watershed( distmap(nuc_th), 2 )
display(colorLabels(nmask), all=TRUE)

voronoiExamp = propagate(seeds = nmask, x = nmask, lambda = 100)
voronoiPaint = colorLabels (voronoiExamp)
display(voronoiPaint)
```

If we can segment the image, we can also rearrange the elements.

```
objects = list(
  seq.int(from = 2, to = max(logo_label), by = 2),
  seq.int(from = 1, to = max(logo_label), by = 2)
)
logos = combine(logo_label, logo_label)
z = rmObjects(logos, objects, reenumerate=FALSE)
display(z, all=TRUE)
showIds = function(image) lapply(getFrames(image), function(frame) unique(as.vector(frame)))

showIds(z)
showIds( reenumerate(z) )
```

We can fill spaces with color, or add an image onto another one as a watermark.

```
filled_logo = fillHull(logo)
display(filled_logo)

rgblogo = toRGB(logo)
points = rbind(c(50, 50), c(100, 50), c(150, 50))
colors = c("red", "green", "blue")
rgblogo = floodFill(rgblogo, points, colors)
display( rgblogo )

display(floodFill(img, rbind(c(200, 300), c(444, 222)), col=0.2, tolerance=0.2))

d1 = dim(img)[1:2]
overlay = Image(dim=d1)
d2 = dim(logo_label)-1

offset = (d1-d2) %/% 2
```

```
overlay[offset[1]:(offset[1]+d2[1]), offset[2]:(offset[2]+d2[2])] = logo_label

img_logo = paintObjects(overlay, toRGB(img), col=c("red", "yellow"), opac=c(1, 0.3), thick=TRUE)

display( img_logo )
```

Geospatial Analysis

Analyzing geospatial data can be quite different from traditional statistical data, so this will just be the barest glimpse of this massive field. First, we load package, and import and plot the data.

```
library(sf)
world <- st_read(system.file("shape/nc.shp", package="sf"))
plot(world)
```

We can load earthquake data and plot it on a map of the world.

```
library(ggplot2)
library(maps)
library(plotly)
data(quakes)
world_map <- map_data("world")
ggplot() +
  geom_polygon(data = world_map, aes(x = long, y = lat, group = group),
               fill = "white", color = "black") +
  geom_point(data = quakes, aes(x = long, y = lat, size = mag,
                                color = depth), alpha = 0.7) +
  scale_size_continuous(range = c(1, 10)) +
  scale_color_gradient(low = "blue", high = "red") +
  labs(
    title = "Global Earthquake Occurrences",
    subtitle = "Magnitude and Depth",
    x = "",
    y = ""
  ) +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5, size = 18),
        plot.subtitle = element_text(hjust = 0.5, size = 14))
```

We can also plot dataframe data using spatial packages, if it's of the correct format.

```
library(sp)
coords <- data.frame(
  x = c(0, 1, 2),
  y = c(0, 1, 0)
)
points <- SpatialPoints(coords)
df <- data.frame(ID = c("A", "B", "C"))
spdf <- SpatialPointsDataFrame(points, df)
plot(spdf, pch = 19, col = "red", main = "Simple SpatialPointsDataFrame")
```

Outliers/Anomaly Detection

We discussed some methods in previous courses, so we'll start with some basic statistical measures.

```
dat <- ggplot2::mpg
summary(dat$hwy)
hist(dat$hwy,
     xlab = "hwy",
     main = "Histogram of hwy",
     breaks = sqrt(nrow(dat))
)
library(ggplot2)
ggplot(dat) +
  aes(x = hwy) +
  geom_histogram(bins = 30L, fill = "#0c4c8a") +
  theme_minimal()

boxplot(dat$hwy,
        ylab = "hwy"
)
ggplot(dat) +
  aes(x = "", y = hwy) +
  geom_boxplot(fill = "#0c4c8a") +
  theme_minimal()
```

We can use the fact that the boxplot calculates outliers to extract them.

```
boxplot.stats(dat$hwy)$out
out <- boxplot.stats(dat$hwy)$out
out_ind <- which(dat$hwy %in% c(out))
out_ind
dat[out_ind, ]

boxplot(dat$hwy,
        ylab = "hwy",
        main = "Boxplot of highway miles per gallon"
)
mtext(paste("Outliers: ", paste(out, collapse = ", ")))
```

We can also identify outliers by specifying percentiles.

```
lower_bound <- quantile(dat$hwy, 0.025)
lower_bound
upper_bound <- quantile(dat$hwy, 0.975)
upper_bound

outlier_ind <- which(dat$hwy < lower_bound | dat$hwy > upper_bound)
outlier_ind

dat[outlier_ind, "hwy"]
dat[outlier_ind, ]
```

```
lower_bound <- quantile(dat$hwy, 0.01)
upper_bound <- quantile(dat$hwy, 0.99)

outlier_ind <- which(dat$hwy < lower_bound | dat$hwy > upper_bound)

dat[outlier_ind, ]
```

We can also use z-scores if we want.

```
dat$z_hwy <- scale(dat$hwy)
hist(dat$z_hwy)
summary(dat$z_hwy)
which(dat$z_hwy > 3.29)
```

There are a number of statistical tests and filters that we can use to identify outliers as well. The Hampel filter uses extreme outliers instead of regular outliers. We also have Grubb's Test, Dixon's Test and the Rosner Test.

```
lower_bound <- median(dat$hwy) - 3 * mad(dat$hwy, constant = 1)
lower_bound
upper_bound <- median(dat$hwy) + 3 * mad(dat$hwy, constant = 1)
upper_bound
outlier_ind <- which(dat$hwy < lower_bound | dat$hwy > upper_bound)
outlier_ind

library(outliers)
test <- grubbs.test(dat$hwy)
test
test <- grubbs.test(dat$hwy, opposite = TRUE)
test
dat[34, "hwy"] <- 212
test <- grubbs.test(dat$hwy)
test
```

We can use the results to remove the problematic points.

```
remove_ind <- which.min(subdat$hwy)
subsubdat <- subdat[-remove_ind, ]
test <- dixon.test(subsubdat$hwy)
test
```

The Rosner Test can test multiple points at once.

```
library(EnvStats)
test <- rosnerTest(dat$hwy,
                   k = 3
)
test
test$all.stats
```

There are a number of other packages you can experiment with. One example:

```
library(mvoutlier)

Y <- as.matrix(ggplot2::mpg[, c("cyl", "hwy")])
res <- aq.plot(Y)
```

The specific package and tests one uses will depend on the kind of data you have. If you have clustering or classification data, you'll use different packages than if you have numerical or regression data. The final reference below [6] has an extremely complete list of methods and contexts to consider.

Resources:

1. https://bioconductor.org/packages/devel/bioc/vignettes/EBImage/inst/doc/EBImage-introduction.html
2. https://guides.library.duke.edu/r-geospatial
3. https://pmarchand1.github.io/atelier_rgeo/rgeo_workshop.html
4. https://pjbartlein.github.io/REarthSysSci/geospatial.html
5. https://www.geeksforgeeks.org/geospatial-data-analysis-with-r/
6. https://rpubs.com/michaelmallari/anomaly-detection-r
7. https://towardsdatascience.com/tidy-anomaly-detection-using-r-82a0c776d523