A Colorful Christmas:

about Stats and Floyd-Steinberg Dithering

Yizhen (Jeremy) Dai / S2395479

07 January, 2020

1 About Colors: A Picture = A Data set.

1.1 Explore your PNG R object

Load the file.

```
# file.info("0_img/xmas.png")[, c("size", "mtime")]
xmas <- png::readPNG(source = '0_img/xmas.png', native = FALSE)
xmas <- xmas[,,-4] # first three matrices only
```

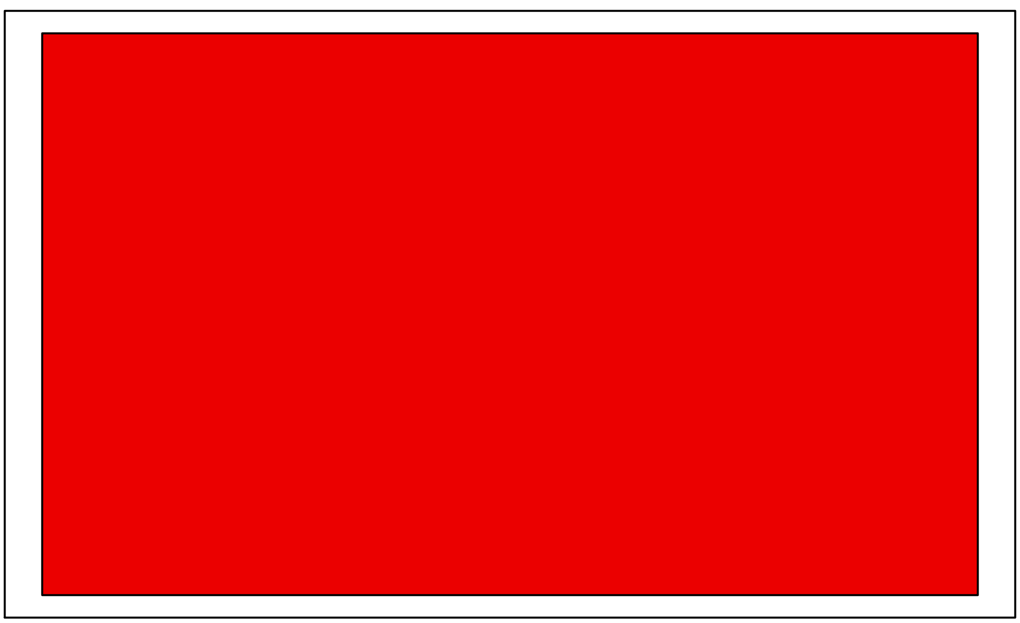
Show the Red, Green and Blue values for the pixel in row 106, and column 467:

```
color <- xmas[106,467,]
color
```

```
## [1] 0.9215686 0.0000000 0.0000000
```

It should be dark red. Let's Draw the color:

```
plot(NULL, xlim=c(0, 1), ylim=c(0, 1),axes=FALSE, frame.plot=TRUE, ann = FALSE)
rect(0, 0, 1, 1, col = rgb(color[1],color[2],color[3]))
```



1.2 From an RGB array to a data.frame

Create the dataframe with first 5 variables: row, column, red, green, and blue

```
index <- expand.grid(1:dim(xmas)[1], 1:dim(xmas)[2]) %>% as.matrix()
df_xmas <- matrix(rep(0,dim(xmas)[1] * dim(xmas)[2] * 5), ncol = 5) #dummy
```

```r
for (i in 1:nrow(index)){
  df_xmas[i,1] = index[i,1] #row
  df_xmas[i,2] = index[i,2] #col
  color = xmas[index[i,1],index[i,2],]
  df_xmas[i,3] = color[1] # red
  df_xmas[i,4] = color[2] #green
  df_xmas[i,5] = color[3] #blue
}
remove(i)
```

Create the 6th variable: rgb_color

```r
df_xmas <- as.data.frame(df_xmas)
names(df_xmas) <- c('row', 'col', 'red', 'green', 'blue')

df_xmas<- df_xmas %>%
  mutate(rgb_color = rgb(red,green,blue)) #rgb_color column

df_xmas$rgb_color =  as.factor(df_xmas$rgb_color) # convert to factor
```

Check:

```r
all.equal(df_xmas, xmas_df)
```

1.3 Number of Unique colors in xmas.png

```r
df_xmas$rgb_color %>% levels() %>% length()
```

```r
df_xmas %>%
  group_by(red,green,blue) %>%
  tally() %>%
  dim()
```

Both have 147839 unique values.

1.4 Creating A Raster To Plot the Picture in R

```r
raster_rgb <- df_xmas %>%
  select(row,col,rgb_color) %>%
  pivot_wider(names_from='col',values_from = 'rgb_color')  %>%
  subset(., select = -c(row) )  %>%
  as.matrix() %>%
  as.raster()
```

```r
my_pic_1 <- plot(raster_rgb)
```

2 A Further Understanding of the RGB Space

2.1 Hexadecimal identifiers for the RGB colors

```
hexadecimal <- c(0:9, LETTERS[1:6])
hdm2columns <- expand.grid(hexadecimal, hexadecimal)
channel <- paste0(hdm2columns[,2], hdm2columns[,1], sep = "")
```

Create the function:

```
get_rgb <- function(red = 0.5, green = 0.3, blue = 0.7, maxColorValue = 1.){
  # floor(x+0.5) for rounding; channel[x+1] for getting hexadecimal
  hex<- (c(red,green, blue)*(255/maxColorValue) + 0.5 + 1)  %>%
    floor(.)   %>%
    channel[.]  %>%
    paste(., collapse = '')
  return(paste0('#',hex))
}
get_rgb(maxColorValue = 1.0)
rgb(red = 0.5, green = 0.3, blue = 0.7, maxColorValue = 1.0)
```

2.2 Create Your Own Palette of RGB colors

```
get_palette <- function(K = 0, n_bit = 0){
  ### warnings
  try(if(K == 0 & n_bit == 0) stop("Type in K or n_bit"))
  try(if(K & n_bit) stop("Type in only K or n_bit"))
  try(if(as.integer(K)!=K | as.integer(n_bit)!= n_bit) stop("Wrong K or n_bit"))
  if(K){n_bit <- log2(K)} # using n_bit for this function
  try(if(n_bit %% 3 != 0) stop("Wrong K or n_bit"))

  ### start the real work
  n <- 2^(n_bit/3)
  col_list <- seq(0,255,length.out=n)
  dat <- expand.grid(Red = col_list, Green = col_list, Blue = col_list)
  cols <- dat %>%
    mutate(cols = rgb(Red,Green,Blue,maxColorValue=255)) %>%
    select(cols)
  ans <- list(cols = as.vector(t(cols)), dat = dat)
  return(ans)
}
```

```
My_RGB_03bit <- get_palette(n_bit=3)
My_RGB_03bit$dat
```

2.3 A Naive Approach to Color Reduction

2.3.a Compress thepicture into the colors from the 3-bit RGB palette

```
close_rgb <- function(red, green, blue, RGB=RGB_03bit) {
  ### calculate distance
  diff <- (c(red,green,blue)*255 + 0.5)  %>%
    floor(.) %>%
    sweep(RGB$dat, 2, .)  # minus by row
```

```r
  ### find the row in RGB$dat that gives min distance
  min_row <- diff^2  %>%
    apply(., MARGIN=1, sum) %>%
    which.min(.)

  ### get rgb_color
  new_rgb <- RGB$dat[min_row,]%>%
    mutate(rgb_color = rgb(Red, Green, Blue,max=255))
  return(new_rgb$rgb_color)
}
```

```r
new_raster<- df_xmas %>%
  rowwise() %>% # necessary for self-defined function
  mutate(new_rgb_color = close_rgb(red, green, blue)) %>%
  select(row,col,new_rgb_color) %>%
  pivot_wider(names_from='col',values_from = 'new_rgb_color')  %>%
  subset(., select = -c(row) )  %>%
  as.matrix() %>%
  as.raster()
```

Draw:

```r
plot(new_raster)
```

3 Floyd-Steinberg dithering algorithm

3.1 Programming your own Floyd-Steinberg algorithm

Create the update_rgb function

```
update_rgb <- function(x, RGB){
  min_row <- sweep(RGB$dat, 2, x)^2 %>% # minus by row
    apply(., 1, sum) %>%
    which.min(.)
  return(RGB$dat[min_row,])
}
```

Create the Floyd-Steinberg function:

Two input arguments: - an array the represents the pictur, like xmas - a matrix, like RGB_03bit$dat

The output : - an array of the colors with which the pixels should get replaced - an array that holds your estimates of the diffused errors for each pixel - the value of your loss function

```
Floyd_Steinberg <- function(pic, RGB){
  err <- pic # error matrix, update later
  nrow <- dim(pic)[1]
  ncol <- dim(pic)[2]
  pic <- floor(pic*255+0.5)
  for (r in 1:nrow){
     for (c in 1:ncol){
       x <- pic[r,c,]   #old pixel
       pic[r,c,] <- update_rgb(x,RGB) %>% unlist(.) %>% as.vector(.) #new pixel
       err[r,c,] <- x - pic[r,c,] #quant_error
       try(pic[r  ,c+1,] <- pic[r  ,c+1,] + err[r,c,] * 7 / 16) #ignore error
       try(pic[r+1,c-1,] <- pic[r+1,c-1,] + err[r,c,] * 3 / 16)
       try(pic[r+1,c  ,] <- pic[r+1,c  ,] + err[r,c,] * 5 / 16)
       try(pic[r+1,c+1,] <- pic[r+1,c+1,] + err[r,c,] * 1 / 16)
     }
  }
  pic = pic/255
  loss <- sum(err^2)
  return(list(img = pic, err_mat = err, loss = loss))
}
```

```
dither_ans_03 <- Floyd_Steinberg(xmas, RGB_03bit)
```

```
plot(NULL, xlim=c(0, 1), ylim=c(0, 1),axes=FALSE, frame.plot=TRUE, ann = FALSE)
grid::grid.raster(dither_ans_03$img)
```

3.2 Plotting the Loss for 3-bit, 9-bit, and 15-bit

```
los <- c(dither_ans_03$loss,sum(dither_09bit$err_mat^2),sum(dither_15bit$err_mat^2))
plot(c(3,9,15),log(los),type='l',xlab='n_bit')
points(c(3,9,15),log(los))
```

4 Statistical Computing on the Floyd-Steinberg algorithm

4.1 Generate a Permutation of the Picture

Create a function that produces a permuted replicate of the `xmas` variable, denoted by $\mathcal{X}^b$. Each pixel $\mathbf{x}_{ij}^b$ in $\mathcal{X}^b$ is a realization of a permutation over i and j of the pixels in $\mathcal{X}$.

```
create_a_perm <- function(xmas,B){
  set.seed(B)

  nrow <- dim(xmas)[1]
  ncol <- dim(xmas)[2]
  n <- nrow *  ncol
  ind <- sample(1:n, n, replace = FALSE)
  perm_pic <- xmas # make a copy, update it later

  for (i in 1:n){
    row_ind <- ceiling(ind[i]/ncol)
    col_ind <- ind[i] %% ncol + 1
    perm_pic[row_ind,col_ind,] <- xmas[ceiling(i/ncol),i %% ncol+1,]
  }
  return(perm_pic)
}
#perm_pic<-create_a_perm(xmas,2020)
#plot(NULL, xlim=c(0, 1), ylim=c(0, 1),axes=FALSE, frame.plot=TRUE, ann = FALSE)
#grid::grid.raster(perm_pic)
```

4.2 Log Loss of Floyd-Steinberg under H0

Write your own function that outputs a variable like xmas_replicates_logloss. input: - an array of an image (like xmas) - a vector of values for K_values that can only belong to the set 2^(3 * (1:5)) - B the number of replicates that need to be created.

```
create_perm_logloss <- function(xmas,bits,B){
  RGB <- parallel::mclapply(bits,get_palette) # Create palettes
  K_n <- length(bits)
  logloss_list <- parallel::mclapply(1:B, function(x) {
    perm_pic <- create_a_perm(xmas,x)
    logloss <- parallel::mclapply(1:K_n, function(y){
      ans <- Floyd_Steinberg(perm_pic, RGB[[y]])
      return(list(bit = 3*bits[y], logloss=ans$loss))
      })
    return(logloss)
    })
  return(logloss_list)
  }
```

```
### Test if the function works
B <- 2
bits <- 2^(3 * (1:5))
ans <- create_perm_logloss(xmas[1:10,1:10,],bits,B)
```

4.4 Visualize the Log Loss under H__0 and for our data

To get an estimate for the expected value of the loss function for each RGB palette under H0:

```
B <- length(xmas_replicates_logloss)
K_n <- length(xmas_replicates_logloss[[1]])
log_loss_k <- matrix(0,nrow=B,ncol=K_n)
for(i in 1:B){
  for(j in 1:K_n){
    log_loss_k[i,j]<- xmas_replicates_logloss[[i]][[j]]$logloss
  }
}

log_loss_perm <- colMeans(log_loss_k)
```

Compute the difference:

```
dither_list <- list(dither_03bit,dither_06bit,dither_09bit,dither_12bit,dither_15bit)
loglos_true <- sapply(dither_list, function(x) log(sum(x$err_mat^2)))

diff <- log_loss_perm - loglos_true
```

Best bit:

```
which.max(diff) %>% c(3,6,9,12,15)[.]
```

Create the data format for ggplot

```
df <- data.frame(bit = c(3,6,9,12,15), h_0 =log_loss_perm, observed = loglos_true)
df_longer <- pivot_longer(df,cols=c('h_0','observed'), names_to = 'type', values_to ='log_loss')

std <- sqrt((1+1/B)) * apply(log_loss_k, 2, sd)
df2<-data.frame(bit = c(3,6,9,12,15), gap = diff, std = std)
```

Draw:

```
p1<-ggplot(aes(x=bit  , y= log_loss) , data = df_longer) +
  geom_line(aes(color = type)) +
  geom_point(aes(fill=type), size=4, shape=21, color='transparent') +
  ggtitle('Loss: H0 and Observed Data') +
  scale_fill_manual(values=c('red','blue')) +
  scale_color_manual(values=c('red','blue')) +
  theme_bw()+
  theme(plot.title = element_text(hjust = 0.5,face='bold'))

p2<-ggplot(aes(x=bit  , y= gap) , data = df2) +
  geom_line() +
  geom_errorbar(aes(ymin=gap-2*std, ymax=gap+2*std), width=2.5, color = 'red', linetype=2) +
  geom_point(fill = 'black', size=4, shape=21) +
  ggtitle('Results Gap Statistic') +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5,face='bold'))+
  scale_x_continuous(limits=c(0, 18),breaks=seq(3,15,3))

gridExtra::grid.arrange(p1,p2,ncol=2)
```

4.6 Alternatives

Could you explain why the GAP statistic is small for too small K, and also small for too large K? For your explanation, relate to the bias variance trade-off.

Ans: - When K is small, there are not many color options. The RGB platte can only provide rough approximation of the original colors. This leads to high *bias*. Therefore, both permutation and original picture will have large but close log loss. Close log loss will give a small GAP statistic. - However, too many color options will cuase high *variancce* since a tiny change in the original color may lead to a different RGB platte color. Therefore, both permutation and original picture will have small but close log loss. Close log loss will give a small GAP statistic.

5. Bonus: Something new, the package `Rcpp` (15 points)

```
cppFunction('int***  Floyd_Steinberg_cpp(int*** pic,int height,int width) {
  int h = width, int w = width;
  for (int x = 1; x < h-1; x ++) {
      for (int y = 2; j < w; y++) {
          int oldPixel = pic[x][y];
          int newPixel = find_closest_palette_color(oldPixel);
          pic[x][y] = newPixel;
          int quant_error = oldPixel - newPixel;
          pic[x 1][y+1] = pic[x 1][y+1] + (quant_error * 7/16);
          pic[x+1][y-1] = pic[x+1][y-1] + (quant_error * 3/16);
          pic[x+1][y  ] = pic[x+1][y  ] + (quant_error * 5/16);
          pic[x+1][y+1] = pic[x+1][y+1] + (quant_error * 1/16);
      }
  }
  return pic;
}')


 new_pic <- Floyd_Steinberg_cpp(xmas*255,dim(xmas)[1],dim(xmas)[2])
```