DATA 605 : Assignment Week4

Ramnivas Singh

09/18/2021

Problem set 1 : With the attached data file, build and visualize eigenimagery that accounts for 80% of the variability.

```
# Load the library
library(jpeg)
library(EBImage)
library(doParallel)
library(foreach)
library(OpenImageR)
library(RSpectra)
```

Prepare for Image Processing

```
# File count 17
files=list.files("C:/Users/ramni/Desktop/CUNY/Data605/Week4",pattern="\\.jpg")[1:17]
# Print file list
files
```

```
##  [1] "RC_2500x1200_2014_us_53446.jpg" "RC_2500x1200_2014_us_53455.jpg"
##  [3] "RC_2500x1200_2014_us_53469.jpg" "RC_2500x1200_2014_us_53626.jpg"
##  [5] "RC_2500x1200_2014_us_53632.jpg" "RC_2500x1200_2014_us_53649.jpg"
##  [7] "RC_2500x1200_2014_us_53655.jpg" "RC_2500x1200_2014_us_53663.jpg"
##  [9] "RC_2500x1200_2014_us_53697.jpg" "RC_2500x1200_2014_us_54018.jpg"
## [11] "RC_2500x1200_2014_us_54067.jpg" "RC_2500x1200_2014_us_54106.jpg"
## [13] "RC_2500x1200_2014_us_54130.jpg" "RC_2500x1200_2014_us_54148.jpg"
## [15] "RC_2500x1200_2014_us_54157.jpg" "RC_2500x1200_2014_us_54165.jpg"
## [17] "RC_2500x1200_2014_us_54172.jpg"
```

View Shoes Function

as per Professor - Dr. Fulton

```
height=120;
width=250;
plot_jpeg = function(path, add=FALSE)
{ jpg = readJPEG(path, native=T) # read the file
  res = dim(jpg)[2:1] # get the resolution, [x, y]
  if (!add) # initialize an empty plot area if add==FALSE
    plot(1,1,xlim=c(1,res[1]),ylim=c(1,res[2]),asp=1,type='n',xaxs='i',yaxs='i',xaxt='n',yaxt=
         'n',xlab='',ylab='',bty='n')
    #rasterImage draws a raster image at the given locations and sizes.
    rasterImage(jpg,1,1,res[1],res[2])
}
```

Load image data into an Array

EBImage is used for image processing and analysis. Loading the data into the array with required dimensions. Resize function scales the images to the specified dimensions.

```
im=array(rep(0,length(files)*height*width*3), dim=c(length(files), height, width,3))
for (i in 1:length(files)){
  temp= EBImage::resize(readJPEG(files[i]),120, 250)
  im[i,,,]=array(temp,dim=c(1, 120, 250,3))
}
# Display dimension of image array
dim(im)
```

```
## [1]  17 120 250   3
```

Vectorize image data

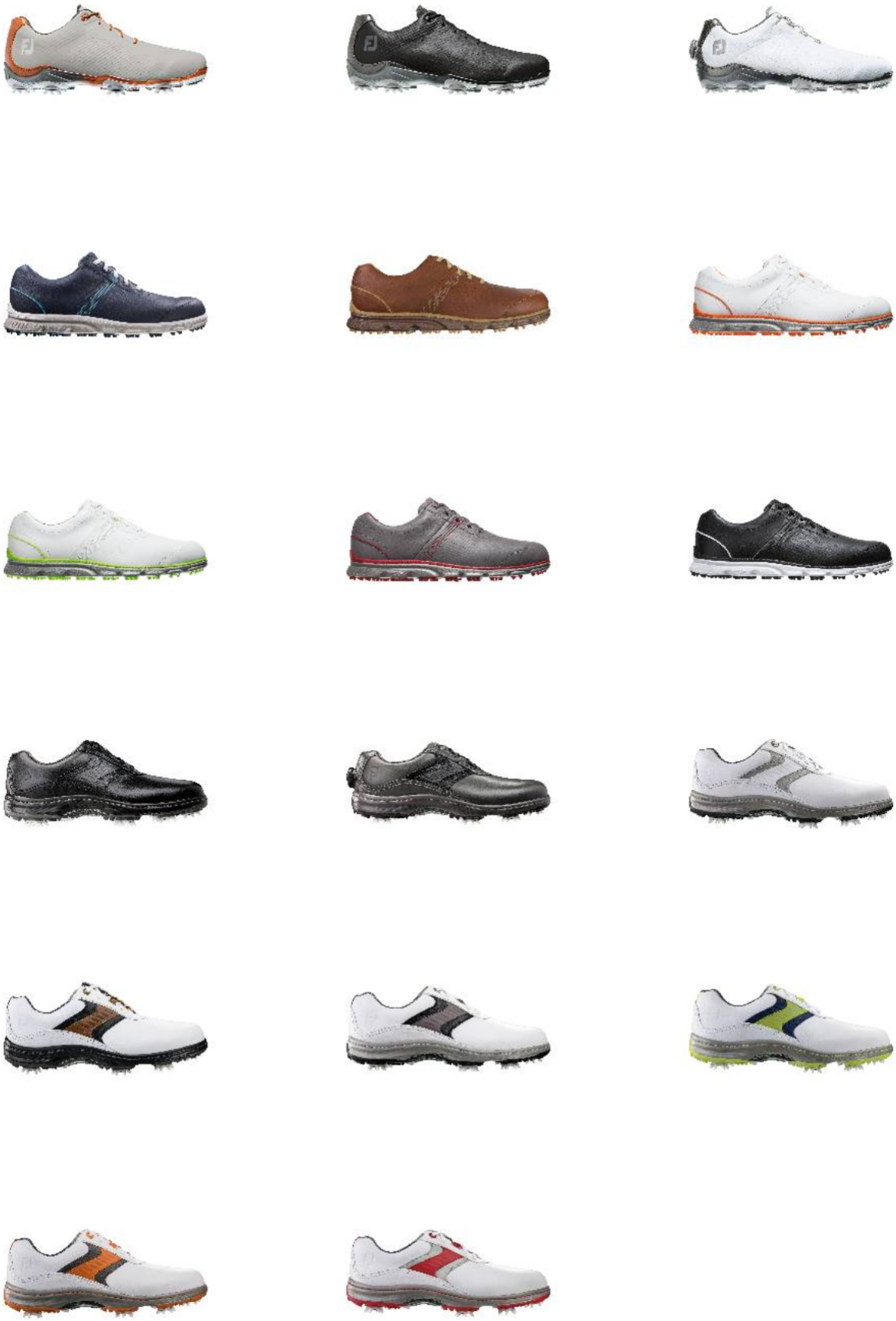
Here we vecorize the array. Matri is created for image data

```
flat=matrix(0, length(files), prod(dim(im)))
for (i in 1:length(files)){
  r=as.vector(im[i,,,1]); g=as.vector(im[i,,,2]);b=as.vector(im[i,,,3])
  flat[i,] <- t(c(r, g, b))
}
shoes=as.data.frame(t(flat))
```

Actual Plots

Render the actual plots. In viewing the images we see that there is quite a bit of fine detail that is tripping it up.

```
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:length(files)){
plot_jpeg(writeJPEG(im[i,,,]))
}
```



Blurring the images might allow a certain refinement.

Covariance and Correlation

Compute the covariance or correlation shoes vector.

```
scaled=scale(shoes, center = TRUE, scale = TRUE)
Sigma_=cor(scaled)
```

Eigenvalues and Eigenvectors

Computes a limited number of eigenvalues and eigenvectors of a matrix. Defaults to “LM”, meaning largest magnitude eigenvalues

```
myeigen=eigs(Sigma_,5,which="LM")
myeigen
```

```
## $values
## [1] 11.7577248  1.6919079  0.9027620  0.4615921  0.3226809
##
## $vectors
##             [,1]        [,2]        [,3]          [,4]        [,5]
##  [1,] -0.2509682  0.05957959 -0.13164043 -0.3829317674  0.30363261
##  [2,] -0.2567363 -0.22945730 -0.09408644 -0.2280539883 -0.04191297
##  [3,] -0.1976323  0.34151792 -0.25573568 -0.6550797488 -0.21408440
##  [4,] -0.2387600 -0.30604918 -0.13860421  0.1317996954 -0.35610013
##  [5,] -0.2533740 -0.24091421 -0.04656705 -0.0007298772  0.32292843
##  [6,] -0.2093423  0.34242619 -0.43083781  0.3269636791  0.03911544
##  [7,] -0.2234912  0.32060796 -0.35937581  0.2583133863  0.10157280
##  [8,] -0.2588356 -0.13442559 -0.28248003  0.1675860269 -0.20507974
##  [9,] -0.2256671 -0.39082581 -0.17492837  0.1596693318 -0.21958539
## [10,] -0.2524909 -0.26772432  0.02895504 -0.0954431720  0.31240957
## [11,] -0.2494301 -0.24543180  0.14446510 -0.0811180944  0.24621644
## [12,] -0.2546833  0.16187856  0.16315929 -0.1123833163 -0.19441641
## [13,] -0.2378155  0.25323691  0.18860636  0.2260163213  0.32469369
## [14,] -0.2431959  0.17804530  0.34712385  0.1309406905 -0.27054930
## [15,] -0.2529605  0.06155803  0.32061584  0.0034921602 -0.33520197
## [16,] -0.2562522  0.12211470  0.25450591  0.1715234606  0.19132981
## [17,] -0.2509163  0.10932046  0.29984410 -0.0407695534 -0.05827053
##
## $nconv
## [1] 5
##
## $niter
## [1] 1
##
## $nops
## [1] 17
```

```
cumsum(myeigen$values) / sum(eigen(Sigma_)$values)
```

```
## [1] 0.6916309 0.7911549 0.8442585 0.8714110 0.8903922
```

We see that effacing the small details has of course improved the variability accounted for. These images seem a more reflective of the actual differences.

Matrix Diagonals

Extract or replace the diagonal of a matrix, or construct a diagonal matrix.

```
scaling=diag(myeigen$values[1:5]^(-1/2)) / (sqrt(nrow(scaled)-1))
eigenshoes=scaled%*%myeigen$vectors[,1:5]%*%scaling
```

Principal Components Analysis

princomp performs a principal components analysis on the given numeric data matrix and returns the results as an object of class princomp.

```
newdata=im
dim(newdata)=c(length(files),height*width*3)
mypca=princomp(t(as.matrix(newdata)), scores=TRUE, cor=TRUE)
```

Eigenshoes (2)

Generate Eigenshoes based on principal components score

```
pcaScores=t(mypca$scores)
dim(pcaScores)=c(length(files),height,width,3)
par(mfrow=c(5,5))
par(mai=c(.001,.001,.001,.001))
for (i in 1:length(files)){
plot_jpeg(writeJPEG(pcaScores[i,,,], bg="white"))
}
```

```
for (i in 1:length(files)){
  temp= EBImage::gblur(EBImage::resize(readJPEG(files[i]),120, 250),sigma = 8, radius=5)
  im[i,,,]=array(temp,dim=c(1, 120, 250,3))
}
flat=matrix(0, length(files), prod(dim(im)))
for (i in 1:length(files)){
  r=as.vector(im[i,,,1]); g=as.vector(im[i,,,2]);b=as.vector(im[i,,,3])
  flat[i,] <- t(c(r, g, b))
}
shoes=as.data.frame(t(flat))
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:length(files)){
plot_jpeg(writeJPEG(im[i,,,]))
}
```

















Variance Capture

```
rpca=round(mypca$sdev[1:17]^2/ sum(mypca$sdev^2),3)
cumsum(rpca)
```

```
##  Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8  Comp.9 Comp.10
##   0.692   0.792   0.845   0.872   0.891   0.908   0.922   0.934   0.944   0.953
## Comp.11 Comp.12 Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
##   0.961   0.969   0.977   0.984   0.990   0.996   1.001
```

Eigenimagery that accounts for 80 percent of the variability

```
PVE <- myeigen$values / sum(myeigen$values)
round(PVE, 3)
```

```
## [1] 0.777 0.112 0.060 0.030 0.021
```

The first principal component explains 69 percent of the variability, and the second principal component explains 10 percent. Together, the first two principal components explain 79 percent of the variability. In order to cover our 80 percent, we would need to take into account the third component for a total of 84 percent variability.