HW4 Example

FUNDAMENTALS OF COMPUTATIONAL MATHEMATICS

Zachary Palmore

2/12/2021

Directions

With the attached data file, build and visualize eigenimagery that accounts for 80% of the variability. Provide full R code and discussion.

Example

Taken from https://rpubs.com/R-Minator/eigenshoes

```
# Ensure an updated version of R before installing
install.packages("installr")
library(installr)
updateR() # As of 2/15/2021 R version is 4.0.4
install.packages('doParallel')
install.packages('foreach')
install.packages('jpeg')
if (!requireNamespace("BiocManager", quietly = TRUE)) # Required to install EMImage from Bioconductor
    install.packages("BiocManager")
BiocManager::install('EBImage')
install.packages('kableExtra')
install.packages('OpenImageR')
```

Loading the packages used:

```
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(foreach)
library(jpeg)
library(EBImage)
library(kableExtra)
library(OpenImageR)
```

```
##
## Attaching package: 'OpenImageR'

## The following objects are masked from 'package:EBImage':
##
##     readImage, writeImage
```

Use of Graphics

Here, we add graphics to the data set.

```
#############Prepare for Image Processing#######################
num=17 # specify the number of images present in the folder
files=list.files("C:/data/images/jpg",pattern="\\.jpg")[1:num]
################################################################
```

Test reading of the files prior to loading them into an array. Ensure that they are read properly as values with a num [1:60, 1:125, 1:3] 1, 1, 1, 1, 1, ... etc within the R environment.

```
file1 <- resize(readJPEG(paste0("C:/data/images/jpg/", files[1])), height/scale, width/scale)
file2 <- resize(readJPEG(paste0("C:/data/images/jpg/", files[2])), height/scale, width/scale)
file17 <- resize(readJPEG(paste0("C:/data/images/jpg/", files[17])), height/scale, width/scale)
```

View Shoes Function

This function is given as something we can copy directly. However, for it to set the parameters for the image array correctly, ensure that the dimensions of the empty array match that of the images being rastered.

```
###################Set Adj. Parameters############################
height=1200; width=2500;scale=20
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=
  rasterImage(jpg,1,1,res[1],res[2])
}
################################################################
```

Load the Data into an Array

This create the empty array. Dimensions are supposed to be extracted from the images themselves with the length of the files. If this is not the case, skip ahead to vectorize and actual plots. This will display the correct dimensions to set the empty array.

```
###################Load#############################
im=array(rep(0,length(files)*height/scale*width/scale*3), dim=c(length(files), height/scale, width/scal
```

Adjust the loop to match the number of images present in the file. For him it was 20. For us it is different (most likely 17 for you)

```
for (i in 1:17){ # specify total number of files
  temp=resize(readJPEG(paste0("C:/data/images/jpg/", files[i])), height/scale, width/scale) # Extracts
  im[i,,,]=array(temp,dim=c(1, height/scale, width/scale,3))} # Adds the image values to the empty arra
################################################
```

Vectorize

Again, be sure to update the matrix and loop with the same number of files present in the folder otherwise it will fail. The error will not be much help.

```
################################################
flat=matrix(0, 17, prod(dim(im))) # adjust to number of files present in folder
for (i in 1:17) { # adjust to number of files present in folder
  newim <- readJPEG(paste0("C:/data/images/jpg/", files[i])) # ensure all file paths are the same
  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

```
####Old Shoes##################
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:17){  # adjust to number of files present in folder
plot_jpeg(writeJPEG(im[i,,,])) # view shoes function applied here; plots the raster data
}
```









Thus, we can now begin the PCA and its evaluation.

```
##################################################
scaled=scale(shoes, center = TRUE, scale = TRUE)
mean.shoe=attr(scaled, "scaled:center") #saving for classification
std.shoe=attr(scaled, "scaled:scale")  #saving for classification...later
##################################################
```

Calculate Covariance (Correlation)

```
##################################################
Sigma_=cor(scaled)
##################################################
```

Get the Eigencomponents

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

```
##  [1] 0.6928202 0.7940449 0.8451072 0.8723847 0.8913841 0.9076337 0.9216282
##  [8] 0.9336889 0.9433871 0.9524454 0.9609037 0.9688907 0.9765235 0.9832209
## [15] 0.9894033 0.9953587 1.0000000
```

```
##################################################
```

Eigenshoes

```
##################################################
scaling=diag(myeigen$values[1:5]^(-1/2)) / (sqrt(nrow(scaled)-1))
eigenshoes=scaled%*%myeigen$vectors[,1:5]%*%scaling
imageShow(array(eigenshoes[,1], c(60,125,3)))
```

```
##################################################
```

Generate Principal Components

Transform the images

The correlation in the PCA variable *mypca* should remain TRUE. If it does not, review the empty (but now filled) matrix dimensions and its eigenvalues. Adjust to fit the TRUE otherwise the view shoes functions will have an error.

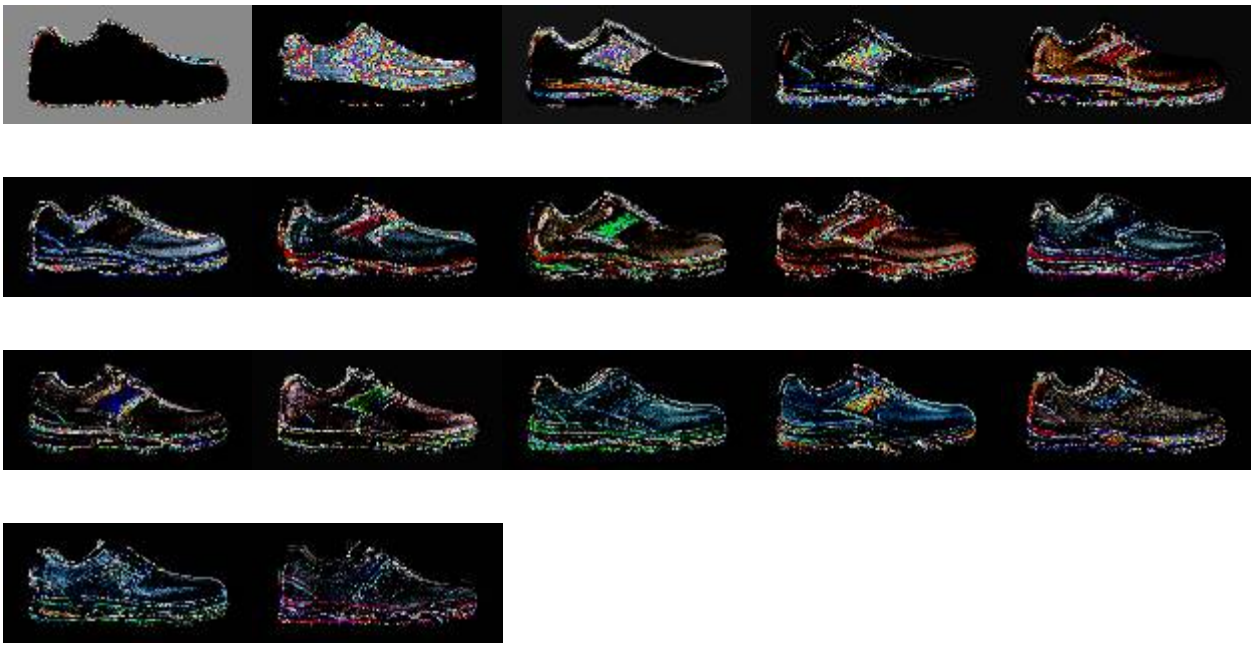
```
###################Generate Variables#############################
height=1200
width=2500
scale=20
newdata=im
dim(newdata)=c(length(files),height*width*3/scale^2)
mypca=princomp(t(as.matrix(newdata)), scores=TRUE, cor=TRUE) # should always be TRUE
##################################################################
```

Eigenshoes

Generate Eigenshoes.

Note that there is no inversion of the PCA results. Although it makes less logical sense to have negative values in the raster image, it should appear

```
##################Eigenshoes#######################################
mypca2=t(mypca$scores)
dim(mypca2)=c(length(files),height/scale,width/scale,3)
par(mfrow=c(5,5))
par(mai=c(.001,.001,.001,.001))
for (i in 1:17){ # adjust to the number of files present in folder
plot_jpeg(writeJPEG(mypca2[i,,,], bg="white"))  #complete without reduction
}
###################################################################
```



From here, we evaluate how well the PCA captured the results compared to the 'actual' images.

Variance capture

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

```
##  Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8  Comp.9 Comp.10
##   0.693   0.794   0.845   0.872   0.891   0.907   0.921   0.933   0.943   0.952
```

```
## Comp.11 Comp.12 Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
##   0.960   0.968   0.976   0.983   0.989   0.995   1.000
```

```
################################################
```

New Data Set

```
################################################
x = t(t(eigenshoes)%*%scaled)
################################################
```