DATA 605: Assignment 04

EigenShoes

Philip Tanofsky

21 February 2021

EigenShoes

Starting point: https://rpubs.com/R-Minator/eigenshoes

```
# Libraries
library(jpeg)
library(EBImage)
library(OpenImageR)
```

Use of Graphics

Add graphics to the data set.

```
# Prepare for Image Processing

# Set constant for number of images to be processed
num_of_images <- 17

# Directory containing the images
img_dir <- "/Users/philiptanofsky/Documents/School/CUNY/MSDS/Courses/DATA605/Week04/jpg/"

# Read in the list of file names ending in ".jpg" from the directory
# These are images provided by the professor
files <- list.files(img_dir, pattern="\\.jpg")[1:num_of_images]

#files: Contains the list of jpg file names
```

View Shoes Function

```
# Set Adjustment Parameters (default values)
height <- 1200
width <- 2500
scale <- 20
colors_cnt <- 3 # Constant represents R,G,B colors of the jpg images

plot_jpeg <- function(path, add=FALSE) {
```

```r
  # Read the file
  jpg <- readJPEG(path, native=T)
  # Get the resolution, [x, y]
  res <- dim(jpg)[2:1]
  # Initialize any empty plot are if add == False
  if (!add) {
    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(jpg,
                1,
                1,
                res[1],
                res[2])
  }
}
```

Load the Data into an Array

```r
# Based on the default values above
# Creating an arry with dimensions[17, 60, 125, 3] with all zeroes to start
img_arr <- array(rep(0, length(files) * height/scale * width/scale * colors_cnt),
            dim=c(length(files), height/scale, width/scale, colors_cnt))

# Loop through the image paths
for (i in 1:num_of_images) {
  temp <- resize(readJPEG(paste0(img_dir, files[i])),
                 height/scale,
                 width/scale)
  # Load the scaled images into the image array declared above
  img_arr[i,,,] <- array(temp, dim=c(1, height/scale, width/scale, colors_cnt))
}
```

Vectorize

```r
# Create matrix of image count rows (17) and image array dimensions as columns (382500)
flat <- matrix(0, num_of_images, prod(dim(img_arr)))

# Loop through images
```

```r
for (i in 1:num_of_images) {
  #newim <- readJPEG(paste0(img_dir, files[i])) # Extra line of code
  # From image array, pull vector for color Red for given image
  r <- as.vector(img_arr[i,,,1])
  # From image array, pull vector for color Green for given image
  g <- as.vector(img_arr[i,,,2])
  # From image array, pull vector for color Blue for given image
  b <- as.vector(img_arr[i,,,3])
  # Take transpose of the color vectors to make it fit the row length of flat matrix
  flat[i,] <- t(c(r, g, b))
}
# Take transpose of the flat matrix
shoes <- as.data.frame(t(flat))

#dim(shoes)
#382500     17
# As expected, now >300k rows with 17 columns
```

Actual Plots

```r
# Plot the original images of the Shoes
par(mfrow=c(3,3))
# mai: set margin in inches
par(mai=c(.3, .3, .3, .3))
# Loop through image array and call the plot_jpeg function
# The image array contains the initial read in of the jpg files
for (i in 1:num_of_images) {
  plot_jpeg(writeJPEG(img_arr[i,,,]))
}
```









Get Eigen components from correlation structure

```
scaled <- scale(shoes, center=TRUE, scale=TRUE)
#scaled object contains rows:382500     cols:17
# Clever way to pull the mean of each column using the scale function
#mean.shoe <- attr(scaled, "scaled:center") # saving for classification
# Clever way to pull the standard deviation of each column using the scale function
#std.shoe <- attr(scaled, "scaled:scale") # saving for classification
```

Calculate Covariance (Correlation)

```
# Create correlation matrix based on the number of images, in this case 17x17
sigma_ <- cor(scaled)

#sigma_
```

Get the eigencomponents

```r
# Compute the eigenvalues and eigenvectors based on the correlation matrix
myeigen <- eigen(sigma_)
cumsum(myeigen$values) / sum(myeigen$values)
```

```
##  [1] 0.6928202 0.7940449 0.8451073 0.8723847 0.8913841 0.9076338 0.9216282
##  [8] 0.9336889 0.9433872 0.9524455 0.9609037 0.9688907 0.9765235 0.9832209
## [15] 0.9894033 0.9953587 1.0000000
```

Eigen shoes

```r
# Why was 5 selected?
pca_comps <- 3
scaling <- diag(myeigen$values[1:pca_comps]^(-1/2)) / (sqrt(nrow(scaled)-1))
scaling
```

```
##              [,1]        [,2]        [,3]
## [1,] 0.0004711401 0.000000000 0.000000000
## [2,] 0.0000000000 0.001232586 0.000000000
## [3,] 0.0000000000 0.000000000 0.001735441
```

```r
myeigen$vectors[,1:pca_comps]
```

```
##             [,1]        [,2]        [,3]
##  [1,] -0.2515577 -0.05962807 -0.14114605
##  [2,] -0.2564669  0.22970932 -0.09482706
##  [3,] -0.1974907 -0.34526438 -0.24576573
##  [4,] -0.2391458  0.30516320 -0.13606194
##  [5,] -0.2525203  0.23895414 -0.06096558
##  [6,] -0.2096918 -0.34776361 -0.42324640
##  [7,] -0.2220439 -0.32176935 -0.36923615
##  [8,] -0.2597468  0.13861061 -0.27362524
##  [9,] -0.2242754  0.39008169 -0.17677165
## [10,] -0.2523894  0.26939880  0.02645111
## [11,] -0.2504276  0.23813195  0.14578328
## [12,] -0.2541524 -0.16064493  0.16973475
## [13,] -0.2374627 -0.25443032  0.18739393
## [14,] -0.2431988 -0.17131145  0.34992145
## [15,] -0.2531910 -0.06188346  0.32463869
## [16,] -0.2571186 -0.11980858  0.24383184
## [17,] -0.2513643 -0.10507094  0.30609685
```

```r
eigenshoes <- scaled %*% myeigen$vectors[,1:pca_comps] %*% scaling
imageShow(array(eigenshoes[,1], c(height/scale, width/scale, colors_cnt)))
```

```
# Dupe from above with different column count ... remove as needed
scaling <- diag(myeigen$values[1:17]^(-1/2)) / (sqrt(nrow(scaled)-1))
eigenshoes <- scaled %*% myeigen$vectors[,1:17] %*% scaling
imageShow(array(eigenshoes[,1], c(height/scale, width/scale, colors_cnt)))
```

Generate Principal Components

Transform the images

```
# Generate variables
height <- 1200
width <- 2500
scale <- 20
# Start with the image array of the original images
newdata <- img_arr

# Convert the dimensions of the array to n x m, instead of n x m x o x p
dim(newdata) <- c(length(files), height*width*colors_cnt/scale^2)


# Transpose the n x m array, and then calculate the principal components
# By transposing the array, now there are only 17 columns, and thus will produce 17 principal component
mypca <- princomp(t(as.matrix(newdata)), scores=TRUE, cor=TRUE)
# contains 17 components
```

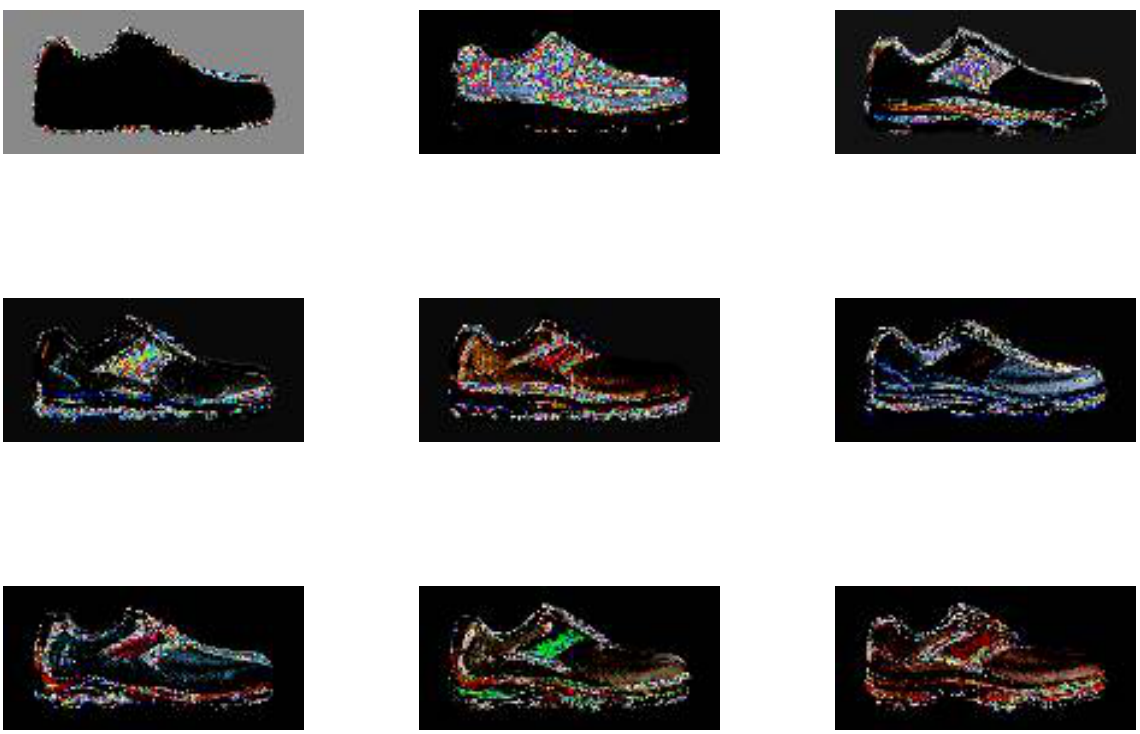
```
#mypca

#dim(mypca$scores)
#22500    17
```

Eigenshoes

Generate Eigenshoes

```
# Transpose the Principal Component Analysis scores back to original matrix dimensions
mypca2 <- t(mypca$scores)
# Reset the dimensions of the the PCA object to fit the initial array dimensions
dim(mypca2) <- c(length(files), height/scale, width/scale, colors_cnt)
#par(mfrow=c(5,5))
#par(mai=c(.001, .001, .001, .001))
par(mfrow=c(3,3))
# mai: set margin in inches
par(mai=c(.3, .3, .3, .3))
# Plot the eigenshoes only based on the PCA scores
for (i in 1:num_of_images) {
  plot_jpeg(writeJPEG(mypca2[i,,,], bg="white")) # Complete without reduction
}
```



Variance Capture

```
num_comps <- 17
a <- round(mypca$sdev[1:num_comps]^2 / sum(mypca$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
```

To account for 80% of the variability, the first 3 components are required to exceed the 80% threshold. As displayed above, the first component accounts for 69.3% and the first two combined accounts for 79.4% of the variability. Thus to reach at least 80% variability, the initial three components are required (84.5% variability accounted for).

From: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

For PCA, standardization is valuable. Value - Mean / Standard Deviation, thus all variables will be on the same scale.

then covariance matrix computation: because some variables are highly correlated in such a way that they contain redundant information covariance matrix is not more than a table that summaries the correlations between all the possible pairs of variables.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components.

Percentage of explained variance decreases with the principal components starting with 1 and going to n

The first principal component accounts for the largest possible variance in the data set.

The eigenvectors of the Covariance matrix are actually the directions of the axes where there is the most variance (most information)