Principal Components Analysis (PCA)

Kenan Sooklall

2/10/2021

The goal of this homework is to show how PCA can take advantage of the linear combination between pixel values to create a lower dimensional representation.

Set up global variables

```
library(EBImage)
library(OpenImageR)
```

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

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

```
library(jpeg)

path <- "/home/kenan/Documents/learning/masters/CUNY-SPS-Masters-DS/DATA_605/homeworks/homework4/images/

files=list.files(path, pattern="*.jpg")
n=length(files)

# shoes dim
height=1200
width=2500
scale=20

# Pokemon dim
#height=224
#width=224
#scale=2
```

Function to read in images from path and plot them

```
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])
}
```

Resize each image and populate vector im with all images $dim(im) = (17, 60, 125, 3)$ where 17 is n the number of images and (60,125,3) are each image dimensions

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

for (i in 1:n){
  im[i,,,]=resizeImage(readImage(paste0(path, files[i])),width=width/scale, height=height/scale)
  }
```

Plot first 9 images

```
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))

for (i in 1:9){
  plot_jpeg(writeJPEG(im[i,,,]))
}
```

To begin PCA each image is flatten into a matrix where each column is one image $mshoes$ will be the matrix of $dim(im) = 22500x17$

```
flat=matrix(0, n, prod(dim(im)[2:4]))
for (i in 1:n) {
  newim <- readImage(paste0(path, files[i]))
  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))
}
mshoes=t(flat)
```

The principal components in PCA are the eigen vectors of the covariance matrix

We compute the covariance matrix by first standardizing each image by $(X - \mu)/\sigma$ Once standardized each image will have a $\mu = 0$
$\sigma = 1$

With the standarized values we compute the covariance matrix to get the linear combination between each shoe, the shape of the covariance matrix is 17x17

```
scaled=scale(mshoes, center = TRUE, scale = TRUE)
means=attr(scaled, "scaled:center") #saving for classification
std=attr(scaled, "scaled:scale")
cshoes=cov(scaled)
```

With the covariance matrix the eigen values are corresponding eigen vectors can be computed

```
edf=eigen(cshoes)
```

The eigen values tell you how much variance can be captured with the *ith* eigen values

```
lambda_dist<-as.data.frame(t(cumsum(edf$values) / sum(edf$values)))
lambda_dist
```

```
##          V1        V2       V3        V4        V5        V6        V7
## 1 0.6835578 0.7832201 0.835167 0.8631119 0.8826992 0.9000067 0.9150577
##          V8        V9       V10       V11       V12       V13       V14
## 1 0.9278809 0.9379629 0.9478206 0.9569254 0.9655323 0.9738415 0.9810516
##         V15       V16 V17
## 1 0.9880682 0.9945565   1
```

It looks like the first 3 eigen values can be used to represent more than 80% of the data with 6 we are at 90% and by 11 it's 95% of the pixel variance.

Now to visualize what an image looks like with only 80%

```
scaling=diag(edf$values[1:3]^(-1/2)) / (sqrt(nrow(scaled)-1))
eigenshoes=scaled%*%edf$vectors[,1:3]%*%scaling
imageShow(array(eigenshoes[,1], c(width/scale,height/scale,3)))
```