# EBC4223 - -Assignment 3

Quang Phong - Robert Agatić

3/7/2022

## INTRODUCTION

In this assignment, the data is processed into a digital format, and the salient facial information is extracted using the Eigenface method (Xia & Ding 2014). Then, an image detection algorithm will be implemented.

```
library(pixmap) #for import, export, plotting and other manipulations of bitmapped images
```

### Question 1: Load data

The data can be imported using the pixmap library and the function pnm(), and stored as a list. Each element of the list represents one image. Using plot() on an element of the list, you can view the image.

```
# Produce a character vector of the names of files in the named directory
allFiles <- list.files(path = "faces", pattern = ".pgm", full.names = T)

# Choose a random subset of 20 files in the list
set.seed(1010)
selectedFiles <- sample(allFiles, 20)

# Look at 20 random files
selectedFiles
```

```
##  [1] "faces/Emma_Watson_0004.pgm"
##  [2] "faces/Tony_Blair_0126.pgm"
##  [3] "faces/Thomas_Scavone_0001.pgm"
##  [4] "faces/Sophia_Loren_0007.pgm"
##  [5] "faces/Richard_Armitage_0007.pgm"
##  [6] "faces/Gloria_Macapagal_Arroyo_0027.pgm"
##  [7] "faces/Richard_Myers_0014.pgm"
##  [8] "faces/Michael_Guiler_0001.pgm"
##  [9] "faces/Dwain_Kyles_0001.pgm"
## [10] "faces/Rick_Stansbury_0001.pgm"
## [11] "faces/Hillary_Clinton_0009.pgm"
## [12] "faces/Donna_Shalala_0002.pgm"
## [13] "faces/Andrew_Niccol_0001.pgm"
## [14] "faces/Holly_Hunter_0005.pgm"
## [15] "faces/Carlos_Moya_0010.pgm"
## [16] "faces/Paul_Burrell_0001.pgm"
## [17] "faces/Delphine_Chuillot_0001.pgm"
## [18] "faces/Frank_Dunham_Jr_0002.pgm"
## [19] "faces/Sanja_Papic_0001.pgm"
## [20] "faces/Grace_Brinell_0001.pgm"
```

```r
# Import 20 images
pictures <- lapply(selectedFiles, FUN = function(f) read.pnm(f))

# View the images
for (x in pictures) {
  plot(x)
}
```

# DATA PREPROCESSING AND EXPLORATION

**Question 2: See pixel representation**

Use the @grey attribute of the pixmap image to see the pixel representation of the image. Note that each image is 64x64 pixels in size.

```
# See dimension
dim(pictures[[1]]@grey)
```

```
## [1] 64 64
```

```
# Look at 10x10 first pixels
pictures[[1]]@grey[1:10, 1:10]
```

```
##             [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,] 0.4352941 0.5411765 0.6313725 0.6823529 0.7058824 0.7254902 0.7333333
##  [2,] 0.4352941 0.5254902 0.6274510 0.6823529 0.7058824 0.7254902 0.7333333
##  [3,] 0.4431373 0.5137255 0.6117647 0.6745098 0.7019608 0.7254902 0.7333333
##  [4,] 0.4588235 0.5294118 0.6039216 0.6627451 0.6941176 0.7294118 0.7411765
##  [5,] 0.4823529 0.5490196 0.6039216 0.6549020 0.6823529 0.7215686 0.7372549
##  [6,] 0.4862745 0.5529412 0.5921569 0.6392157 0.6745098 0.7137255 0.7333333
##  [7,] 0.4666667 0.5529412 0.5882353 0.6313725 0.6745098 0.7137255 0.7333333
##  [8,] 0.4392157 0.5490196 0.5921569 0.6352941 0.6745098 0.7137255 0.7333333
##  [9,] 0.4274510 0.5490196 0.6000000 0.6352941 0.6705882 0.7098039 0.7294118
## [10,] 0.4196078 0.5529412 0.6117647 0.6352941 0.6549020 0.6980392 0.7333333
##             [,8]      [,9]     [,10]
##  [1,] 0.7450980 0.7607843 0.7686275
##  [2,] 0.7411765 0.7568627 0.7647059
##  [3,] 0.7450980 0.7568627 0.7686275
##  [4,] 0.7568627 0.7686275 0.7725490
##  [5,] 0.7568627 0.7764706 0.7803922
##  [6,] 0.7607843 0.7803922 0.7843137
##  [7,] 0.7529412 0.7764706 0.7803922
##  [8,] 0.7529412 0.7725490 0.7725490
##  [9,] 0.7529412 0.7725490 0.7725490
## [10,] 0.7490196 0.7529412 0.7490196
```

**Question 3: compute the average face**

Create a matrix where each column represents a pixel, and each row an image. By applying colMeans() to this matrix, we can compute the "average" face. Plot this face using the image() (hint: Make sure to convert the mean vector into a 64x64 matrix again, and use byrow = TRUE).

```r
# Create empty list
empty_list <- vector(mode="list", length=20)

# Add all pixels of an image to each row
for (i in 1:20) {
  empty_list[[i]] <- as.vector(pictures[[i]]@grey)
}

# Turn the list into the matrix
matrix1 <- matrix(unlist(empty_list), byrow=TRUE, nrow = 20)
```

```r
# Compute the average face by taking the average of each column
meanVector <- colMeans(matrix1)
length(meanVector)
```

```
## [1] 4096
```

```r
meanVector[1:10]
```

```
##  [1] 0.3560784 0.3558824 0.3594118 0.3623529 0.3654902 0.3656863 0.3686275
##  [8] 0.3709804 0.3739216 0.3774510
```

```r
# Convert the mean vector into a 64x64 matrix again
averageFace <- matrix(meanVector,  byrow=TRUE, nrow=64)

# Plot this average face
image(averageFace, col=grey(seq(0,1,length=256)))
```


```r
# Plot this average face upside down
image(averageFace[, nrow(averageFace):1],
      col=grey(seq(0,1,length=256)))
```


**Question 4: Reduce the large set of image files**

Apply principal components analysis (PCA) to the matrix with all images using the prcomp() PCA reduces the large set of image files to a smaller set of vectors called principal components. This makes analysis a lot easier.

```
pca <- prcomp(matrix1, scale = TRUE)
names(pca)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

**Question 5: Determine the number of principal components to retain**

Determine an appropriate number of principal components to retain, using the results from the summary of the prcomp Select the number of components such that 90% of the cumulative variance is explained.

```r
summary(pca)
```

```
## Importance of components:
##                            PC1     PC2      PC3      PC4     PC5     PC6
## Standard deviation     39.348 24.9289 18.36715 16.59909 14.2827 12.4757
## Proportion of Variance  0.378  0.1517  0.08236  0.06727  0.0498  0.0380
## Cumulative Proportion   0.378  0.5297  0.61208  0.67935  0.7292  0.7671
##                             PC7      PC8      PC9    PC10    PC11    PC12
## Standard deviation     12.17103 11.54650 10.45961 9.63680 9.37874 8.30415
## Proportion of Variance  0.03617  0.03255  0.02671 0.02267 0.02147 0.01684
## Cumulative Proportion   0.80332  0.83587  0.86258 0.88525 0.90673 0.92356
##                           PC13    PC14    PC15    PC16    PC17    PC18    PC19
## Standard deviation     7.77328 7.34357 7.19188 6.58530 6.47527 6.05804 5.00183
## Proportion of Variance 0.01475 0.01317 0.01263 0.01059 0.01024 0.00896 0.00611
## Cumulative Proportion  0.93831 0.95148 0.96411 0.97470 0.98493 0.99389 1.00000
##                            PC20
## Standard deviation     3.015e-14
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

We will retain 11 principal components (from a total of 20).

**Question 6: Plot eigenfaces**

The eigenvectors related to these principal components represent the eigenfaces. They are stored in the $rotation attribute of the prcomp object, each column representing an eigenvector. Plot the first 10 eigenfaces using the image() function, and describe what you see.

```r
for (y in 1:10) {
  a <- matrix(pca$rotation[,y],  byrow=TRUE, nrow=64)
  image(a[, nrow(a):1], col=grey(seq(0,1,length=256)))
}
```

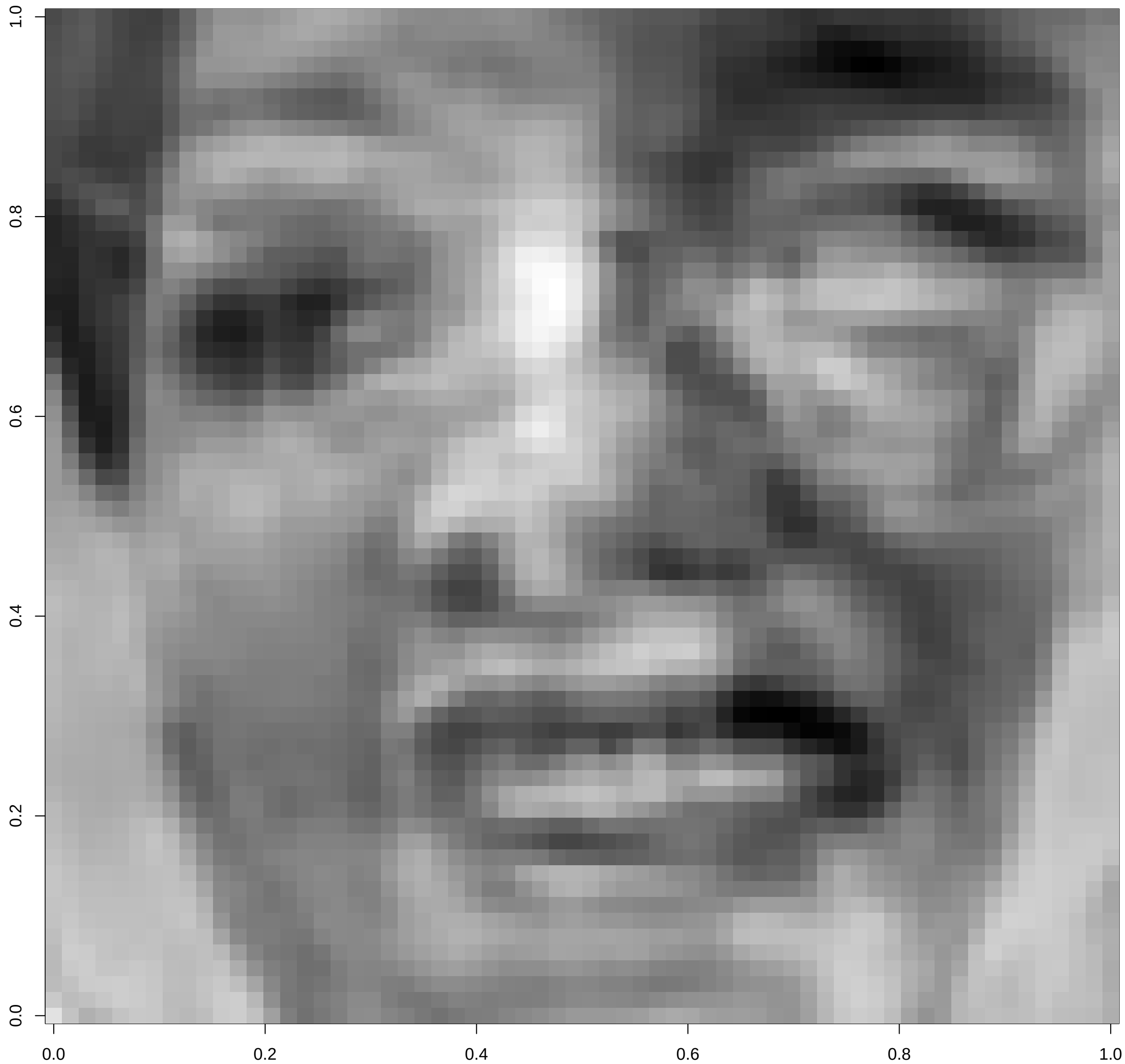

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

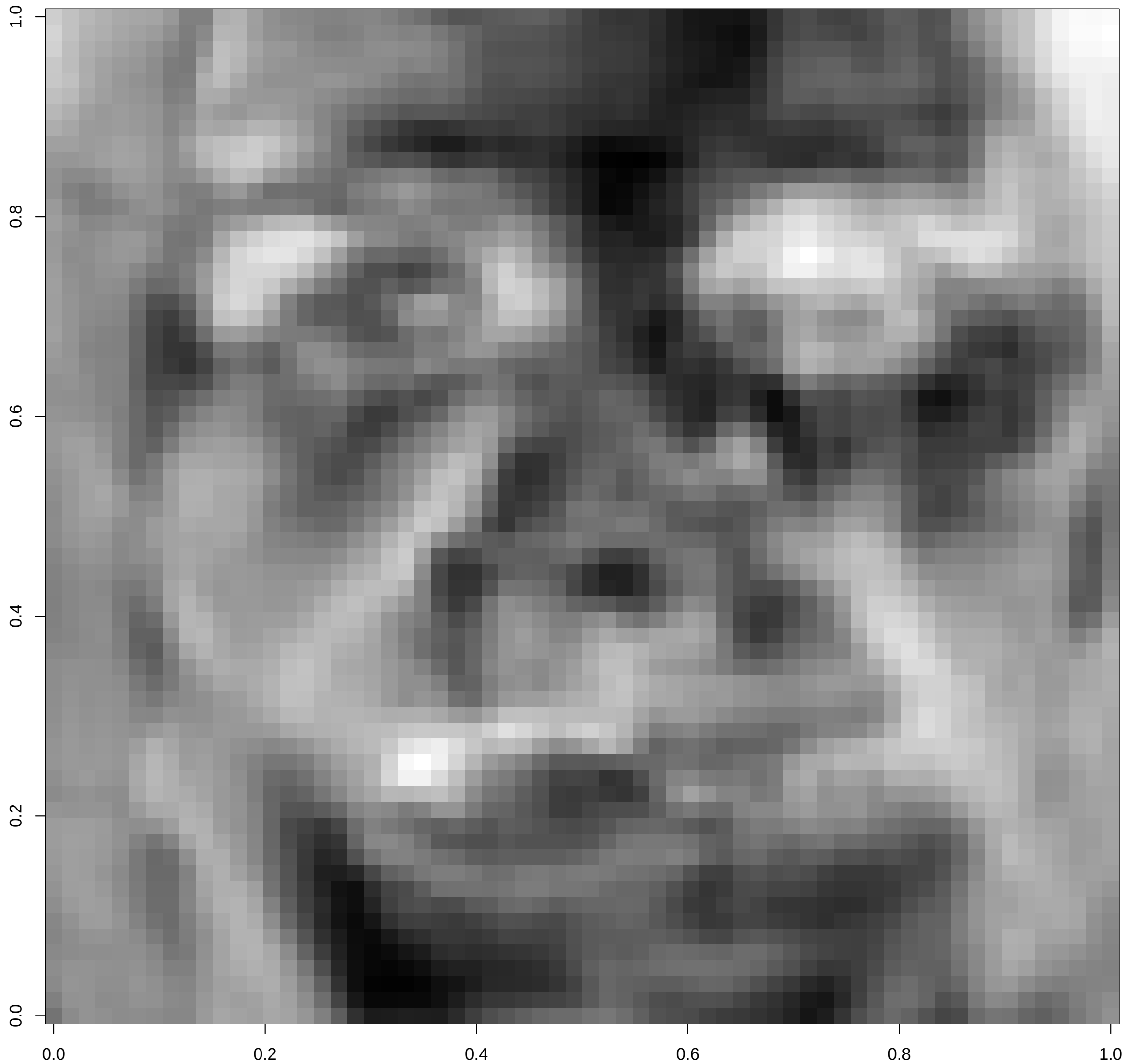
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


**Quesion 7: Recover an image** We can also recover the original faces by multiplying the eigenvectors with the x attribute of the prcomp object, i.e. prcomp$x[, 1 : n]$rotation[,1:n]) where n represent the number of components determined in 5). To this outcome, add the average face vector as well (see Xia and Ding 2014). Plot an original and recovered face for comparison

```
# Recover the original faces by multiplying the eigenvectors with the x attribute of the prcomp object
recoveredFaces <- pca$x[,1:11] %*% t(pca$rotation[,1:11])
```

```
# Plot the original face 1
plot(pictures[[1]])
```

```r
# Convert the 1st vector in recoveredFaces into a 64x64 matrix again
recoveredFace1 <- matrix(recoveredFaces [1,],  byrow=TRUE, nrow=64)

# Plot the recovered face 1 upside down
image(recoveredFace1[, nrow(recoveredFace1):1],
      col=grey(seq(0,1,length=256)))
```


**Question 8: Image detection** Let's built a simple image detection algorithm. Image detection works by comparing a focal image to the set of eigenfaces. This works by computing the Euclidean distance between the focal image vector, and the eigenface vector (computed in 7). Compute the Euclidean distance between one image of your choice, and all eigenface vectors. Which eigenface combination has minimum distance? This should be the focal image you selected.

Here we use a picture of Emma Watson already in the database and used in the sample.

```
euclidean <- function(a, b) sqrt(sum((a - b)^2))

for (i in 1:11) {
  print(paste(i, ":", euclidean(matrix1[[1]], pca$rotation[,i])))
}
```

```
## [1] "1 : 28.8162659189231"
```

```
## [1] "2 : 27.8251084156166"
## [1] "3 : 27.7874233828756"
## [1] "4 : 27.9422387986948"
## [1] "5 : 27.870963948529"
## [1] "6 : 27.7980028487924"
## [1] "7 : 27.7594211509889"
## [1] "8 : 27.8781190444991"
## [1] "9 : 27.8925259507965"
## [1] "10 : 27.896909779924"
## [1] "11 : 27.8755688826885"
```

It has the smallest distance to eigenface 7.