Emotion recognition: eigenfaces

Diego Hernández Jiménez

12/6/2022

The area of face recognition has changed a lot in the last 10 years or so. Convolutional neural networks achieve incredible levels of perfomance in multiple situations and are the state of the art. As far as I know, they outperform most of the alternatives. However it's worth visiting some the older techniques.

One of those is that proposed by Turk & Pentland (1991), the "eigenfaces" approach we could call it. It's based on matrix factorization via the Singular Value Decomposition (SVD), one the fundamental matrix factorization techniques in linear algebra. And also one of the most widely used. In psychology, for example, we can encounter the SVD when performing PCA or even in the language processing area and psychology of memory if we use the model of Latent Semantic Analysis (Landauer et al, 2013). And that's why I love it.

Here the aim is to recognize facial expressions, not just faces as a whole.The dataset I'm using was created by Ebner et al. (2010). Technically is just a sample of the full dataset, but it's enough. It contains 72 close-up photos. 6 people doing 6 different facial expressions 2 times each. Aspects like sex, age (young,middle-aged,old) are balanced.

```
library(imager)
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'imager'
```

```
## The following object is masked from 'package:magrittr':
##
##     add
```

```
## The following objects are masked from 'package:stats':
##
##     convolve, spectrum
```

```
## The following object is masked from 'package:graphics':
##
##     frame
```

```
## The following object is masked from 'package:base':
##
##     save.image
```

```
path <- "C:\\Users\\Diego\\Rprojects\\eigenfaces\\FACES_database"

img_files <- list.files(path,pattern='*.jpg')
str(img_files)
```

```
##  chr [1:72] "004_o_m_a_a.jpg" "004_o_m_a_b.jpg" "004_o_m_d_a.jpg" ...
```

Pre-processing

Like in most, if not all,image processing scenarios, first we have to transform the raw pictures in some format we can work with. In this case we'll work with bidimensional photos (grayscale images), with less resolution (reduced by 50%) to decrease the computational cost later on, and cropped so the image contains only the most relevant parts to recognize the facial expression.

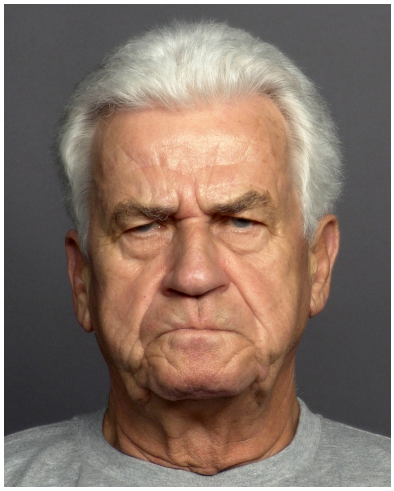
```
ex <- load.image(paste0(path,'\\',img_files[1])) |>
   grayscale(drop=T) |>
   imresize(scale=.5) |>
   crop.borders(nx=250,ny=300)


# image dimension after processing it.
# we will need this in many occasions
reference_dim <- dim(ex)[1:2]
```

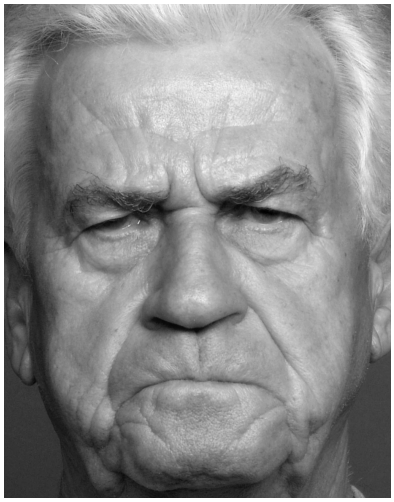
```
faces <- c()
for (i in 1:length(img_files)){
   im <- load.image(paste0(path,'\\',img_files[i])) |>
      grayscale(drop=T) |>
      imresize(scale=.5) |>
      crop.borders(nx=cropX,ny=cropY) |>
      cbind()

   faces <- cbind(faces,im)
}

save(faces,file=paste0(path,'\\','allfaces.Rdata'))
```

This is how a face looks before and after pre-processing

```
load(paste0(path,'\\','allfaces.Rdata'))

original <- load.image(paste0(path,'\\',img_files[1]))
processed <- as.cimg(faces[,1],dim=reference_dim)
# save.image(processed,paste0(path,'\\','processed.png'))


par(mfrow=c(1,2))
plot(original,axes=F,main='original')
plot(processed,axes=F,main='processed')
```

original **processed**





In a machine learning context it's important to split the data because we want to assess the generalization of the classifier, its perfomance on unseen data. Here I choose the simplest strategy. I split the data in two sets.

```
# 6 subjects, each has two pictures of six different facial expressions
# the pictures are sorted: (subject1,expression1,version1),(subject1,expression1,version2)...
# That means pictures in even positions are just the copies or second versions,
# so we can split the data into two equivalent sets

version1 <- seq(1,length(img_files),by=2)
version2 <- seq(2,length(img_files),by=2)
train_names <- img_files[version1]; test_names <- img_files[version2]
Xtrain <- faces[,version1]; Xtest <- faces[,version2]
```

Eigenface extraction via SVD

Here I follow the steps of the procedure proposed by Turk & Pentland (1991). It is no different from performing a principal component analysis. If our images are represented by the matrix $\mathbf{X}$, with p = (width · height) rows and n columns (as many as different images there are), then what we have to do is:

1. Center the images:

($\mathbf{1}$ is a p × 1 vector)

$$\mathbf{\Psi}_{1\times n} = \frac{\mathbf{1}^{\intercal}\mathbf{X}}{n}$$

$$\mathbf{C}_{p\times n} = \mathbf{X} - \mathbf{1}\Psi$$

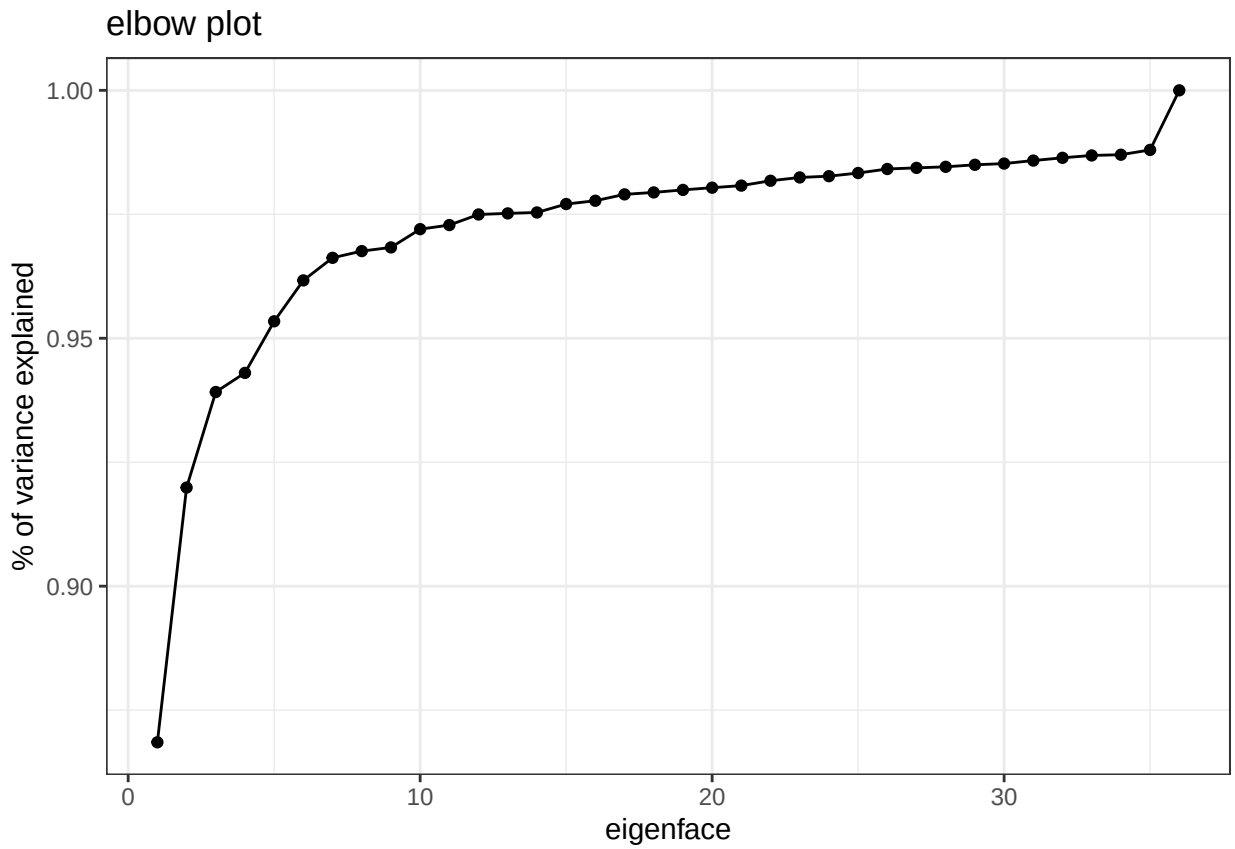
```
# An easier way to view the centering process:

# avg face: compute the mean value for every pixel
psi <- rowMeans(Xtrain)
# as.cimg(avgface,dim=reference_dim) |> save.image(paste0(path,'\\','avgface.png'))

# for each column (face), subtract the avg face
C <- apply(Xtrain,MARGIN=2,FUN=function(face) face-psi)
```

2. Apply the (truncated) SVD to the modified matrix:

$$\mathbf{C} \approx \mathbf{U}_{p\times k}\mathbf{\Sigma}_{k\times k}\mathbf{V}^{\intercal}_{k\times p}$$

k is the number of components (eigenfaces) selected. When $k < n$ and the matrix of images has more rows than columns, the decomposition is not equal to the original matrix but it's the best approximation in terms of the Frobenius norm (least squares error) (Brunton & Kutz, 2020)

```
SVD <- svd(C)
prop_var <- SVD$d/sum(SVD$d)

library(ggplot2)
ggplot(data.frame(var_exp=1-prop_var),aes(1:36,var_exp)) +
   geom_point() +
   geom_line() +
   labs(title='elbow plot',
        x='eigenface',y='% of variance explained') +
   theme_bw()
```



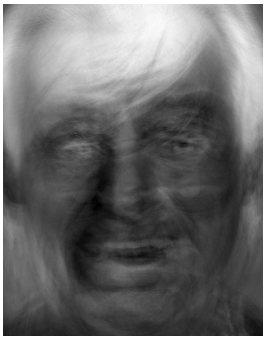
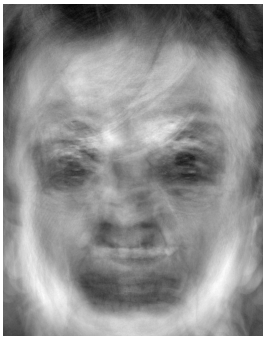
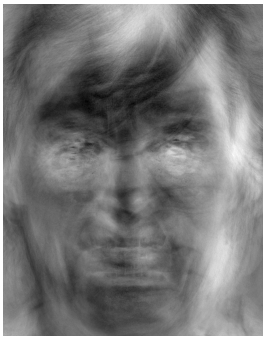
One reasonable choice of the number k of eigenfaces or components, based on this plot would be 10. It's not a very high number, accounts for approximately 97% of the variance and there isn't much gain in keeping more eigenfaces.

```
# another approach: based only on variance explained
# k <- which((1-prop_var) >= 0.95)[1]
k <- 10
U <- SVD$u[,1:k]
D <- SVD$d[1:k]
V <- SVD$v[,1:k]

# str(U);str(D);str(V)
# for (comp in 1:k){
#    as.cimg(U[,comp],dim=reference_dim) |> save.image(paste0(path,'\\','eigface',comp,'.png'))
# }
```

An example of the eigenfaces:

```
par(mfrow=c(1,3))
for (comp in 1:3){
   eigf <- load.image(paste0(path,'\\created\\','eigface',comp,'.png'))
   plot(eigf,axes=F,main=paste0('eigenface ',comp))
   rm(eigf) # free memory
}
```

eigenface 1 **eigenface 2** **eigenface 3**

Classification

The **U** matrix contains the basis vectors (eigenfaces) that span the new generated low-dimensional "face space". We can take advantage of that to build a classifier. We can take new images and project them on the new face space. We can do the same with the original (training) faces. As a result, for each image we obtain a set of k "weights" representing the importance of each component (eigenface) in that face. Now we use them as input of a knn classifier. We match the test face with the most similar training face in terms of, in this case, euclidean distance.

1. Project training faces on the new space.

$$\mathbf{\Omega}_{k\times n} = \mathbf{U}^\intercal \mathbf{C}$$

Each projected face is now a vector $\boldsymbol{\omega} = \mathbf{\Omega}_j$

2. Take some new image $\mathbf{x}_{p\times 1}$, center it and project it.

$$\hat{\omega} = \mathbf{U}^\intercal(\mathbf{x} - \mathbf{\Psi}^\intercal)$$

(remember that $\mathbf{\Psi}$ was 1 × p, hence the transpose)

3. Compare the "eigenweights" generated. Match the test face with the most similar face of the training set

$$j = \arg\min_j(\|\hat{\boldsymbol{\omega}} - \boldsymbol{\Omega}_j\|)$$

```
# person_age_sex_expression_version
splittrain <- strsplit(train_names,'_'); splittest <- strsplit(test_names,'_')

omegatrain <- t(U) %*% C
correct <- data.frame(list('all'=0,'person'=0,'age'=0,'sex'=0,'emo'=0))



for (t in 1:ncol(Xtest)){
   testface <- Xtest[,t]
   omegatest <- t(U) %*% (testface-psi)
   dists <- apply(omegatrain,2,function(omg) norm(omegatest-omg,type='2'))
   clasif <- which.min(dists)

   correct[t,'all'] <- t == clasif
   correct[t,'person'] <- splittest[[t]][1] == splittrain[[clasif]][1]
   correct[t,'age'] <- splittest[[t]][2] == splittrain[[clasif]][2]
   correct[t,'sex'] <- splittest[[t]][3] == splittrain[[clasif]][3]
   correct[t,'emo'] <- splittest[[t]][4] == splittrain[[clasif]][4]

   # par(mfrow=c(1,2))
   # as.cimg(testface,dim=reference_dim) |> plot(main='test',axes=F)
   # as.cimg(Xtrain[,clasif],dim=reference_dim) |> plot(main='classified',axes=F)
}


cat('empirical: \n')
```

```
## empirical:
```

```
colMeans(correct)
```

```
##       all    person       age       sex       emo
## 0.6111111 1.0000000 1.0000000 1.0000000 0.6111111
```

```
cat('expected by chance: \n')
```

```
## expected by chance:
```

```
c(all=1/36,person=6/36,age=12/36,sex=18/36,emo=6/36)
```

```
##        all     person        age        sex        emo
## 0.02777778 0.16666667 0.33333333 0.50000000 0.16666667
```

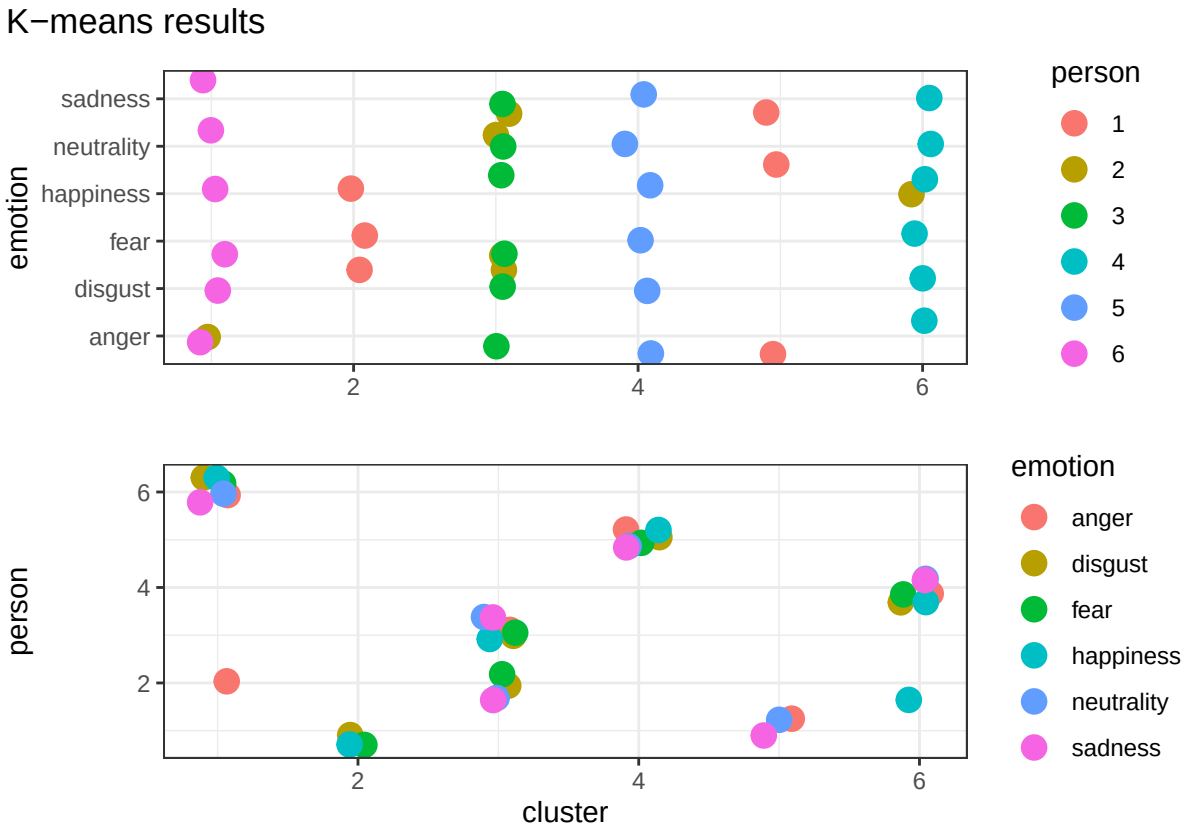
Interpretation and conclusions

Given the small dataset and the priors (expected perfomance by chance) we could say we have a pretty good face recognition system. However, it's fair to acknowledge that the models made almost the same pose in version 1 and version 2. Our system recognizes the person in this ideal conditions. Also, the system doesn't do what was intended to do. The aim was facial expression recognition but as we can see in the following plot, the features extracted (the eigenfaces) allow to generate clusters of individuals, but not of expressions.

```
omegatest <- t(U) %*% (apply(Xtest,2,FUN=function(face) face-rowMeans(Xtest)))
df <- data.frame(t(omegatest),
                 person=rep(1:6,each=6),
                 emotion=rep(c('anger','disgust','fear','happiness','neutrality','sadness'),times=6))

km <- kmeans(df[,1:10],6)
df$cl <- km$cluster

set.seed(134)
g1 <- ggplot(df,aes(cl,emotion,col=factor(person))) +
   geom_jitter(size=4,width=0.1) +
   labs(x='',y='emotion',col='person') +
   theme_bw()
set.seed(134)
g2 <- ggplot(df,aes(cl,person,col=emotion)) +
   geom_jitter(size=4,width=0.15) +
   labs(x='cluster',y='person',col='emotion') +
   theme_bw()

library(patchwork)
```

```
## Warning: package 'patchwork' was built under R version 4.1.3
```

```
g1 / g2 + plot_annotation(title='K-means results')
```



It's interesting to note that the K-means solution doesn't even allow perfect discrimination of individuals.The first person, for instance, which we would intuitively consider as unique cluster, we see that can be divided in two subclusters. The happinness, fear and disgust expressions seem to be similar enough to one cluster, separated of the rest of expressions of that person, which form a second cluster.

In any case, it was a very fun project.

References

Brunton, S. L., & Kutz, J. N. (2020). Singular Value Decomposition (SVD). In S.L. Brunton & J.N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*, (pp. 1-47). Cambridge University Press.

Ebner, N., Riediger, M., & Lindenberger, U. (2010). FACES—A database of facial expressions in young, middle-aged, and older women and men: Development and validation. *Behavior research Methods*, 42, 351-362. doi:10.3758/BRM.42.1.351.

Landauer, T. K., McNamara, D. S., Dennis, S., & Kintsch, W. (Eds.). (2013). *Handbook of latent semantic analysis*. Psychology Press.

Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1), 71-86.