GUÉNA Vincent
MELLERIO Antoine

03/11/2020

ASSIGNEMENT 1
EigenFaces

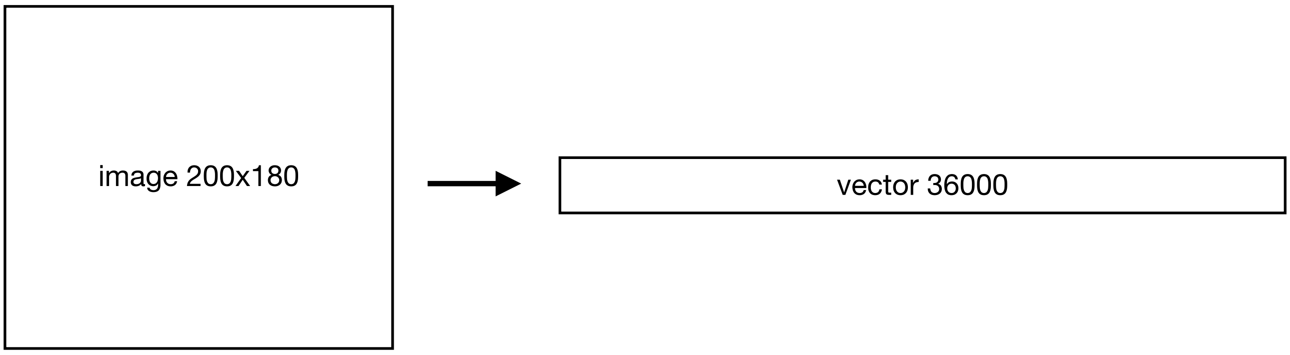
a. Principal Component Analysis function

Our objective is here to build a function that apply the principal component analysis to any photograph.

First we need to convert the picture into usable data :

1. we build our dataset where each row represents an image.

2. we built the function im2vect( ) which convert an image 200x180 into a single vector of length 36000.



3. we convert the image into a black and white image because the most important information is contained into the shape of the image. Indeed the colors can change with the surroundings of the person when the picture was taken, with the light, etc.

4. We build the matrix D with all the names of the file contained in the data base. All the image are concatenated into the matrix Data.

We now want to calculate the eigenvectors of the covariance matrix of our data base. Unfortunately, the number of columns is much higher than the number of rows, the size of the covariance matrix should be 36000^2, which is not computable.
To solve this problem, we compute the covariance matrix of the transposed of the data base :

$$G = [x_{i,j} - \mu]$$

where μ is the mean of Data and $x_{i,j}$ a number from Data

We find ϕ_s the eigenvectors of Σ_s, and then compute $G^t\phi_s$ which are the eigenvectors of Σ_l, the covariance matrix of our data base.

$$\Sigma_l = \frac{1}{n-1} G^T G, \quad (l = \text{long}),$$

$$\Sigma_s = \frac{1}{n-1} G G^T, \quad (s = \text{sort}),$$

As the eigenvalues of Σ_s and Σ_l are the same, we computed the proportion of variance retained by each eigenvectors using the eigenvectors of Σ_s.

b. <u>Predictive function</u>

Our objective is now to build a function using a K-nn classifier.
This function that we call Recogn get as input the name of the photo in which we want to identify the person, and the feature k of the K-nn classifier :

Recogn = function(photo, k)

The Recogn( ) function proceeds as following :

Steps	Function used
1 - import the data of the black and white pixels of the photo	Im2vect( )
2 - project this data in the principal component basis	
3 - calculate the distances with all our datas that were also projected	dist( )
4 - if the distances are all higher than a limit value, the input person is not in our data base : return 0	
5 - if not, get the names of the people associated with the k smallest distances	sort( )
6 - outputting the one appearing the most	sort( )

Given this process, our program need 3 features to work :
- the number of principal component to take into account
- the number of neighbors of the k-nn classifier
- the condition to determine a person is not in the database

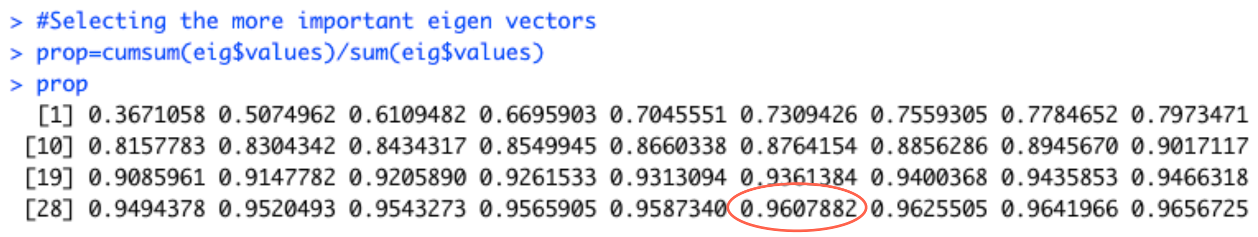
Let's see what we need to choose for these features to optimize the program.

c. <u>Choosing the optimized features</u>

1. Number of principal components

The number of principal components taken into account has an impact on the proportion of the whole data set taken into account.

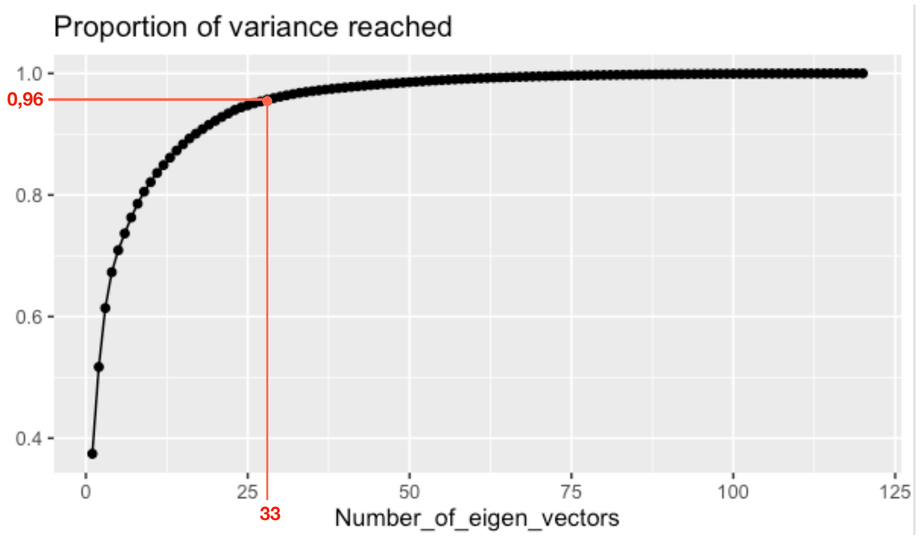
We compute this influence with the whole dataset. Then we select as many components as possible to take into account most of the variance without getting the computing too long.



```
> #Selecting the more important eigen vectors
> prop=cumsum(eig$values)/sum(eig$values)
> prop
 [1] 0.3671058 0.5074962 0.6109482 0.6695903 0.7045551 0.7309426 0.7559305 0.7784652 0.7973471
[10] 0.8157783 0.8304342 0.8434317 0.8549945 0.8660338 0.8764154 0.8856286 0.8945670 0.9017117
[19] 0.9085961 0.9147782 0.9205890 0.9261533 0.9313094 0.9361384 0.9400368 0.9435853 0.9466318
[28] 0.9494378 0.9520493 0.9543273 0.9565905 0.9587340 0.9607882 0.9625505 0.9641966 0.9656725
```

Using the 33 first components give us 96% of the variance which seems acceptable.

We plot this relation to make sure that increasing the number of components doesn't give us that much improvement on the promotion of variance taken into account.



The slop of the curve has obviously decreased much at x=33. It is a satisfying feature for our program.

—> N_PC = 33

2. Number of neighbors of the k-nn classifier

The database we are using is made of the photographs of 25 people. Each of them has 6 photographs. If we want to identify one of them, we expect that the distances calculated by our algorithm between the new photo and the data set will be lower with the 6 pictures of the person.

We verify that by taking out from the database the photo 1AT and computing all the distances between it and the rest of the dataset. With the variable ordenados we print the first 9 results :

```
$x
   1BT.jpg    1DT.jpg    1FT.jpg    1CT.jpg    1ET.jpg    9DT.jpg    9FT.jpg    9ET.jpg
  115.8011   757.9208   856.9859   858.2208  1174.6928  2539.4857  2593.4641  2678.3759
```

As expected the smallest distance are the ones with the same person. Hence we can use 6 as the number of neighbors of the k-nn classifier.

—> N_neighbors = 6

3. Identifying conditions

The first approach of this question is to look at the evolution of the distances. This pattern may have discontinuities that we could use to identify when the pictures we compare our new photo with are no longer from the same person. However this really depends on the similarities of the people of our dataset, and thus is not very reliable.

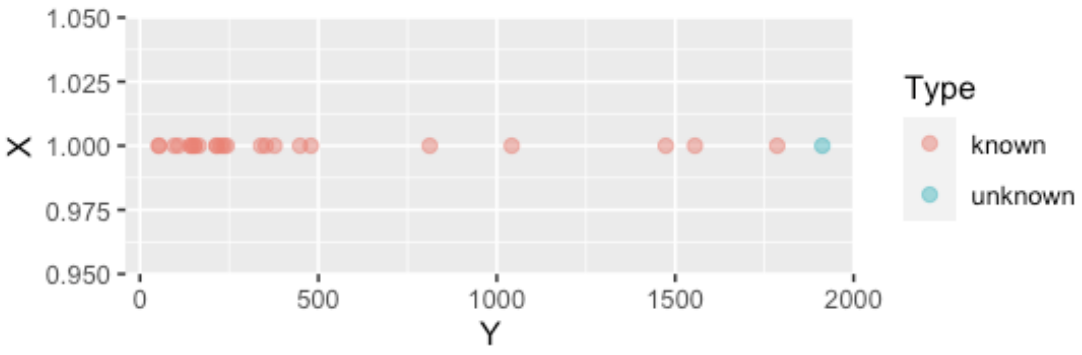
A second approach is to try to find a threshold of the distances from which the new picture can be identified within our database or not. We need to compare the usual distances got with a picture of someone who is in our database and someone who isn't.
First, we need a picture from each person as impute to out algorithm in order to get the minimal distances with picture of people that ARE in the database. Hence we take out of the database all the 'AT' pictures :

```
#taking out 1 picture per person to check the program
Data_train = Data[-seq.int(1, nrow(Data), by=6),]
```

Then, we take one person completely out of the database, the number 1, in order to get the minimal distances with picture of people that ARE NOT in the database :

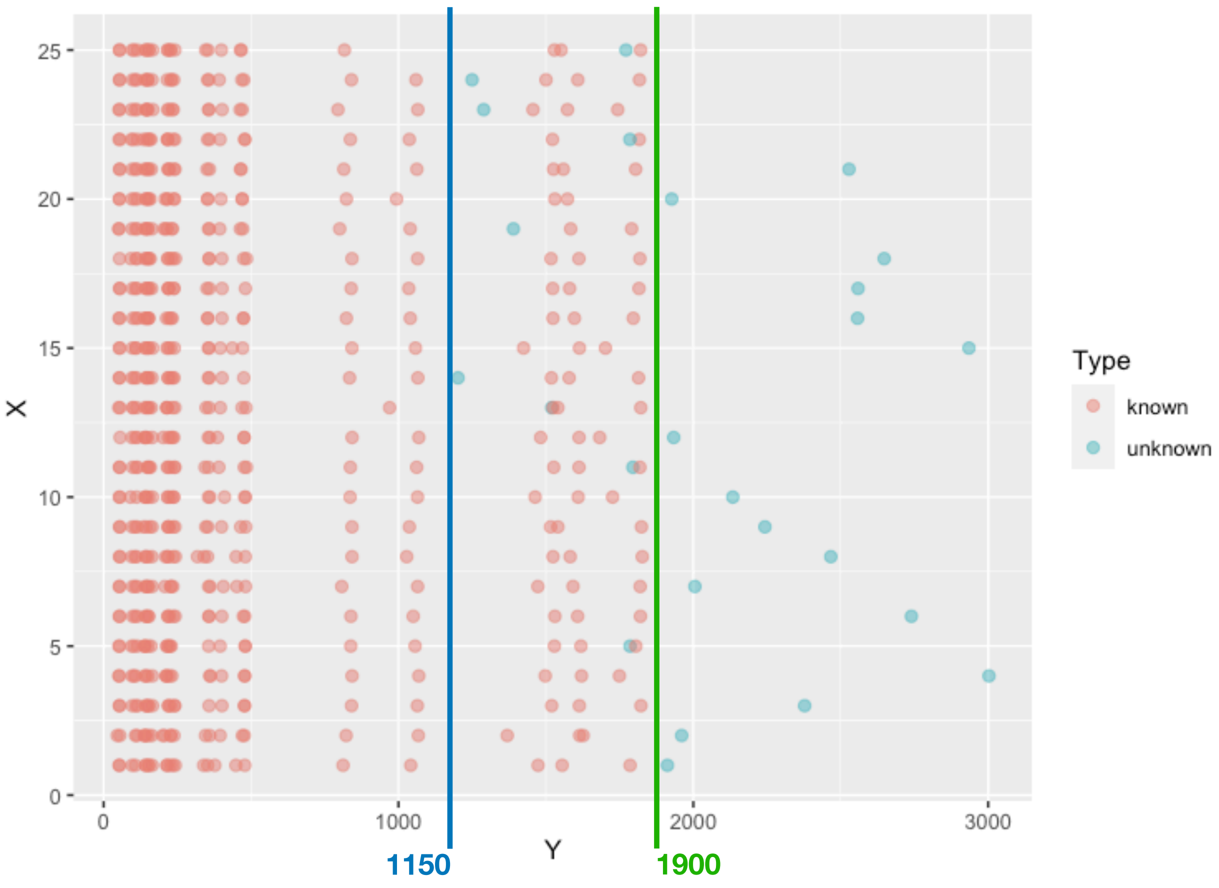
```
#taking out one person to test the function with a new person
Data_train = Data_train[-c(1:5),]
```

Now we want to compare the minimal distances of the removed person and of the people still in the database. To make it seeable we plot these distances along a ligne, categorizing the points between the one that correspond to an unknown person, and the others that correspond to known people :



In this graph, X is the number of the person taken out of the database, and Y are the minimum distances of the 1st pictures of the 25 people with the rest of the dataset.
We now have a first idea of the value to put on the threshold to distinguish known to unknown people.

However, selecting this threshold over one computing has the risk to be influence by a ‘luck' dimension.
To leverage this phenomenon, we decide to process the same analysis changing the person taken out of the database. We then plot all the results on the same graph :



We can see ‘vertical lines’ that correspond to the minimal distances of the first picture of a same person. It doesn’t change when the removed person changes, which is normal.

In addition, the points are not perfectly aligned because the removal of a person has an influence on the principal components, and thus on the distances calculated. Though this influence is quite negligible as the points are almost aligned.

We can now trace the limit ligne to spare the known from the unknown points.

- If our priority is not to identify wrongly people that are in our dataset, then the best fit is Y=1900 as shown by the green line. Then 9/600 = 1,5% of the people we identified were actually unknown.
  Example of application : categorizing of people.

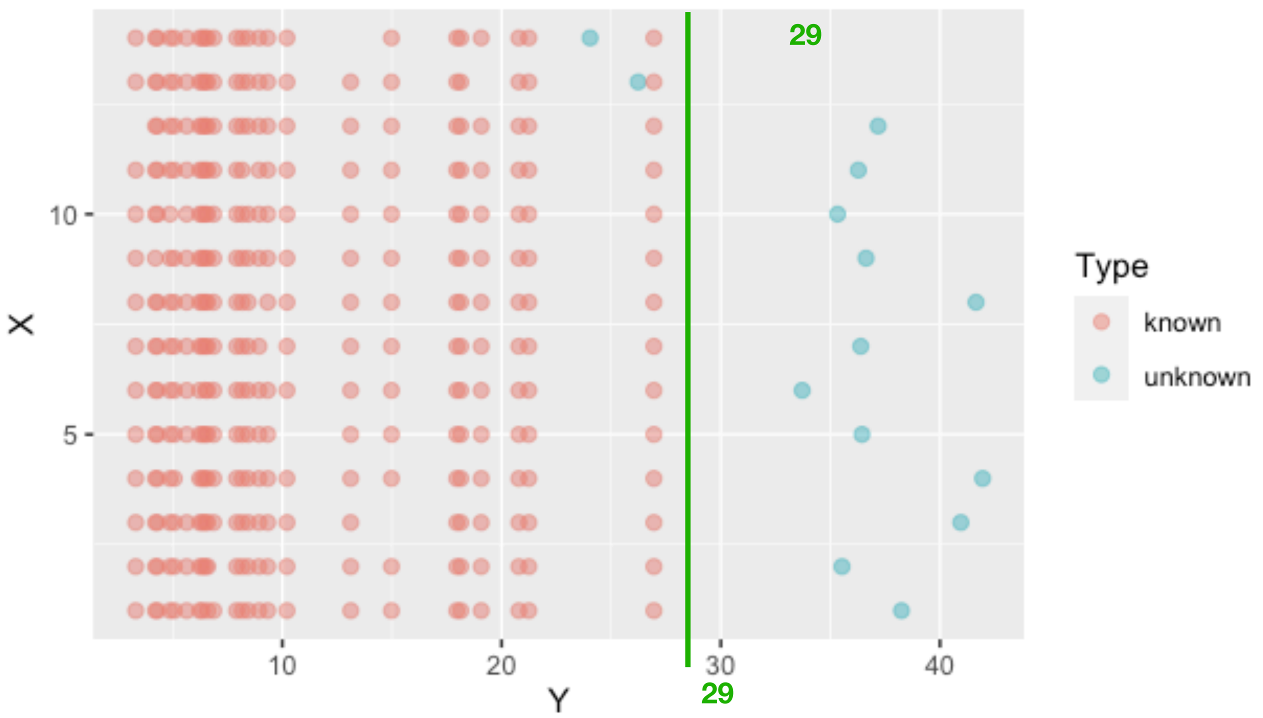
- if our priority is to not identify someone as someone we know when we actually do not, then the right fit is Y=1150 as shown by the blue line. Then, we identify 72/600 = 12,7% of the people we actually know as unknown.
  Example of application : security check to enter a highly confidential building, we let people that are in our database enter.

As we don’t have any error preference, we simply chose the limit that give the fewer errors : we choose to use the green line as our identifying limit with an error of 9/625 = 1,44%.

—> Dist_lim = 1900

d. <u>Computing without PCA and comparison</u>

We execute our whole database without projecting it into the principal component base.
The number of neigbors used for the k-nn remains the same.
We just need ton select the right recognition limit. Hence we compute the same graph of the minimal distances with known and unknown people :



For the same reasons than with PCA, we set the identification limit to Y=29.
We can see that we get an error of 2/35 = 0,6%. This result is quite accurate and even a bit better than the predictions using PCA (error of 1,44%).

However, a non-negligible difference between the two processing ways is the duration of the execution. We calculated these times using Sys.time( ) function :

k-nn	Time of execution
with PCA	0,12 seconds
with the whole dataset	12,45 seconds

We can see that computing the k-nn classifier with PCA is 10 times shorter than without. Even though the total time of 10 seconds can appear not that long, we are here using a very restrained database of only 150 images. Hence this factor is huge and using PCA can considerably improve the executing time.

e. <u>Conclusion</u>

We were able to create an algorithm using PCA and a k-nn classifier in order to recognize the person of a picture with an accuracy of 98,54%.

Even though it doesn't improve the accuracy, using the PCA before the classifier is very important as it reduces the execution time by a factor 10.