INDIAN STATISTICAL INSTITUTE

BACHELOR OF STATISTICS

PROJECT II

VECTOR AND MATRICES II

Face Recognition with Eigen Analysis

Purushottam Saha (BS2119)
Instructor: Arnab Chakraborty
June 2022

Abstract

In Face Recognition, each picture is of several thousand of dimensions, so comparing those directly is really inefficient and also noise catchy, But as every photo has the structure and many facial features similar, so the dimension the facial data actually spans, is much less, in fact most of the data (as we are going to see) lies in 10-15 dimensions, which can be used to store pictures efficiently, and more over to answer the question if a new picture is of someone in our database (and if it is, then whose) or not.

1 Introduction

A picture is generally considered as a matrix, but considering it as a vector, we get a 180×200 dimensional vector, i.e. a 36,000 dimensional vector. So storing similar photos can be inefficient in this way. What we can exploit is the term 'similar', i.e. the pictures are not that much different in structure as they have many similar features, so the dimension of the space spanned by the pictures is much less than 36000. So, we need an appropriate set of ortho-normal vectors (those vectors are also of dimension 36000, so they are also re-presentable as pictures) such that the pictures are approximately (we will see the actual meaning of this term later) linear dependent on them, i.e. those vectors create an ONB for the pictures. Our task is to minimise the dimension as much as we can, without losing much information. We do so by **PCA** .

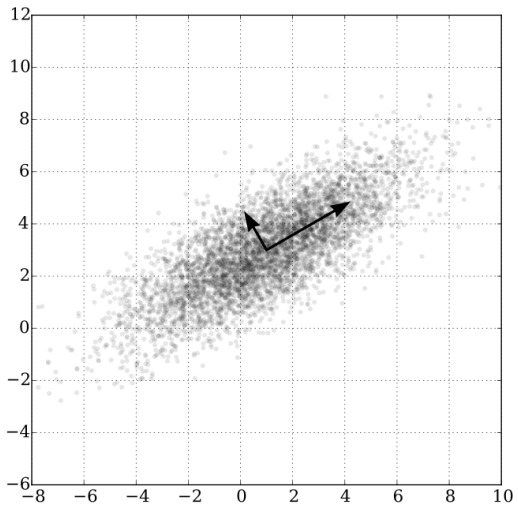


Figure 1: PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

2 What is PCA ?

Principal Components of a collection of points are a sequence of vectors which are the best fitted lines to the data being perpendicular to the previous vectors, where best fitted lines are those which minimizes the average squared distances from the point to the line. So, **PCA** or **Principal Component Analysis** is computing those Principal Components to perform a change of basis and ultimately reduce the dimension of the data (if possible).

The same idea can be restated as, we need to maximize the variance of the projections. From the Lemma 1 in the Appendix, we see that the Variance of a linear combination of the data is indeed a Quadratic Form, i.e.

$$var(v^T X) = v^T \Sigma v$$

, where v is the vector with coefficients of linear combination, and Σ is the Covariance Matrix. Now from Lemma 2, we see that the maximization of the Quadratic Form occurs when v is an eigenvector of Σ corresponding to the maximum eigenvector, and not only that, this works in a layered way, i.e. the second largest eigenvector causes the maximization of the Quadratic Form for vectors perpendicular to v and so on.

So what we need to perform for Face Recognition in terms of PCA that, we need the eigenvectors of the Covariance Matrix, each pixel being a Random Variable. First thought about this will be that this is very computationally inefficient, as the Covariance Matrix is of dimension 36000 $\times$ 36000, as there are 36000 pixels in each picture; but this can be handled by the Lemma 4, as though X^TX is of dimension 36000 $times$ 36000, X being the pictures (say 50 pictures are there, as in our data set, XX^T is of dimension 50 $\times$ 50, of which we can easily calculate the eigenvectors of and by the lemma, eigenvalues of both are same, also we get our required eigenvectors using Lemma 5, more about this in the Implementation.

3 Something More! Clustered Data?

Now we come to our exact problem, we have 50 pictures of 10 person, 5 pictures each. Now we know about each photo, that whose picture is whose. It is obvious that pictures of same person is quite similar than pictures of dif-

ferent people. So, indeed, there are clusters in the data. If we see the new coefficients of the pictures in the reduced basis (by PCA, referred as scores in R Code), we see that the clusters, i.e. photos of same person are not similar, hence while reducing the dimension, we have lost the information about the clustering, which can be really useful for us for Facial Recognition. For that we bring the concept of With-in Cluster Covariance and In-between Cluster Covariance!

3.1 With-in Cluster Covariance

There are 10 clusters in our data, each cluster having 5 data points. For each cluster, we consider the Covariance Matrix of those 5 points, and we sum them up. Hence we get a matrix, which we term as the With-in Cluster Covariance Matrix, W.

3.2 In-between Cluster Covariance

For the In-between Cluster Covariance, we consider the mean vector of a cluster as the representing point of the cluster and hence we consider each cluster as a point. Now we get 10 mean vectors for 10 clusters and hence get the In-between Cluster Covariance Matrix B.

From the definition of those Covariances, we can understand that Now, we want to maximise the distances in between clusters and simultaneously want to minimize the distances within clusters themselves, so from the previous

definitions we understand that we actually want to maximize

$$\frac{X^T B X}{X^T W X}$$

over X, which by Lemma 6 is the largest eigenvalue of $W^{-1}B$ and by Lemma 7 the process is layered just as PCA.

4 Implementation

4.1 Test Data

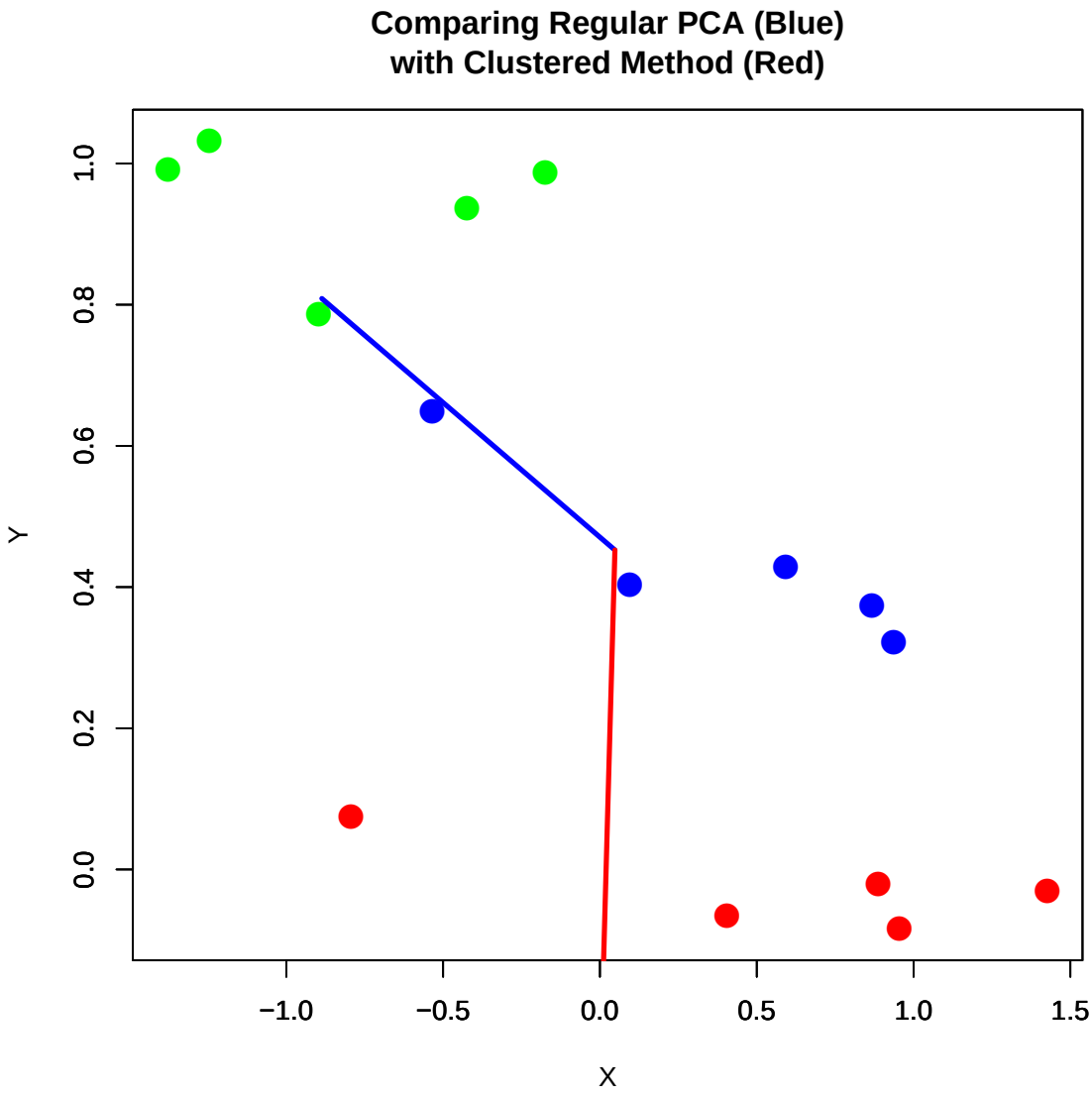
We first perform the Cluster Analysis discussed on the Test Data.

4.2 R Code: Test Data

```
# Test Data
dat <- read.csv("testdat.csv")

tProcess = function(){
  data = dat[1:15,1:2]
  # General PCA : to see the direction prescribed
  # by general PCA
  M = cov(data)
  e = eigen(M)
```

```
  O = apply(e$vectors,2,function(x) x/sqrt(sum(x*x)))

  # Within Cluster Covariance Matrix
  W1 = cov(data[1:5,])  #cov of the first 5 points
  # (cluster 1)
  W2 = cov(data[6:10,])
  W3 = cov(data[11:15,])
  W = (5-1)*(W1+W2+W3)/(15-1) # combined
  # (the denominator is n-1)
  # In Between Cluster Covariance Matrix
  m1 = apply(data[1:5,],2,mean)
  # Mean of the first cluster
  m2 = apply(data[6:10,],2,mean)
  m3 = apply(data[11:15,],2,mean)
  B = 5*cov(rbind(m1,m2,m3))
  # Covariance matrix when all points in a
  # cluster equals the cluster mean
  A = solve(W)%*%B # We needed to maximize X'BX/X'WX,
  # which is the largest eig value of W^-1B i.e. A
  eig <- eigen(A)
  Q = apply(eig$vectors,2,function(x) x/sqrt(sum(x*x)))
  # Normalizing the eigen vectors
  scores = as.matrix(data) %*% as.matrix(Q)
  # Scores, i.e. data in the eigen basis
  scores = scores[,-2] # Last part of the data can be
  # removed so as to support that the data is
  # affine space of 1 dim
  return(list(centre=data,onb=Q,scores=scores,
  values=eig$values,non_clus_onb=O))
}
```

```
p = tProcess()
rangex = range(dat[1]) # Range of X values
rangey = range(dat[2]) # Range of y values
mean_data = apply(dat,2,mean) # Mean Data
plot(dat[1:5,],main="Comparing Regular PCA (Blue)
with Clustered Method (Red)",col="red",
xlab="X",ylab="Y",
xlim = rangex,ylim = rangey,
pch=16,cex=2) # First Cluster
par(new=TRUE)
# So as to continue to plot on same window
plot(dat[6:10,],main="",col="blue",
xlab="",ylab="",
xlim = rangex,ylim = rangey,
pch=16,cex=2) # Second Cluster
par(new=TRUE)
plot(dat[11:15,],main="",
col="green",xlab="",ylab="",
xlim = rangex,ylim = rangey,pch=16,cex=2)
# Third Cluster
segments(x0=mean_data[1],y0=mean_data[2],
x1=(mean_data[1]+p$non_clus_onb[1,1]),
y1=(mean_data[2]+p$non_clus_onb[2,1]),
col="blue",lwd=3)
# direction of spread of data from Regular PCA
segments(x0=mean_data[1],y0=mean_data[2],
x1=(mean_data[1]+p$onb[1,1]),
y1=(mean_data[2]+p$onb[2,1]),
col="red",lwd=3)
```




```
# Modified direction of spread of data
# considering clusters
```

Clearly the basis got from general PCA (Blue one) follows the spread of the overall data cloud, which is dominated by the scatter within the clusters and not the scatter between them, where as the basis got from Clustered approach(Red one) follows the spread of clusters, i.e. is vertical. So indeed Clustered Approach is better for Clustered Data.

4.3 Face Data

Now we apply the same Algorithm for the Face Data. As for Clustered Approach, we need to compute eigenvalues of eigenvalue of $W^{-1}B$, each of W and B are of dimension 36000×36000, we first reduce the dimension of the pictures using PCA, and then apply Clustered Approach to the first 10 most significant eigenvectors as the basis to make the second approach Computationally feasible.

4.4 R Code: Face Data

```
# We start with a bunch of facial photographs,
# 5 photos of 10 persons each.
# All the images are 200£\times£180 in size.
# So they are 36000 dimensional
# vectors. Our aim is to reduce the dimension
# as much as possible without
# losing our ability to recognize the faces.

# We start with loading the images.

library(jpeg) # Required library

loadImages = function() { # The function loads
# all the images
  name = c("male.pacole","male.pspliu",
  "male.sjbeck","male.skumar","male.rsanti",
  "female.anpage","female.asamma","female.klclar",
  "female.ekavaz","female.drbost")
```

```
  x = matrix(0,nrow=10*5,ncol=200*180)
  # X is the data matrix
  k = 0 # K is the counter

  for(i in 1:10) { # i represents which person's
  # photograph is considered
    for(j in 1:5) { # j represents what numbered
  # photograph is considered for the fixed person
      k = k + 1  # hence k is the overall counter
      # The first argument below should be
      # the location of the face folder.
      filename = paste("./grayfaces/",name[i],
      ".",j,".jpg",sep="")
      x[k,] = as.vector(readJPEG(filename))
      # so the photo is converted into a vector
      # of dim 36,000
    }
  }
  return(x)
}

# Next, we perform PCA. The following function
# does precisely this, but without using R's
# built-in tools like prcomp or princomp,
# we perform the operations explicitly so that
# you can appreciate what goes on
# "behind the scene".

process = function(x) {
```

```
meanx = apply(x,2,mean)
# We are getting the mean picture
y = scale(x,scale=F) # Rows of y are the cases,
# origin shifted to mean
# Now as origin is shifted to mean, so
# second order raw moment is variance,
# hence the matrix Y'Y is Covariance matrix of x
# But next we need eigen values of the same,
# and calculating eig values of
# 36000x36000 matrix is impractical.
# Hence we use the trick that AB and BA have
# the nonzero eig values in common, and
# YY' is only 50x50 instead of 36000x36000,
# so we compute YY' and take the eig values
# of the same.
A = y %*% t(y)
eig = eigen(A) # Eig vector associated with
# largest eig value maximize x'Ax, and that
# s what we use in PCA

P = t(y) %*% eig$vec[,-50] # Now, columns of P
# are eig vectors of Y'Y (Y'YY'x = lambda*Y'x),
# last vector is not that important as the total
# is just an affine space in 49 dimension
Q = apply(P,2,function(x) x/sqrt(sum(x*x)))
# #Columns of Q form onb for rowspace of Y,
# just scaling the orthogonal basis to onb

scores = y %*% Q # scores[i,] is the tuple representatio
score_cluster = apply(scores,1,function(x) Re(sum(x)))
```

```
# A suitable norm to represent the picture
plot(score_cluster, main="Norm Plot by doing  PCA
(no clusters)",
ylab="Norm",pch=16,cex=2) # Plotting the norm
# We see that the clusters are not clear,
# as we have not spent any affort to minimise
# the distance between clusters,
# So next we are going to do that.

# Now comes the part of the in-between and
# within covariance matrices
# To start, we only consider the
# first 10 eigen-vectors : claiming that
# 10 dimension is a good estimate for
# the dimension spanned by the facial data

# Hence we have the truncated data:
trunc_onb = Q[,1:10] # Only first 10 eig vectors
# (ordered in decreasing value of
# corresponding eig value)
trunc_data = y %*% trunc_onb

# Within Covariance:
W = (5-1)*(cov(trunc_data[1:5,])+
cov(trunc_data[6:10,])+
cov(trunc_data[11:15,])+
cov(trunc_data[16:20,])+
cov(trunc_data[21:25,])+
cov(trunc_data[26:30,])+
cov(trunc_data[31:35,])+
```

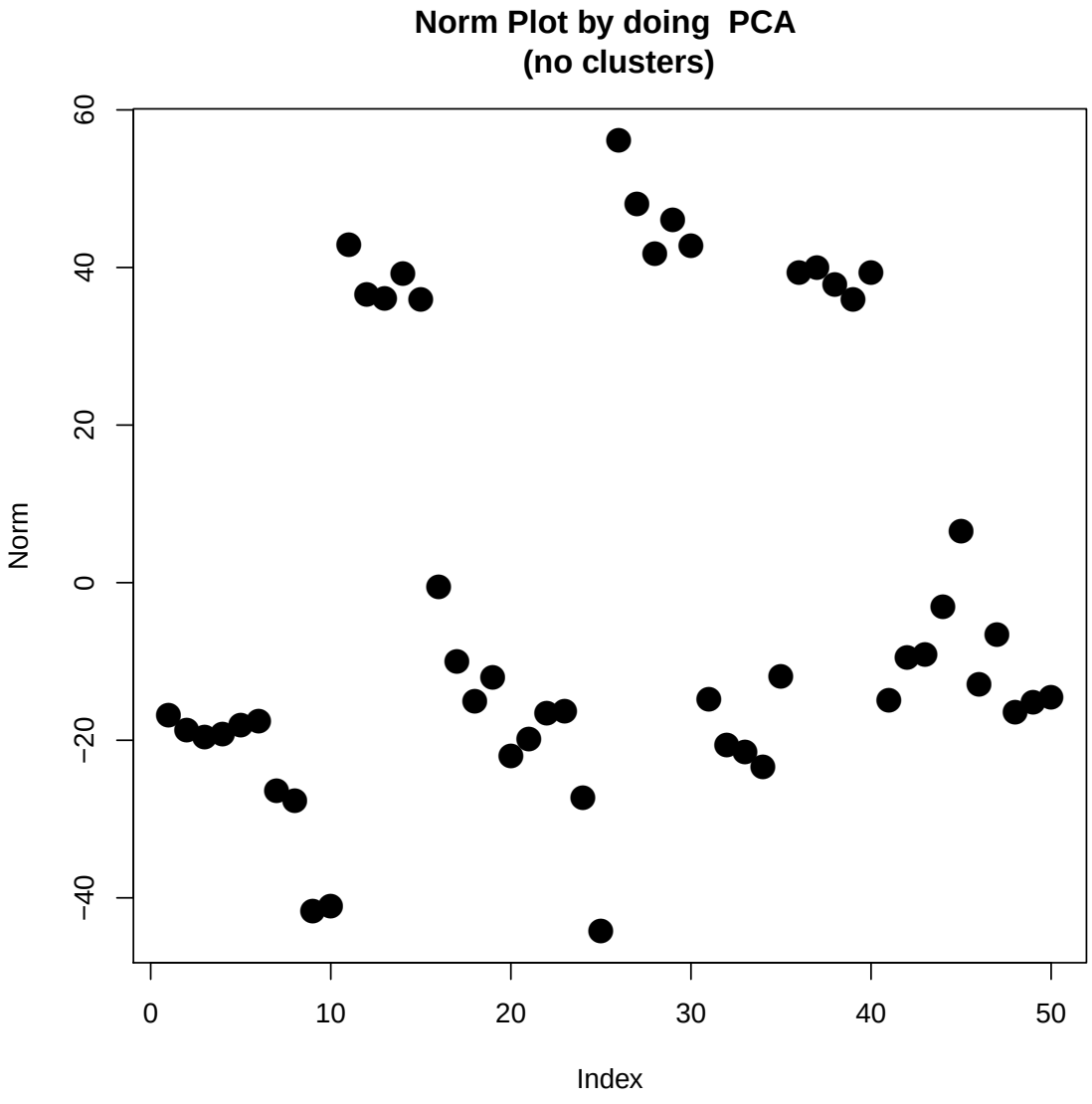
```
cov(trunc_data[36:40,])+
cov(trunc_data[41:45,])+
cov(trunc_data[46:50,]))/(50-1)
# In between Covariance:
B = 5*cov(rbind(apply(trunc_data[1:5,],2,mean),
apply(trunc_data[6:10,],2,mean),
apply(trunc_data[11:15,],2,mean),
apply(trunc_data[16:20,],2,mean),
apply(trunc_data[21:25,],2,mean),
apply(trunc_data[26:30,],2,mean),
apply(trunc_data[31:35,],2,mean),
apply(trunc_data[36:40,],2,mean),
apply(trunc_data[41:45,],2,mean),
apply(trunc_data[46:50,],2,mean)))

# We want to maximize x'Bx/x'Wx, which is
# max eig value of W'B, hence :
M = solve(W)%*%B # Solve(W) is W^-1
new_eig <- eigen(M) # Eigen value and vectors of M
new_coord = apply(new_eig$vectors,2,
function(x) x/sqrt(sum(x*x)))
# Again normalizing to get the onb
new_scores = trunc_data %*% new_coord
# New scores are new tuple representations
# of the pictures in the new onb
# print(new_scores)

plot(new_scores[,1],new_scores[,2],col = c(1,
1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,
5,5,5,5,5,6,6,6,6,6,7,7,7,7,7,8,8,8,8,8,
```

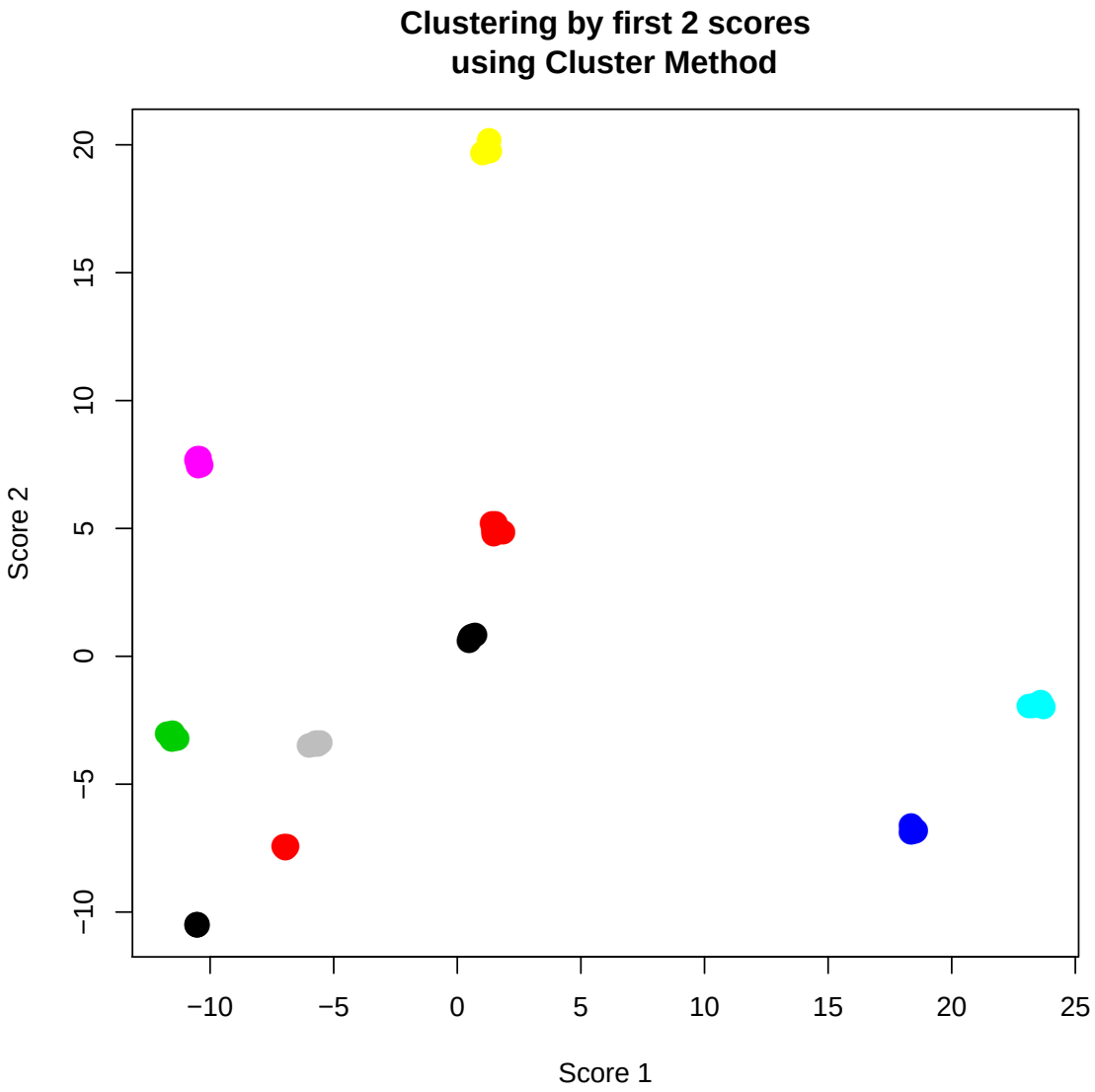
```
  9,9,9,9,9,10,10,10,10,10),pch=16,cex=2,
  main ="Clustering by first 2 scores
  using Cluster Method",
  xlab="Score 1",ylab="Score 2")
  # Plotting first two score for each image
  new_score_clusters = apply(new_scores,1,
  function(x) Re(sum(x))) # A suitable norm
  # to identify the cluster of vectors (10 dim)
  plot(new_score_clusters,col = c(1,
  1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,
  5,5,5,5,5,6,6,6,6,6,7,7,7,7,7,8,8,8,8,8,
  9,9,9,9,9,10,10,10,10,10),pch=16,cex=2,
  main="Clustering using Norm
  using Cluster Method",ylab="Norms")
  # Plotting the norm

  return(list(centre = meanx, onb = Q, scores = scores,
  values = eig$values, new_onb = trunc_onb,
  new_scores = new_scores,
  new_norm = new_score_clusters,
  new_values = new_eig$values))
}

# Each principal component is again
# a 200*180 dimensional vector, and hence may
# be considered as an image. It might be
# instructive to take a look at these.
# The following function helps you to just that.

showFace = function(newCoord, i) {
```

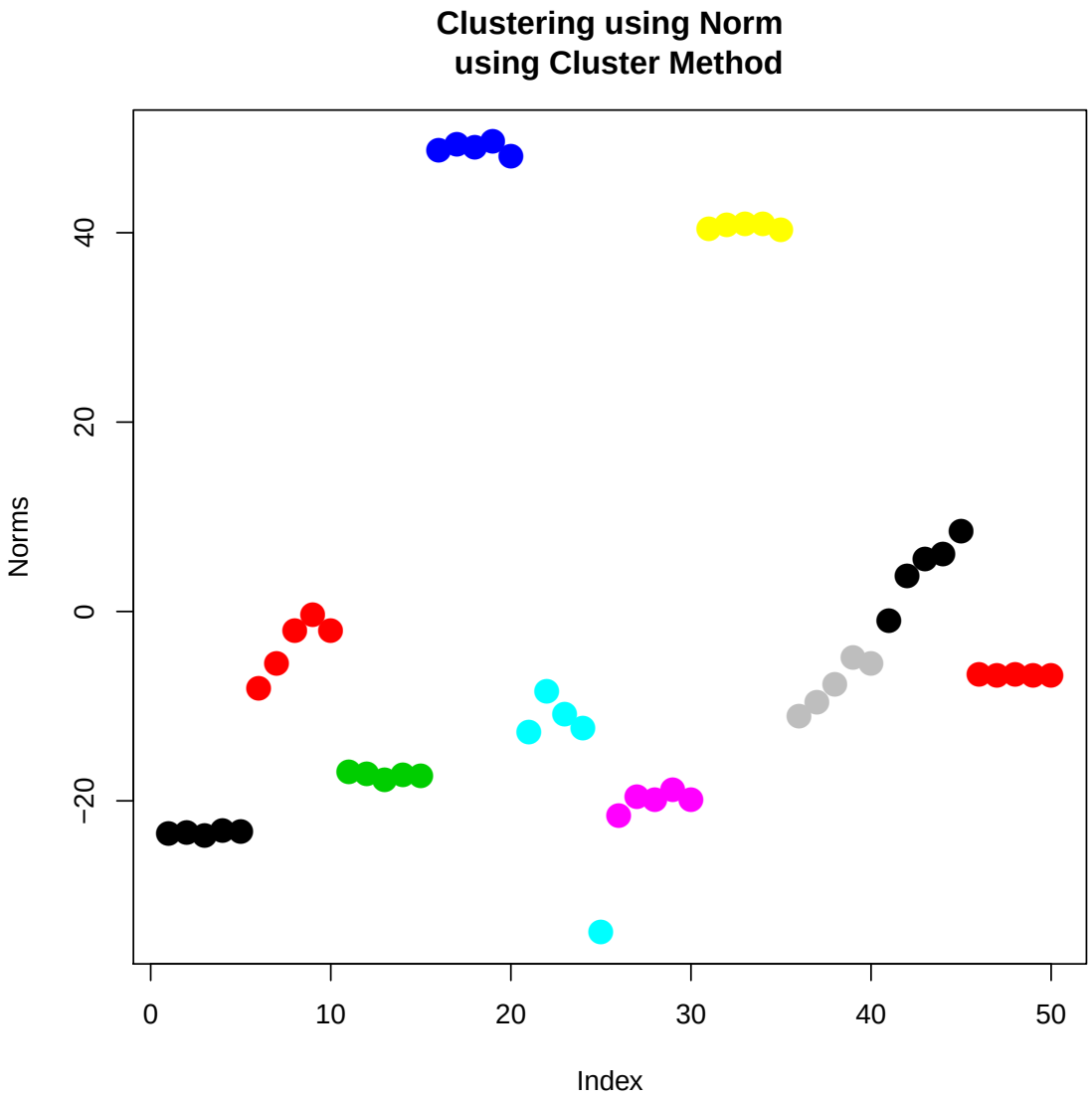
```
  plot(1:2,ty='n',main="0") # Window is created
  y = abs(newCoord$onb[,i]) # mod value of
  # components of vector from the onb
  extreme = range(y) # Range of the same
  y = (y-extreme[1])/(extreme[2]-extreme[1])
  # Normalizing to 0 to 1
  dim(y) = c(200,180) # making it a matrix,
  # to show as a picture
  rasterImage(as.raster(y),1,1,2,2)
}

# The next function reconstucts the faces
# from the principal components.
# It starts with the "mean face" and then
# gradually adds the details.
# You'll need to hit "enter" to step
# through the process.

showSteps = function(newCoord,i) {
  meanx = newCoord$centre
  # Data are extracted from newCoord
  Q = newCoord$onb
  scores = newCoord$scores
  values = newCoord$values
  expl = 100*cumsum(newCoord$values)/sum(newCoord$values)
  # The list of increasing accuracy of
  # photo while adding layers constructed
  # from eigen values of which eigen vector layer
  # is already added
  coeff = as.vector(scores[i,])
```

```
# Coefficients of picture in new eigen basis

plot(1:2,ty='n',main="0") # Window created
y = meanx
dim(y) = c(200,180)
rasterImage(as.raster(y),1,1,2,2)
# Mean picture plotted
readline()
# Enter is required to go to next step
for(k in 1:49) {
  if(k==1)
    temp = Q[,1]*coeff[1]
    # Image Layer corresponding to most
    # significant eigen vector
  else
    temp=Q[,1:k] %*% as.vector(coeff[1:k])
    # Image Layer corresponding to
    # most significant eigen vector remaining
  recons = meanx + temp # Image layer added
  recons[recons<0]=0
  # Hard coding pixel to 0 if value is < 0
  recons[recons>1]=1
  # Hard coding pixel to 1 if value is > 1
  dim(recons) = c(200,180)
  # making it a matrix, to show as a picture
  plot(1:2,ty='n',main=paste(k,":",
  values[k],", ",expl[k]))
  # Creating the window again
  rasterImage(as.raster(recons),1,1,2,2)
  # Showing the Image
```

```
    readline()
    # Waiting for enter to go to next step
  }
}


x = loadImages() # Images are loaded
newcoord = process(x) # Computations relate to PCA is done
```




Clustering by first 2 scores
using Cluster Method
Score 2
Score 1
20
15
10
5
0
−5
−10
−10
−5
0
5
10
15
20
25




```
# showSteps(newcoord,3) # Reconstruction of a Sample Imag
```

Hence we see the Clusters are separated but “distance” within Clusters are also minimised, which was our goal of the project!

5 Conclusion

We have seen the PCA method for dimension reduction and also the Clustered Data method, which serves our pur-

pose. After this stage, when a new picture is there, we can first apply our onb we got from PCA to get the 50 - dim scores (components) and then take first 10 and hence compute scores based on Clustered approach, then we can see the first two scores and see: if they are matching with some cluster, then we say the new picture belongs to the cluster, i.e. wee know whose picture it is; and if we do not get the any cluster which is close, we conclude the picture is not of anyone whose pictures we have. Hence we conclude our project.

6 Appendix

6.1 Lemma 1

Let $X_1, X_2, ...X_n$ be n random variables and $X = (X_1, X_2, ..., X_n)^T$. Let v be any vector. So, $v^T X$ is a linear combination of the given random variables. Then,

$$var(v^T X) = v^T \Sigma v$$

where Σ is the covariance matrix of $X_1, X_2, ...X_n$.

Proof: For any vector $v \in \mathbb{R}^n$, $v^T X = v_1 X_1 + v_2 X_2 + ... + v_n X_n$
$\therefore var(v^T X) = var(v_1 X_1 + v_2 X_2 + ... + v_n X_n)$
$= \sum_{i=1}^{n} \sum_{j=1}^{n} cov(X_i, X_j) v_i v_j = v^T \Sigma v$,
where Σ is the covariance matrix of $X_1, X_2, ...X_n$.

6.2 Lemma 2

For any quadratic form $x^{\mathrm{T}}\mathrm{Ax}$, and

$$\max_{\|x\|=1} x^T Ax = \max_{\mathbf{y}\neq 0} \frac{\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y}}{\mathbf{y}^{\mathrm{T}}\mathbf{y}} = \lambda$$

where λ is the largest eigenvalue of A. Moreover, $\frac{y^T Ay}{y^T y} = \lambda \Leftrightarrow y$ is an eigenvector of A corresponding to λ
Proof: Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be the characteristic roots of A and

$$\mathbf{A} = \lambda_1 \mathbf{u}_1 \mathbf{u_1}^{\mathrm{T}} + \lambda_2 \mathbf{u}_2 \mathbf{u}_2^{\mathrm{T}} + \cdots + \lambda_n \mathbf{u}_n \mathbf{u}_n^T$$

be a spectral decomposition of A. Since $u_1, \ldots, u_n$ form an ortho-normal basis of $\mathbb{R}^n$, any y can be written as $\mathbf{y} = \alpha_1 \mathbf{u}_1 + \cdots + \alpha_n \mathrm{u}_n$ for some α's and then $\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} = \lambda_1 \alpha_1^2 + \cdots + \lambda_n \alpha_n^2$ and $\mathbf{y}^{\mathrm{T}}\mathbf{y} = \alpha_1^2 + \cdots + \alpha_n^2$. So $\mathbf{y}^{\mathrm{T}}\mathrm{A}\boldsymbol{y}/\mathbf{y}^{\mathrm{T}}\mathbf{y} \leq \lambda_1$. Now let $k = \max\left\{j : \lambda_1 = \lambda_j\right\}$. If $k = n, \mathbf{A} = \lambda_1 \mathrm{I}$ and we are done. Next let $k < n$. Then each side of the equivalence in the theorem holds $\Longleftrightarrow \alpha_{k+1} = \cdots = \alpha_n = 0$, and so we are done.

6.3 Lemma 3

Show that the maximisation actually works in a layered way. In order words, show that the largest value for that quadratic form in Lemma 2 (with the added constraint that x is perpendicular to the subspace spanned by the eigenvectors corresponding to the largest eigenvalue) is the next largest eigenvalue of A. Moreover, $\frac{y^T Ay}{y^T y} = \lambda \Leftrightarrow y$ is an eigenvector of A corresponding to λ, where λ is the next largest eigenvalue.

Proof: The proof follows directly from the way we proved the previous theorem. The y has no component in u_1 as y is a vector orthogonal to the maximum eigen-space. So, $\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} = \lambda_2\alpha_2^2 + \cdots + \lambda_n\alpha_n^2$ and $\mathbf{y}^{\mathrm{T}}\mathbf{y} = \alpha_2^2 + \cdots + \alpha_n^2$. And, then we proceed as we did in the previous proof and we are done.

6.4 Lemma 4

Show that the nonzero eigenvalues of AB is same as the non-zero eigenvalues of BA. Proof: If v is a eigenvector of AB, then $ABv = \lambda v$, where λ is the corresponding eigenvalue.
Pre-multiplying both sides with B, we get,
$\implies BA(Bv) = \lambda(Bv)$
So, by the definition of eigenvalues, we get that λ is also an eigenvalue of BA.

6.5 Lemma 5

If v is an eigenvector of AB corresponding to non-zero eigenvalue λ, then Bv is an eigenvector of BA corresponding to λ.
Proof: The proof follows similarly from the previous theorem. We get
$ABv = \lambda v$
$\therefore$ Pre-multiplying B on both sides we get,
$\implies BA(Bv) = \lambda(Bv)$
So, again by the definition of eigenvalues, we get that Bv is an eigenvector of BA corresponding to λ.

6.6 Lemma 6

Let B be p.d. Then

$$\max_{x \neq 0} \frac{x^T A x}{x^T B x} = \lambda_{max}(B^{-1}A)$$

where $\lambda_{max}(B^{-1}A)$ is the largest eigenvalue of $B^{-1}A$. Also, the maximum is attained at x_0 iff x_0 is an eigenvector of $B^{-1}A$ corresponding to $\lambda_{max}(B^{-1}A)$.
Proof: Let, $\mu = \lambda_{max}(B^{-1}A)$ and $B = C^T C$ where C is non-singular. Writing $Cx = y$ we get

$$\max_{x \neq 0} \frac{x^T A x}{x^T B x} = \max_{y \neq 0} \frac{y^T (C^{-1})^T A C^{-1} y}{y^T y} = \lambda_{max}((C^{-1})^T A C^{-1})$$

Now the characteristic roots of $((C^{-1})^T A C^{-1})$ are all real and, by Lemma 4 are the same as the characteristic roots of $(C^{-1}(C^{-1})^T A) = (B^{-1}A)$. Hence, $\max_{x \neq 0} \frac{x^T A x}{x^T B x} = \lambda_{max}(B^{-1}A)$ follows. Also, $\frac{x_0^T A x_0}{x_0^T B x_0} = \mu$ iff Cx_0 is an eigenvector of $((C^{-1})^T A C^{-1})$ corresponding to μ which is equivalent to: x_0 is an eigenvector of $(B^{-1}A)$ corresponding to μ.

6.7 Lemma 7

Show that Lemma 6 works in Layered Way. The proof of this lemma follows directly from Lemma 3, Lemma 2 and Lemma 6

7 Reference

Face Data

Project Details

Geeks for Geeks: LDA