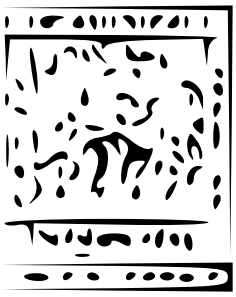

INDIAN STATISTICAL INSTITUTE

Project II - Vectors and Matrices

Application of Principal Component Analysis in facial recognition

Prepared by: Manas Sharma
Instructor: Arnab Chakraborty
Date: June 15, 2022

Abstract

This project concerns itself with trying to use the method of principal component analysis, which is very literal for a method to talk about the principal or in a certain sense, most important components to any data set.
This can be very useful since a lot of times when we have something that can be arranged in order of decreasing importance, we can at times remove the least important elements! This allows us a way to reduce the information required to represent the data, or what we formally would term as reducing the dimension. Hence intuitively, PCA tries to represent our data in terms of a basis, ordered importance wise. And then we can remove the least important elements if they tell us very little about our data set.
In this project, we first talk about PCA in a more abstract sense, though still clearly focused on a very important real life problem of reducing the amount of data required to represent something. Then we talk about what images are in digital terms, hence connecting our abstract notion of PCA to a possible real life application. Finally we talk about how to actually implement an actual face recognition process in R. The algorithm can similarly be transformed to suit any other programming language.

Contents

1 The Desire to See More with Less

1.1 Introduction

We shall start by a more informal understanding about PCA. Let's look at a nice two dimensional set.

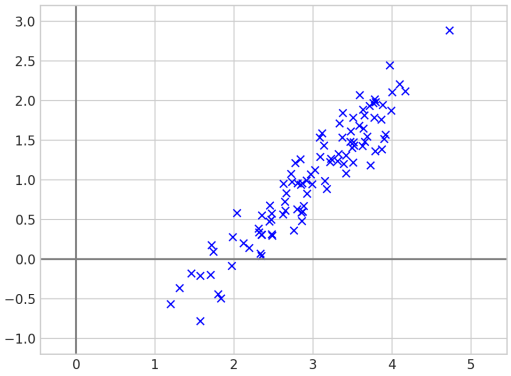

Figure 1.1: A random data-set

This data has a very clear direction of spread. We now digress slightly to give an analogy, I promise it's related. Let's talk about how we usually remember other people. When we wish to describe a person to someone who hasn't already seen them, it is almost absurd to talk about how similar they look to the general populace. What you will remember of course is in what manner they were different.

Figure 1.2: Three people all wearing glasses

The above three people are clearly different people. So when you describe them, using gender or physical characteristics is clearly the way to go and not whether they wear glasses or whether they are wearing a shade of blue. In essence, to talk about these three different people in the most efficient manner, you want to talk about where they differ the most.

Coming back to the data, its intuitively clear that we get quite a good description of our data, even if we just take one direction, if that direction is along the maximum variance. This line tells us quite a lot about our data!

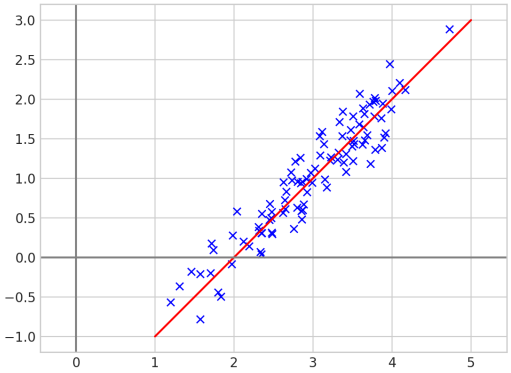

Figure 1.3: Principal Direction

In fact this line is the essence of PCA, If this line is the direction of maximum variance, then clearly the line perpendicular to it will represent the direction of least variance. And as we know, two linearly independent vectors are enough to represent the whole 2-D space, the direction of maximum variance and least variance are together the so called principal components! With decreasing order of significance to the data. We shall in the next part,we formally define what we mean by direction of maximum variance and even give an interesting form for it.

1.2 A Tale of PCA, Quadratic Forms, and Variation (A lot of Variation)

In this subsection, we shall tread with much more rigour. Let's start with a set of N random variables each being observed M times. Clearly, this is most naturally represented in a $M \times N$ matrix with each row corresponding to a random variable. It can also be seen as N points in an M dimensional space. Say all these random variables have zero mean, other wise, we just shift our space by the mean vector.

$$X_{M\times N} = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \cdots & \cdots & \cdots \\ x_{1M} & \cdots & x_{MN} \end{bmatrix}$$

We now present a very interesting result, so fundamental, it is one of the major reasons for the study of quadratic forms.

Theorem 1.1. *For any coefficient vector v and a random vector Y (all entries i.i.d) both with dimension n,*

$$Var(v^T Y) = v^T \Sigma v$$

where Σ is the covariance matrix of the random vector Y.

Proof. Now $v^T Y = (v_1 Y_1 + \cdots + v_n Y_n)(1)$. We know by properties of variance,

$$Var(\sum_{l=1}^{m} a_l) = \sum_{l=1}^{m} Var(a_l) + \sum_{i \neq j}^{m} Cov(a_i, a_j)(2)$$

Now applying (2) to (1), we get

$$var(v^T Y) = \sum_{l=1}^{n} Var(v_l Y_l) + \sum_{i \neq j}^{m} Cov(v_i Y_i, v_j Y_j) \tag{1}$$

$$\implies var(v^T Y) = \sum_{l=1}^{n} v_l^2 Var(Y_l) + \sum_{i \neq j}^{m} v_i v_j Cov(Y_i, Y_j)$$

This now translates directly into the definition of a quadratic form.

Since Σ is a symmetric matrix with i^{th} diagonal entry $Var(Y_i)$ and i, j^{th} off-diagonal entry $Cov(Y_i, Y_j)$. Hence,

$$Var(v^T Y) = v^T \Sigma v$$

□

This results in a very direct extension which we mention as a lemma. We will only provide a sketch for the proof since it is nothing but calculations.

Corollary 1.1.1. *For any coefficient vector v with dimension M and a matrix $X_{M \times N}$ with columns $X_1, \cdots, X_N$ as the points.*

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

where Σ is the covariance matrix corresponding to the rows of the matrix $X_{M \times N}$

Proof. We can clearly associate each row as a single random variable with N observations. Hence Σ is just the covariance matrix of these random variables. □

This now sets us up for some interesting discoveries. Let's first talk about the topic I promised you in this section. Variance in a direction. There are many way to interpret how something varies along a direction, but probably the most intuitive way would be to

talk about the normal variance for the orthogonal projections along that direction which are clearly scalars. The next image illustrates this.

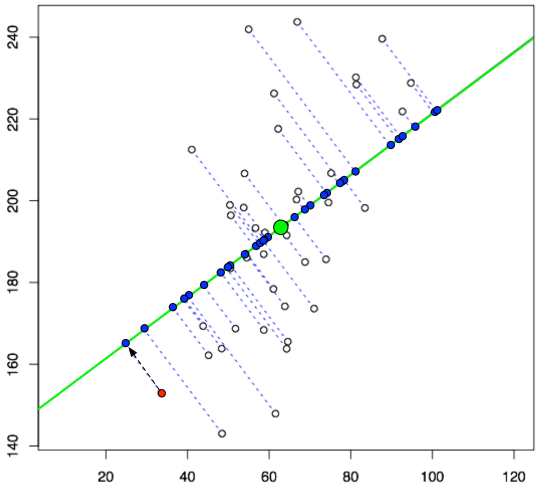

Figure 1.4: Projections along a direction

Since it makes intuitive sense, we define for our purposes, the variance in a direction as the variance of the orthogonal projections on that direction vector.

Where we know the orthogonal projection would be ,

$$Proj(X_i, v) = \frac{< X_i, v >}{\|v\|}$$

(Here $< X_i, v >$ is the dot product between X_i and v.) For generality, we shall now assume $\|v\| = 1$.

Stating this formally, with my own personal convention,

Definition 1. *We define* ***variance in a direction*** *for a set of points encapsulated in a matrix X in the direction of vector v where $\|v\| = 1$, as*

$$\sigma_v^2 = Var(v^T X_1, \cdots, v^T X_N)$$

where $X_1, \cdots, X_N$ are columns of $X_{M \times N}$,

Can you see the connection yet to previously proved facts? The **Corollary 1.1.1.** shows us a way to talk about dot products. Matrix multiplication has a certain interpretation as being the dot product between the first matrix's rows and the second matrix's columns. Hence by the the **Definition 1.** and **Corollary 1.1.1.**, we write;

$$\sigma_v^2 = v^T \Sigma v$$

Hence now our wish to find our principal components, each explaining something about the data, independently from the others in an order of obvious descent.

1.3 When Varying from the Mean is Good in Statistics!

As the title of this subsection states, and you might have already inferred, when we wish to find our principal components, we will do so by maximizing the σ_v^2 from amongst all v where $\|v\| = 1$ as we stated before. The v that gives us this is our foremost principal component, it is the direction that tells us the most about our data since it is the direction where the deviation from the general(i.e the mean) is the most. Just like we saw before with a simple example in **Fig 1.3**.

To this end we prove our next lemma,

Lemma 1.2 (Maximization of the Quadratic Form)**.** *For a symmetric matrix A with order n with eigenvalues $\lambda_n \leq \cdots \leq \lambda_1$ and vector v with same dimensions,*

$$\max_{\|v\|=1} v^T A v = \lambda_1$$

and this happens $\iff$ v is an eigenvector corresponding to the eigenvalue λ_1.

Proof. We know by Spectral Decomposition, every real symmetric matrix can be decomposed into a product of P^T, D and P, where P^T's columns denote the eigenvectors of A making an ONB of $\mathcal{C}(A)$ and D is a diagonal matrix with A's eigenvalues as it's entries. Say without loss of generality they are ordered as $\{\lambda_1, \cdots, \lambda_n\}$. Also call the i^{th} column of P as e_i Hence $\{e_1, \cdots, e_n\}$ is an ONB for $\mathbb{R}^n$.

Let's substitute A as $P^T D P$ in $v^T A v$. We get

$$v^T A v = v^T P^T D P v.$$

Calling $Pv = u$, we have $v^T A v = u^T D u$. Expanding this, we get

$$u^T D u = \sum_{i=1}^{n} \lambda_i u_i^2$$

(Here u_i is the i^th component of u.) Now by assumption, $\sum_{i=1}^n u_i^2 = 1$, And clearly $0 \leq u_i^2$, $\forall i \in \{1, \cdots, n\}$. Hence we have $v^T A v$ as a convex combination of λ_is therefore its maximum value is clearly at λ_1 and the minimum is at λ_n. Now, $\sum_{i=1}^n \lambda_i u_i^2 = \lambda_1 \implies$ that all u_is $= 0$ except for i $= 1$. Hence $\forall$i$\neq$1, $(Pv)_i = 0$. Hence $v = e_1$ where e_1 is the eigenvector corresponding to λ_1 □

This is by itself an absolutely amazing result! But we aren't done yet. Since the set of eigenvectors in P is an ONB, it remains an ONB under truncation, just of a smaller subspace. In fact this subspace is the orthogonal complement of the span of the removed elements intersection with the span of the original ONB. Hence if I remove the vector v_1 from the ONB, $u_1 = 0$ always!. Hence we can see that for this subspace, the maximal value of $v^T A v$ will now be λ_2.

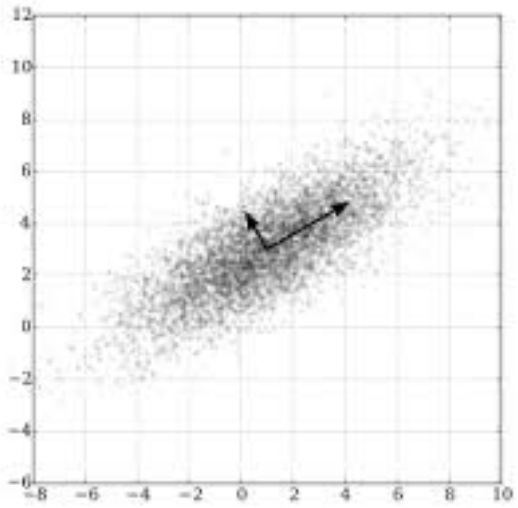

Figure 1.5: The less significant direction

This results in an amazing insight. The maximization works in a layered way, one after the other. We first get e_1 as the vector inducing the maximal value for $v^T Av$. But if we remove its effect from the space, reducing it to a subspace, the orthogonal complement of e_1 to be precise, then the maximal value of $v^T Av$ will change to λ_2 which has the second highest eigenvalue.

Let's state this formally in the following corollary without proof since we have already discussed a sketch of it.

Corollary 1.2.1 (Maximization is layered)**.** *For a symmetric matrix A with order n with eigenvalues $\lambda_n \leq \cdots \leq \lambda_1$ and vector v with same dimensions, Say e_1 is a vector corresponding to lambda$_1$, then,*

$$\max_{\substack{\|v\|=1 \\ v \in e_1^{\perp}}} v^T Av = \lambda_2$$

and this happens $\iff$ v is an eigenvector corresponding to the eigenvalue λ_2.

The above two statements are immensely useful as one might be able to see from above thought process that we have kept throughout. We wish to find components independent from each other's effect and that also maximize σ_v^2 over all v such that $\|v\| = 1$. And the above lemma and its corollary give us exactly what we want! σ_v^2 is also of the form $v^T Av$ and hence its maximization problem is solved in exactly the same manner!

We have now reached the final frontier for this Section. In the final subsection, we state in all it's glory, the actual method of principal analysis!

1.4 Revealing the Facade!

We now state the actual method in all formality and while most of you have guessed, also explain the actual benefit of doing all this!

Theorem 1.3 (**Principal Component Analysis**). *If the following process is to be followed to obtain* $\{e_1, \cdots, e_N\}$ *then, this aforementioned set of vectors is an ONB with each* e_k *maximizing* σ_v^2 $\forall v \in span\{e_1, \cdots, e_{k-1}\}^\perp$. *We shall call* $\{e_1, \cdots, e_N\}$ *as the principal components.*

1. *Take your* $X_{M\times N}$ *as your data matrix with columns as points.*

2. *Shift the points by the mean to change the affine space(subspace shifted by a vector) back to the subspace.*

3. *Calculate the covariance matrix for rows of X. It gives you an* $M \times M\Sigma$ *which is symmetric.*

4. *Find an ONB eigenbasis for* Σ.

5. *Label this ONB as* $\{e_1, \cdots, e_N\}$

Proof. The proof is the easily done and is the result of all the effort that we have gone through. By **Corollary 1.1.1** we have that σ_v^2 is a quadratic form with the matrix Σ.**Lemma 1.2** and **Corollary 1.2.1**, then give us the desired result! □

As I promised, I will indeed state the use that we teased in the start. **Dimensional Reduction**!. The capability to represent an almost similar amount of information with much less degrees of freedom? Are you ready? Will you be able to handle it? So here it is!

To reduce the information required to represent the data, just truncate the principal component set from the end suitably.

What? Disappointed? You really shouldn't be. It's what I promised. "Dimensional Reduction". I reduced the dimension of the principal component set. **Tada!**

Honestly, I am not really joking here. You can see it intuitively, if the amount of variance they represent decreases as we move forward in the set. So the last few elements, for a data that's supposed to lower dimensional than what it is in its representation, the last few vectors would contribute almost zero variance to the net data. So we can actually remove them!

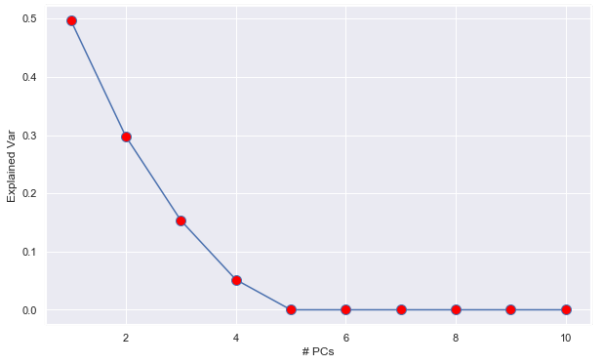

Figure 1.6: Variance for each direction

Look at this photo above, We see in the direction of the final vector, there is barely any change. That vector can be clearly discarded and we can still represent almost the whole data-set in the 2 dimensional subspace. This is what PCA boils down to in the end.

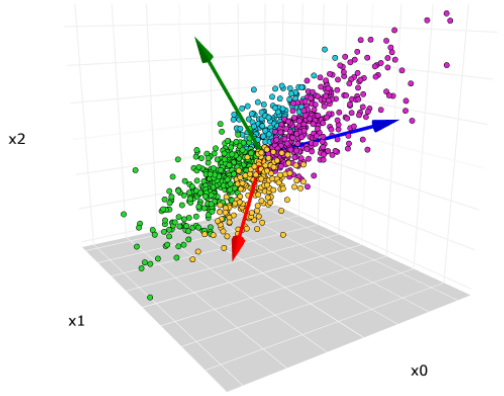

Figure 1.7: Three Dimensional Example

This is the fact that makes PCA so beautiful to me personally. It is the capability to visualize every step that we take. As we have done till now. But its time to physically visualize these steps in a concrete setting. Let's talk about how we do recognize faces with PCA.

2 Seeing Patterns in Gray

In this section, we deal with shades of gray. Especially 256 of them. Because that's how many values a pixel can take in our current setting, each for a shade of gray in essence. Let's look at the picture of one of the people we have in our data-set. A 180X200 pixel photo to be exact.

Figure 2.1: anpage

In this section we talk through the way of R code! It is a language I hope you are familiar with. Because the method gains even more clarity with an example. Let's start. Go through it slowly. Code, especially written by an amateur like me is generally very hard to read. But the start should be relatively easy since it's not my code, it's my instructor's. And he is way better of a teacher than I will ever be.

```
library(jpeg) #We should have jpeg already installed
#------
loadImages = function() {
  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)
  #------
  k = 0

  for(i in 1:10) {
    for(j in 1:5) {
      k = k + 1
      #The first argument below should be the location of the folder.
      filename = paste("./assets/",name[i],".",j,".jpg",sep="")
      x[k,] = as.vector(readJPEG(filename))
    }
  }
  #------
  return(x)
```

```
}
#-----------------
#Next, we perform PCA. This is exactly what I mentioned above!

process = function(x) {
  meanx = apply(x,2,mean)

  y = scale(x,scale=F) #Rows of y are the cases
  A = y %*% t(y)

  #To calculate eigenvalues of Y'Y, we calculate eigenvalues of YY'!
  #Because by a very standard result,
  # non zero eigenvalues of AB and BA are equivalent --- (Fact 1)


  eig = eigen(A)

  P = t(y) %*% eig$vec[,-50]

  #Columns of P are e vectors of Y'Y

  Q = apply(P,2,function(x) x/sqrt(sum(x*x)))

  #Columns of Q form onb for rowspace of Y


  scores = y %*% Q
  #scores[i,] is for i-th image
  return(list(centre=meanx,onb=Q,scores=scores,values=eig$values))
}
#-----------------
#Each principal component is again a 200*180 vector, and hence 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")
  y = abs(newCoord$onb[,i])
  extreme = range(y)
  y = (y-extreme[1])/(extreme[2]-extreme[1])
  dim(y) = c(200,180)
  rasterImage(as.raster(y),1,1,2,2)
}
#-----------------
```

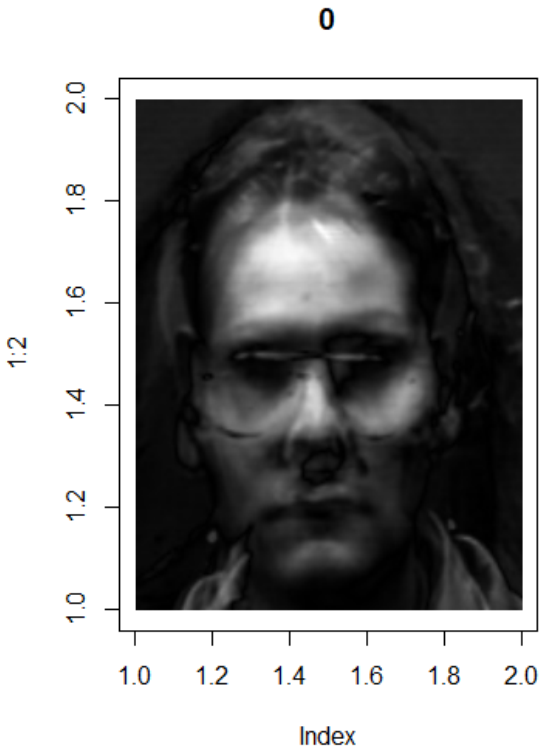

Figure 2.2: The first principle component as a picture!

2.1 What people varying the most look like

This is an interesting picture, even though it's definitely not what I hope a person looks like (Or the power of my eyes has really gone up) But it tells what varies the most for the data! One of the clear winners in that aspect are spectacles! Hair is also a serious contender here. Both make sense. This allows us to actually see how our variance maximization is working out

```
#The next function reconstructs 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$centre
  Q = newCoord$onb
  scores = newCoord$scores
  values = newCoord$values

  coeff = as.vector(scores[i,])

  plot(1:2, ty='n', main="0")
  y = meanx
  dim(y) = c(200,180)
```

```
  plot(as.raster(y))
  readline()
  for(k in 1:49) {
    if(k==1)
      temp = Q[,1]*coeff[1]
    else
      temp=Q[,1:k] %*% as.vector(coeff[1:k])
    recons = meanx + temp
    recons[recons<0]=0
    recons[recons>1]=1
    dim(recons) = c(200,180)
    plot(as.raster(recons))
    readline()
  }
}

filename = "./assets/male.pacole.1.jpg"
tmp=readJPEG(filename)

rasterImage(tmp[90:130,70:115],-3,0,3,25)

x = loadImages()
newcoord = process(x)
showSteps(newcoord,26)

pc1 = newcoord$onb[,1]
showFace(newcoord,1)
expl = 100*cumsum(newcoord$values)/sum(newCoord$values)
# This checks the explanation
plot(expl,type = 'l')
print(expl[10]) #96.77246
print(expl[9]) #95.67942
print(expl[8]) #93.60762

#Hence in the spirit of alpha = 0.05, we choose 9 principal components!

princ_scores = newcoord$scores[,1:9] #9 principal components matrix
```

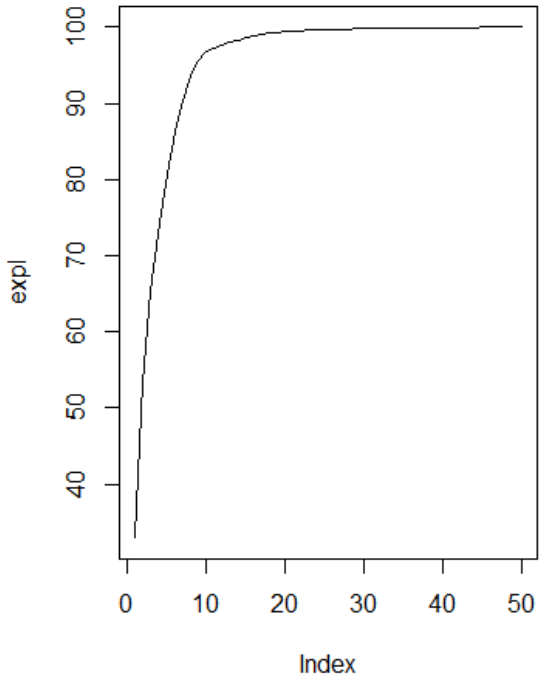

Figure 2.3: How much each component explains!

2.2 How to actually find people(Cluster Analysis)

Its clear how the curve starts plateauing at 9 values, and hence we chose 9 as our number of principal components. Now there is a new problem. We have managed to reduce our data to a $50/times9$ matrix. From 36000×36000, this is a huge leap. But there is an issue. We now wish to write a facial recognition software. All that we did before was dimension reduction. But to do dimension reduction, we have to compare. Compare with each photo, or each photo cluster of size 5 which we know. This is arduous. And this becomes a real pain when the number of clusters increase.

Wouldn't it be great if we could somehow do PCA on clusters? Well our original PCA doesn't work for example in the picture below.

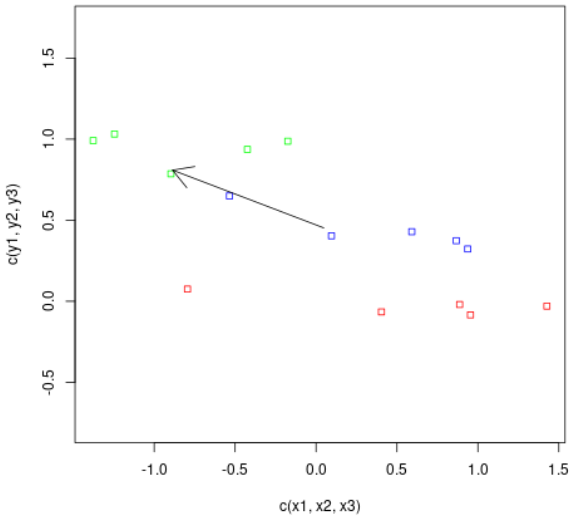

Figure 2.4: Small cluster example!

So if we could somehow make a matrix for covariance within the clusters W and between the cluster B, then maximizing $\frac{x^TBx}{x^TWx}$ where W is positive definite would hopefully maximize the covariance between cluster and minimize the covariance within, simultaneously!

So to work towards that, let's prove a couple lemmas!

Lemma 2.1. *Let W be p.d. Then*

$$\max_{x\neq 0} \frac{x^TBx}{x^TWx} = \lambda_{max}(W^{-1}B)$$

where $\lambda_{max}(W^{-1}B)$ is the largest eigenvalue of $W^{-1}B$. Also, the maximum is attained at x_0 iff x_0 is an eigenvector of $W^{-1}B$ corresponding to $\lambda_{max}(W^{-1}B)$.

[Maximization for between cluster variance]

Proof. Let, $\Lambda = \lambda_{max}(W^{-1}B)$ and $W = C^TC$ where C is non-singular. Writing $Cx = y$ we get

$$\max_{x\neq 0} \frac{x^TBx}{x^TWx} = \max_{y\neq 0} \frac{y^T(C^{-1})^TBC^{-1}y}{y^Ty} = \lambda_{max}((C^{-1})^TBC^{-1})$$

Now the characteristic roots of $((C^{-1})^TBC^{-1})$ are all real and, by **Fact 1** are the same as the characteristic roots of $(C^{-1}(C^{-1})^TB) = (W^{-1}B)$. Hence, $\max_{x\neq 0} \frac{x^TBx}{x^TWx} = \lambda_{max}(W^{-1}B)$ follows. Also, $\frac{x_0^TBx_0}{x_0^TWx_0} = \mu$ iff Cx_0 is an eigenvector of $((C^{-1})^TBC^{-1})$ corresponding to Λ which is equivalent to: x_0 is an eigenvector of $(W^{-1}B)$ corresponding to Λ. □

Now like **Corollary 1.2.1**, we wish to prove layering of this cluster variance maximization as well! In fact it almost direct from **Corollary 1.2.1** and **Lemma 2.1** by the same way

we derived **Corollary 1.2.1** from **Lemma 2.1**. Hence we state the layering property as a corollary without a proper proof.

Corollary 2.1.1 (Maximization is layered)**.** *Let W be p.d. Then*

$$\max_{x\neq 0 x\in x_0^{\perp}} \frac{x^T B x}{x^T W x} = \lambda_{2^{nd}max}(W^{-1}B)$$

where $\lambda_{2^{nd}max}(W^{-1}B)$ is the second largest eigenvalue of $W^{-1}B$. Also, the maximum is attained at x_0 iff x_0 is an eigenvector of $W^{-1}B$ corresponding to $\lambda_{2^{nd}max}(W^{-1}B)$.

```
#--------
clust_size = 5
clust_num = 10
Within_Matrices = list()
#List of all W_i matrices
#(Which calculate variance within cluster of photos)
for (i in 1:clust_num) {
  j = clust_size*(i-1)
  Within_Matrices[[i]] = cov(princ_scores[(j+1):(j+clust_size),])
}
W = (clust_size-1)*Reduce('+',Within_Matrices)/(clust_num*clust_size-1)

#W is our wanted within matrix!

Between_Vectors = list()
#List of all means of vectors of each cluster
for (i in 1:clust_num) {
  j = clust_size*(i-1)
  Between_Vectors[[i]] = apply(princ_scores[(j+1):(j+clust_size),],2,mean)
}
mean_mat = Between_Vectors[[1]] #Mean matrix
for (i in 1:clust_num) {
  mean_mat = rbind(mean_mat, Between_Vectors[[i]])
}
B = cov(mean_mat)

#B is our between matrix!

W_inv = solve(W)

#W inverse. Non singular since ONB with non zero eigenvalues
#We write next function to do PCA in essence with matrice W-1B!

cluster_ana = function(x) {
  A = W_inv %*% B
  eig = eigen(A)
  Q = apply(eig$vec,2,function(x) x/sqrt(sum(x*x)))
  #Columns of Q form onb for rowspace of Y
```

```
  clust_scores = x %*% Q
  return(clust_scores)
  #clust_scores is the score matrix for cluster positions!
}

#This is the function with stores ranges!
rl = list() #List with range vectors as entries
for (i in 1:clust_num) {
  rl[[i]] = list(r1 = range(scores_clust[i,1]), r2 = range(scores_clust[i,2
}

rl
scores_clust = cluster_ana(princ_scores)
scores_clust

#Final function to check if scores lie in a certain area!
cluster_check = function(x) {
  scores = cluster_ana(x)
  cv = scores[1:2]
  for (i in 1:clust_num){
    if
    ((rl[[i]]$r1[1]<=cv[1])&&
    (cv[1]<=rl[[i]]$r1[2])&&
    (rl[[i]]$r2[1]<=cv[2])&&
    (cv[2]<=rl[[i]]$r2[2])){
      print(i)}
    else {
      next}
  }
}

#Checks for first image! We are done!
cluster_check(princ_scores[1,])
```

3 Ending Note

PCA is one of those visualizable ML techniques that fills my heart with joy! Honestly I have discussed everything I wished to discuss. But I have a slight final note to leave.

Regression and PCA are different!

Look at this. We do interpret PCA as minimizing the distance of the point from the direction line. But that's orthogonal distance, not vertical! So it is indeed confusing when we talk about regression and PCA together. But here's a great picture to clear your doubts!

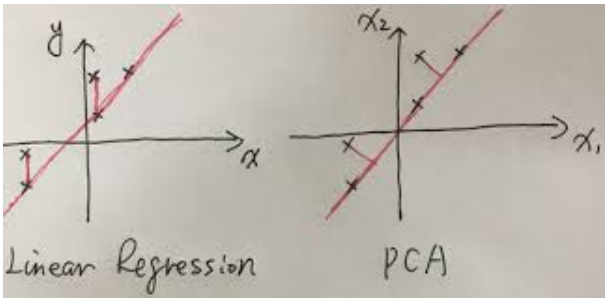

Figure 3.1: Definitely not confusing :p

[1] [2]

References

[1] A. Chakrabarty, "PCA in terms of R Code," April 2022. [Online]. Available: https://www.isical.ac.in/~arnabc/talks/eigen.r

[2] A. Chakraborty, *Method to download and extract the image dataset in code*, April 2022. [Online]. Available: https://www.isical.ac.in/~arnabc/talks/eigface.html

List of Figures