Principal Components Analysis

-Applied Multivariate Analysis & Statistical Learning-

MMA chapter 12.1 to 12.8 and ISLR 10.2

Lecturer: Darren Homrighausen, PhD

Preamble:

- Give some geometric intuition for PCA
- Briefly overview how PCA is motivated/computed
- Go through examples such as digits and faces

Overview

Representation learning is the idea that performance of ML methods is highly dependent on the choice of data representation

For this reason, a lot of work is dedicated towards transforming the data into the relevant features and then using these as inputs

This idea is as old as statistics itself, really,

(E.g. Pearson (1901), where PCA was first introduced)

However, the idea is constantly revisited in a variety of fields and contexts

Overview

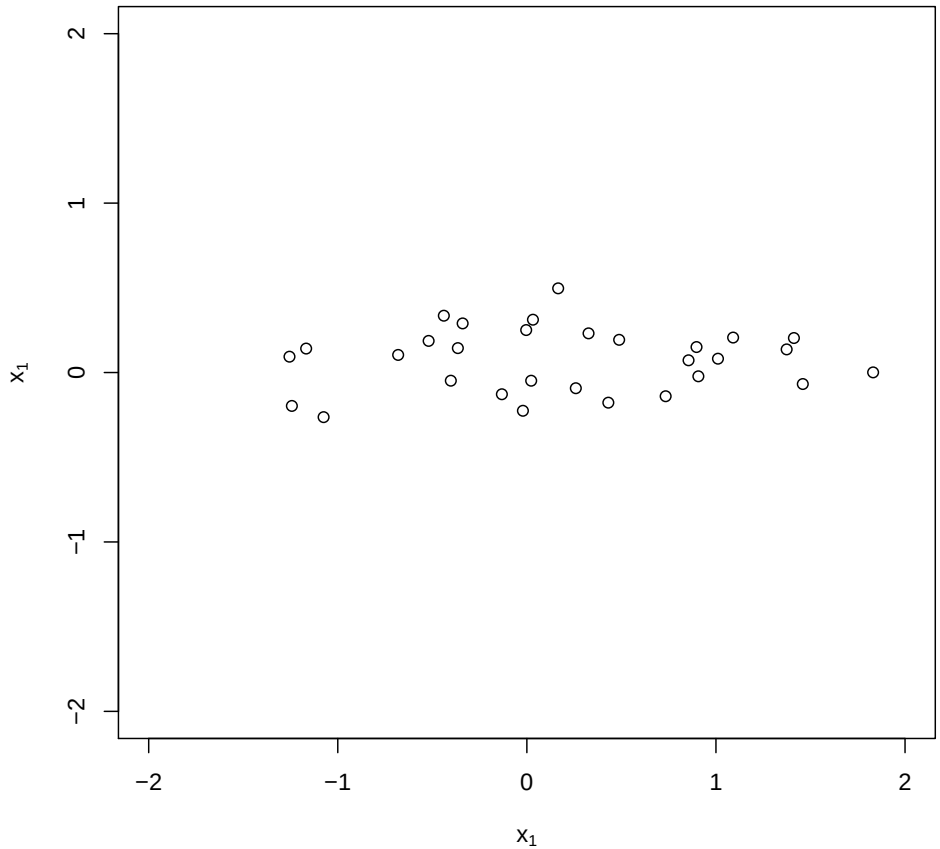
Commonly, these learned representations capture 'low level' information like overall shape types

Other sharp features, such as edges in images, aren't captured

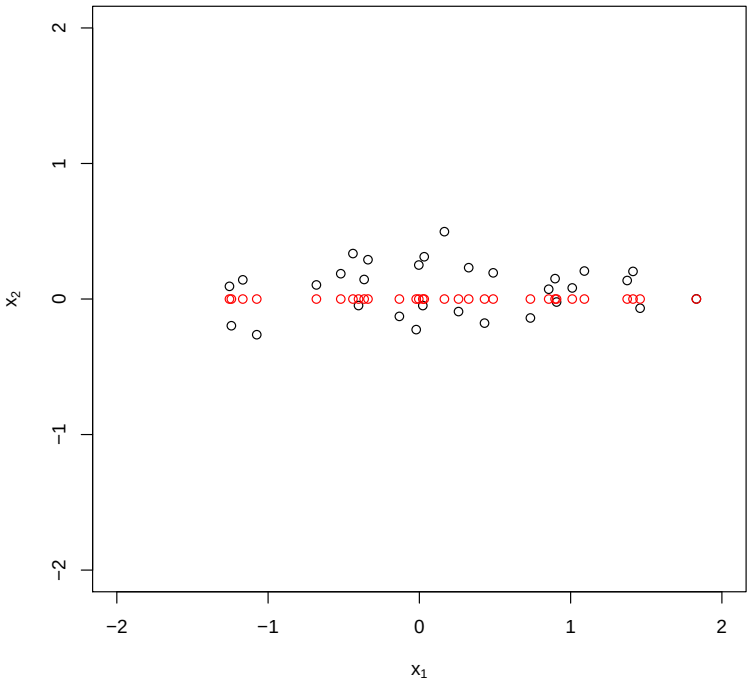
It is possible to quantify this intuition for PCA at least

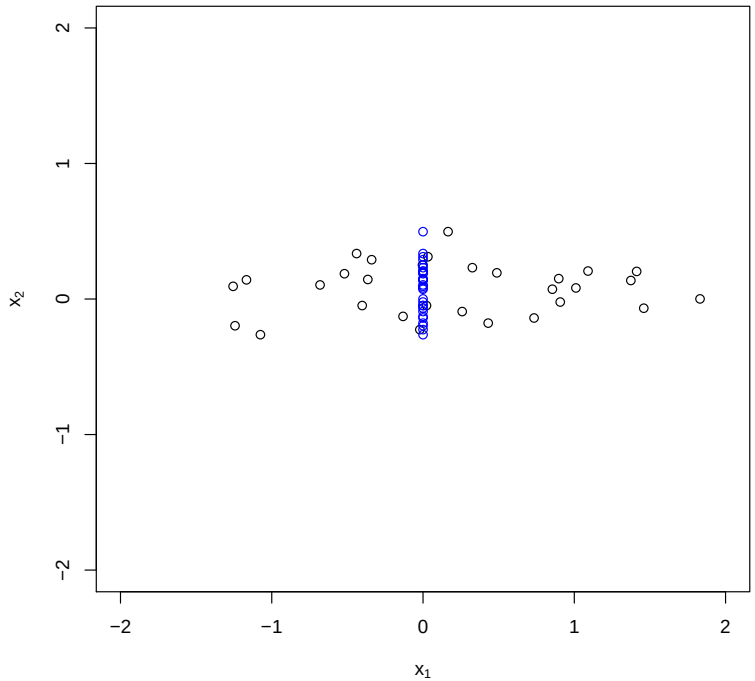
First Example

Suppose I want to keep either the feature x_1 or x_2



First Example

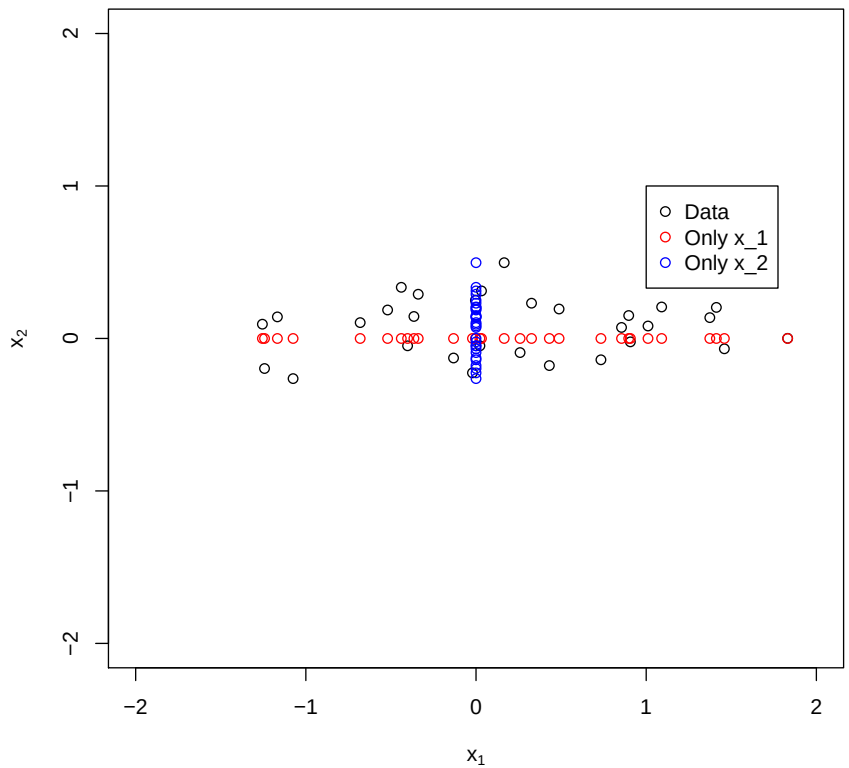




First Example

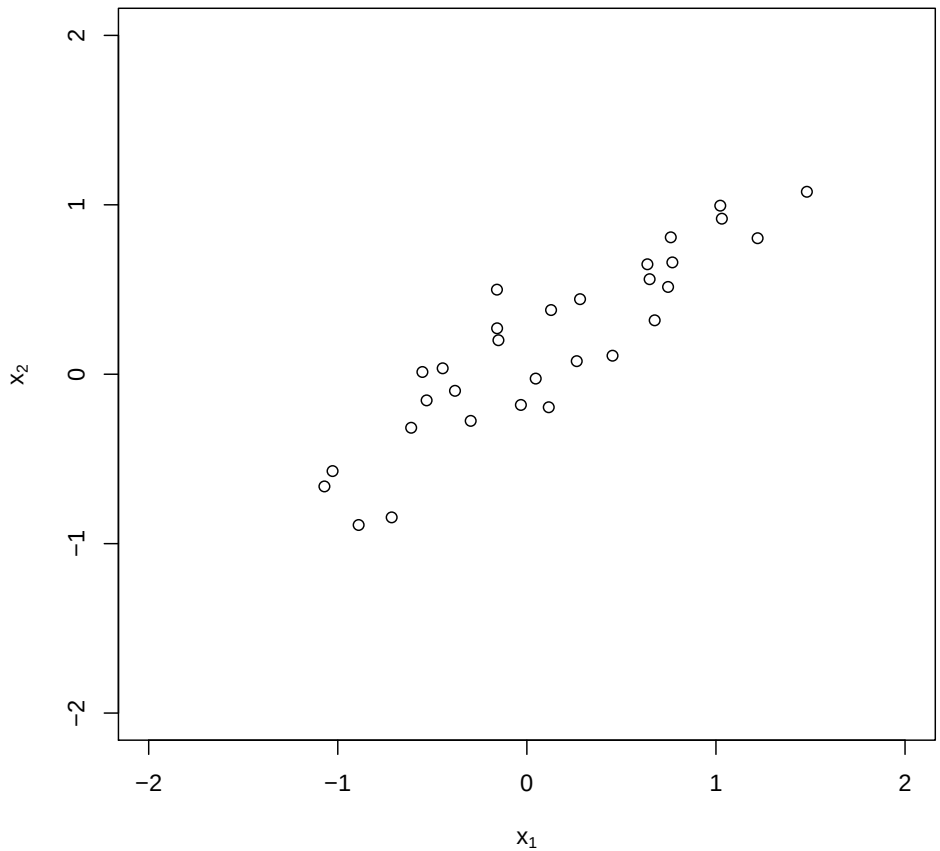
We can see that

- We more faithfully preserve the structure of the data by keeping x_1 and setting x_2 to zero than the opposite
- We don't lose that much structure by setting x_2 to zero.

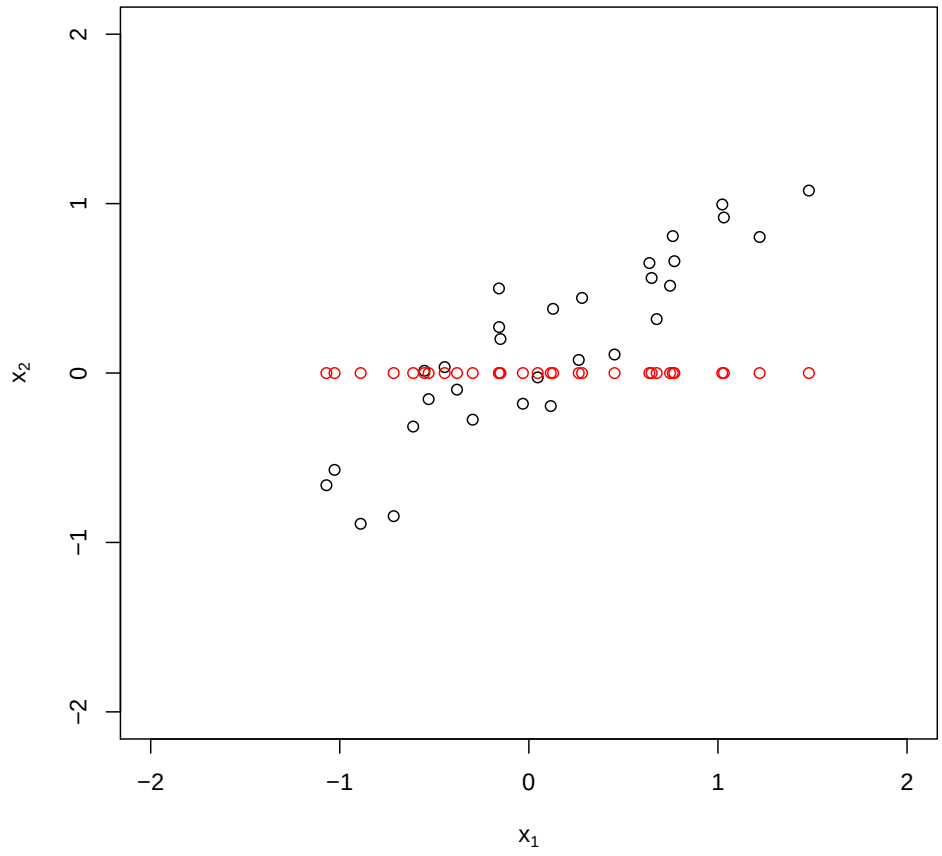


Second Example

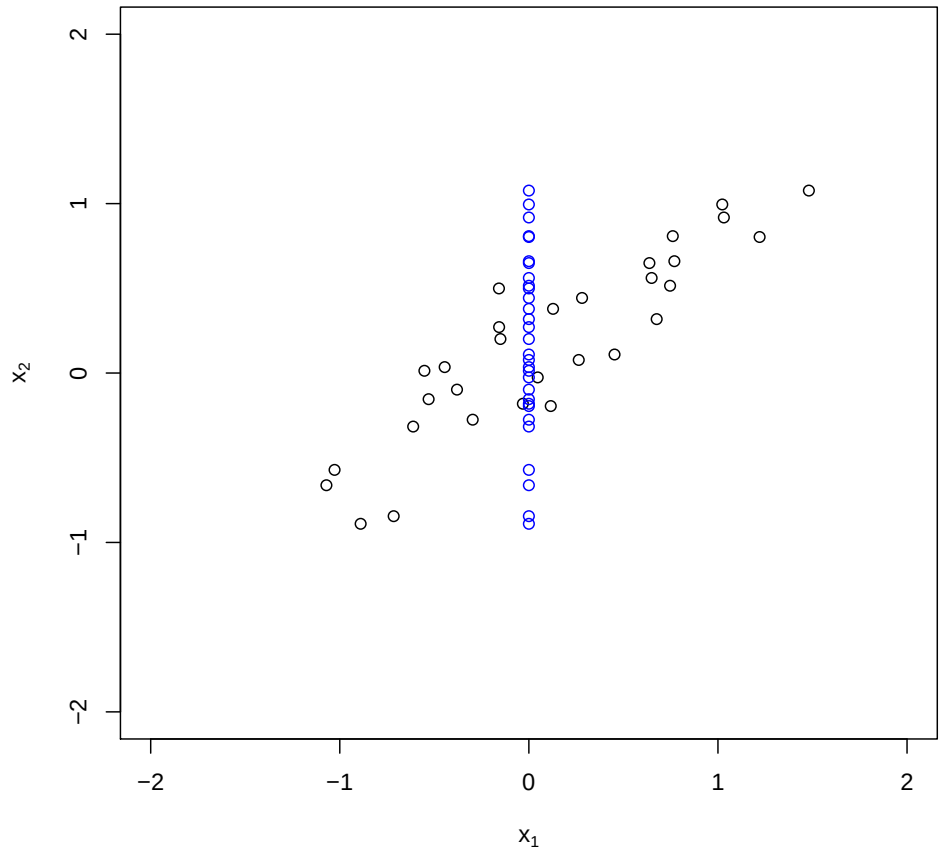
An important aspect of the First Example is that x_1 and x_2 aren't correlated with each other. What if they are correlated?



Second Example

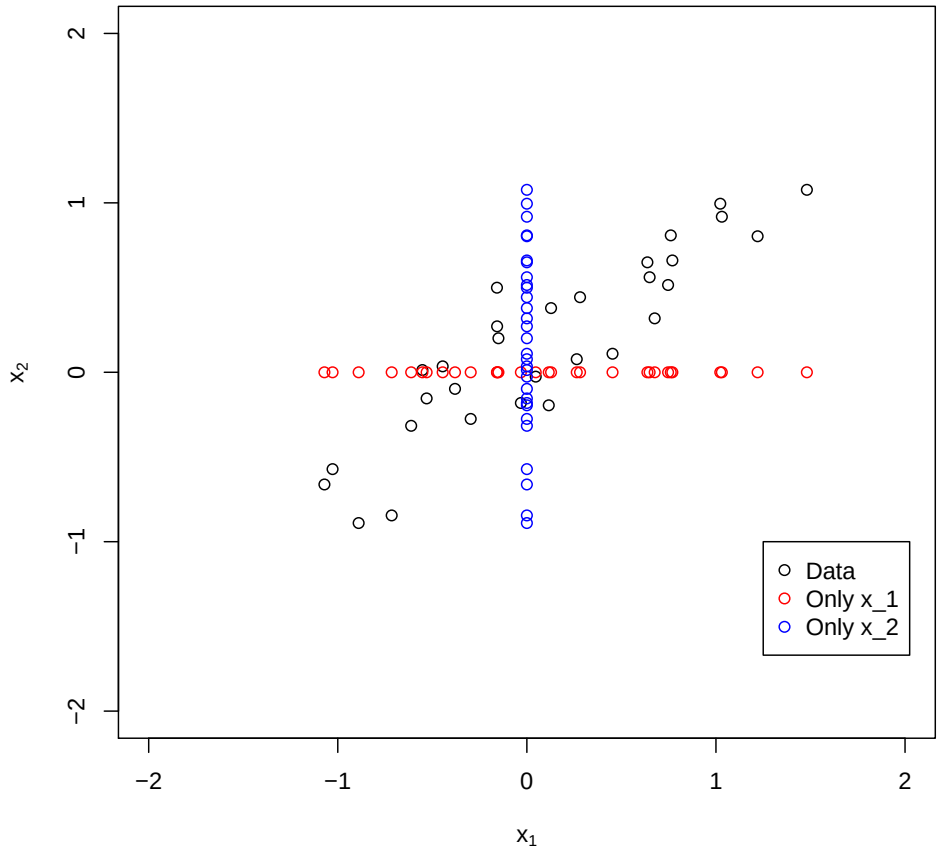


Second Example



Second Example

Eliminating either feature incurs substantial information loss

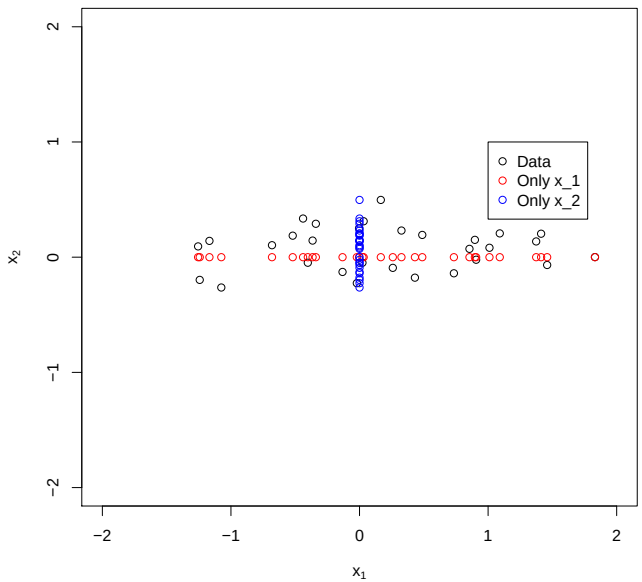


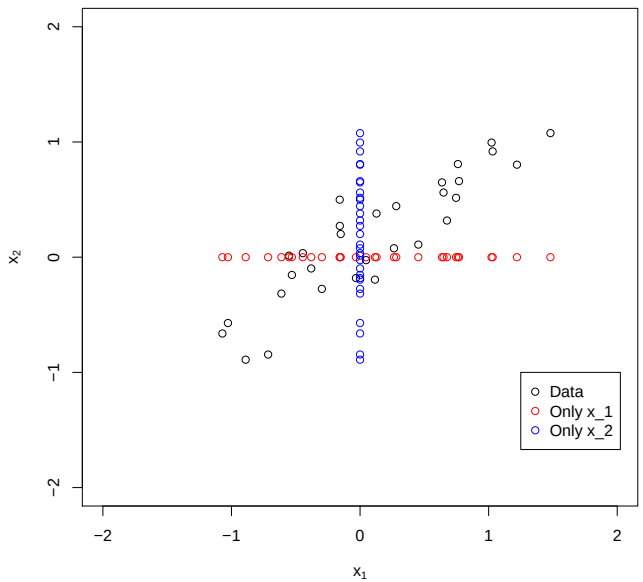
Comparing the examples

There isn't that much structurally different between the First and Second Examples

In fact, the Second Example is a rotation of the First Example.

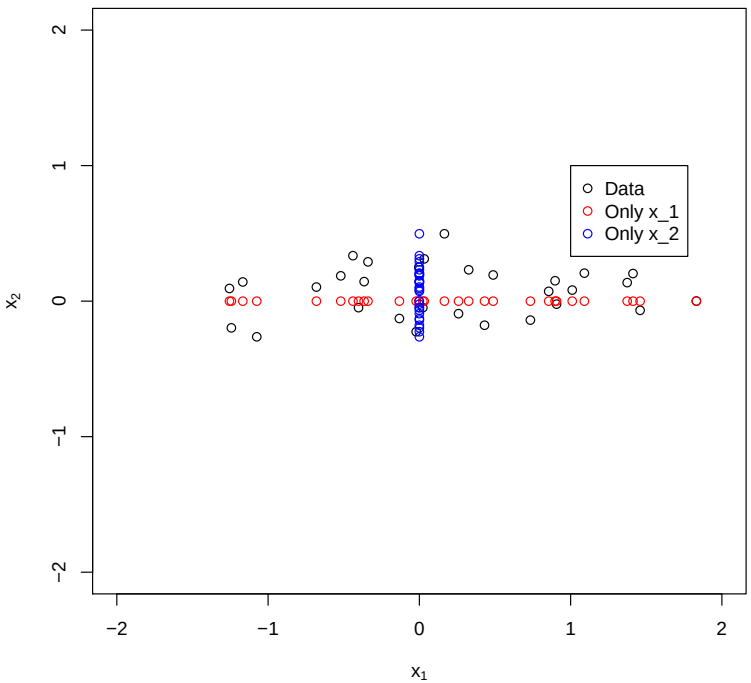
(Look back at the SVD notes, rotation in this sense means the same as for orthogonal matrices)

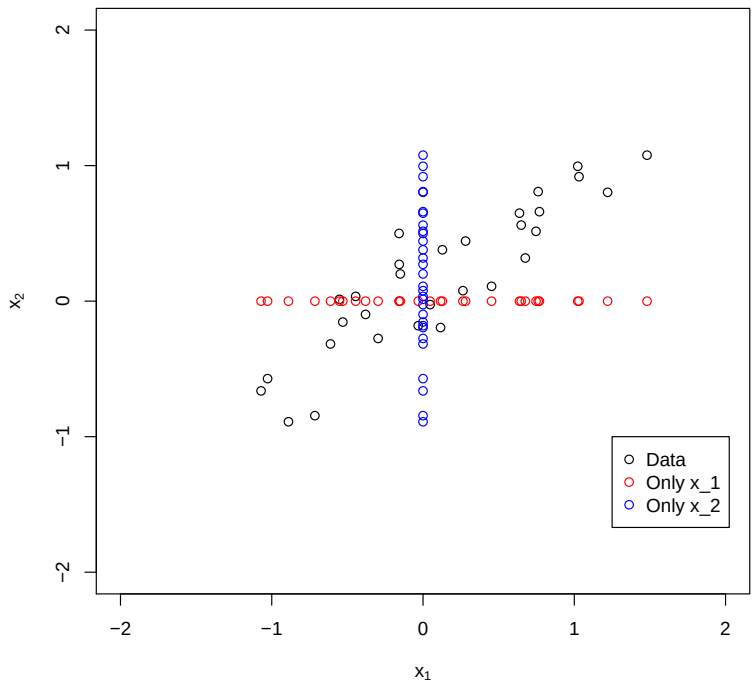




Comparing the examples

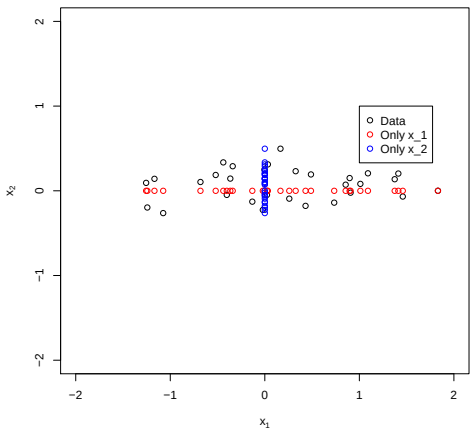
If we knew how to rotate our data so that the Second Example looked like the First Example, we would be able to preserve more structure after reducing dimensions

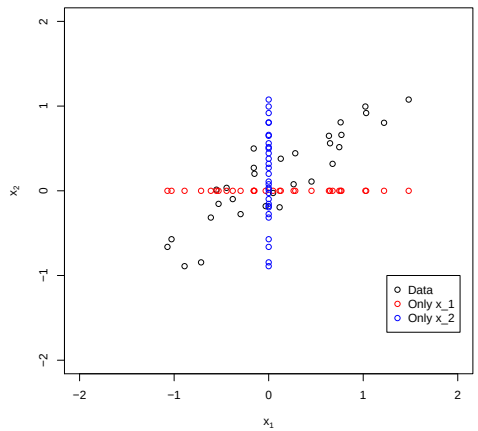


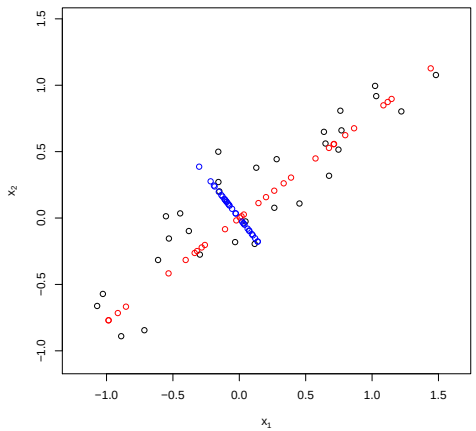


PCA

It turns out that Principal Components Analysis (PCA) gives us exactly this rotation.







Principal components analysis (PCA)

Principal components analysis (PCA)

PCA is an (unsupervised) dimension reduction technique

It solves various equivalent optimization problems

(Maximize variance, minimize ℓ_2 distortions, find closest subspace of a given rank,...)

At its core, we are finding linear combinations of the original (centered) observations

$$Z_{ij} = \alpha_j^\top X_i$$

such that

$$\alpha_1 = \underset{||a||_2=1}{\arg\max}\, \mathbb{V}[a^\top X_i]$$

$$\alpha_2 = \underset{||a||_2=1, a^\top \alpha_1 = 0}{\arg\max}\, \mathbb{V}[a^\top X_i]$$

$$\vdots$$

How to compute it?

PCA can be computed by getting the SVD of the column centered (and usually scaled) feature matrix

$$\mathbb{X} - \overline{\mathbb{X}} = UDV^\top$$

In this case, if we seek to retain Q PCs, then

- The $Z_Q = U_Q D_Q$ are the first Q PC (scores). Written another way, we have

$$Z_j = (\mathbb{X} - \overline{\mathbb{X}})v_j = u_j d_j$$

- and V_Q is the rotation

  (The entries in the matrix V_Q are usually called the loadings)

If we want to find the first Q principal components, the relevant optimization program is:

$$\min_{\mu,(\lambda_i),V_Q} \sum_{i=1}^{n} ||X_i - \mu - V_K \lambda_i||^2$$

(Here, V_Q is restricted to be an orthogonal matrix)

This representation shows that we are trying to reconstruct lower dimensional representations of the features

PCA

$$\min_{\mu,(\lambda_i),V_Q} \sum_{i=1}^{n} ||X_i - \mu - V_Q\lambda_i||^2$$

We can partially optimize for μ and (λ_i) to find

- $\hat{\mu} = \overline{X}$
- $\hat{\lambda}_i = V_Q^\top(X_i - \hat{\mu})$

We can find

$$\min_V \sum_{i=1}^{n} \left|\left|(X_i - \hat{\mu}) - VV^\top(X_i - \hat{\mu})\right|\right|^2$$

where V is constrained to be orthogonal

(This is the so called Steifel manifold of rank-Q orthogonal matrices)

PCA in R

PCA in R

We can compute PCA in R either via the built in function:

```
PCA.out = prcomp(X,scale=TRUE)
```

(Only use prcomp, not princomp. Much more numerically stable!)

Or, we can directly use the SVD

```
svd.out = svd(scale(X,scale=TRUE))
```

PCA in R

```
PCA.out = prcomp(X,scale=TRUE)
> names(PCA.out)
[1] "sdev"     "rotation" "center"   "scale"    "x"
> dim(X)
[1] 100  10
> dim(PCA.out$rotation)
[1] 10 10
> dim(PCA.out$x)
[1] 100  10
```

The coordinates of...

- the observations are in PCA.out$x (that is, the Z)

  (Known as scores)
- the features are in PCA.out$rotation (That is, the V)

  (Known as loadings)

PCA in R

```
> PCA.out$rotation[1:2,1:3]
            PC1         PC2        PC3
[1,] -0.3797434 0.007642462 -0.3559232
[2,] -0.2505855 0.479266913 -0.1575462
> PCA.out$x[1:2,1:3]
            PC1        PC2        PC3
[1,] -1.3056426 -0.5296034 -0.9157294
[2,] -0.3535175  0.5285959  1.4482028
> svd.out$v[1:2,1:3]
            [,1]        [,2]       [,3]
[1,] -0.37974339 0.007642462 -0.3559232
[2,] -0.25058548 0.479266913 -0.1575462
> (UD = svd.out$u %*% diag(svd.out$d))[1:2,1:3]
            [,1]        [,2]       [,3]
[1,] -1.3056426 -0.5296034 -0.9157294
[2,] -0.3535175  0.5285959  1.4482028
```

Scaling the features

If we do either

```
PCA.out = prcomp(X,scale=TRUE)
```

or

```
svd.out = svd(scale(X,scale=TRUE))
```

we need to decide whether to scale the features

(Important: always center the features)

As a general rule, scale if the features are measured in different units

Digits example

Digits example

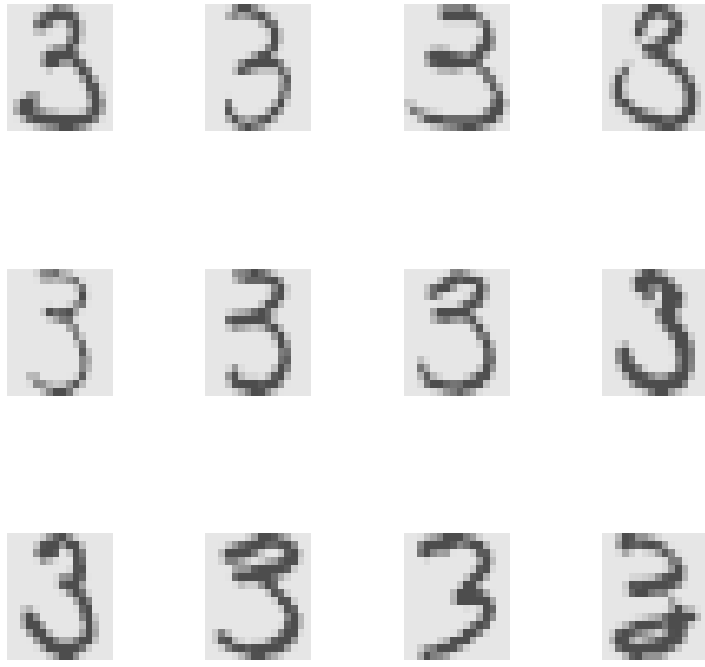


Figure: http://www-stat.stanford.edu/∼tibs/ElemStatLearn/

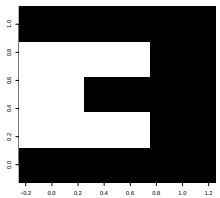
658 handwritten 3's, each drawn by a different person

Each image is 16x16 pixels, each taking values between -1 and 1.

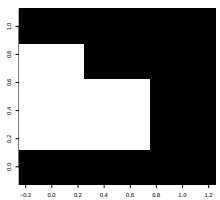
Digits example: How does this fit with previous examples?

Think about each pixel location as a measurement

Consider these simple drawings of 3's. We convert this to an observation in a matrix by unraveling it along rows

$X_1 =$ 

$$X_1 = [1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1]^\top$$

$X_2 =$ 

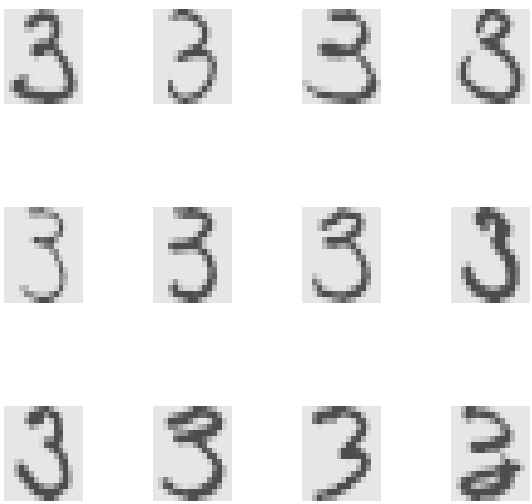
$$X_2 = [1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1]^\top$$

(Here, let black be 1 and white be 0)

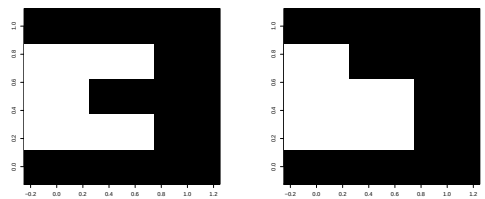
Digits example

We will consider digits with...

- more pixels ($p = 256$)
- a continuum of intensities



Vs.



Sample plotting code

```
plot.digit = function(x,zlim=c(-1,1)) {
  cols = gray.colors(100)[100:1]
  image(matrix(x,nrow=16)[,16:1],col=cols,
        zlim=zlim,axes=FALSE)
}
```

Digits example

Eventually, we will learn how to classify these digits. But, for now, let's look at all 658 digits via PCA

```
load("../data/digits.Rdata")

threesCenter = scale(threes,scale=FALSE)
svd.out = svd(threesCenter)

pcs    = svd.out$v
scores = svd.out$u%*%diag(svd.out$d)
```

Or, using prcomp:

```
out = prcomp(threes,scale=F)
pcs = out$rot
scores = out$x
```

(Note that here we aren't scaling: the measurements are already on a consistent scale)

Digits example

We can plot the scores of the first two principal components versus each other:

```
plot(scores[,1],scores[,2],xlab = 'PC1',ylab='PC2',
    main='Plot of First Two PCs')
```

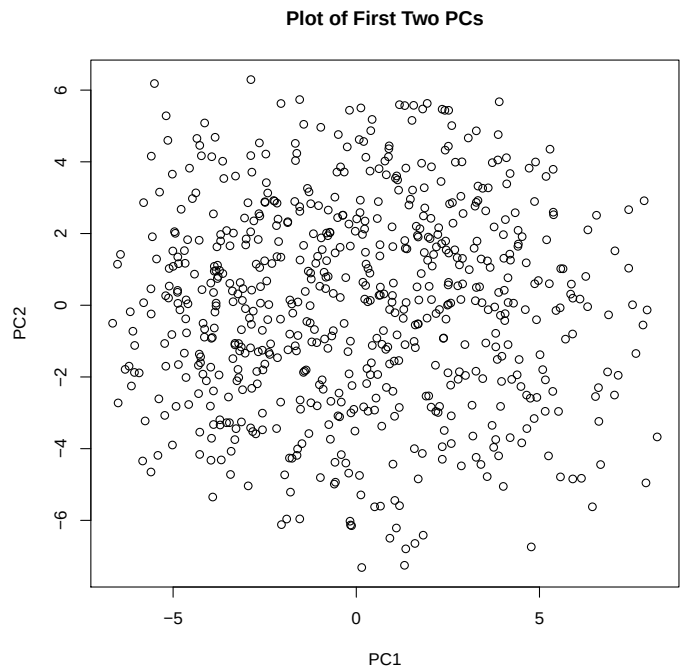
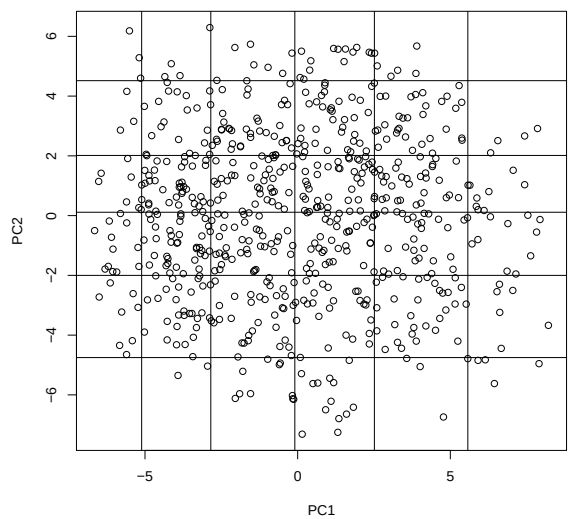


Figure: Each circle in this plot represents a hand written '3'.

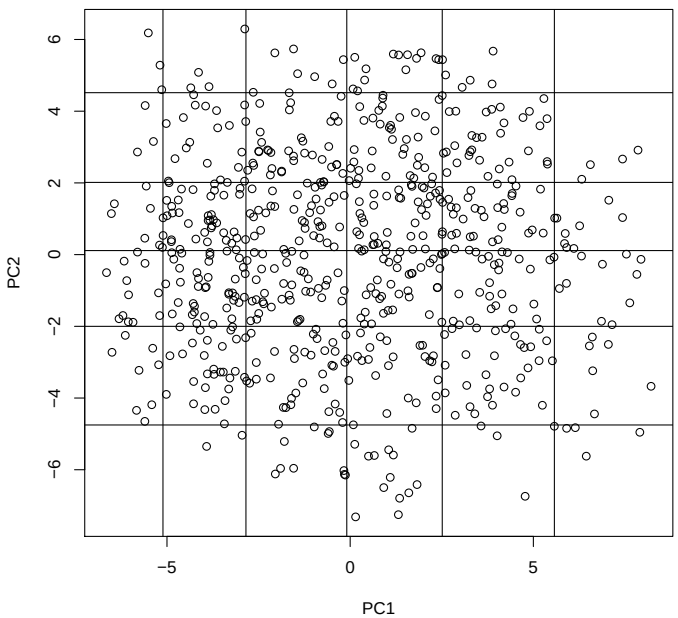
Digits example

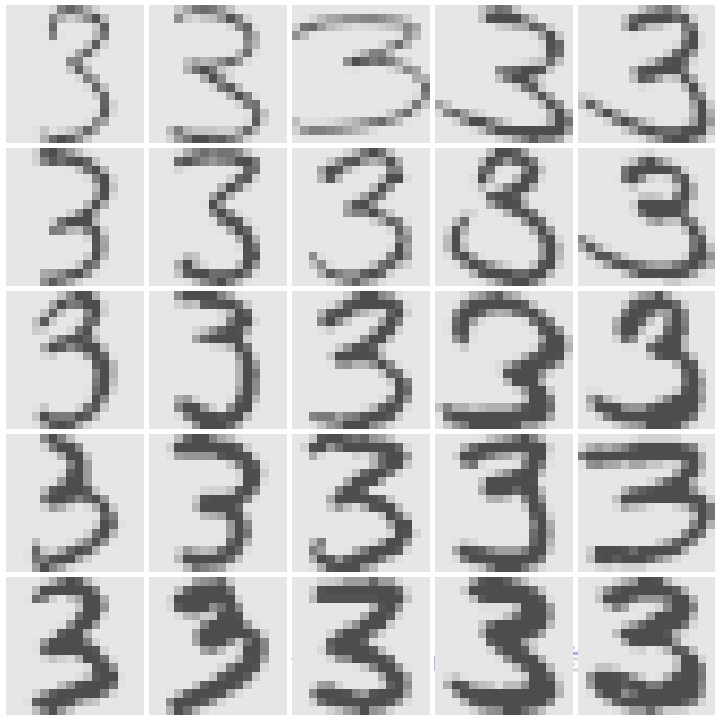
```
quantile.vec = c(0.05,0.25,0.5,0.75,0.95)
quant.score1 = quantile(scores[,1],quantile.vec)
quant.score2 = quantile(scores[,2],quantile.vec)
plot(scores[,1],scores[,2],xlab = 'PC1',ylab='PC2')
for(i in 1:5){
  abline(h = quant.score2[i])
  abline(v = quant.score1[i])
}
identify(scores[,1],scores[,2],n=25) #to find points
```



Digits example

```
pcs.order = c(73,238,550,82,640,284,84,133,4,322,392,241,
    554,220,500,247,344,142,405,649,184,149,234,375,176)
par(mfrow=c(5,5))
par(mar=c(.2,.2,.2,.2))
for(i in pcs.order){
  plot.digit(threes[i,])
}
```

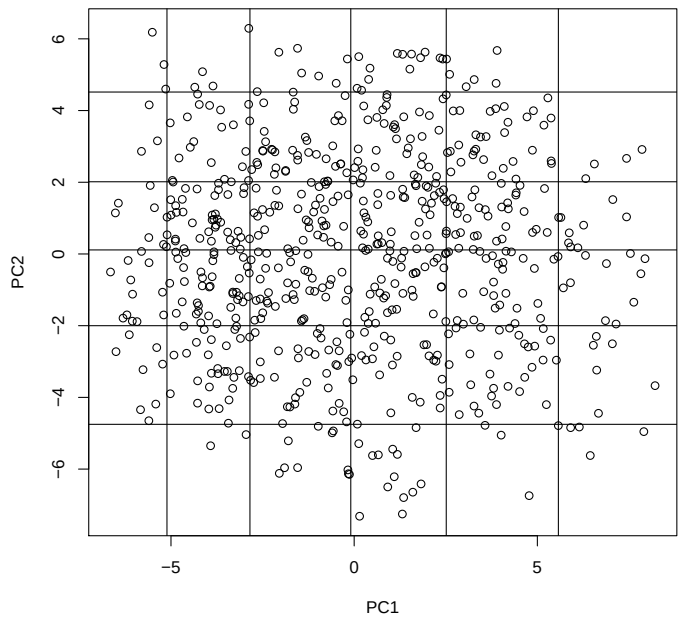


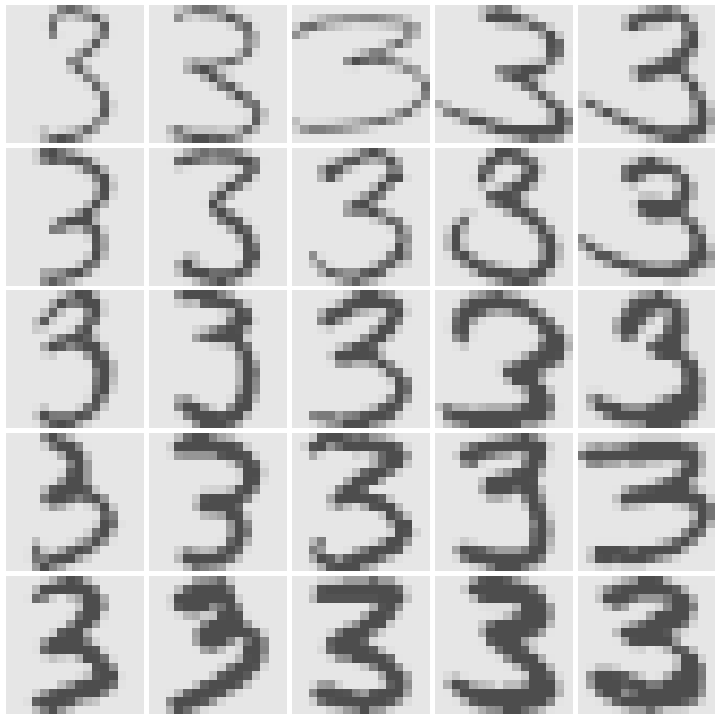


Digits example

The 3's get lighter as the location on PC2 increases.

The 3's get more elongated bottom swoops as the location along PC1 increases (also, the 3's tend to get wider)





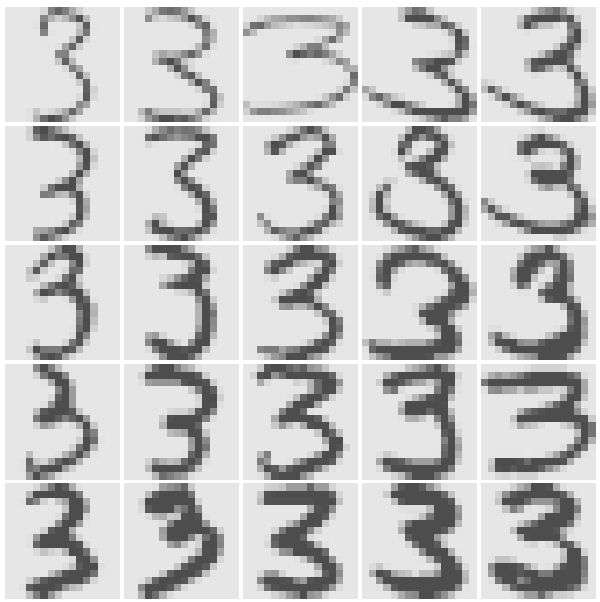
Digits example: How many dimensions?

Each digit is represented by a vector in $\mathbb{R}^{256}$

(as each square is 16x16 pixels)

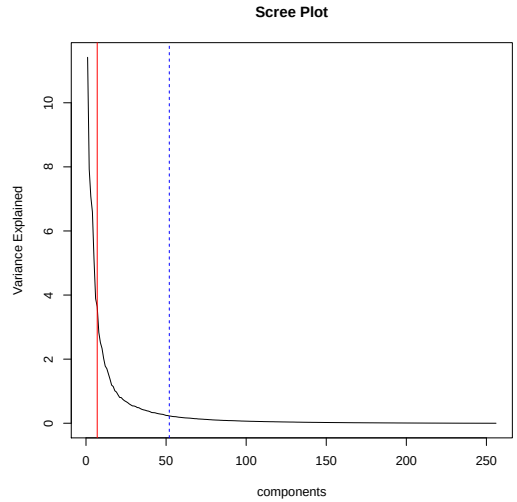
However, hopefully we can reduce this number by re-expressing the digits via PCA

(For instance, the top-right pixel is always 0 and hence that feature is uninteresting)



Digits example: How many dimensions?

Let's look at a scree plot

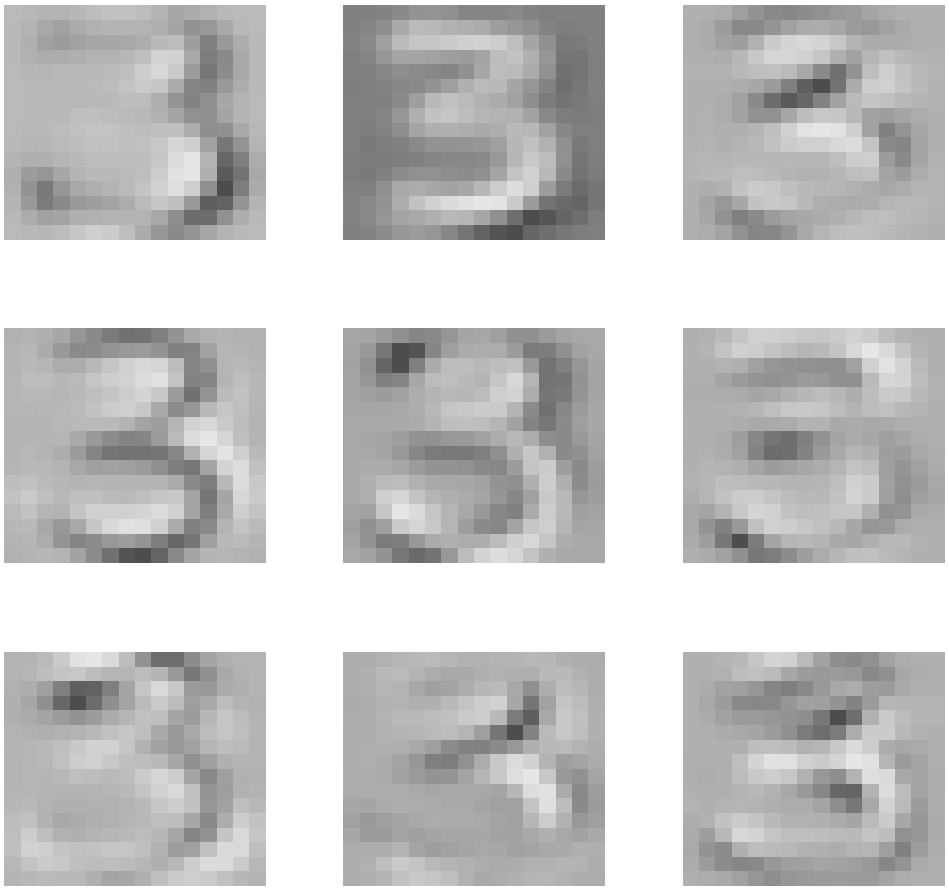


We put vertical lines when 50% and 90% of the variance has been explained (at 7 and 52 PCs, respectively)

```
> min(which(cumsum(out$sdev**2/sum(out$sdev**2))>.5))
[1] 7
> min(which(cumsum(out$sdev**2/sum(out$sdev**2))>.9))
[1] 52
```

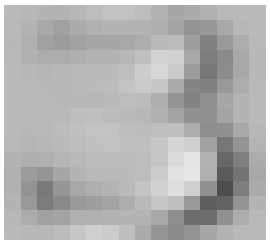
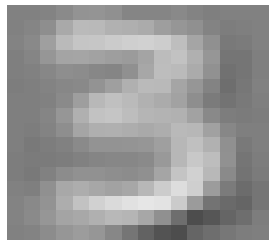
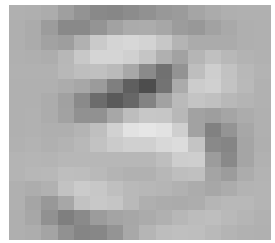
Digits example: Loadings

Lastly, we can also look at the loadings as well:

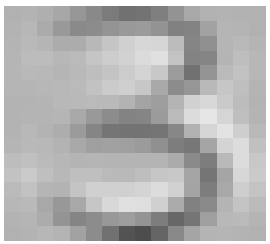
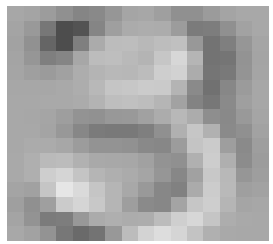
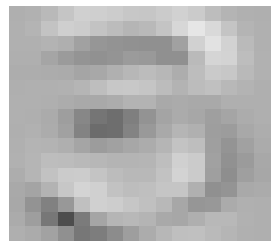


Digits example: Loadings

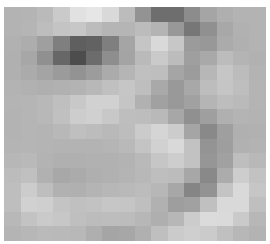
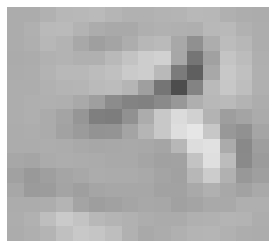
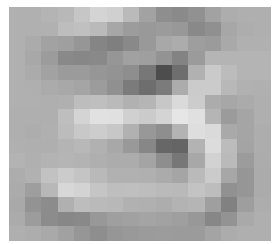
Lastly, we can also look at the loadings as well:

1st PC: Takes a compact 3 and smears it out

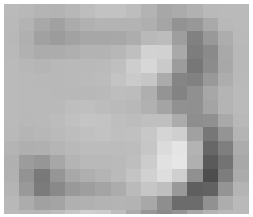
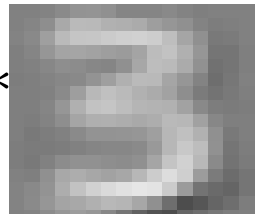
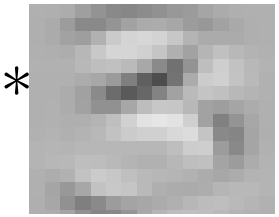




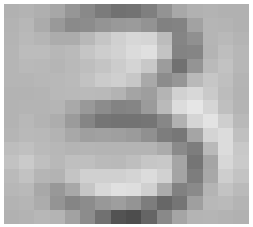
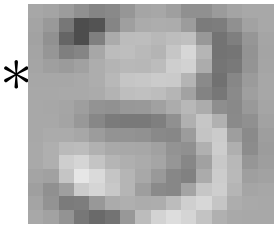
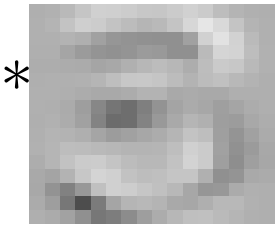
2nd PC: Deletes a portion of the inner part of a 3 and augments the outer (right) part

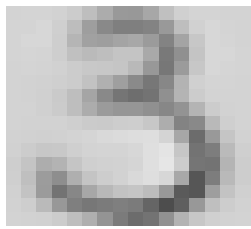




3rd PC: Moves a 3 down and tips it to the right

Digits example: Loadings and scores

2.51∗  +0.63∗  +2.02∗ 

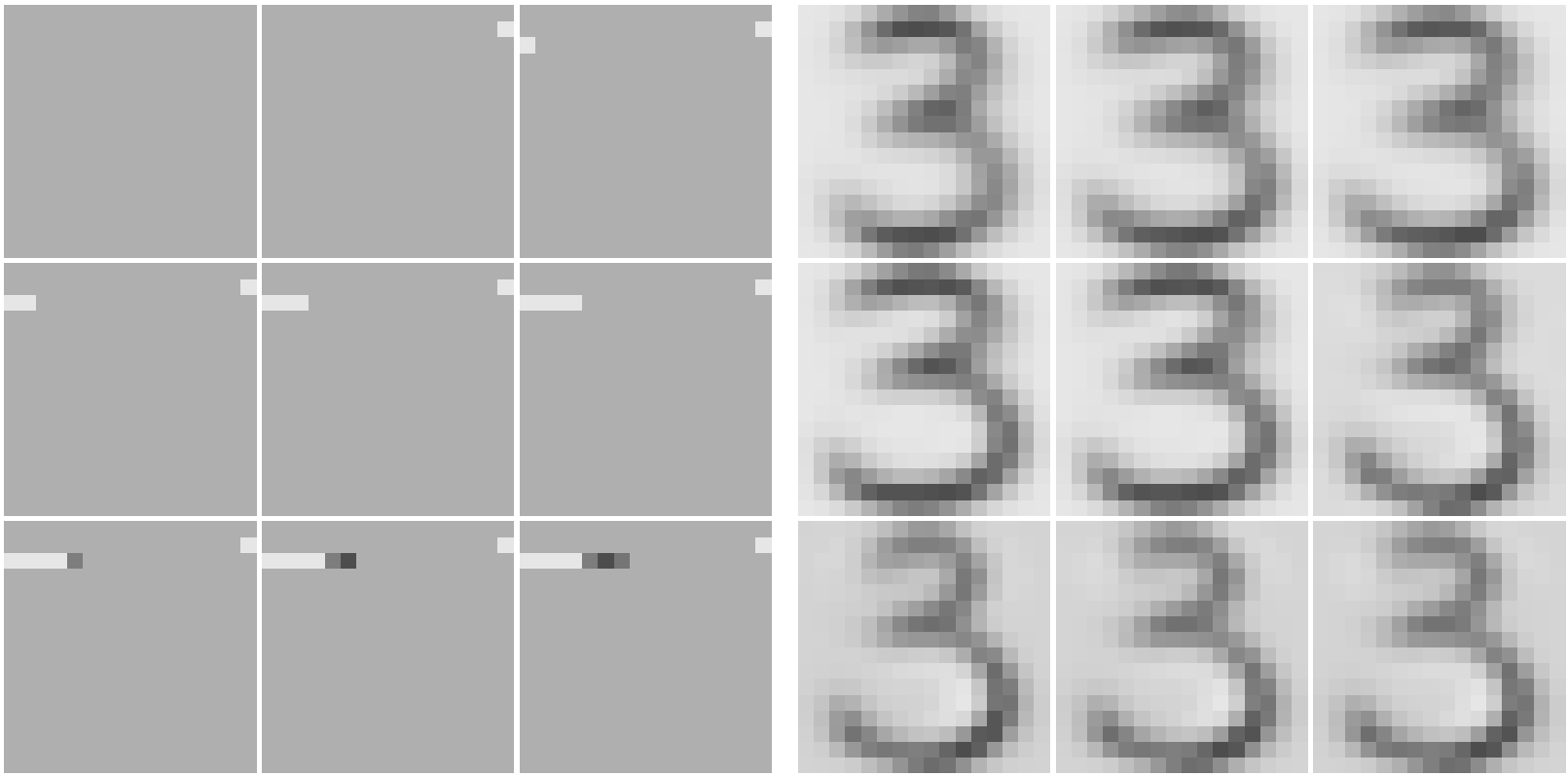
0.16∗  −4.55∗  +1.96∗  =



```
> round(scores[1,1:6],2)
  PC1   PC2   PC3   PC4   PC5   PC6
 2.52  0.64  2.02  0.17 -4.55  1.97
```

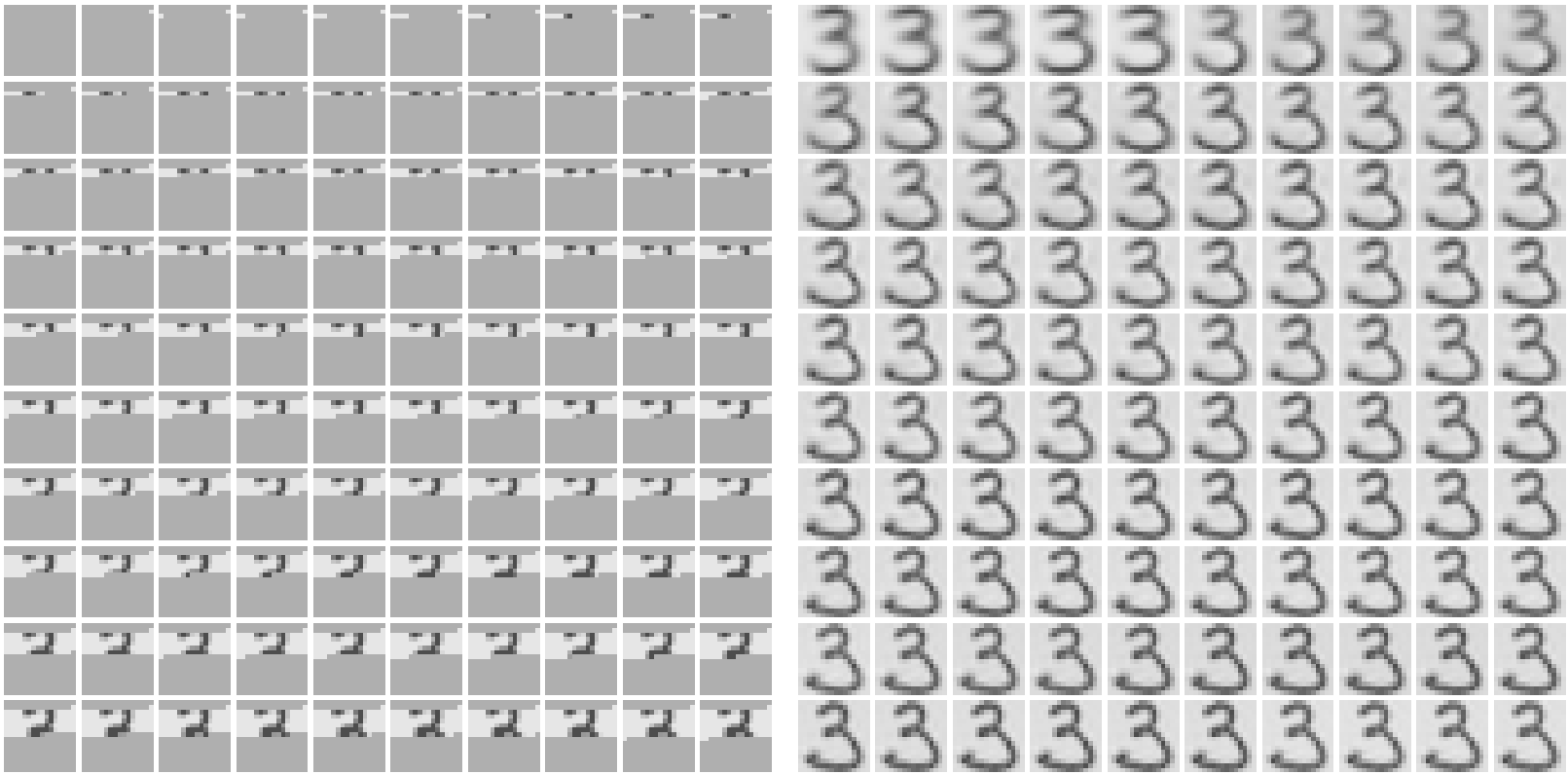
In pixel axis versus PCA axis

Using 9 axis dimensions



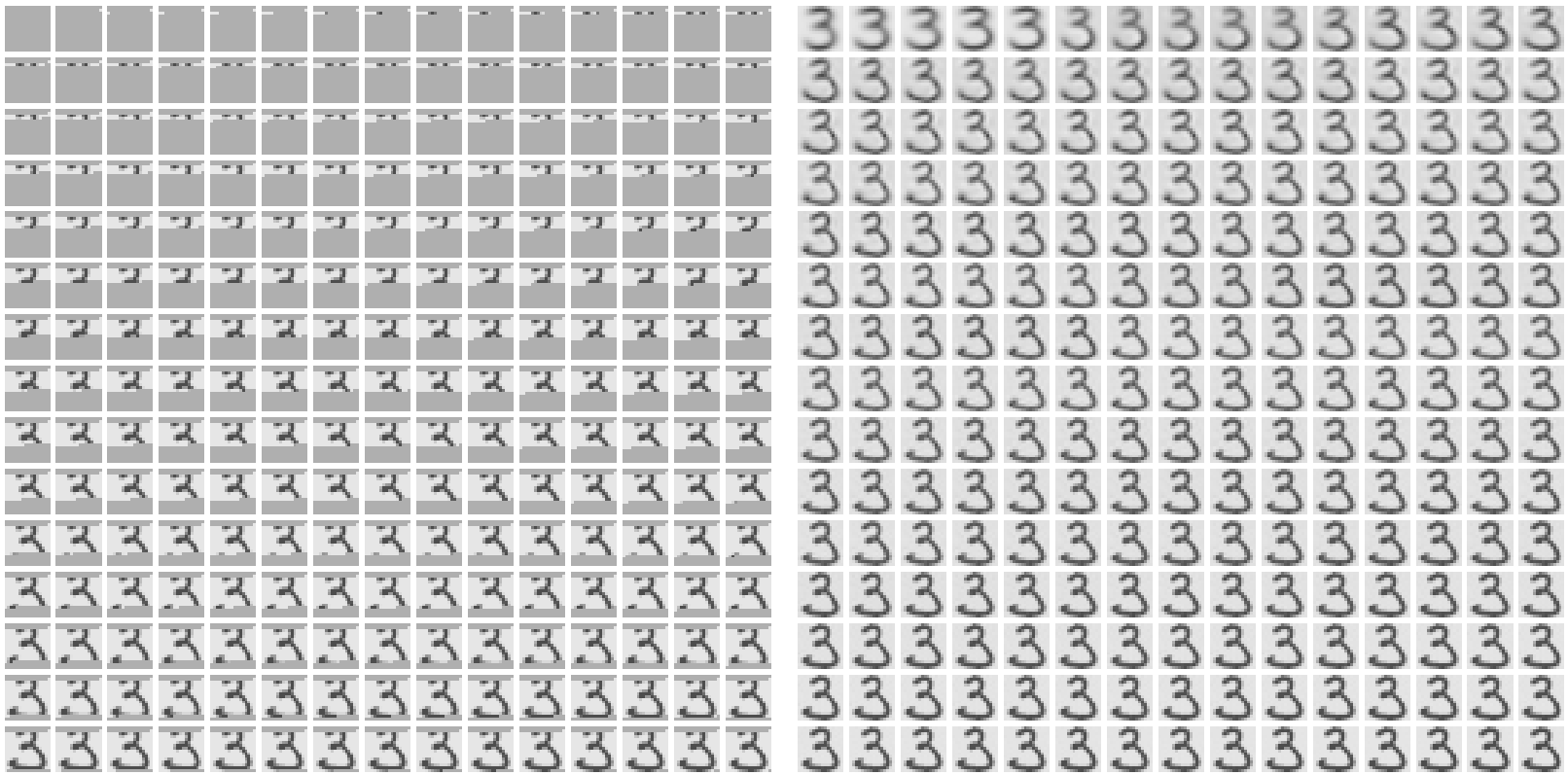
In pixel axis versus PCA axis

Using 100 axis dimensions

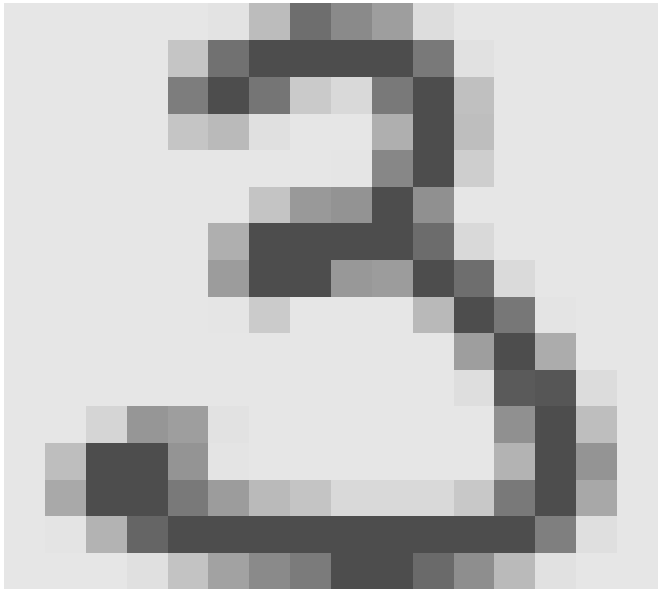


In pixel axis versus PCA axis

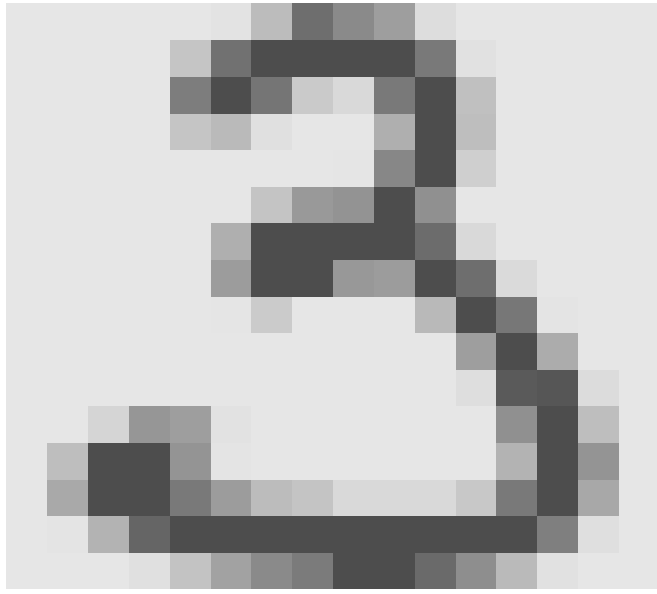
Using 225 axis dimensions



Decomposing in PCA axis only



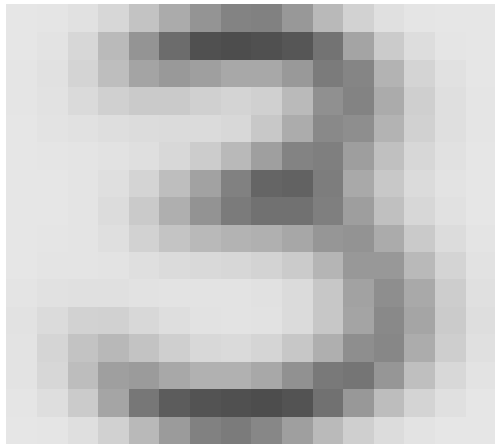
255 PCs



256 PCs

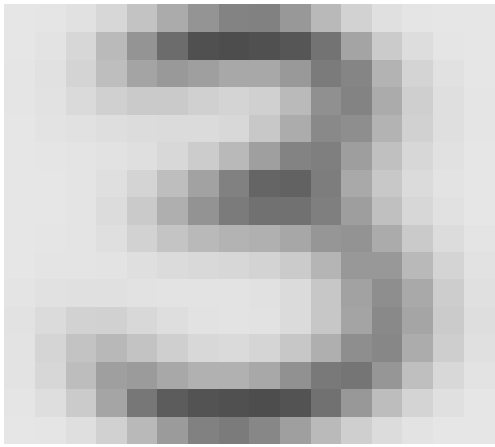
In pixel axis versus PCA axis

What is this mystery figure?



In pixel axis versus PCA axis

What is this mystery figure?



This is the mean (From centering $\mathbb{X}$: $(\mathbb{X} - \overline{\mathbb{X}}) = UDV^\top$)

(that is, the origin of the PCA axis, or $\overline{\mathbb{X}}$)[1]

```
plot.digit(attributes(digitsCenter)$'scaled:center')
```

Example: Facial recognition

Images

- There are 575 total images
- Each image is 92×112 pixels and grey scale
- These images come from the Sheffield face database

  (See `http://www.face-rec.org/databases/` for this and other databases)

Faces

Faces

The overall amount of dimension reduction we get is given by the choice of Q:

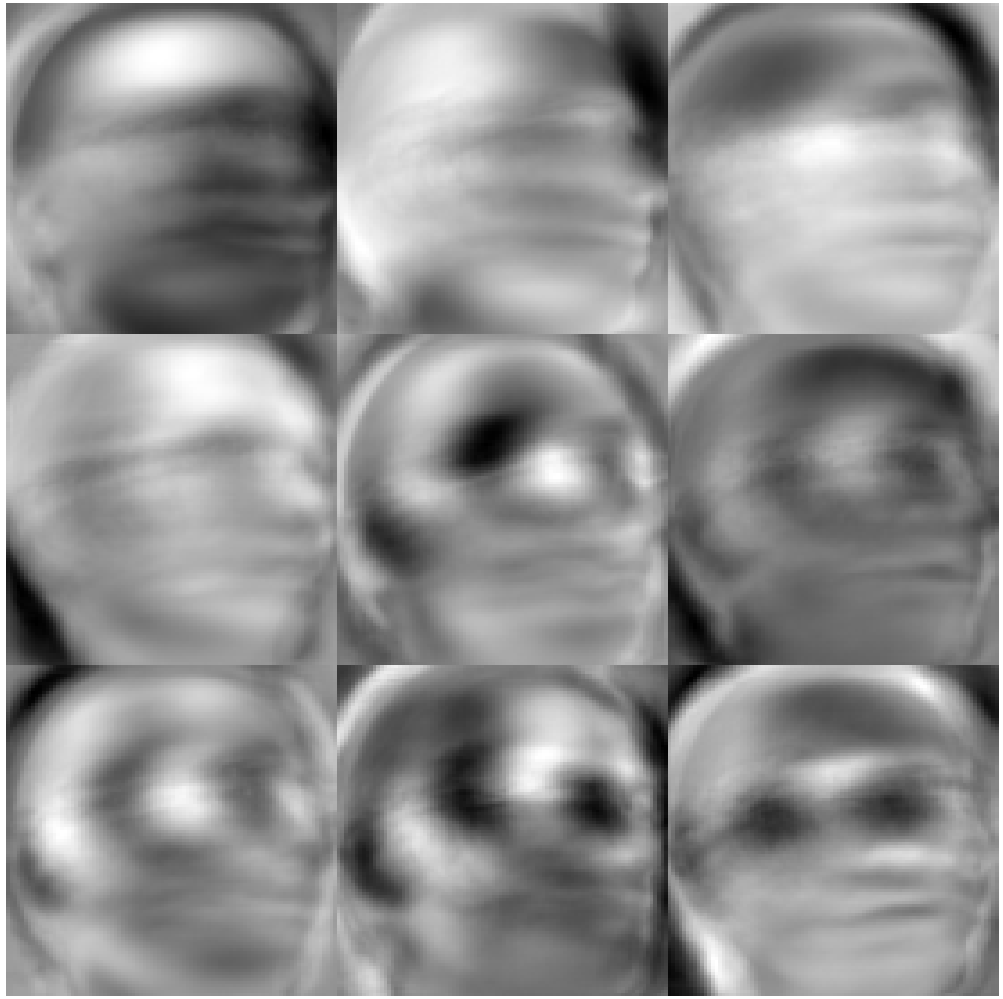
$$\text{Varying levels of } Q\text{:} \quad \tilde{\mathbb{X}} = \sum_{q=1}^{Q} d_q u_q v_q^\top + \overline{\mathbb{X}}$$

Reminder: We can compute PCA in R by

```
svdOut   = svd(scale(X,scale=F))
pcBasis  = svdOut$v
pcScores = X %*% pcBasis
```

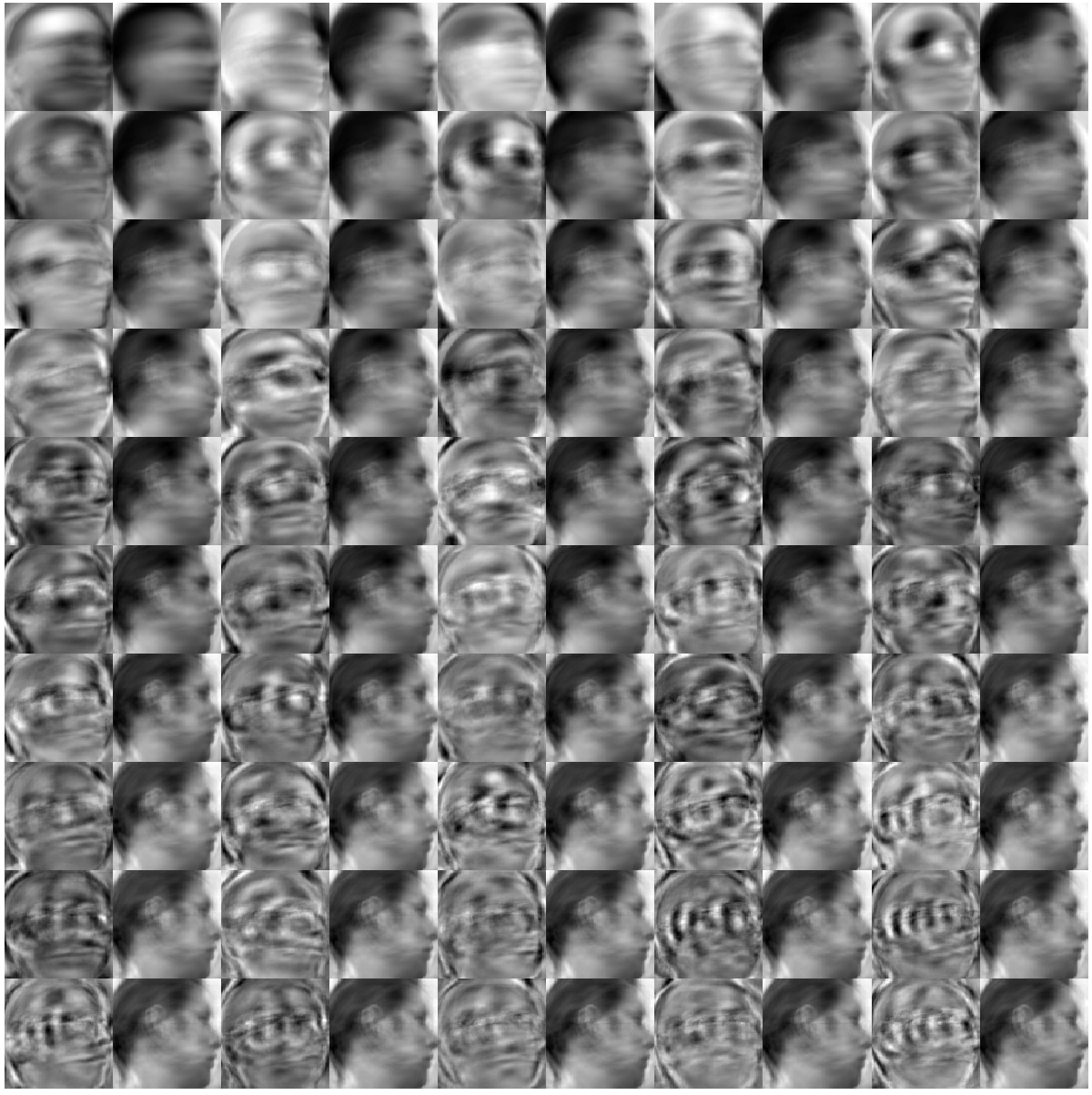
Let's apply this to the faces

Faces: v_q for $q = 1, \ldots, 9$



Faces: $\tilde{\mathbb{X}}$ for $Q = 1, \ldots, 9$

Faces: v_q and $\tilde{X}$



Postamble:

- Give some geometric intuition for PCA

  (PCA estimate the axes that best represents our data if it were approximately multivariate normal)
- Briefly overview how PCA is motivated/computed

  (PCA can be motivated as a minimum ℓ_2 distortion or maximum variance dimension reduction)
- Go through examples such as digits and faces