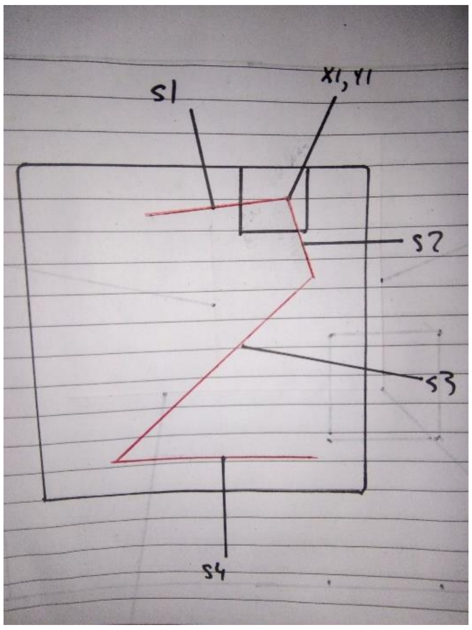
Mark Priestley

Student ID: 19375096

Question 1

I created my model 2s with 4 strokes. The code is on the next page.

This is a diagram showing the different strokes of my model.



The position of x1,y1 is randomly selected within the box.

S1 and s2 are supposed to resemble the curve of the 2.

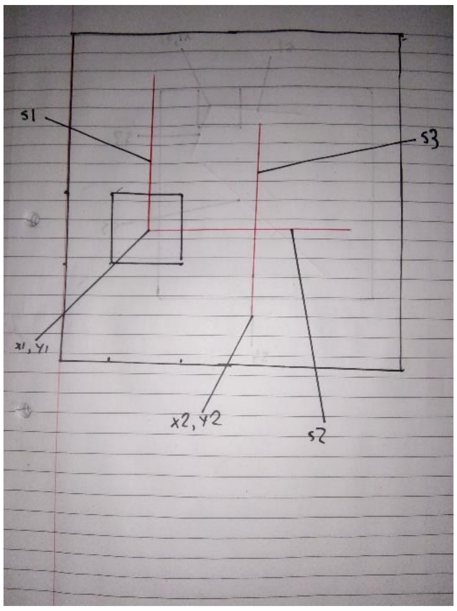
All the points are chosen in relation to x1,y1, ie x2,y2 is the endpoint of s2, and x3,y3 is the endpoint of s3.

One problem I had testing the model is that s3 would sometimes be too long, and go off the image. So after creating s3, I found the strokes endpoint. If that endpoint is off the screen, I tried to change the values of len3 and theta3.

```
theta3=runif(1,0.6*pi,0.85*pi)
len3=runif(1,8,12)
x3=x2 - len3*sin(theta3)
y3=y2 + len3*cos(theta3)
j = 0
while ((!(x3 >= 0 & x3 <= 7) | !(y3 >= 0 & y3 <= 7)) & j < 100 )
{
  theta3=runif(1,0.6*pi,0.85*pi)
  len3=runif(1,8,12)
  x3=x2 - len3*sin(theta3)
  y3=y2 + len3*cos(theta3)
  j = j + 1
}
```

```
stroke=readRDS("stroke.RDS")

# 4 strokes

for (i in 1:20)
{
# Creating s1
x1=runif(1,8,12)
y1=runif(1,13,16)
len1=runif(1,5,10)
theta1=runif(1,0.5*pi,0.7*pi)
width1=runif(1,0.5,3)

s1=stroke(x1,y1,theta1,len1,width1)
#image(c(1:16),c(1:16),s1,col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))

# Creating s2
# x1 y1 the same

# Need to make sure p3 is not off the screen, {x2, y2 >= 0}
theta2=runif(1,1*pi,1.4*pi)
len2=runif(1,4,8)
width2=runif(1,0.5,3)

s2=stroke(x1,y1,theta2,len2,width2)
#im=s1+s2
#im[im > 1]=1  #this thresholds the image so that the max brightness is 1
#image(c(1:16),c(1:16),im,col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))

# Creating s3
# x2,y2 endpoints of s2
x2=x1 - len2*sin(theta2)
y2=y1 + len2*cos(theta2)
theta3=runif(1,0.6*pi,0.85*pi)
len3=runif(1,8,12)
x3=x2 - len3*sin(theta3)
y3=y2 + len3*cos(theta3)
j = 0
while ((!(x3 >= 0 & x3 <= 7) | !(y3 >= 0 & y3 <= 7)) & j < 100 )
{
  theta3=runif(1,0.6*pi,0.85*pi)
  len3=runif(1,8,12)
  x3=x2 - len3*sin(theta3)
  y3=y2 + len3*cos(theta3)
  j = j + 1
}
width3=runif(1,0.5,3)

s3=stroke(x2,y2,theta3,len3,width3)


# Creating s4
# x2,y2 endpoints of s3
x3=x2 - len3*sin(theta3)
y3=y2 + len3*cos(theta3)

theta4=runif(1,-0.53*pi,-0.47*pi)
len4=runif(1,7,12)
width4=runif(1,0.5,3)

s4=stroke(x3,y3,theta4,len4,width4)

im=s1+s2+s3+s4
im[im > 1]=1  #this thresholds the image so that the max brightness is 1
image(c(1:16),c(1:16),im,col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
readline()
}
```

I created my 4s with 3 strokes.



X1 and y1 are selected within a box.

Y2 needed to be less than y1, and the endpoint of s3 should be about halfway on the line s1.

```
for (i in 1:20)
{
# Creating s1
# x1 boundary, (3 -> 5)
x1=runif(1,2,5)
# y1 boundary (4 -> 7)
y1=runif(1,4,7)
# Length (5,13)
len1=runif(1,6,12)
# Angle (-0.1*pi -> 0*pi)
theta1=runif(1,-0.1*pi,0*pi)
# Width (0.5 -> 3)
width1=runif(1,0.5,3)

s1=stroke(x1,y1,theta1,len1,width1)


# Creating s2
# x and y points same as s1
# Length (7 -> 13)
len2=runif(1,7,16)
# Angle (0.43*pi -> 0.57*pi)
theta2=runif(1,-0.57*pi,-0.43*pi)
# Width (0.5 -> 3)
width2=runif(1,0.5,3)

s2=stroke(x1,y1,theta2,len2,width2)

# Creating s3
# Needs to cross s2 at about half it's length
# x2 needs to find half the distance of the length of s2, and create a point -2,+5 around that
x1e = x1 - len2*sin(theta2) # endpoint of x1
half_s2 = (x1 + x1e) %/% 2
x2=runif(1,-1+half_s2,3+half_s2)

# y2 needs to be lower than y1
y2=runif(1,y1-13,y1-2)

# Length (9 -> 13)
y1e = y1 + len1*cos(theta1) # endpoint of y1

len3=runif(1,y1-y2+((y1e+y1) %/% 2)-8,y1-y2+((y1e+y1) %/% 2))
#len3=runif(1,3,9)

# Angle
theta3=runif(1,-0.05*pi,0.05*pi)

# Width (0.5 -> 3)
width3=runif(1,0.5,3)

s3=stroke(x2,y2,theta3,len3,width3)

im=s1+s2+s3
im[im > 1]=1  #this thresholds the image so that the max brightness is 1
image(c(1:16),c(1:16),im,col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))

readline()
}
```

Question 2

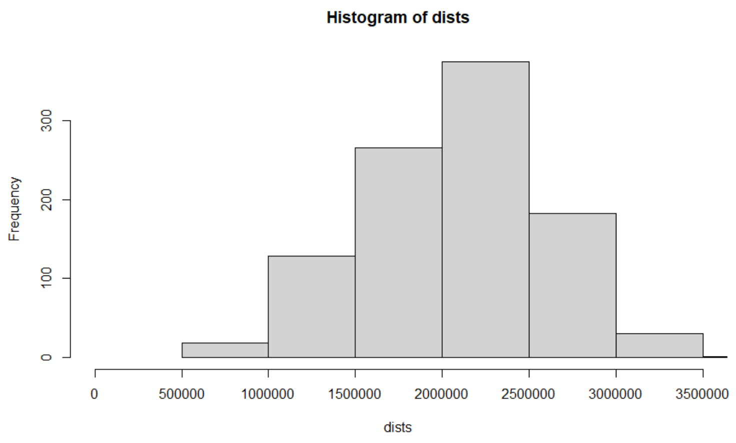
Fitting random 2 to real 2s

I firstly ran the code from Q1 in a for loop 10000 times, and added it to a matrix b2. I also took note of the parameters I used, and stored those in the matrix params.

```
stroke=readRDS("final_assessment/stroke.RDS")
findnns=readRDS("final_assessment/findnns.rds")

b2=matrix(0,256,10000)

# Parameters to take note of
params=matrix(0,14,10000)
for (i in 1:10000)
{
 # Creating s1
 x1=runif(1,8,12)
 y1=runif(1,13,16)
 len1=runif(1,5,10)
 theta1=runif(1,0.5*pi,0.7*pi)
 width1=runif(1,0.5,3)

 s1=stroke(x1,y1,theta1,len1,width1)

 # Creating s2
 # x1 y1 the same

 # Need to make sure p3 is not off the screen, {x2, y2 >= 0}
 theta2=runif(1,1*pi,1.4*pi)
 len2=runif(1,4,8)
 width2=runif(1,0.5,3)

 s2=stroke(x1,y1,theta2,len2,width2)

 # Creating s3
 # x2,y2 endpoints of s2
 x2=x1 - len2*sin(theta2)
 y2=y1 + len2*cos(theta2)
 theta3=runif(1,0.6*pi,0.85*pi)
 len3=runif(1,8,12)
 x3=x2 - len3*sin(theta3)
 y3=y2 + len3*cos(theta3)
 j = 0
 while ((!(x3 >= 0 & x3 <= 7) | !(y3 >= 0 & y3 <= 7)) & j < 100 )
 {
  theta3=runif(1,0.6*pi,0.85*pi)
  len3=runif(1,8,12)
  x3=x2 - len3*sin(theta3)
  y3=y2 + len3*cos(theta3)
  j = j + 1
 }
 width3=runif(1,0.5,3)

 s3=stroke(x2,y2,theta3,len3,width3)


 # Creating s4
 # x2,y2 endpoints of s3
 x3=x2 - len3*sin(theta3)
 y3=y2 + len3*cos(theta3)

 theta4=runif(1,-0.53*pi,-0.47*pi)
 len4=runif(1,7,12)
 width4=runif(1,0.5,3)

 s4=stroke(x3,y3,theta4,len4,width4)

 im=s1+s2+s3+s4
 im[im > 1]=1  #this thresholds the image so that the max brightness is 1

 im = im*256
 im = im[,16:1] #we have to turn the image upside-down so that it matches the real digits

 b2[,i]=im
 # Take note of parameters
 p=c(x1,x2,x3,y1,y2,y3,theta1,theta2,theta3,theta4,len1,len2,len3,len4)
 params[,i]=p
}
```

I then loaded the real dataset, and fit the random 2s to the real 2s.

```
d2=scan("digits/Zl2d.dat",nlines=1000,n=256000)
d2=matrix(d2,256,1000)
proc.time()
nbrs2 = matrix(0,40,1000)
for(i in c(1:1000))nbrs2[,i]=findnns(b2,d2[,i])
proc.time()

nearest = b2[,nbrs2[1,]]
dists = nearest - d2
dists = apply(dists^2,2,sum)

hist(dists,xlim=c(0,3500000))
```



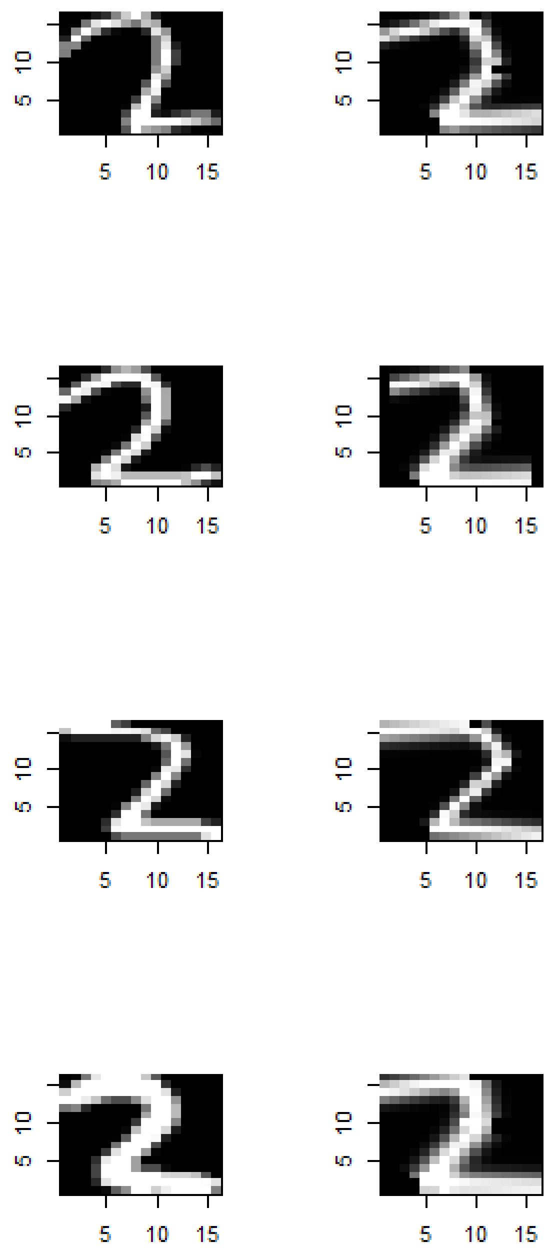
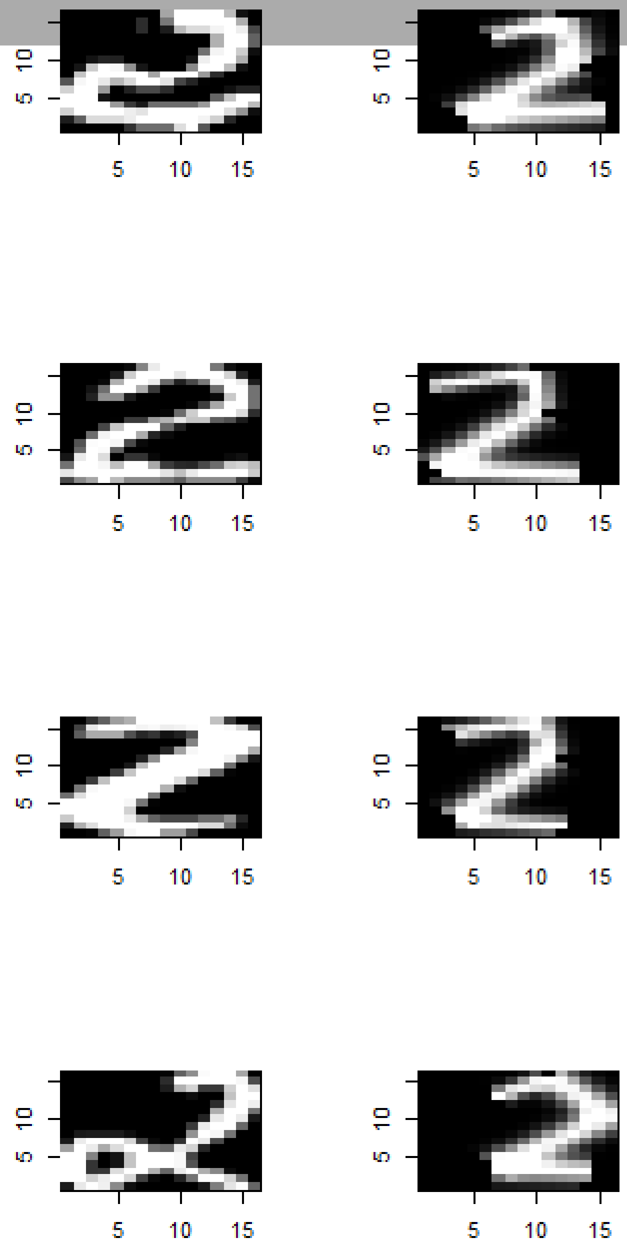
Looking at the best matches (< 800,000)

```
best_fits=dists<800000
best_matches=nearest[,best_fits]
best_matches_real=d2[,best_fits]
par(mfrow=c(4,2))

for (i in 1:4)
{
 z = matrix(best_matches_real[,i],16,16) # the real digit
 image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
 z = matrix(best_matches[,i],16,16)
 image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
}
```

Looking at the worst matches(> 3,200,000)

```
worst_fits=dists>3200000
worst_matches=nearest[,worst_fits]
worst_matches_real=d2[,worst_fits]
dim(worst_matches)
# 9 worst matches

par(mfrow=c(4,2))

for (i in 4:7)
{
  z = matrix(worst_matches_real[,i],16,16) # the real digit
  image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
  z = matrix(worst_matches[,i],16,16)
  image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
}
```

Best Matches **Worst Matches**

The best matches width of strokes are accurate, and the angle and length of the long diagonal stroke (s3 in my model) are also accurate.

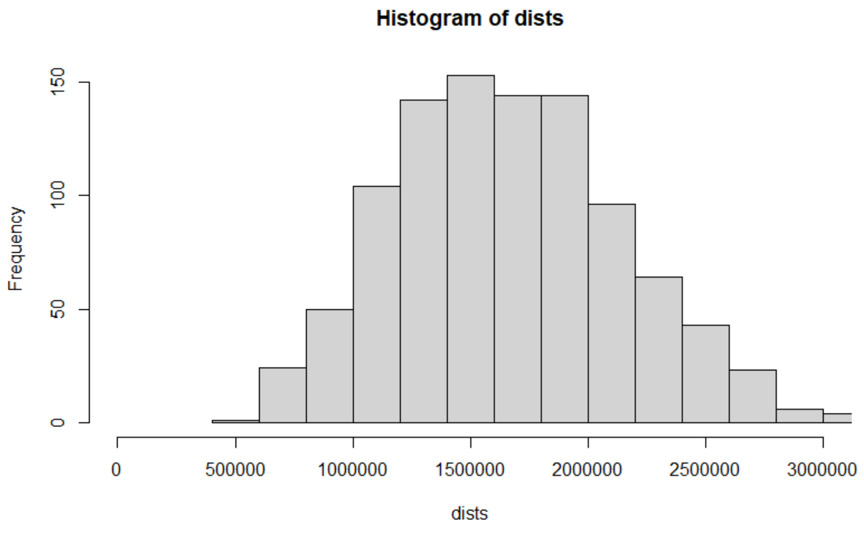
The worst matches have a combination of different problems, the position of x2 and y2 (the top of the diagonal stroke s3) is a problem in some. I believe the most common problem is some of the digits have too many “curves”, for example the real digit from the digits above has a curve transitioning from the diagonal stroke to the bottom stroke. This is quite difficult to adapt my model to, I could add in a few more strokes but I feel it is unique enough the I shouldn’t change my model for it.

Fitting random 4 to real 4s

```
b4=matrix(0,256,10000)
params=matrix(0,10,10000)

for (i in 1:10000)
{
# Creating s1
# x1 boundary, (3 -> 5)
x1=runif(1,2,5)
# y1 boundary (4 -> 7)
y1=runif(1,4,7)
# Length (5,13)
len1=runif(1,6,12)
# Angle (-0.1*pi -> 0*pi)
theta1=runif(1,-0.1*pi,0*pi)
# Width (0.5 -> 3)
width1=runif(1,0.5,3)

s1=stroke(x1,y1,theta1,len1,width1)

# Creating s2
# x and y points same as s1
# Length (7 -> 13)
len2=runif(1,7,16)
# Angle (0.43*pi -> 0.57*pi)
theta2=runif(1,-0.57*pi,-0.43*pi)
# Width (0.5 -> 3)
width2=runif(1,0.5,3)

s2=stroke(x1,y1,theta2,len2,width2)

# Creating s3
# Needs to cross s2 at about half it's length
# x2 needs to find half the distance of the length of s2, and create a point -2,+5 around that
x1e = x1 - len2*sin(theta2) # endpoint of x1
half_s2 = (x1 + x1e) %/% 2
x2=runif(1,-1+half_s2,3+half_s2)

# y2 needs to be lower than y1
y2=runif(1,y1-13,y1-2)

# Length (9 -> 13)
y1e = y1 + len1*cos(theta1) # endpoint of y1

len3=runif(1,y1-y2+((y1e+y1) %/% 2)-8,y1-y2+((y1e+y1) %/% 2))
#len3=runif(1,3,9)

# Angle
theta3=runif(1,-0.05*pi,0.05*pi)

# Width (0.5 -> 3)
width3=runif(1,0.5,3)

s3=stroke(x2,y2,theta3,len3,width3)

im=s1+s2+s3
im[im > 1]=1  #this thresholds the image so that the max brightness is 1
im = im*256
im = im[,16:1] #we have to turn the image upside-down so that it matches the real digits

b4[,i]=im

p=c(x1,x2,y1,y2,theta1,theta2,theta3,len1,len2,len3)
params[,i]=p
}
```

Checking the fits

```
d4=scan("digits/Zl4d.dat",nlines=1000,n=256000)
d4=matrix(d4,256,1000)

nbrs4 = matrix(0,40,1000)
proc.time()
for(i in c(1:1000))nbrs4[,i]=findnns(b4,d4[,i])
proc.time()

nearest = b4[,nbrs4[1,]]
dim(nearest)

dists = nearest - d4
dists = apply(dists^2,2,sum)

hist(dists,xlim=c(0,3000000))
```

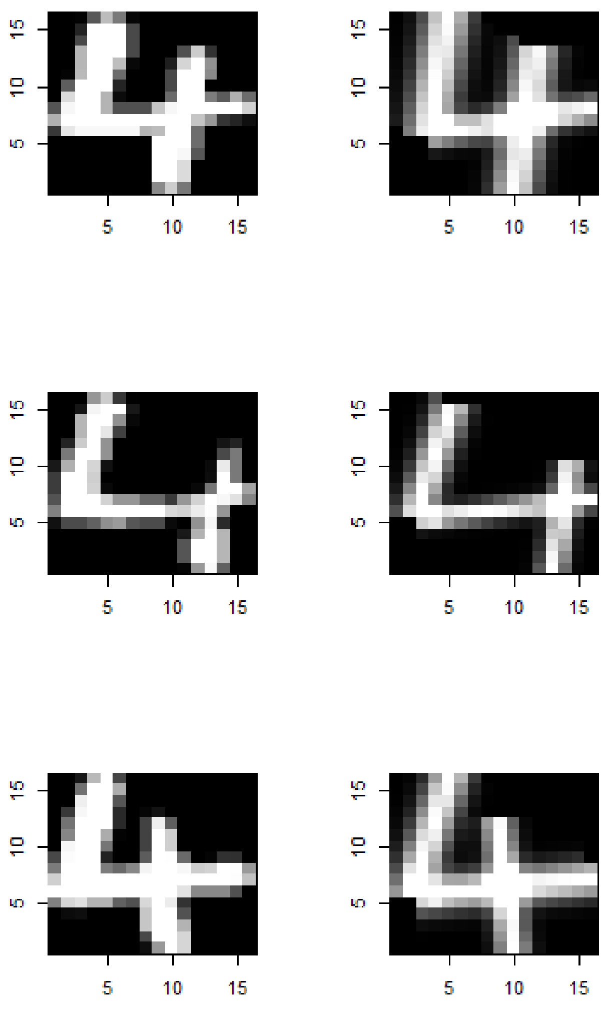


On average, the 4s are more accurate than the 2s, probably because it is mainly straight lines. However, it is still not great.

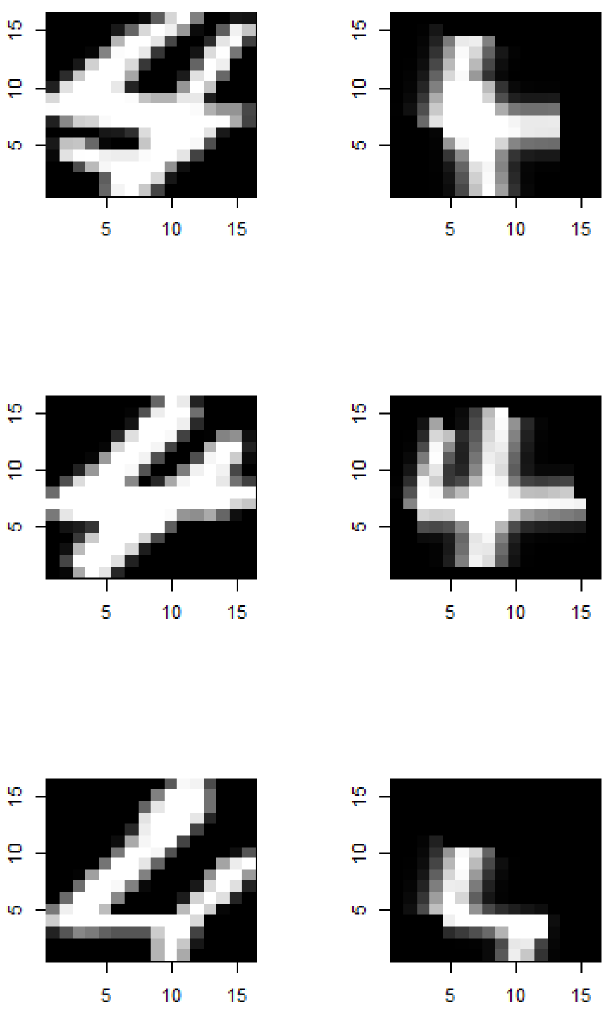
Looking at best and worst matches

```
best_fits=dists<650000
best_matches=nearest[,best_fits]
best_matches_real=d4[,best_fits]

dim(best_matches)

par(mfrow=c(3,2))
for (i in 1:3)
{
  z = matrix(best_matches_real[,i],16,16) # the real digit
  image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
  z = matrix(best_matches[,i],16,16)
  image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
  readline()
}
worst_fits=dists>3000000
worst_matches=nearest[,worst_fits]
worst_matches_real=d4[,worst_fits]

dim(worst_matches)

par(mfrow=c(3,2))
for (i in 1:3)
{
  z = matrix(worst_matches_real[,i],16,16) # the real digit
  image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
  z = matrix(worst_matches[,i],16,16)
  image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))
  readline()
}
```

Best Matches

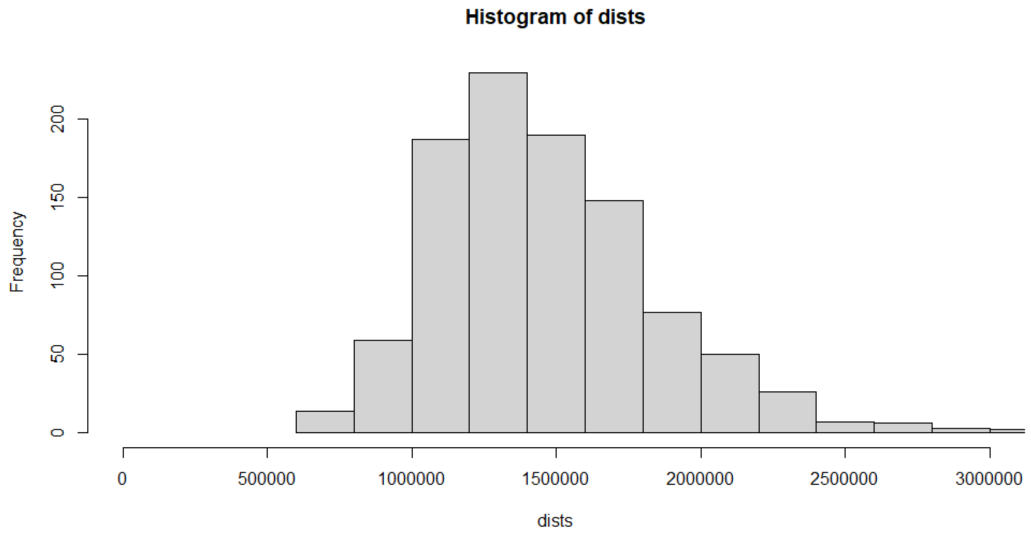


Worst Matches

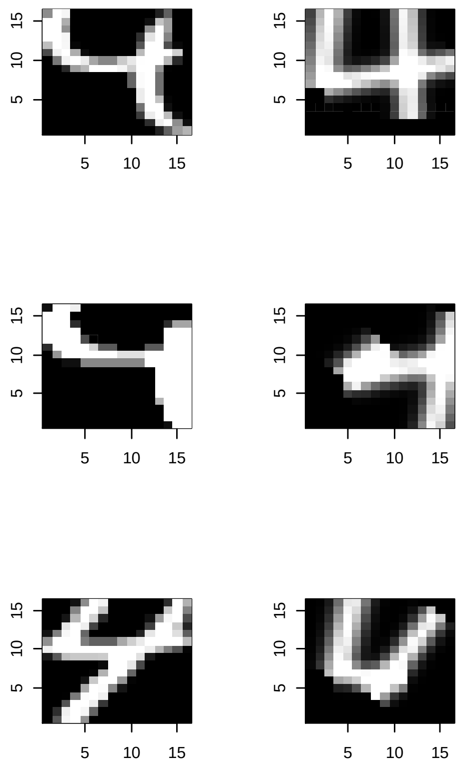


The best matches are quite accurate, the angle and length of all the strokes are good.

The problem with the worst digits seem to be the angle of the strokes, specifically s1 and s3. I decided to expand the range of angles available on the s1 and s2, to see if that improves the accuracy. (set the lower limit of theta 1 and theta2 to -0.3*pi). Repeating all the steps above, I got a new histogram.



This has slightly increased the accuracy, the largest bin is around 1,300,000, instead of 1,500,000 as before. Looking again at the worst matches



The problem with these matches might be the unusually high value for y1. I don't think these problems could be fixed with an extra stroke or a change in parameters, as I believe the issue is the unlikely combination of stroke angle and y position. Changing these parameters may bring the overall accuracy down, even if it improves those outliers accuracy.

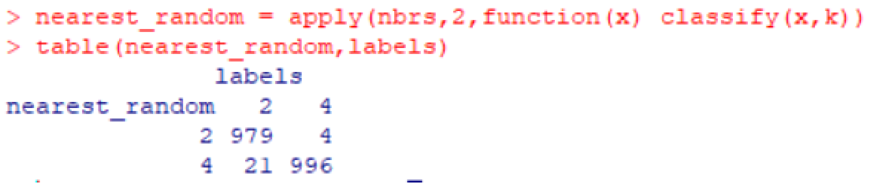
I saved the parameter matrices from the random 2s and new random 4s, and the b2 and b4 matrices, for the next few questions.

Question 3

I loaded in the b2 and b4 matrices, and column banded them to make the matrix b. I did the same for the real digits. I also created the classify function.

```
findnns=readRDS("final_assessment/findnns.rds")
classify = function(nns,k)
{
 labels = c(rep("2",10000),rep("4",10000))
 digits= labels[nns[1:k]] #we include the first nearest neighbour
 t=table(digits)
 m=which.max(t)
 names(m)
}
b2=readRDS("final_assessment/b2.rds")
b4=readRDS("final_assessment/b4.rds")

d2=scan("digits/Zl2d.dat",nlines=1000,n=256000)
d2=matrix(d2,256,1000)
d4=scan("digits/Zl4d.dat",nlines=1000,n=256000)
d4=matrix(d4,256,1000)

d=cbind(d2,d4)
b=cbind(b2,b4)

nbrs = matrix(0,40,2000)
for(i in c(1:2000))nbrs[,i]=findnns(b,d[,i])
```

Using this, I found the confusion matrix

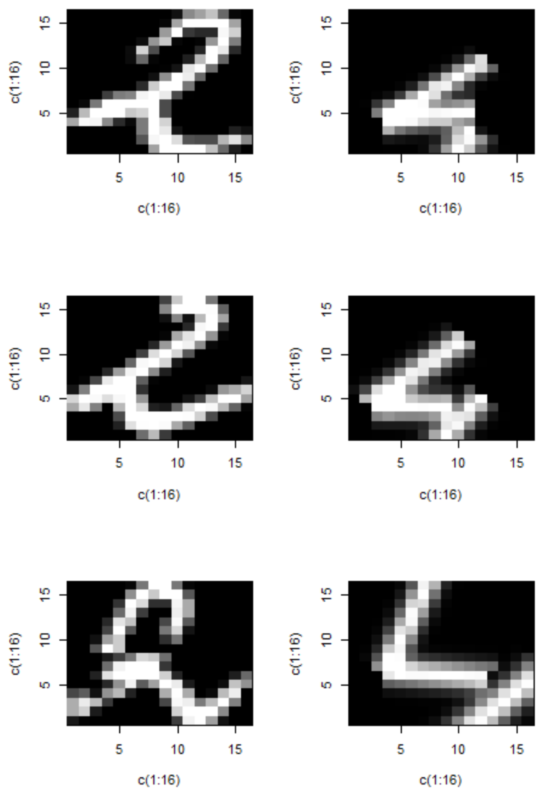


```
> nearest_random = apply(nbrs,2,function(x) classify(x,k))
> table(nearest_random,labels)
              labels
nearest_random   2   4
             2 979   4
             4  21 996
```

The real 4s are quite accurately matched, but the real 2s are matched to random 4s 21 times.

Looking at wrongly matched real 2s.

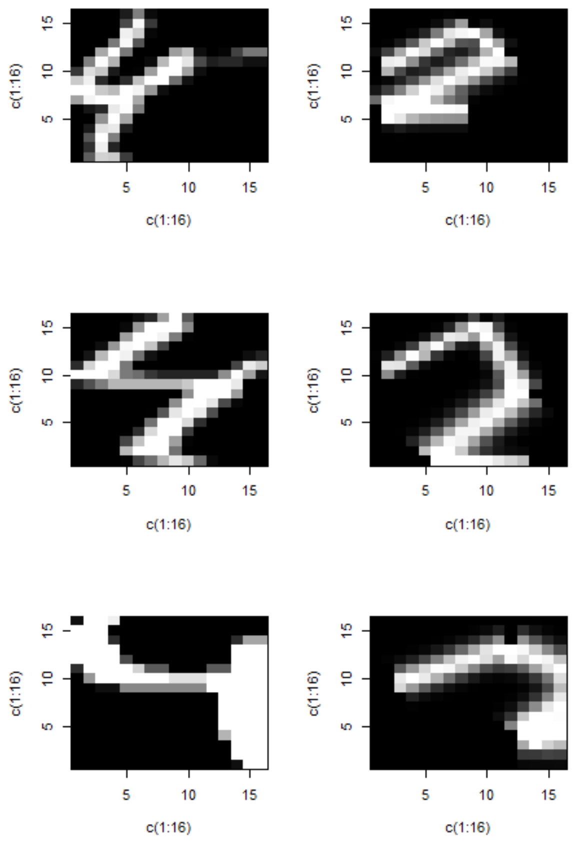
```
misclassified = which(labels == "2" & nearest_random == "4")

par(mfrow=c(3,2))
for(i in misclassified)
{
 z=matrix(d[,i],16,16)
 image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
 z=matrix(b[,nbrs[1,i]],16,16)
 image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
}
```



The upper stroke of the 4 matches the long diagonal stroke of the 2, the upper cross stroke of the 4 (s3 in my model) is poorly positioned in the examples, which may also be causing problems.

There is a lot of “curves” on the real 2s, which may explain why my random 2s did not match

Doing the same above but with real 4s and random 2s.



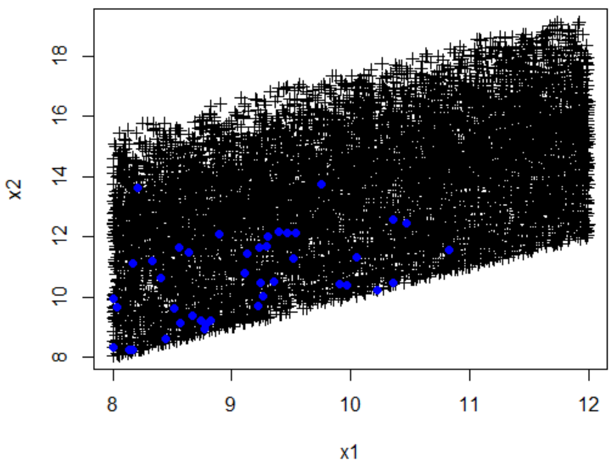
The 2 upper strokes of the real 4s are similar to s1 and s3 of the random 2s.

Question 5

I reran the code from q2 for the 2s. I set any best fit value to anything less than 1,200,000. This gave about 60 points. The params matrix for 2s is a 14 x 10000 matrix, with the 14 parameter values being x1, x2, x3, y1, y2, y3, theta1, theta2, theta3, theta4, len1, len2, len3 and len4. The line 'nbrs[1,best_fits]' will select the 60 or so best matches to the real digits. 'params[1,nbrs4[1,best_fits]' will then select the first parameter (x1) for the best matches.

There are a lot of different combinations of variables, so I selected some which look interesting.

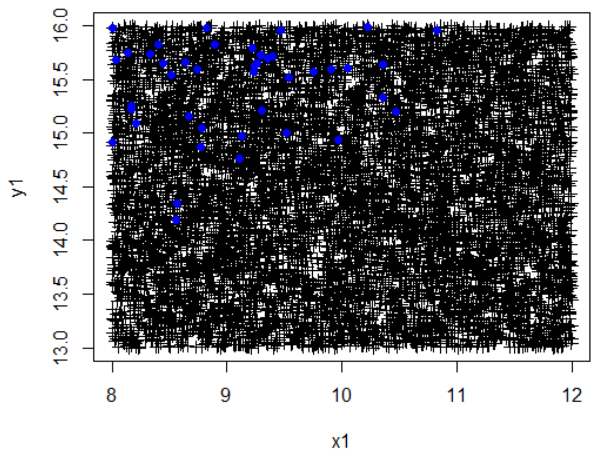
```
plot(params[1,],params[2,],pch=3,xlab="x1",ylab="x2")
points(params[1,nbrs2[1,best_fits]],params[2,nbrs2[1,best_fits]],col="blue",pch=19)
```



The black plus signs are the entire 10,000 dataset for x1 and x2, while the blue points are the best fits parameters for x1 and x2.

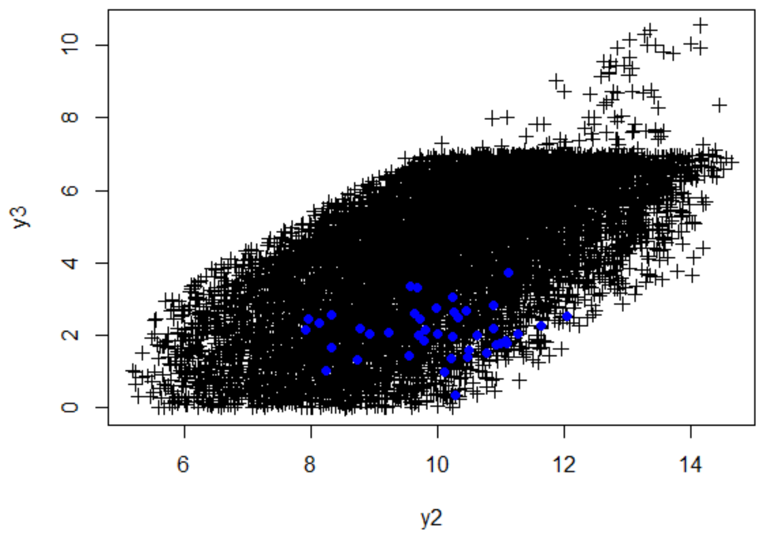
They seem to cluster in the bottom left of the plot, so maybe lower the limit of the lower parameter might give better fits.

```
plot(params[1,],params[4,],pch=3,xlab="x1",ylab="y1")
points(params[1,nbrs2[1,best_fits]],params[4,nbrs2[1,best_fits]],col="blue",pch=19)
```



The best fits build up in the top left of the image. This suggest a high y value generally gives better points (16 is the highest it goes).

```
plot(params[5,],params[6,],pch=3,xlab="y2",ylab="y3")
points(params[5,nbrs2[1,best_fits]],params[6,nbrs2[1,best_fits]],col="blue",pch=19)
```

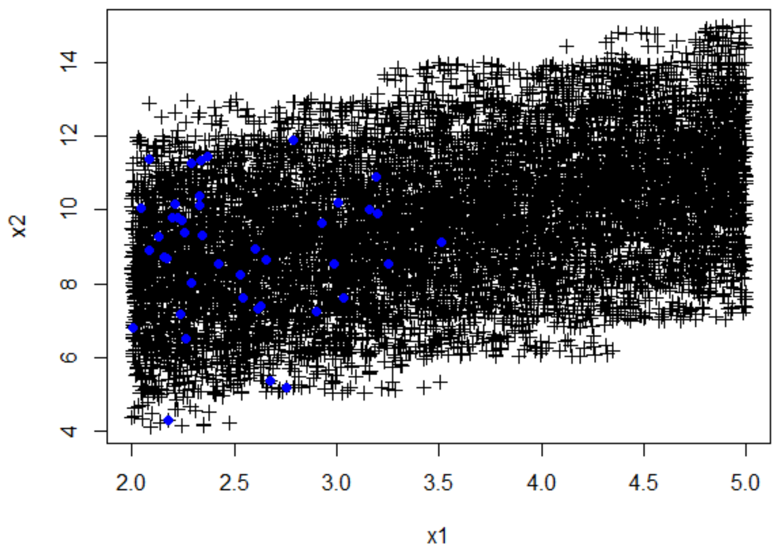


There seems to be a build up around a y2 value of 10 and a y3 value of 2.

The few examples above show there is a definite correlation between the different values, they are not independent.

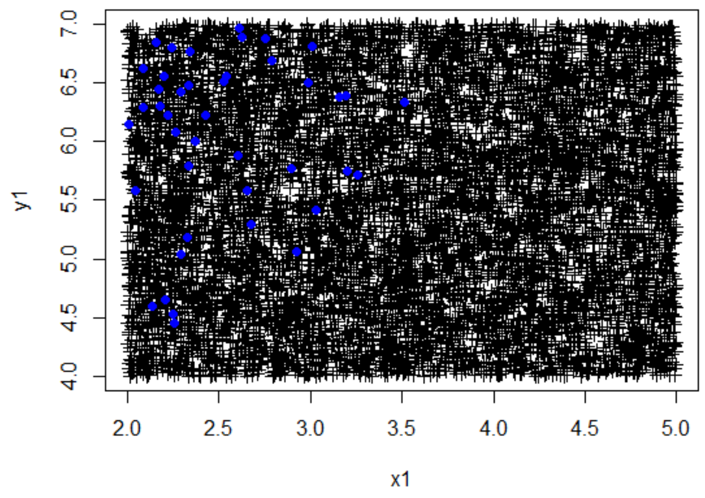
I did the same for the 4s. The parameters for the 4 are x1, x2, y1, y2, theta1, theta2, theta3, len1, len2 and len3. I set the best fit limit as anything less than 960,000 (about 60 values).

```
plot(params[1,],params[2,],pch=3,xlab="x1",ylab="x2")
points(params[1,nbrs4[1,best_fits]],params[2,nbrs4[1,best_fits]],col="blue",pch=19)
```



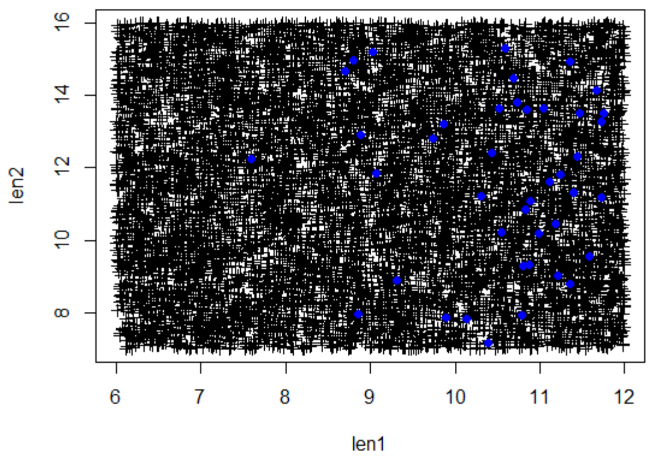
There does seem to be a build up on the left side of the plot. The x2 value is not that significant, however, the best fit parameter for x1 does build up on the lower values.

```
plot(params[1,],params[3,],pch=3,xlab="x1",ylab="y1")
points(params[1,nbrs4[1,best_fits]],params[3,nbrs4[1,best_fits]],col="blue",pch=19)
```



There is a build up of best points on the top left of the plot.

```
plot(params[8,],params[9,],pch=3,xlab="len1",ylab="len2")
points(params[8,nbrs4[1,best_fits]],params[9,nbrs4[1,best_fits]],col="blue",pch=19)
```



There is a build up of points on the right side of the plot.

There does seem to be a correlation between the different parameters of the random 4s.

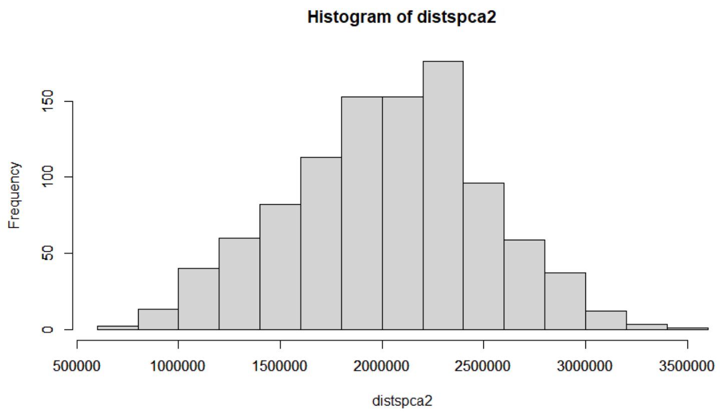
Question 6

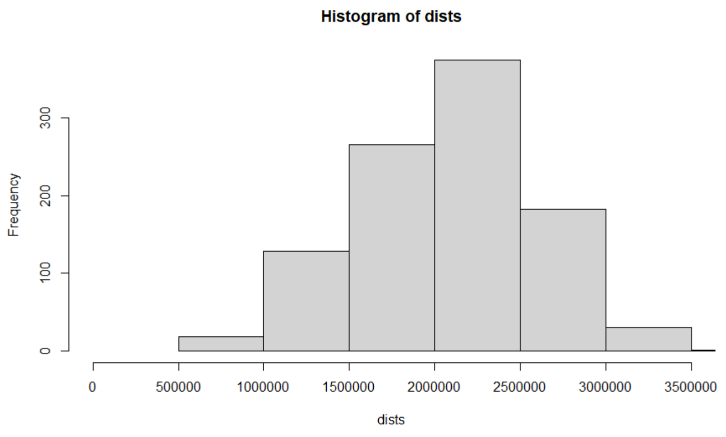
Using PCA to create random 2s.

```
params2=readRDS("params2.rds")
best_fits2=readRDS("best_fits2.rds")
stroke=readRDS("final_assessment/stroke.RDS")
findnns=readRDS("final_assessment/findnns.rds")

gp=params2[,best_fits2]


gp=t(gp)
c=var(gp)
e=eigen(c)
p=gp %*% e$vectors
plot(e$values)
mp=apply(p,2,mean) #computes the mean for each eigenvectors

#this code generated 10000 random values for each eigenvector
#the random values come from a Normal distribtuion
ranv = matrix(rnorm(140000),14,10000)
ranv = ranv * sqrt(e$values) #sets the standard deviation
ranv = ranv+mp # sets the mean for each eigenvectros

#this line reconstructs the parameter vectors from the #eignvectors
#ranv3 contains the 14 parameters for the model
ranv3=t(ranv)%*% t(e$vectors)

b2=matrix(0,256,10000)
# x1, x2, x3, y1, y2, y3, theta1, theta2, theta3, theta4, len1, len2, len3, len4
# 1  2  3  4  5  6    7     8     9    10   11   12   13   14
for (i in 1:10000)
{
x1=ranv3[i,1]
x2=ranv3[i,2]
x3=ranv3[i,3]
y1=ranv3[i,4]
y2=ranv3[i,5]
y3=ranv3[i,6]
theta1=ranv3[i,7]
theta2=ranv3[i,8]
theta3=ranv3[i,9]
theta4=ranv3[i,10]
len1=ranv3[i,11]
len2=ranv3[i,12]
len3=ranv3[i,13]
len4=ranv3[i,14]
width1=runif(1,0.5,3)
width2=runif(1,0.5,3)
width3=runif(1,0.5,3)
width4=runif(1,0.5,3)

s1=stroke(x1,y1,theta1,len1,width1)
s2=stroke(x1,y1,theta2,len2,width2)
s3=stroke(x2,y2,theta3,len3,width3)
s4=stroke(x3,y3,theta4,len4,width4)

im=s1+s2+s3+s4
im[im > 1]=1  #this thresholds the image so that the max brightness is 1

im = im*256
im = im[,16:1] #we have to turn the image upside-down so that it matches the real digits

b2[,i]=im
}
```

I then compared the real 4s to the nearest random 4, to get a histogram of the error.

```
d2=scan("digits/Zl2d.dat",nlines=1000,n=256000)
d2=matrix(d2,256,1000)

proc.time()
nbrs2pca = matrix(0,40,1000)
for(i in c(1:1000))nbrs2pca[,i]=findnns(b2,d2[,i])
proc.time()

nearestpca2 = b2[,nbrs2pca[1,]]
dim(nearestpca2)

#this code calculates the distance between each real one
#and its nearest randomly generated one
distspca2 = nearestpca2 - d2
distspca2 = apply(distspca2^2,2,sum)

hist(distspca2)
```

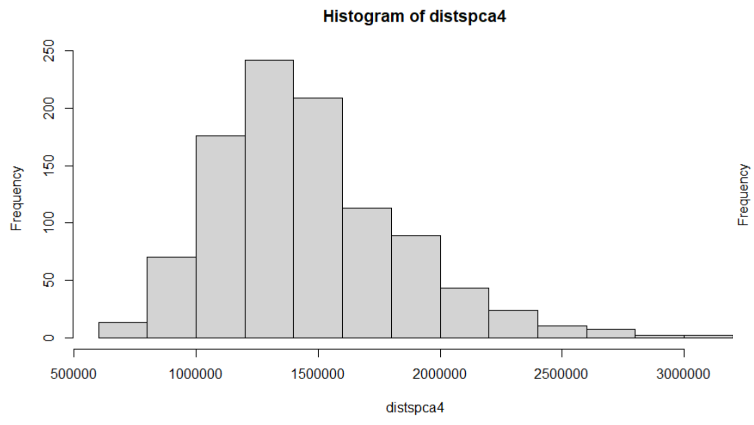


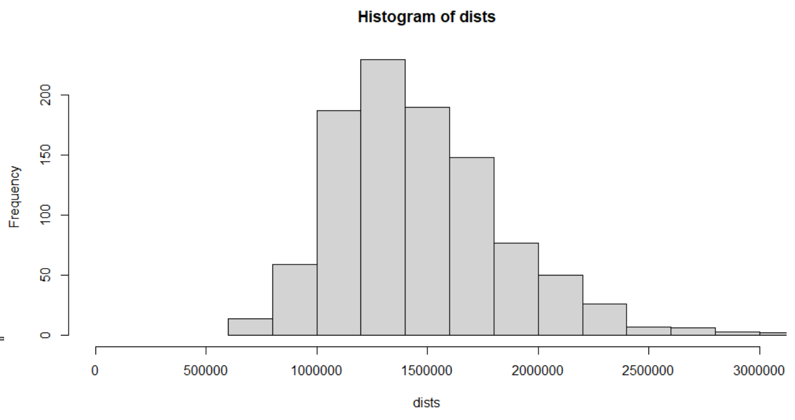


Comparing the new distances to the histogram earlier, there does seem to be a slight improvement.

I did the same for 4s.

```
params4=readRDS("params4.rds")
best_fits4=readRDS("best_fits4.rds")
stroke=readRDS("final_assessment/stroke.RDS")
findnns=readRDS("final_assessment/findnns.rds")

gp=params4[,best_fits4]


gp=t(gp)
c=var(gp)
e=eigen(c)
p=gp %*% e$vectors
plot(e$values)
mp=apply(p,2,mean) #computes the mean for each eigenvectors

#this code generated 10000 random values for each eigenvector
#the random values come from a Normal distribtuion
ranv = matrix(rnorm(100000),10,10000)
ranv = ranv * sqrt(e$values) #sets the standard deviation
ranv = ranv+mp # sets the mean for each eigenvectros

#this line reconstructs the parameter vectors from the #eignvectors
#ranv3 contains the 10 parameters for the model
ranv3=t(ranv)%*% t(e$vectors)

b4=matrix(0,256,10000)
# x1, x2, y1, y2, theta1, theta2, theta3, len1, len2, len3
# 1  2  3  4      5      6      7    8    9    10
for (i in 1:10000)
{
x1=ranv3[i,1]
x2=ranv3[i,2]
y1=ranv3[i,3]
y2=ranv3[i,4]
theta1=ranv3[i,5]
theta2=ranv3[i,6]
theta3=ranv3[i,7]
len1=ranv3[i,8]
len2=ranv3[i,9]
len3=ranv3[i,10]
width1=runif(1,0.5,3)
width2=runif(1,0.5,3)
width3=runif(1,0.5,3)

s1=stroke(x1,y1,theta1,len1,width1)
s2=stroke(x1,y1,theta2,len2,width2)
s3=stroke(x2,y2,theta3,len3,width3)

im=s1+s2+s3
im[im > 1]=1  #this thresholds the image so that the max brightness is 1

im = im*256
im = im[,16:1] #we have to turn the image upside-down so that it matches the real digits

b4[,i]=im
}
d4=scan("digits/Zl4d.dat",nlines=1000,n=256000)
d4=matrix(d4,256,1000)

proc.time()
nbrs4pca = matrix(0,40,1000)
for(i in c(1:1000))nbrs4pca[,i]=findnns(b4,d4[,i])
proc.time()

nearestpca4 = b4[,nbrs4pca[1,]]
dim(nearestpca4)
distspca4 = nearestpca4 - d4
distspca4 = apply(distspca4^2,2,sum)

hist(distspca4)
```

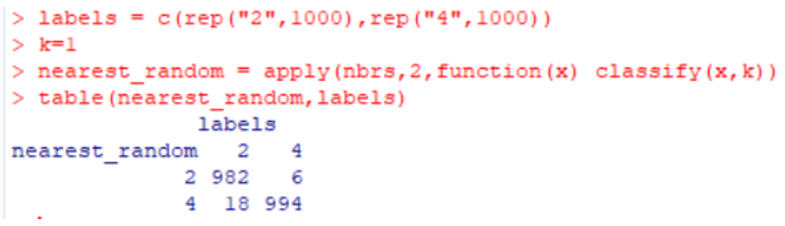




The histograms are very similar, there might be a very marginal improvement with the PCA model 4s.

I found the confusion matrix for the model 2s and 4s.

```
findnns=readRDS("final_assessment/findnns.rds")
classify = function(nns,k)
{
  labels = c(rep("2",10000),rep("4",10000))
  digits= labels[nns[1:k]] #we include the first nearest neighbour
  t=table(digits)
  m=which.max(t)
  names(m)
}

b2=readRDS("pca_b2.rds")
b4=readRDS("pca_b4.rds")

d2=scan("digits/Zl2d.dat",nlines=1000,n=256000)
d2=matrix(d2,256,1000)
d4=scan("digits/Zl4d.dat",nlines=1000,n=256000)
d4=matrix(d4,256,1000)

d=cbind(d2,d4)
b=cbind(b2,b4)

nbrs = matrix(0,40,2000)
for(i in c(1:2000))nbrs[,i]=findnns(b,d[,i])
```



```
> labels = c(rep("2",1000),rep("4",1000))
> k=1
> nearest_random = apply(nbrs,2,function(x) classify(x,k))
> table(nearest_random,labels)
              labels
nearest_random   2   4
             2 982   6
             4  18 994
```

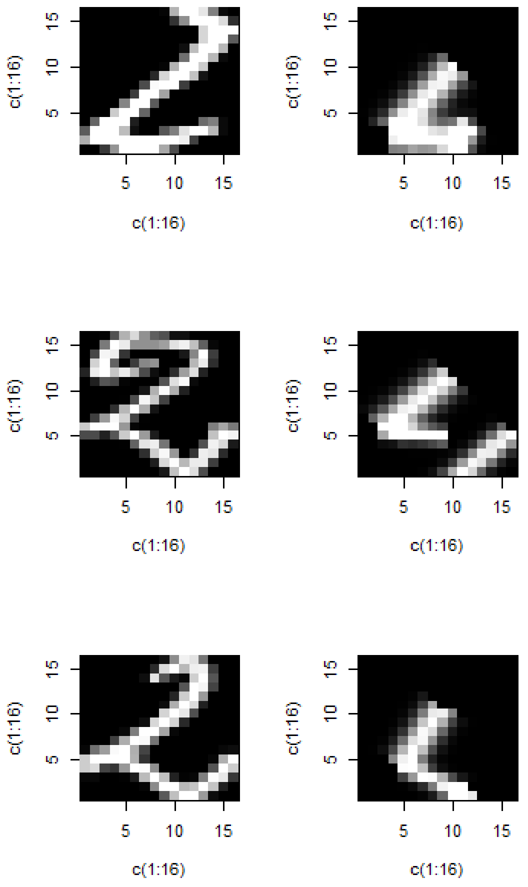
Compared to the models in Q3, which had a confusion matrix of

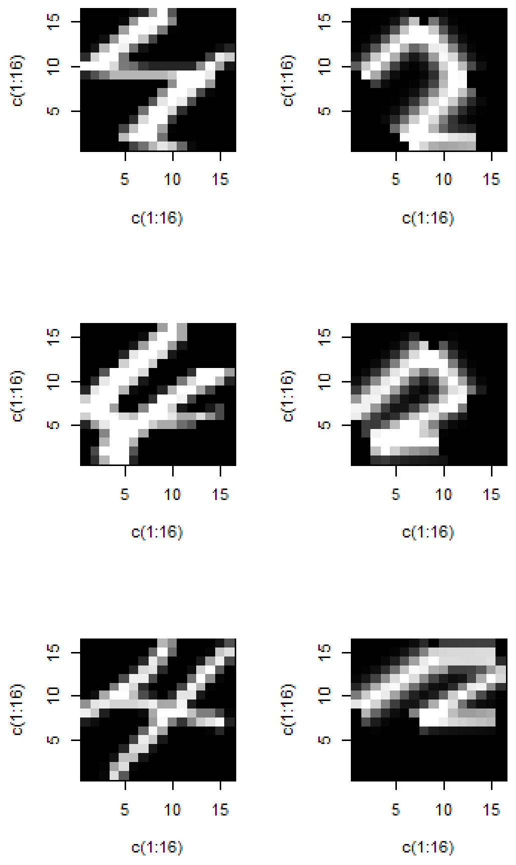
	Real 2	Real 4
Random 2	979	4
Random 4	21	996

98.75% accuracy for the old model, 98.8% for the PCA model, so no significant increase in accuracy.

Finding the misclassified digits.

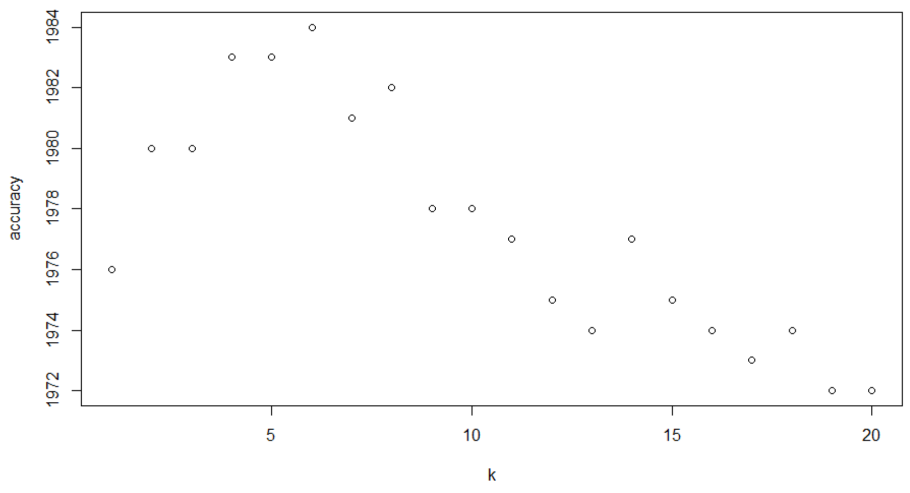
```
misclassified = which(labels == "4" & nearest_random == "2")

par(mfrow=c(3,2))
for(i in misclassified[1:3])
{
 z=matrix(d[,i],16,16)
 image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
 z=matrix(b[,nbrs[1,i]],16,16)
 image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
}
```





Finding the accuracy with different numbers of k

```
accuracy=c()
for (k in 1:20)
{
nearest_random = apply(nbrs,2,function(x) classify(x,k))
accuracy=append(accuracy,sum(diag(table(nearest_random,labels))))
}
plot(seq(1,20),accuracy,xlab="k")
```

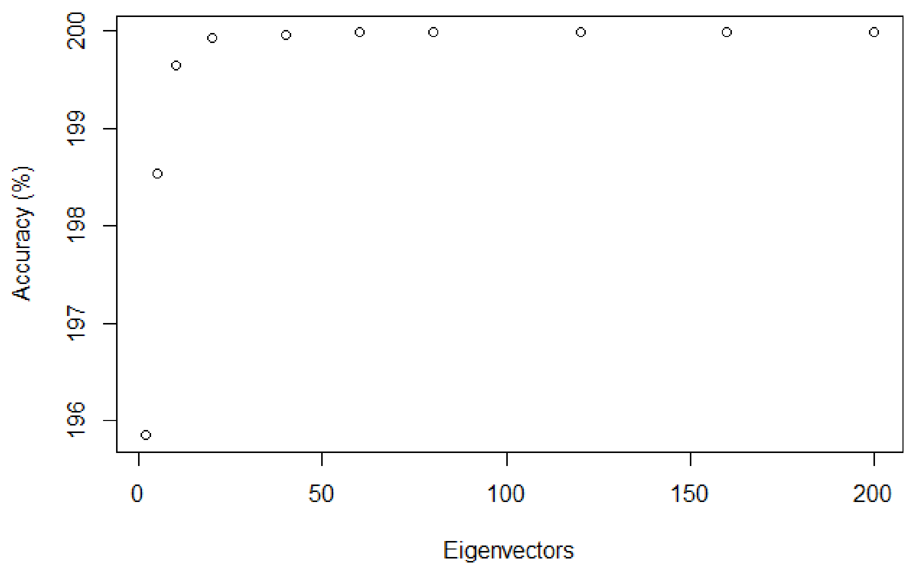


6 seems to be best value for k.

Question 7

I used the box method to classify the random digits, to speed up execution time. I used k = 6, as it was the most accurate when classifying the real digits.

```
b=t(b)
c=var(b)
e=eigen(c)
plot(e$values)
p=b %*% e$vectors[,1:256]

p=t(p)


labels=c(rep("2",10000),rep("4",10000))
acc=c()


for(n in c(2,5,10,20,60,80,120,160,200))
{
 nbrsbox = matrix(0,40,20000)
 system.time(
 for(i in c(1:20000))
 {
  box=which(abs(p[1,]-p[1,i]) < 600 & abs(p[2,]-p[2,i]) < 500)
  nnb=findnns(p[1:n,box],p[1:n,i])
  nbrsbox[,i]=box[nnb]
 }
 )
 results = apply(nbrsbox,2,function(x) classify(x,2)) # k = 6
 t1=table(results,labels)
 print(t1)
 acc=append(acc,sum(diag(t1)))
}
plot(c(2,5,10,20,40,60,80,120,160,200),acc/200,xlab="Eigenvectors",ylab="Accuracy (%)")
```



The accuracy rises very quickly and levels off with about 20 eigenvectors. The more eigenvectors you include, the longer execution time is going to be, so between 20 and 40 eigenvectors seems to be the ideal number.

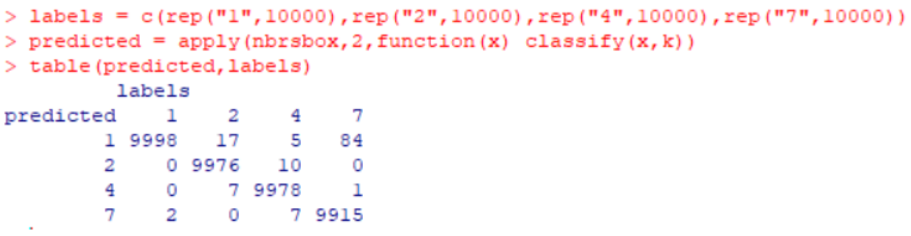
Question 8

I loaded in the matrices for the random 1s, 2s, 4s and 7s.

I used the box method again to find the nearest neighbours, with k set to 6.

I then found the confusion matrix

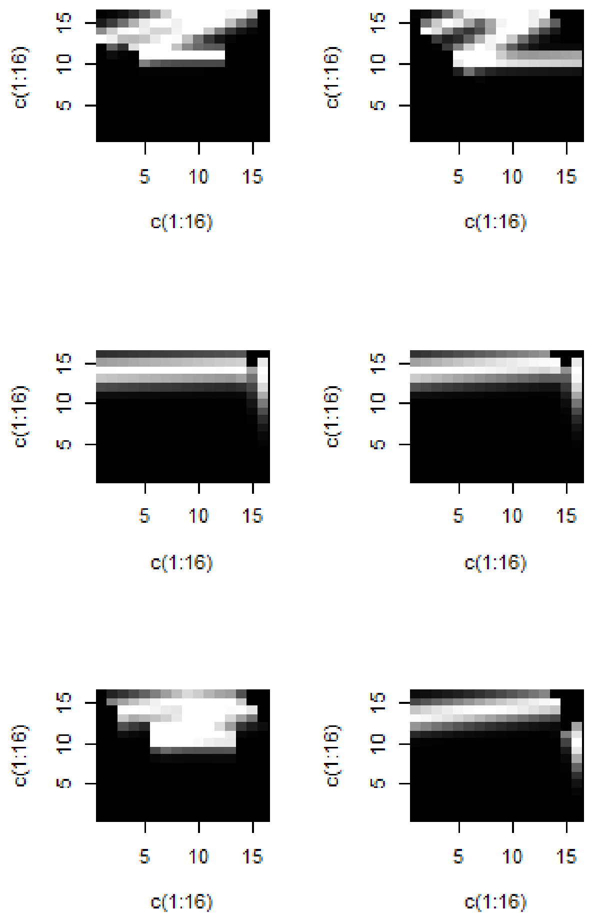
```
classify = function(nns,k)
{
  labels = c(rep("1",10000),rep("2",10000),rep("4",10000),rep("7",10000))
  digits= labels[nns[1:k]] #we include the first nearest neighbour
  t=table(digits)
  m=which.max(t)
  names(m)
}

# Load in random digits
b1=readRDS("b1.rds")
b2=readRDS("pca_b2.rds")
b4=readRDS("pca_b4.rds")
b7=readRDS("b7.rds")

b=cbind(b1,b2,b4,b7)

b=t(b)
c=var(b)
dim(c)
e=eigen(c)
plot(e$values)
pb=b %*% e$vectors[,1:20]

pb=t(pb)

nbrsbox = matrix(0,40,40000)
for(i in c(1:40000))
{
  box=which(abs(pb[1,]-pb[1,i]) < 600 & abs(pb[2,]-pb[2,i]) < 500)
  nnb=findnns(pb[1:20,box],pb[1:20,i])
  nbrsbox[,i]=box[nnb]
  if(i %% 1000 == 0) # indicator to see progress
  {
    print(i)
  }
}
```



```
> labels = c(rep("1",10000),rep("2",10000),rep("4",10000),rep("7",10000))
> predicted = apply(nbrsbox,2,function(x) classify(x,k))
> table(predicted,labels)
         labels
predicted    1    2    4    7
        1 9998   17    5   84
        2    0 9976   10    0
        4    0    7 9978    1
        7    2    0    7 9915
```

I used k = 6 instead of k = 1, so I looked at one example when a 2 was misclassified as a 1.

```
misclassified = which(labels == "2" & predicted == "1")

par(mfrow=c(3,2))

i=misclassified[1]
z=matrix(b[,i],16,16)
image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
z=matrix(b[,nbrsbox[2,i]],16,16)
image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
z=matrix(b[,nbrsbox[3,i]],16,16)
image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
z=matrix(b[,nbrsbox[4,i]],16,16)
image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
z=matrix(b[,nbrsbox[5,i]],16,16)
image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
z=matrix(b[,nbrsbox[6,i]],16,16)
image(c(1:16),c(1:16),z[,c(16:1)],col=gray(c(0:255)/255))
```



The top left digit is the misclassified digit in this case, and the other 5 digits are its nearest neighbours. There a few 1s, whose upper stroke matches the unusually high position of the 2.

Using different values for k might improve the accuracy.