HW10

Ines Pancorbo

3/20/2020

2. a)

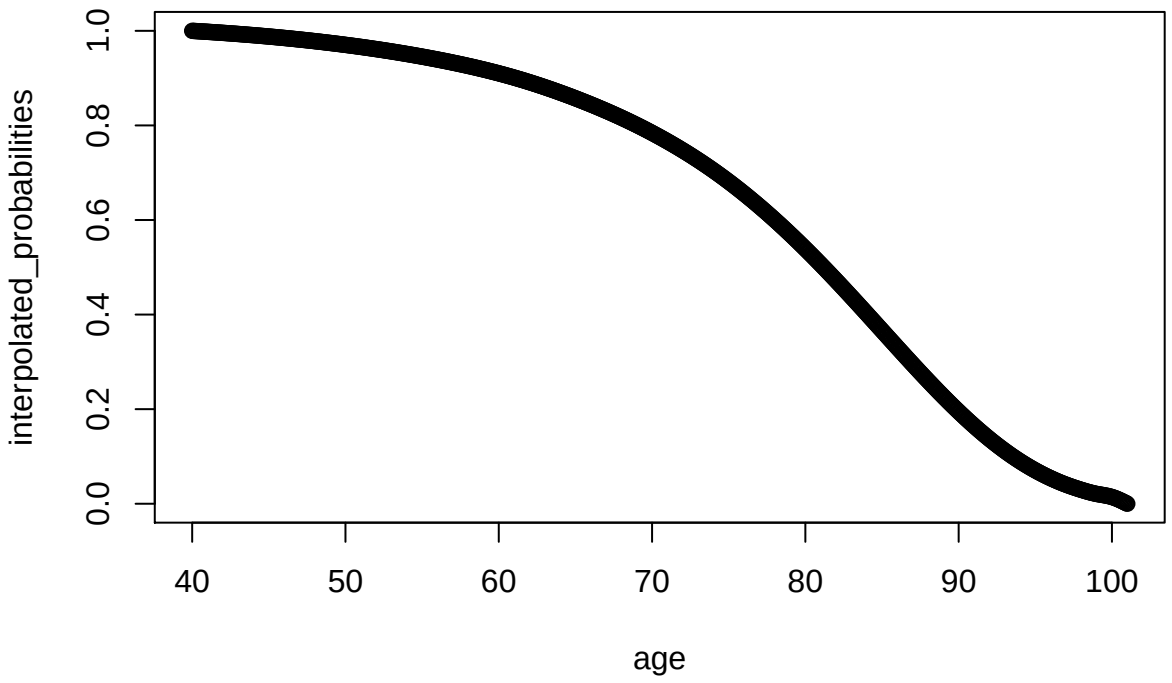
```
library(readxl)
US_Life_Expectancy_2003 <- read_excel("US Life Expectancy 2003.xls")
#View(US_Life_Expectancy_2003)

# new column with probability of not dying at age t
US_Life_Expectancy_2003$px <- 1 - US_Life_Expectancy_2003$qx
#View(US_Life_Expectancy_2003)
```

```
mult <- function(x){
  prod <- 1
  for (i in 41:x){
    prod <- prod*US_Life_Expectancy_2003$px[i]
  }
  return(prod)
}
```

```
# We want the probabilities the 40 - year old lives past 40, 41, 42, ..., 100.
# The 40 - year old has a probability of living past 40 if he/she does not die at 40,
# the 40 - year old has a probability of living past 41 if he/she does not die at 40,
# and at 41, etc.
# In general, the probability of living past 40+t is the product of the probabilities
# of not dying at 40, not dying at 41, not dying at 42, ..., not dying at 40+t.

N <- nrow(US_Life_Expectancy_2003)
US_Life_Expectancy_2003$probabilities <- NA
US_Life_Expectancy_2003$probabilities[40] <- 1

for (i in 41:N){
  US_Life_Expectancy_2003$probabilities[i] <- mult(i)
}

probabilities <- US_Life_Expectancy_2003$probabilities[40:N]
```

```
# cubic spline with t values (0,1,2,...,61) and
# corresponding probabilities of living past 40+t
L <- splinefun(x=seq(0,length(probabilities)-1,1),
               y=probabilities,
               method="natural")

# interpolate for other t values
```

```r
interpolated_probabilities <- L(seq(0, length(probabilities)-1, 1/12))
age <- 40 + seq(0, length(probabilities)-1, 1/12)

plot(age, interpolated_probabilities)
```



2. b)

For each month the 40 year old lives, he/she makes a 200 payment. The PV of that payment is PV $= 200e^{-0.05i/12}$, i being the month past 40 (ex: present is month 0, payment due next month correpsonds to payment on month 1, payment due the following month corresponds to payment on month 2, etc).

The probability that he/she makes a payment on month i is $L(i/12)$, the probability that he/she is alive on that month. We have the data for month 12, month 24, month 48, etc, and we interpolate using the spline for the probabilities that he/she makes a payment on the remainding month i's. Theoretically, i goes up to infinity. So the infinite sum of the PV of the payment due on month i times the probability he/she is alive on month i is the expected value of the PV of the payments. In other words: E[PV] $= \sum_{i=1}^{\infty} L(i/12)200e^{-0.05i/12}$.

Since in practice we cannot sum to ∞, we can assume that the maximum he/she can live is 101 years (given our data) $\implies$ 61 years $\implies$ 732 months. So we sum up to $m = 732$.

```r
interpolated_probabilities <- interpolated_probabilities[2:length(interpolated_probabilities)]

months <- seq(1,732,1)
PV <- 200*exp(-0.05*(months)/12)

cat("E[PV] =", sum(PV*interpolated_probabilities))
```

```
## E[PV] = 39373.04
```

3. a)

```
# power iteration and norm functions

norm <- function(x){
  sqrt(sum(x^2))
}


power_iteration <- function(A, start) {
  start1 <- matrix(start[,1],nrow=ncol(A), ncol = 1)/norm(start[,1])
  start2 <- matrix(start[,2],nrow=ncol(A), ncol = 1)/norm(start[,2])
  RQ1 <- t(start1) %*% A %*% start1
  RQ2 <- t(start2) %*% A %*% start2

  repeat {

    start <- A %*% start
    start <- qr.Q(qr(start))

    start1 <- matrix(start[,1],nrow=ncol(A), ncol = 1)/norm(start[,1])
    start2 <- matrix(start[,2],nrow=ncol(A), ncol = 1)/norm(start[,2])

    new_RQ1 <- t(start1) %*% A %*% start1
    new_RQ2 <- t(start2) %*% A %*% start2

    if (abs(new_RQ1 - RQ1) < 10^-10 && abs(new_RQ2 - RQ2) < 10^-10){
      break
    }
    else{
      RQ1 <- new_RQ1
      RQ2 <- new_RQ2
    }
  }

  return (list(eigenvectors=start, lambda1=RQ1, lambda2=RQ2))
}
```

```
setwd("/Users/inespancorbo/MATH504/HW7")
mtrain <- as.matrix(read.csv("mnist_train.csv", header=F))
mtest <- as.matrix(read.csv("mnist_test.csv", header=F))
```

```
x <- mtrain[,2:ncol(mtrain)]
x <- x/250

mean <- colMeans(x)
x <- x - rep(mean, rep.int(nrow(x), ncol(x)))

data <- x[1:1000,]

# covariance matrix
omega <- t(data) %*% data

# starting point
```

```
start <- matrix(runif(ncol(omega)*2), nrow=ncol(omega), ncol=2)

# power iteration
power_iteration_result <- power_iteration(omega,start)
```

```
# Double-check power iteration
cat(power_iteration_result$lambda1, power_iteration_result$lambda2, "\n")
```

```
## 5330.832 4171.797
```

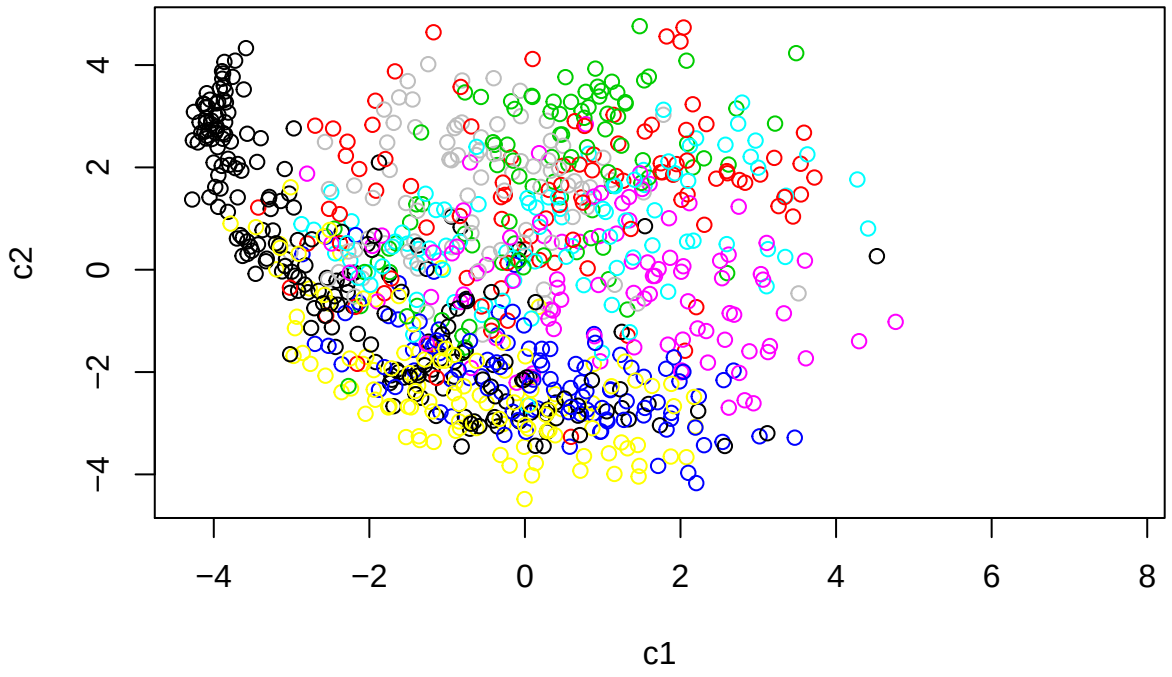
```
cat(eigen(omega)$values[order(abs(eigen(omega)$values), decreasing = T)][1:2])
```

```
## 5330.832 4171.797
```

From above one can see that convergence happened. So we can use the two eigenvectors calculated via power_iteration().

```
v1 <- power_iteration_result$eigenvectors[,1]
v2 <- power_iteration_result$eigenvectors[,2]
```

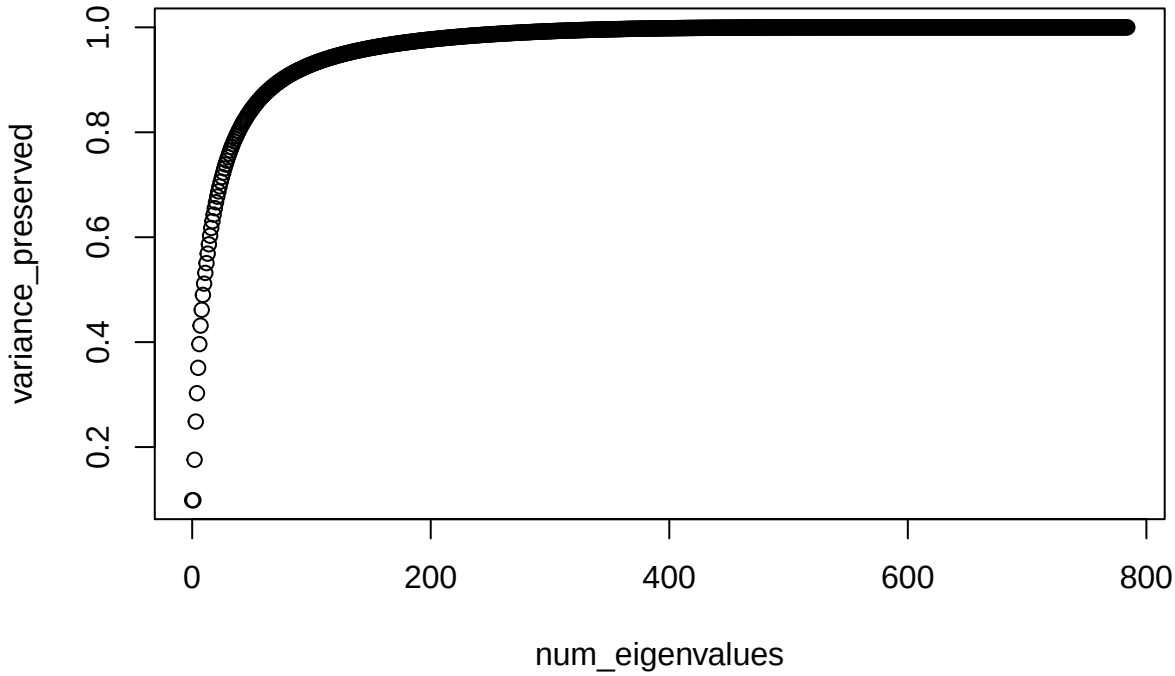
```
# projecting and plotting
color <- mtrain[1:1000,1]
c1 <- data %*% v1
c2 <- data %*% v2
plot(c1, c2, col=color)
```



3. b) i.

```
total_eigenvalues <- length(eigen(omega)$values)
eigenvalues <- eigen(omega)$values[order(abs(eigen(omega)$values), decreasing = T)]
eigenvectors <- eigen(omega)$vectors[,order(abs(eigen(omega)$values), decreasing = T)]
```

```r
variance <- function(k){
  return (sum(eigenvalues[1:k])/sum(eigenvalues))
}

num_eigenvalues <- seq(0, total_eigenvalues, 1)
variance_preserved <- sapply(num_eigenvalues, variance)

plot(num_eigenvalues, variance_preserved)
```



The graph above shows the fraction of the total variance preserved by all possible k-dimensional PCAs (k being the number of eigenvalues you keep)

3. b) ii.

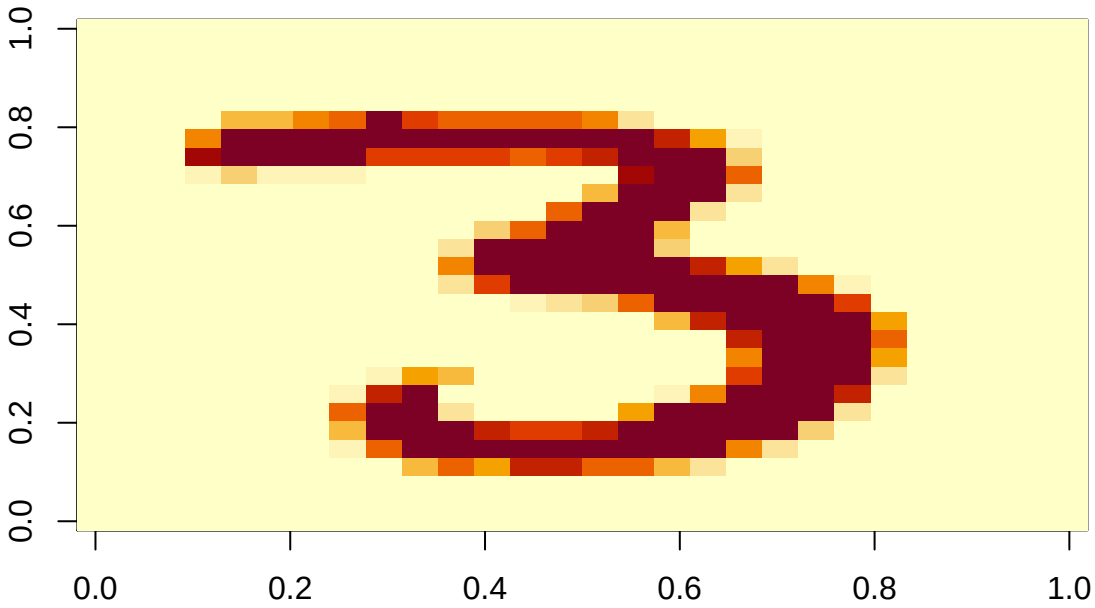
I will choose k = 1, k = 5, k = 20, k = 50, k = 70, k = 90.

```r
dimensions <- c(1,5,20,50,70,90)

for (k in dimensions){
  cat("dimension = ",k,"|| variance preserved = ", variance_preserved[k], "\n")
}
```

```
## dimension =  1 || variance preserved =  0.09861799
## dimension =  5 || variance preserved =  0.3026929
## dimension =  20 || variance preserved =  0.6553314
## dimension =  50 || variance preserved =  0.8411197
## dimension =  70 || variance preserved =  0.8896523
## dimension =  90 || variance preserved =  0.9183383
```

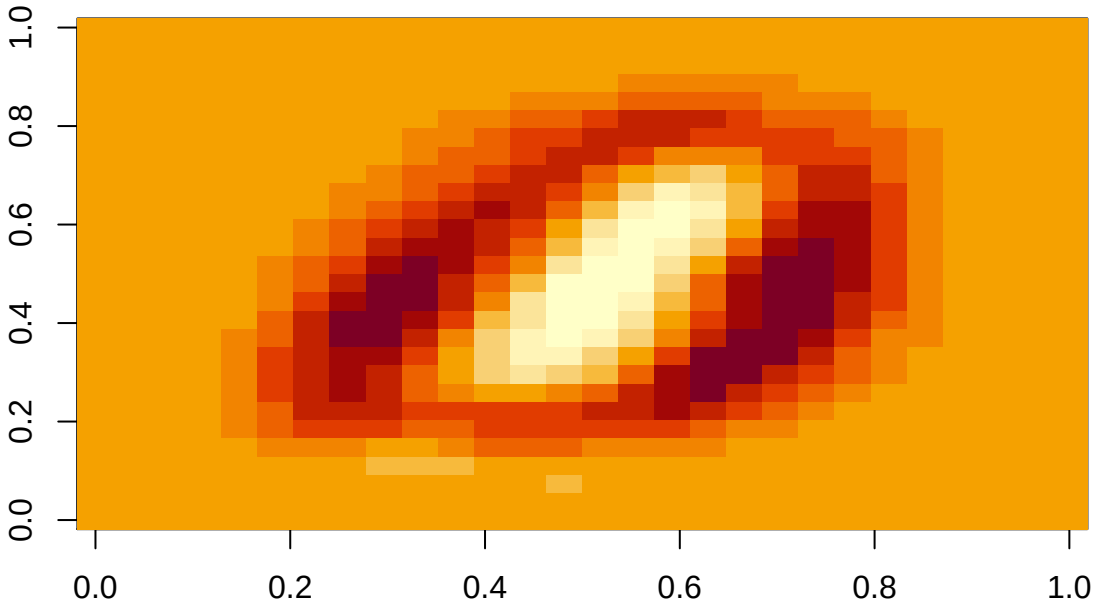
```r
show_image <- function(m, oriented=T)
{
  im <- matrix(m, byrow=T, nrow=28)
```

```
  if (oriented) {
    im_orient <- matrix(0, nrow=28, ncol=28)
    for (i in 1:28)
      im_orient[i,] <- rev(im[,i])

    im <- im_orient
  }
  image(im)
}

image13 <- mtrain[13,2:ncol(mtrain)]
show_image(image13)
```

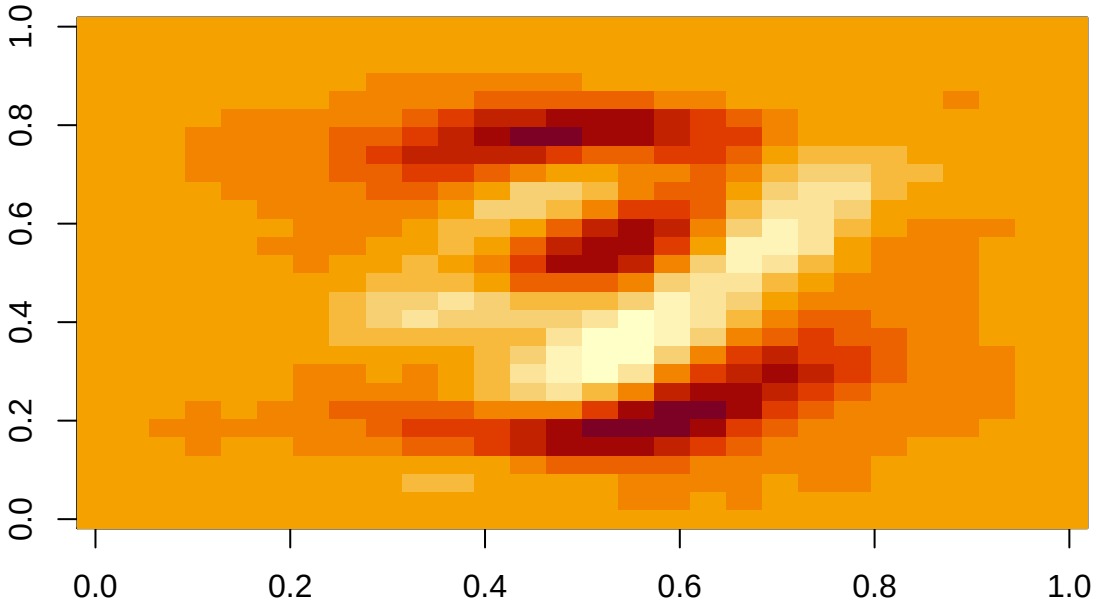



```
image13 <- function(k){
  v <- eigenvectors[,1:k]
  c <- t(v) %*% x[13,]
  image13 <- v %*% c
  show_image(image13)
}
```

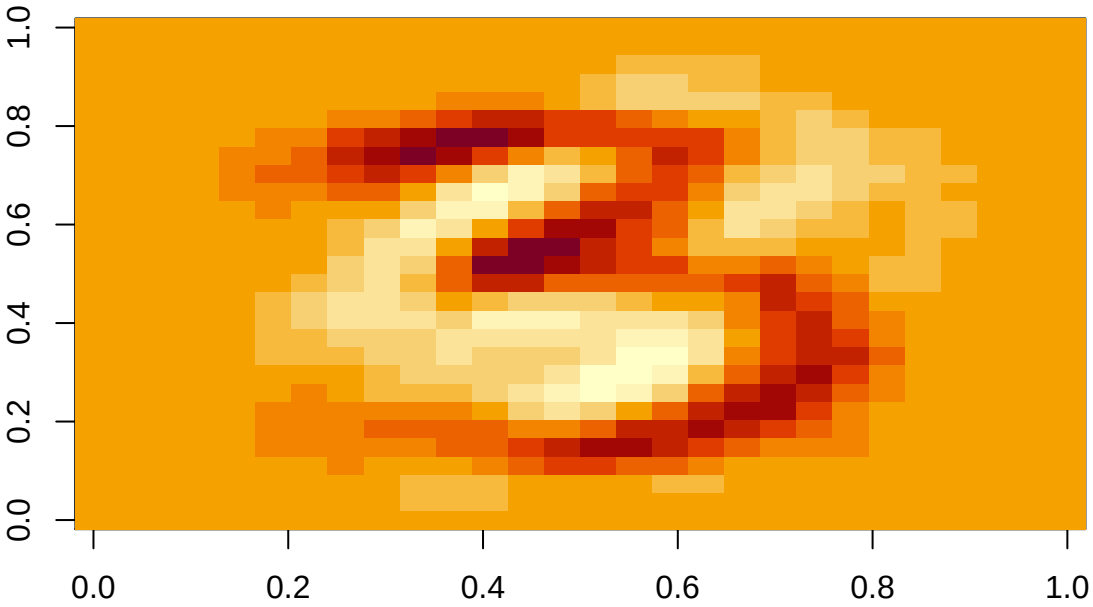
```
for (k in dimensions){
  print(image13(k))
}
```



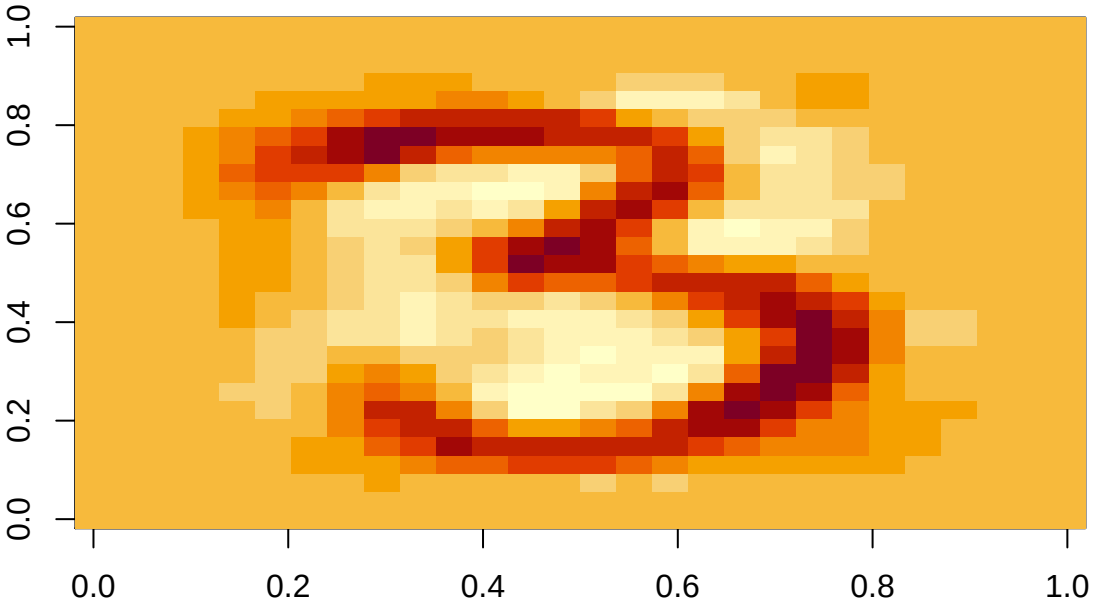

```
## NULL
```



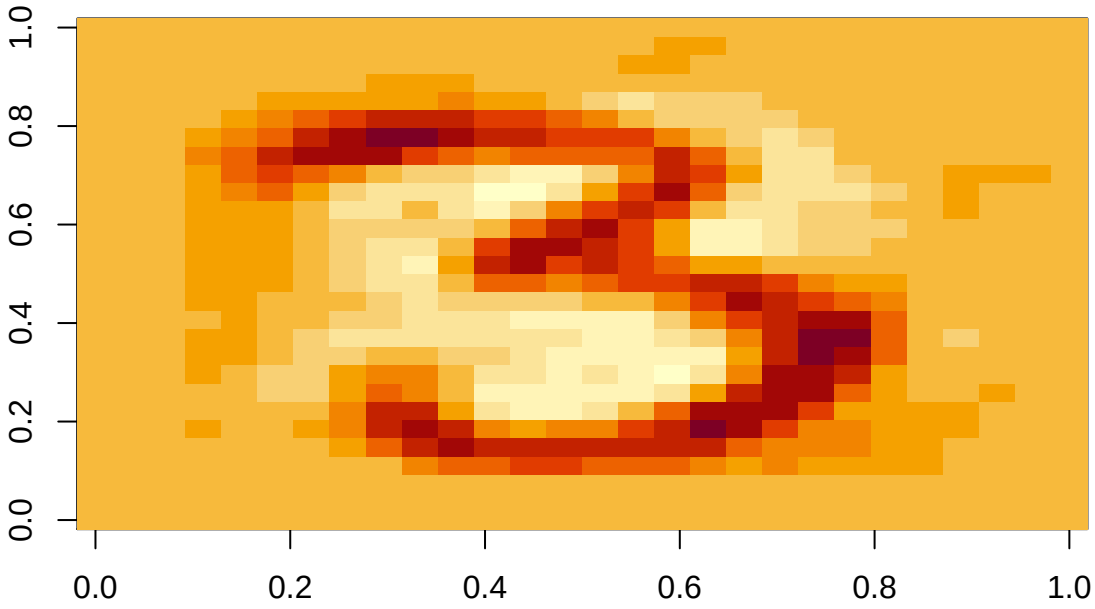

```
## NULL
```



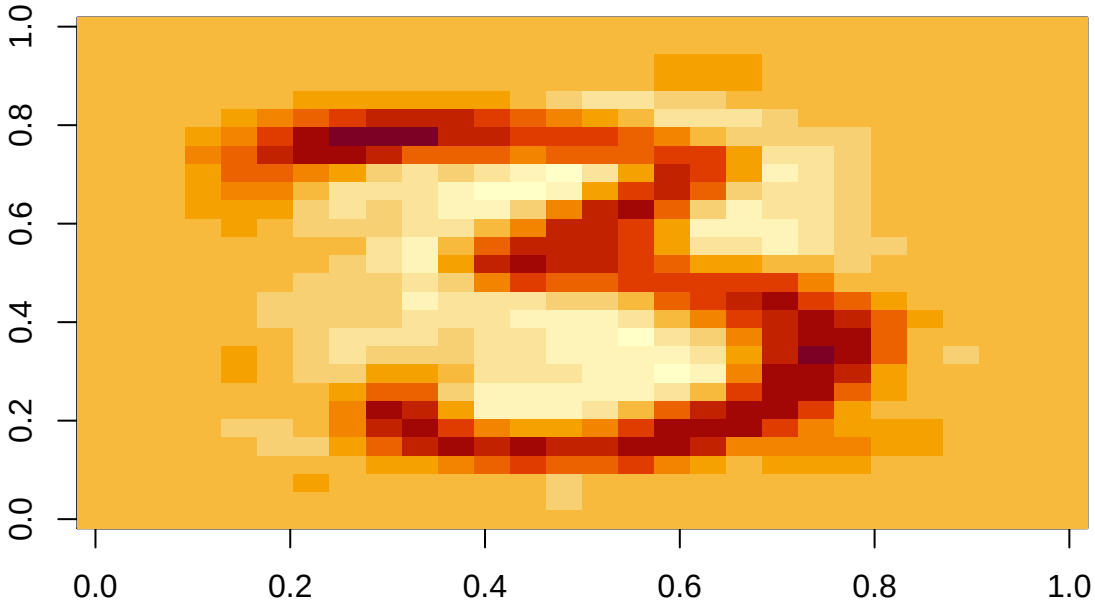

```
## NULL
```




```
## NULL
```




```
## NULL
```




```
## NULL
```

Given the above, I will choose $k = 90$ as the dimension as you can distinguish that the number is a 3. So dimension $\sim k = 90$ captures the dataset well.

3. b) iii.

```
# calculating coefficients
v <- eigenvectors[,1:90]
c <- x %*% v
```

```
# adding a column with 1s and preparing the y-variable (for accuracy)
c <- cbind(rep(1, nrow(c)), c)
y <- mtrain[,1]
```

```r
y <- ifelse(y == 3, 1, 0)
```

```r
# log likelihood function

log_L <- function(alpha, x, y) {
  sum(-(1-y)*(x %*% alpha) - log(1 + exp(-x %*% alpha)))
}


# gradient of log likelihood

grad_logL <- function(alpha, x, y) {

  # randomly choose an index
  r <- sample(1:nrow(x), 1)
  xr <- x[r,]
  yr <- y[r]

  alpha_sum <- -xr %*% alpha
  grad <- (yr - 1) * xr + xr * exp(alpha_sum)/(1+exp(alpha_sum))

  return(grad)
}

# hessian of log likelihood
#hessian_logL <- function(alpha, x){

#  h <- matrix(0, ncol=length(alpha), nrow=length(alpha))
#  N <- nrow(x)

  # randomly choose a batch
#  sample <- sample(1:N, 1000, replace=FALSE)

#  for(i in sample){
#    v <- matrix(x[i,], nrow=ncol(x), ncol=1)
#    exp_alpha_sum <- exp(-v %*% alpha)
#    h <- h - v %*% t(v) * exp_alpha_sum/(1 + exp_alpha_sum)^2
#  }
#  return (h)
#}

# stochastic newton's method

#SNM <- function(alpha,x,y,eps = 1e-10, print = FALSE) {
#  g <- grad_logL(alpha,x,y)
#  i <- 1

#  repeat{
#    h <- hessian_logL(alpha,x)
#    g <- grad_logL(alpha,x,y)
#    l <- log_L(alpha,x,y)
#    s <- 1
#    delta <- -solve(h,g)
```

```
#    if (print){
#      cat("iteration:", i, "logL value:", l, "\n")
#   }

#    while(l > log_L(alpha+ s*delta,x,y)){
#      s <-s/2
#    }

#    alpha <- alpha + s*delta
#
#    if (abs(l-log_L(alpha,x,y)) < eps){
#      break
#    }

#    i <- i + 1
#  }

#  return(list(max=alpha, iter=i))
#}


# Stochastic gradient ascent.

SGA <- function(alpha, x, y, eps = 1e-4, max_iter = 10^5){

  g <- grad_logL(alpha,x,y)
  l <- log_L(alpha,x,y)
  s <- 0.1
  iter <- 0
  x_values <- c(iter)
  y_values <- c(l)


  while(iter < max_iter){

    alpha <- alpha + s*g
    next_l <- log_L(alpha,x,y)

    # end while loop sooner
    if (abs(l-next_l) < eps){
      break
    }

    l <- next_l
    g <- grad_logL(alpha,x,y)
    iter <- iter + 1

    if(iter %% 100 == 0){
      x_values <- append(x_values, iter)
      y_values <- append(y_values, l)
    }

  }
```

```
  return(list(max=alpha, iter=iter, x_values=x_values, y_values=y_values))
}
```

```
alpha <- rep(0,ncol(c))
SNM_result <- SGA(alpha, c, y)
```

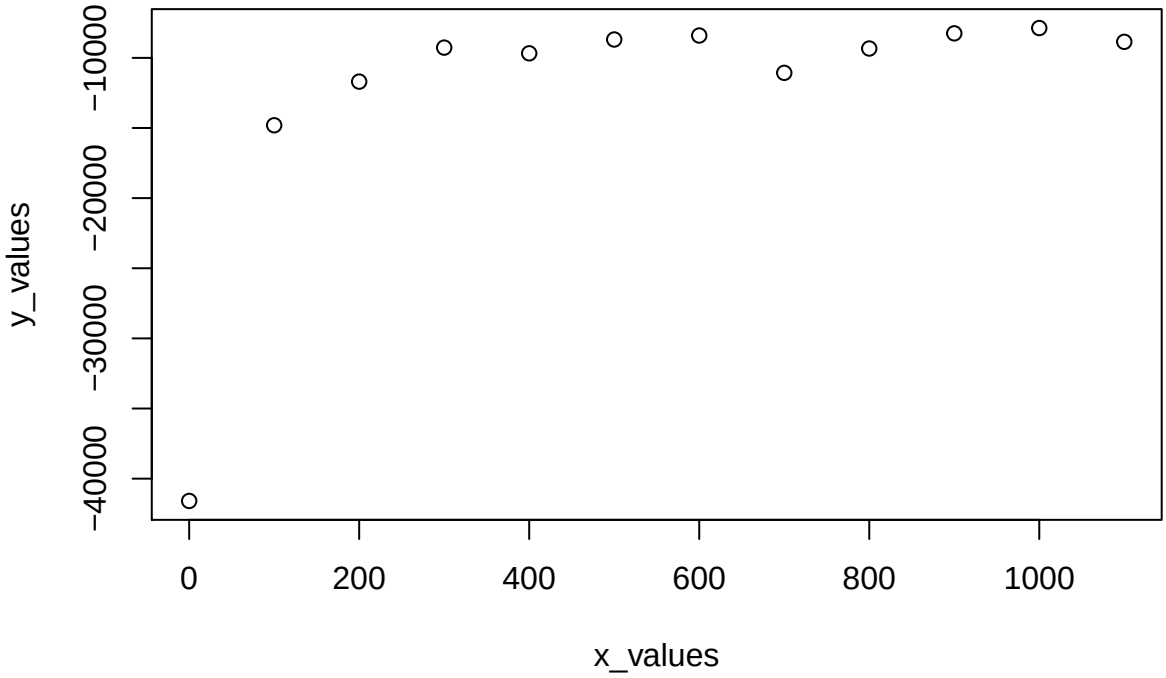
```
max <- SNM_result$max
SNM_result$iter
```

```
## [1] 1121
```

```
x_values<-SNM_result$x_values
y_values<-SNM_result$y_values
```

In terms of time, running this code, took me approx 5 minutes (it would have taken me less if I hadn't decided to append every 100 iterations), compared to approx 30ish mins in hw 7 where I was not using a stochastic gradient approach.

```
plot(x_values, y_values)
```




```
y_test <- mtest[,1]
y_test <- ifelse(y_test == 3, 1, 0)


x_test <- mtest[,2:ncol(mtest)]
x_test <- x_test/250
mean <- colMeans(x_test)
x_test <- x_test - rep(mean, rep.int(nrow(x_test), ncol(x_test)))

# calculating coefficients
c_test <- x_test %*% v
c_test <- cbind(rep(1,nrow(c_test)), c_test)
```

```r
classifier <- function(max, x, y, cutoff){
  p <- 1/(1+exp(-x %*% max))
  p <- ifelse(p >= cutoff, 1, 0)

  c <- abs(p-y)
  accuracy <- (1-sum(c)/length(y))
  return(accuracy)
}
```

```r
cat("cutoff = 0.9 || accuracy =", classifier(max, c_test, y_test, 0.9), "\n")
```

```
## cutoff = 0.9 || accuracy = 0.9546
```

In hw 7 I used a $p = 0.9$. Using the same p to be able to compare, you can see that the accuracy you get is still close to 90% when reducing dimension to $k = 90$ (in hw 7 I got an accuracy of 89.89%)