Homework3

Tianyi Fang

September 27, 2017

problem1: K-means Algorithm

1.Load in images and labels, store data as digits and labels

```
load_mnist <- function() {
  # load image files
  load_image_file <- function(filename) {
    ret = list()
    f = file(filename,'rb')
    readBin(f, 'integer', n = 1, size = 4, endian = 'big')#magic number 2051
    n    = readBin(f, 'integer', n = 1, size = 4, endian = 'big')# number of images60000
    nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')#number of rows 28
    ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')#num of col 28
    x = readBin(f, 'integer', n = n * nrow * ncol, size = 1, signed = FALSE)
    ret$x = matrix(x, ncol=nrow*ncol, byrow=TRUE)
    close(f)
    ret
  }
  # load label files
  load_label_file <- function(filename) {
    f = file(filename,'rb')
    readBin(f,'integer',n=1,size=4,endian='big')
    n = readBin(f,'integer',n=1,size=4,endian='big')
    y = readBin(f,'integer',n=n,size=1,signed=F)
    close(f)
    y
  }
  # load images
  train <<- load_image_file('train-images-idx3-ubyte')
  ##test <<- load_image_file('t10k-images-idx3-ubyte')
  # load labels
  train$y <<- load_label_file('train-labels-idx1-ubyte')
  ##test$y <<- load_label_file('t10k-labels-idx1-ubyte')
}

# helper function for visualization
show_digit <- function(arr784, col=gray(12:1/12), ...) {
  image(matrix(arr784, nrow=28)[,28:1], col=col, ...)
}
load_mnist()
#select the first 1000 rows
digits <- train$x[1:1000,]
labels <- train$y[1:1000]
dim(digits)#1000*784
```

```
## [1] 1000  784
```

```
length(labels)#1000
```

```
## [1] 1000
```

```
# test of subset of data
#fit = randomForest::randomForest(y ~ ., data = train[1:1000, ])
#fit$confusion
#test_pred = predict(fit, test)
#mean(test_pred == test$y)
#table(predicted = test_pred, actual = test$y)
```

2.Write a function to perform K-means clustering

Set initial variables k, cluster center, e_distance, assignment matrix r, loss vector to record value of loss function of each k-mean interations

```
my_kmeans <- function(digits, k, N){
  #n times over random cluster initialization
  overall_loss_matrix <- matrix(NA, ncol=100, nrow = N)#every element is overalllossfunction of a time

  for(i in 1:N){
    #assign initial variables
    cluster_center <- matrix(rep(0, k*784), ncol = 784, nrow = k)
    set.seed(i)
    #randomly select k images as initial clusters
    initial_index <- sample(1:1000, size = k)
    cluster_center <- digits[initial_index, ]
    e_distance <- matrix(rep(0, k*1000), ncol = 1000, nrow = k)
    zero_r <- matrix(rep(0, k*1000), ncol = 1000, nrow = k)
    loss_value<- rep(NA,100)#every element is lossfunction of cluster j

    #set stop criteria for while loop
    old_min_index_xc <- rep(-Inf, 1*1000)
    new_min_index_xc <- rep(0, 1*1000)
    loop <- 0 #while loop time
    while(!all(old_min_index_xc==new_min_index_xc)){
    #repeat the following steps until old_r == new_r, which means assignment matrix doesnot change anym
      new_r <- zero_r
      old_min_index_xc <- new_min_index_xc
      #reset new_r to 0, otherwise, it will be overwritten every loop
      loop <- loop + 1
      #form distance matrix
      for(j in 1:k){
        e_distance[j,] <- rowSums((t(t(digits)-cluster_center[j,]))^2)
      }
      #get assignment vector
      #return the cluster(which has least distance with xi) index of xi
      #return the min distance of xi to each cluster
      new_min_index_xc <- apply(e_distance, 2, which.min)
      new_min_value_xc <- apply(e_distance, 2, min)
      for(j in 1:k){
        #index_in_j <- rep(0, 1000)
        #update new cluster assignment r
        new_r[j,(new_min_value_xc %in% e_distance[j,])] <- 1
```

```
        index_in_j <- which(new_r[j, ]==1)
        position_in_j <- digits[index_in_j, ]
        #update new cluster center, if there is no obs in cluster_j, random assign a cluster mean
        if(length(index_in_j)==0){
          cluster_center[j,] <- rnorm(1*784, ncol=784)
        }else{
        cluster_center[j, ] <- colMeans(position_in_j)
        }
      }
      # for loopth loop,  loss function is the sum of all min distance(xi,cj)
      loss_value[loop] <- sum(new_min_value_xc)
    }
    #overall_loss_matrix records N row(for N trials), loop col(differ for each trial)
    overall_loss_matrix[i,] <- loss_value
  }
  result <- list(new_r, cluster_center, overall_loss_matrix)
  return(result)
}
#test
#my_kmeans(digits, 5, 5)
```

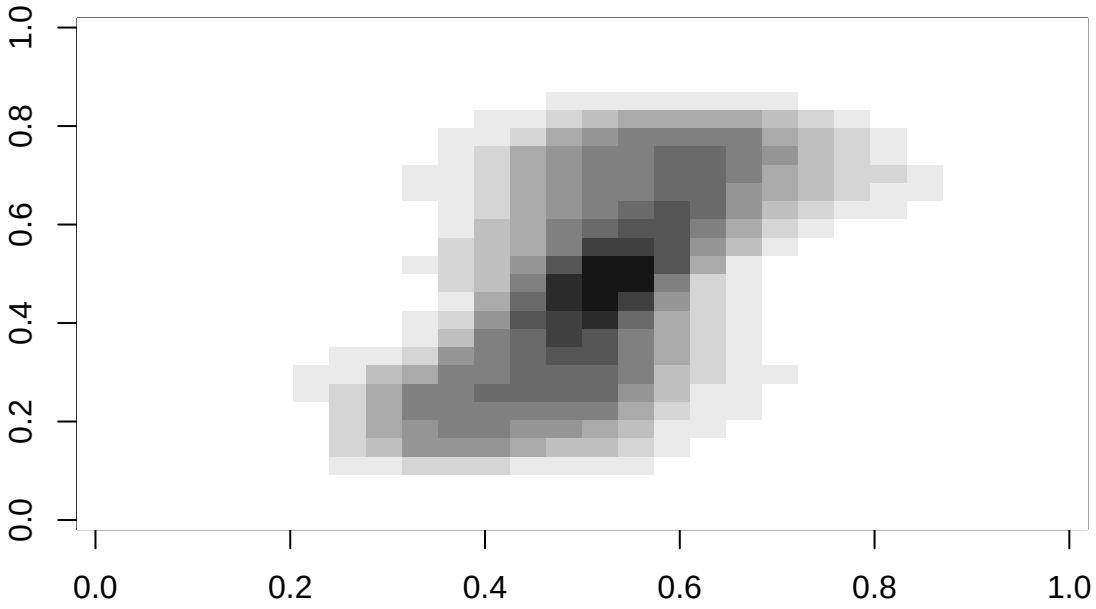
3.Explain the stopping criteria

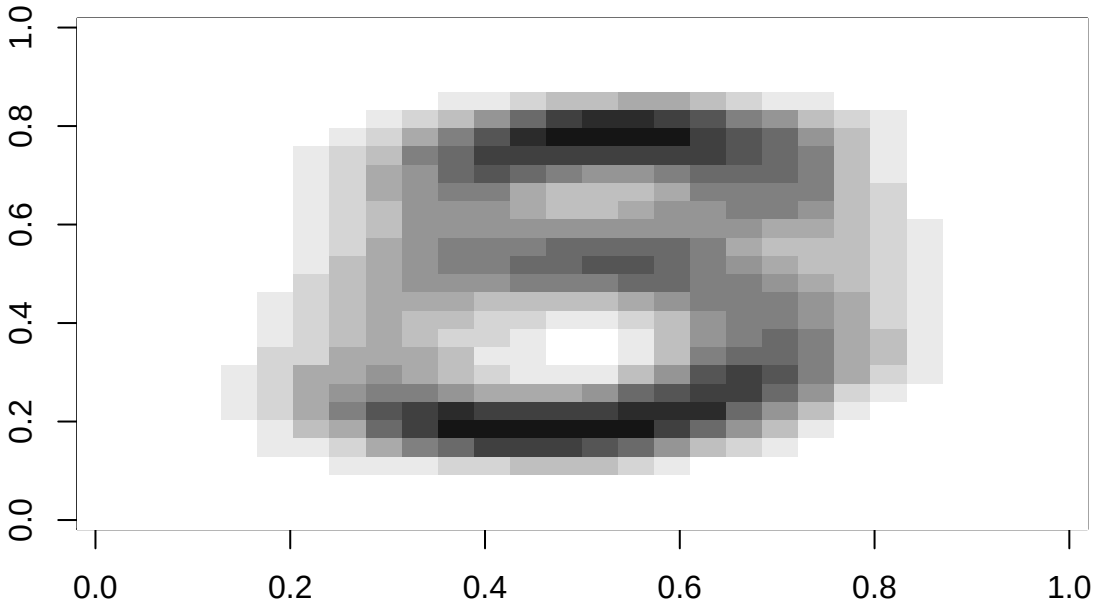
For the stopping criteria in second loop, I choose to compare cluster assignments, if the last cluster assignment is equal to the current cluster assignment, then we stop. That is, when no more images change their cluster center, the cluster centriod does not change anymore, we get the best solution. ####4.Plot the cluster means and the best loss function trends. Take a look at k=5

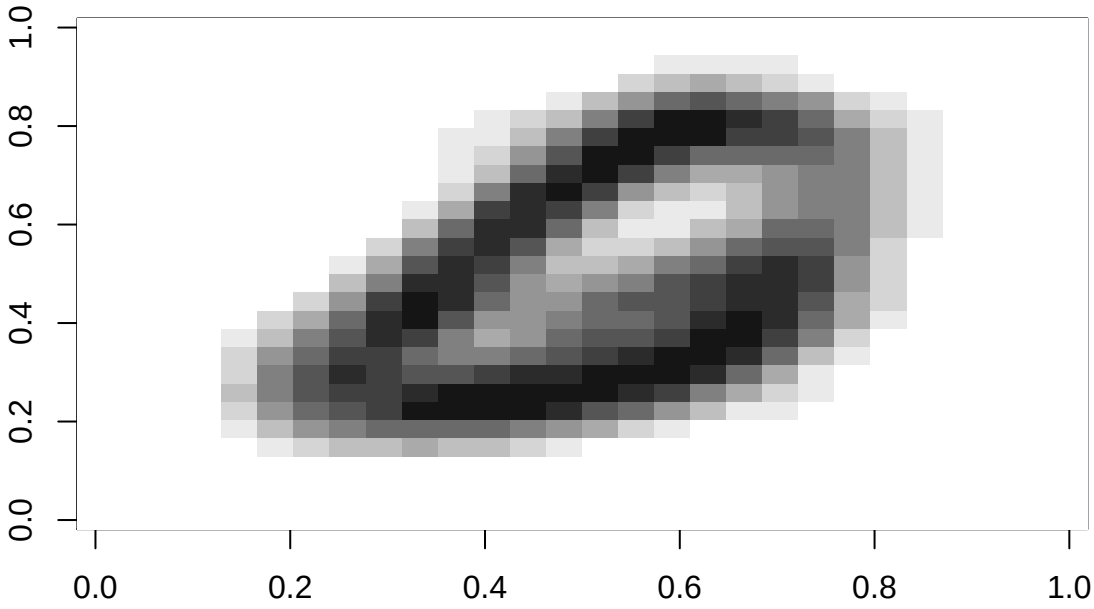
```
library(ggplot2)
```

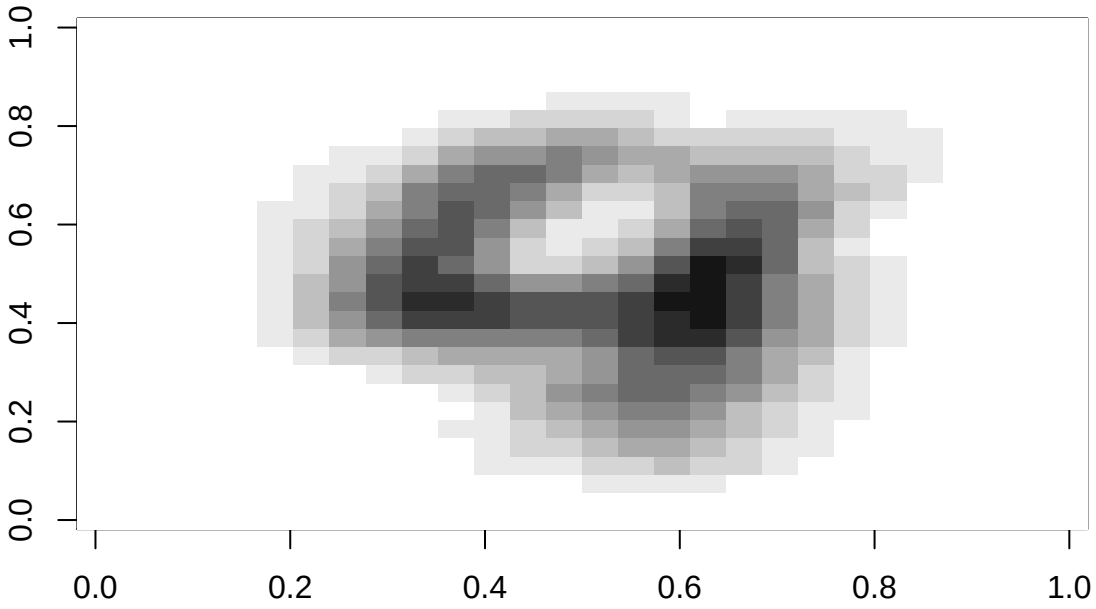
```
## Warning: package 'ggplot2' was built under R version 3.4.1
```

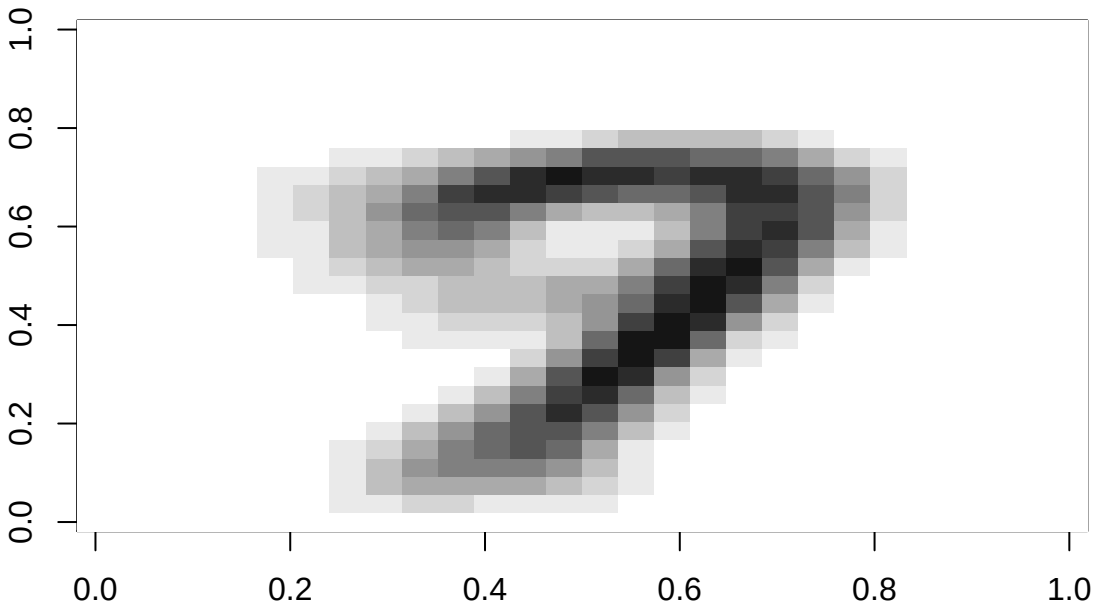
```
#k=5
k_5 <- my_kmeans(digits, 5, 25)
k_5_center <- k_5[[2]]
k_5_loss <- k_5[[3]]
#show_digit(k_5_center)
for(i in 1:5){
  show_digit(k_5_center[i,])
}
```


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

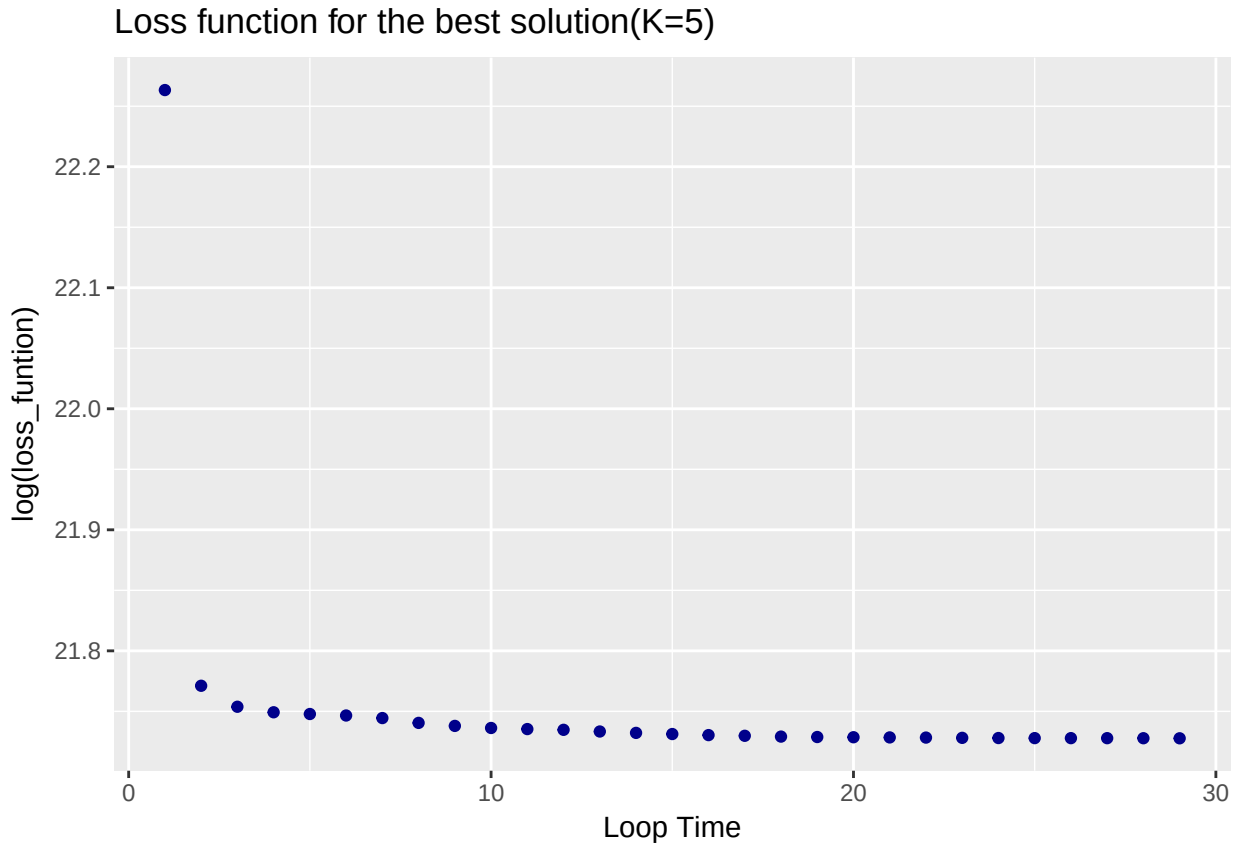

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

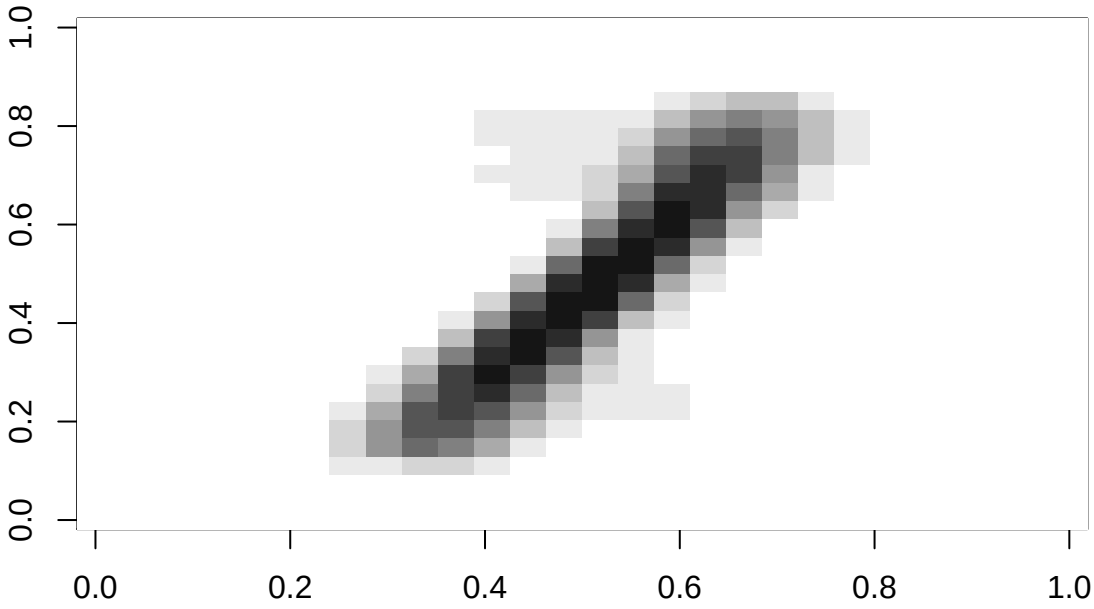

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

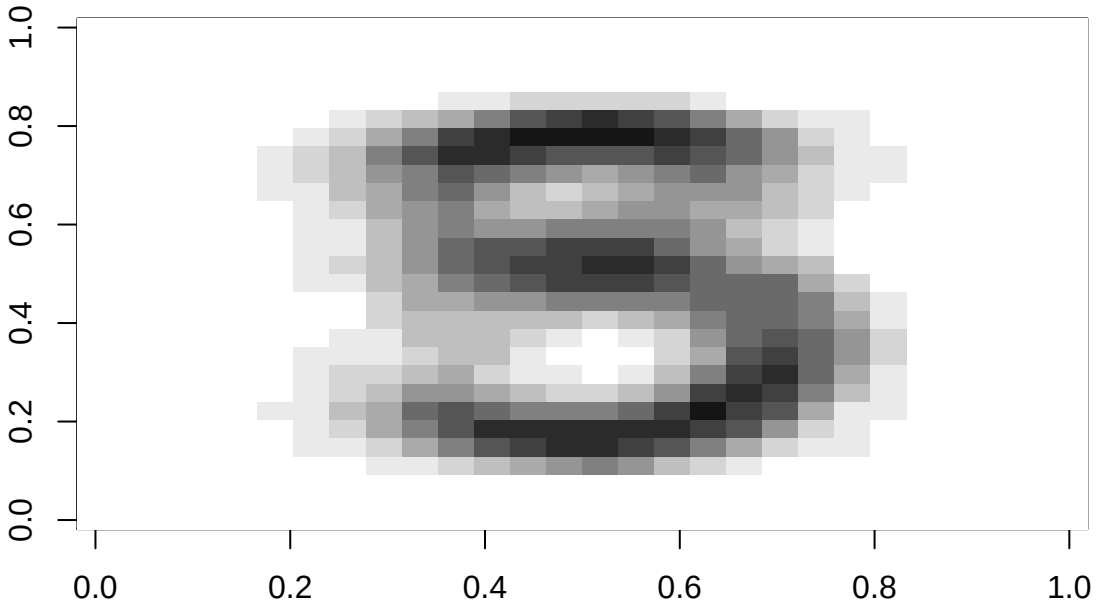


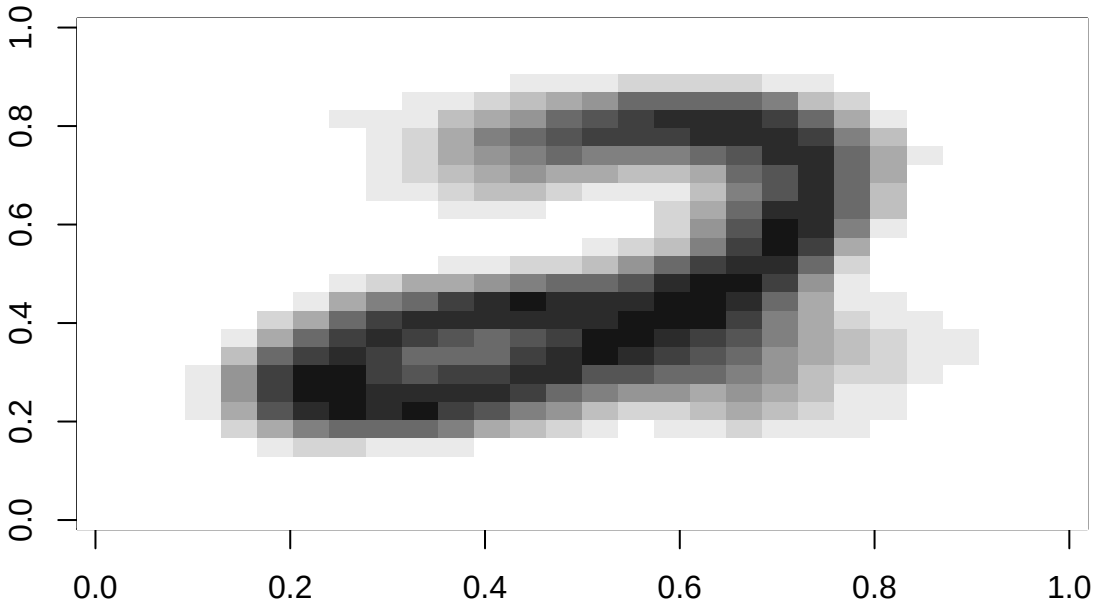

```
#find the best situation among N=25, then give the loss_function plot
k_5_loss_min <- apply(log(k_5_loss),1, min, na.rm = TRUE)
k_5_loss_best <- log(k_5_loss[which.min(k_5_loss_min), ])
k_5_loss_best <- na.omit(k_5_loss_best)

xais <- 1:length(k_5_loss_best)
best_loss_plot <- data.frame(xais, k_5_loss_best)
plot_5 <-ggplot(best_loss_plot, aes(xais, k_5_loss_best)) + geom_point(color = "darkblue") + labs(x="Lo
plot_5
```

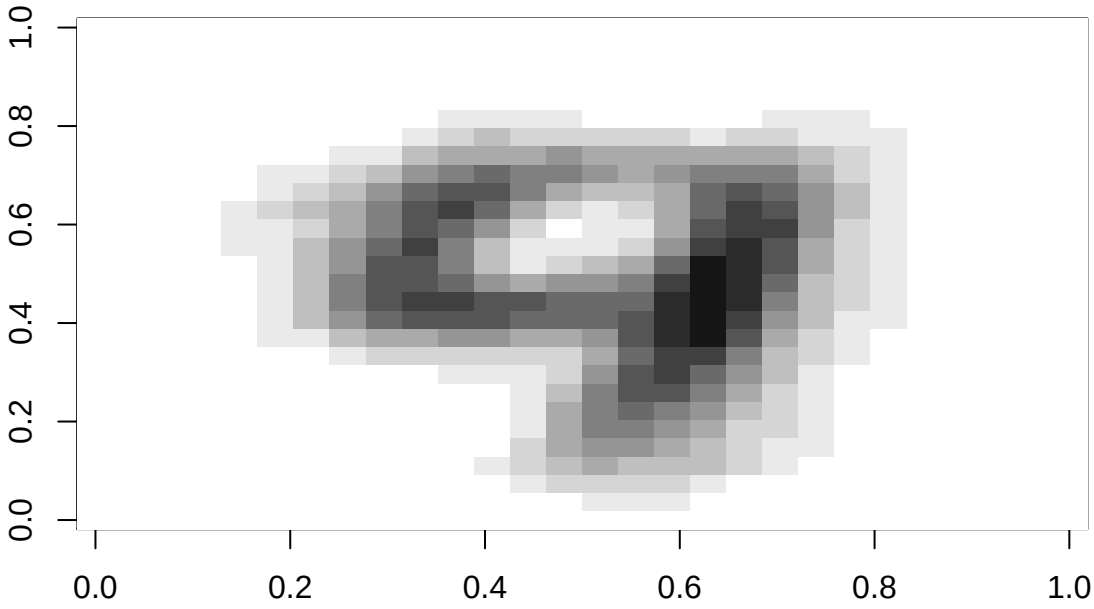


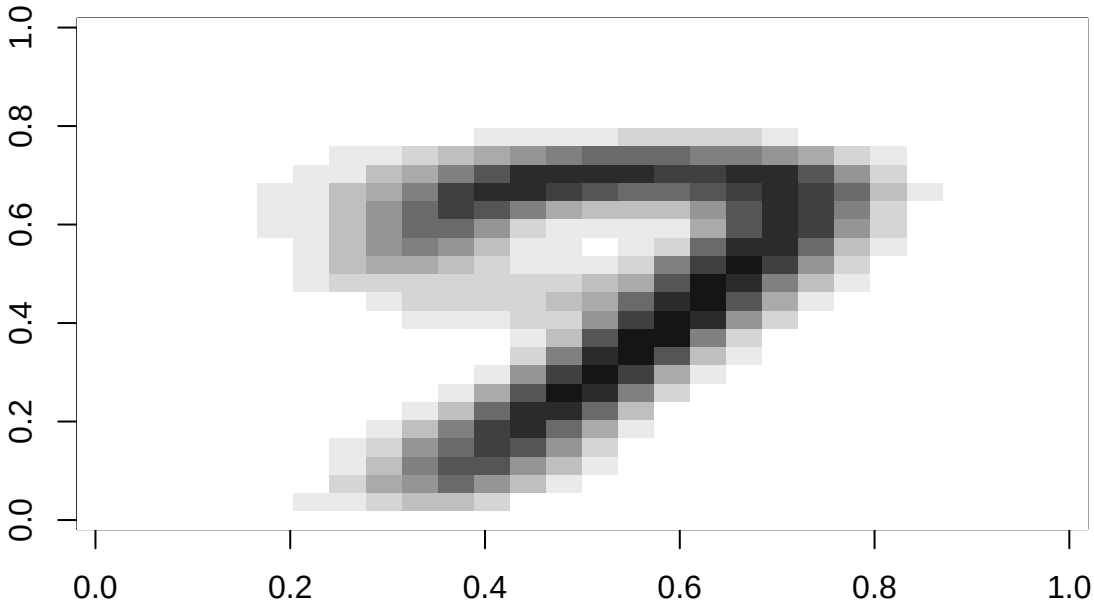

```
#k=10
k_10 <- my_kmeans(digits, 10,25)
k_10_center <- k_10[[2]]
k_10_loss <- k_10[[3]]
#
#show_digit(k_10_center)
for(i in 1:10){
  show_digit(k_10_center[i,])
}
```

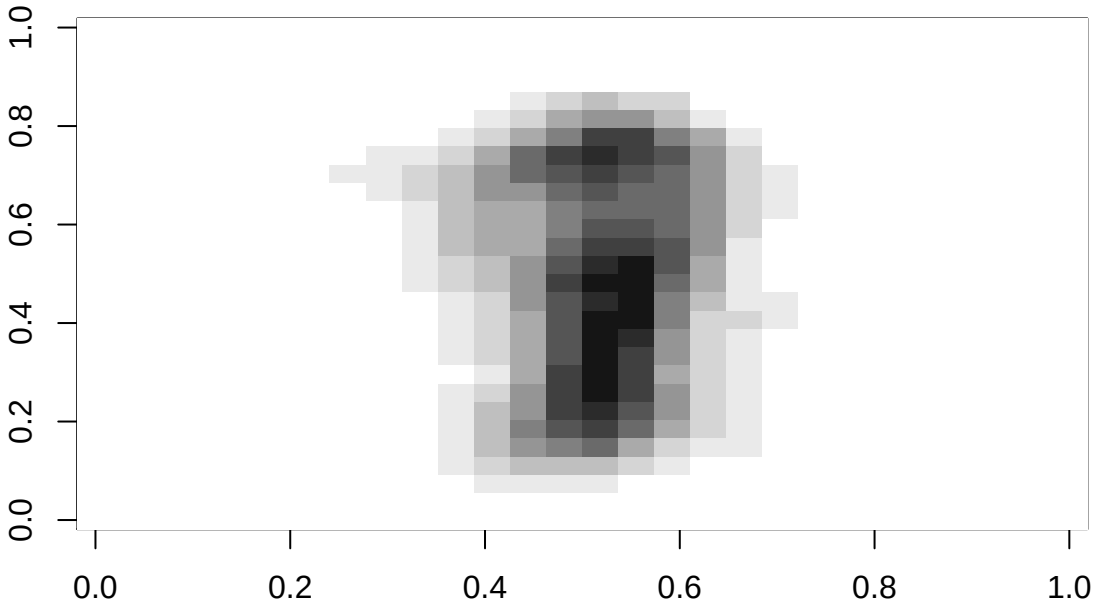

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

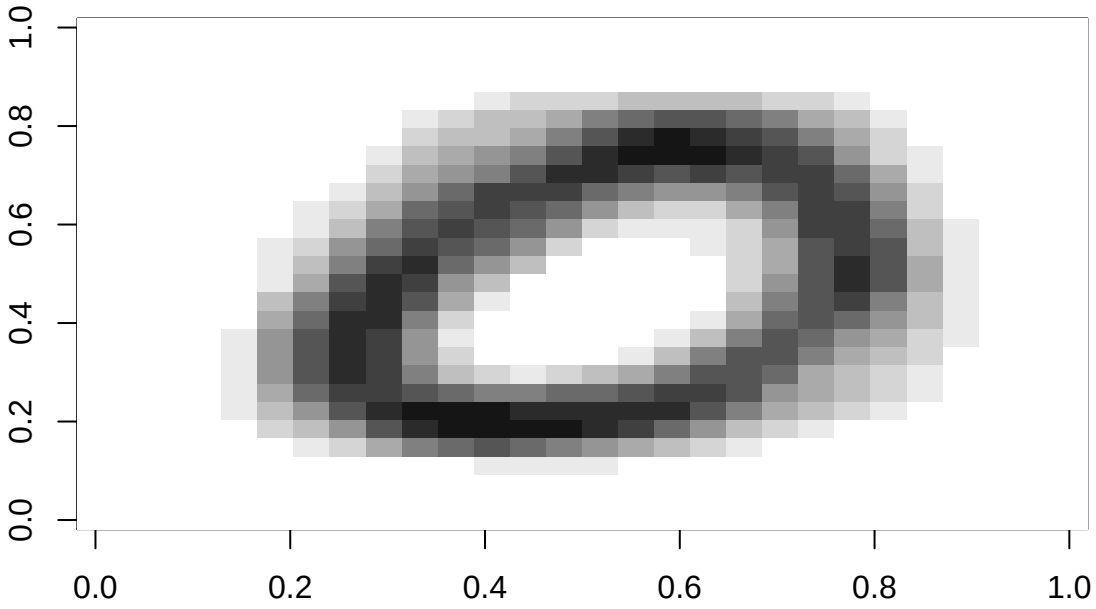

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

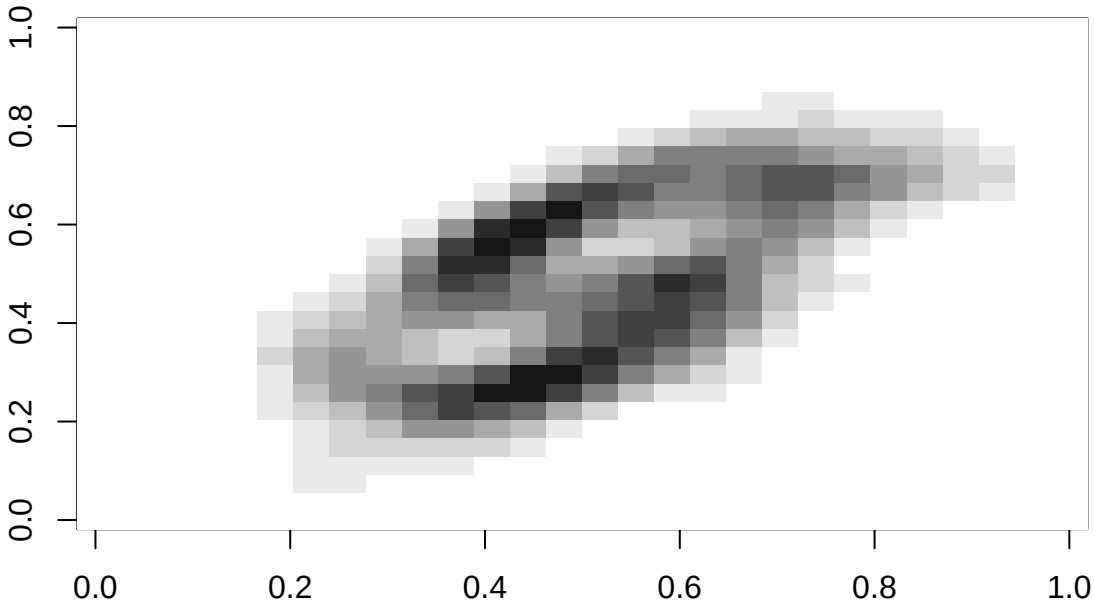

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

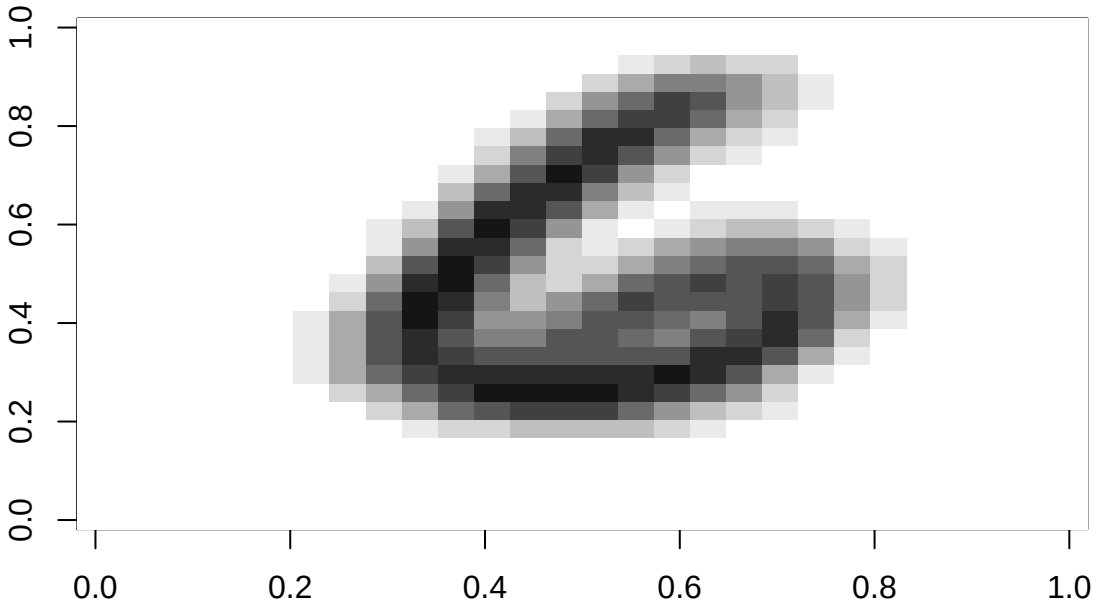

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

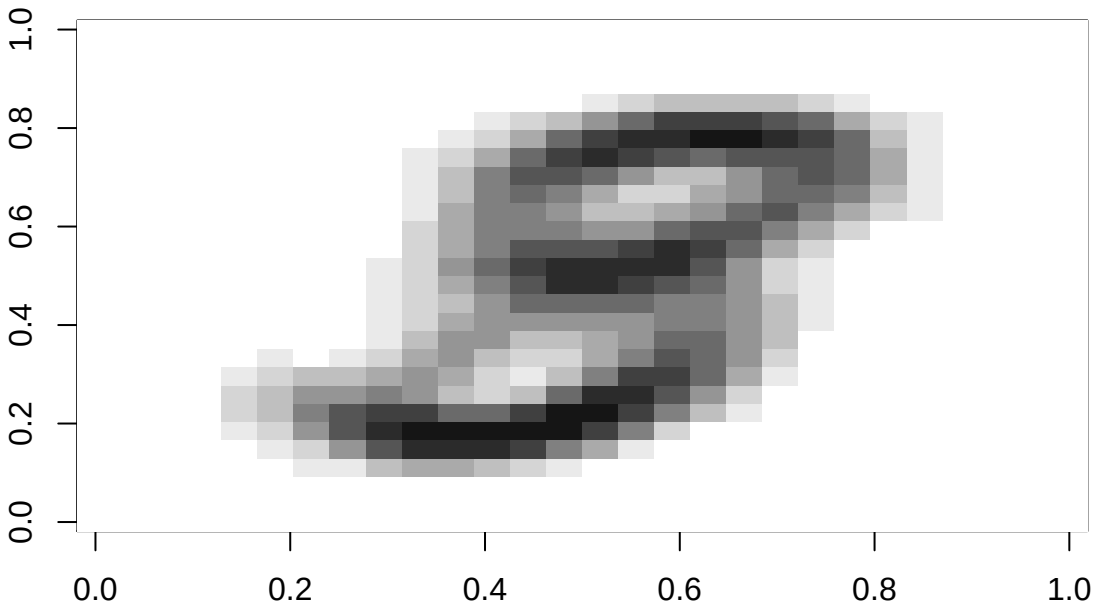

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

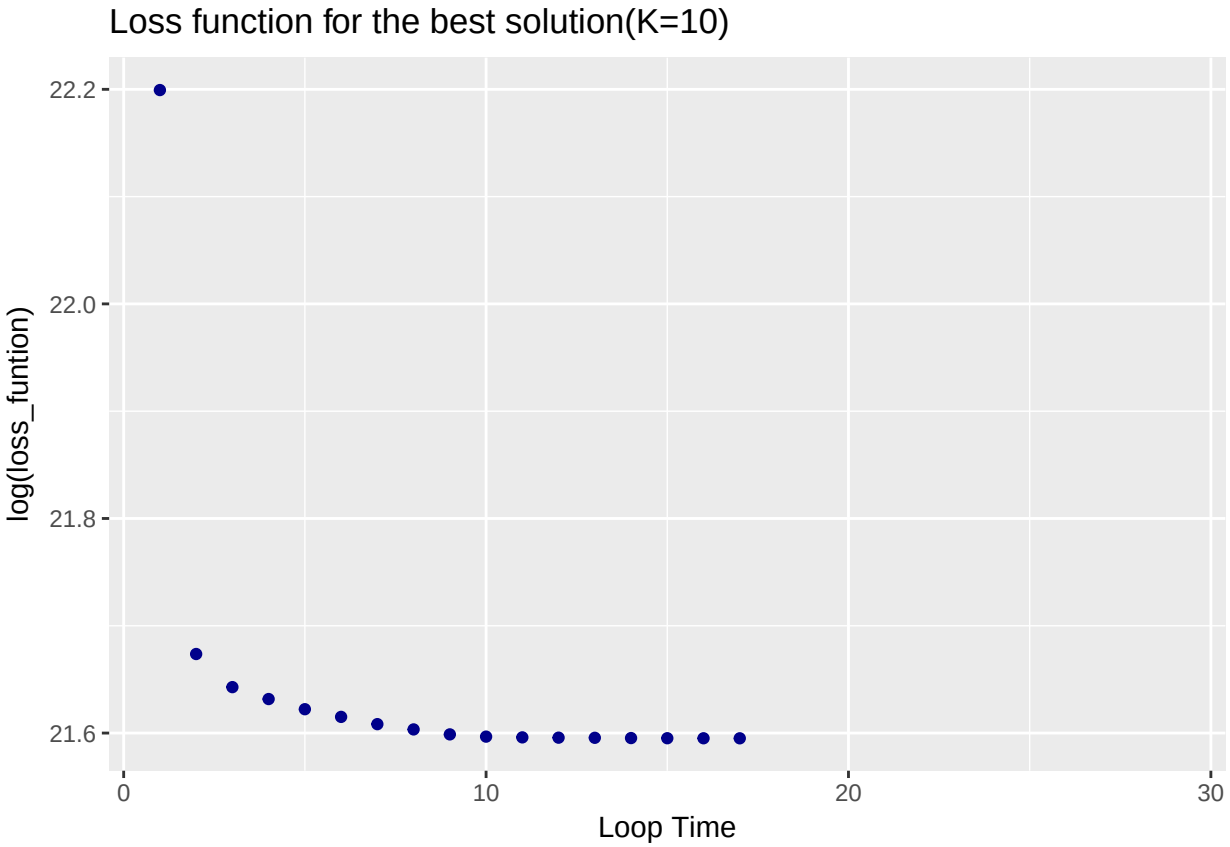

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

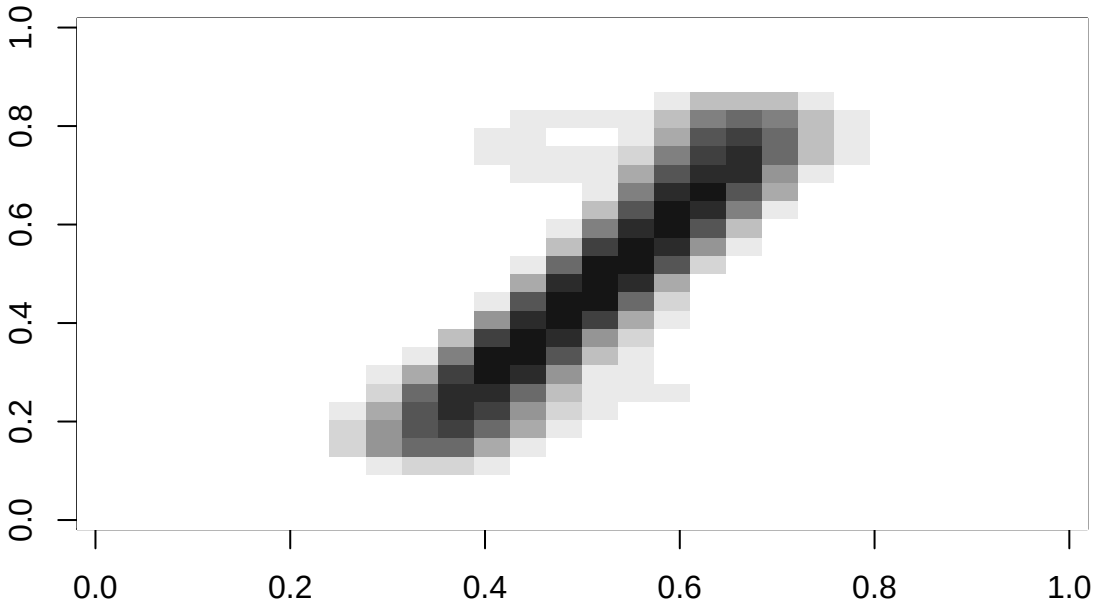


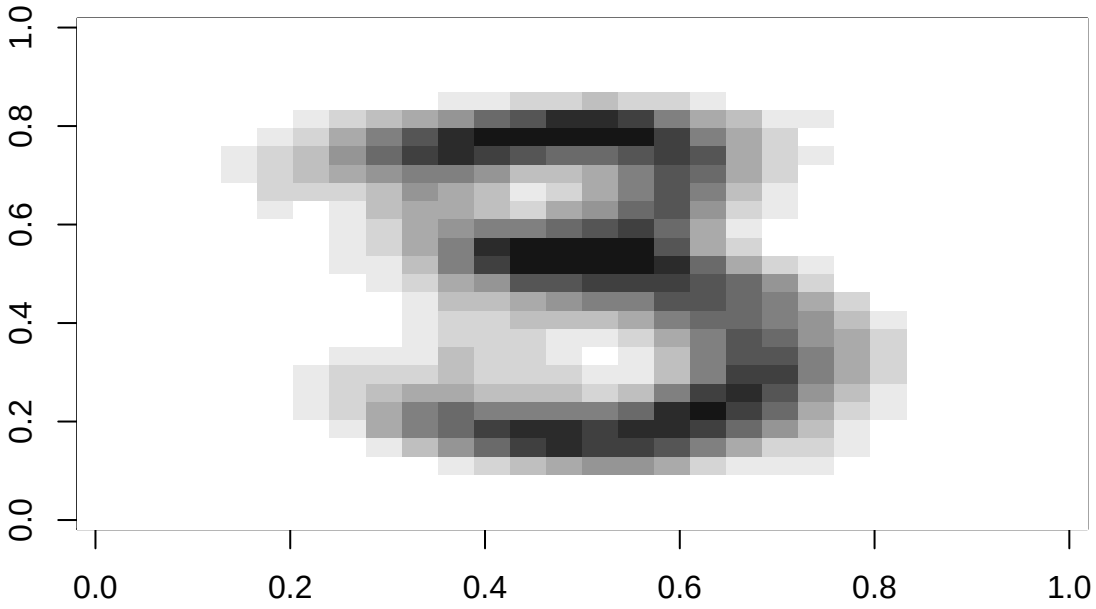

```
#find the best situation among N=25, then give the loss_function plot
k_10_loss_min <- apply(log(k_10_loss),1, min, na.rm = TRUE)
k_10_loss_best <- log(k_10_loss[which.min(k_10_loss_min), ])
length(k_10_loss_best) <- length(k_5_loss_best) #length =17 >>29

plot_10 <-ggplot(best_loss_plot, aes(xais, k_10_loss_best)) + geom_point(color = "darkblue") + labs(x="
plot_10
```

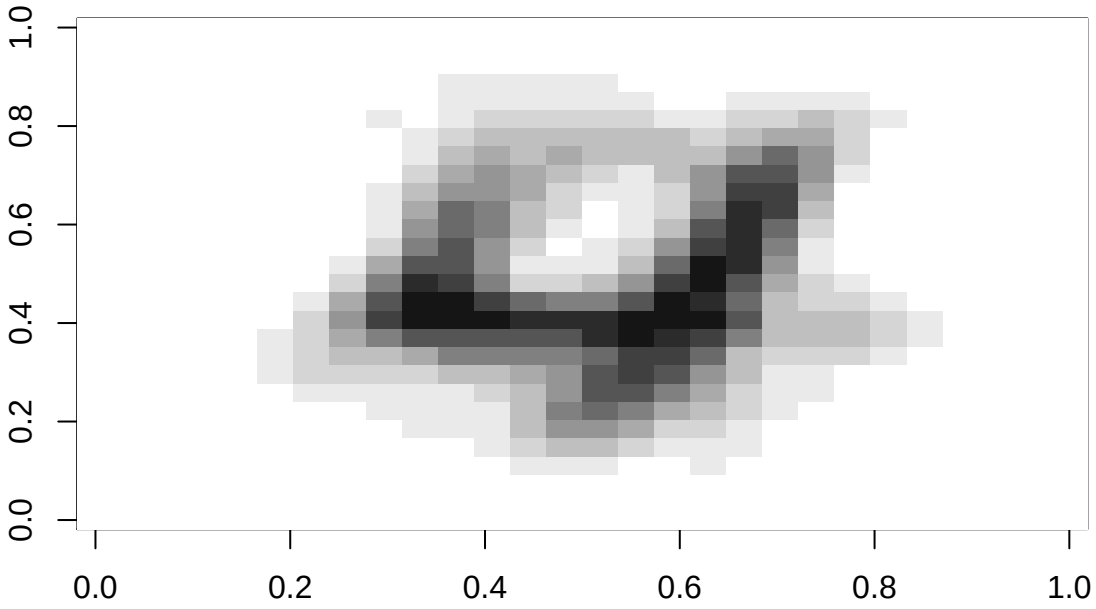
```
## Warning: Removed 12 rows containing missing values (geom_point).
```

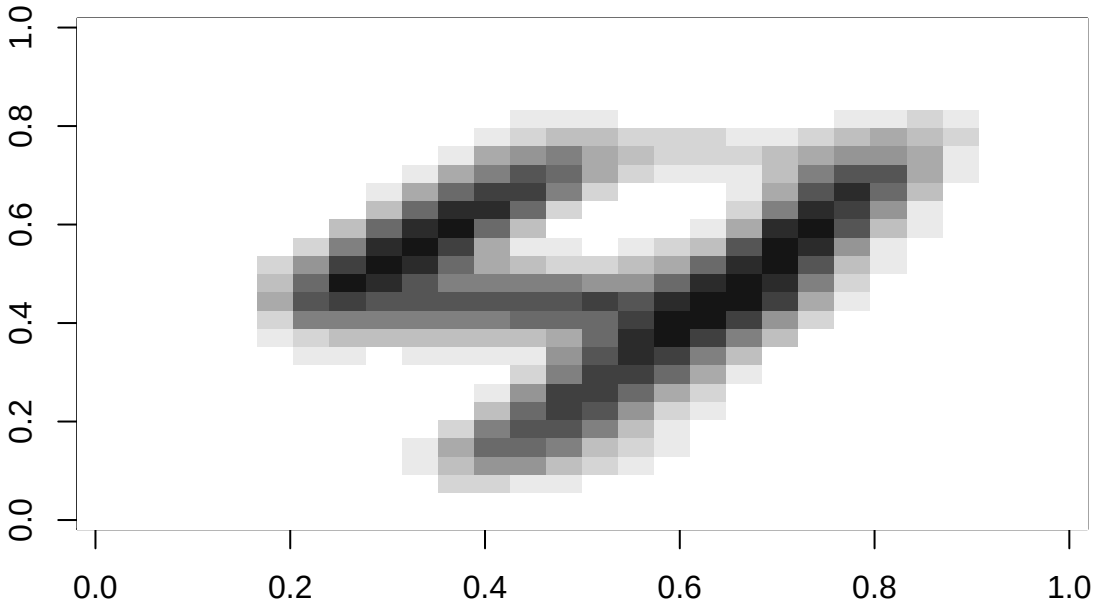


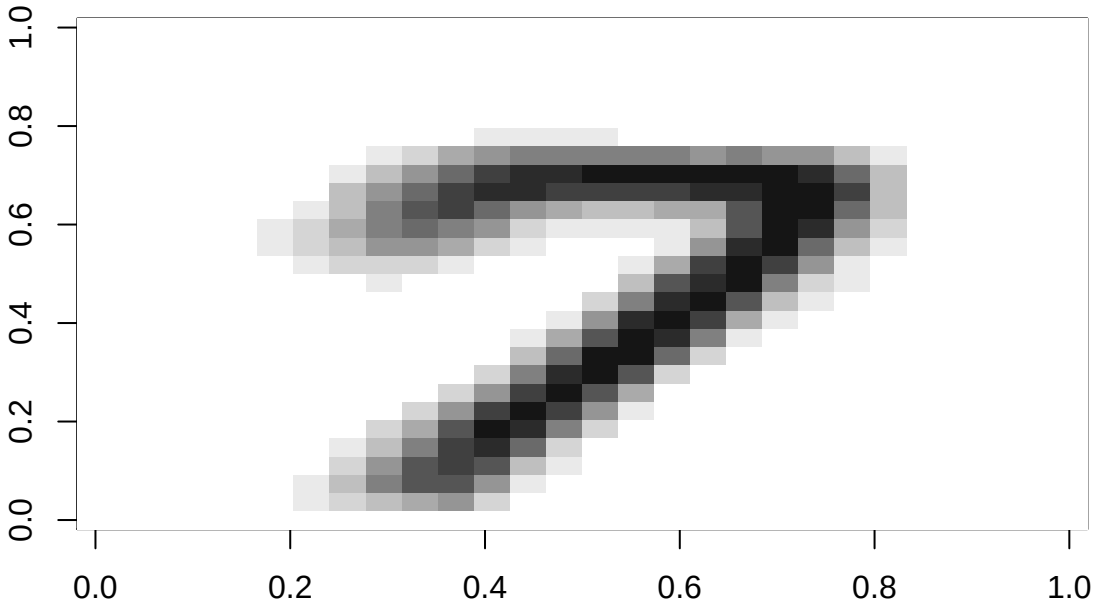

```
k_20 <- my_kmeans(digits, 20, 25)
k_20_center <- k_20[[2]]
k_20_loss <- k_20[[3]]
#show_digit(k_20_center)
for(i in 1:20){
  show_digit(k_20_center[i,])
}
```

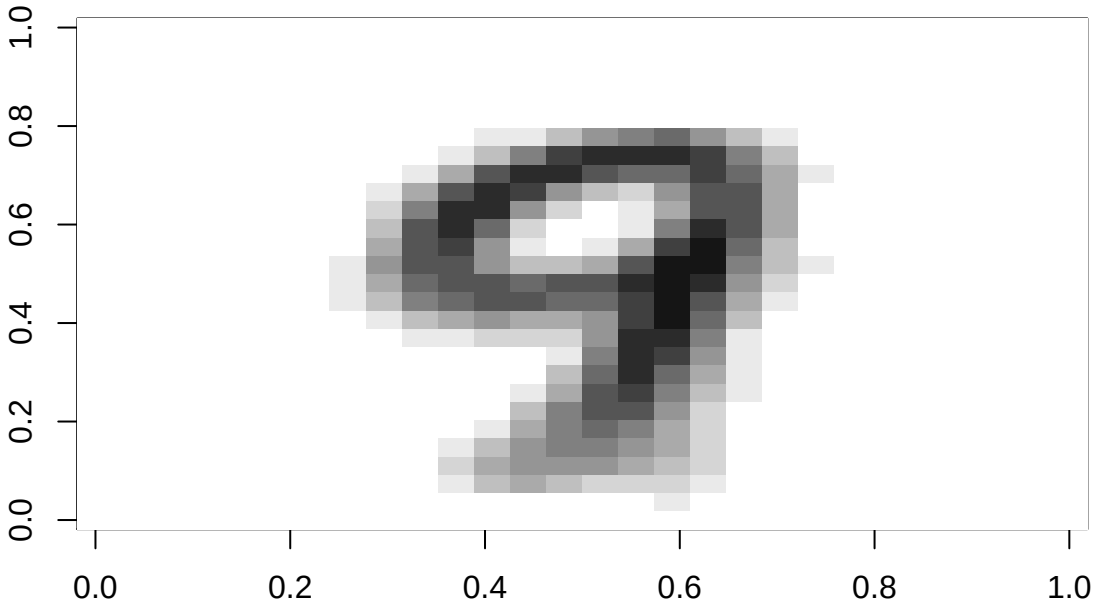

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

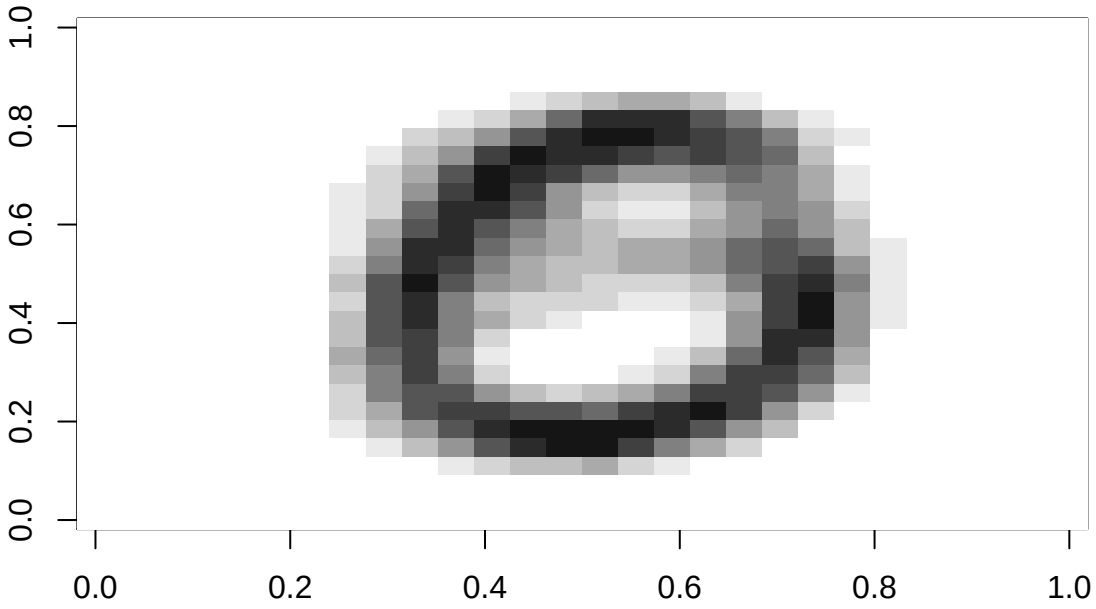

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

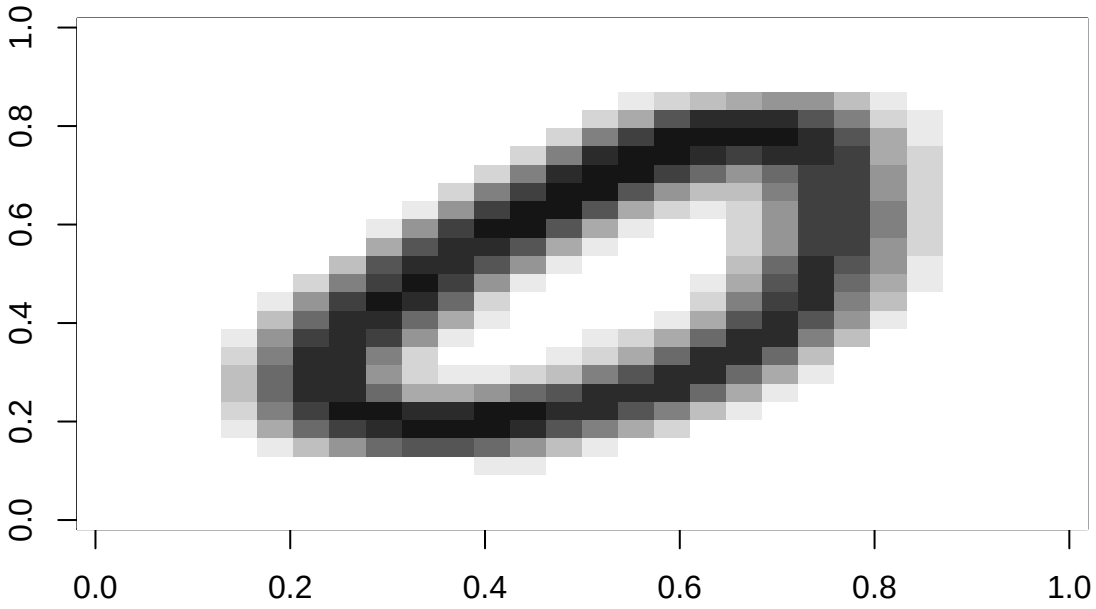

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

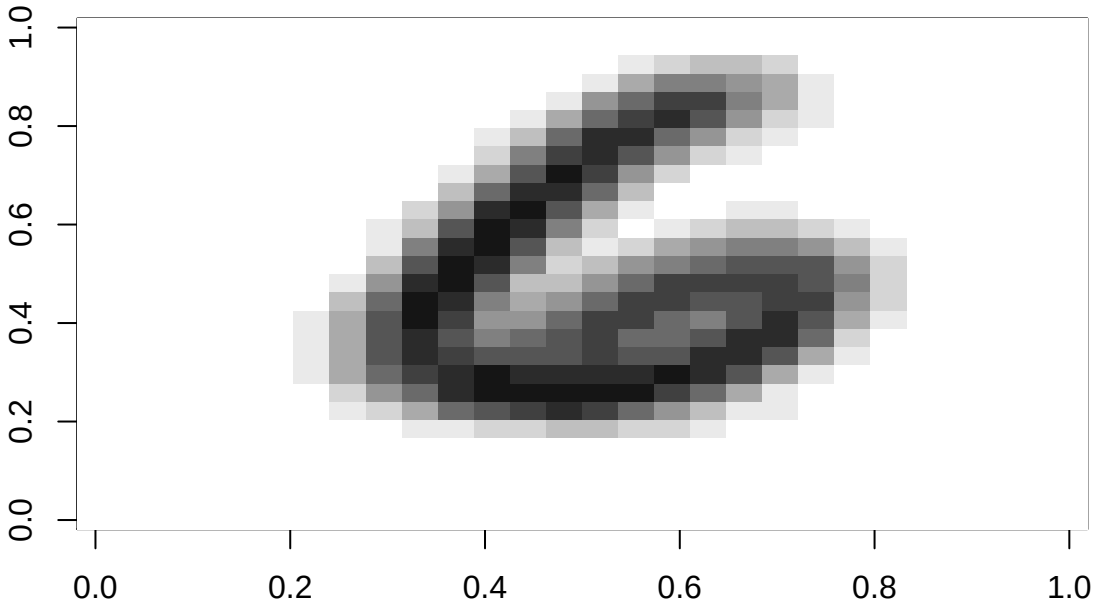

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

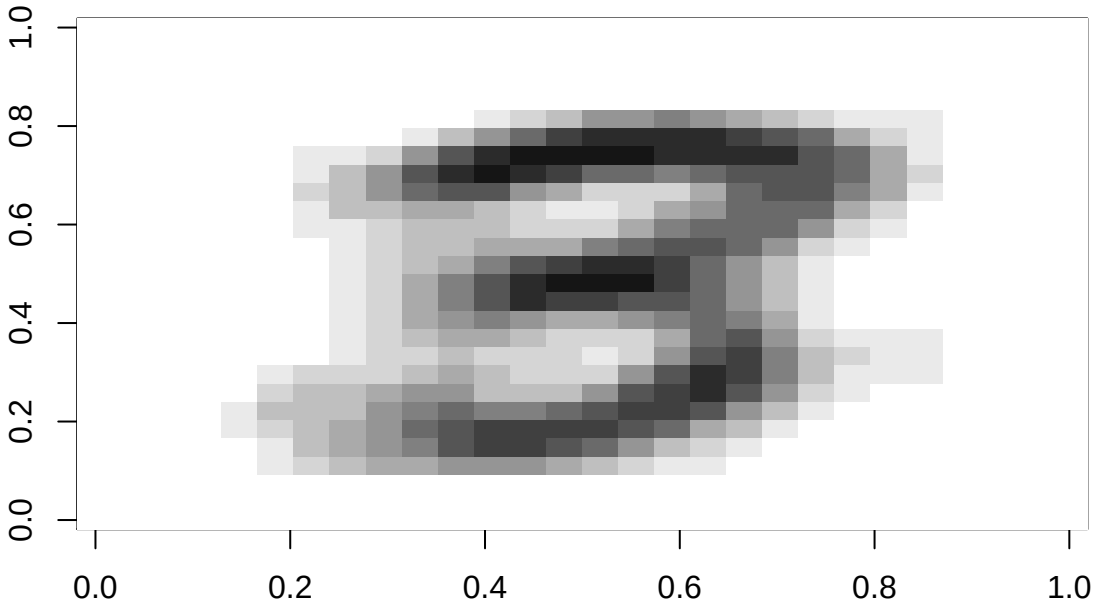

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

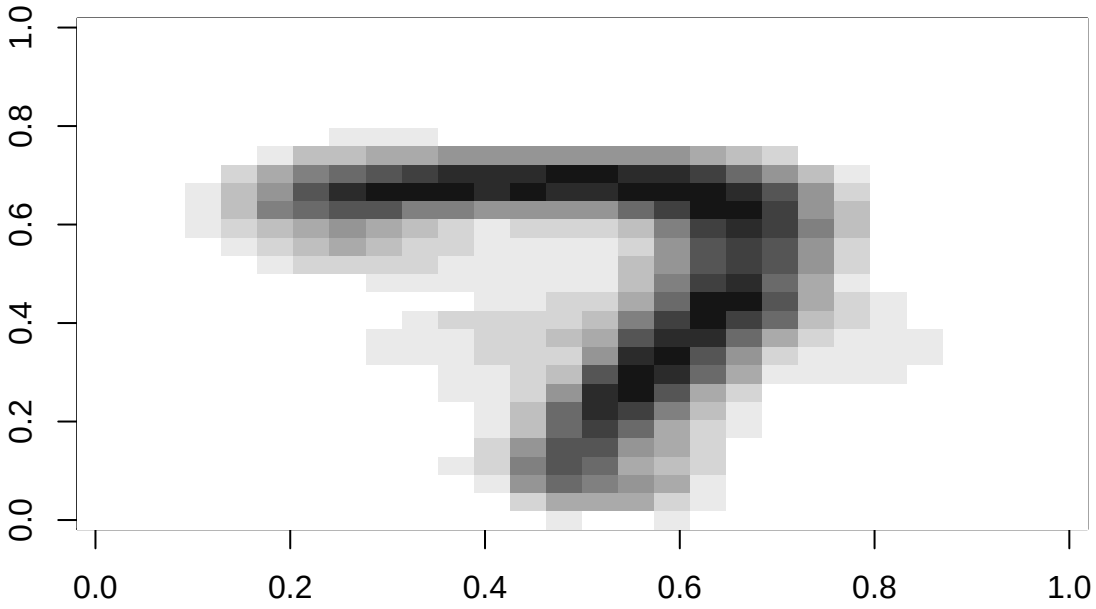

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

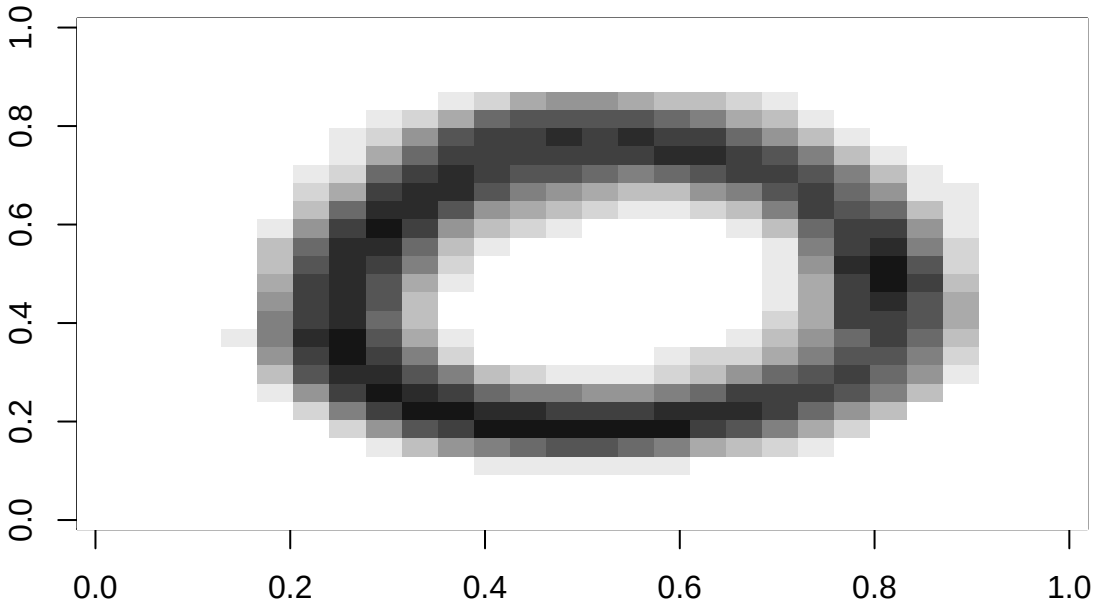

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

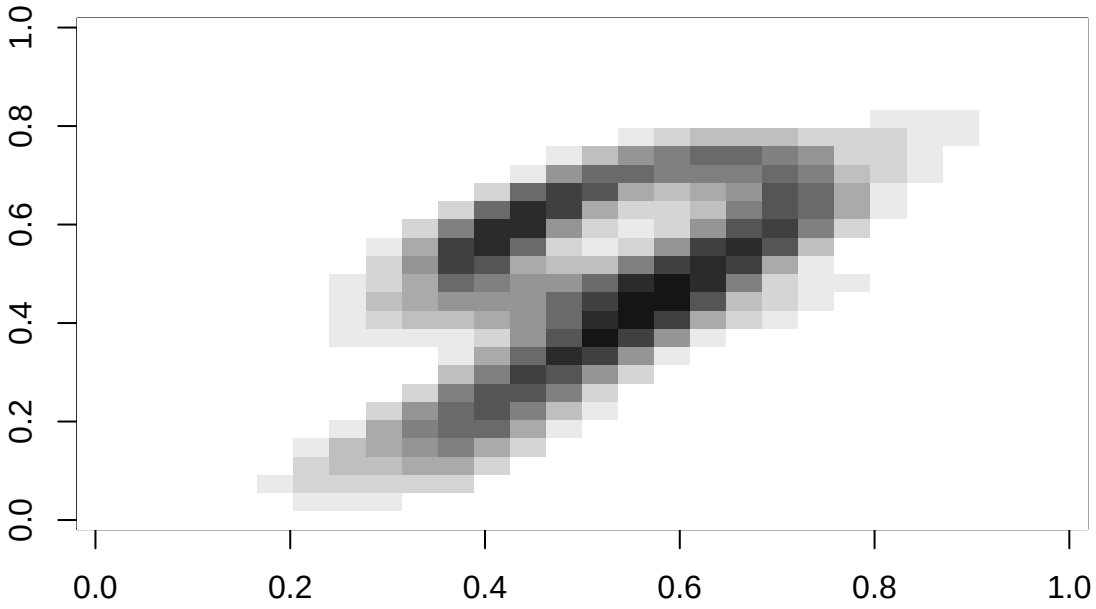

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

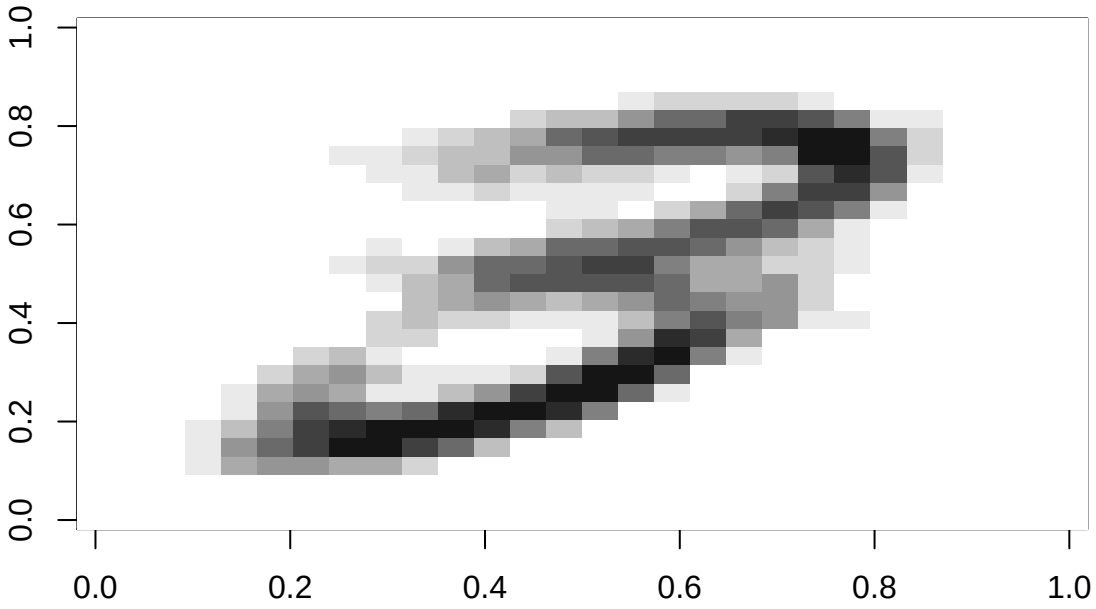

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

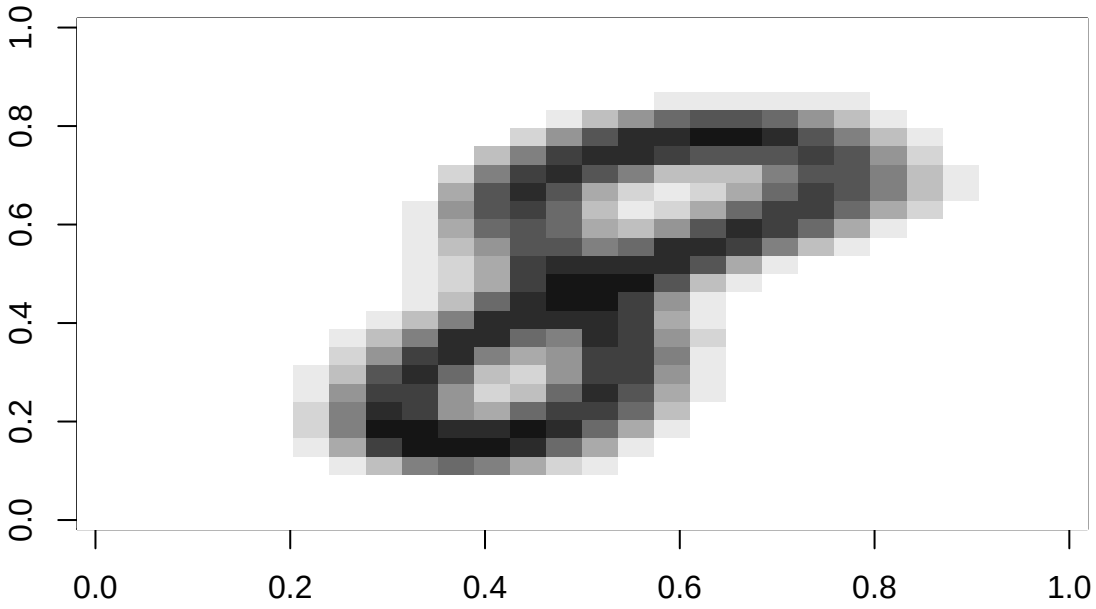

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

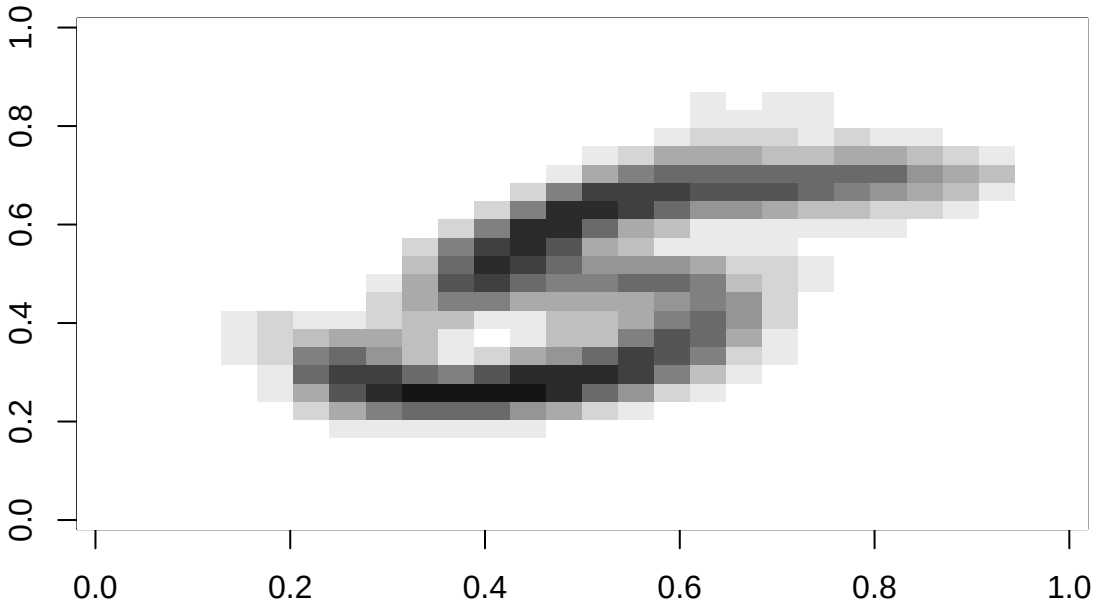

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

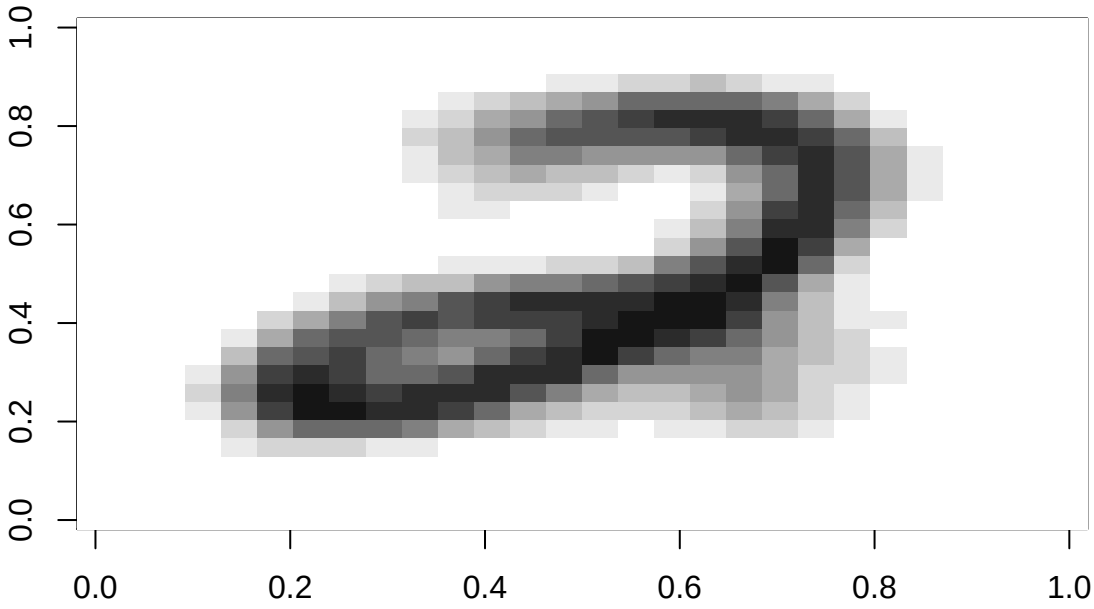

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

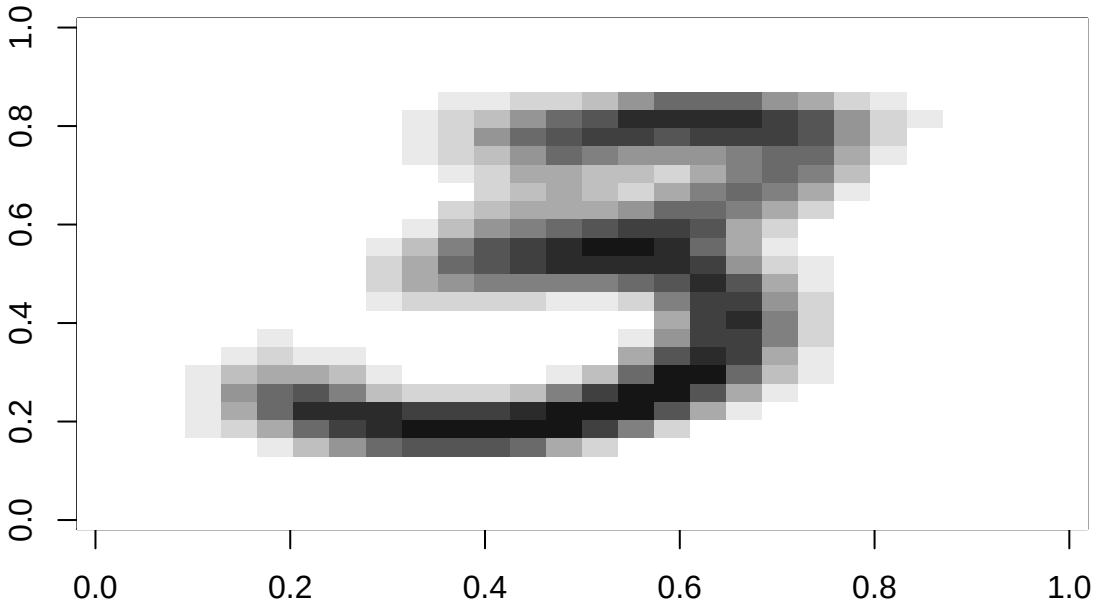

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

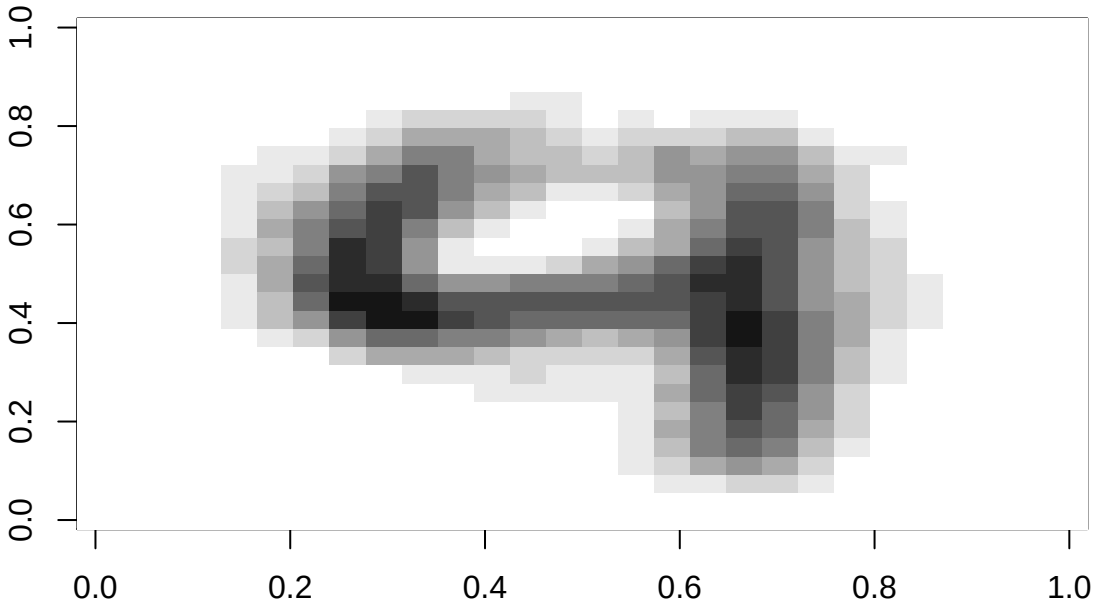

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

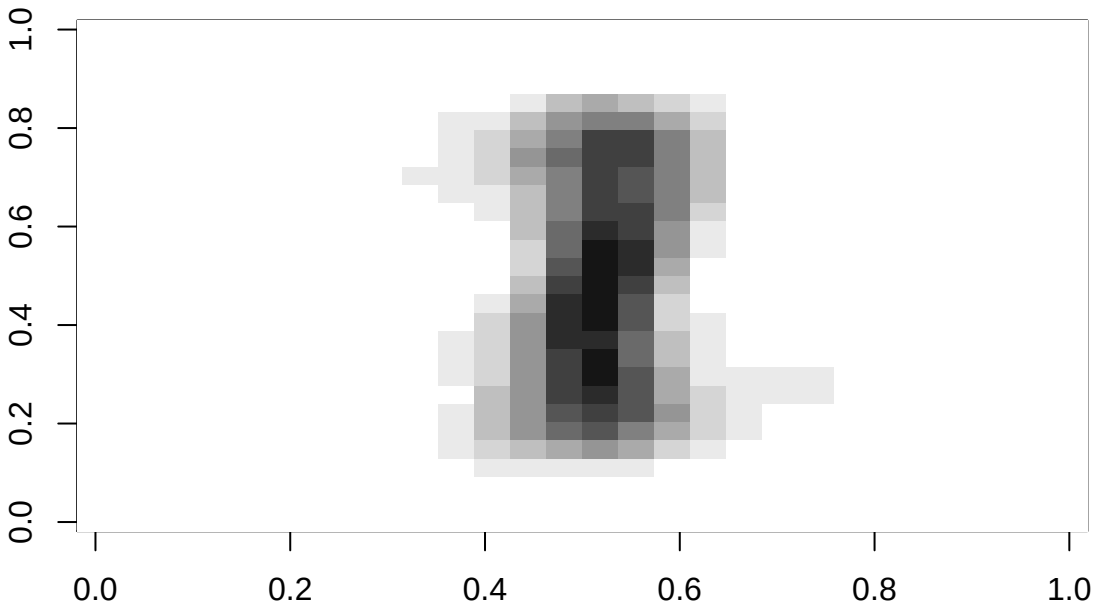

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

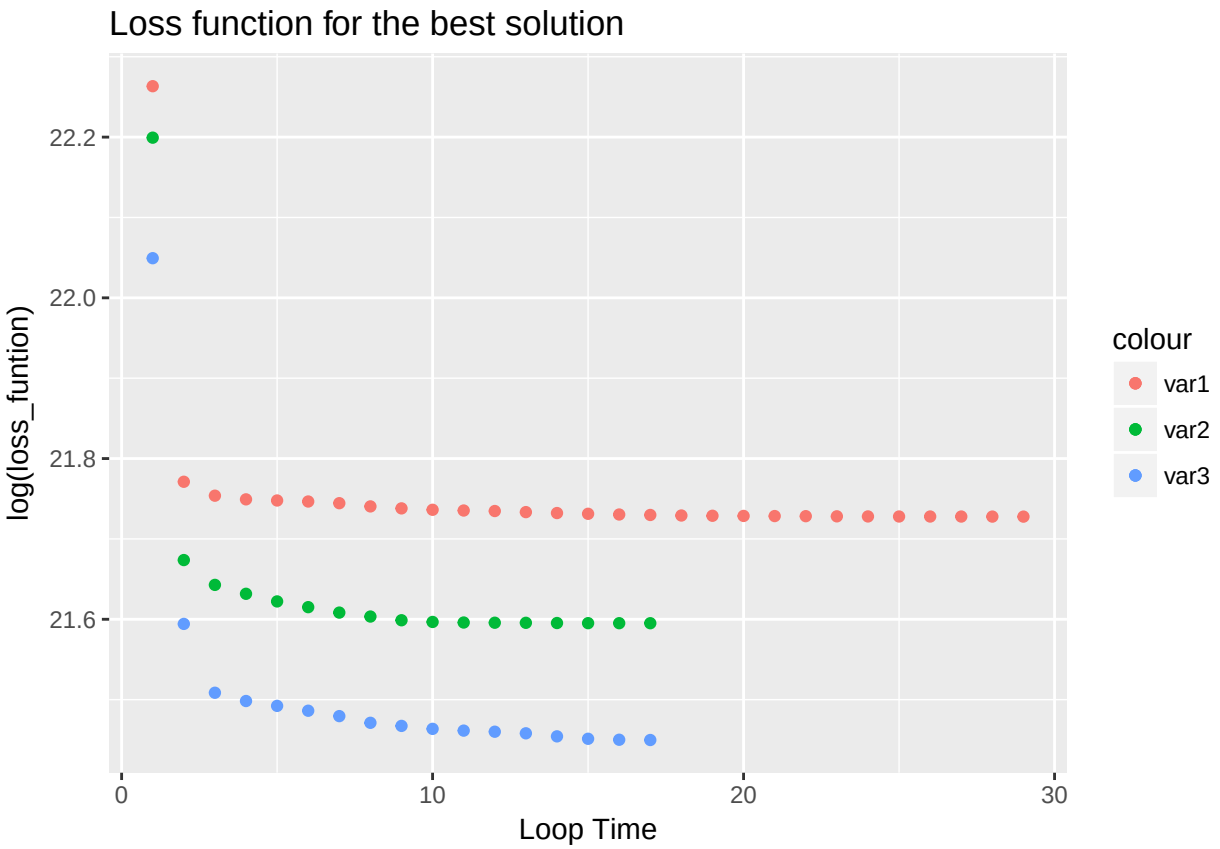



```
#find the best situation among N=25, then give the loss_function plot
k_20_loss_min <- apply(log(k_20_loss),1, min, na.rm = TRUE)
k_20_loss_best <- log(k_20_loss[which.min(k_20_loss_min), ])
length(k_20_loss_best) <- length(k_5_loss_best) #length =17 >>29
plot_20 <-ggplot(best_loss_plot, aes(xais, k_20_loss_best)) + geom_point(color = "darkblue") + labs(x="
```

```
x <- 1:29
total_loss_plot <- data.frame(x, var1= k_5_loss_best,
                              var2 = k_10_loss_best,
                              var3 = k_20_loss_best)
g <-ggplot(total_loss_plot, aes(x=x)) + geom_point(aes(y=var1, color="var1")) +
             geom_point(aes(y=var2, color="var2"))+
             geom_point(aes(y=var3, color="var3")) +
             labs(x="Loop Time", y="log(loss_funtion)", title="Loss function for the best solution")
g
```

```
## Warning: Removed 12 rows containing missing values (geom_point).
```

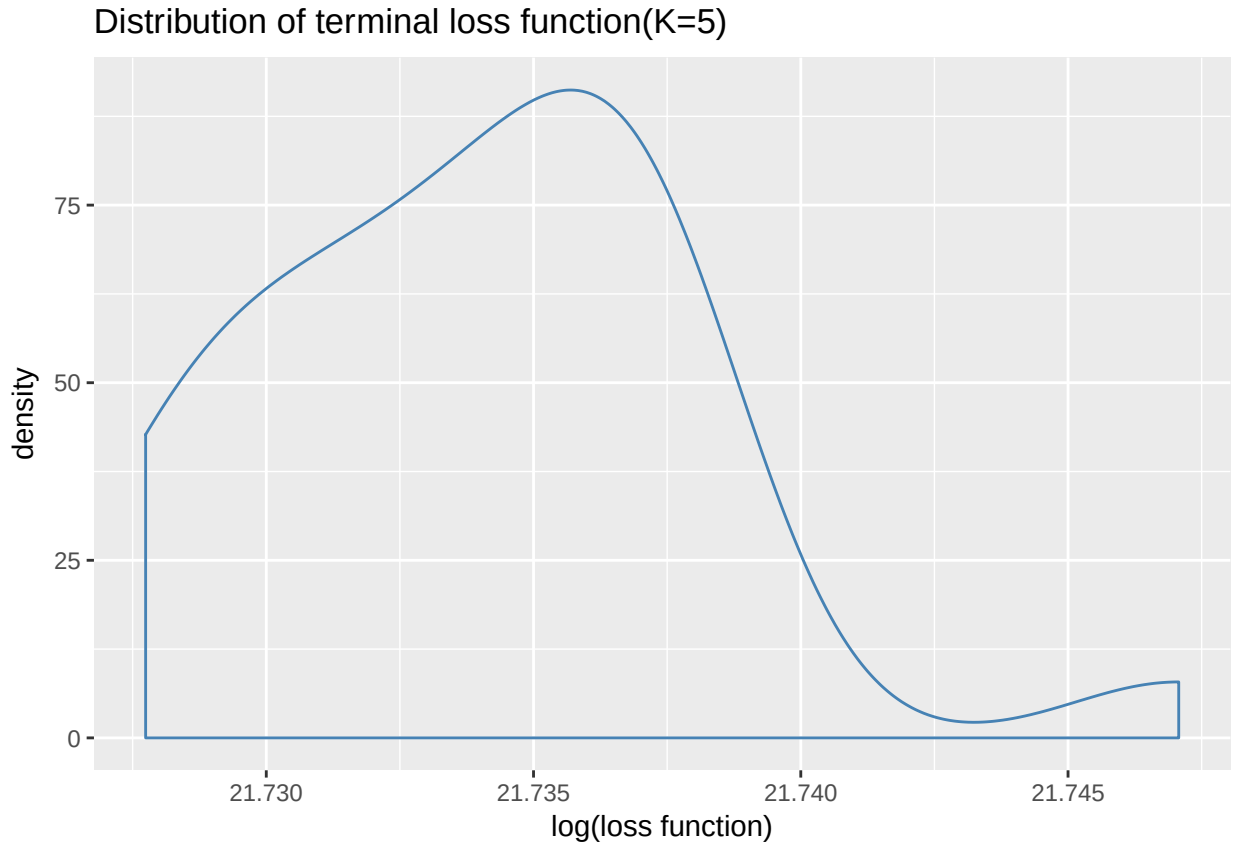
```
## Warning: Removed 12 rows containing missing values (geom_point).
```



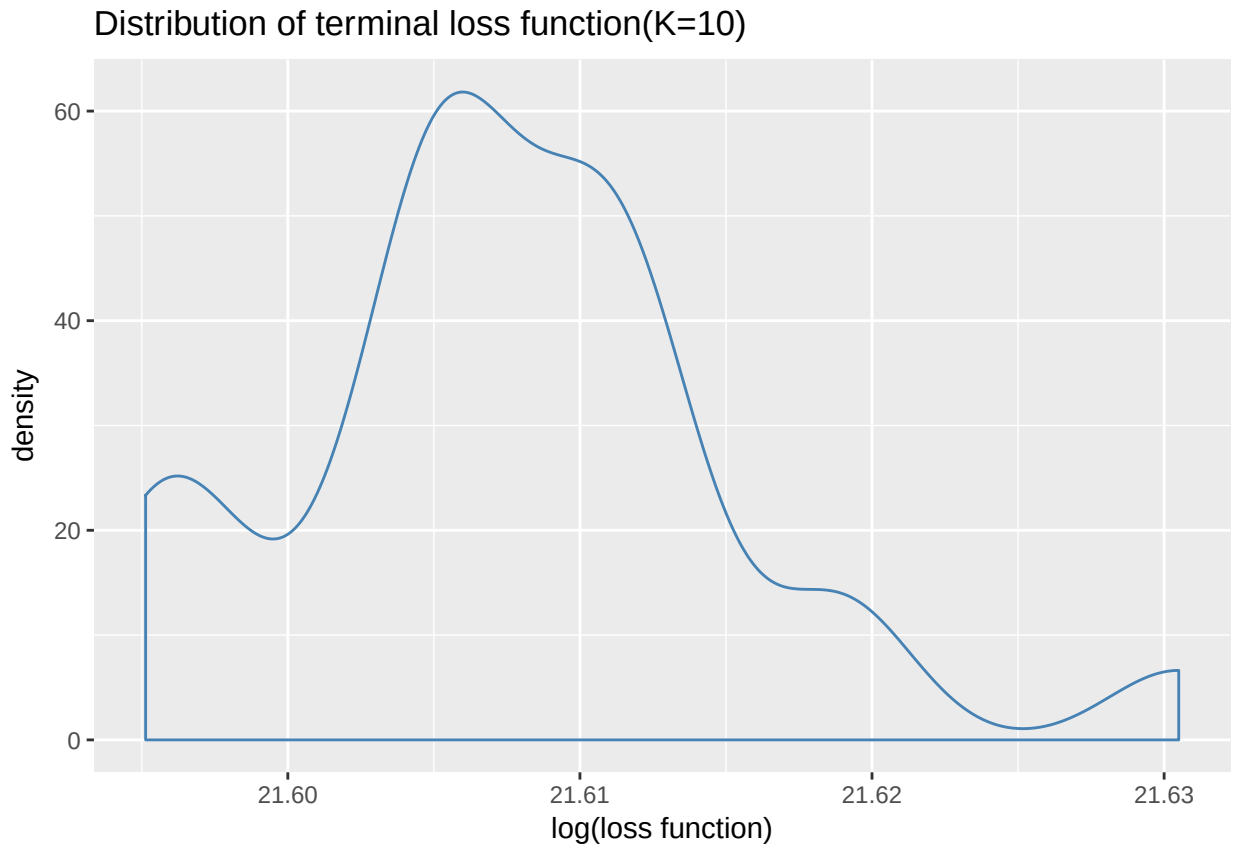
In this plot, var1 is k=5, var2 is K=10, var3 is K=20.

5. For each setting of K, plot the distribution of terminal loss function values.

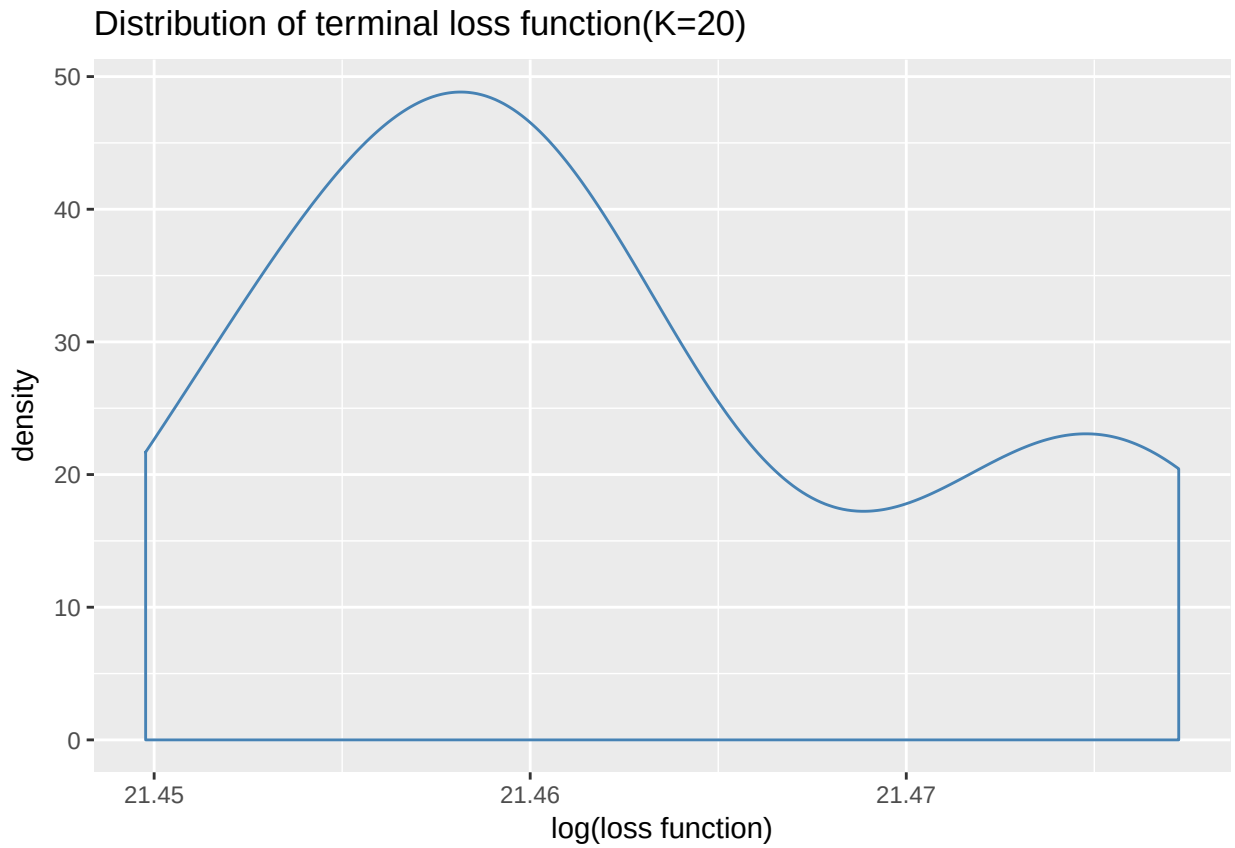
```
#find the min value of loss function for each initialization
#k=5
xnum <- 1:25
loss_plot_5 <- data.frame(xnum, k_5_loss_min)
p <- ggplot(data=loss_plot_5, aes(x=k_5_loss_min))
p + geom_density(color="steelblue") + labs(x="log(loss function)", y="density", title="Distribution of
```




```
#k=10
loss_plot_10 <- data.frame(xnum, k_10_loss_min)
p <- ggplot(data=loss_plot_10, aes(x=k_10_loss_min))
p + geom_density(color="steelblue") + labs(x="log(loss function)", y="density", title="Distribution of
```




```
#k=20
loss_plot <- data.frame(xnum, k_20_loss_min)
p <- ggplot(data=loss_plot, aes(x=k_20_loss_min))
p + geom_density(color="steelblue") + labs(x="log(loss function)", y="density", title="Distribution of
```



####6.Explain briefly how to choose k Ideally, we will have a priori knowledge about the true grouping, which can help us choose the number of k. In that case, we know there are handwriting 10 numbers, so we might try different k aroung 10. But if we know nothing about the observations that need to be clustered, we may first try sqrt(n/2) = k, in this case, k is about 25. Or, we check the homogeneity within the clusters changes for various values of k.

7.

```
digits2 <- train$x[1:100,]
labels2 <- train$y[1:100]
dim(digits2)#100*784
```

```
## [1] 100 784
```

```
length(labels2)#100
```

```
## [1] 100
```

```
my_kmedoids <- function(digits2, k, N){
  overall_loss_matrix <- matrix(NA, ncol=100, nrow = N)

  for(i in 1:N){
    #assign initial variables
    cluster_center <- matrix(rep(0, k*784), ncol = 784, nrow = k)
    set.seed(i)
    #randomly select k images as initial clusters
    initial_index <- sample(1:100, size = k)
    cluster_center <- digits2[initial_index, ]
```

```
    e_distance <- matrix(rep(0, k*100), ncol = 100, nrow = k)
    zero_r <- matrix(rep(0, k*100), ncol = 100, nrow = k)
    loss_value<- rep(NA,100)#every element is lossfunction of cluster j

    #set stop criteria for while loop
    old_min_index_xc <- rep(-Inf, 1*100)
    new_min_index_xc <- rep(0, 1*100)
    loop <- 0 #while loop time
    while(!all(old_min_index_xc==new_min_index_xc)){
      new_r <- zero_r
      old_min_index_xc <- new_min_index_xc
      #reset new_r to 0, otherwise, it will be overwritten every loop
      loop <- loop + 1
      #form distance matrix
      for(j in 1:k){
         e_distance[j,] <- rowSums((t(t(digits2)-cluster_center[j,]))^2)
      }

      new_min_index_xc <- apply(e_distance, 2, which.min)
      new_min_value_xc <- apply(e_distance, 2, min)
      for(j in 1:k){
        #update new cluster assignment r
        new_r[j,(new_min_value_xc %in% e_distance[j,])] <- 1
        index_in_j <- which(new_r[j, ]==1)
        position_in_j <- digits2[index_in_j, ]
        #update new cluster center
        cluster_center[j, ] <- get_prototype(position_in_j)
      }
      loss_value[loop] <- sum(new_min_value_xc)
    }
    overall_loss_matrix[i,] <- loss_value
  }
  result <- list(new_r, cluster_center, overall_loss_matrix)
  return(result)
}
#test
#my_kmeans(digits2, 5, 5)
```

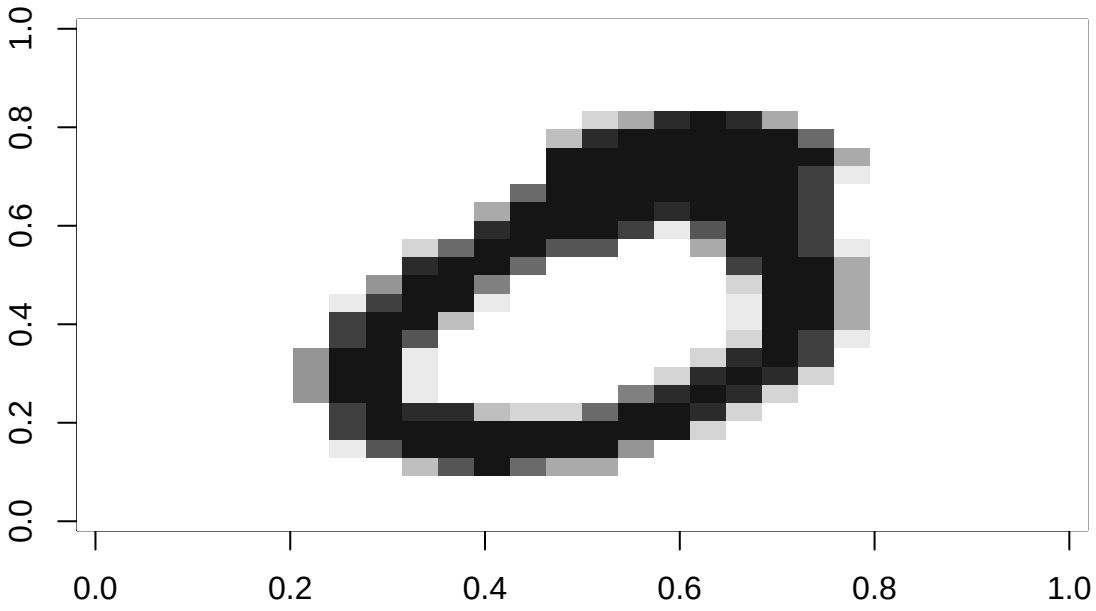
Bonus Problem

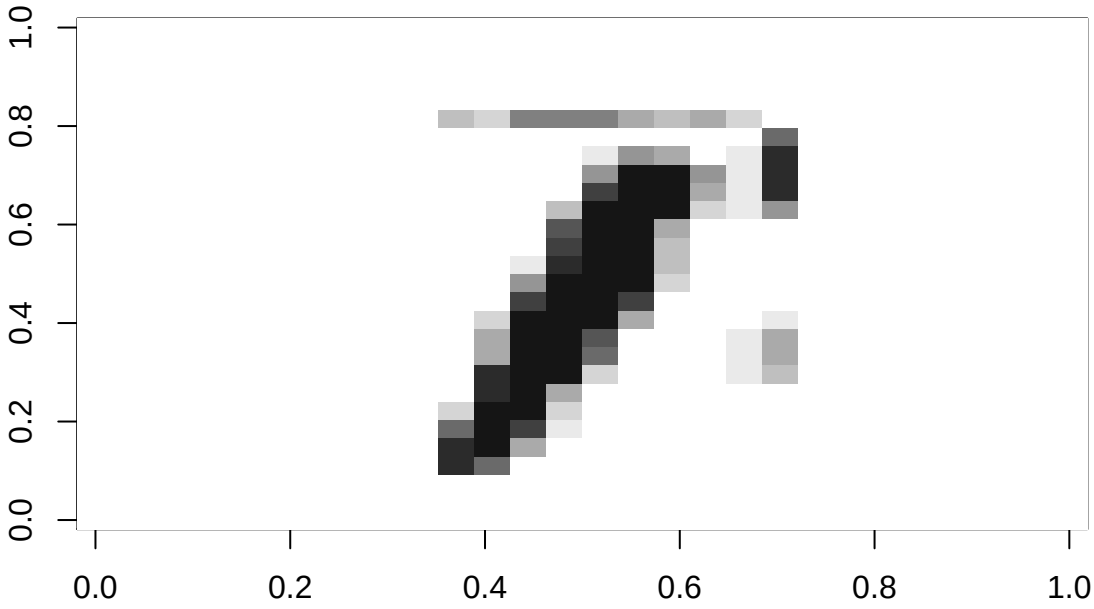
```
#digits_test <- digits[1:20, 1:10]
#set.seed(2)
#r <- sample(c(0,1), 200, replace=TRUE)
#m <- sample(1:10,5)
#x_in_j <- digits[m,]
#n <- dim(x_in_j)[1]
get_prototype <- function(x_in_j){
  #get e_distance between xu and other x
  distance <- as.matrix(dist(x_in_j))#dim=(n,n)
  distance_uv <- colSums(distance)#vector of distance: sum(d(xi to others))
  min_distance <- which.min(distance_uv)#get the index of medoid
  new_center <- x_in_j[min_distance, ]
```

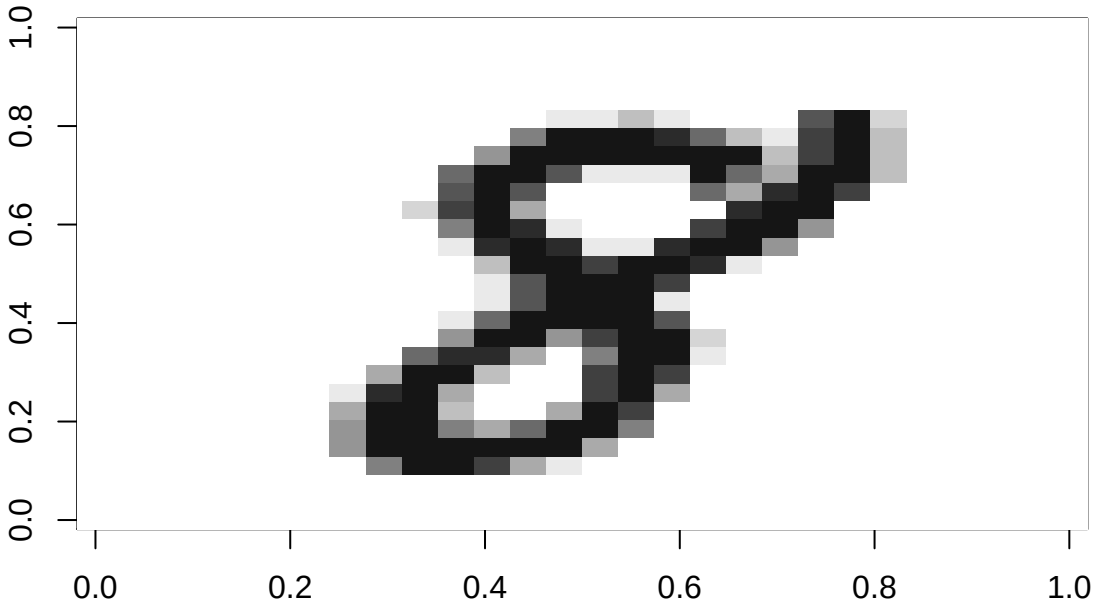
```
  return(new_center)
}
#get_prototype(x_in_j)
```

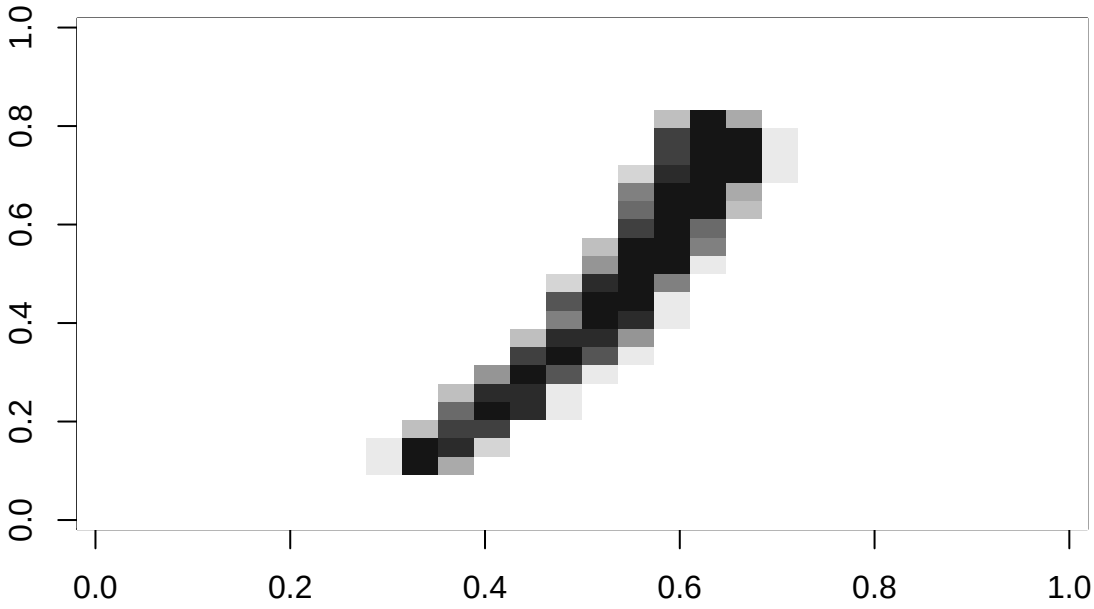
8.

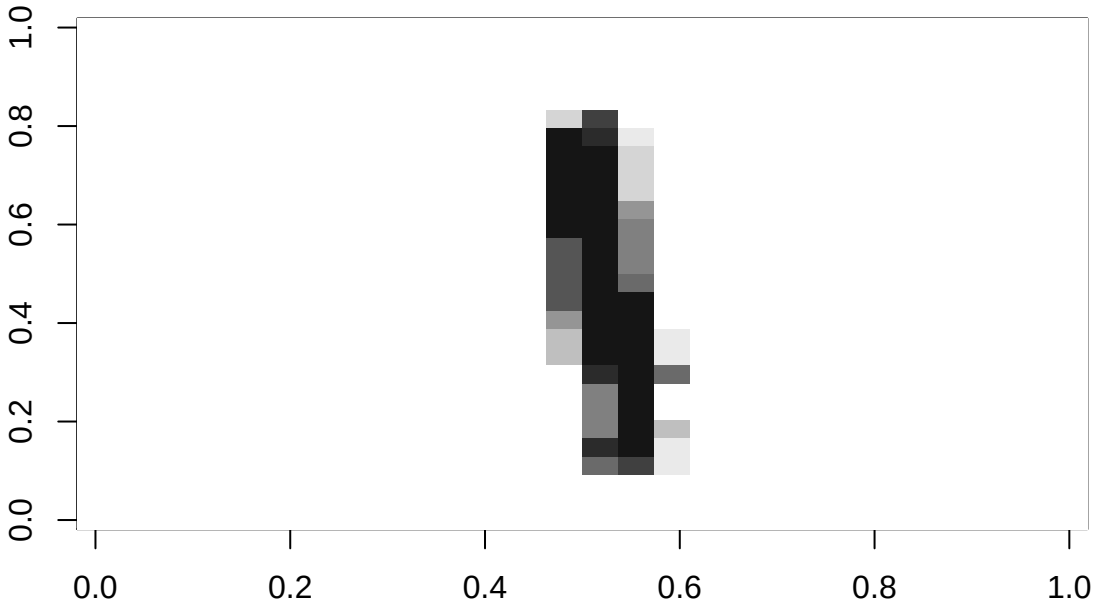
```
km_5 <- my_kmedoids(digits2, 5, 10)
km_5_center <- km_5[[2]]
#show_digit(k_5_center)
for(i in 1:5){
  show_digit(km_5_center[i,])
}
```



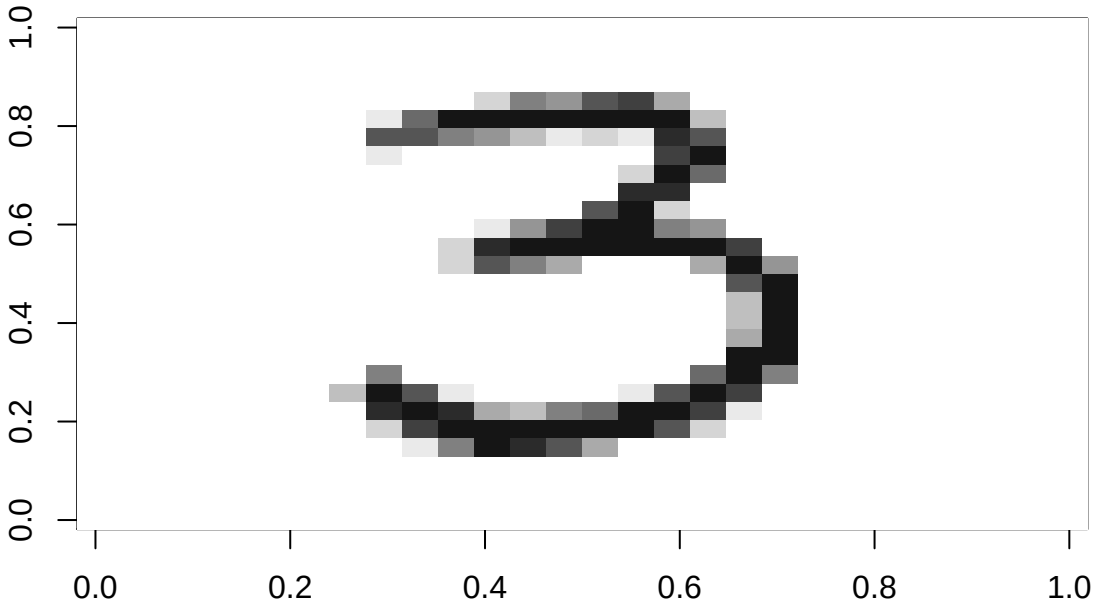

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

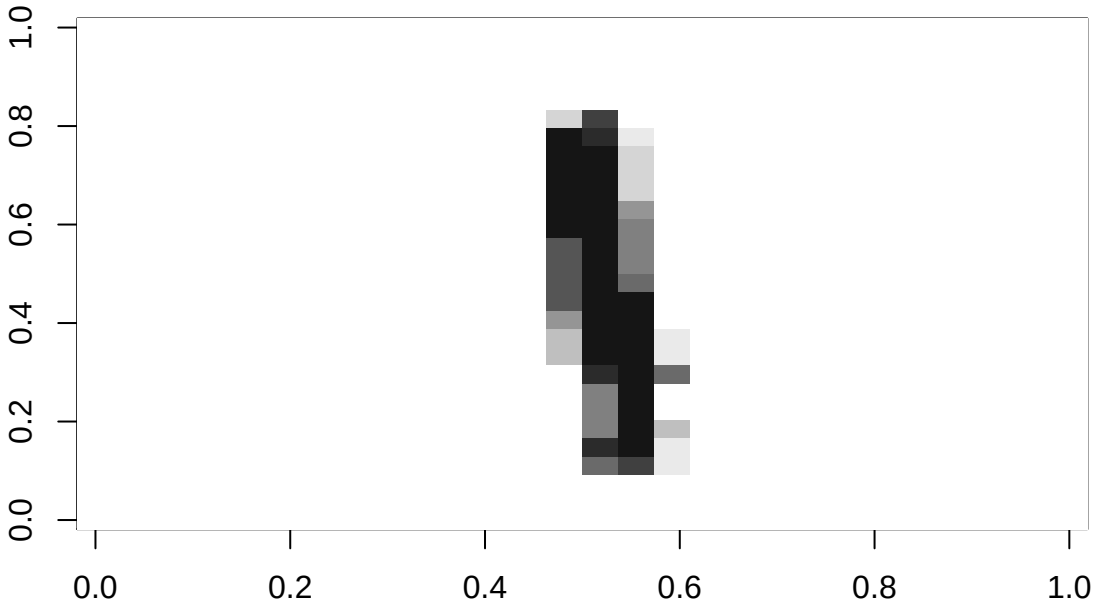

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

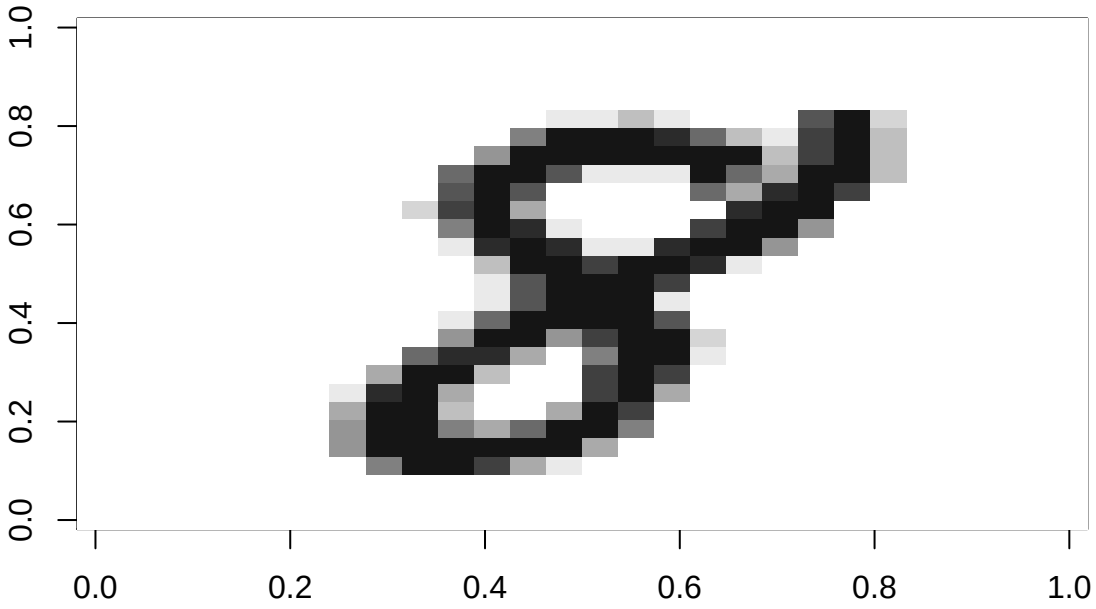

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

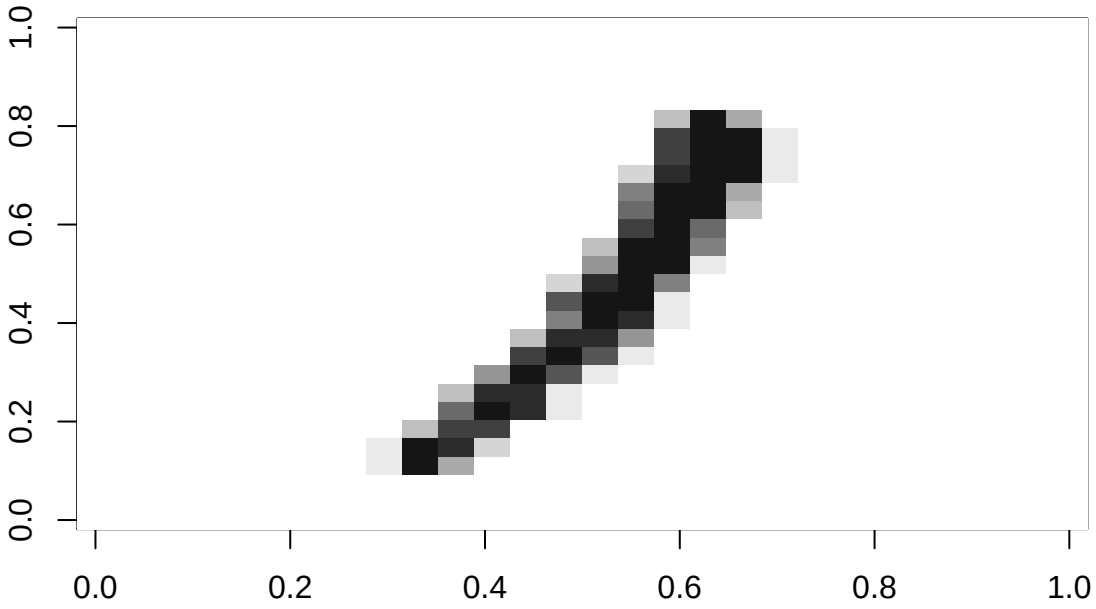


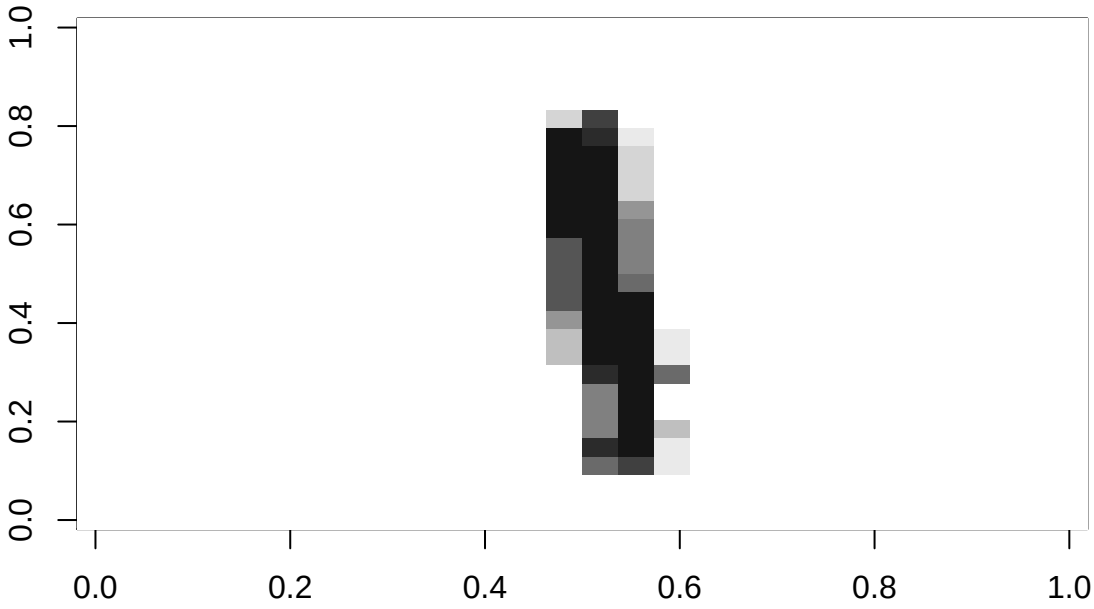

```
km_10 <- my_kmedoids(digits2, 10, 10)
km_10_center <- km_10[[2]]
#show_digit(k_5_center)
for(i in 1:10){
  show_digit(km_10_center[i,])
}
```

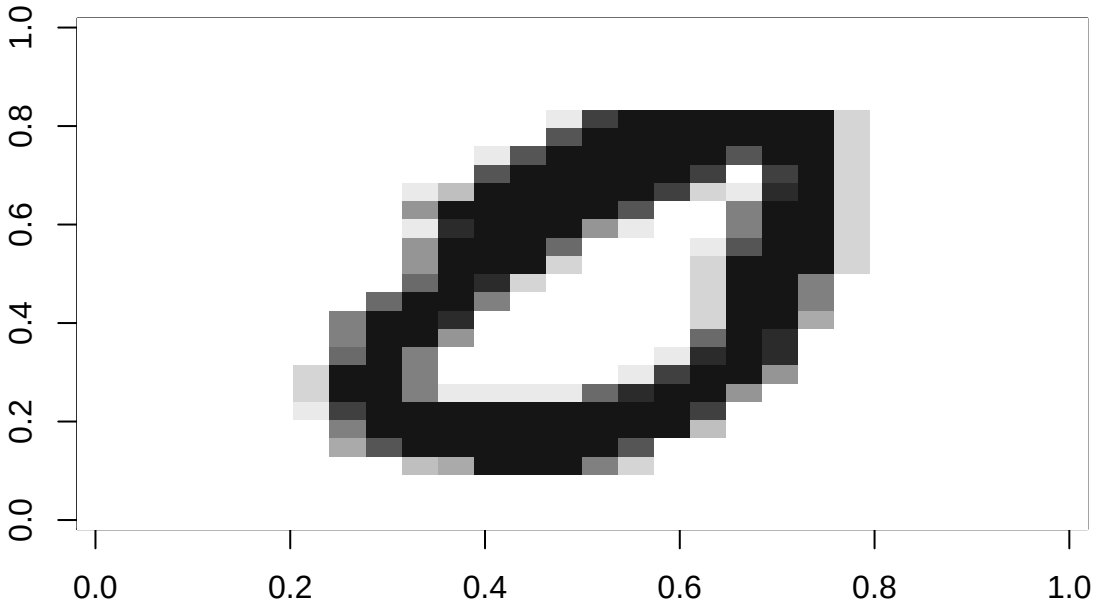

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

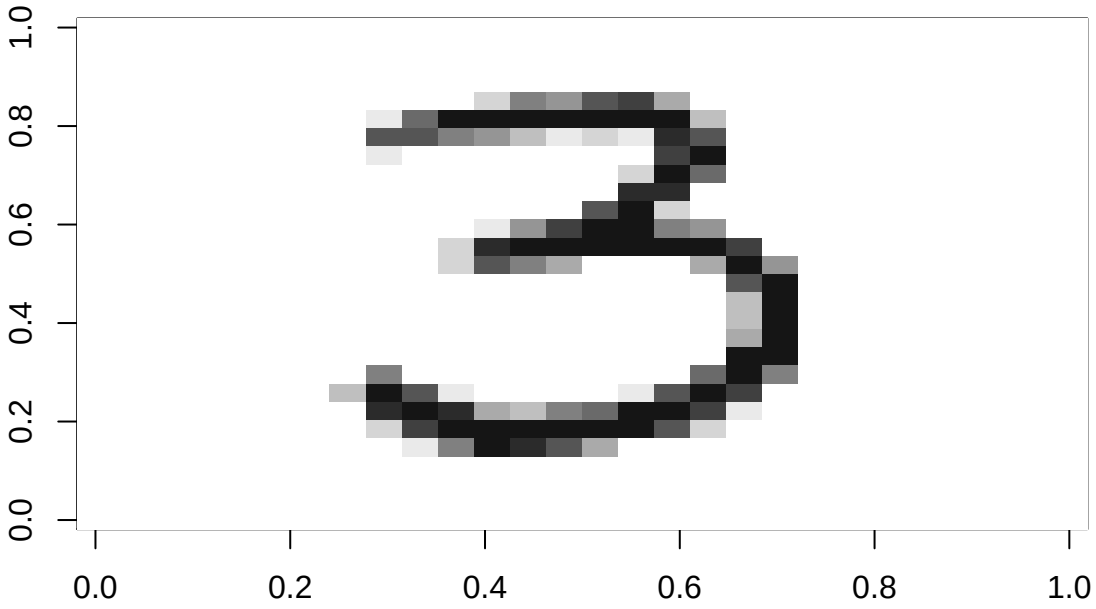

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

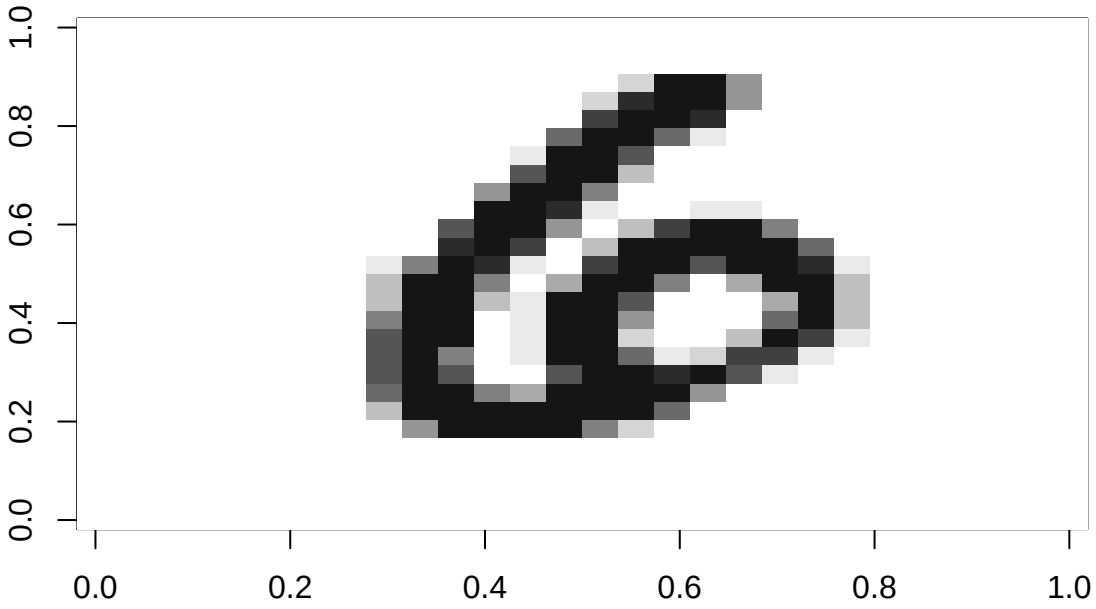

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

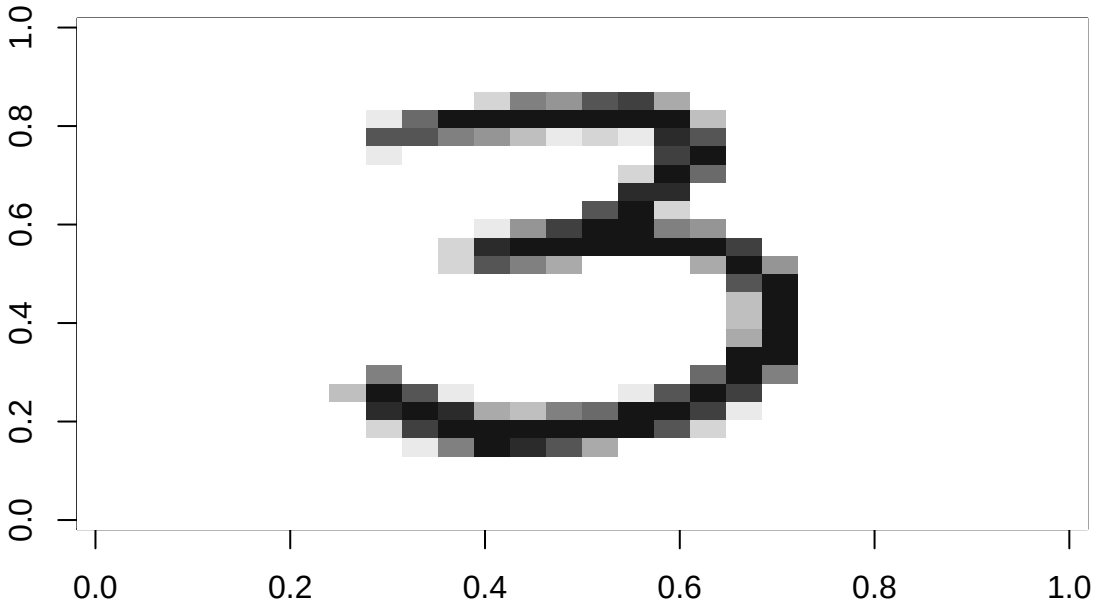

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

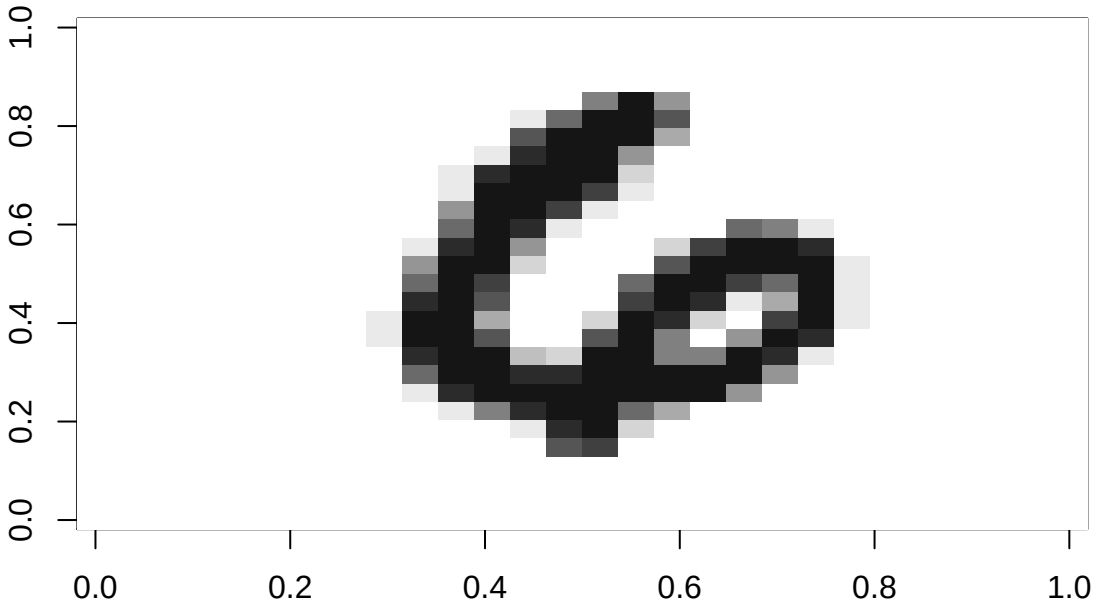

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

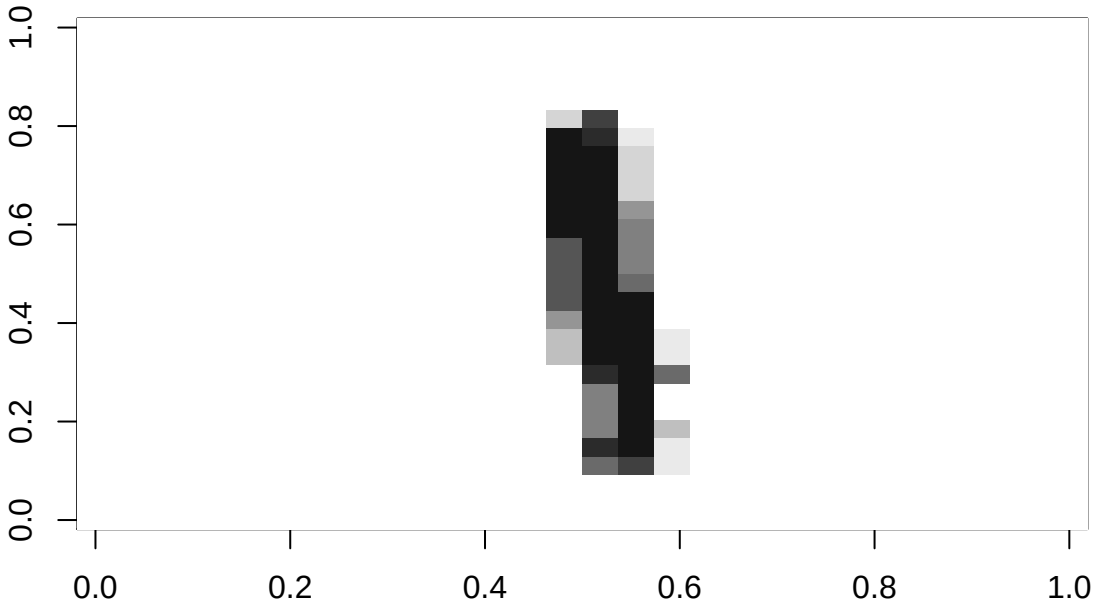

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

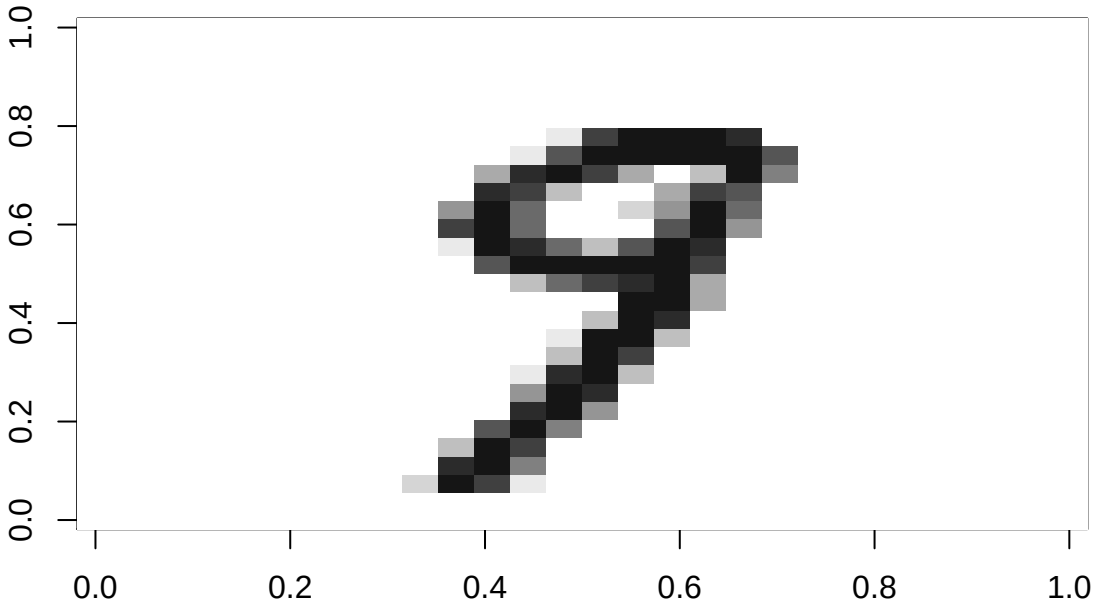

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

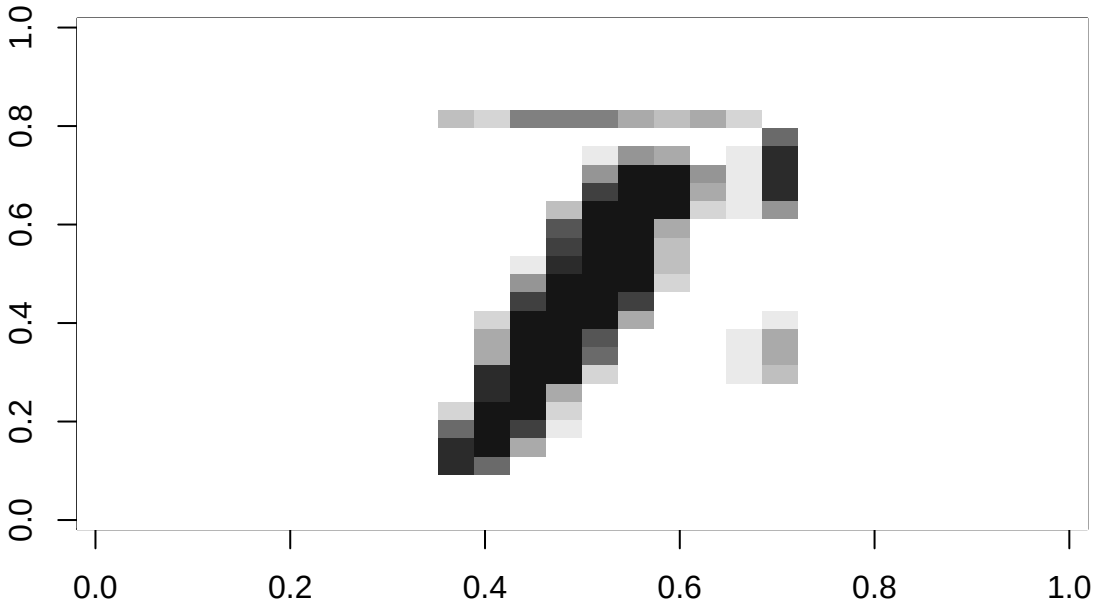

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

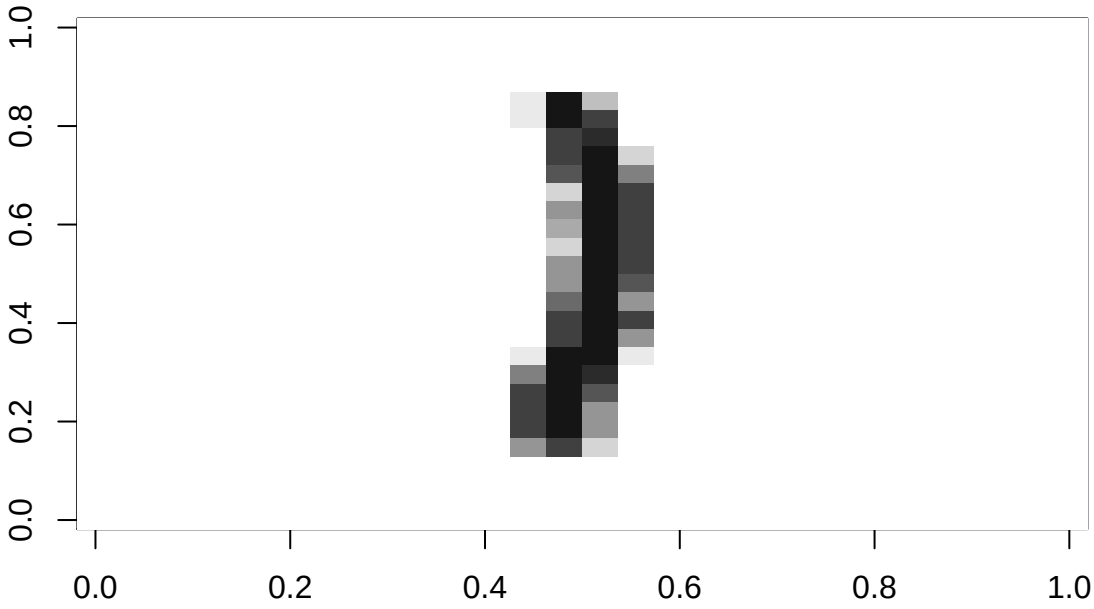


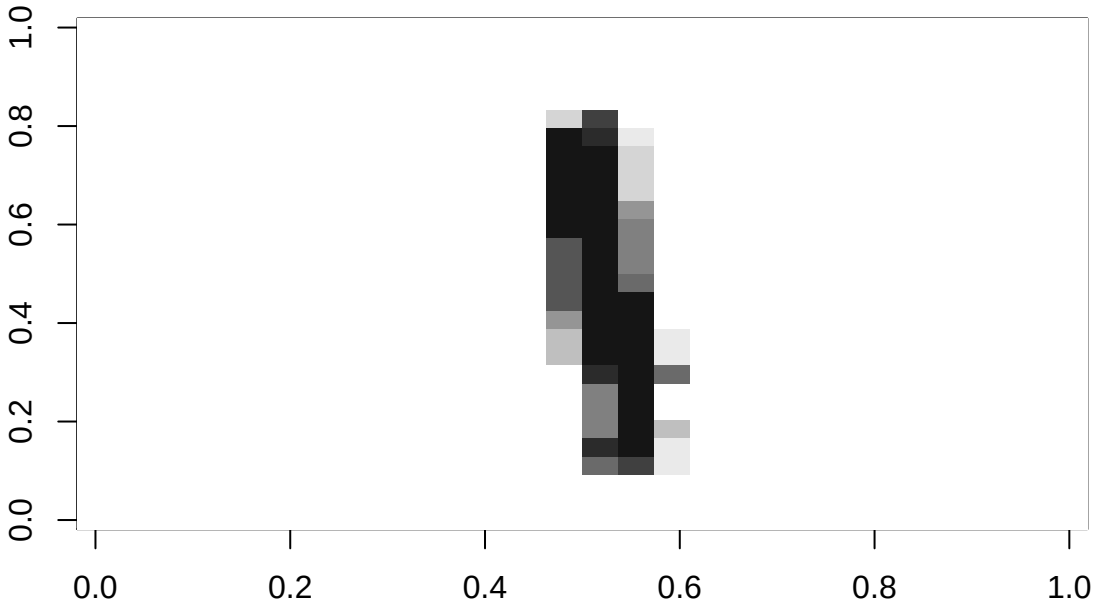

```
km_20 <- my_kmedoids(digits2, 20, 10)
km_20_center <- km_20[[2]]
#show_digit(k_5_center)
for(i in 1:20){
  show_digit(km_20_center[i,])
}
```

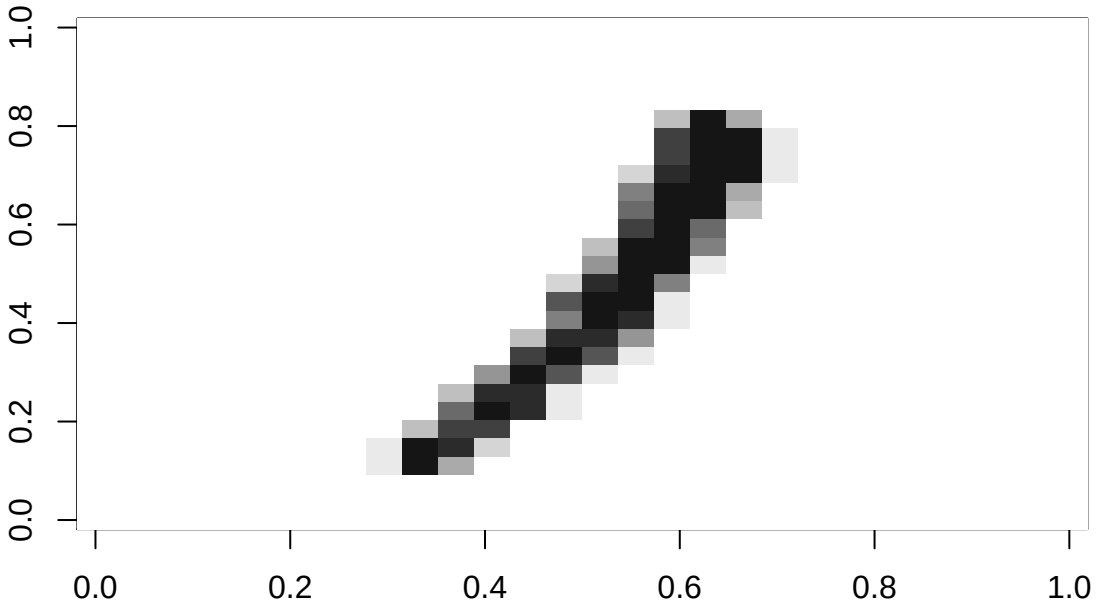

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

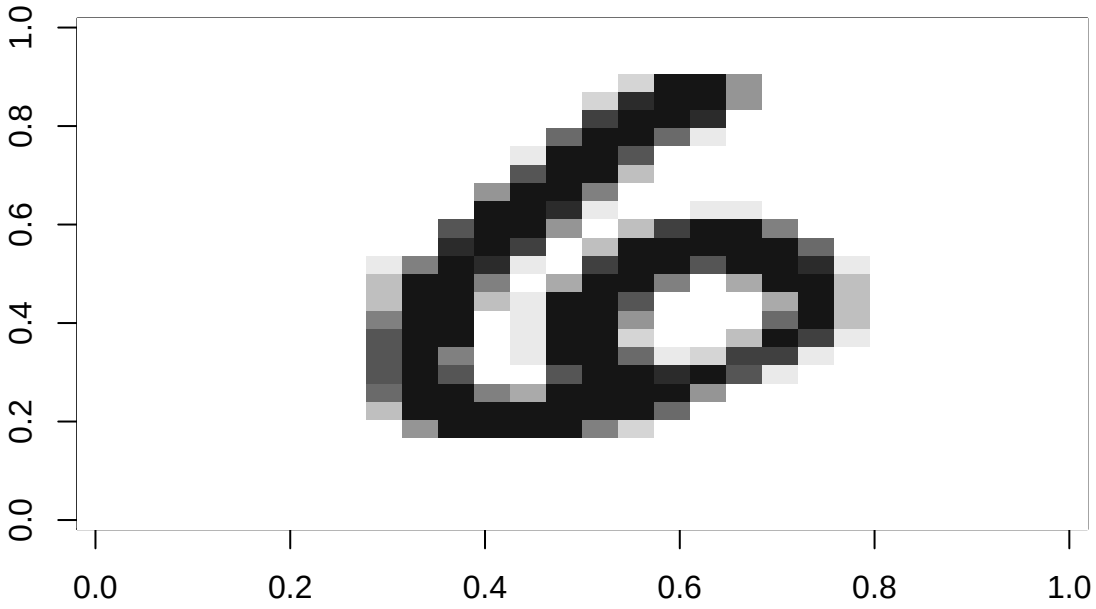

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

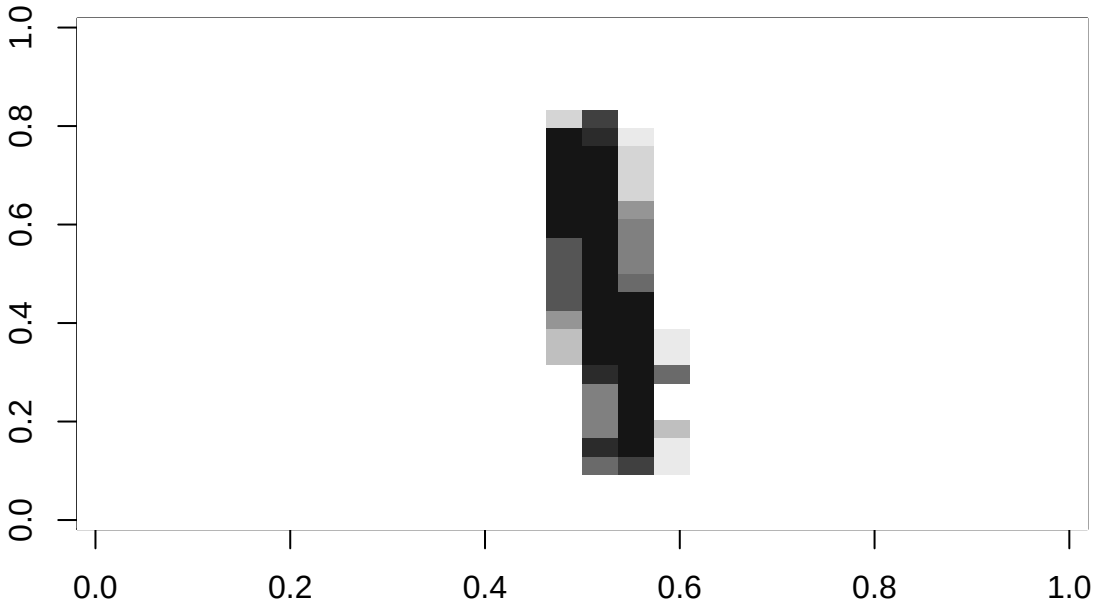

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

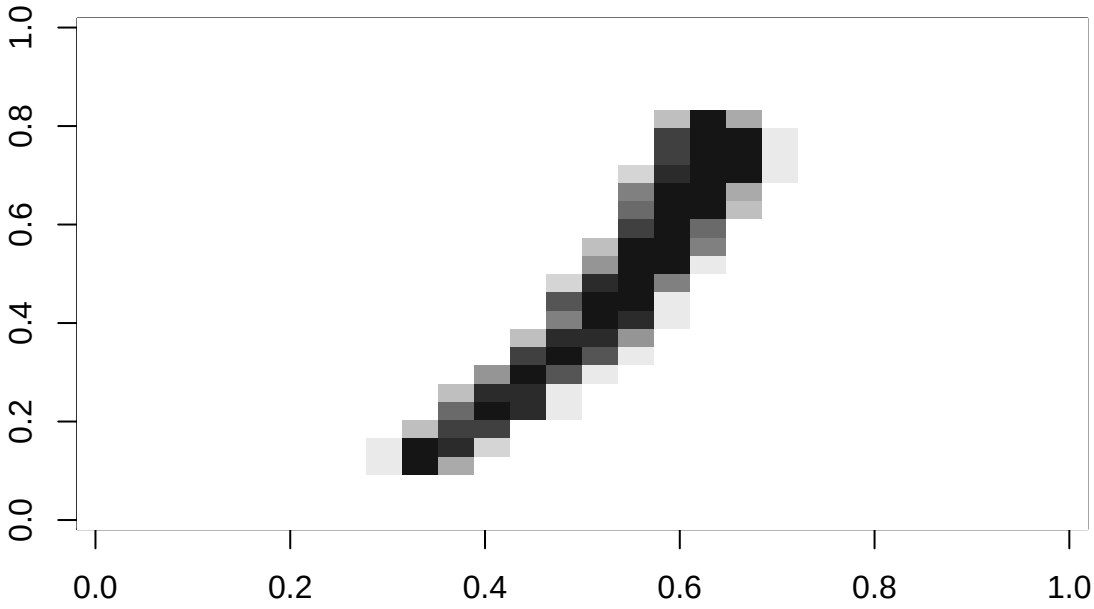

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

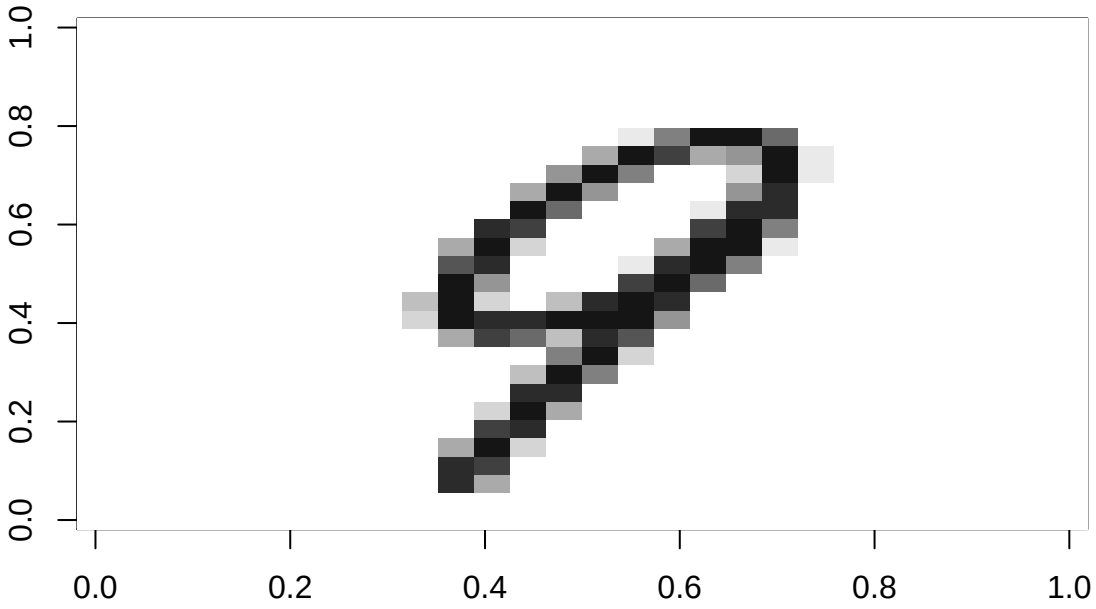

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

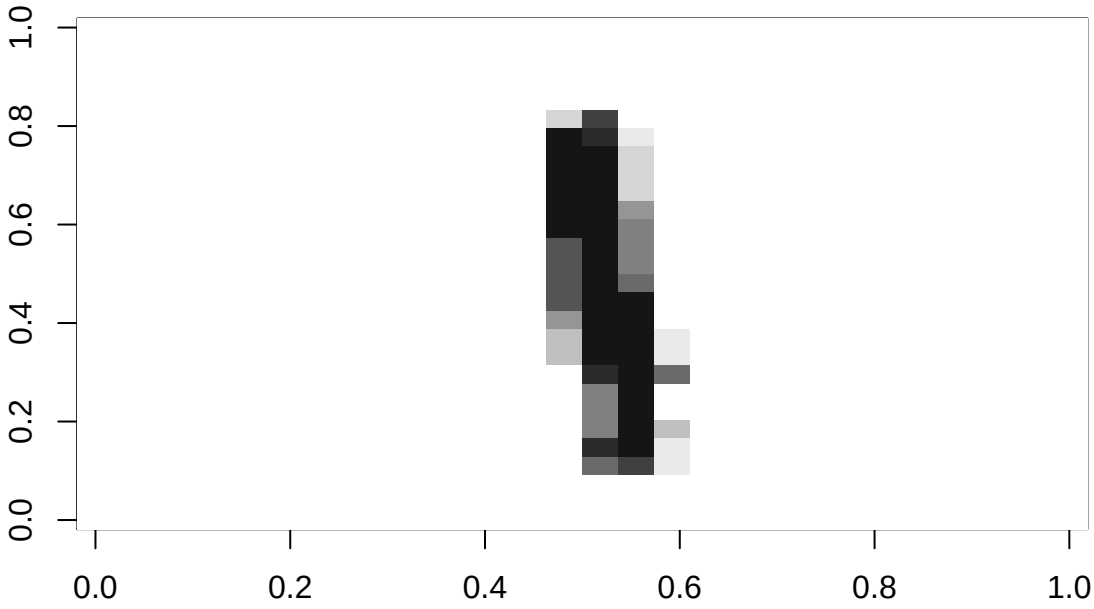

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

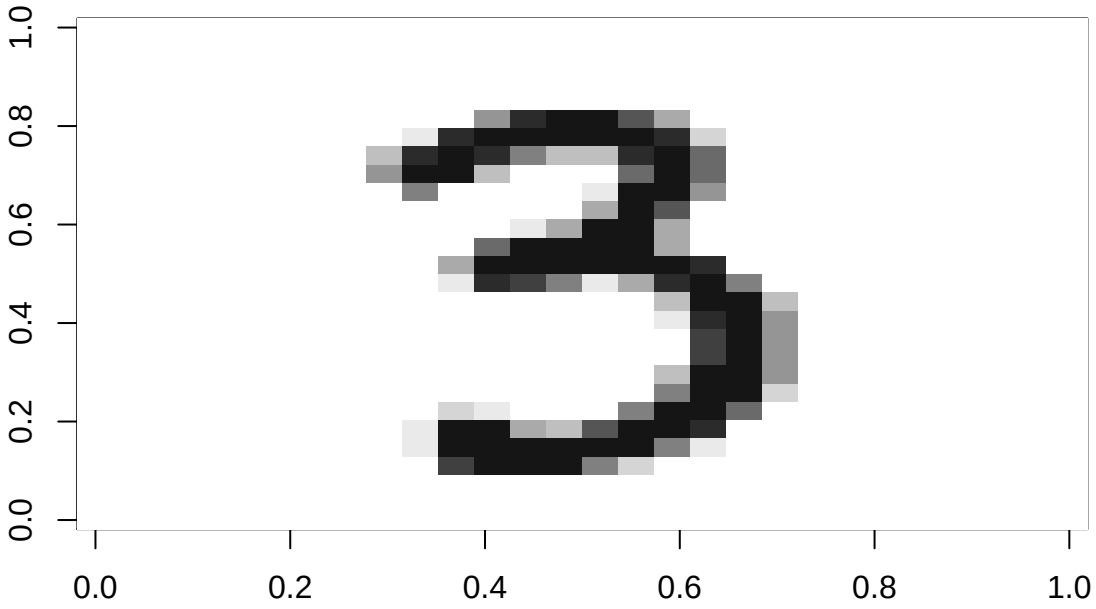

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

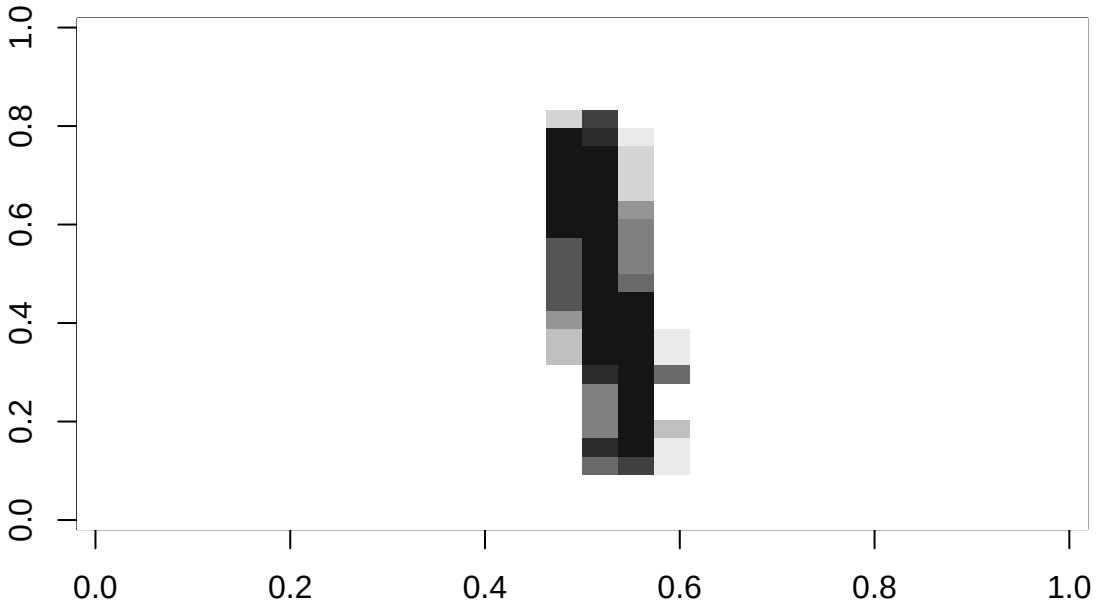

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

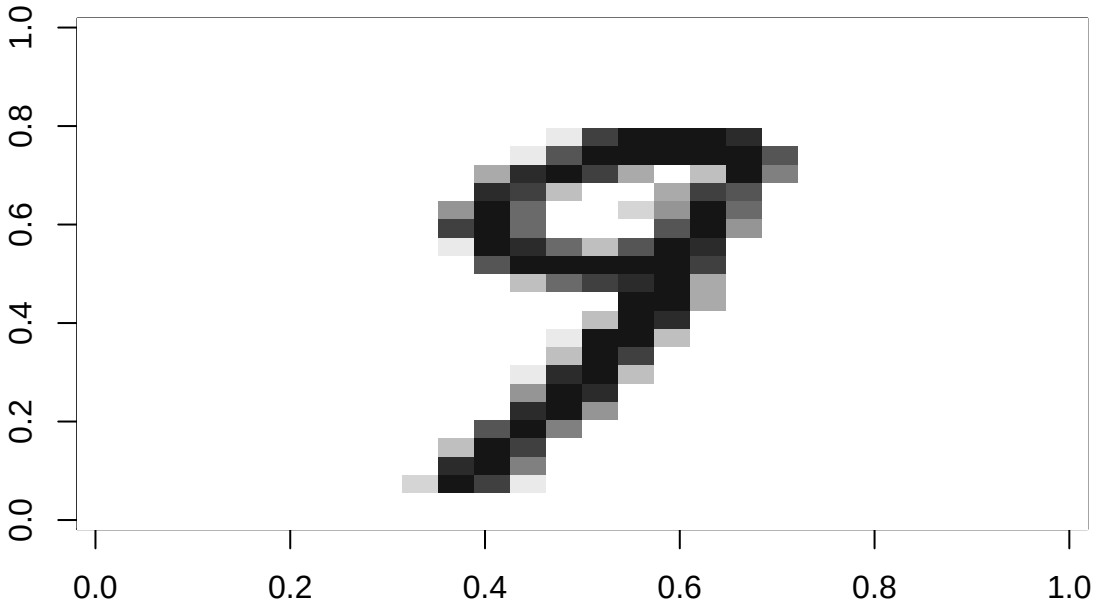

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

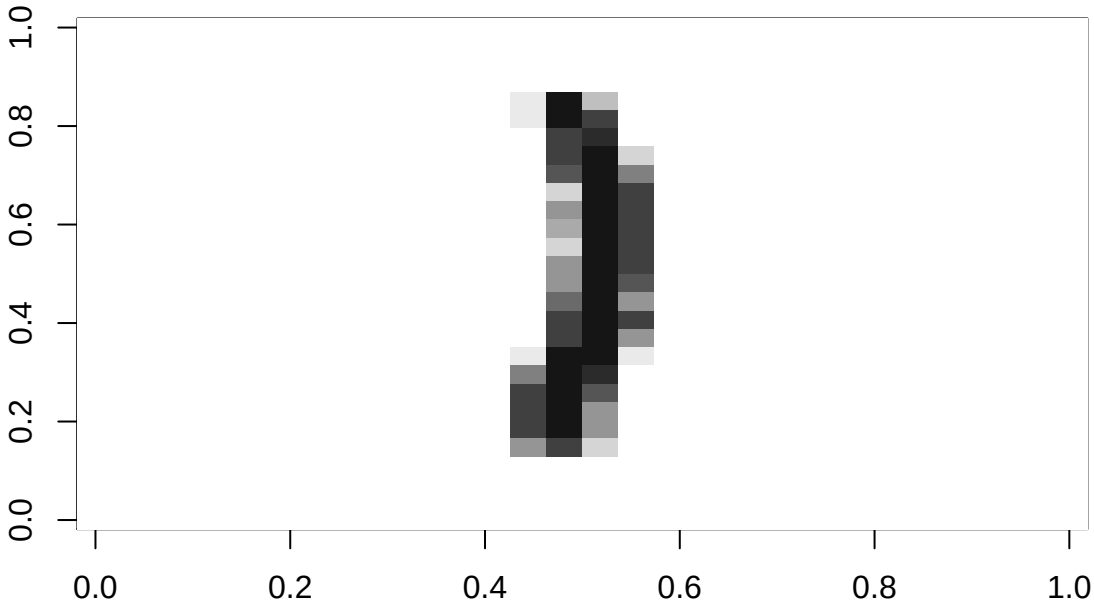

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

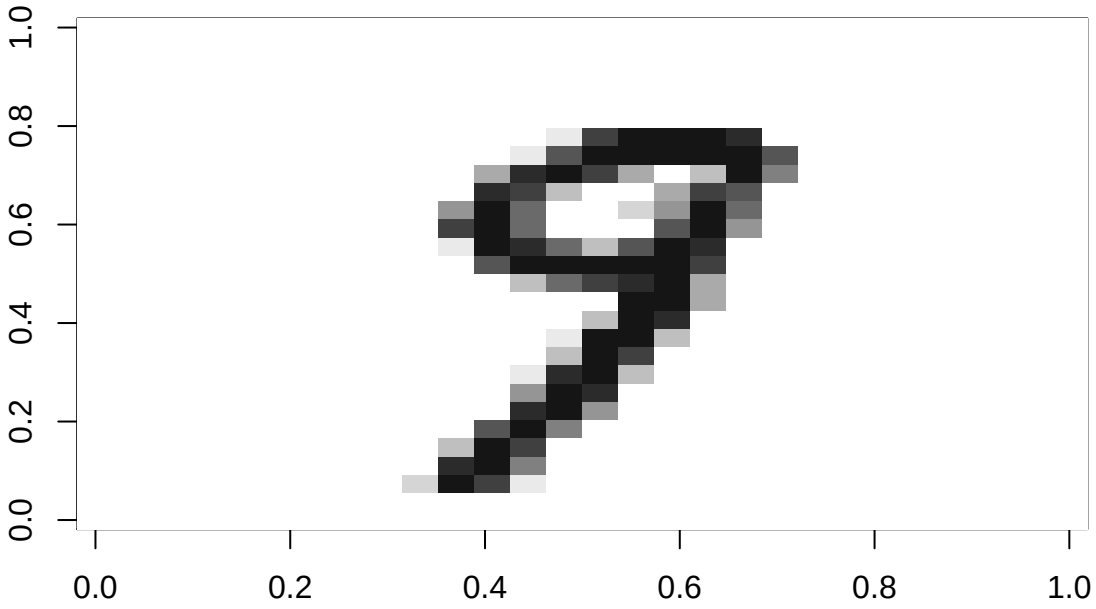

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



###Problem2. HHM ####1.

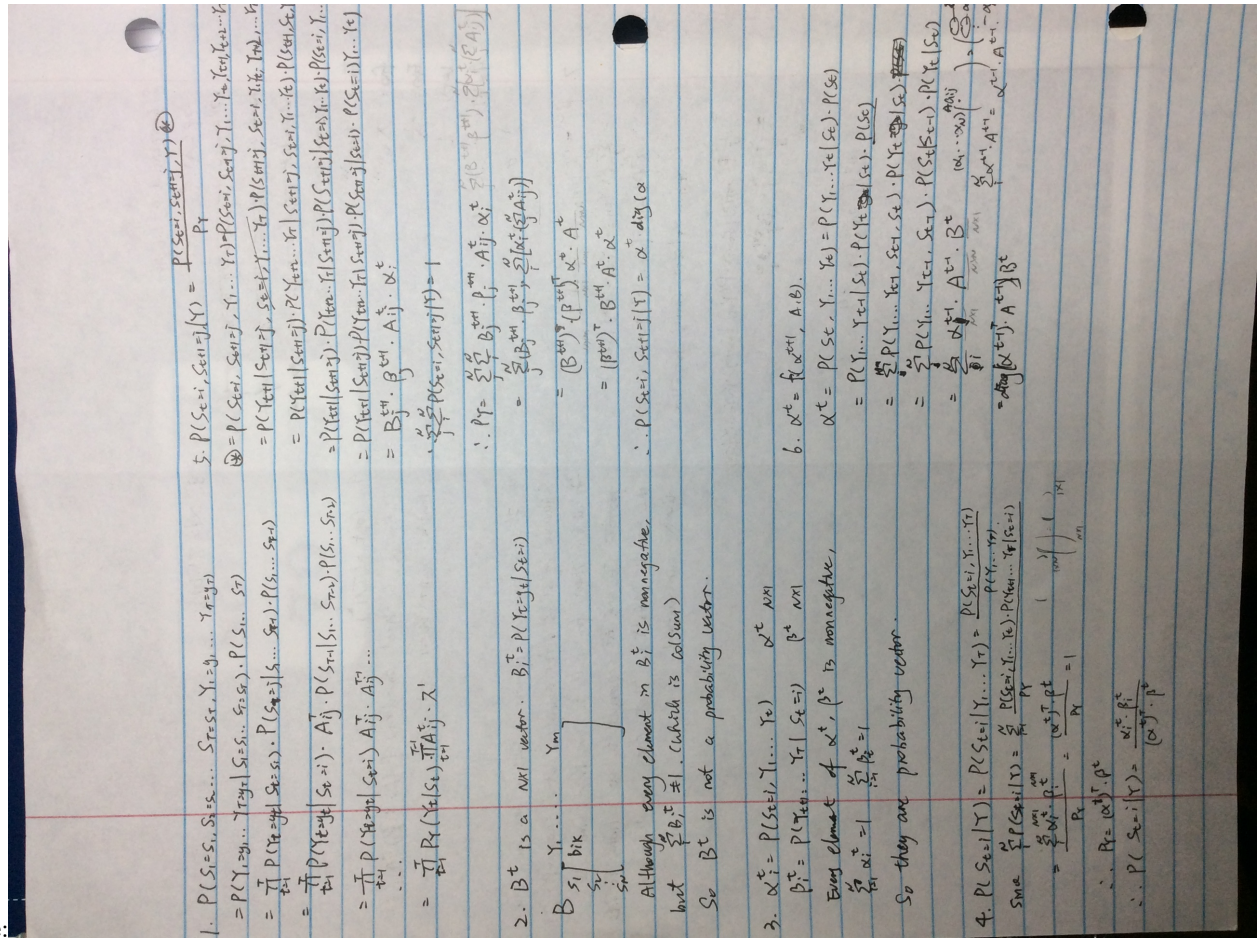
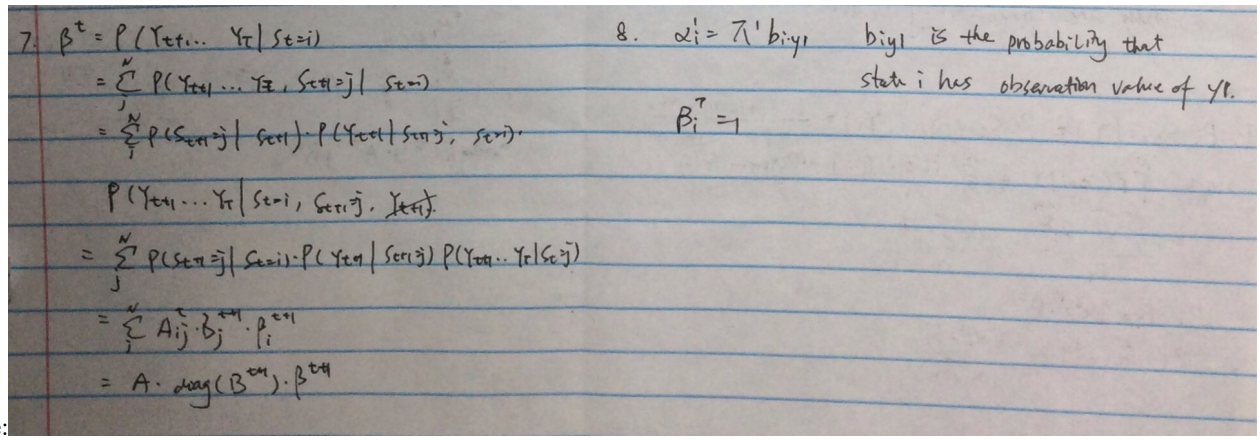
image: 

image: 

9.

It's a bad idea to calculate the most like sequence of states, because that need to list all possible results for each time. For example, if set $S_t = argmax P(S_t|Y)$ for all t, then for each t, we need go through all states to see whether it fits the max $P(S_t|Y)$, which will cost much more than dynamic programming method.