STAT545HOMEWORK3

Xiangyu Xu

September 21, 2017

Problem1

```
if(!require('ggplot2')){
install.packages("ggplot2")
library(ggplot2)
}

## Loading required package: ggplot2

load_mnist <- function() {
  filename <- "C:/Users/Joey/Desktop/STAT545HW3"

  load_image_file <- function(filename) {

    ret = list()

    f = file(filename,'rb')

    readBin(f,'integer',n=1,size=4,endian='big')

    ret$n = readBin(f,'integer',n=1,size=4,endian='big')

    nrow = readBin(f,'integer',n=1,size=4,endian='big')

    ncol = readBin(f,'integer',n=1,size=4,endian='big')

    x = readBin(f,'integer',n=ret$n*nrow*ncol,size=1,signed=F)

    ret$x = matrix(x, ncol=nrow*ncol, byrow=T)

    close(f)

    ret

  }

  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

  }

  train <<- load_image_file('C:/Users/Joey/Desktop/STAT545HW3/train-
images.idx3-ubyte')

  train$y <<- load_label_file('C:/Users/Joey/Desktop/STAT545HW3/train-
labels.idx1-ubyte')
}


show_digit <- function(arr784, col=gray(12:1/12), ...) {

  image(matrix(arr784, nrow=28)[,28:1], col=col, ...)

}

load_mnist()
digits=train$x[1:1000,]
labels=train$y[1:1000]

my_kmeans <- function(digits, K, N){
  #K means number of clusters
  #N means number of runninng K-means
    terminal_loss <- rep(0,N)
    n <- dim(digits)[1]
    f <- dim(digits)[2]
  for (ini in 1:N){
    set.seed(ini)
    #randomly assign observations to clusters
    No_of_cluster <- ceiling(runif(n,0,1) * K)
    last_No_of_cluster <- rep(0,n)
    mean_cluster <- matrix(0, K, f)
    eu_distance <- matrix(0, n, K)
    loss <- c(0)
    loop_loss <- 0
    while(!identical(No_of_cluster,last_No_of_cluster)){
      loop_loss <- loop_loss + 1
      for (k in 1:K){
        index <- which(No_of_cluster == k)
        if(sum(index) > 0){
```

```
        #calculate the corresponding cluster center for every cluster
          mean_cluster[k,] <- apply(digits[index, ], 2, mean)
        }
        #if empty cluster happens
        else{
          mean_cluster[k,] <- rnorm(f,0,1)
        }
      }
      for(k in 1:k){
        #calculate the encludean distance
        eu_distance[,k] <- apply(t((t(digits) - mean_cluster[k,])^2), 1, sum)
        last_No_of_cluster <- No_of_cluster
      }
        No_of_cluster <- apply(eu_distance, 1, which.min)
        loss[loop_loss] <- sum(apply(eu_distance, 1, min))
    }
      if(ini == 1){
        Final_loss_seq <- loss[-1]
        Final_Cluster_Assignment <- No_of_cluster
        Final_Cluster_Parameter <- mean_cluster
        terminal_loss[ini] <- loss[length(loss)]
      }
      else{
        if(loss[length(loss)]<Final_loss_seq[length(Final_loss_seq)]){
           Final_loss_seq <- loss[-1]
           Final_Cluster_Assignment <- No_of_cluster
           Final_Cluster_Parameter <- mean_cluster
        }
        else{
           Final_loss_seq <- Final_loss_seq
           Final_Cluster_Assignment <- No_of_cluster
           Final_Cluster_Parameter <- mean_cluster
        }
           terminal_loss[ini] <- loss[length(loss)]
      }
    }
  return(list(Final_Cluster_Parameter <- Final_Cluster_Parameter,
              Final_Cluster_Assignment <- Final_Cluster_Assignment,
              Final_loss_seq <- Final_loss_seq,
              terminal_loss <- terminal_loss))
}
```

3

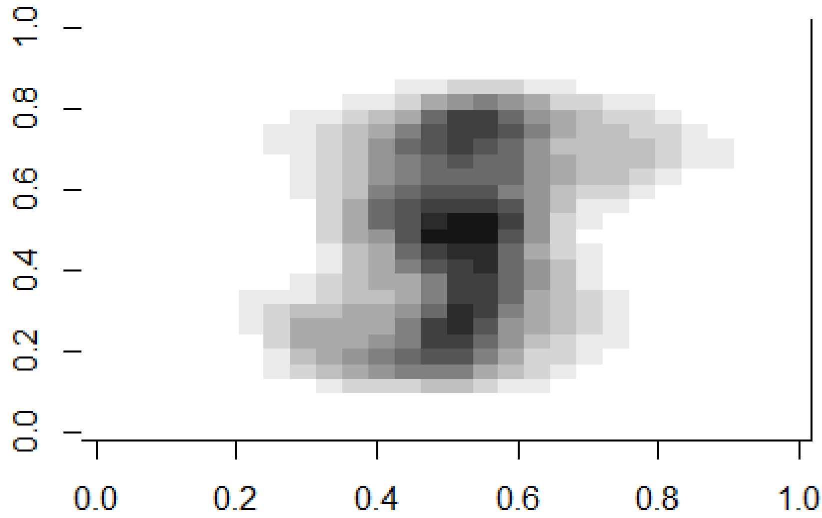
When the cluster assignment to every vector stop changing, the iteration can be stopped.

4. k=5

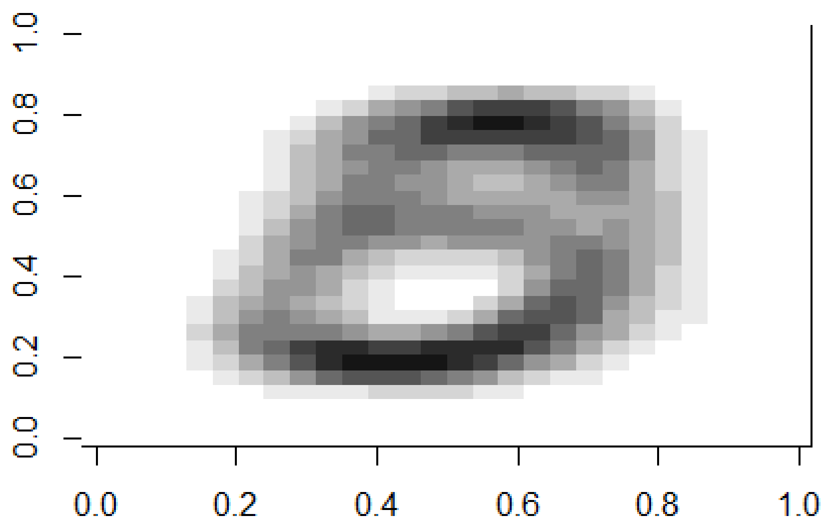
```
k_5 <- my_kmeans(digits,5,25)
par(mfcol <- c(2,3))
```

```
## NULL
```

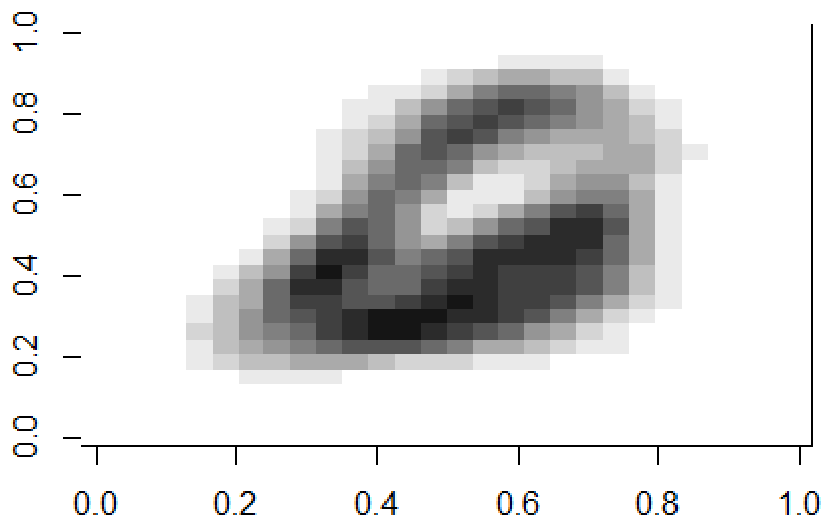
```r
for (k in 1:5){
show_digit(k_5[[1]][k,])
}
```



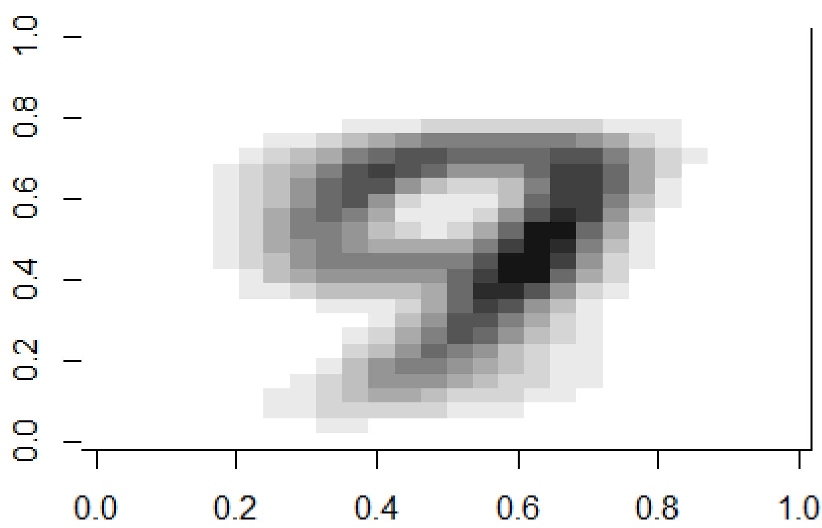
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



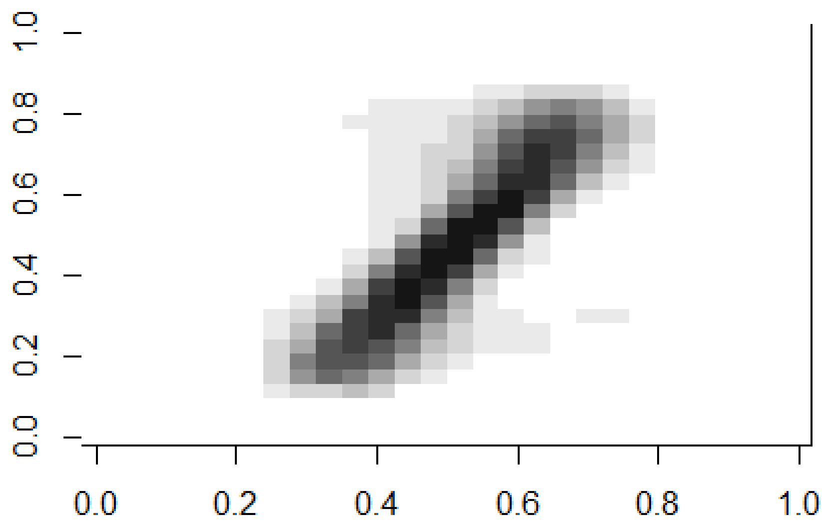
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



0.0
0.2
0.4
0.6
0.8
1.0

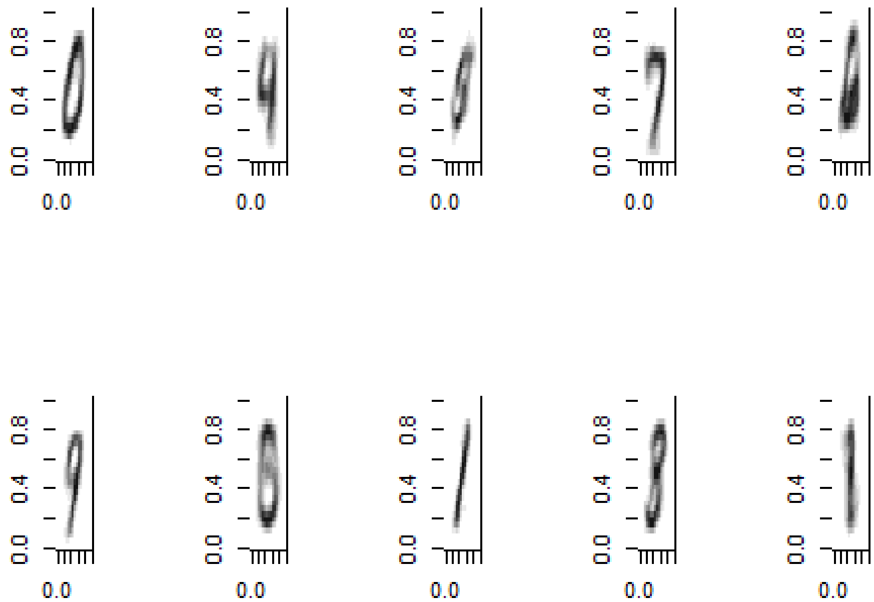


0.0
0.2
0.4
0.6
0.8
1.0



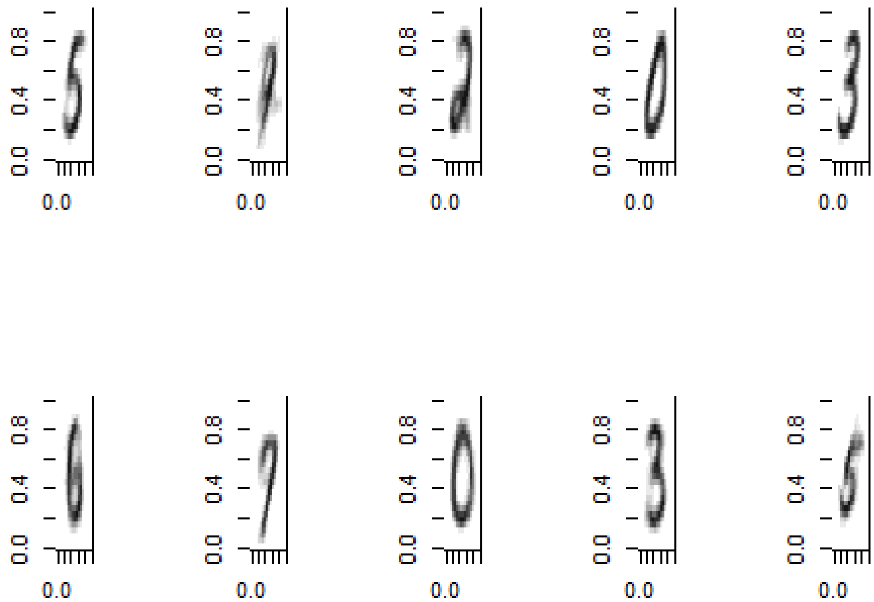
k=10

```
k_10=my_kmeans(digits,10,20)
par(mfcol=c(2,5))
for (k in 1:10){
  show_digit(k_10[[1]][k,])
}
```

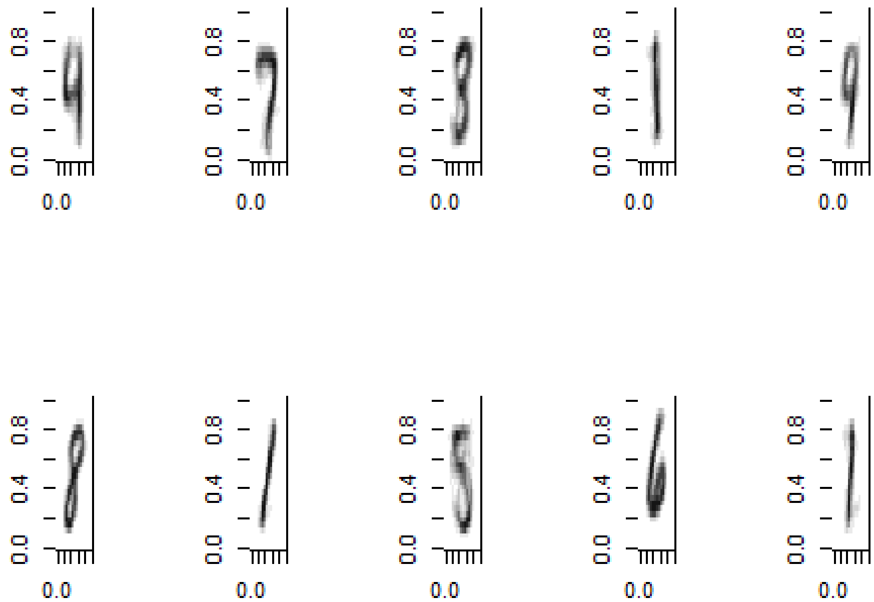


k=20

```
k_20=my_kmeans(digits,20,20)
par(mfcol=c(2,5))
for (k in 1:10){
show_digit(k_20[[1]][k,])
}
```

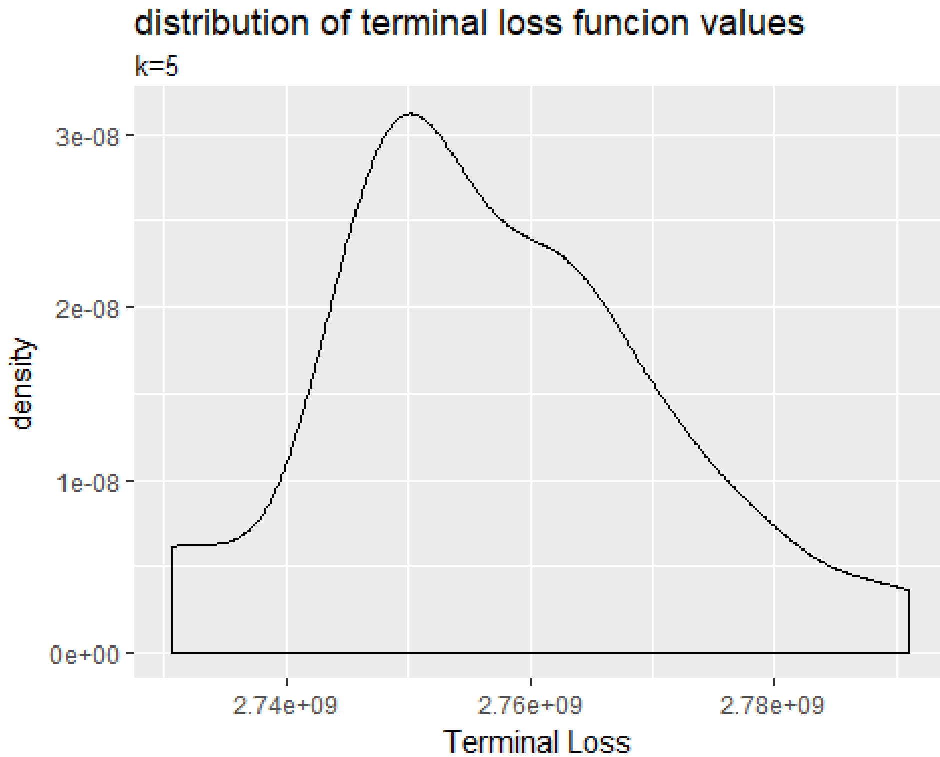



```
par(mfcol=c(2,5))
for (k in 11:20){
show_digit(k_20[[1]][k,])
}
```



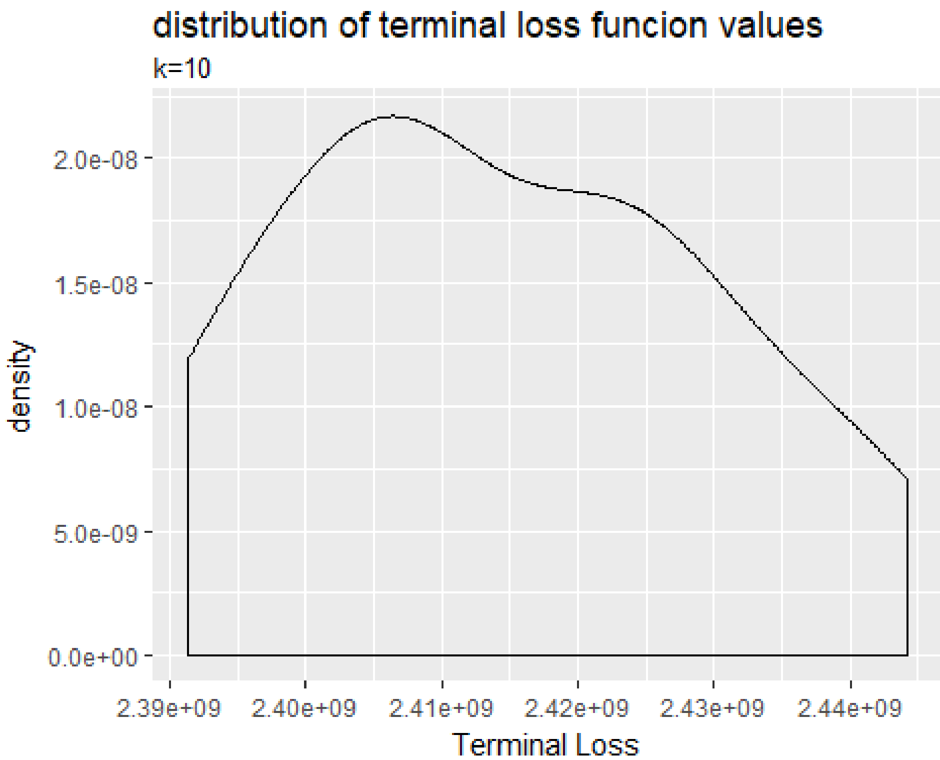
5. k=5

```
k_5_df <- data.frame(k_5[[4]])
ggplot(k_5_df,aes(x <- data.frame(k_5[[4]]))) + geom_density(colour="black")
+ ggtitle(title <-  "distribution of terminal loss funcion values", subtitle
<- "k=5" ) + labs(x="Terminal Loss")

## Don't know how to automatically pick scale for object of type data.frame.
Defaulting to continuous.
```



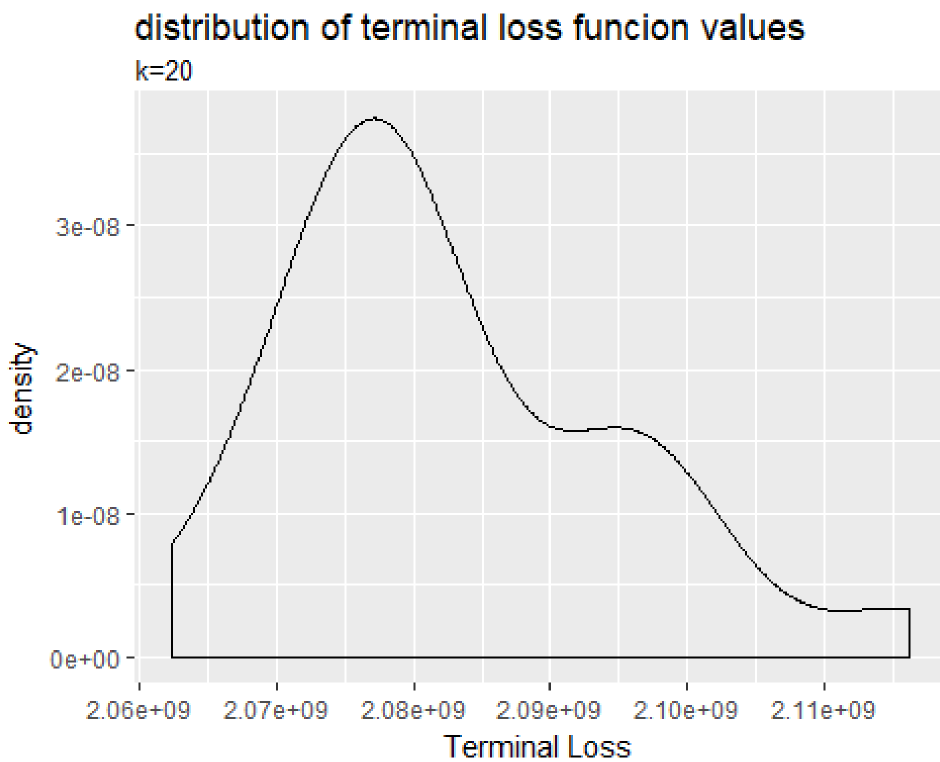
k=10

```
k_10_df <- data.frame(k_10[[4]])
ggplot(k_10_df,aes(x <- data.frame(k_10[[4]]))) +
geom_density(colour="black") + ggtitle(title <-  "distribution of terminal
loss funcion values", subtitle <- "k=10" ) + labs(x="Terminal Loss")

## Don't know how to automatically pick scale for object of type data.frame.
Defaulting to continuous.
```



k=20

```
k_20_df <- data.frame(k_20[[4]])
ggplot(k_20_df,aes(x <- data.frame(k_20[[4]]))) +
geom_density(colour="black") + ggtitle(title <-  "distribution of terminal
loss funcion values", subtitle <- "k=20" ) + labs(x="Terminal Loss")

## Don't know how to automatically pick scale for object of type data.frame.
Defaulting to continuous.
```



Apparently, if we choose k=1, the error would be the largest. As the k increases, error will decrease, and it can become 0 if k equal to the number of data points. So we should strike a balance between maximum compression of data using a single cluster, and maximum accuracy by assigning each data point to its own cluster. We could implement the elbow method. Choose a number of k so that adding another cluster doesn't give much better modeling of the data. More precisely, if we plot the percentage of variance against the k, the first cluster will add much information, but at the elbow point, the gain of variance explained will drop, thus the the corresponding k to the elbow point is the value we should choose.

Bonus 7.

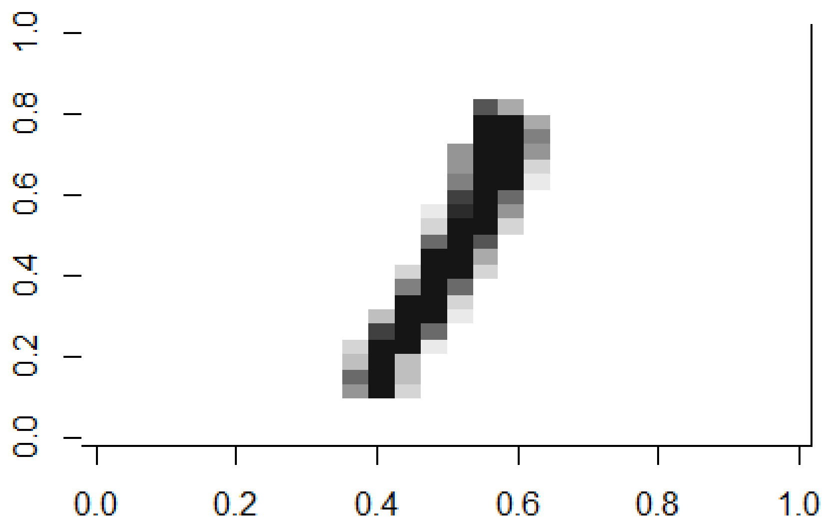
```
get_prototype <- function(data){
  dim_data <- dim(data)[1]
  for (ii in 1:dim_data){
    distance_medoids <- sum((t(data) - data[ii,])^2)
    if(ii == 1){
      final_distance_medoids <- distance_medoids
      center <- ii
    }
    else{
      if(distance_medoids <= final_distance_medoids){
        final_distance_medoids <- distance_medoids
        center <- ii
      }
      else{
```

```
        final_distance_medoids <- final_distance_medoids
        center <- center
      }
    }
  }
  return(data[center,])
}

my_kmedoids <- function(digits, K, N){
  #K means number of clusters
  #N means number of runninng K-means
    terminal_loss <- rep(0,N)
    n <- dim(digits)[1]
    f <- dim(digits)[2]
  for (ini in 1:N){
    set.seed(ini)
    #randomly assign observations to clusters
    No_of_cluster <- ceiling(runif(n,0,1) * K)
    last_No_of_cluster <- rep(0,n)
    mean_cluster <- matrix(0, K, f)
    eu_distance <- matrix(0, n, K)
    loss <- c(0)
    loop_loss <- 0
    while(!identical(No_of_cluster,last_No_of_cluster)){
      loop_loss <- loop_loss + 1
      for (k in 1:K){
        index <- which(No_of_cluster == k)
        if(sum(index) > 0){
        #calculate the corresponding cluster center for every cluster
          mean_cluster[k,] <- get_prototype(digits[index,])
        }
      }
      for(k in 1:k){
        #calculate the encludean distance
        eu_distance[,k] <- apply(t((t(digits) - mean_cluster[k,])^2), 1, sum)
        last_No_of_cluster <- No_of_cluster
      }
        No_of_cluster <- apply(eu_distance, 1, which.min)
        loss[loop_loss] <- sum(apply(eu_distance, 1, min))
    }
      if(ini == 1){
        Final_loss_seq <- loss[-1]
        Final_Cluster_Assignment <- No_of_cluster
        Final_Cluster_Parameter <- mean_cluster
        terminal_loss[ini] <- loss[length(loss)]
      }
      else{
        if(loss[length(loss)]<Final_loss_seq[length(Final_loss_seq)]){
           Final_loss_seq <- loss[-1]
           Final_Cluster_Assignment <- No_of_cluster
```

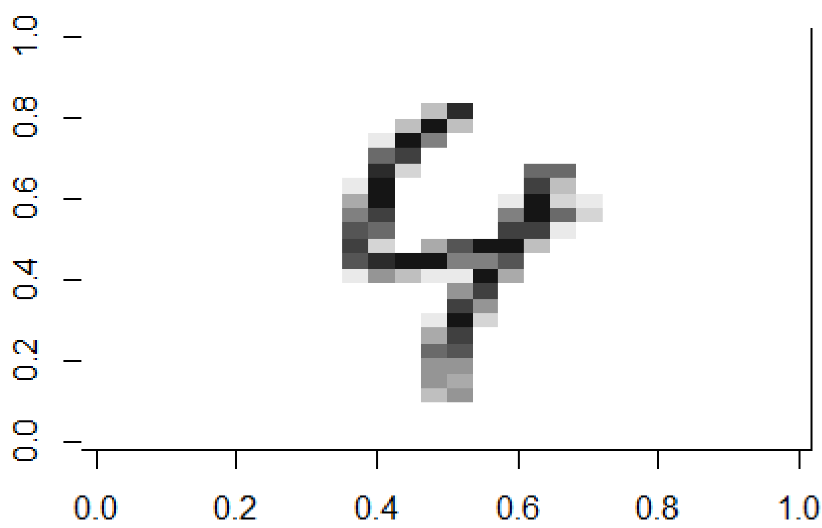
```
            Final_Cluster_Parameter <- mean_cluster
         }
         else{
            Final_loss_seq <- Final_loss_seq
            Final_Cluster_Assignment <- No_of_cluster
            Final_Cluster_Parameter <- mean_cluster
         }
            terminal_loss[ini] <- loss[length(loss)]
       }
     }
  return(list(Final_Cluster_Parameter <- Final_Cluster_Parameter,
               Final_Cluster_Assignment <- Final_Cluster_Assignment,
               Final_loss_seq <- Final_loss_seq,
               terminal_loss <- terminal_loss))
}
```

k=5

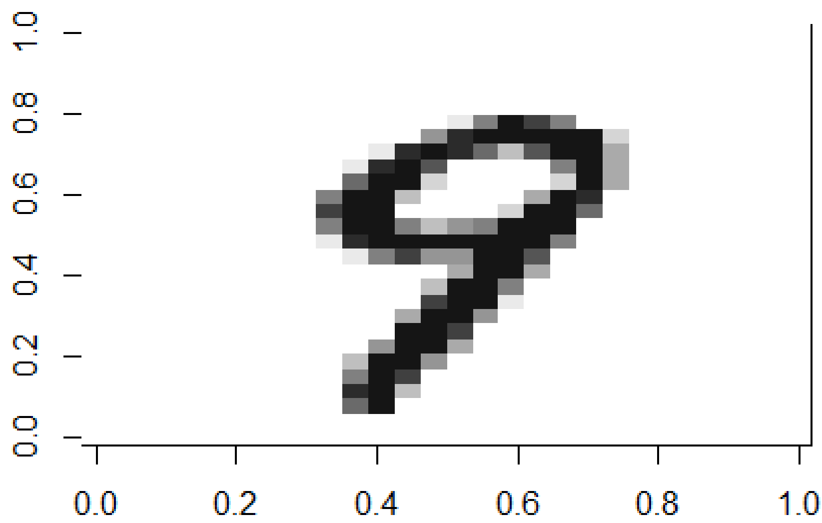
```
bonus_k_5 <- my_kmedoids(digits,5,20)
par(mfcol <- c(2,3))

## NULL

for (k in 1:5){
show_digit(bonus_k_5[[1]][k,])
}
```

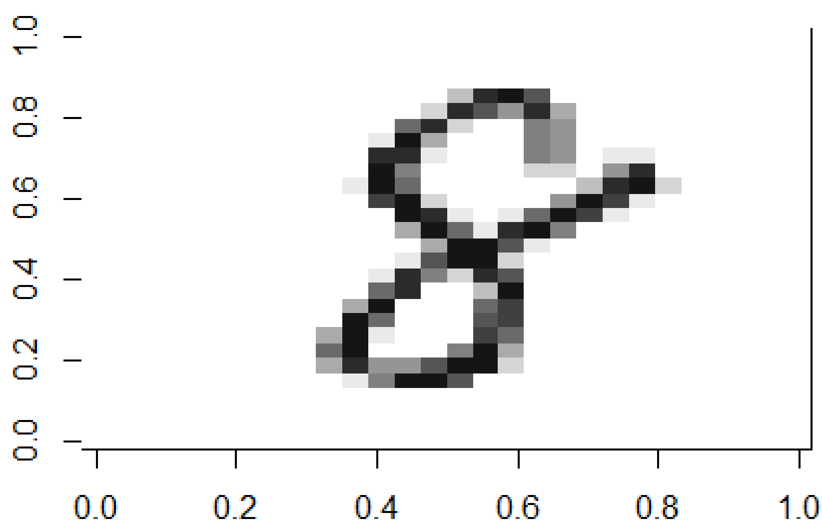


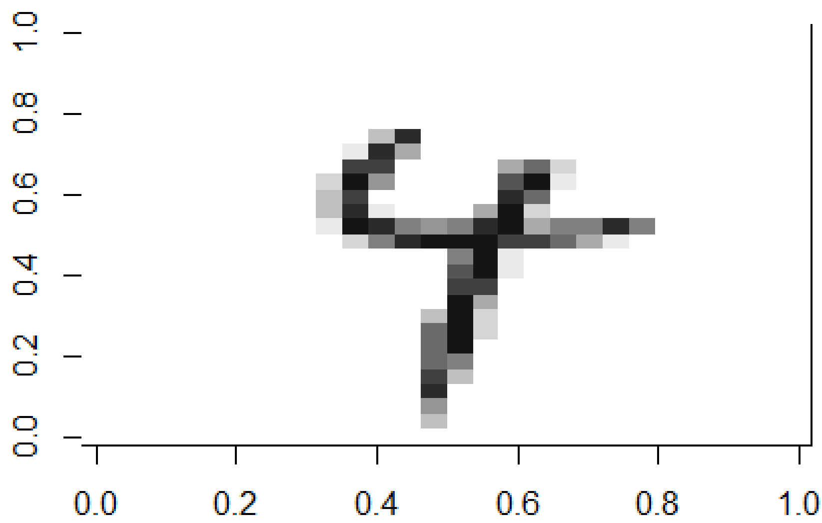
0.0
0.2
0.4
0.6
0.8
1.0



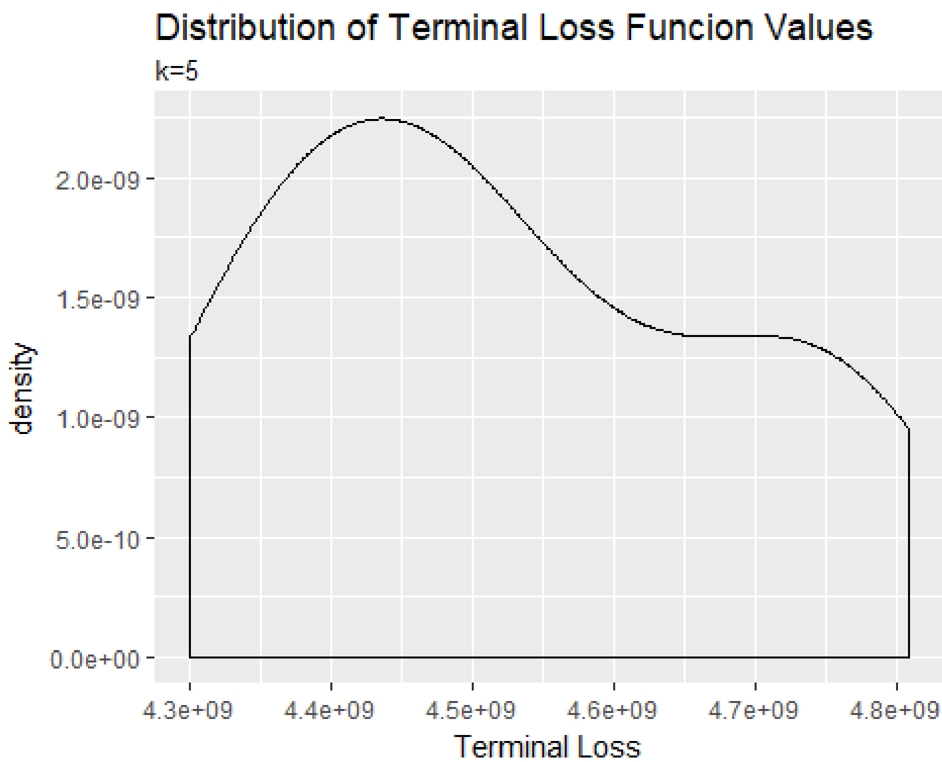
0.0
0.2
0.4
0.6
0.8
1.0


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

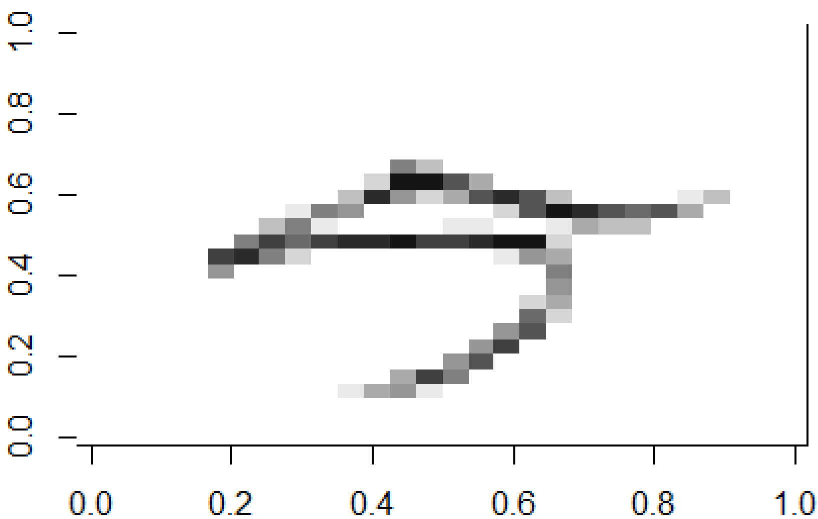


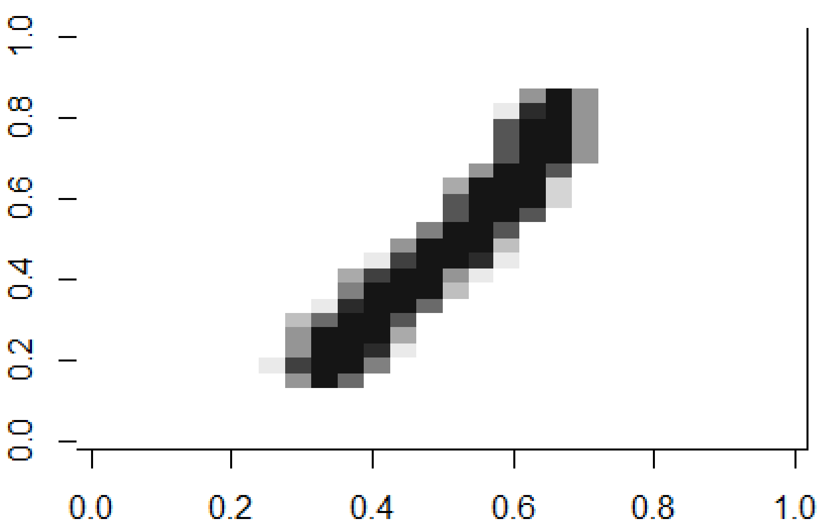

```
bonus_k_5_df <- data.frame(bonus_k_5[[4]])
ggplot(bonus_k_5_df,aes(x <- data.frame(bonus_k_5[[4]]))) +
geom_density(colour="black") + ggtitle(title <-  "Distribution of Terminal
Loss Funcion Values", subtitle <- "k=5" ) + labs(x="Terminal Loss")

## Don't know how to automatically pick scale for object of type data.frame.
Defaulting to continuous.
```

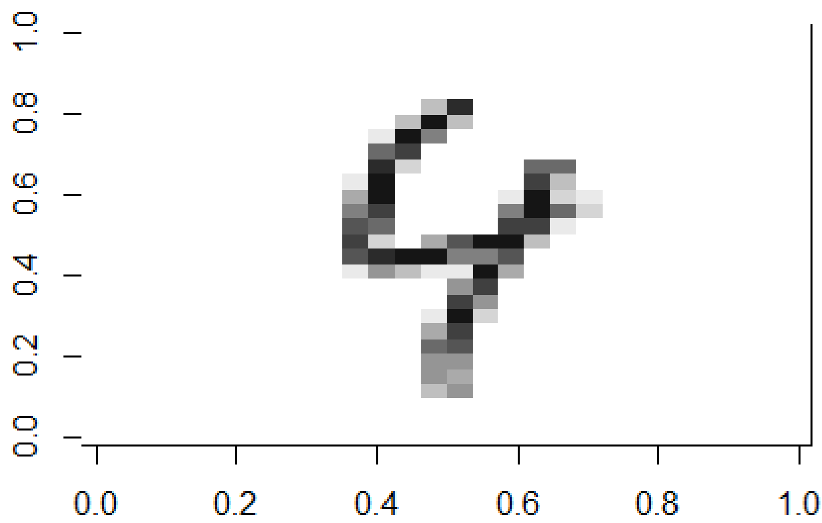


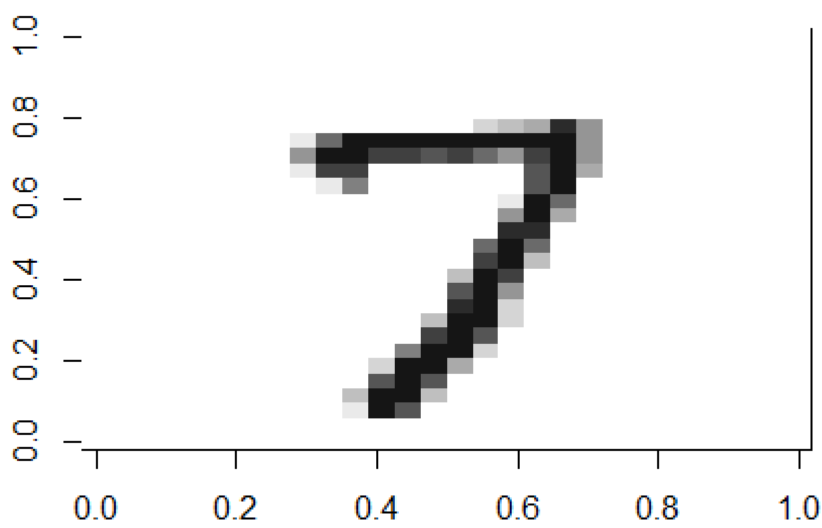
k=10

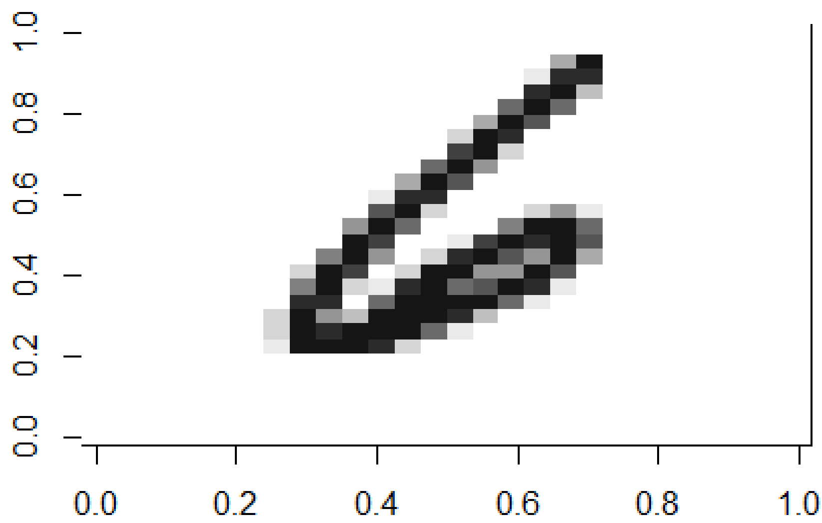
```
bonus_k_10 <- my_kmedoids(digits,10,20)
par(mfcol <- c(2,5))

## NULL

for (k in 1:10){
show_digit(bonus_k_10[[1]][k,])
}
```

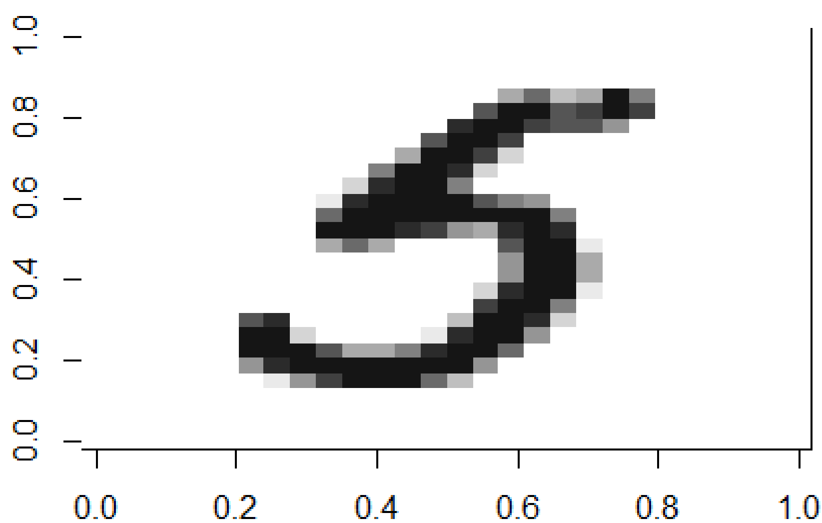

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

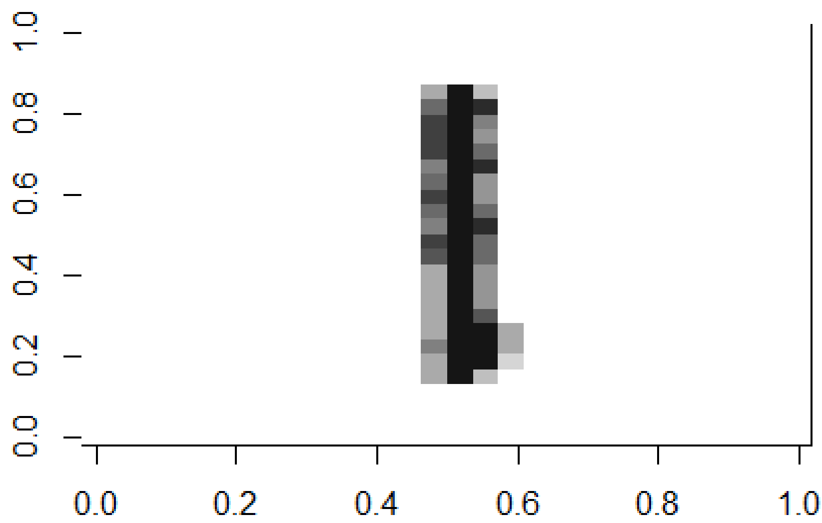

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


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

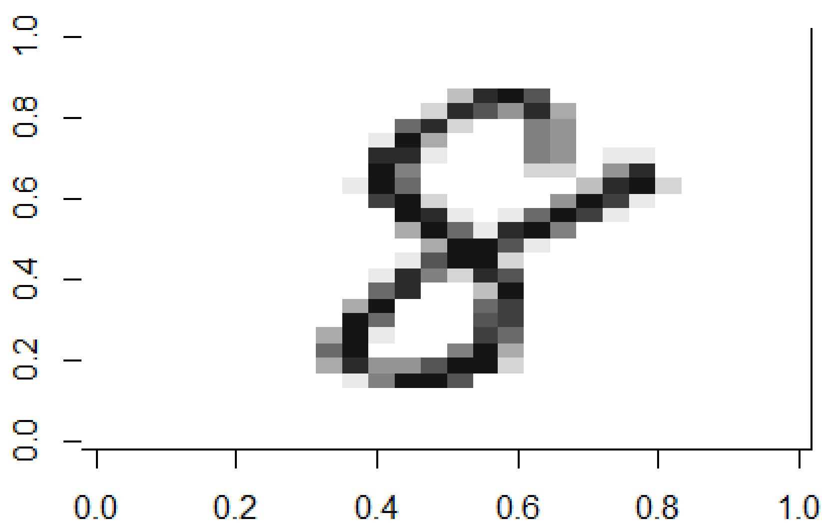

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


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

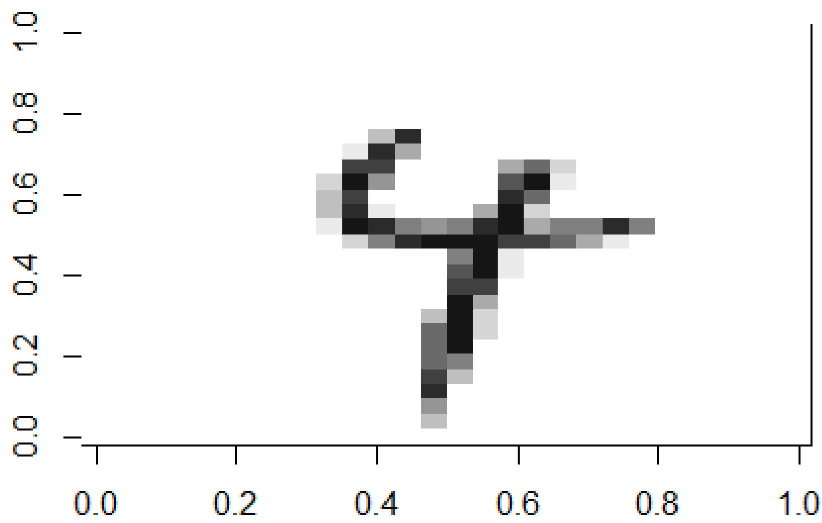

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

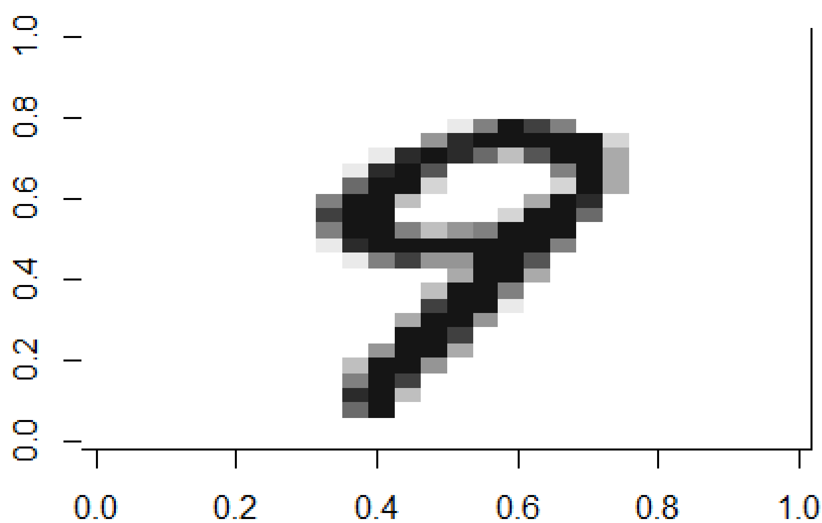


0.0
0.2
0.4
0.6
0.8
1.0



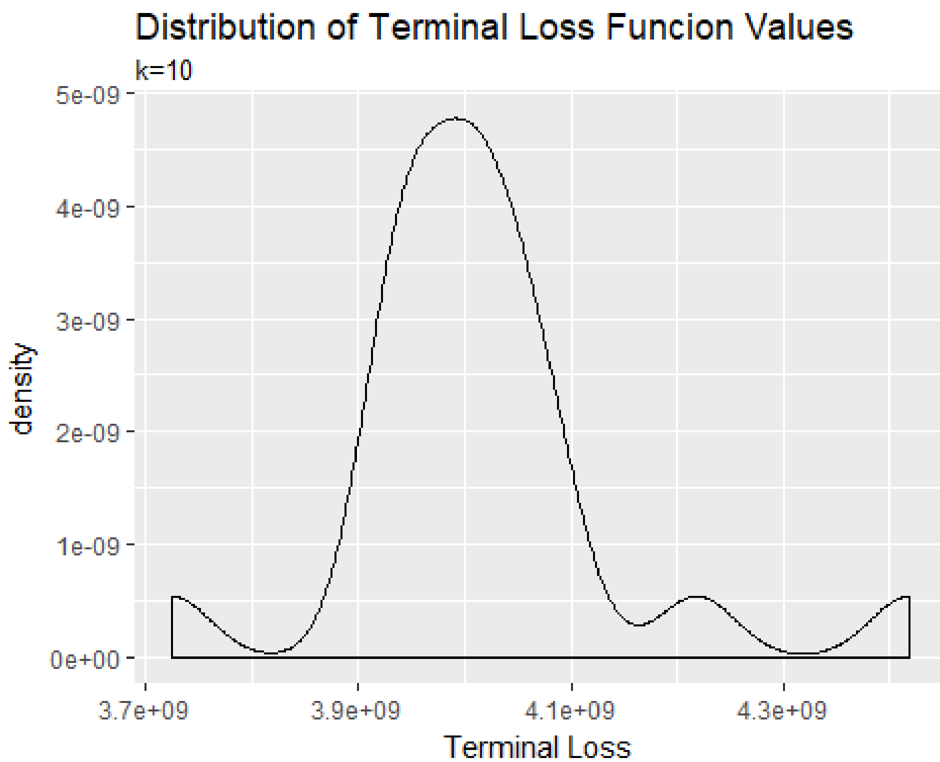
0.0
0.2
0.4
0.6
0.8
1.0





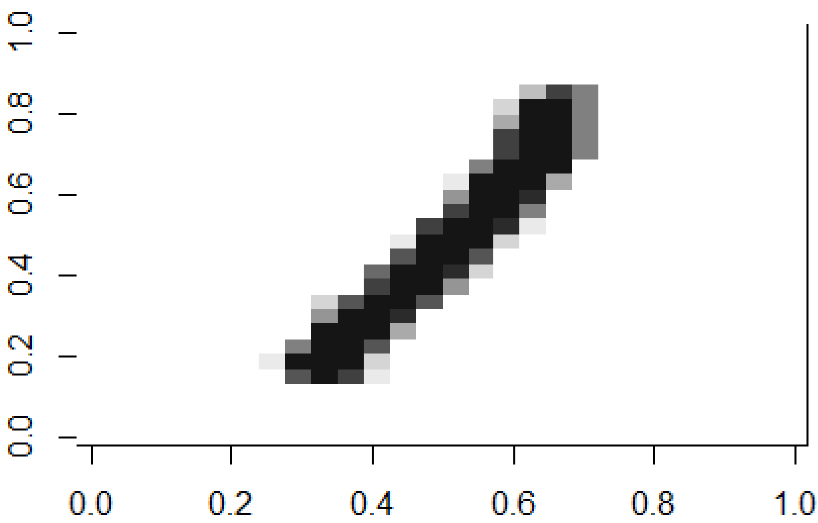

```
bonus_k_10_df <- data.frame(bonus_k_10[[4]])
ggplot(bonus_k_10_df,aes(x <- data.frame(bonus_k_10[[4]]))) +
geom_density(colour="black") + ggtitle(title <-  "Distribution of Terminal
Loss Funcion Values", subtitle <- "k=10" ) + labs(x="Terminal Loss")
```

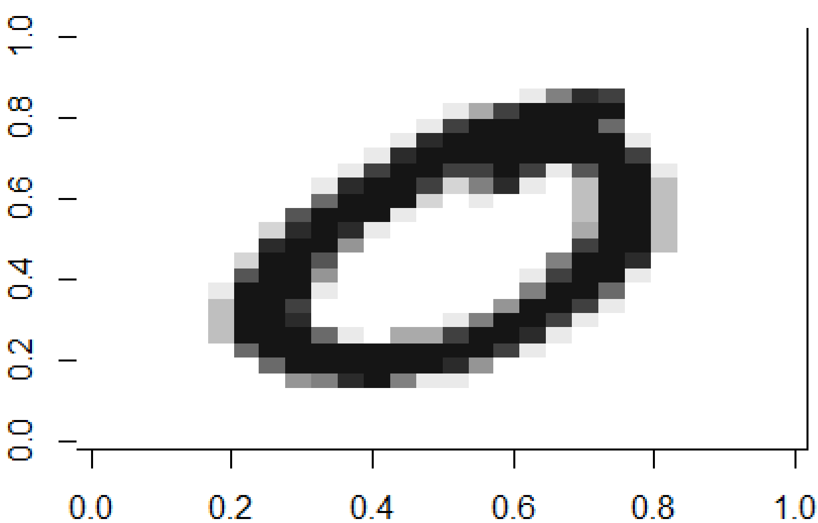
```
## Don't know how to automatically pick scale for object of type data.frame.
Defaulting to continuous.
```

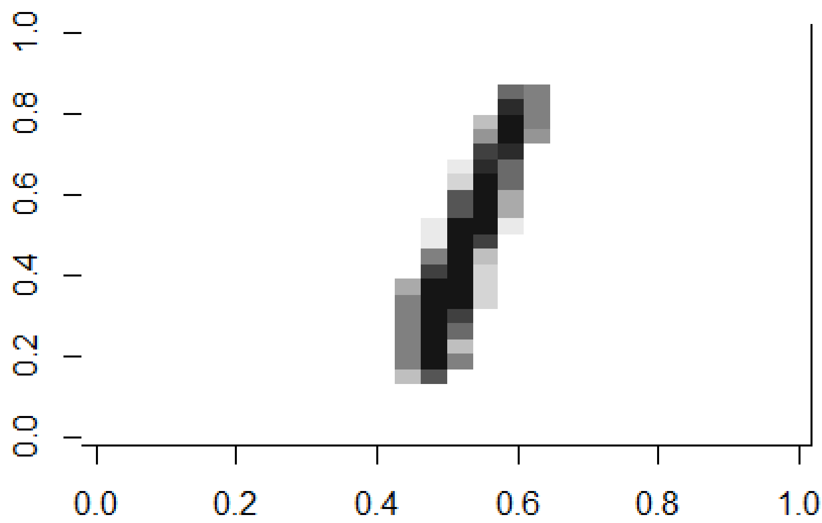


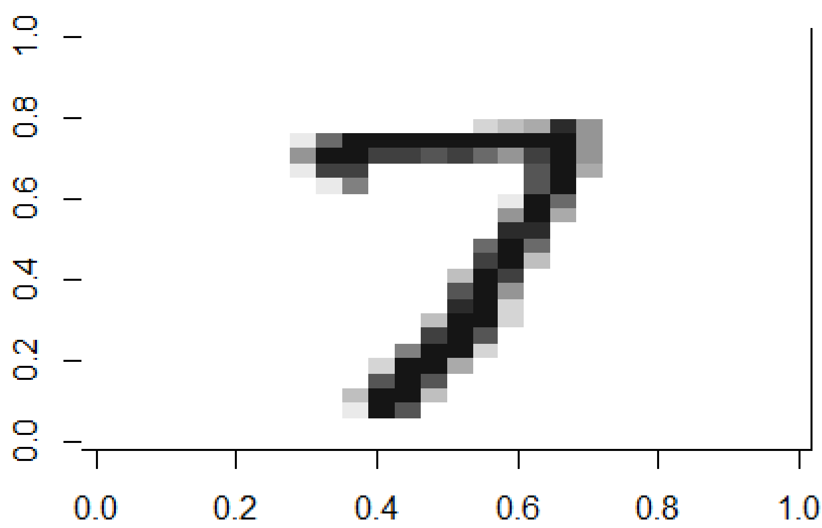
k=20

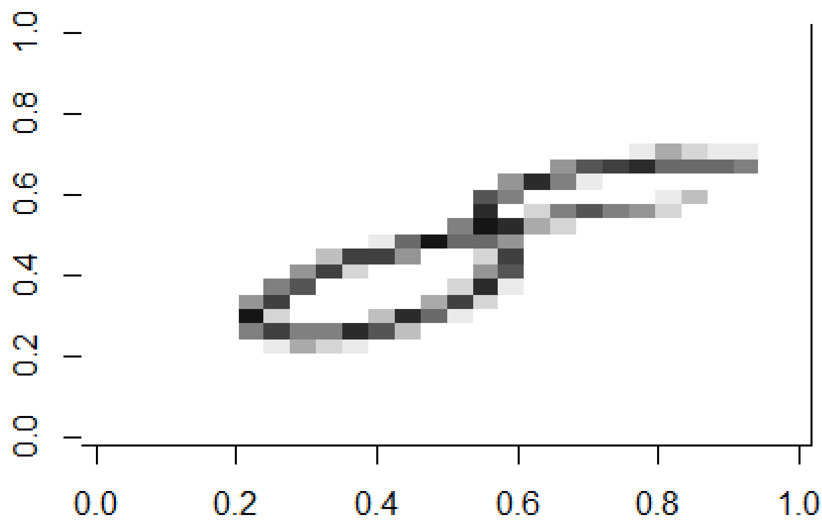
```
bonus_k_20 <- my_kmedoids(digits,20,20)
par(mfcol <- c(2,5))

## NULL

for (k in 1:10){
show_digit(bonus_k_20[[1]][k,])
}
```

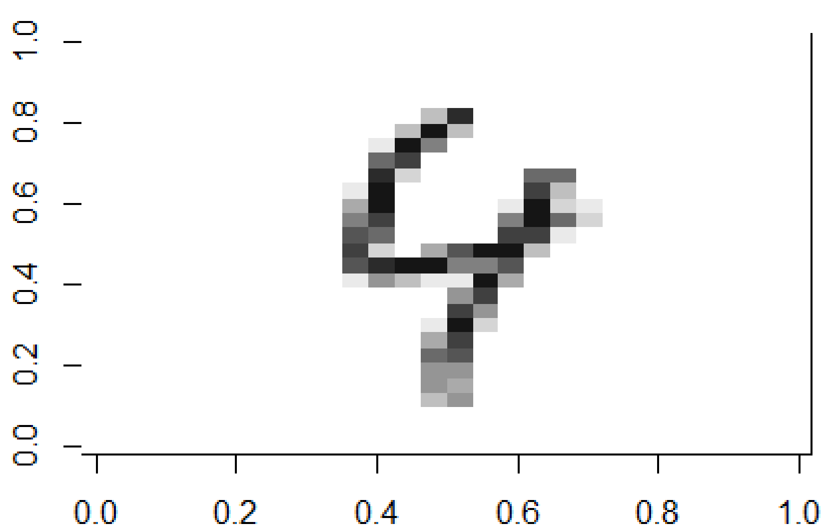

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

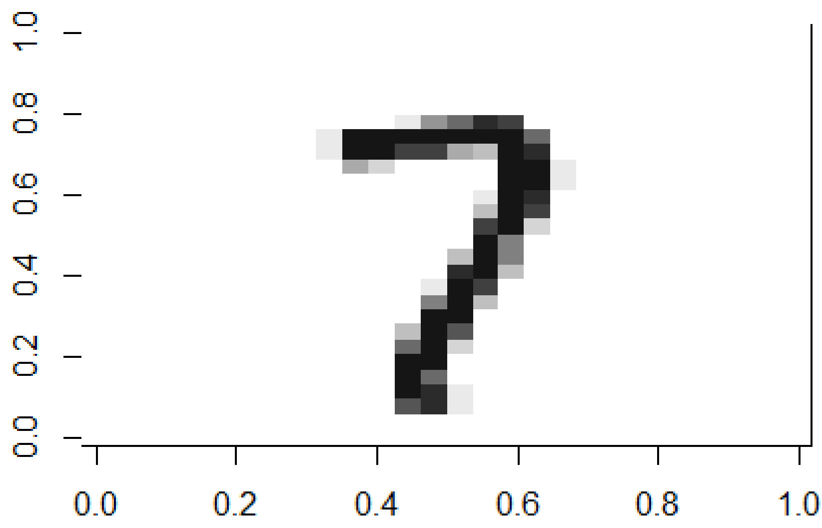

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

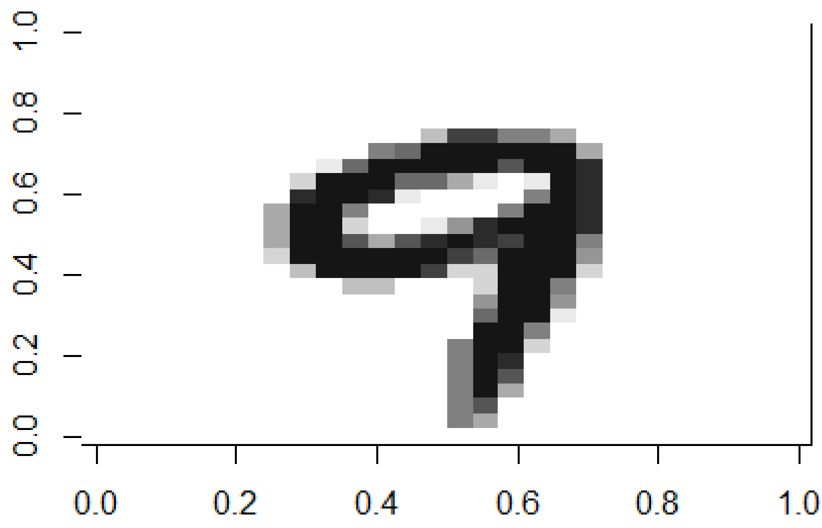

0.0
0.2
0.4
0.6
0.8
1.0

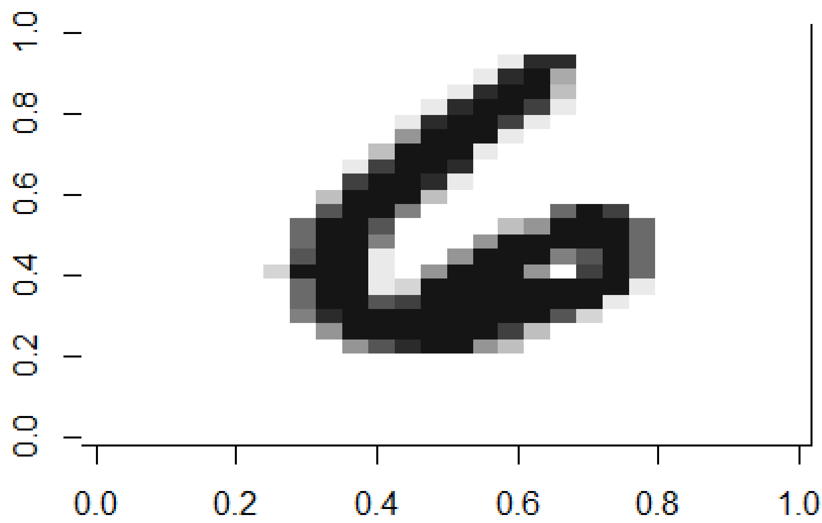

0.0
0.2
0.4
0.6
0.8
1.0

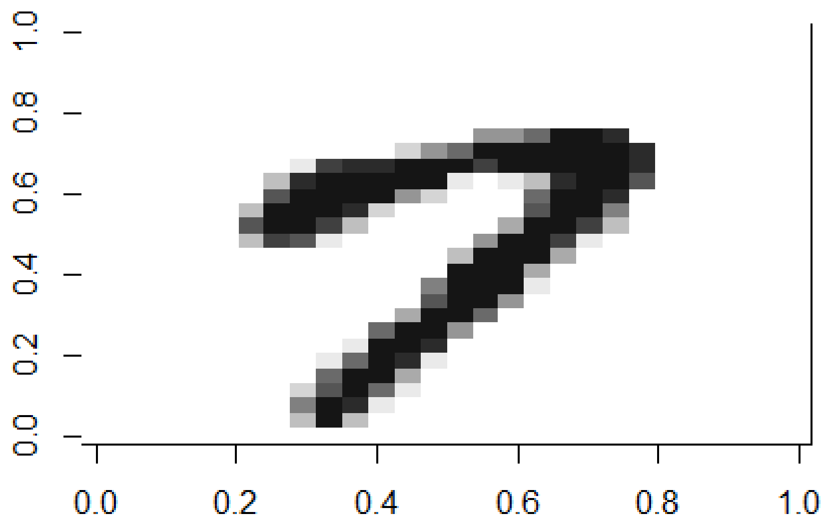

0.0
0.2
0.4
0.6
0.8
1.0


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

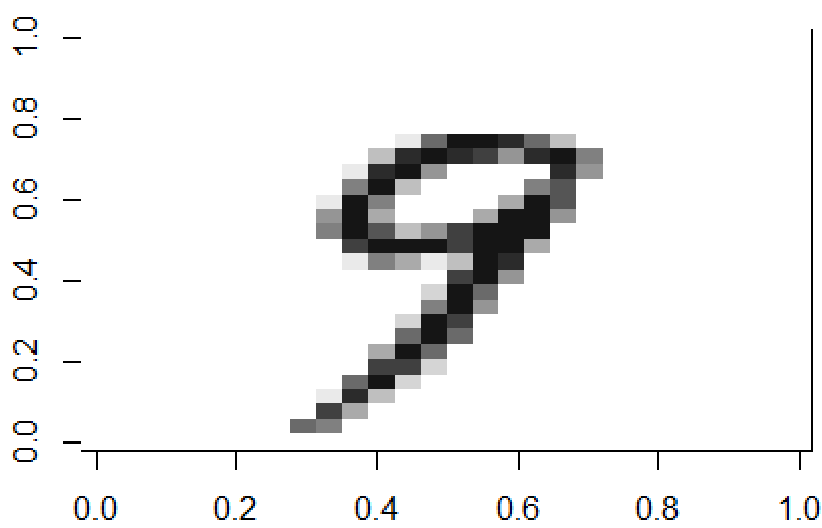

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

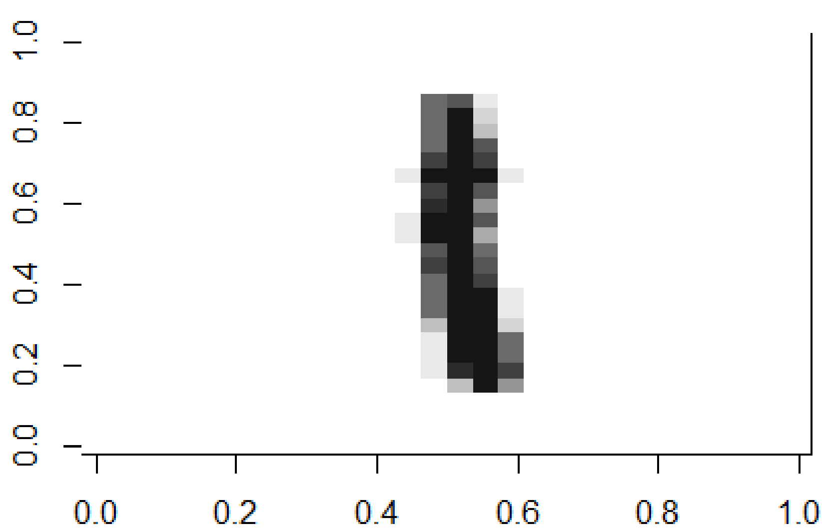


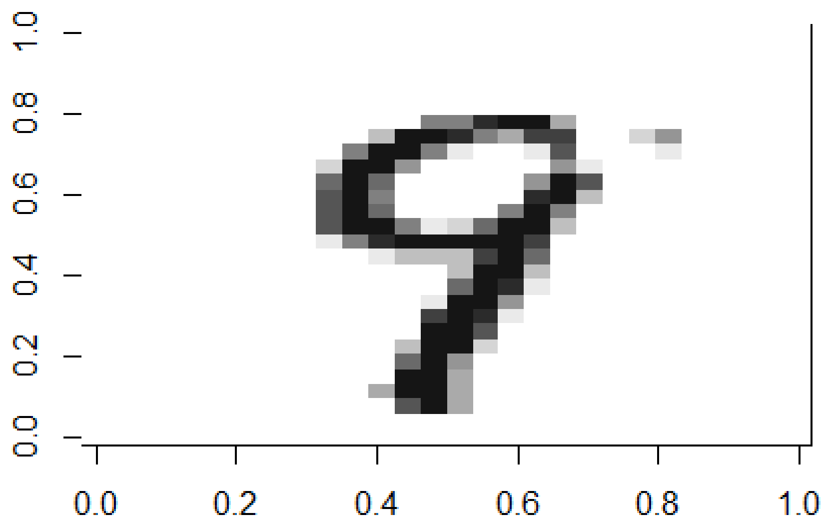


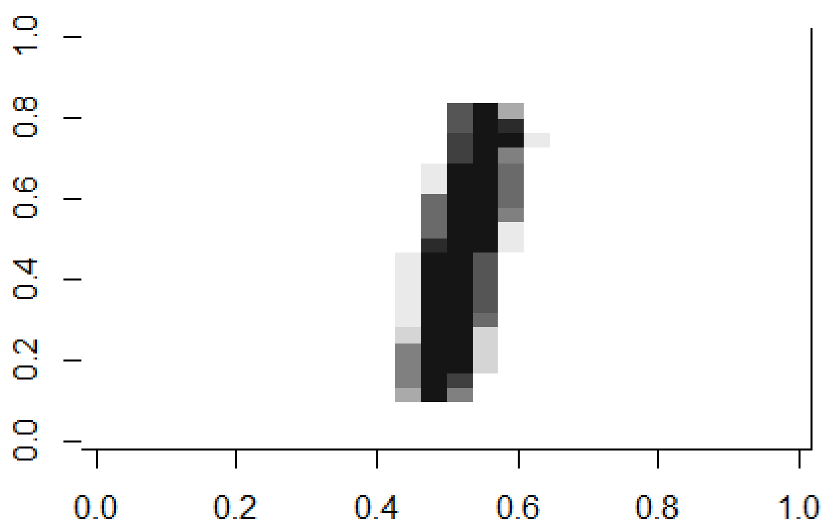

```
par(mfcol <- c(2,5))

## NULL
```

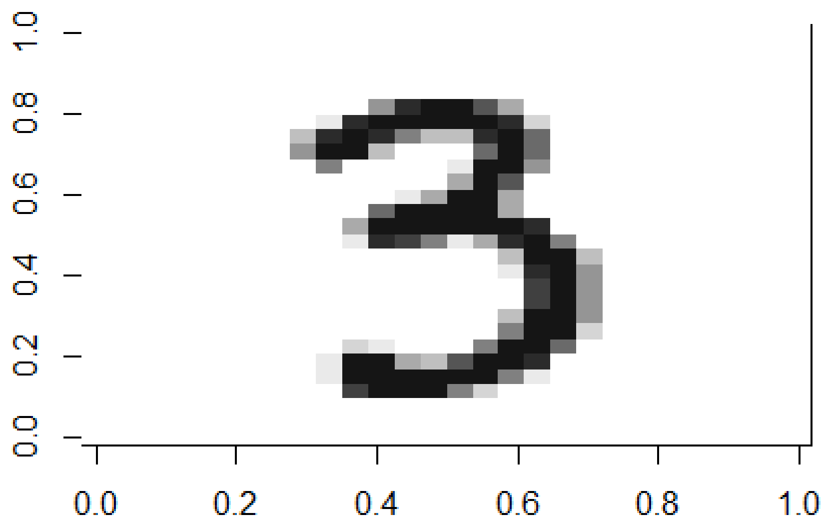
```
for (k in 11:20){
show_digit(bonus_k_20[[1]][k,])
}
```

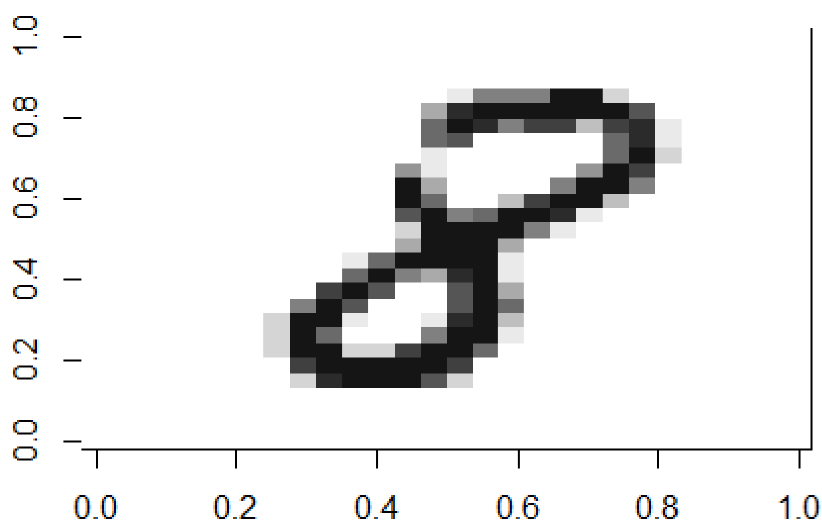


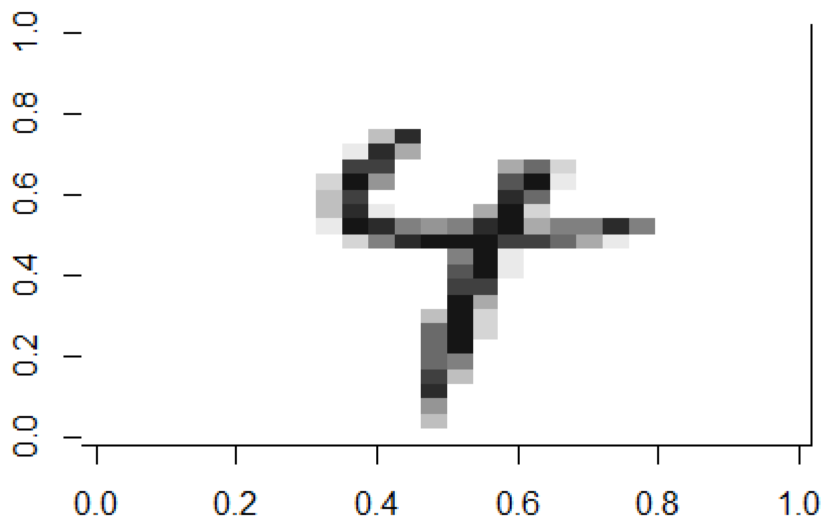



0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

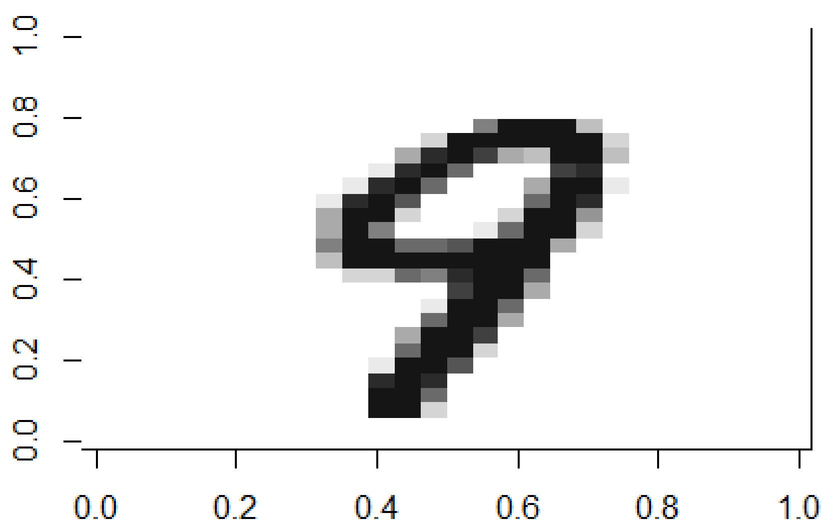

0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0

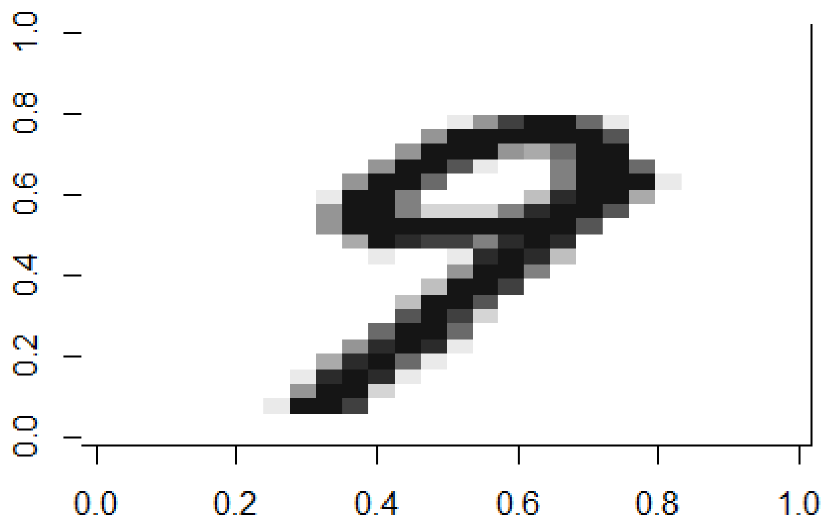

0.0
0.2
0.4
0.6
0.8
1.0

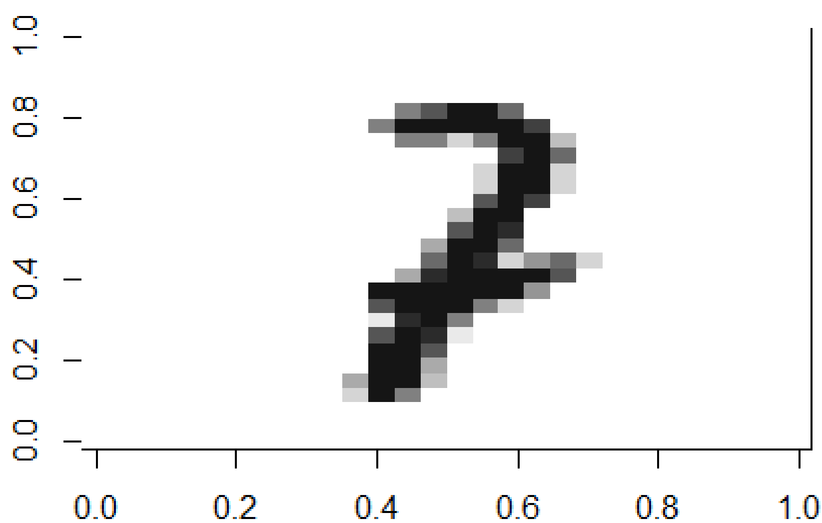


0.0
0.2
0.4
0.6
0.8
1.0



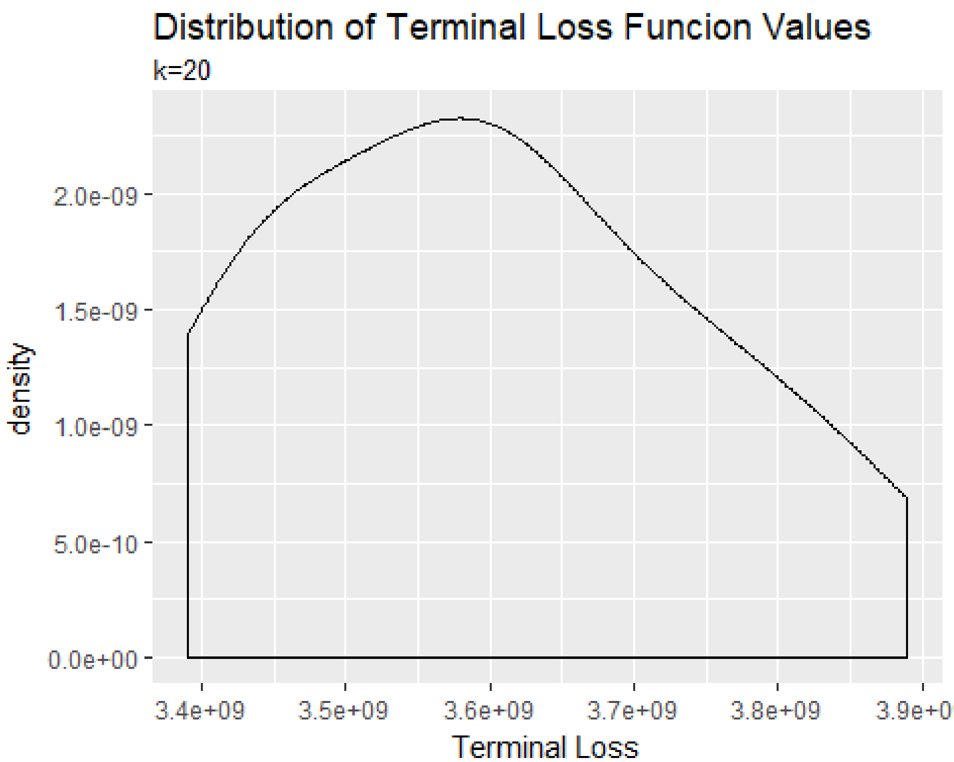
0.0
0.2
0.4
0.6
0.8
1.0






```
bonus_k_20_df <- data.frame(bonus_k_20[[4]])
ggplot(bonus_k_20_df,aes(x <- data.frame(bonus_k_20[[4]]))) +
geom_density(colour="black") + ggtitle(title <-  "Distribution of Terminal
Loss Funcion Values", subtitle <- "k=20" ) + labs(x="Terminal Loss")
```

```
## Don't know how to automatically pick scale for object of type data.frame.
Defaulting to continuous.
```



Compared with K-means, K-medoids has less variance on terminal loss, which indicates k-medoids would be less sensitive to the effect caused by initialization. However, the terminal loss of k-medoids is higher than k-means,that means k-means do a better job on the effect of clustering.

Problem2.

```
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-1.png')
```

$$= \quad \pi^1_{s_1} \prod_{t=1}^{T-1} P_Y(Y_t = y_t \mid S_t = s_t) A_{s_t, s_{t+1}} P_Y(Y_T = y_T \mid S_T = s_T)$$

No, this is not a probability vector.

No, they are not probability vectors.

```
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-3.png')
```

$$\mathrm{P}(S_t = \mathrm{i} \mid \mathrm{Y}) = \frac{\alpha_i^t \beta_i^t}{\alpha^t * \beta^t}$$

```r
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-4.png')
```

$$\mathrm{P}(S_t = \mathrm{i}, S_{t+1} = \mathrm{j} \mid \mathrm{Y}) = \frac{\alpha_i^t \beta_i^t}{\alpha^t * \beta^t} A_{ij}$$

```r
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-5.png')
```

$$\alpha^t = (\alpha^{t-1})^T diag(A)\, B^t$$

```r
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-6.png')
```

$$\beta^t = A(diag(\beta^{t+1})B^t)$$

```r
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-7.png')
```

For α^t, we start from $\alpha^1 = diag(\pi^1)\, B^1$ and then solve α^2, α^3 ..... α^T.

For β^t, we start from $\beta^t = B^T$, then we solve β^{T-1}, β^{T-2}......β^1

```r
knitr::include_graphics('c:/Users/Joey/Desktop/stat545hw3-8.png')
```

$\mathrm{O}(\mathrm{N}^2(\mathrm{T} - \mathrm{t})) + \mathrm{O}(\mathrm{N}^2\mathrm{t}) = \mathrm{O}(\mathrm{N}^2\mathrm{T})$

Because the cost of computation would be pretty high.