STAT545 HW3

Yi Yang

9/26/2018

1. Problem 1: The K-means algorithm.

The MNIST dataset is a dataset of 28×28 images of hand-written digits. Download it from http://yann.lecun.com/exdb/mnist/ (you only really need the training images and labels though). To read these images in R, use the script from https://gist.github.com/brendano/39760. Make sure you understand this. Note that the show_digit command displays a particular digit.

1. Since the dataset is quite large, restrict yourself to the first 1000 training images, and their labels. Store these as variables called digits and labels. digits should be a 1000×784 matrix (or its transpose). Include R code.

2. Write a function my_kmeans to perform a k-means clustering of the 1000 images of digits. Use Euclidean distance as your distance measure between images (which can be viewed as vectors in a 784 dimensional space). Your function should take 3 arguments, the matrix digits, the number of clusters K and the number of initializations N. You code should consist of 3 nested loops. The outermost (from 1 to N) cycles over random cluster initializations (i.e. you will call k-means N times with different initializations). The second loop (this could be a for or while loop) is the actual k-means algorithm for that initialization, and cycles over the iterations of k-means. Inside this are the actual iterations of k-means. Each iteration can have 2 successive loops from 1 to K: the first assigns observations to each cluster and the second recalculates the means of each cluster. These should not require further loops. (You will probably encounter empty clusters. It is possible to deal with these in clever ways, but here it is sufficient to assign empty clusters a random mean (just like you initialized them)). Since your initializations are random, make your results repeatable by using the set.seed() command at the beginning (you can also make the seed value a fourth argument). Your function should return:

(a) the cluster parameters and cluster assignments for the best solution
(b) the sequence of values of the loss-function over k-means iterations for the best solution (this should be non-increasing) (recall from the slides that the k-means loss function is the sum of the squared distances of observations from their assigned means)
(c) The set of N terminal loss-function values for all initializations.

Do not hardcode the number of images or their size. Include R code.

3. Explain briefly what stopping criteria you used (i.e. the details of the second loop).

4. Run your code on the 1000 digits for $K = 5, 10, 20$. Set N to a largish number e.g. 25 (if this takes too long, use a smaller number).For each setting of K, plot the cluster means (using show_digit) as well as the evolution of the loss-function for the best solution (you can use a semi-log plot if that is clearer). You do not have to print the other values returned by the function e.g. the cluster assignments, or the values of the cluster means etc., just plots is sufficient

5. For each setting of K, plot the distribution of terminal loss function values (using ggplot's geom_density()).

6. Explain briefly how you might choose the number of clusters K.

Bonus for an extra 20 points:

7. Modify your code to do k-medoids. You only need to change one part of your previous code viz. the part that calculates the cluster prototype given cluster assignments. The cleanest way to do this is to define a function called get_prototype that takes a set of observations and returns the prototype. For k-means this function just returns the mean of the observations. Note that the mean can also be defined as

$$\mu = \arg\min_x \sum_{i=1}^{|D|} (x - d_i)^2$$

Here $D = (d_1, \ldots, D_{|D|})$ is the set of images input to get_prototype, and the mean need not be part of this set. For k-medoids, the prototype is defined as

$$\mu = \arg\min_{x \in D} \sum_{i=1}^{|D|} (x - d_i)^2$$

In other words it finds an element in D that minimizes the sum-squared distance. Include R code for your implementation of get_prototype for k-mediods. You can use as many for loops as you want, but the simplest is to loop over each observation assigned to that cluster, calculate the sum-squared distance when that is set as the prototype, and return the best choice.

8. Run k-medoids for $K = 5, 10, 20$. Since this might take longer, you can use smaller values of N as well as fewer images (e.g. just 100 digits), but report what numbers you used. For each choice of K, show the cluster prototypes. Comment on the quality of the cluster prototypes, as well as the value of the loss function vs k-means.

Solution:

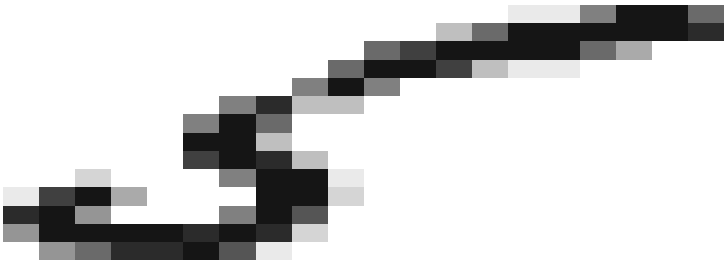
1. read the first 1000 digits and their labels

```
# Credit to https://gist.github.com/brendano/39760
# Load the MNIST digit recognition dataset into R

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
}

show_digit <- function(arr784, col=gray(12:1/12), ...) {
  image(matrix(arr784, nrow=28)[,28:1], col=col,axes=F, ...)
```

```
}

# load the first 1000 digits and their labels
digits.list <- load_image_file('./data/train-images-idx3-ubyte')
digits <- digits.list$x[1:1000,]
digits <- digits / 255.0 # normalize pixels
labels <- load_label_file('./data/train-labels-idx1-ubyte')
dim(digits)
```

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

```
# snippets to test the function of show_digits
show_digit(digits[12,])
```



```
labels[12]
```

```
## [1] 5
```

```
class(labels[12])
```

```
## [1] "integer"
```

2. K-means function is given by

```
# define helper functions
# get distance between a single digit and all the centroids of clusters
getDist <- function(single.digits, centroids){
  dist <- colSums((t(centroids) - single.digits)^2)
  return(dist)
}
# update the centroid for a single cluster by calculating the mean of digits in this cluster
single.centroid <- function(ric, all.digits){
  num.digits.in.cluster <- sum(ric)
  if (num.digits.in.cluster > 0){
    return((ric %*% all.digits) / num.digits.in.cluster )
  }
  else{
    return(all.digits[sample(nrow(all.digits),1),])
  }
}
# K-means algorithm starts here
# K is num of clusters, N is num of different initializations
my_kmeans <- function(all.digits, K, N){
  set.seed(16)
  best.cluster.parameters <- NULL # centroids
  best.cluster.assignments <- NULL # ric
  best.loss.seq <- c()
  terminal.losses <- c()
```

```
  best.terminal.loss <- Inf
  for (i in seq(1,N)){
    centroids <- all.digits[sample(nrow(all.digits),K), ]
    dists <- apply(all.digits,1,getDist,centroids=centroids)
    dists <- t(dists)
    loss <- sum(apply(dists, 1, min))
    last.loss <- loss + 1
    loss.seq <- c(loss)
    while (abs(last.loss - loss) >= 1e-4) {
      rics <- apply(dists,1,function(d) ifelse(d==min(d),1,0))
      rics <- t(rics)
      centroids <- apply(rics, 2, single.centroid, all.digits=all.digits)
      dists <- apply(all.digits, 1, getDist, centroids=t(centroids))
      dists <- t(dists)
      last.loss <- loss
      loss <- sum(apply(dists, 1, min))
      loss.seq <- c(loss.seq,loss)
    }
    terminal.losses <- c(terminal.losses,loss)
    if (best.terminal.loss > loss){
      best.terminal.loss <- loss
      best.loss.seq <- loss.seq
      best.cluster.assignments <- rics
      best.cluster.parameters <- t(centroids)
    }
  }
  return(list(best.cluster.parameters=best.cluster.parameters, best.cluster.assignments=best.cluster.as
}
```

3. The stopping criterion is set to be that the difference of the loss function in the last iteration and the following iteration is smaller than 1×10^{-4}.
4. The cluster means are plotted below, as well as the best loss sequence.

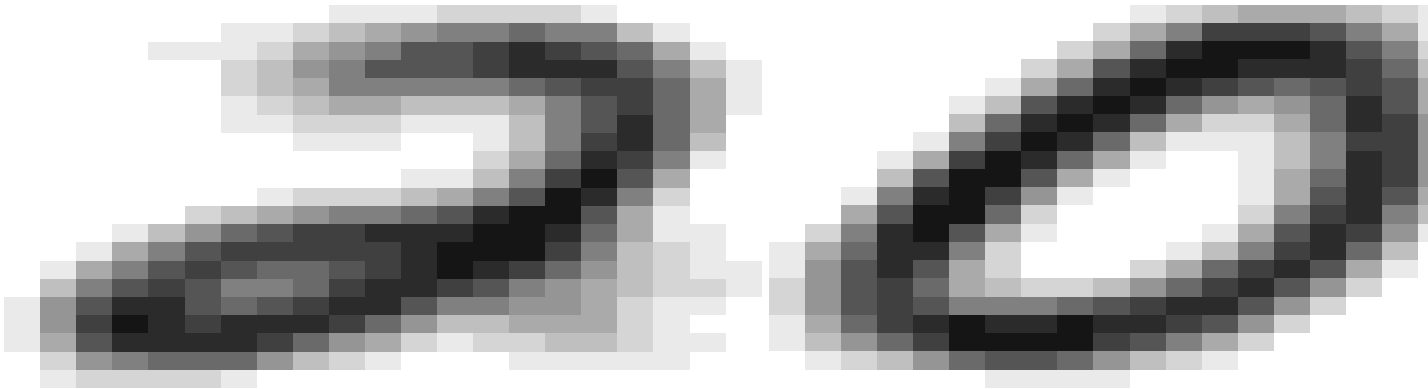
```
K <- c(5,10,20)
N <- 15
results <- list()
for (k in K){
  results <- append(results, list(my_kmeans(digits,k,N)))
}
```

- Case $K = 5$

```
for (id in seq(1,5)){
  show_digit(results[[1]]$best.cluster.parameters[id,])
}
```

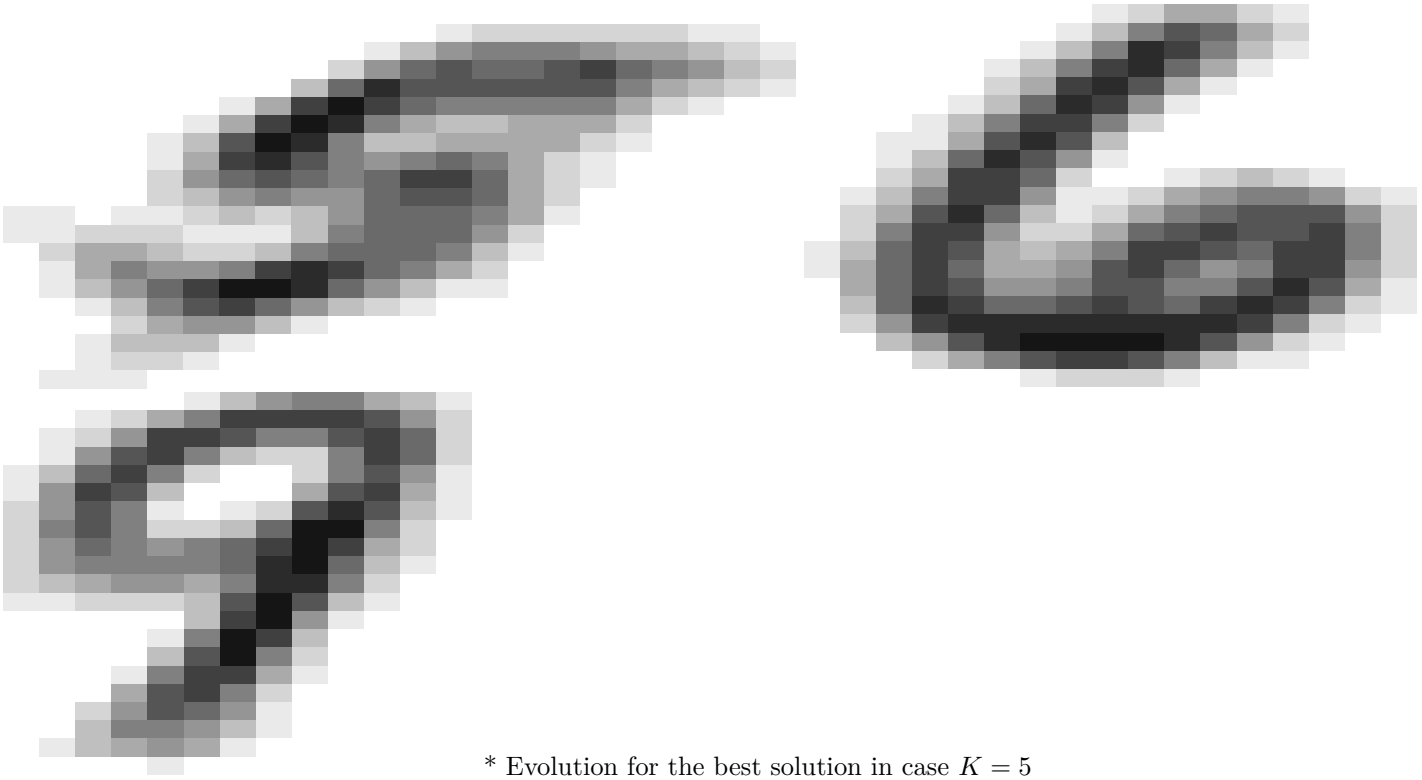
* Case $K = 10$

```
for (id in seq(1,10)){
  show_digit(results[[2]]$best.cluster.parameters[id,])
}
```



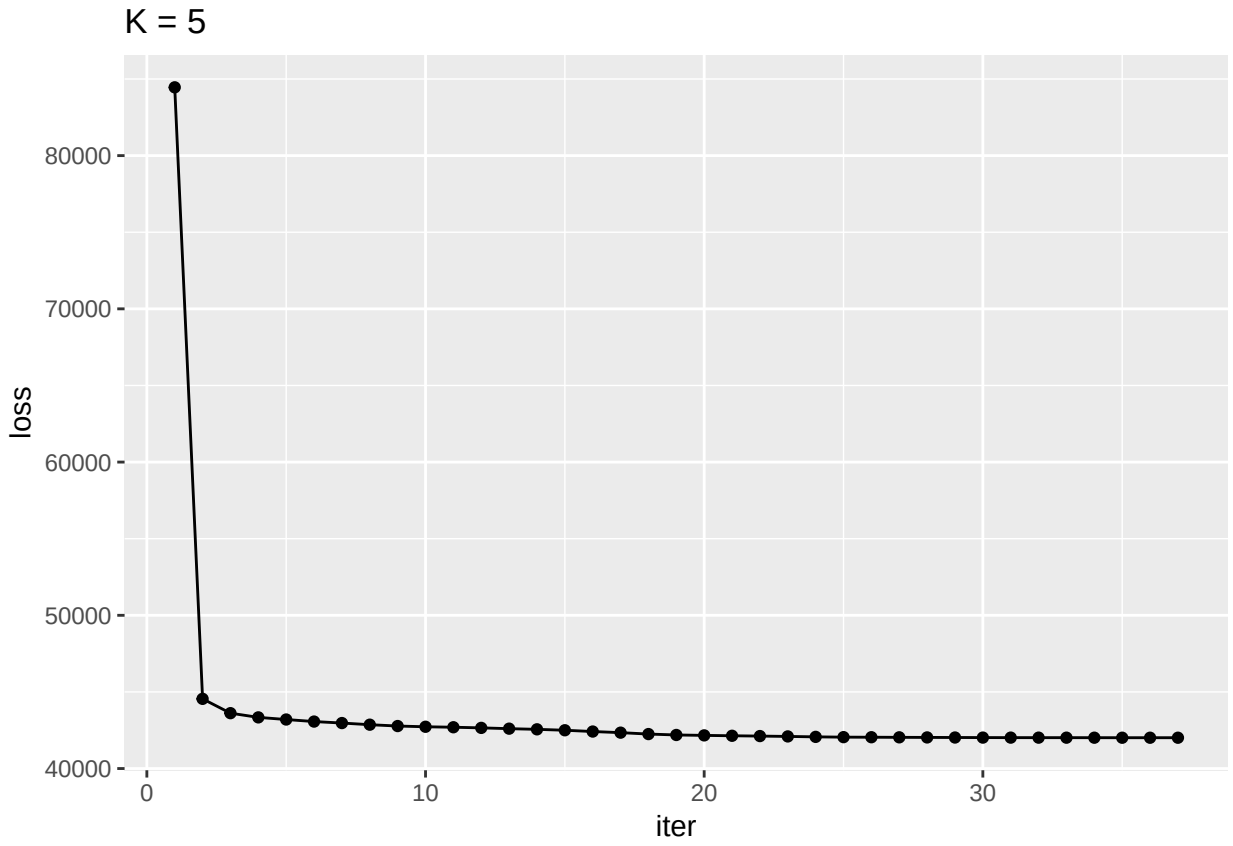
* Case $K = 20$

```
for (id in seq(1,20)){
  show_digit(results[[3]]$best.cluster.parameters[id,])
}
```



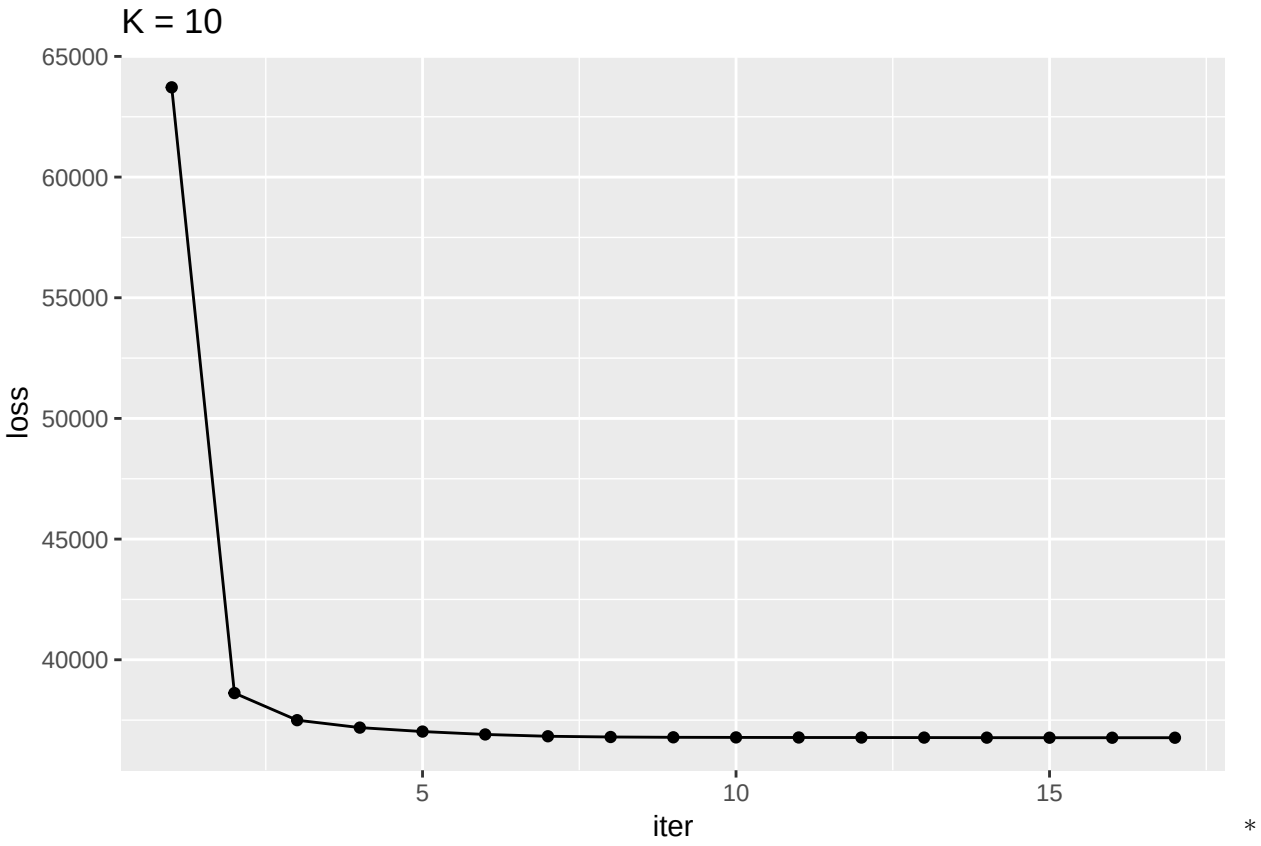
* Evolution for the best solution in case $K = 5$

```
library(ggplot2)
best.loss.seq.df <- data.frame(loss=results[[1]]$best.loss.seq,iter=seq(1,length(results[[1]]$best.loss
ggplot(best.loss.seq.df,aes(x=iter,y=loss)) +
  geom_line() + geom_point() + ggtitle(label = 'K = 5')
```



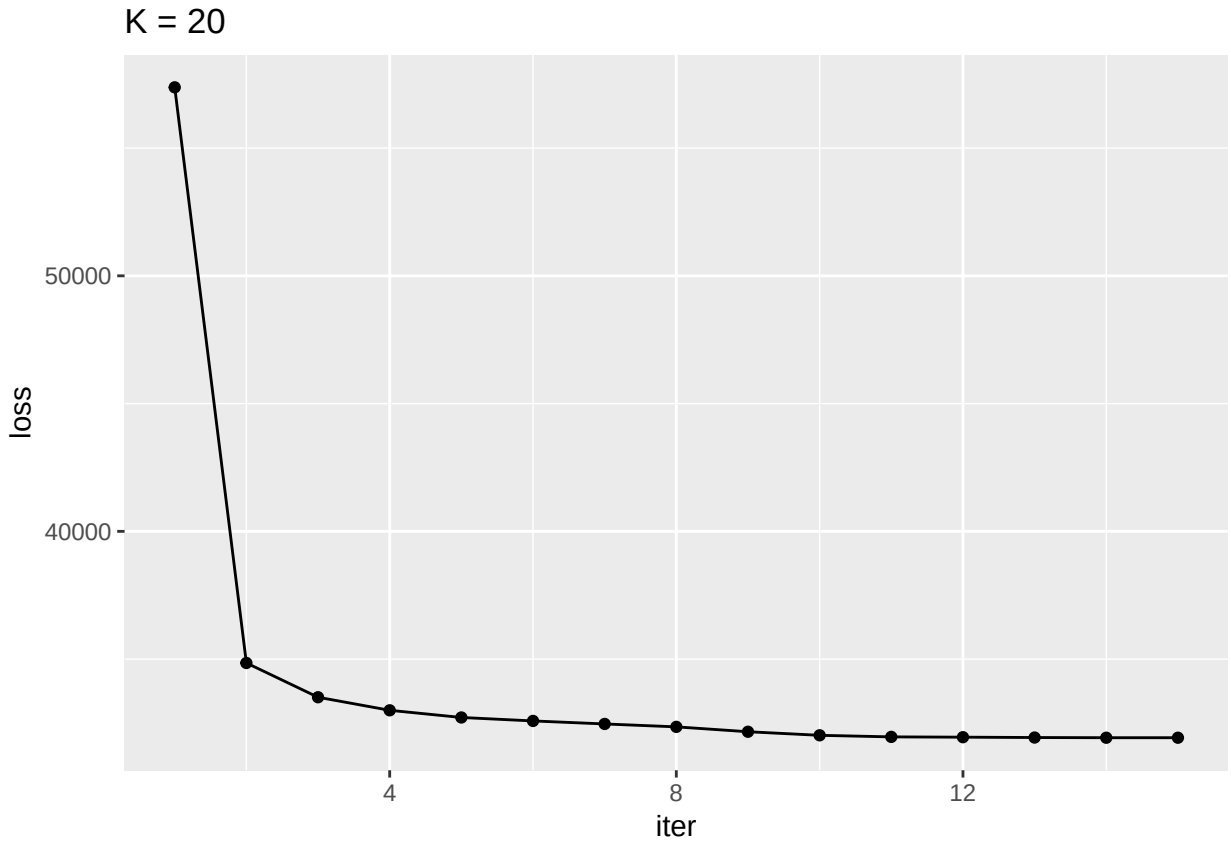
Evolution for the best solution in case $K = 10$

```
best.loss.seq.df <- data.frame(loss=results[[2]]$best.loss.seq,iter=seq(1,length(results[[2]]$best.loss
ggplot(best.loss.seq.df,aes(x=iter,y=loss)) +
  geom_line() + geom_point() + ggtitle(label = 'K = 10')
```



Evolution for the best solution in case $K = 20$

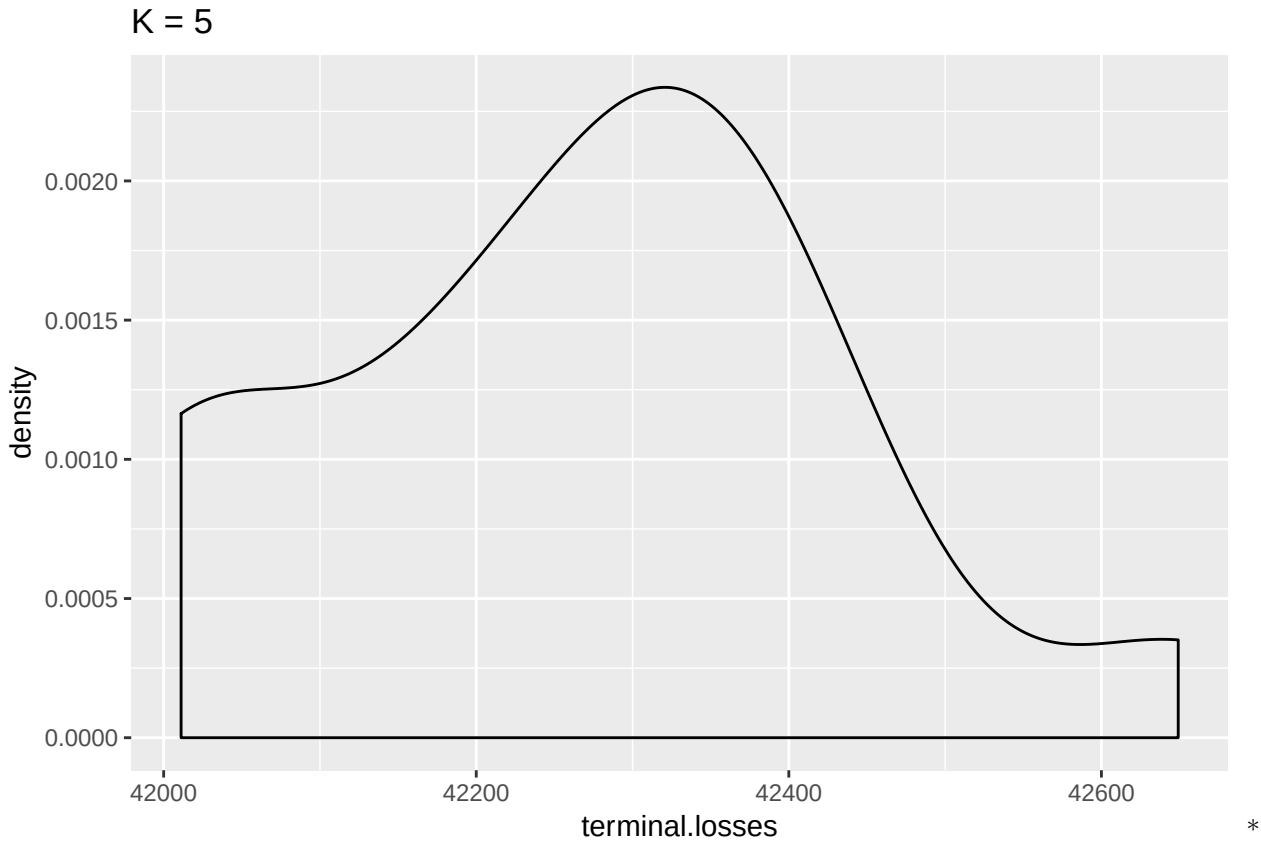
```
best.loss.seq.df <- data.frame(loss=results[[3]]$best.loss.seq,iter=seq(1,length(results[[3]]$best.loss
ggplot(best.loss.seq.df,aes(x=iter,y=loss)) +
  geom_line() + geom_point() + ggtitle(label = 'K = 20')
```



5. The distribution of terminal loss function values are given by

- Case $K = 5$

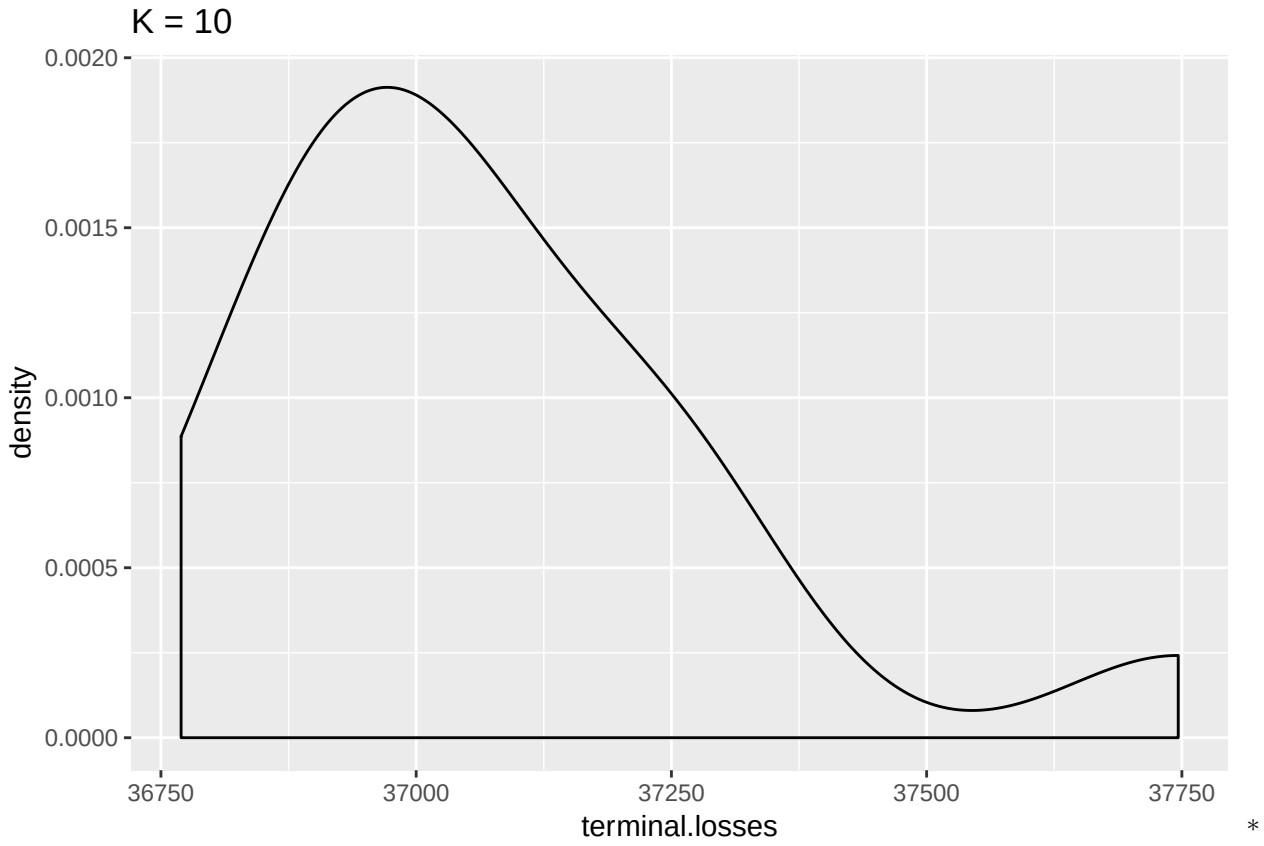
```
terminal.losses.df <- data.frame(terminal.losses=results[[1]]$terminal.losses)
ggplot(terminal.losses.df,aes(x=terminal.losses)) +
  geom_density() + ggtitle(label = 'K = 5')
```



*

Case $K = 10$

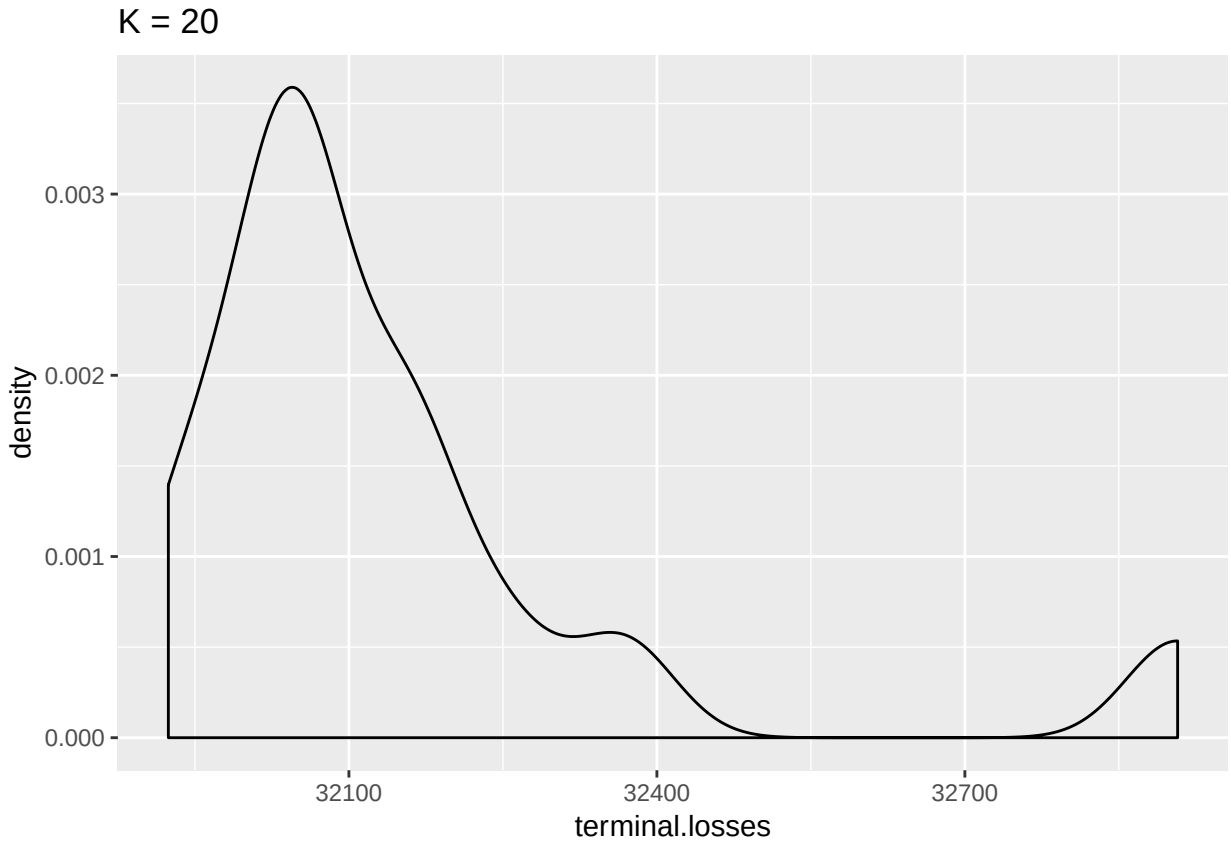
```
terminal.losses.df <- data.frame(terminal.losses=results[[2]]$terminal.losses)
ggplot(terminal.losses.df,aes(x=terminal.losses)) +
  geom_density() + ggtitle(label = 'K = 10')
```



*

Case $K = 20$

```
terminal.losses.df <- data.frame(terminal.losses=results[[3]]$terminal.losses)
ggplot(terminal.losses.df,aes(x=terminal.losses)) +
  geom_density() + ggtitle(label = 'K = 20')
```



6. Plot best terminal losses with a set of different K values, we show the curve will look like an elbow. Picking the K value around the elbow point is appropriate.

Bonus for an extra 20 points:

7. For K-medoids method, we only need to modify function single.centroid.

```
# get_prototype function is a substitute to function single.centroid
get_prototype <- function(ric, all.digits){
  digits.in.cluster <- all.digits[ric==1,]
  smallest.dist <- Inf
  closest.digit <- digits.in.cluster[1,]
  if (nrow(digits.in.cluster) == 0){
    return(all.digits[sample(nrow(all.digits),1), ])
  }
  for (i in seq(1,nrow(digits.in.cluster))){
    digit <- digits.in.cluster[i, ]
    dist.to.others <- sum(getDist(digit,digits.in.cluster))
    if (dist.to.others < smallest.dist){
      closest.digit <- digit
      smallest.dist <- dist.to.others
    }
  }
  return(closest.digit)
}
# define function my_kmedoids
my_kmedoids <- function(all.digits, K, N){
```

```
  set.seed(26)
  best.cluster.parameters <- NULL # centroids
  best.cluster.assignments <- NULL # ric
  best.loss.seq <- c()
  terminal.losses <- c()
  best.terminal.loss <- Inf
  for (i in seq(1,N)){
    centroids <- all.digits[sample(nrow(all.digits),K), ]
    dists <- apply(all.digits,1,getDist,centroids=centroids)
    dists <- t(dists)
    loss <- sum(apply(dists, 1, min))
    last.loss <- loss + 1
    loss.seq <- c(loss)
    while (abs(last.loss - loss) >= 1e-4) {
      rics <- apply(dists,1,function(d) ifelse(d==min(d),1,0))
      rics <- t(rics)
      centroids <- apply(rics, 2, get_prototype, all.digits=all.digits)
      dists <- apply(all.digits, 1, getDist, centroids=t(centroids))
      dists <- t(dists)
      last.loss <- loss
      loss <- sum(apply(dists, 1, min))
      loss.seq <- c(loss.seq,loss)
    }
    terminal.losses <- c(terminal.losses,loss)
    if (best.terminal.loss > loss){
      best.terminal.loss <- loss
      best.loss.seq <- loss.seq
      best.cluster.assignments <- rics
      best.cluster.parameters <- t(centroids)
    }
  }
  return(list(best.cluster.parameters=best.cluster.parameters, best.cluster.assignments=best.cluster.as
}
```

8. Let us take $N = 10$ and first 100 digits. The cluster prototypes are clearer than those obtained from K-means method. The loss function values are relatively higher than K-means method.

```
digits.smaller <- digits[1:100, ]
K <- c(5,10,20)
N <- 10
results <- list()
for (k in K){
  results <- append(results, list(my_kmedoids(digits,k,N)))
}
```

- Case $K = 5$

```
for (id in seq(1,5)){
  show_digit(results[[1]]$best.cluster.parameters[id,])
}
```

* Case $K = 10$

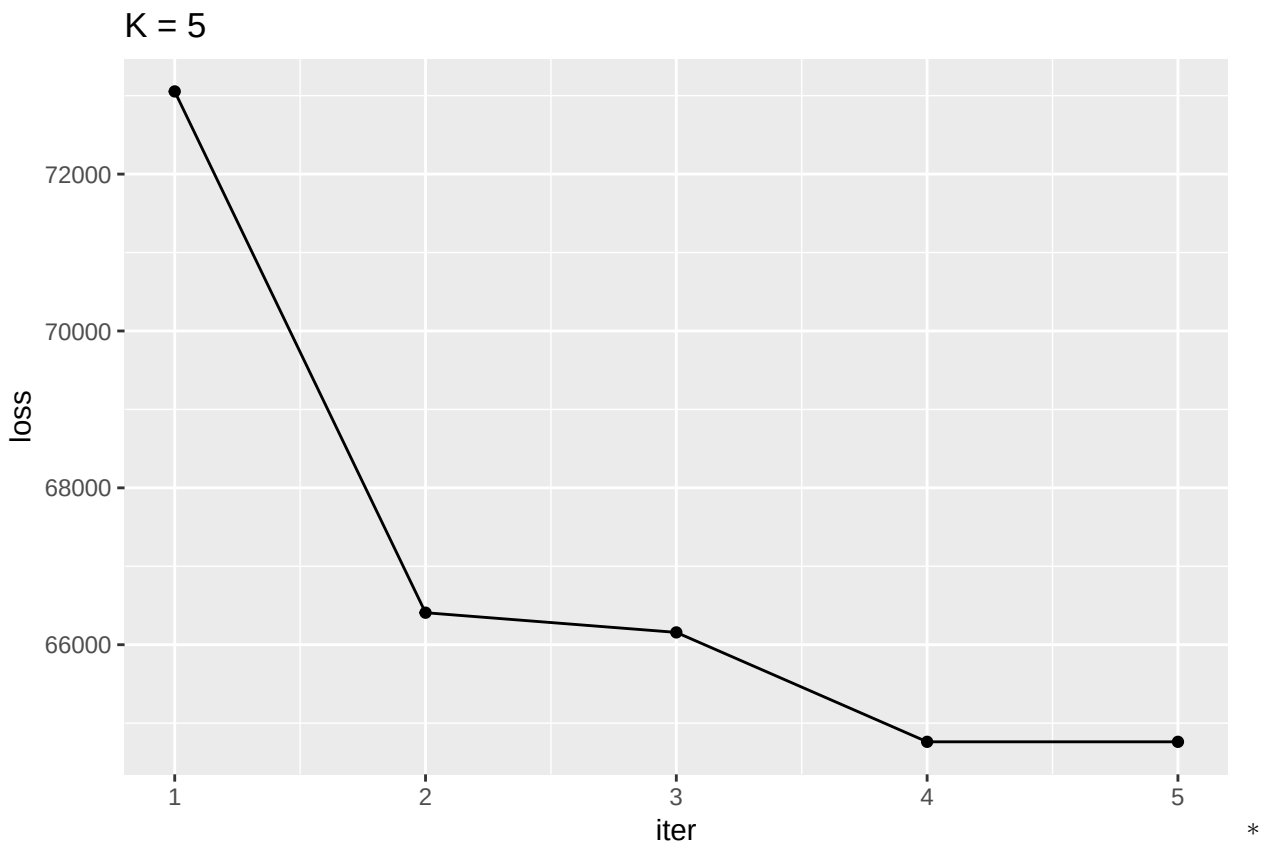
```
for (id in seq(1,10)){
  show_digit(results[[2]]$best.cluster.parameters[id,])
}
```

* Case $K = 20$

```
for (id in seq(1,20)){
  show_digit(results[[3]]$best.cluster.parameters[id,])
}
```

* Evolution for the best solution in case $K = 5$

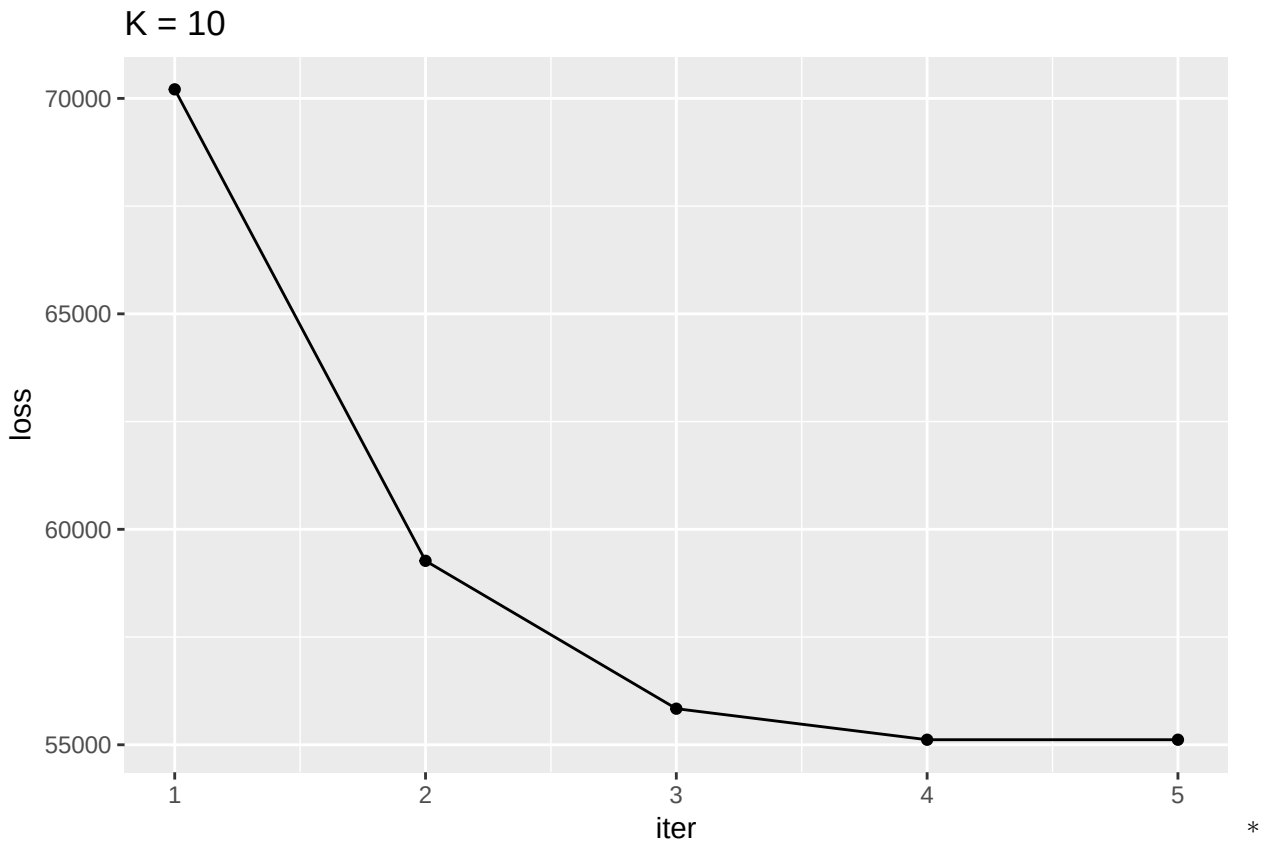
```
best.loss.seq.df <- data.frame(loss=results[[1]]$best.loss.seq,iter=seq(1,length(results[[1]]$best.loss
ggplot(best.loss.seq.df,aes(x=iter,y=loss)) +
  geom_line() + geom_point() + ggtitle(label = 'K = 5')
```



*

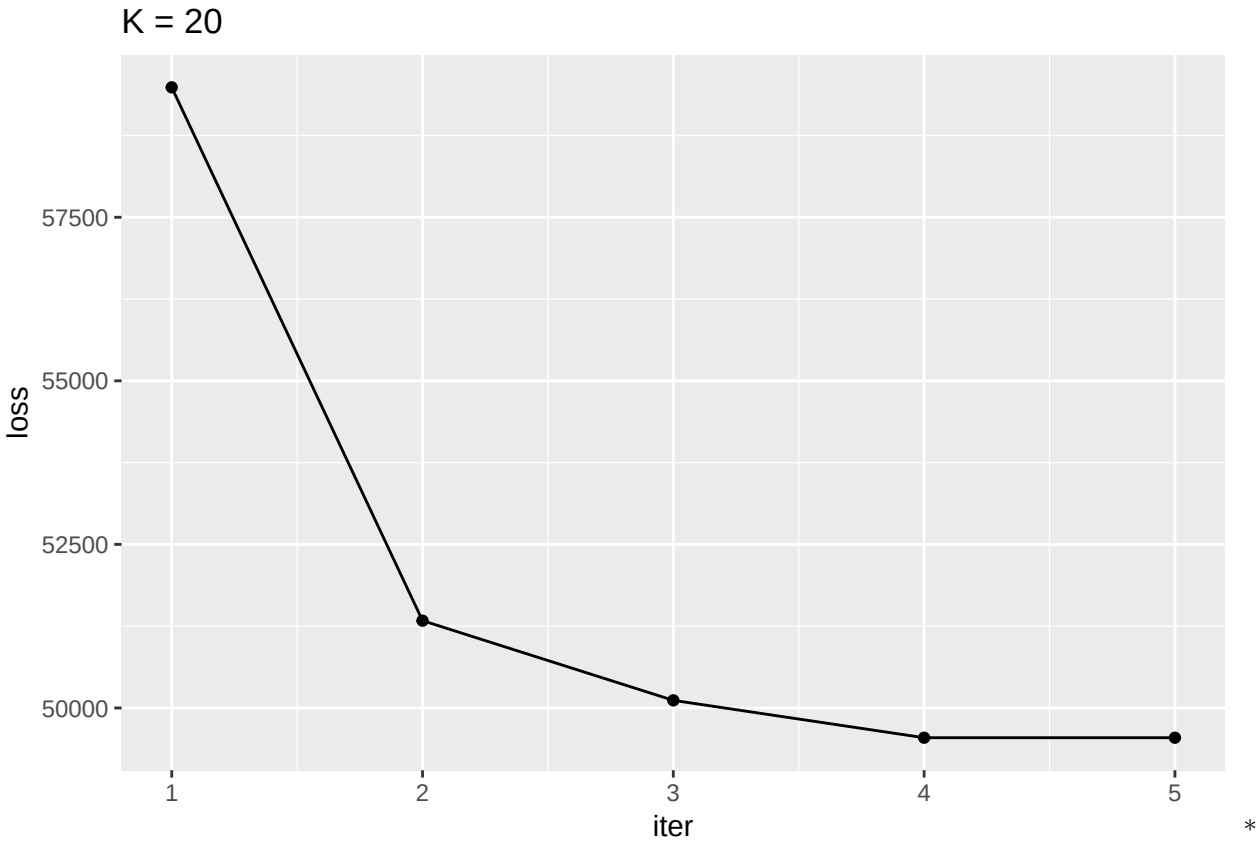
Evolution for the best solution in case $K = 10$

```
best.loss.seq.df <- data.frame(loss=results[[2]]$best.loss.seq,iter=seq(1,length(results[[2]]$best.loss
ggplot(best.loss.seq.df,aes(x=iter,y=loss)) +
  geom_line() + geom_point() + ggtitle(label = 'K = 10')
```



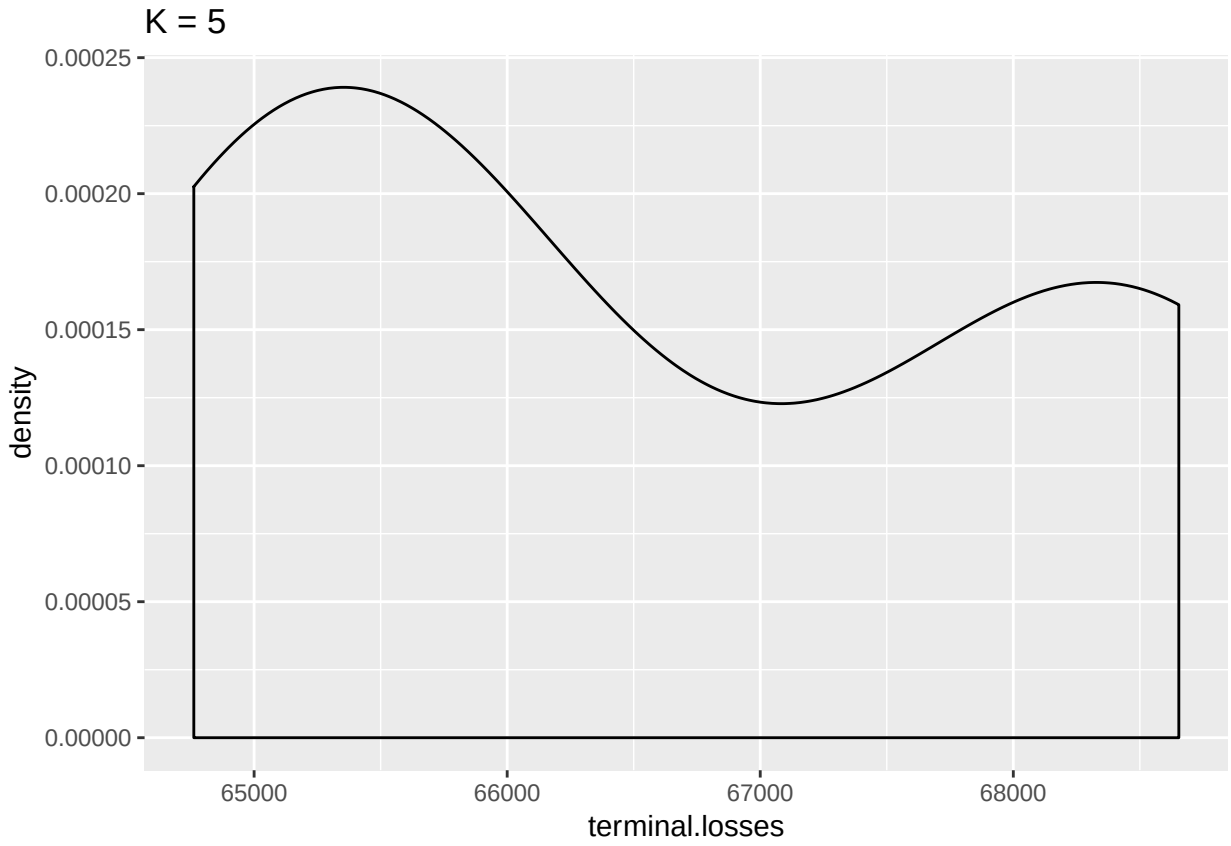
Evolution for the best solution in case $K = 20$

```
best.loss.seq.df <- data.frame(loss=results[[3]]$best.loss.seq,iter=seq(1,length(results[[3]]$best.loss
ggplot(best.loss.seq.df,aes(x=iter,y=loss)) +
  geom_line() + geom_point() + ggtitle(label = 'K = 20')
```



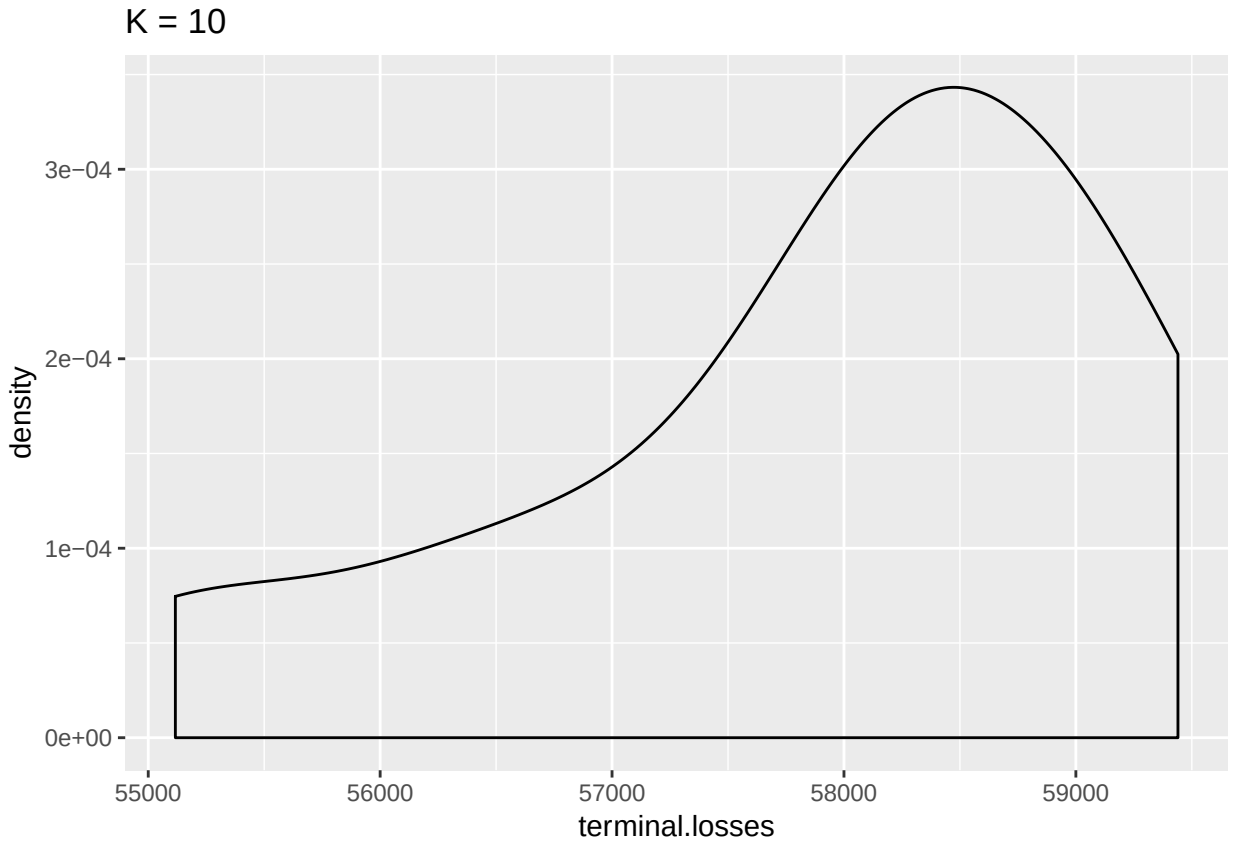
distribution of terminal loss function values in case $K = 5$

```
terminal.losses.df <- data.frame(terminal.losses=results[[1]]$terminal.losses)
ggplot(terminal.losses.df,aes(x=terminal.losses)) +
  geom_density() + ggtitle(label = 'K = 5')
```



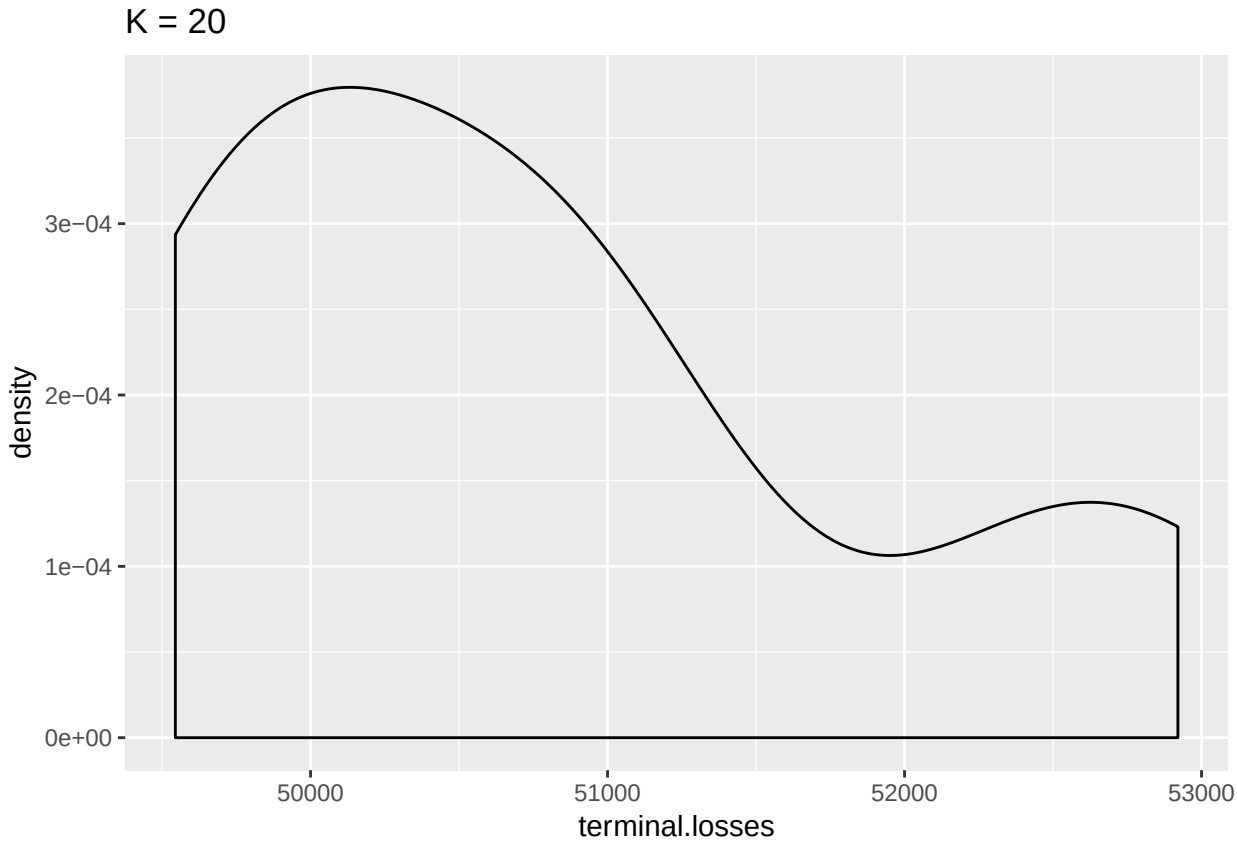
distribution of terminal loss function values in case $K = 10$

```
terminal.losses.df <- data.frame(terminal.losses=results[[2]]$terminal.losses)
ggplot(terminal.losses.df,aes(x=terminal.losses)) +
  geom_density() + ggtitle(label = 'K = 10')
```



distribution of terminal loss function values in case $K = 20$

```
terminal.losses.df <- data.frame(terminal.losses=results[[3]]$terminal.losses)
ggplot(terminal.losses.df,aes(x=terminal.losses)) +
  geom_density() + ggtitle(label = 'K = 20')
```



2. Problem 2: Finite-state Hidden Markov models (HMMs)

(Continued from the problem on Markov chains from the previous homework.)

Suppose now that we do not observe the state S_t of the Markov chain. Instead, at time t we observe Y_t. Y_t can be anything: integers, reals, vectors, images. The only condition is that the probability distribution of Y_t depends only on S_t (and not e.g. on S_{t-1}). Write this as $P_Y(Y_t|S_t)$, with $P_Y(\cdot|S_t = i)$ giving the probability (or probability density) of Y_t given $S_t = i$ (for simplicity, we let this same probability hold for all t). Also write $\mathbf{Y} = (Y_1, ...Y_T)$.

Our HMM model defines a probability distribution over $(\mathbf{S}, \mathbf{Y})$.

1. Write down $P(S_1 = s_1, S_2 = s_2, \ldots, S_T = s_T, Y_1 = y_1, \ldots, Y_T = y_t)$ in terms of π^1, A and P_Y.

For the earlier 'Markov chain' problem, we could quite efficiently calculate $\pi_i^t = P(S_t = i)$, the marginal prior probability of S_t. We will now calculate the marginal posterior probabilities $P(S_t = i|\mathbf{Y})$. Look at the scanned notes for the Kalman filter for reference.

From now onwards, we will fix the values of $\mathbf{Y}$, since these are our observations. Then define an $N \times 1$ vector B^t, with $B_i^t = P(Y_t = y_t|S_t = i)$.

1. Is this a *probability* vector (i.e. is it a nonnegative, adding up to 1)?

Define α and β messages:

$$\alpha_i^t := P(S_t = i, Y_1 = y_1, \ldots, Y_t = y_t) = P(S_t = i, \mathbf{Y}_{1:t})$$

$$\beta_i^t := P(Y_{t+1} = y_{t+1}, \ldots, Y_T = y_T|S_t = i) = P(\mathbf{Y}_{t+1:T}|S_t = i)$$

Above := means 'which we define to be equal to', or 'which we will call'.

2. α^t and β^t are both $N \times 1$ vectors. Are these vectors *probability* vectors?

3. Write $P(S_t = i|\mathbf{Y})$ in terms of α^t and β^t. Include the normalization term (i.e summation of both sides over i must give 1). In the scanned notes (e.g. top paragraph of page 2), we ignore the normalization constant, but we can easily calculate it since we know that probabilities must sum or integrate to 1. Use matrix (or vector) notation. You might have to use a transpose (e.g. $(\alpha^t)^T$). Also, define $\mathbf{1}$ as an $N \times 1$ vector of ones, and note that $\sum_{i=1}^N a_i^t = \mathbf{1}^T a^t$. Hint: First write it explicitly with summations (compare with the Kalman filter).

4. Write $P(S_t = i, S_{t+1} = j|\mathbf{Y})$ in terms of α^t, β^t, A and B. Include the normalization term and use matrix (or vector) notation.

5. Write α^t as a function of α^{t-1}, A and B. Use matrix notation. An operation you'll need here is the element-wise product of two vectors. In R this is easy, just write $V_1 * V_2$ for two vectors V_1 and V_2. In matrix notation, the simplest way to write this is as $diag(V_2) \cdot V_1$, where $diag(V_2)$ is an $N \times N$ matrix whose diagonal is V_2 and whose other elements are 0. Verify that $diag(V_2) \cdot V_1$ is a vector whose ith element is the product of the ith elements of V_1 and V_2.

6. Write β^t as a function of β^{t+1}, A and B. Use matrix notation.

7. How will you calculate the first α and β at the beginning of the forward and backward pass?

8. Hopefully, you can now see a dynamic programming algorithm that sequentially calculates the $\alpha^{t\prime}$s, and then the $\beta^{t\prime}$s. These can then be combined to calculate $P(S_t|\mathbf{Y})$ for any t. The overall algorithm is called the Baum-Welch algorithm. Write down the cost in terms of T and N.

9. Imagine instead we wanted to calculate the most like sequence of states (rather than the marginal probabilities). One approach is to set $S_t = \arg\max P(S_t|\mathbf{Y})$ for all t. Why is this a bad idea?

Solution:

1. The joint probability is given by

$$\begin{aligned}
P(S_1 = s_1, \ldots, S_T = s_T, Y_1 = y_1, \ldots, Y_T = y_t) &= P(Y_1 = y_1, \ldots, Y_T = y_T|S_1 = s_1, \ldots, S_T = s_T)P(S_1 = s_1, \ldots, S_T = s \\
&= \prod_{t=1}^T P_Y(Y_t = y_t|S_t = s_t) \cdot P(S_1 = s_1, \ldots, S_T = s_T) \\
&= \prod_{t=1}^T P_Y(Y_t = y_t|S_t = s_t) \cdot P(S_1 = s_1) \cdot \prod_{i=2}^T P(S_i = s_i|S_{i-1} = s_{i-1}) \\
&= \prod_{t=1}^T P_Y(Y_t = y_t|S_t = s_t) \cdot \pi_{s_1}^1 \cdot \prod_{i=2}^T A_{s_{i-1}, s_i}
\end{aligned}$$

2. No, B^t is not a probability vector.

3. α^t is not a probability vector since $\sum_i \alpha_i^t = P(\mathbf{Y}_{1:t})$. β^t is not a probability vector.

4. The probability is given by

$$\begin{aligned}
P(S_t = i|\mathbf{Y}) &= \frac{P(S_t = i, \mathbf{Y})}{P(\mathbf{Y})} \\
&= \frac{P(S_t = i, \mathbf{Y}_{1:t})P(\mathbf{Y}_{t+1:T}|S_t = i)}{(\alpha^t)^\top \beta^t} \\
&= \frac{\alpha_i^t \beta_i^t}{(\alpha^t)^\top \beta^t}
\end{aligned}$$

5. The probability is given by

$$\begin{aligned}
P(S_t = i, S_{t+1} = j|\mathbf{Y}) &= \frac{P(S_t = i, S_{t+1} = j, \mathbf{Y})}{P(\mathbf{Y})} \\
&= \frac{P(S_t = i, \mathbf{Y}_{1:t})P(S_{t+1} = j, \mathbf{Y}_{t+1:T}|S_t = i, \mathbf{Y}_{1:t})}{(\alpha^t)^\top \beta^t} \\
&= \frac{\alpha_i^t P(\mathbf{Y}_{t+1:T}|S_t = i, S_{t+1} = j, \mathbf{Y}_{1:t})P(S_{t+1} = j|S_t = i, \mathbf{Y}_{1:t})}{(\alpha^t)^\top \beta^t} \\
&= \frac{\alpha_i^t P(\mathbf{Y}_{t+2:T}|S_t = i, S_{t+1} = j, \mathbf{Y}_{1:t+1})P(Y_{t+1} = y_{t+1}|S_t = i, S_{t+1} = j, \mathbf{Y}_{1:t})P(S_{t+1} = j|S_t = i)}{(\alpha^t)^\top \beta^t} \\
&= \frac{\alpha_i^t P(\mathbf{Y}_{t+2:T}|S_{t+1} = j)P(Y_{t+1} = y_{t+1}|S_{t+1} = j)P(S_{t+1} = j|S_t = i)}{(\alpha^t)^\top \beta^t} \\
&= \frac{\alpha_i^t \beta_j^{t+1} B_j^{t+1} A_{ij}}{(\alpha^t)^\top \beta^t}
\end{aligned}$$

Dividing α^t and β^t by their length to get the normalized forward and backward messaging vectors, i.e.,

$$\tilde{\alpha_i^t} = \frac{\alpha_i^t}{\mathbf{1}^\top \alpha^t}, \quad \tilde{\beta_i^t} = \frac{\beta_i^t}{\mathbf{1}^\top \beta^t}$$

Replacing α_i^t and β_i^t with their normalized versions in $P(S_t = i, S_{t+1} = j|\mathbf{Y})$ can render the results of this question.

5. The relational formaula is derived as follows

$$\begin{aligned}
\alpha_i^t &= P(S_t = i, \mathbf{Y}_{1:t}) \\
&= \sum_j P(S_t = i, S_{t-1} = j, \mathbf{Y}_{1:t}) \\
&= \sum_j P(S_{t-1} = j, \mathbf{Y}_{1:t-1})P(S_t = i, Y_t = y_t|S_{t-1} = j, \mathbf{Y}_{1:t-1}) \\
&= \sum_j \alpha_j^{t-1} P(Y_t = y_t|S_t = i, S_{t-1} = j, \mathbf{Y}_{1:t-1})P(S_t = i|S_{t-1} = j, \mathbf{Y}_{1:t-1}) \\
&= \sum_j \alpha_j^{t-1} P(Y_t = y_t|S_t = i)P(S_t = i|S_{t-1} = j) \\
&= \sum_j \alpha_j^{t-1} B_i^t A_{ji}
\end{aligned}$$

The matrix form is given by

$$\alpha^t = diag(B^t)\left(A^\top \alpha^{t-1}\right)$$

6. The relational formula is derived as follows

$$\begin{aligned}
\beta_i^t &= P(\mathbf{Y}_{t+1:T}|S_t = i) \\
&= \sum_j P(S_{t+1} = j, \mathbf{Y}_{t+1:T}|S_t = i) \\
&= \sum_j P(S_{t+1} = j|S_t = i)P(\mathbf{Y}_{t+1:T}|S_t = i, S_{t+1} = j) \\
&= \sum_j P(S_{t+1} = j|S_t = i)P(\mathbf{Y}_{t+2:T}|Y_{t+1} = y_{t+1}, S_t = i, S_{t+1} = j)P(Y_{t+1} = y_{t+1}|S_t = i, S_{t+1} = j) \\
&= \sum_j P(S_{t+1} = j|S_t = i)P(\mathbf{Y}_{t+2:T}|S_{t+1} = j)P(Y_{t+1} = y_{t+1}|S_{t+1} = j) \\
&= \sum_j A_{ij} \beta_j^{t+1} B_j^{t+1}
\end{aligned}$$

Therefore, the matrix form is given by

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

7. The initial values are given by

$$\alpha_i^1 = P(Y_1 = y_1|S_1 = i)P(S_1 = i) = B_i^1\pi_i^1$$

$$\beta_i^T = 1$$

8. The cost of Baum-Welch algorithm is $2(N + N^2)T + 2 + N = O(TN^2)$.
9. It is a bad idea since the complexity is $O(T^3(D + d)^3)$. If the size of observations is large, the time is consumption is huge and the system will crack down.