Laboration group report in Advanced Machine Learning

Laboration 2

732A96

Duc Tran
William Wiik
Mikael Montén
Johannes Hedström

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

23 september 2024

Contents

The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to model the behavior of a robot that walks around a ring. The ring is divided into 10 sectors. At any given time point, the robot is in one of the sectors and decides with equal probability to stay in that sector or move to the next sector. You do not have direct observation of the robot. However, the robot is equipped with a tracking device that you can access. The device is not very accurate though: If the robot is in the sector i, then the device will report that the robot is in the sectors [i - 2, i + 2] with equal probability.

1 Question 1

Build a hidden Markov model (HMM) for the scenario described above. Note that the documentation for the initHMM function says that the emissionProbs matrix should be of size [number of states]x[number of states]. This is wrong. It should be of size [number of states]x[number of symbols]. The package works correctly, though. It is just the documentation that is wrong.

```
# Vector with the names of the states.
states <- as.character(1:10)

# Vector with the names of the symbols.
symbols <- str_c("Symbol_",seq(1, 10, 1))

# Vector with the starting probabilities of the states.
startProbs <- rep(0.1, 10)


# Stochastic matrix containing the transition probabilities between the states.
transProbs <- matrix(0, 10, 10)
diag(transProbs) <- 0.5

# Transitions probabilities 1/2 probability to stay or change
transProbs <- matrix(0,nrow=10,ncol=10) # empty matrix
diag(transProbs) <- 0.5 # 0.5 to stay
diag(transProbs[,-1]) <- 0.5 # 0.5 to move to NEXT sector
transProbs[10,1] <- 0.5 # left bottom corner(step from 10 to 1)


# Stochastic matrix containing the emission probabilities of the states.
# Output of the robot at each state
emissionProbs <- matrix(0,nrow=10, ncol=10) # empty matrix

for (i  in 1:10) { # filling the matrix with probabilities, each row sums to 1
    j <- (i-2):(i+2)

    # if we are going over 10 then go back to 1 and upwards
    j[j>10] <- j[j>10] - 10

    # if we are getting a value under 1 then go from 10 and downwards
    j[j<1] <- j[j<1] + 10

  emissionProbs[i,j] <- 1/5
}

hmm <- initHMM(States = states,
               Symbols = symbols,
               startProbs = startProbs,
               transProbs = transProbs,
               emissionProbs = emissionProbs)
```

Table 1: Transitions probabilities

	1	2	3	4	5	6	7	8	9	10
Symbol_1	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Symbol_2	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Symbol_3	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0
Symbol_4	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0
Symbol_5	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0
Symbol_6	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0
Symbol_7	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0
Symbol_8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0
Symbol_9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5
Symbol_10	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5

Table 2: Emission probabilities[i - 2, i + 2]

	1	2	3	4	5	6	7	8	9	10
Symbol_1	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2
Symbol_2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2
Symbol_3	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0
Symbol_4	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0
Symbol_5	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0
Symbol_6	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0
Symbol_7	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0
Symbol_8	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2
Symbol_9	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2
Symbol_10	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2

2 Question 2

Simulate the HMM for 100 time steps.

```r
set.seed(12345)

sim <- simHMM(hmm = hmm, # A Hidden Markov Model
              length = 100) # The length of the simulated sequence of observations and states
```

3 Question 3

Discard the hidden states from the sample obtained above. Use the remaining observations to compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path.

$$\text{Filtering: } p\left(z^t \mid x^{0:t}\right) = \frac{\alpha\left(z^t\right)}{\sum_{z^t} \alpha\left(z^t\right)}$$

$$\text{Smoothing: } p\left(z^t \mid x^{0:T}\right) = \frac{\alpha\left(z^t\right)\beta\left(z^t\right)}{\sum_{z^t} \alpha\left(z^t\right)\beta\left(z^t\right)}$$

```r
# Filter
# A matrix containing the forward probabilities.
# The probabilities are given on a logarithmic scale
alpha <- forward(hmm = hmm, observation = sim$observation) %>% exp() # not normalized

# Smoothing
# A matrix containing the backward probabilities
# The probabilities are given on a logarithmic scale
beta <- backward(hmm = hmm, observation = sim$observation) %>% exp()
smoothing <- alpha * beta # not normalized

# The most probable path.
# Output: A vector of strings, containing the most probable path of states.
most_prob_path <- viterbi(hmm = hmm, observation = sim$observation)
```

4 Question 4

Compute the accuracy of the filtered and smoothed probability distributions, and of the most probable path. That is, compute the percentage of the true hidden states that are guessed by each method.

```r
# FILTER/FORWARD ================================================================
# Every column needs to sum to 1 (margin = 2 = column)
alpha_norm <- prop.table(alpha, margin = 2)

# Get the most probably state for each time step
fwd_state <- apply(alpha_norm, MARGIN = 2, FUN = which.max)

# Accuracy
acc_filter <- mean(fwd_state == sim$states)


# SMOOTHING/BACKWARD ============================================================

# Every column needs to sum to 1 (margin = 2 = column)
beta_norm <- prop.table(smoothing, margin = 2)

# Get the most probably state for each time step
bwd_state <- apply(beta_norm, MARGIN = 2, FUN = which.max)

# Accuracy
acc_smoothing <- mean(bwd_state == sim$states)


# Most probable path ============================================================

# Accuracy
acc_most_prob_path <- mean(most_prob_path == sim$states)
```

```r
df <- data.frame("Filtered" = acc_filter,
                 "Smoothed" = acc_smoothing,
                 "Most_probable_path" = acc_most_prob_path)

kable(df, caption = "Accuracy of the three methods")
```

Table 3: Accuracy of the three methods

Filtered	Smoothed	Most_probable_path
0.53	0.74	0.56

5 Question 5

Repeat the previous exercise with different simulated samples. In general, the smoothed distributions should be more accurate than the filtered distributions. Why ? In general, the smoothed distributions should be more accurate than the most probable paths, too. Why ?

```r
get_hmm_acc <- function(hmm, nseq) {


  sim <- simHMM(hmm = hmm, # A Hidden Markov Model
                length = nseq) # The length of the simulated sequence of observations and states


  # FILTER/FORWARD ===============================================================
  alpha <- forward(hmm = hmm, observation = sim$observation) %>% exp()

  alpha_norm <- prop.table(alpha, margin = 2)

  fwd_state <- apply(alpha_norm, MARGIN = 2, FUN = which.max)

  acc_filter <- mean(fwd_state == sim$states)



  # SMOOTHING/BACKWARD ===========================================================
  beta <- backward(hmm = hmm, observation = sim$observation) %>% exp()

  beta_norm <- prop.table(beta * alpha, margin = 2)

  bwd_state <- apply(beta_norm, MARGIN = 2, FUN = which.max)

  acc_smoothing <- mean(bwd_state == sim$states)



  # Most probable path ===========================================================
  most_prob_path <- viterbi(hmm = hmm, observation = sim$observation)

  acc_most_prob_path <- mean(most_prob_path == sim$states)



  return(data.frame("Filtered" = acc_filter,
                    "Smoothed" = acc_smoothing,
                    "Most_probable_path" = acc_most_prob_path))


}
```

```r
set.seed(12345)

nr_sim <- 100
mat_acc_sim <- matrix(NA, nr_sim , 3)
colnames(mat_acc_sim) <- c("Filtered", "Smoothed", "Most_probable_path")

for(i in 1:nr_sim) {

  accs <- get_hmm_acc(hmm = hmm, nseq = 100)

  mat_acc_sim[i,1] <- accs$Filtered
  mat_acc_sim[i,2] <- accs$Smoothed
  mat_acc_sim[i,3] <- accs$Most_probable_path

}


kable(as.data.frame(apply(mat_acc_sim, MARGIN = 2, mean)),
      caption = "Mean accuracy for 100 different simulated samples",
      col.names = "")
```

Table 4: Mean accuracy for 100 different simulated samples

Filtered	0.5293
Smoothed	0.6789
Most_probable_path	0.4924

The smoothed distributions are more accurate than filtered distributions, because smoothing use both past and future observations, while filtered distributions only use past and present observations. By using more information from the observations, it makes sense that the smoothed distributions are more accurate.

The smoothed distributions are also more accurate than the most probable path, because smoothed distributions are taking account the uncertainties for every time step. The most probable path outputs the whole sequence that is most probable and does not predicts at each time step.

6 Question 6

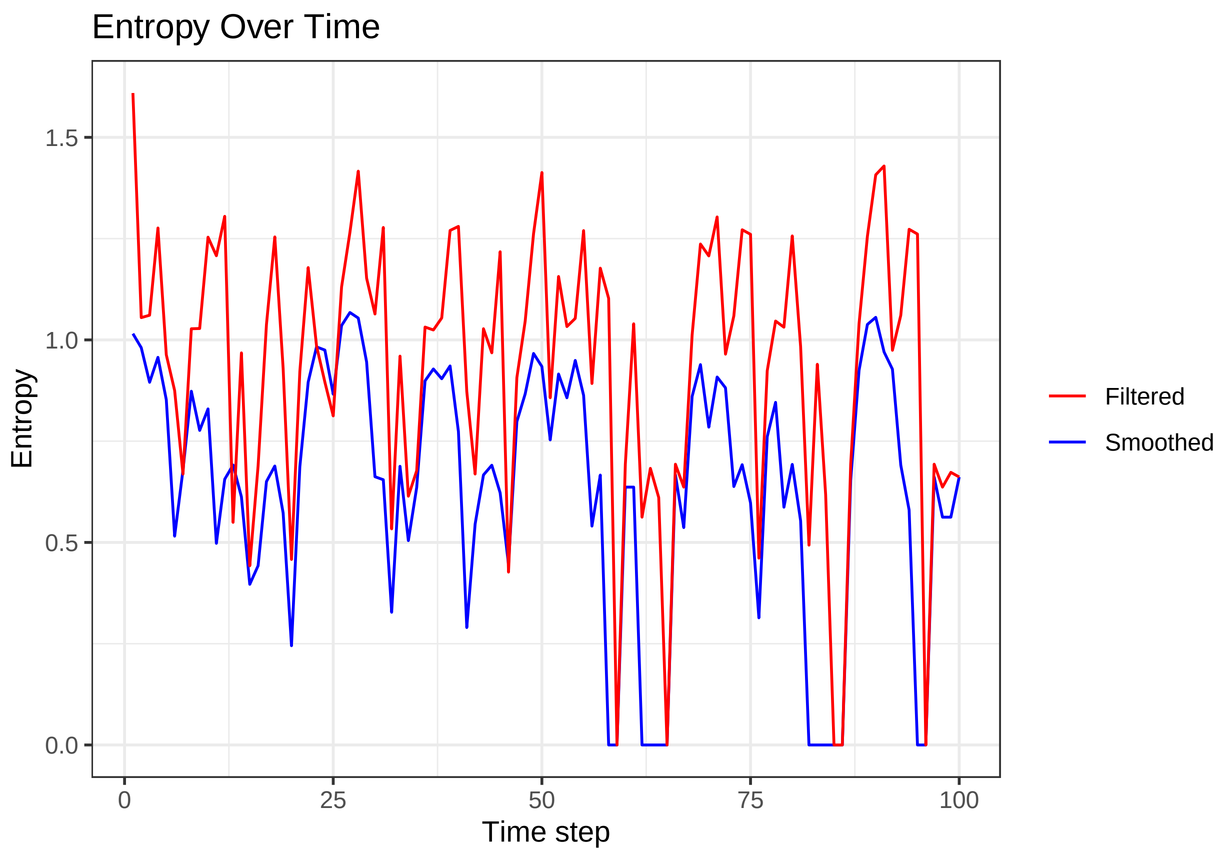
Is it always true that the later in time (i.e., the more observations you have received) the better you know where the robot is ?

```r
ent_smo <- apply(beta_norm, MARGIN = 2, FUN = entropy.empirical)
ent_filt <- apply(alpha_norm, MARGIN = 2, FUN = entropy.empirical)



df <- data.frame(Time = 1:100, Entropy_Smoothed = ent_smo, Entropy_Filtered = ent_filt)

# Plot with ggplot
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = Entropy_Smoothed, color = "Smoothed")) +
  geom_line(aes(y = Entropy_Filtered, color = "Filtered")) +
  xlab("Time step") +
  ylab("Entropy") +
  ggtitle("Entropy Over Time") +
  theme_bw() +
  scale_color_manual(values = c("Smoothed" = "blue", "Filtered" = "red")) +
  theme(legend.title = element_blank()) +
  labs(color = "Legend")
```



A low entropy value indicates that the robot's path is fairly predictable. If the entropy is 0, it means we can determine the robot's state with complete certainty. However, as shown in the graph, entropy values do not gradually decrease with time, meaning that it's not necessarily true that the later in time more the better you know where the robot is. Due to the rapid increases back to relatively high entropy after periods of very low entropy, the low entropy cannot be assumed to be certain.

7 Question 7

Consider any of the samples above of length 100. Compute the probabilities of the hidden states for the time step 101.

```r
beta <- backward(hmm = hmm, observation = sim$observation) %>% exp()
beta_norm <- prop.table(beta * alpha, margin = 2)


probs_step101 <- beta_norm[,100] %*% transProbs


probs_step101 <- t(probs_step101)
rownames(probs_step101) <- str_c("State_",seq(1, 10, 1))
colnames(probs_step101) <- "Probabilty step 101"
kable(probs_step101)
```

	Probabilty step 101
State_1	0.0000
State_2	0.1875
State_3	0.5000
State_4	0.3125
State_5	0.0000
State_6	0.0000
State_7	0.0000
State_8	0.0000
State_9	0.0000
State_10	0.0000

The most probable state for step 101 is state 3.

8 Statement of contribution

We compared everyone results and decided to go for the code structure from William & Johannes.
To ensure consistency, we compared each individual report by using the same seed. This allowed us to verify that everyone had implemented the correct functions with the appropriate arguments.

In the discussion section, we collectively summarized the input from all people. We settled on the code and interpretations that resonated best with everyone. Each person contributed to the discussions by sharing insights and perspectives on the topic.

9 Appendix

```r
set.seed(12345)
library(HMM)
library(ggplot2)
library(stringr)
library(dplyr)
library(knitr)
library(entropy)



# Vector with the names of the states.
states <- as.character(1:10)

# Vector with the names of the symbols.
symbols <- str_c("Symbol_",seq(1, 10, 1))

# Vector with the starting probabilities of the states.
startProbs <- rep(0.1, 10)


# Stochastic matrix containing the transition probabilities between the states.
transProbs <- matrix(0, 10, 10)
diag(transProbs) <- 0.5

# Transitions probabilities 1/2 probability to stay or change
transProbs <- matrix(0,nrow=10,ncol=10) # empty matrix
diag(transProbs) <- 0.5 # 0.5 to stay
diag(transProbs[,-1]) <- 0.5 # 0.5 to move to NEXT sector
transProbs[10,1] <- 0.5 # left bottom corner(step from 10 to 1)


# Stochastic matrix containing the emission probabilities of the states.
# Output of the robot at each state
emissionProbs <- matrix(0,nrow=10, ncol=10) # empty matrix

for (i  in 1:10) { # filling the matrix with probabilities, each row sums to 1
    j <- (i-2):(i+2)

    # if we are going over 10 then go back to 1 and upwards
    j[j>10] <- j[j>10] - 10

    # if we are getting a value under 1 then go from 10 and downwards
    j[j<1] <- j[j<1] + 10

  emissionProbs[i,j] <- 1/5
}
```

```r
hmm <- initHMM(States = states,
               Symbols = symbols,
               startProbs = startProbs,
               transProbs = transProbs,
               emissionProbs = emissionProbs)
```

```r
rownames(transProbs) <- symbols
colnames(transProbs) <- states
rownames(emissionProbs) <- symbols
colnames(emissionProbs) <- states
knitr::kable(transProbs, caption='Transitions probabilities')

knitr::kable(emissionProbs, caption='Emission probabilities[i - 2, i + 2]')

set.seed(12345)

sim <- simHMM(hmm = hmm, # A Hidden Markov Model
              length = 100) # The length of the simulated sequence of observations and states
```

```r
# Filter
# A matrix containing the forward probabilities.
# The probabilities are given on a logarithmic scale
alpha <- forward(hmm = hmm, observation = sim$observation) %>% exp() # not normalized

# Smoothing
# A matrix containing the backward probabilities
# The probabilities are given on a logarithmic scale
beta <- backward(hmm = hmm, observation = sim$observation) %>% exp()
smoothing <- alpha * beta # not normalized

# The most probable path.
# Output: A vector of strings, containing the most probable path of states.
most_prob_path <- viterbi(hmm = hmm, observation = sim$observation)
```

```r
# FILTER/FORWARD ================================================================
# Every column needs to sum to 1 (margin = 2 = column)
alpha_norm <- prop.table(alpha, margin = 2)
```

```r
# Get the most probably state for each time step
fwd_state <- apply(alpha_norm, MARGIN = 2, FUN = which.max)

# Accuracy
acc_filter <- mean(fwd_state == sim$states)


# SMOOTHING/BACKWARD ============================================================

# Every column needs to sum to 1 (margin = 2 = column)
beta_norm <- prop.table(smoothing, margin = 2)

# Get the most probably state for each time step
bwd_state <- apply(beta_norm, MARGIN = 2, FUN = which.max)

# Accuracy
acc_smoothing <- mean(bwd_state == sim$states)


# Most probable path ============================================================

# Accuracy
acc_most_prob_path <- mean(most_prob_path == sim$states)



df <- data.frame("Filtered" = acc_filter,
                 "Smoothed" = acc_smoothing,
                 "Most_probable_path" = acc_most_prob_path)

kable(df, caption = "Accuracy of the three methods")



get_hmm_acc <- function(hmm, nseq) {


  sim <- simHMM(hmm = hmm, # A Hidden Markov Model
                length = nseq) # The length of the simulated sequence of observations and states


  # FILTER/FORWARD ================================================================
  alpha <- forward(hmm = hmm, observation = sim$observation) %>% exp()

  alpha_norm <- prop.table(alpha, margin = 2)

  fwd_state <- apply(alpha_norm, MARGIN = 2, FUN = which.max)
  acc_filter <- mean(fwd_state == sim$states)



  # SMOOTHING/BACKWARD ============================================================
  beta <- backward(hmm = hmm, observation = sim$observation) %>% exp()

  beta_norm <- prop.table(beta * alpha, margin = 2)

  bwd_state <- apply(beta_norm, MARGIN = 2, FUN = which.max)

  acc_smoothing <- mean(bwd_state == sim$states)



  # Most probable path ============================================================
  most_prob_path <- viterbi(hmm = hmm, observation = sim$observation)

  acc_most_prob_path <- mean(most_prob_path == sim$states)



  return(data.frame("Filtered" = acc_filter,
                    "Smoothed" = acc_smoothing,
                    "Most_probable_path" = acc_most_prob_path))


}

set.seed(12345)

nr_sim <- 100
mat_acc_sim <- matrix(NA, nr_sim , 3)
colnames(mat_acc_sim) <- c("Filtered", "Smoothed", "Most_probable_path")

for(i in 1:nr_sim) {

  accs <- get_hmm_acc(hmm = hmm, nseq = 100)

  mat_acc_sim[i,1] <- accs$Filtered
  mat_acc_sim[i,2] <- accs$Smoothed
  mat_acc_sim[i,3] <- accs$Most_probable_path

}


kable(as.data.frame(apply(mat_acc_sim, MARGIN = 2, mean)),
      caption = "Mean accuracy for 100 different simulated samples",
      col.names = "")
```

```r
ent_smo <- apply(beta_norm, MARGIN = 2, FUN = entropy.empirical)
ent_filt <- apply(alpha_norm, MARGIN = 2, FUN = entropy.empirical)



df <- data.frame(Time = 1:100, Entropy_Smoothed = ent_smo, Entropy_Filtered = ent_filt)

# Plot with ggplot
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = Entropy_Smoothed, color = "Smoothed")) +
  geom_line(aes(y = Entropy_Filtered, color = "Filtered")) +
  xlab("Time step") +
  ylab("Entropy") +
  ggtitle("Entropy Over Time") +
  theme_bw() +
  scale_color_manual(values = c("Smoothed" = "blue", "Filtered" = "red")) +
  theme(legend.title = element_blank()) +
  labs(color = "Legend")



beta <- backward(hmm = hmm, observation = sim$observation) %>% exp()
beta_norm <- prop.table(beta * alpha, margin = 2)


probs_step101 <- beta_norm[,100] %*% transProbs


probs_step101 <- t(probs_step101)
rownames(probs_step101) <- str_c("State_",seq(1, 10, 1))
colnames(probs_step101) <- "Probabilty step 101"
kable(probs_step101)
```