Adv ML Lab2

Andreas Stasinakis(andst745) & Mim Kemal Tekin(mimte666) & Stefano Toffol(c) & Bruno Barakat(bruba569)

September 11, 2019

Contents

Problem

In this lab we are asked to model the following problem using the package `HMM` from R: a robot is moving around a ring. The ring is divided into 10 sectors and the robot may move to the next one or staying in its current position with equal probability. The starting point is random and we have no way to record its decisions. We do have however the log of all its movement, recorded with a tracking device. The data is not trustworthy though: each time the device makes an error of ± 2, meaning that the data correspond with equal probability to any of the point in the interval $[i-2, i+2]$, where i is the real position of the robot.

Question 1 - Build a HMM model (mimte666)

In this question we are asked to make a HMM model for the scenario described above.
The code used is the following:

```
# Denominate the states of our model
N_state <- 10
sectors <- as.character(1:N_state)

# The data has an error in the recordings but we assume a cyclical structure:
# a robot in sector 1 will never record its current position as 0 or -1 but will
# instead record position 10 or 9.
records <- sectors

# In the beginning our robot is equally likely to be in any state
start_prob <- rep(0.1, N_state)

# The transition matrix has 0.5 on the main diagonal and 0.5 in the successive diagonal
move_prob <- c(0.5, 0.5)
emission_range <- 2
# Define transition matrix
transition_row <- double(N_state)
transition_row[1:length(move_prob)] <- move_prob
trans_mat <- matrix(rep(transition_row, N_state), nrow = N_state, byrow = T)
for(i in 2:N_state){
  trans_mat[i, ] <- shift(transition_row, i-1)
}

# Similar story for the way the data is collected, only that this time we have
# 5 possible, equally likely, values --> Prob is 0.2
total <- 2 * emission_range + 1
emission_row <- rep(1/total, N_state)
emission_row[(emission_range+2) : (N_state-emission_range)] <- 0
records_mat <- matrix(rep(double(N_state), N_state), nrow = N_state, byrow = T)
for(i in 1:N_state){
  records_mat[i, ] <- shift(emission_row, i-1)
}

# All the ingredients are ready, we can now fit the model!
fit_MM <- initHMM(sectors, records, start_prob, trans_mat, records_mat)
```

Question 2 - Simulation from the model (steto820)

In this question we are asked to simulate the outcome of a 100-steps simulation.
The task can be achieved with a simple line of code:

```
# Reproducible simulation:
set.seed(12345)
simulation1 <- simHMM(fit_MM, 100)
```

We can briefly summarize the results with this visualization:

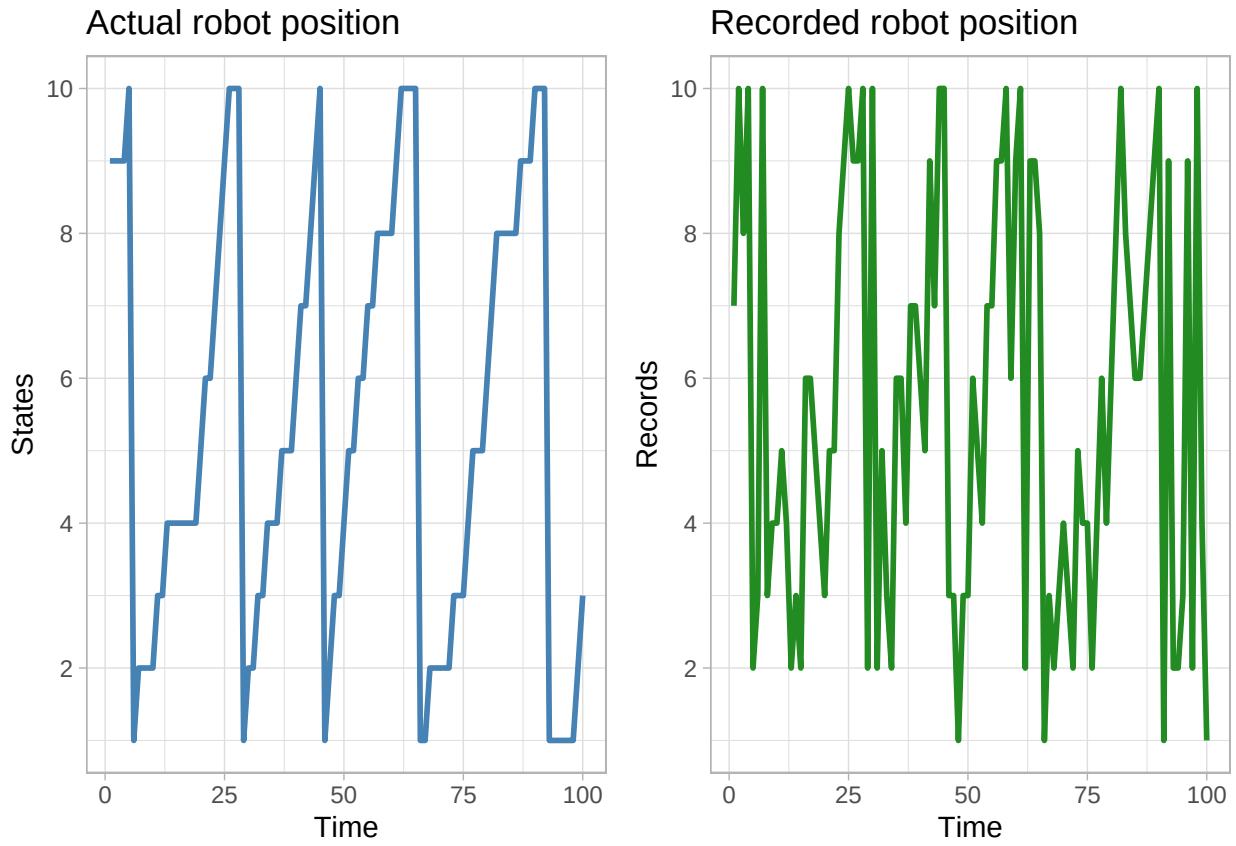


Figure 1: Outcome of the first 100-step simulation

Question 3 - Filtered and smoothed probability distributions (steto820)

In this task we are asked to evaluate the filtered and smoothed probability distribution for each of the 100 time points based exclusively on the records and compute the most likely path:

```
# Compute the smoothed probability distribution for each recorded position:
smoothed1 <- posterior(fit_MM, Records)
# Find the most likely value for each point in time:
path_smoothed1 <- apply(smoothed1, 2, which.max)

# Compute the filtered probability distribution for each recorded position:
filtered1 <- forward(fit_MM, Records)
# Transform the output in probability:
unnorm_filtered1 <- exp(filtered1)
# Render 0 all the points that previously where -Inf, then normalize
ind_inf1 <- which(filtered1==-Inf)  # It avoids computational instabilities
unnorm_filtered1[ind_inf1] <- 0
norm_filtered1 <- apply(unnorm_filtered1, 2, function(x) x/sum(x))
# Find the most likely value for each point in time:
path_filtered1 <- apply(norm_filtered1, 2, which.max)

# Find the most likely path:
path_viterbi1 <- as.numeric(viterbi(fit_MM, Records))

# Plot it:
```

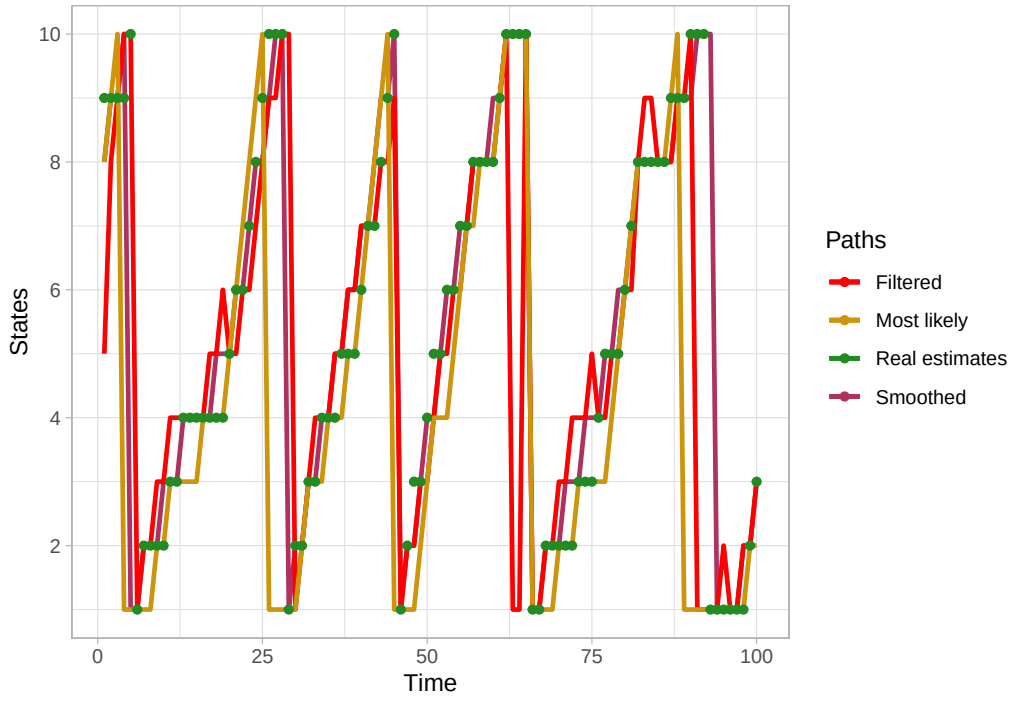


Figure 2: Comparison between the three estimathed paths, represented by a colored line, and the previously discovered states, displeyed as green dots.

Question 4 - Accuracy (steto820)

In this tasked we are asked to compute the accuracy (percentage) of the filtered, smoothed and viterbi estimates. The hidden states discovered in Question 2 will be treated as the true values.

```
# Compute the various accuracies
accuracy_smoothed1 <- sum(path_smoothed1==States)
accuracy_filtered1 <- sum(path_filtered1==States)
accuracy_viterbi1 <- sum(path_viterbi1==States)
```

Table 1: Accuracy rates for the three methods used (percentage).

Smoothed	**Filtered**	**Viterbi**
74 %	53 %	56 %

As we can see, of the three methods the one with the worst accuracy is the filtered one, with barely 53%, performing badly overall. If we also consider the graph plotted before, we can identify how sometimes the filtered estimates make the robot go back, ulteriorly decreasing the accuracy. This happens because in certain moments the model gives the exact same weight to more than one state, which results in taking as predicted state the first one in order (default behaviour of the `which.max` function).
While the Viterbi estimates perform similarly to the filtered ones, the smoothed probabilities lead to almost a 75% accuracy, resulting in the best estimates and a great method overall.

Question 5 - Different samples (steto820)

In this question we are asked to repeat what was done above but with different experiments, i.e. generating new data using different random seeds. We will adapt the same chunks of code as above and only report the following table summarizing the accuracy of other 9 different experiments:

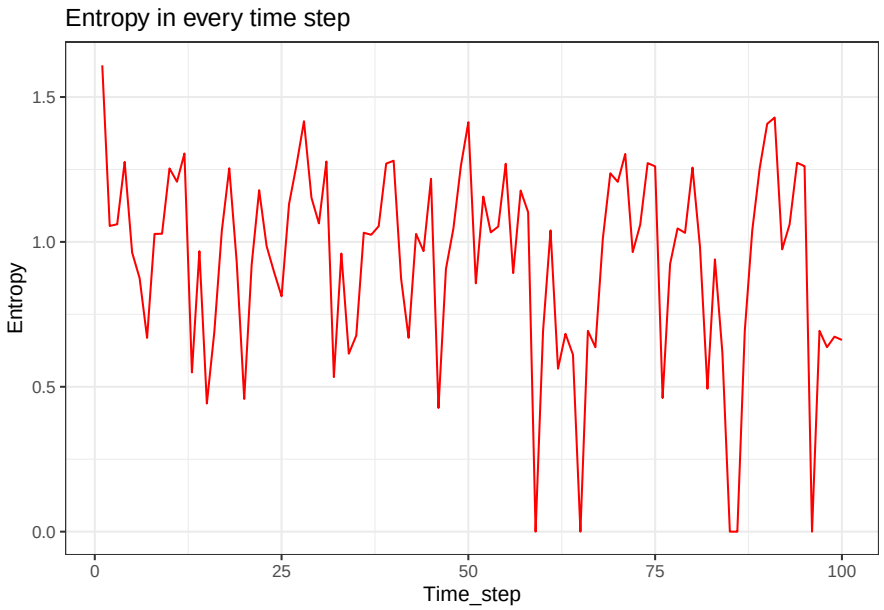
Table 2: Accuracy rates for the three methods used (percentage) across 10 different experiments.

Seeds	**Smoothed**	**Filtered**	**Viterbi**
12345	74 %	53 %	56 %
11111	62 %	53 %	48 %
22222	69 %	59 %	43 %
33333	62 %	52 %	42 %
44444	61 %	56 %	52 %
55555	60 %	51 %	37 %
66666	57 %	51 %	33 %
77777	72 %	53 %	43 %
88888	77 %	60 %	48 %
99999	64 %	51 %	42 %

The table shows what we were expecting: the smoothing estimates are always more accurate than the other two methods. In fact the former uses all the available observations to build the probabilities: both the forward and the backwards approaches are combined to estimate the probability distributions.
The fitered distribution on the other hand makes use exclusively of the backward approach, with the sole focus of fitting the model to the present time rather than considering the "forecast" point of view (incapsulated in the forward algorithm).
The Viterbi algorithm instead of predicting the marginal probabilities of each state tries to create the model that is most likely to generate the observed records. In a problem such as this one, with a really high entropy (50% chance of jumping to the next state and and equal 20% to observe one of the five possible records) the outcome of the Viterbi algorithm heavily depends on the realizations of the process and will rarely be able to correctly guess the correct parameters.

Question 6 - Entropy (andst745)

In this question we are asked if increasing the amount of generated observations will lead to an increase of accuracy or not. We are also suggested to evaluate the entropy of the filtered distribution using the function `entropy.empirical(·)` from the package `entropy`.
To achieve this task we will compute the entropy of the estimated probability distribution for each moment in time.



As we can see from the plot above, we can not say that the more observations the better the accuracy of the HMM. Specifically, we plot the entropy of the distribution for each time step.
The entropy of the filtered distribution is a measure of the uncertainty. Therefore, the higher the entropy, the more uncertanty our distribution has. If the number of observations was important, we would have observed that while the time step is increasing, the entropy reduces as well. In this case though, the entropy is fluctuating with the passing of time. Therefore, we can not agree that more observations give a better knowledge for the robot's position.

Question 7 - Prediction (bruba569)

The goal is to compute $p\left(Z^{101}|X^{1:100}\right)$.

Using the law of total probability for conditional events, we can write, using all possible states of Z^{100} as a finite partition of the sample space:

$$p\left(Z^{101}|X^{1:100}\right) = \sum_{Z^{100}} p\left(Z^{101}|Z^{100}, X^{0:100}\right) p\left(Z^{100}|X^{0:100}\right)$$

Then, using the Markovian assumption, it leads to

$$p\left(Z^{101}|X^{0:100}\right) = \sum_{Z^{100}} p\left(Z^{101}|Z^{100}\right) p\left(Z^{100}|X^{0:100}\right)$$

$p\left(Z^{100}|X^{0:100}\right)$ has been computed, it's the last point of the filtering distribution. We compute $\sum_{Z^{100}} p\left(Z^{101}|Z^{100}\right) p\left(Z^{100}|X^{0:100}\right)$ using the transition matrix. For each possible state Z^{100}, the probability is distributed into the other states: we only need to make a matrix multiplication to get the predictions.

Table 3: Smoothed prediction for $t = 101$.

States									
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0.1875	0.5	0.3125	0	0	0	0	0	0

Appendix

```
knitr::opts_chunk$set(echo = F, message = F, error = T, warning = F,
                      fig.align='center', out.width="70%")


library(knitr)
library(kableExtra)

shift  <- function(v, times=1){
  n <- length(v)
  times <- times %% n
  return(c(v[-(1:(n-times))], v[(1:(n-times))]))
}


# Setup
library(HMM)


# Denominate the states of our model
N_state <- 10
sectors <- as.character(1:N_state)

# The data has an error in the recordings but we assume a cyclical structure:
# a robot in sector 1 will never record its current position as 0 or -1 but will
# instead record position 10 or 9.
records <- sectors

# In the beginning our robot is equally likely to be in any state
start_prob <- rep(0.1, N_state)

# The transition matrix has 0.5 on the main diagonal and 0.5 in the successive diagonal
move_prob <- c(0.5, 0.5)
emission_range <- 2
# Define transition matrix
transition_row <- double(N_state)
transition_row[1:length(move_prob)] <- move_prob
trans_mat <- matrix(rep(transition_row, N_state), nrow = N_state, byrow = T)
for(i in 2:N_state){
  trans_mat[i, ] <- shift(transition_row, i-1)
}

# Similar story for the way the data is collected, only that this time we have
# 5 possible, equally likely, values --> Prob is 0.2
total <- 2 * emission_range + 1
emission_row <- rep(1/total, N_state)
emission_row[(emission_range+2) : (N_state-emission_range)] <- 0
records_mat <- matrix(rep(double(N_state), N_state), nrow = N_state, byrow = T)
for(i in 1:N_state){
  records_mat[i, ] <- shift(emission_row, i-1)
}

# All the ingredients are ready, we can now fit the model!
```

```r
fit_MM <- initHMM(sectors, records, start_prob, trans_mat, records_mat)


# Reproducible simulation:
set.seed(12345)
simulation1 <- simHMM(fit_MM, 100)


# Visualize the simulation
library(ggplot2)
library(gridExtra)

# Data for the plot
Time <- 1:100
States <- as.numeric(simulation1$states)
Records <- as.numeric(simulation1$observation)
d_plot <- data.frame(Time, States, Records)

p1 <- ggplot(d_plot, aes(x = Time, y = States)) +
  geom_line(col = "steelblue", size = 1) +
  scale_y_continuous(breaks = seq(2, 10, 2)) +
  ggtitle("Actual robot position") +
  theme_light()
p2 <- ggplot(d_plot, aes(x = Time, y = Records)) +
  geom_line(col = "forestgreen", size = 1) +
  scale_y_continuous(breaks = seq(2, 10, 2)) +
  ggtitle("Recorded robot position") +
  theme_light()

grid.arrange(p1, p2, ncol = 2)


# Compute the smoothed probability distribution for each recorded position:
smoothed1 <- posterior(fit_MM, Records)
# Find the most likely value for each point in time:
path_smoothed1 <- apply(smoothed1, 2, which.max)

# Compute the filtered probability distribution for each recorded position:
filtered1 <- forward(fit_MM, Records)
# Transform the output in probability:
unnorm_filtered1 <- exp(filtered1)
# Render 0 all the points that previously where -Inf, then normalize
ind_inf1 <- which(filtered1==-Inf)  # It avoids computational instabilities
unnorm_filtered1[ind_inf1] <- 0
norm_filtered1 <- apply(unnorm_filtered1, 2, function(x) x/sum(x))
# Find the most likely value for each point in time:
path_filtered1 <- apply(norm_filtered1, 2, which.max)

# Find the most likely path:
path_viterbi1 <- as.numeric(viterbi(fit_MM, Records))

# Plot it:
```

```r
ggplot(data.frame(Time = 1:100, Smoothed = unname(path_smoothed1),
                  Filtered = unname(path_filtered1), Viterbi = path_viterbi1,
                  Estimated = States)) +
  geom_line(aes(x = Time, y = Smoothed, col = "Smoothed"), size = 1) +
  geom_line(aes(x = Time, y = Filtered, col = "Filtered"), size = 1) +
  geom_line(aes(x = Time, y = Viterbi, col = "Most likely"), size = 1) +
  geom_point(aes(x = Time, y = Estimated, col = "Real estimates")) +
  scale_y_continuous(breaks = seq(2, 10, 2)) +
  scale_color_manual(
    values = c("Smoothed" = "maroon",
               "Filtered" = "red",
               "Most likely" = "darkgoldenrod3",
               "Real estimates" = "forestgreen"), name = "Paths") +
  labs(y = "States") +
  theme_light()


# Compute the various accuracies
accuracy_smoothed1 <- sum(path_smoothed1==States)
accuracy_filtered1 <- sum(path_filtered1==States)
accuracy_viterbi1 <- sum(path_viterbi1==States)


# Present them in a table:
library(kableExtra)
kable(data.frame(Smoothed = paste(accuracy_smoothed1, "%"),
                 Filtered = paste(accuracy_filtered1, "%"),
                 Viterbi = paste(accuracy_viterbi1, "%")),
      "latex", booktabs = T, align = "c",
      caption = "Accuracy rates for the three methods used (percentage).") %>%
  row_spec(0, bold = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 10)


# A function to repeat the processes above for a set number of seeds
repeat_experiment <- function(observations = 100) {

  # Random seeds to test
  seeds <- seq(11111, 99999, by = 11111)

  # Object to return with the stored accuracies
  accuracies <- data.frame(Seeds = seeds,
                           Smoothed = NA,
                           Filtered = NA,
                           Viterbi = NA)


  # For cycle: redo the above questions 9 more times:
  for(simulation in 1:length(seeds)) {

    # Regenerate the data
    set.seed(seeds[simulation])
    temp_sim <- simHMM(fit_MM, observations)
```

```
    temp_records <- as.numeric(temp_sim$observation)
    temp_states <- as.numeric(temp_sim$states)

    # Recompute probabilities:
    # Smoothed...
    temp_smoothed <- posterior(fit_MM, temp_records)
    temp_path_smoothed <- apply(temp_smoothed, 2, which.max)
    # ...Filtered...
    temp_filtered <- forward(fit_MM, temp_records)
    temp_unnorm_filtered <- exp(temp_filtered)
    temp_ind_inf <- which(temp_filtered==-Inf)
    temp_unnorm_filtered[temp_ind_inf] <- 0
    temp_norm_filtered <- apply(temp_unnorm_filtered, 2, function(x) x/sum(x))
    temp_path_filtered <- apply(temp_norm_filtered, 2, which.max)
    # ...and Viterbi.
    temp_path_viterbi <- as.numeric(viterbi(fit_MM, temp_records))

    # Compute and assign the various accuracies
    accuracies$Smoothed[simulation] <- paste(sum(temp_path_smoothed==temp_states), "%")
    accuracies$Filtered[simulation] <- paste(sum(temp_path_filtered==temp_states), "%")
    accuracies$Viterbi[simulation] <- paste(sum(temp_path_viterbi==temp_states), "%")

  }

  # Add our first seed
  accuracies <- rbind(c(12345,
                        paste(accuracy_smoothed1,  "%"),
                        paste(accuracy_filtered1,  "%"),
                        paste(accuracy_viterbi1,  "%")),
                      accuracies)

  return(accuracies)

}

# Run the function and show the result
kable(repeat_experiment(), "latex", booktabs = T, align = "c",
      caption = "Accuracy rates for the three methods used (percentage) across
      10 different experiments.") %>%
  row_spec(0, bold = T) %>%
  column_spec(1, border_right = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 10)


# We calculate the entropy for each time step.
# Each time step has each one distribution(Sums to 1), so we calculate the
# entropy for t1. After that we are in t2, which means we have more information.
# We want to see if that will give us less uncertaincy.
library(entropy)
entropies = rep(0, ncol(norm_filtered1))

for (i in 1:ncol(norm_filtered1)) {
  entropies[i] = entropy.empirical(norm_filtered1[,i])
```

```
}


# Plot the entropies for each time step
time_step = seq(1,100)

ggplot()+
  geom_line(mapping = aes(x = time_step, y = entropies), color = "red")+
  labs(title = "Entropy in every time step",x = "Time_step", y = "Entropy" )+
  theme_bw()


# Matrix multiplication between the transposed transiction matrix and the last time
prediction101 <- t(fit_MM$transProbs) %*% norm_filtered1[,100]
prediction101 <- as.data.frame(t(prediction101))
colnames(prediction101) <- 1:10

kable(prediction101, "latex", booktabs = T, align = "c",
      caption = "Smoothed prediction for $t = 101$.") %>%
  row_spec(0, bold = T) %>%
  add_header_above(c("States" = 10), italic = T) %>%
  kable_styling(latex_options = "hold_position", font_size = 10)
```