LAB 2: HIDDEN MARKOV MODELS

Phillip Hölscher

25 9 2019

Contents

Libraries

```r
# used libraries in this lab
library(HMM)
library(entropy)
library(ggplot2)
library(ggplot2)
```

Case

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.

Question 1

Build a hidden Markov model (HMM) for the scenario described above.

Result question 1

```r
#### we have to create a HMM based on the senario above
## in the package HMM is a function called initHMM
## for this we need to init following arguments:
# States - Vector with the names of the states.
# Symbols - Vector with the names of the symbols.
# startProbs - Vector with the starting probabilities of the states.
# transProbs - Stochastic matrix containing the transition
#              probabilities between the states. - Z
# emissionProbs - Stochastic matrix containing the emission probabilities of the states. - X

# States
states = seq(from = 1, to = 10, by = 1)

# Symbols
symbols = seq(from = 1, to = 10, by = 1)

# startProbs
startprobs = rep(0.1,10)

# transProbs
# there is a 0.5 stay and 0.5 move probabiliy
transprobs = matrix(0, nrow = 10, ncol = 10)
diag(transprobs) <- 0.5
diag(transprobs[,-1]) <- 0.5
transprobs[10,1] <- 0.5

# emissionProbs
```

```r
emissionprobs = c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2,
                  0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2,
                  0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0,
                  0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0,
                  0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0,
                  0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0,
                  0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0,
                  0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2,
                  0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2,
                  0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)
emissionprobs = matrix(data = emissionprobs,
                          nrow = 10,
                          ncol = 10)


# we init all needed parameter to init an HMM
# column - & row - state
hmm <- initHMM(startProbs = startprobs,
               States = states,
               Symbols = symbols,
               transProbs = transprobs,
               emissionProbs = emissionprobs)
```

The initialized hidden Markov model

```r
hmm
```

```
## $States
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $Symbols
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $startProbs
##   1   2   3   4   5   6   7   8   9  10
## 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##
## $transProbs
##     to
## from   1   2   3   4   5   6   7   8   9  10
##   1  0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   2  0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   3  0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
##   4  0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
##   5  0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
##   6  0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
##   7  0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
##   8  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
##   9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
##   10 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5
##
## $emissionProbs
##       symbols
## states   1   2   3   4   5   6   7   8   9  10
##     1  0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
##     2  0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
##     3  0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
##     4  0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
##     5  0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
##     6  0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
##     7  0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
##     8  0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
##     9  0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
##     10 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

Question 2

Simulate the HMM for 100 time steps.

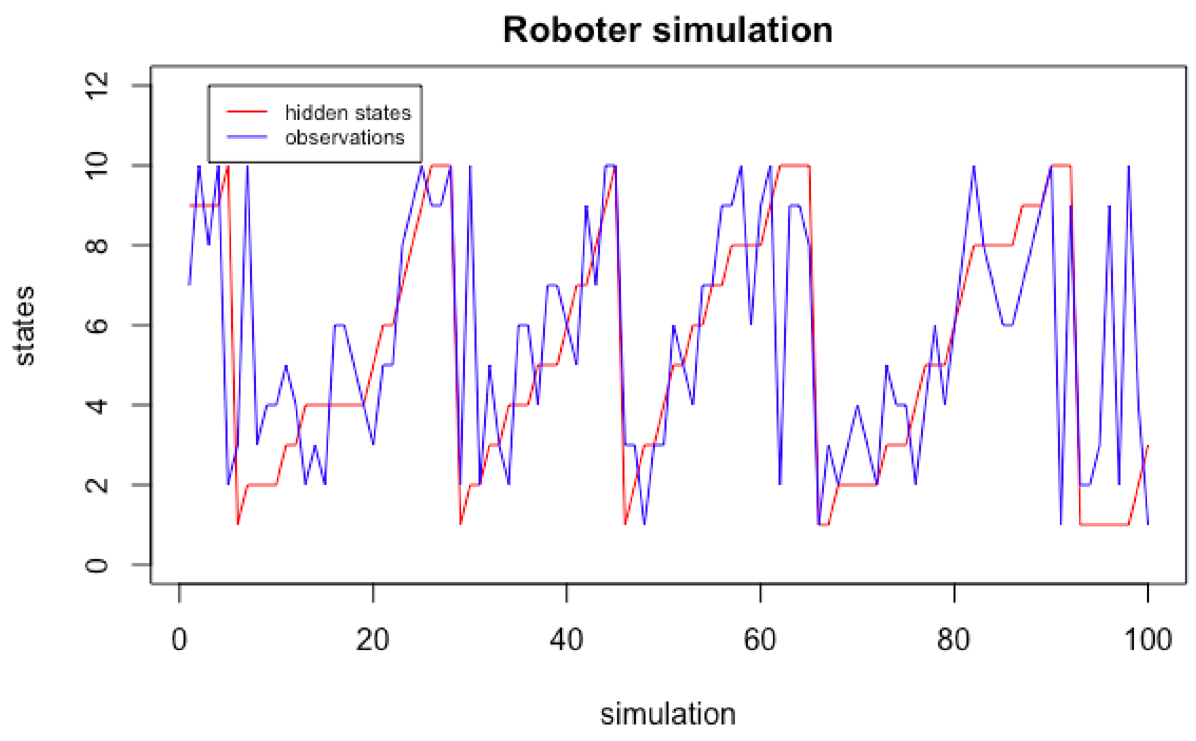
Result question 2

```
set.seed(12345)
# run simulations
simulation_nr = 100
simulation = simHMM(hmm = hmm,
                    length = simulation_nr)
```

The simulation of the hidden Markov model, where the variable *states* represent the hidden state and *observation* the observed state.

```
## $states
##   [1]  9  9  9  9 10  1  2  2  2  2  3  3  4  4  4  4  4  4  4  5  6  6  7
##  [24]  8  9 10 10 10  1  2  2  3  3  4  4  4  5  5  5  6  7  7  8  9 10  1
##  [47]  2  3  3  4  5  5  6  6  7  7  8  8  8  8  9 10 10 10 10  1  1  2  2
##  [70]  2  2  2  3  3  3  4  5  5  5  6  7  8  8  8  8  8  9  9  9 10 10 10
##  [93]  1  1  1  1  1  1  2  3
##
## $observation
##   [1]  7 10  8 10  2  3 10  3  4  4  5  4  2  3  2  6  6  5  4  3  5  5  8
##  [24]  9 10  9  9 10  2 10  2  5  3  2  6  6  4  7  7  6  5  9  7 10 10  3
##  [47]  3  1  3  3  6  5  4  7  7  9  9 10  6  9 10  2  9  9  8  1  3  2  3
##  [70]  4  3  2  5  4  4  2  4  6  4  6  8 10  8  7  6  6  7  8  9 10  1  9
##  [93]  2  2  3  9  2 10  4  1
```

In this plot we can observe the previous simulation.



```
######### Also ggplot does not work!!!!
##### I save every picutre, save it as picture in load it into the the report!!!!
```

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.

Result question 3

In this and next part of the task we will filtered probability, smoothed probability distributions and calculate the most probable path. For this we will use a certain function for the respective algorithm and then do some calculations to create a prediction for the respective algorithm. This is then compared with the hidden state to produce a prediction accuracy. For the *filtering* do I use the *forward* algorithm, to compute the smoothed do I use the **posterior** function. We generate the alpha and beta with this function, which we need to run the *forward-backward* algorithm. With the viterbi algorihum do we compute the most probable path, this algorithm does not allow to jump between the states and chooses the next state of the current state.

```r
# compute filtered probability distributions
forwar_method_log = forward(hmm = hmm, observation = simulation$observation)
# compute smoothed probability distributions
smoothed = posterior(hmm = hmm, observation = simulation$observation)
# Compute most probable path
viterbi_method = viterbi(hmm = hmm, observation = simulation$observation)
```

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.

Hint: Note that the function *forward* in the *HMM* package returns probabilities in log scale. You may need to use the functions *exp* and *prop.table* in order to obtain a normalized probability distribution. You may also want to use the functions *apply* and *which.max* to find out the most probable states. Finally, recall that you can compare two vectors A and B elementwise as A==B, and that the function *table* will count the number of times that the different elements in a vector occur in the vector.

Result question 4

In this task part I calculate a prediction accruacy for all three algorithms. The calculation is well commented to follow the steps.

- Forward

```
##### compute filtered proability distributions

# Remove the log-transformation
forwar_method = exp(forwar_method_log)
# Normalizing
forwar_method_norm = prop.table(forwar_method, margin = 2)
# margin over the columns -> 2
# Checking which probability in each column is the highest
forwar_method_prob = apply(forwar_method_norm, MARGIN = 2, FUN = which.max)
# margin over the columns -> 2
# Accuracy for the filter
accuracy_filter = sum(forwar_method_prob == simulation$states)/
  length(simulation$states)
```

- Smoothing

```
#####  compute smoothed proability distributions

# Normalizing
norm_smoothed = prop.table(smoothed, margin = 2)
most_prob_smoothed = apply(norm_smoothed, MARGIN = 2, FUN = which.max)
# Accuracy for the smoothed
accuracy_smoothed = sum(most_prob_smoothed == simulation$states) /
  length(simulation$states)
```

- Most proable path

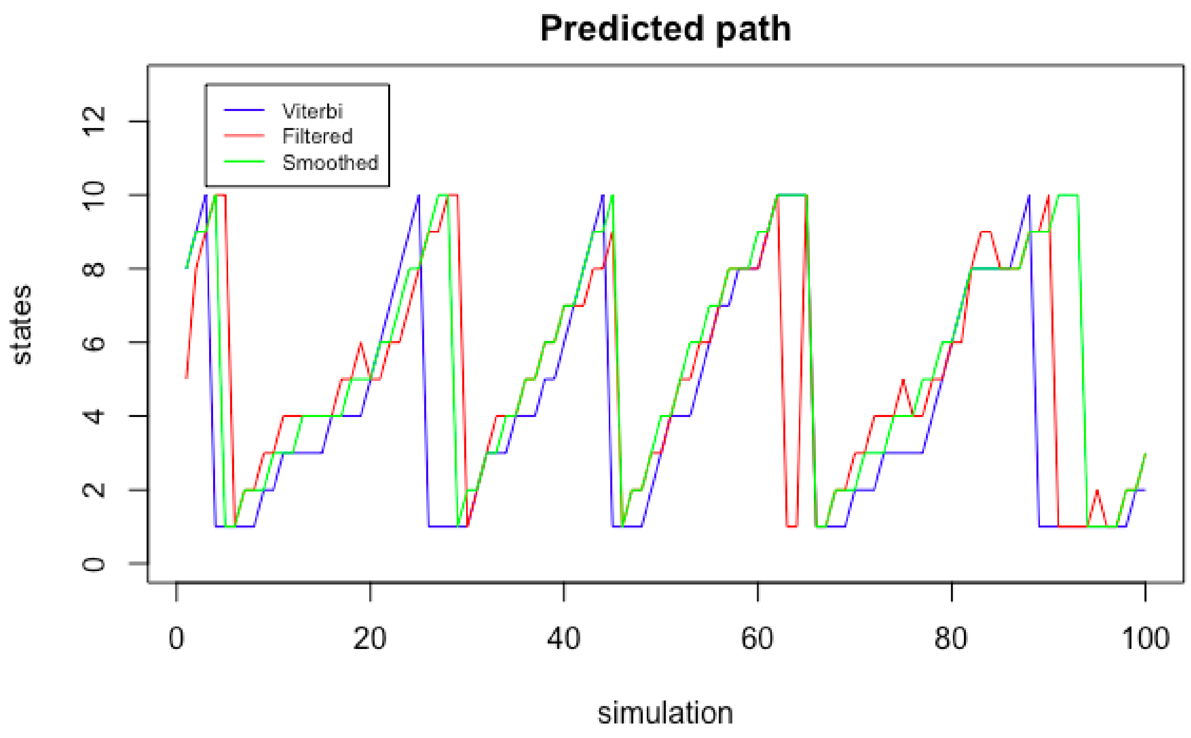
```
#####  most probable path
accuracy_viterbi = sum(viterbi_method == simulation$states) /
  length(simulation$states)
```

```
## The accuracy of the following methods
```

filtered	smoothed	viterbi
0.53	0.74	0.56

The prediction accuracy of the smoothed is highest and the other two methods are very similar.

In this visualization is the state prediction for each algorithm. In the first simulation it can already be seen that the start value is different. It has to be said that a similar process can be seen with all methods. In the case of the forward algorithm (filtered) it can be seen that this also goes back to states. That's one, you can see that the Foward algorithm goes by probabilities and doesn't consider the rules, because in the case description at the beginning of the lab you can see that the robot can only stop or move forward, this was initialized in the *transProbs*. Furthermore, jumps from forward from state 9 to state 1 can be seen, thus the forward skips the state 10. These phenomena do not occur with the viterbi algorithm, because it follows the most prob path which does excludes jumps.



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 ?

Result question 5

For this task, I created a function to perform the preceding steps more efficiently. In this part of the task a hidden Markov model was simulated 3 times, as we did in Question 2. The difference is that 3 more simulations were done with different seed. All simulations have a length of 100 again.

```
# create a new simulation
# run simulations
simulation_nr_e5 = 100

### create different samlpe sets
## do this, by running the simulation with different seeds

# caste 1
set.seed(123456)
simulation_e5_1 = simHMM(hmm = hmm, length = simulation_nr_e5)

# caste 2
set.seed(1234567)
simulation_e5_2 = simHMM(hmm = hmm, length = simulation_nr_e5)

# caste 3
set.seed(12345678)
simulation_e5_3 = simHMM(hmm = hmm, length = simulation_nr_e5)
```

```
## Result case 1
```

filtered	smoothed	viterbi
0.59	0.75	0.55

```
## Result case 2
```

filtered	smoothed	viterbi
0.48	0.51	0.43

```
## Result case 3
```

filtered	smoothed	viterbi
0.48	0.74	0.48

The results of the simulation and the calculations can be seen here. In all three simulations *smoothed* has the best prediction accuracy. The *filtered* method is in 2 cases better than *most probable path*. However, it is hard to say why *filtered* is better than *most probable path*. There is no rule for that and it is harder to explain mathematically. In task part 4 you can see that the *most probable path* has a better prediction accuracy than *filtered*.

The smoothing algorithm used forward and backward algorithm, so this method uses more information than the *forward* algorithm alone. Strictly speaking allows the algorithm to take into account any past observations of output for computing more accurate results.

The *most proable path*, on the other hand, does not always choose the next highest probability, but the next possible step with the highest probability. This restriction results in the *viterbi* not performing as well as the forward-backward algorithm.

Question 6

Is it true that the more observations you have the better you know where the robot is ?

Hint: You may want to compute the entropy of the filtered distributions with the function *entropy.empirical* of the package *entropy*.

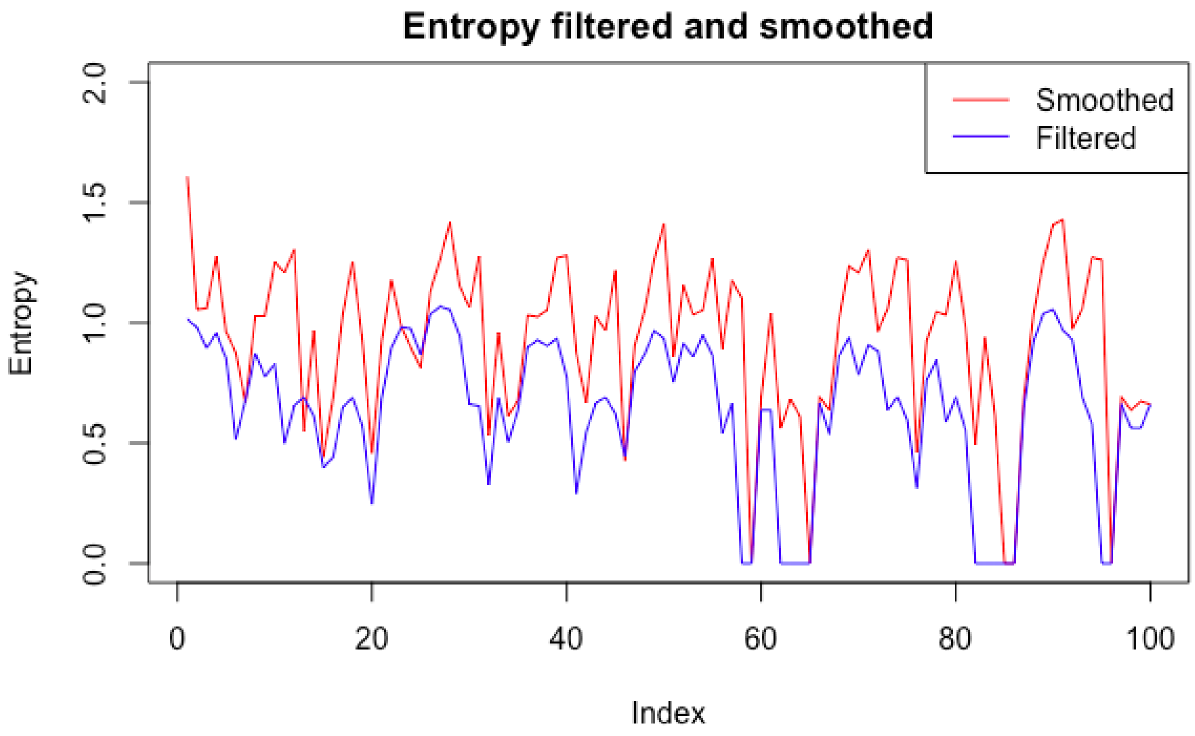
Result question 6

The empirical entropy estimator is a plug-in estimator: in the definition of the Shannon entropy the bin probabilities are replaced by the respective empirical frequencies.

The empirical entropy estimator is the maximum likelihood estimator. If there are many zero counts and the sample size is small it is very inefficient and also strongly biased.

```
#?entropy.empirical()

# use the result from question 1-2-3-4
# not results from 5
entropy_filtered = apply(X = forwar_method_norm, MARGIN = 2, FUN = entropy.empirical)
entropy_smoothed = apply(X = norm_smoothed, MARGIN = 2, FUN = entropy.empirical)
```



At first it´s important to explain what the entropy means, Entropy is a measure of uncertainty which goes up when the uncertainty is high and goes down when the uncertainty goes down. If the entropy is at 0, this means that there is virtually no uncertainty. This is the case several times with the filter method, less with the smoothing method. However, the question is whether increasing iterations makes us safer where the robot is. Due to the plot, this statement cannot be confirmed. For this a steadily sinking entropy would have to be present. However, even after 100 iterations it can still be seen that the entropy fluctuates strongly. This makes sense, because our problem case is stochastic, as we initialized at the transition probabilities. So desent matter which method we look at, the entropy will not convert to 0.

Question 7

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

I use the last state, which is the state 100, our prior inforamtion, to compute the prediction with the given transition probabilities.

Result question 7

```r
# Generate probabilities of the hidden state
# for time step 101

posterior = forwar_method_norm[,100] # The last information of the robot aka the prior
pred_101 = posterior %*% transprobs
```

```
## Probabilities of the hidden states for the time step 101:
```

```
##         [,1]
##  [1,] 0.0000
##  [2,] 0.1875
##  [3,] 0.5000
##  [4,] 0.3125
##  [5,] 0.0000
##  [6,] 0.0000
##  [7,] 0.0000
##  [8,] 0.0000
##  [9,] 0.0000
## [10,] 0.0000
```

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
# if packages is not installed, eval = T
install.packages("HMM")
install.packages("entropy")
# used libraries in this lab
library(HMM)
library(entropy)
library(ggplot2)
library(ggplot2)
rm(list = ls())
set.seed(12345)

#### we have to create a HMM based on the senario above
## in the package HMM is a function called initHMM
## for this we need to init following arguments:
# States - Vector with the names of the states.
# Symbols - Vector with the names of the symbols.
# startProbs - Vector with the starting probabilities of the states.
# transProbs - Stochastic matrix containing the transition
#              probabilities between the states. - Z
# emissionProbs - Stochastic matrix containing the emission probabilities of the states. - X

# States
states = seq(from = 1, to = 10, by = 1)

# Symbols
symbols = seq(from = 1, to = 10, by = 1)

# startProbs
startprobs = rep(0.1,10)

# transProbs
# there is a 0.5 stay and 0.5 move probabiliy
transprobs = matrix(0, nrow = 10, ncol = 10)
diag(transprobs) <- 0.5
diag(transprobs[,-1]) <- 0.5
transprobs[10,1] <- 0.5

# emissionProbs
emissionprobs = c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2,
                  0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2,
                  0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0,
                  0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0,
                  0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0,
                  0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0,
                  0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0,
                  0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2,
                  0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2,
                  0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)
emissionprobs = matrix(data = emissionprobs,
                          nrow = 10,
                          ncol = 10)


# we init all needed parameter to init an HMM
# column - & row - state
hmm <- initHMM(startProbs = startprobs,
               States = states,
               Symbols = symbols,
               transProbs = transprobs,
               emissionProbs = emissionprobs)
hmm
set.seed(12345)
# run simulations
simulation_nr = 100
simulation = simHMM(hmm = hmm,
                    length = simulation_nr)

# # extract the observations - X
# observations = simulation$observation
# # extract the hidden states - Z
# state_place = simulation$states

simulation
knitr::include_graphics("result_q2.png")
plot(x = 1:simulation_nr, y = states,
     type = "l", col = "red", ylim = c(0,12),
     xlab = "simulation", ylab = "states", main = "Roboter simulation")
lines(x = 1:simulation_nr, y = observation, type = "l", col = "blue")
legend(3, 12, legend=c("hidden states", "observations"),
       col=c("red", "blue"), lty=1, cex=0.7)
####### ggplot version
##### I dont know why the plot() creates a error message,
##### so I create the same plot via ggplot

ggplot() +
  geom_line(aes(x = 1:simulation_nr, y = simulation$states, color = "states")) +
  geom_line(aes(x = 1:simulation_nr, y = simulation$observation, color = "observation"))


######### Also ggplot does not work!!!!
##### I save every picutre, save it as picture in load it into the the report!!!!
# compute filtered probability distributions
forwar_method_log = forward(hmm = hmm, observation = simulation$observation)
# compute smoothed probability distributions
smoothed = posterior(hmm = hmm, observation = simulation$observation)
# Compute most probable path
viterbi_method = viterbi(hmm = hmm, observation = simulation$observation)

##### compute filtered proability distributions

# Remove the log-transformation
forwar_method = exp(forwar_method_log)
# Normalizing
```

```
forwar_method_norm = prop.table(forwar_method, margin = 2)
# margin over the columns -> 2
# Checking which probability in each column is the highest
forwar_method_prob = apply(forwar_method_norm, MARGIN = 2, FUN = which.max)
# margin over the columns -> 2
# Accuracy for the filter
accuracy_filter = sum(forwar_method_prob == simulation$states)/
  length(simulation$states)

#####  compute smoothed proability distributions

# Normalizing
norm_smoothed = prop.table(smoothed, margin = 2)
most_prob_smoothed = apply(norm_smoothed, MARGIN = 2, FUN = which.max)
# Accuracy for the smoothed
accuracy_smoothed = sum(most_prob_smoothed == simulation$states) /
  length(simulation$states)



#####  most probable path
accuracy_viterbi = sum(viterbi_method == simulation$states) /
  length(simulation$states)

cat("The accuracy of the following methods")
accruacy_table = data.frame("filtered" = accuracy_filter,
                            "smoothed" = accuracy_smoothed,
                            "viterbi" = accuracy_viterbi)
knitr::kable(accruacy_table)

knitr::include_graphics("result_q4.png")
plot(x = 1:simulation_nr, y = viterbi_method,
     type = "l", col = "blue",  ylim = c(0,13),
     xlab = "simulation", ylab = "states", main = "Predicted path")
lines(x =1:simulation_nr, y = forwar_method_prob, col =  "red")
lines(x =1:simulation_nr, y = most_prob_smoothed, col = "green")
legend(3, 13, legend=c("Viterbi", "Filtered", "Smoothed"),
       col=c("blue", "red", "green"), lty=1, cex=0.7)
######### Function
function_e3_e4 = function(hmm, observations, state_place){

  ################ Q3

  # compute filtered probability distributions
  forwar_method_log = forward(hmm = hmm, observation = observations)
  # compute smoothed probability distributions
  smoothed = posterior(hmm=hmm, observations)
  # Compute most probable path
  viterbi_method = viterbi(hmm = hmm, observation = observations)


  ############## Q4
  ##### compute filtered proability distributions
  # Remove the log-transformation
  forwar_method = exp(forwar_method_log)
  # Normalizing
  forwar_method_norm = prop.table(forwar_method, margin = 2) # margin over the columns -> 2
  # Checking which probability in each column is the highest
  forwar_method_prob = apply(forwar_method_norm, MARGIN = 2, FUN = which.max)
  # margin over the columns -> 2
  # Accuracy for the filter
  accuracy_filter = sum(forwar_method_prob == state_place) / length(state_place)


  #####  compute smoothed proability distributions

  # Normalizing
  norm_smoothed = prop.table(smoothed, margin = 2)
  most_prob_smoothed = apply(norm_smoothed, MARGIN = 2, FUN = which.max)
  # Accuracy for the smoothed
  accuracy_smoothed = sum(most_prob_smoothed == state_place) / length(state_place)


  #####  computemost probable path
  accuracy_viterbi = sum(viterbi_method == state_place) / length(state_place)


  ######## accruacy
  accruacy_table = data.frame("filtered" = accuracy_filter,
                              "smoothed" = accuracy_smoothed,
                              "viterbi" = accuracy_viterbi)

   # return all computed values
    resutls = list("forwar_method_log" = forwar_method_log,
                  "forwar_method" = forwar_method,
                  "forwar_method_norm" =forwar_method_norm,
                  "forwar_method_prob" = forwar_method_prob,
                  "accuracy_filter" = accuracy_filter,
                  "smoothed" =  smoothed,
                  "norm_smoothed" = norm_smoothed,
                  "most_prob_smoothed" = most_prob_smoothed,
                  "accuracy_smoothed" = accuracy_smoothed,
                  "viterbi_method" = viterbi_method,
                  "accuracy_viterbi" = accuracy_viterbi,
                  "accruacy_table" = accruacy_table)


}
# create a new simulation
# run simulations
simulation_nr_e5 = 100

### create different samlpe sets
## do this, by running the simulation with different seeds

# caste 1
```

```r
set.seed(123456)
simulation_e5_1 = simHMM(hmm = hmm, length = simulation_nr_e5)

# caste 2
set.seed(1234567)
simulation_e5_2 = simHMM(hmm = hmm, length = simulation_nr_e5)

# caste 3
set.seed(12345678)
simulation_e5_3 = simHMM(hmm = hmm, length = simulation_nr_e5)


# compute the results with the function for all three cases

# case 1
function_result_e5_1 = function_e3_e4(hmm = hmm,
                                      observations = simulation_e5_1$observation,
                                      state_place = simulation_e5_1$states)
cat("Result case 1")
knitr::kable(function_result_e5_1$accruacy_table)

# case 2
function_result_e5_2 = function_e3_e4(hmm = hmm,
                                      observations = simulation_e5_2$observation,
                                      state_place = simulation_e5_2$states)
cat("Result case 2")
knitr::kable(function_result_e5_2$accruacy_table)

# case 3
function_result_e5_3 = function_e3_e4(hmm = hmm,
                                      observations = simulation_e5_3$observation,
                                      state_place = simulation_e5_3$states)
cat("Result case 3")
knitr::kable(function_result_e5_3$accruacy_table)

###### Interpretaion of the results and question 5
#### Internet found in the interpreation

# - Wikipedia
#
# At each single observation in the sequence, probabilities to be used for calculations at the next obs
#
# This probability includes the forward probabilities covering all events up to time t as well as the b
# It should be noted, however, that the term "most probable state" is somewhat ambiguous. While the mos
#?entropy.empirical()

# use the result from question 1-2-3-4
# not results from 5
entropy_filtered = apply(X = forwar_method_norm, MARGIN = 2, FUN = entropy.empirical)
entropy_smoothed = apply(X = norm_smoothed, MARGIN = 2, FUN = entropy.empirical)
knitr::include_graphics("result_q6.png")

plot(entropy_filtered,main = "Entropy filtered and smoothed",
     type = "l", col = "red",
     ylab = "Entropy", ylim = c(0,2))
lines(entropy_smoothed, col = "blue")
legend(x = "topright", lty=1,
       legend = c("Smoothed","Filtered"),
       col = c("red","blue"))


# Generate probabilities of the hidden state
# for time step 101

posterior = forwar_method_norm[,100] # The last information of the robot aka the prior
pred_101 = posterior %*% transprobs

cat("Probabilities of the hidden states for the time step 101: ")
t(pred_101)
```