TDDE15 Lab2

William Bergekrans

2021-10

Question 1

The robot can initially be in 10 different positions, a uniform prior is therefore assumed for the initial state. Furthermore, for every time step it is known that the robot either stays or goes to the next sector with probability 0.5. It is not possible to observe the sector directly, which means that the actual position is a hidden variable. The observations available are GPS-readings of the robot's position that can be in [i-2, i+2] with equal probability.

The transmission matrix used is:

S	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S.1	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S.2	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0
S.3	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0
S.4	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0
S.5	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0
S.6	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0
S.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0
S.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5
S.9	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5

This means that for every state the probability is 0.5 to stay and 0.5 to go to the next state.

The emission probability matrix is:

S	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2
S.1	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2
S.2	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0
S.3	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0
S.4	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0
S.5	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0
S.6	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0
S.7	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2
S.8	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2
S.9	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2

This means that the observed state can be any of 5 states with equal probability.

```
# Code for question 1

states <- c("S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9", "S10")
symbols <- states

startProbs <- rep(0.1, 10) # Assume a uniform prior
```

```
transProbs <- t(as.matrix(data.frame(
  "S"= c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0),
  "S"= c(0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0),
  "S"= c(0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0),
  "S"= c(0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0),
  "S"= c(0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0),
  "S"= c(0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0),
  "S"= c(0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0),
  "S"= c(0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0),
  "S"= c(0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5),
  "S"= c(0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5)
)))

emissionProbs <- t(as.matrix(data.frame(
  "S"= c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2),
  "S"= c(0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2),
  "S"= c(0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0),
  "S"= c(0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0),
  "S"= c(0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0),
  "S"= c(0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0),
  "S"= c(0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0),
  "S"= c(0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2),
  "S"= c(0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2),
  "S"= c(0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)
)))

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

Question 2

With the generated Hidden Markov Model from question 1, I shall now simulate the network for 100 time steps.

```
# Code for question 2
set.seed(34952)
simulated_HMM <- simHMM(hmm, 100)
```

Question 3

First the hidden states from our sample are discarded, the hidden states are the sampled "true" states where the robot is located. The forward and backward probabilities are calculated using the functions forward() and backward() from the HMM package.

The filtered probability distribution is calculated with the calculated forward probabilities. The forward function generates the α values.

$$\text{filtered distribution: } p(z^t|x^{0:t}) = \frac{\alpha(z^t)}{\sum_{z^t} \alpha(z^t)}$$

The filtered distribution is used to answer how probable the different states are at time t, which is the last observed time step.

The smoothed probability distribution is calculated using both the forward and backward probabilities. The backward function generates the β values.

$$\text{filtered distribution: } p(z^t|x^{0:T}) = \frac{\alpha(z^t)\beta(z^t)}{\sum_{z^t}\alpha(z^t)\beta(z^t)}$$

The smoothed probabilities are used to answer how probable the states are at a given time point T which can be any of the observed time points. The smoothed distribution therefore looks back in time compared to the filtered one which always looks at what currently is.

The third algorithm that I use is the Viterbi algorithm which is used for calculating the path with the highest probability given our observations. For this task I also use the viterbi()-function in the HMM package.

```
# Code for question 3
observations <- simulated_HMM$observation

forward <- exp(forward(hmm, observations))
backward <- exp(backward(hmm, observations))

filtered <- matrix(rep(0,1000), 10, 100)
for (i in 1:100) {
  filtered[,i] <- forward[,i] / sum(forward[,i])
}

smoothing <- matrix(rep(0,1000), 10, 100)
for (i in 1:100) {
  smoothing[,i] <- (forward[,i] * backward[,i]) / (sum(forward[,i]*backward[,i]))
}

# The most probable path using the viterbi algorithm
probable_path <- viterbi(hmm, observations)

# colSums(filtered)
# colSums(smoothing)
# All columns add to 1, valid distributions!
```

Question 4

To calculate the accuracy I use the probability distributions calculated in the previous question. For each observations I assume that the model chooses the state which has the highest probability. I then compare the guessed state with the true state from the original data for both the smoothing and filtered algorithms. The predicted path generated by the Viterbi algorithm can be compared directly with the correct path as it is not a probability distribution.

The misclassification errors for the three different algorithms are:

Table 1: Misclassification errors, sim 1

Filtered algorithm	Smoothing algorithm	Viterbi algorithm
0.42	0.32	0.51

```
# Code for question 4

# Get the most likely state for each observation.
pred_filtered <- states[apply(filtered, 2, which.max)]
pred_smoothing <- states[apply(smoothing, 2, which.max)]

# Confusion matrices
```

```r
filtered_cm <- table(pred_filtered, simulated_HMM$states)
smoothing_cm <- table(pred_smoothing, simulated_HMM$states)

# Misclassification error function
misclassification <- function(predicted, true) {
  confusion_matrix <- table(predicted, true)
  error <- 1-(sum(diag(confusion_matrix)) / sum(confusion_matrix))
  return(error)
}

# Misclassification errors
filtered_error <- misclassification(simulated_HMM$states, pred_filtered)
smoothings_error <- misclassification(simulated_HMM$states, pred_smoothing)
viterbi_error <- misclassification(simulated_HMM$states, probable_path)

errors <- data.frame(
  filtered_error,
  smoothings_error,
  viterbi_error
)
```

Question 5

Here the task in question 4 is repeated with different simulated observation from the HMM.

The prediction errors for these simulation are:

Table 2: Misclassification errors, sim 2

Filtered algorithm	Smoothing algorithm	Viterbi algorithm
0.56	0.33	0.51

Even though the results are not exactly the same for the two simulations, the internal ranking between the three algorithms predictive ability is the same. The smoothing algorithm is clearly the best, followed by the filtered algorithm, and lastly the Viterbi algorithm in both our simulated cases.

I believe this is logical result because if we consider the differences between the filtered and smoothing algorithm, it is mainly that the filtered algorithm only use forward probabilities. What this means is that the filtered algorithm only looks in one direction compared to the smoothing algorithm which looks in two directions. If the filtered algorithm is used on a time step which is not the last, then a lot of information is dismissed that the smoothing algorithm use for making its prediction.

The viterbi does not return a predictive distribution, it returns a path that is viable and as optimal as possible. If the filtered algorithm returns a valid path this will be the same as the viterbi path. Due to this extra criteria that viterbi has it will not perform as good.

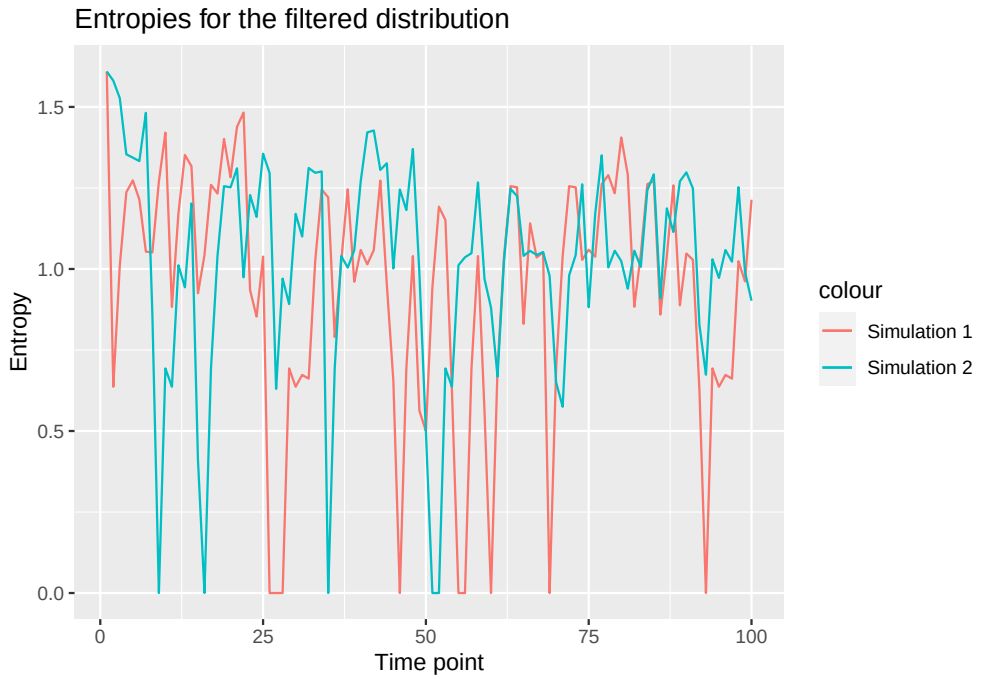
```r
# Code for question 5
set.seed(23458)
new_simulation <- simHMM(hmm, 100)

new_observations <- new_simulation$observation

forward <- exp(forward(hmm, new_observations))
backward <- exp(backward(hmm, new_observations))
```

```r
new_filtered <- matrix(rep(0,1000), 10, 100)
for (i in 1:100) {
  new_filtered[,i] <- forward[,i] / sum(forward[,i])
}

new_smoothing <- matrix(rep(0,1000), 10, 100)
for (i in 1:100) {
  new_smoothing[,i] <- (forward[,i] * backward[,i]) / (sum(forward[,i]*backward[,i]))
}

# The most probable path using the viterbi algorithm
probable_path <- viterbi(hmm, new_observations)

pred_filtered <- states[apply(new_filtered, 2, which.max)]
pred_smoothing <- states[apply(new_smoothing, 2, which.max)]

# Misclassification errors
filtered_error <- misclassification(new_simulation$states, pred_filtered)
smoothings_error <- misclassification(new_simulation$states, pred_smoothing)
viterbi_error <- misclassification(new_simulation$states, probable_path)

errors <- data.frame(
  filtered_error,
  smoothings_error,
  viterbi_error
)
```

Question 6

To know where the robot currently is (at the latest time step) the filtered method is used. The question to answer is whether the method is more accurate if there are more observations, i.e. more time steps before the position is estimated. To explore this I calculate the entropy for the filtered distribution. The entropy expresses how certain the model is of its decision, the density of the probability mass. If the entropy is 0 all the probability mass is in one point and if it is uniform/ larger the probability mass is evenly distribution across all possible values.

The entropy for the filtered distributions of the two simulations are:



There are plenty of examples in the graphs very the entropy is lower for earlier time steps than later. This means that more information does not make the model more certain of the robot's position. Because the robot moves around new noise and uncertainties are always added to the environment. If the robot was static the position would be more certain given more observations.

```
# Code for question 6
entropy_sim1 <- integer(dim(filtered)[2])
entropy_sim2 <- integer(dim(new_filtered)[2])

for (i in 1:dim(filtered)[2]) {
  entropy_sim1[i] <- entropy.empirical(filtered[,i])
  entropy_sim2[i] <- entropy.empirical(new_filtered[,i])
}

df <- data.frame(
  "sim1" = entropy_sim1,
  "sim2" = entropy_sim2
)

ggplot(df, aes(x = 1:100)) +
  geom_line(aes(y=sim1, col = "Simulation 1")) +
  geom_line(aes(y=sim2, col = "Simulation 2")) +
  xlab("Time point") +
  ylab("Entropy") +
  ggtitle("Entropies for the filtered distribution")
```

Question 7

Given the samples generated in the first simulation I shall compute the probabilities of the hidden states for time step 101, which is one time step into the future. Because of the network structure of HMM a hidden state at T+1 is independent from all states at T-1 and below given T, i.e. the hidden state at T+1 depends only on the state at T. In order to calculate the probabilities the observed values in the 100th time step

should be multiplied with the transition model.

Table 3: Probabilities for the hidden variables, T = 101

0.3425926	0.3518519	0.1574074	0.0277778	0	0	0	0	0	0.1203704

```
# Code for question 7
t101 <- filtered[,100] %*% transProbs
kbl(t101, caption = "Probabilities for the hidden variables, T = 101", position = "h")
```