732A96 Advanced Machine Learning

LAB 2: HIDDEN MARKOV MODELS

Johannes Hedström

STIMA
Department of Computer and Information Science
Linköpings universitet

2024-10-21

Contents

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

Description

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.

```r
# States
states <- as.character(1:10)
# Movements for the robot
sym <- paste0('State_',states)
# starting probabilities, equal for all states
startprobs <- rep(1/10,10)

# Transitions probabilities 1/2 probability to stay or change

t_prob <- matrix(0,nrow=10,ncol=10) # empty matrix
diag(t_prob) <- 0.5  # 0.5 to stay
diag(t_prob[,-1]) <- 0.5 # 0.5 to move to NEXT sector
t_prob[10,1] <- 0.5 # left bottom corner(step from 10 to 1)

# emissionprobs
e_prob <- 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

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

```
hmm <-initHMM(states,sym, startprobs,transProbs=t_prob,e_prob)

rownames(t_prob) <- sym
colnames(t_prob) <- states
rownames(e_prob) <- sym
colnames(e_prob) <- states
knitr::kable(t_prob, caption='Transitions probabilities')
```

Table 1: Transitions probabilities

	1	2	3	4	5	6	7	8	9	10
State_1	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
State_2	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
State_3	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0
State_4	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0
State_5	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0
State_6	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0
State_7	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0
State_8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0
State_9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5
State_10	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5

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

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

	1	2	3	4	5	6	7	8	9	10
State_1	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2
State_2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2
State_3	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0
State_4	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0
State_5	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0
State_6	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0
State_7	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0
State_8	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2
State_9	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2
State_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.

```
sims <- simHMM(hmm,100) # simulate 100 steps

sim_state <- sims$states
sim_obs <- sims$observation
```

```
knitr::kable(sim_obs[1:10],caption='First ten simulated observations')
```

Table 3: First ten simulated observations

x
State_7
State_10
State_8
State_10
State_2
State_3
State_10
State_3
State_4
State_4

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.

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

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

```
# probabilities are given on the logaritmic scale and must therefore be transformed
# with exp
# forward for filter
alpha <- exp(forward(hmm,sim_obs))

# backwards for smoothed
# alpha*beta/ sum of each columns to make them proportional
beta <- exp(backward(hmm, sim_obs))
smooth_pd <-  (alpha*beta)
# Viterbi algorithm for the most likely path
most_prob_path <- viterbi(hmm,sim_obs)
```

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.

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 element wise 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

```
# picking out the highest probability from the filter pd
filt_path <- as.character(unname(apply(prop.table(alpha,2),2,which.max)))

# picking out the highest probability from the smoothed pd
smooth_path <-  as.character(unname(apply(prop.table(smooth_pd,2),2,which.max)))

# number of correct devided by total = mean
acc_f <- mean(filt_path==sim_state)
acc_s <- mean(smooth_path==sim_state)
acc_mpp <- mean(most_prob_path==sim_state)

# dataframe with the accuracy
df_acc <- data.frame('Filter'=acc_f, 'Smoothed'=acc_s, 'Viterbi' = acc_mpp)

knitr::kable(df_acc, caption='Percentage of true hidden states for each method ')
```

Table 4: Percentage of true hidden states for each method

Filter	Smoothed	Viterbi
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 ?

```
# a function to run simulation, filer, smooth etc
# returns a dataframe with the accuracy for each probability distribution
# and the most probable path
hmm_sims <- function(nr_of_sims){

  sims <- simHMM(hmm,nr_of_sims) # simulate 100 steps
  sim_obs <- sims$observation
  sim_state <- sims$states

  alpha <- exp(forward(hmm,sim_obs))
  # backwards for smoothed
  beta <- exp(backward(hmm, sim_obs))
  # alpha*beta/ sum over states alpha*beta
  smooth_pd <-  (alpha*beta)

  # Viterbi algorithm for the most likely path
```

```r
  most_prob_path <- viterbi(hmm,sim_obs)

  # picking out the highest probability from the filter pd
  filt_path <- as.character(unname(apply(prop.table(alpha,2),2,which.max)))
  # picking out the highest probability from the smoothed pd
  smooth_path <-  as.character(unname(apply(prop.table(smooth_pd,2),2,which.max)))

  # number of correct devided by total(100
  acc_f <- sum(filt_path==sim_state)/100
  acc_s <- sum(smooth_path==sim_state)/100
  acc_mpp <- sum(most_prob_path==sim_state)/100

  # craeting a dataframe with each accuracy
  df_acc <- data.frame('Filter'=acc_f, 'Smoothed'=acc_s, 'Viterbi' = acc_mpp)
  df_acc # returning the dataframe
}

rownames(df_acc) <-paste0('Simulation: ',1)
# running the simulation 5 times and saving the results in the dataframe from before
for(i in 2:20){

  df_acc <- rbind(df_acc,hmm_sims(100))
  rownames(df_acc)[i] <-paste0('Simulation: ',i)
}


knitr::kable(head(df_acc,5), caption='First 5 simulations')
```

Table 5: First 5 simulations

	Filter	Smoothed	Viterbi
Simulation: 1	0.53	0.74	0.56
Simulation: 2	0.46	0.68	0.61
Simulation: 3	0.49	0.77	0.65
Simulation: 4	0.49	0.58	0.56
Simulation: 5	0.60	0.79	0.65

```r
means <- t(colMeans(df_acc)) # mean for each column
knitr::kable(means, caption='Mean accuracy for the distributions (20 simulations)')
```

Table 6: Mean accuracy for the distributions (20 simulations)

Filter	Smoothed	Viterbi
0.5045	0.672	0.512

The smoothed distribution provides the highest accuracy compared to other distributions. This is expected because the filtering distribution only uses information available up to time t for its estimate, whereas smoothing

incorporates both the information up to t and all subsequent information. Essentially, smoothing uses all available evidence and operates offline, unlike filtering, which is an online process.

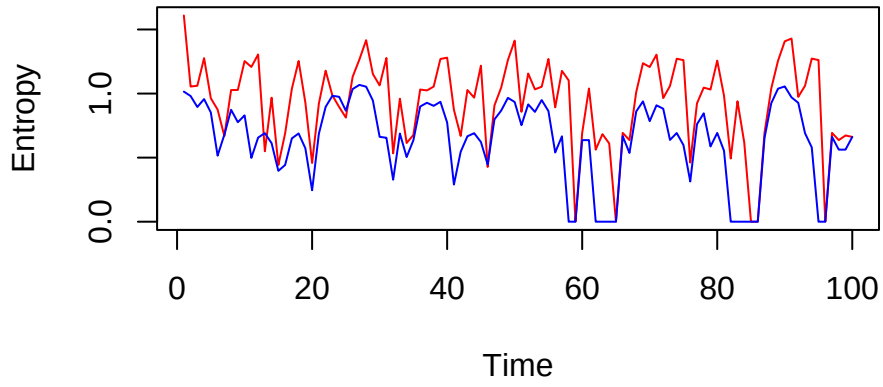
The viterbi algorithm always chooses the most probable path, not the next most probable step, which mean that it can chose steps with lower probability within the whole path.

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

```
# entropy for smooth_norm
ent_sm <- apply(X=(alpha*beta)/colSums(alpha*beta), MARGIN=2, FUN=entropy.empirical)
# entropy for filter_pd
ent_f <- apply(X=prop.table(alpha,2), MARGIN=2, FUN=entropy.empirical)


plot(y=ent_f, x=1:100, type='l',col='red',ylab='Entropy', xlab='Time')
lines(y=ent_sm,x=1:100, col='blue' )
```



Entropy values can be interpreted as how certain we are of the prediction, a value of 0 means that we are 100% sure about the next step. We can see that entropy for the smooth distribution(Blue line) have lower values than for the filtered(Red line), meaning we are more certain of the predictions from that distribution.

There is no clear trend over time, even though we get some entropy values as 0 around time 60 and time 85, but other than those 2 valleys we seem to have close to the same pattern as in the beginning.

7 Question 7

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

```
# smoothed probs for step 101
step101 <-((alpha*beta)/colSums(alpha*beta))[,100] %*% t_prob

step101<-t(step101)# rotating
rownames(step101) <- sym

knitr::kable(step101, caption='Probabilities for step 101')
```

Table 7: Probabilities for 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 likely state for step 101 is state 3 for the smoothed distribution and it will be the same for the filter distribution as when you're predicting with the smooth distribution, you cant utilize observations after that as its unknown.