Advanced Machine Learning Lab 2

Mikael Montén

2024-10-22

```r
library("tidyverse")
library("HMM")
library("entropy")
```

Questions

Model the behavior of a robot that walks in a 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

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

```r
# transition probabilities matrix
# every diagonal has a 50% chance of occurring as the robot might stay
# the shifted to the right diagonal contains the other 50% chance
# if the robot were to step away from state 10 it would return to 1
trans = diag(10)*0.5
for(j in 2:10){
  for(i in j-1){
    trans[i,j] = 0.5
  }
}
trans[10,1] = 0.5

# emission probabilities matrix
# i +-2 => 5 possible states i.e 0.2% prob in each
emis = matrix(0,10,10)
for(i in 3:8){
  for(j in (i-2):(i+2)){
    emis[i,j] <- 0.2
  }
}
emis[9,c(7,8,9,10,1)] <- 0.2
emis[10,c(8,9,10,1,2)] <- 0.2
emis[1,c(9,10,1,2,3)] <- 0.2
emis[2,c(10,1,2,3,4)] <- 0.2

# start probabilities are 10% for each state as we have 10 states
```

```r
start <- rep(0.1,10)

# states and symbols are identical in this case so only using numbers to reference them
states <- 1:10
symbols <- 1:10
```

Table 1: Transition matrix

	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8	State 9	State 10
Step 1	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Step 2	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Step 3	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0
Step 4	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0
Step 5	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0
Step 6	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0
Step 7	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.0
Step 8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0
Step 9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5
Step 10	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5

Table 2: Emission matrix

	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8	State 9	State 10
Step 1	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2
Step 2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2
Step 3	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0
Step 4	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0
Step 5	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0	0.0
Step 6	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0	0.0
Step 7	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	0.0
Step 8	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2
Step 9	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2
Step 10	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2

```r
# initialize hmm
hmm <- initHMM(states,symbols,startProbs = start,transProbs = trans,emissionProbs = emis)
```

2

Simulate the HMM for 100 time steps

```r
set.seed(12345)
simulation <- simHMM(hmm, 100) # simulate the HMM for 100 time steps
```

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.

```r
obs <- simulation$observation # discard hidden states and use only remaining observations

# computations for forward and backward algorithm, anti-logging result from forward
alpha <- exp(forward(hmm, obs))
beta <- exp(backward(hmm, obs))

# marginalized & normalized distributions for each time point
filter <- prop.table(alpha,2)
smooth <- prop.table(alpha*beta,2)

# most probable path
viterbi <- viterbi(hmm,obs)
```

4

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

```r
# most probable states
filter_pred <- as.vector(apply(filter,2,which.max))
smooth_pred <- as.vector(apply(smooth,2,which.max))

# accuracies calculate from confusion matrices for each method
filter_conf <- sum(diag(table(filter_pred,simulation$states)))/sum(table(filter_pred,simulation$states)
smooth_conf <- sum(diag(table(smooth_pred,simulation$states)))/sum(table(smooth_pred,simulation$states)
viterbi_conf <- sum(diag(table(viterbi,simulation$states)))/sum(table(viterbi,simulation$states))

# append accuracies into table
accuracies = data.frame(filter_conf,smooth_conf,viterbi_conf)
```

Table 3: Accuracies for the different methods

	Filtered	Smoothed	Viterbi
Accuracies	0.53	0.74	0.56

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
hmm_simulations <- function(hmm, n){
  set.seed(12345)
  filter_conf <- c()
  smooth_conf <- c()
  viterbi_conf <- c()

  for(i in 1:n){ # n = amount of models to simulate for
    # same functions as previously used but in a loop to easy simulate several models
    simulation <- simHMM(hmm, 100)
    obs <- simulation$observation
    alpha <- exp(forward(hmm, obs))
    beta <- exp(backward(hmm, obs))
    filters <- prop.table(alpha,2)
    smooths <- prop.table(alpha*beta,2)
    viterbis <- viterbi(hmm,obs)
    filter_pred <- as.vector(apply(filters,2,which.max))
    smooth_pred <- as.vector(apply(smooths,2,which.max))
    filter_conf[i] <- sum(diag(table(filter_pred,simulation$states)))/sum(table(filter_pred,simulation$
    smooth_conf[i] <- sum(diag(table(smooth_pred,simulation$states)))/sum(table(smooth_pred,simulation$
    viterbi_conf[i] <- sum(diag(table(viterbis,simulation$states)))/sum(table(viterbi,simulation$states
  }

  # calculate mean of all accuracies computed
  accuracies <- data.frame("Filtered"=mean(filter_conf),"Smoothed"=mean(smooth_conf),"Viterbi"=mean(vit
  return(accuracies)
}

summed_acc <- hmm_simulations(hmm, 50) # 50 simulations ran
```

Table 4: Mean accuracies for the different methods simulated 50 times

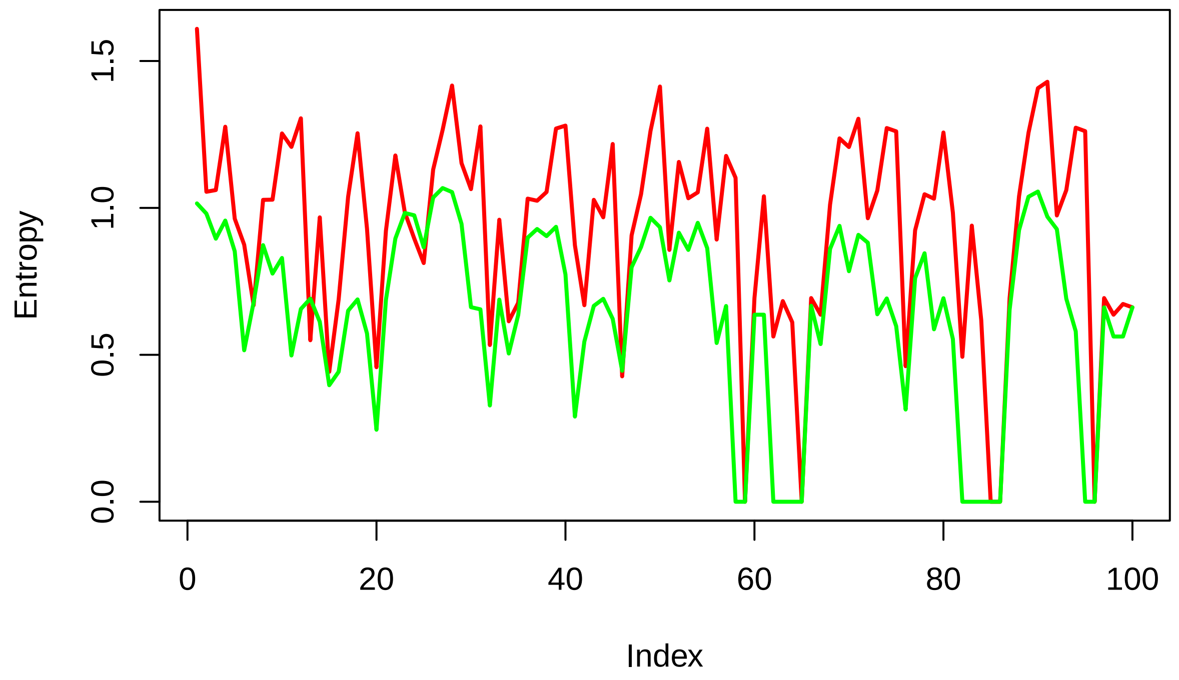
	Filtered	Smoothed	Viterbi
Accuracy	0.527	0.6838	0.5134

The filtered distribution provides the probability of being in each state at each time step, given the observation up to that time step, which is obtained through the forward algorithm making use of past observations. The smoothed distribution, obtained through the forward-backward algorithm, uses observations from all time steps to calculate the probability of being in each state at each time step. This effectively smooths the estimates by incorporating future information. The Viterbi algorithm on the other hand, seeks the single most probable sequence of hidden states given the observations. Comparing smoothed distribution to Viterbi, the latter is constrained to a single path and might miss states that were individually most likely. As the smoothed distribution provides the most likely state for each time step, this usually gives a better accuracy than Viterbi. The smoothed distribution is in general more accurate than filtered distributions because they incorporate more information i.e. at time t it considers observations $0:T$, whereas the filtered uses $0:t$.

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?

```
entropy <- matrix(0,100,2)
entropy[,1] <- as.vector(apply(filter,2,entropy.empirical)) # calculate entropy for filtered dist
entropy[,2] <- as.vector(apply(smooth,2,entropy.empirical)) # calculate entropy for smoothed dist
```



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. The figure shows some form of pattern indicating that the entropy is decreasing as after around 50 timesteps both the smoothed and filtered distribution hit 0 entropy. However, as there is no general decrease in trend and the low points of entropy are immediately followed by higher entropy, it's uncertain and nothing can really be assumed regarding the effect of time on entropy. Therefore it is not always true that the later in time, the better you know where the robot is.

7

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

```
# use last sampled smoothed dist probabilities to generate new states using transition matrix
pred_step <- smooth[,100] %*% trans
```

	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8	State 9	State 10
Step 101	0	0.1875	0.5	0.3125	0	0	0	0	0	0

The most likely state in step 101 is state 3 with 0.5 probability.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning=FALSE, fig.align = 'center')
set.seed(12345)
library("tidyverse")
library("HMM")
library("entropy")
# transition probabilities matrix
# every diagonal has a 50% chance of occurring as the robot might stay
# the shifted to the right diagonal contains the other 50% chance
# if the robot were to step away from state 10 it would return to 1
trans = diag(10)*0.5
for(j in 2:10){
  for(i in j-1){
    trans[i,j] = 0.5
  }
}
trans[10,1] = 0.5

# emission probabilities matrix
# i +-2 => 5 possible states i.e 0.2% prob in each
emis = matrix(0,10,10)
for(i in 3:8){
  for(j in (i-2):(i+2)){
    emis[i,j] <- 0.2
  }
}
emis[9,c(7,8,9,10,1)] <- 0.2
emis[10,c(8,9,10,1,2)] <- 0.2
emis[1,c(9,10,1,2,3)] <- 0.2
emis[2,c(10,1,2,3,4)] <- 0.2

# start probabilities are 10% for each state as we have 10 states
start <- rep(0.1,10)

# states and symbols are identical in this case so only using numbers to reference them
states <- 1:10
symbols <- 1:10
colnames(trans) <- paste0("State ",1:10)
rownames(trans) <- paste0("Step ",1:10)
knitr::kable(trans,caption="Transition matrix")
colnames(emis) <- paste0("State ",1:10)
rownames(emis) <- paste0("Step ",1:10)
knitr::kable(emis,caption="Emission matrix")
# initialize hmm
hmm <- initHMM(states,symbols,startProbs = start,transProbs = trans,emissionProbs = emis)
set.seed(12345)
simulation <- simHMM(hmm, 100) # simulate the HMM for 100 time steps
obs <- simulation$observation # discard hidden states and use only remaining observations

# computations for forward and backward algorithm, anti-logging result from forward
alpha <- exp(forward(hmm, obs))
beta <- exp(backward(hmm, obs))
```

```r
# marginalized & normalized distributions for each time point
filter <- prop.table(alpha,2)
smooth <- prop.table(alpha*beta,2)

# most probable path
viterbi <- viterbi(hmm,obs)
# most probable states
filter_pred <- as.vector(apply(filter,2,which.max))
smooth_pred <- as.vector(apply(smooth,2,which.max))

# accuracies calculate from confusion matrices for each method
filter_conf <- sum(diag(table(filter_pred,simulation$states)))/sum(table(filter_pred,simulation$states)
smooth_conf <- sum(diag(table(smooth_pred,simulation$states)))/sum(table(smooth_pred,simulation$states)
viterbi_conf <- sum(diag(table(viterbi,simulation$states)))/sum(table(viterbi,simulation$states))

# append accuracies into table
accuracies = data.frame(filter_conf,smooth_conf,viterbi_conf)
colnames(accuracies)=c("Filtered","Smoothed","Viterbi")
rownames(accuracies)="Accuracies"

knitr::kable(accuracies,caption="Accuracies for the different methods")
hmm_simulations <- function(hmm, n){
  set.seed(12345)
  filter_conf <- c()
  smooth_conf <- c()
  viterbi_conf <- c()

  for(i in 1:n){ # n = amount of models to simulate for
    # same functions as previously used but in a loop to easy simulate several models
    simulation <- simHMM(hmm, 100)
    obs <- simulation$observation
    alpha <- exp(forward(hmm, obs))
    beta <- exp(backward(hmm, obs))
    filters <- prop.table(alpha,2)
    smooths <- prop.table(alpha*beta,2)
    viterbis <- viterbi(hmm,obs)
    filter_pred <- as.vector(apply(filters,2,which.max))
    smooth_pred <- as.vector(apply(smooths,2,which.max))
    filter_conf[i] <- sum(diag(table(filter_pred,simulation$states)))/sum(table(filter_pred,simulation$
    smooth_conf[i] <- sum(diag(table(smooth_pred,simulation$states)))/sum(table(smooth_pred,simulation$
    viterbi_conf[i] <- sum(diag(table(viterbis,simulation$states)))/sum(table(viterbi,simulation$states
  }

  # calculate mean of all accuracies computed
  accuracies <- data.frame("Filtered"=mean(filter_conf),"Smoothed"=mean(smooth_conf),"Viterbi"=mean(vit
  return(accuracies)
}

summed_acc <- hmm_simulations(hmm, 50) # 50 simulations ran
rownames(summed_acc) <- c("Accuracy")
knitr::kable(summed_acc,caption="Mean accuracies for the different methods simulated 50 times",row.name
entropy <- matrix(0,100,2)
entropy[,1] <- as.vector(apply(filter,2,entropy.empirical)) # calculate entropy for filtered dist
```

```r
entropy[,2] <- as.vector(apply(smooth,2,entropy.empirical)) # calculate entropy for smoothed dist
plot(entropy[,1], type="l",col="red",lwd=2,ylab="Entropy");lines(entropy[,2],type="l",col="green",lwd=2
# use last sampled smoothed dist probabilities to generate new states using transition matrix
pred_step <- smooth[,100] %*% trans
rownames(pred_step) <- c("Step 101")
colnames(pred_step) <- paste0("State ",1:10)
knitr::kable(pred_step,digits=5)
```