Adv. ML. Lab 2

Simon Jorstedt

2024-09-23

Problem 1

In this lab we will construct and simulate a Hidden Markov Model where we imagine an agent walking along a cycle of states such that the agent in each time step will stay or move on to the next step with equal probabilities. There are ten states, so the agent can move from state 1 to 2, 2 to 3, etc, and from state 10 to 1. The hidden states in the model will represent the true position of the agent, while the emissions will be randomly selected from the set $\{i-2, i-1, i, i+1, i+2\}$ when the agent is in state i. For the simulations, we let the agent start in a uniformly randomly selected state. Below we create the programmatic setup such as the transition probability matrix, the emission probability matrix and the HMM. The states are the integers $1, \cdots, 10$, and in this case so are the emission symbols.

```
library(HMM)

startProbs <- rep(0.1, 10)

transProbs <- matrix(c(c(0.5, 0.5,  0,   0,   0,   0,   0,   0,   0,     0),
                       c(0,   0.5,  0.5, 0,   0,   0,   0,   0,   0,     0),
                       c(0,   0,    0.5, 0.5, 0,   0,   0,   0,   0,     0),
                       c(0,   0,    0,   0.5, 0.5, 0,   0,   0,   0,     0),
                       c(0,   0,    0,   0,   0.5, 0.5, 0,   0,   0,     0),
                       c(0,   0,    0,   0,   0,   0.5, 0.5, 0,   0,     0),
                       c(0,   0,    0,   0,   0,   0,   0.5, 0.5, 0,     0),
                       c(0,   0,    0,   0,   0,   0,   0,   0.5, 0.5,   0),
                       c(0,   0,    0,   0,   0,   0,   0,   0,   0.5,   0.5),
                       c(0.5, 0,    0,   0,   0,   0,   0,   0,   0,     0.5)
                       ), nrow = 10, ncol = 10, byrow = TRUE)

emissionProbs <- matrix(c(c(0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2),
                          c(0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2),
                          c(0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0),
                          c(0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0),
                          c(0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0),
                          c(0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0),
                          c(0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0),
                          c(0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2),
                          c(0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2),
                          c(0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2)
                          ), nrow = 10, ncol = 10, byrow = TRUE)

# Initialise a HMM model
HMM_1 <- initHMM(States = 1:10,
```

```
                  Symbols = 1:10,
                  startProbs = startProbs,
                  transProbs = transProbs,
                  emissionProbs = emissionProbs)
```

Problem 2

Now we shall simulate the HMM that we initiated previously. This comes down to just running the `simHMM` function.

```
set.seed(322347)
HMM_sim1 <- simHMM(HMM_1, length = 100)
HMM_sim2 <- simHMM(HMM_1, length = 100)
HMM_sim3 <- simHMM(HMM_1, length = 100)
HMM_sim4 <- simHMM(HMM_1, length = 100)
```

Problem 3

Now we shall use our emission observations above to compute the smoothed and filtered probability distributions using `HMM::backward` and `HMM:forward` respectively. We will also compute the most probable path using the Viterbi algorithm. We create a custom function to calculate the most probable "filtered" and "smoothed" states.

```
likeliest_states <- function(HMM, Simulation){
  alphas <- exp(forward(HMM, observation = Simulation$observation))
  betas <- exp(backward(HMM, observation = Simulation$observation))

  filtered_probabilities <- prop.table(alphas, margin=2)
  smoothed_probabilities <- prop.table(alphas*betas, margin=2)

  filtered_states <- apply(filtered_probabilities, FUN=which.max, MARGIN=2)
  smoothed_states <- apply(smoothed_probabilities, FUN=which.max, MARGIN=2)

  return(list(filtered_states, smoothed_states))
}
```

```
# Find the most likely state for each time step, from the filtered and smoothed
# probabilities.
likely_states_1 <- likeliest_states(HMM_1, HMM_sim1)

filtered_states_1 <- likely_states_1[[2]]
smoothed_states_1 <- likely_states_1[[1]]

viterbi_run1 <- HMM::viterbi(HMM_1, HMM_sim1$observation)
```

Problem 4 and 5

Now we compute the accuracy of filtered and smoothed probability distributions and of the most probable path. We find that the smoothed distribution is consistently more accurate than the filtered distribution,

which makes sense, since those probabilities are calculated with more knowledge (of all emissions).

The viterbi algorithm essentially works iteratively up from the beginning of the observations, which means that when estimating the true state at a particular time step, it does not use information about the future observed states, even though this is available. Again, this motivates why the smoothing probabilities turn out to be more accurate as state predictors.

```
# Simulation 1
likely_states_1 <- likeliest_states(HMM_1, HMM_sim1)
cat("Simulation 1 accuracies:\n",
    "Filtered accuracy sim 1: ",  mean(HMM_sim1$states == likely_states_1[[1]]), "\n",
    "Smoothed accuracy sim 1: ",  mean(HMM_sim1$states == likely_states_1[[2]]), "\n",
    "Viterbi  accuracy sim 1: ",  mean(HMM_sim1$states == viterbi(HMM_1, HMM_sim1$observation)))
```

```
## Simulation 1 accuracies:
##  Filtered accuracy sim 1:  0.5
##  Smoothed accuracy sim 1:  0.73
##  Viterbi  accuracy sim 1:  0.5
```

```
# Simulation 2
likely_states_2 <- likeliest_states(HMM_1, HMM_sim2)
cat("Simulation 2 accuracies:\n",
    "Filtered accuracy sim 2: ",  mean(HMM_sim2$states == likely_states_2[[1]]), "\n",
    "Smoothed accuracy sim 2: ",  mean(HMM_sim2$states == likely_states_2[[2]]), "\n",
    "Viterbi  accuracy sim 2: ",  mean(HMM_sim2$states == viterbi(HMM_1, HMM_sim2$observation)))
```

```
## Simulation 2 accuracies:
##  Filtered accuracy sim 2:  0.56
##  Smoothed accuracy sim 2:  0.73
##  Viterbi  accuracy sim 2:  0.52
```

```
# Simulation 3
likely_states_3 <- likeliest_states(HMM_1, HMM_sim3)
cat("Simulation 3 accuracies:\n",
    "Filtered accuracy sim 3: ",  mean(HMM_sim3$states == likely_states_3[[1]]), "\n",
    "Smoothed accuracy sim 3: ",  mean(HMM_sim3$states == likely_states_3[[2]]), "\n",
    "Viterbi  accuracy sim 3: ",  mean(HMM_sim3$states == viterbi(HMM_1, HMM_sim3$observation)))
```

```
## Simulation 3 accuracies:
##  Filtered accuracy sim 3:  0.54
##  Smoothed accuracy sim 3:  0.66
##  Viterbi  accuracy sim 3:  0.49
```

Problem 6

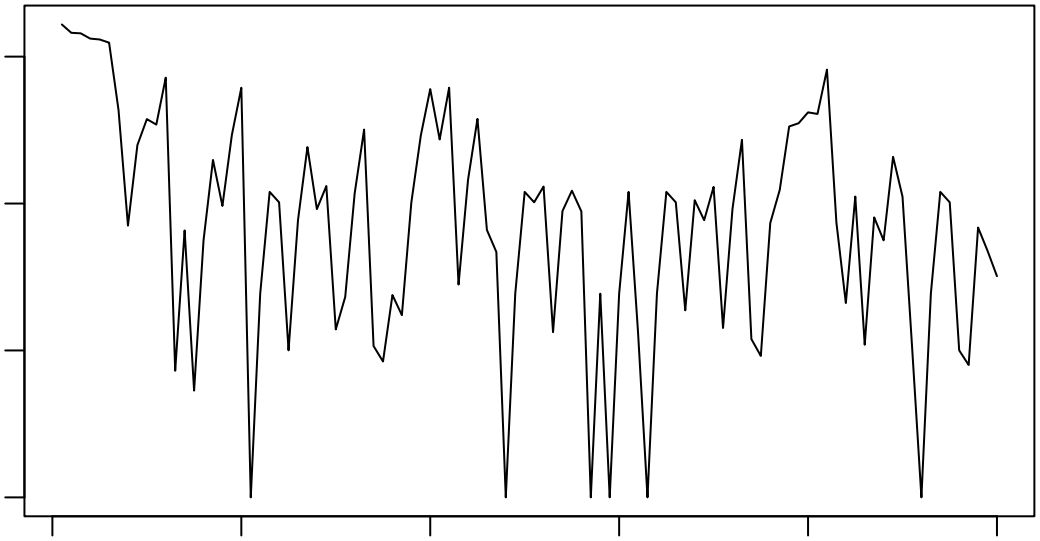
It should in general be true that over time, more observations lead to an increase in accuracy of the estimate of the robots location. In Figure 1 below we see that while there is a slight decrease in the beginning, and some dips into low entropy, the entropy appears to stay around 0.8. This is likely because of the randomness in the system which will always remain.

```r
library(entropy)

# Compute the filtered probabilities of simulation run 2
alphas <- exp(forward(HMM_1, observation = HMM_sim2$observation))
filtered_probabilities <- prop.table(alphas, margin=2)

# Compute entropy of the filtered probabilities
filter_entropy <- c()
for (i in 1:length(filtered_probabilities[1,])){
  filter_entropy <- c(filter_entropy, entropy.empirical(filtered_probabilities[,i]))
}

# Plot entropy over time
plot(filter_entropy, type="l", xlab="Time index", ylab="Entropy",
     main="Fig 1: Entropy over time")
```



Problem 7

Now we consider one of the samples above, and we estimate the probabilities of being in each of the ten states one time step in the future, given all the observed emissions of that sample.

```r
# p(z^(t+1) | x^(1:t))
probabilities_z101 <- list()

for (state in 1:10){
  # The next state is the state that the agent potentially will jump to.
  next_state <- if (state == 10) 1 else state+1

  if (state == 1) cat("Probabilities of being in state z at time 101, given all the observed states up t

  # Calculate probability of being in state i in next time step
  P_staying <- filtered_probabilities[state,100] * 0.5
  P_moving <- filtered_probabilities[next_state,100] * 0.5
  prob_of_state <- P_staying + P_moving
```

```r
  cat("State", state, ":", prob_of_state, "\n")

  # Save the probabilities
  probabilities_z101[[paste0("State", state)]] <- prob_of_state
}
```

```
## Probabilities of being in state z at time 101, given all the observed states up to time 100
## State 1 : 0.3333333
## State 2 : 0
## State 3 : 0
## State 4 : 0
## State 5 : 0
## State 6 : 0
## State 7 : 0
## State 8 : 0.01851852
## State 9 : 0.1666667
## State 10 : 0.4814815
```