Advanced Machine Learning

Lab 2: Hidden Markov Models

Jakob Berggren

2023-09-18

Scenario

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

Question 1 asks to build a Hidden Markov Model (HMM) for the scenario described above. To do this, I first specify the different states, one for each sector that the robot can be in. There are also 10 symbols.

```r
states <- 1:10 # Each Sector represent a state
symbols <- 1:10
```

Then we create the transition and emission probability matrices according to the above scenario. The robot decides with equal probability to stay in the current sector or move to the next one, which results in a transition matrix as such:

```r
# Transition probabilities as described.
transition_probs <- matrix(c(0.5,0.5,0,0,0,0,0,0,0,0,
                             0,0.5,0.5,0,0,0,0,0,0,0,
                             0,0,0.5,0.5,0,0,0,0,0,0,
                             0,0,0,0.5,0.5,0,0,0,0,0,
                             0,0,0,0,0.5,0.5,0,0,0,0,
                             0,0,0,0,0,0.5,0.5,0,0,0,
                             0,0,0,0,0,0,0.5,0.5,0,0,
                             0,0,0,0,0,0,0,0.5,0.5,0,
                             0,0,0,0,0,0,0,0,0.5,0.5,
                             0.5,0,0,0,0,0,0,0,0,0.5),
                           nrow = length(states),
                           ncol = length(states),
                           byrow = TRUE)
```

As stated, the tracking device is not very accurate, resulting it reporting that the robot is in the sectors [i - 2, i + 2] with equal probability. This results in the following emission matrix.

```r
# Emission probabilities as described.
emission_probs <- matrix(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),
                         nrow = length(states),
                         ncol = length(symbols),
                         byrow = TRUE)
```

The start probabilities is set so that the robot starts in any of the 10 sectors with equal probability. The HMM is then initialized using the initHMM function from the HMM library.

```r
# Equal probability for each state to start.
start_probs <- rep(0.1, length(states))

hmm <- initHMM(states, symbols, start_probs, transition_probs, emission_probs)
```

Question 2

Question 2 asks to simulate the HMM for 100 time steps. This is simply done using the built in simHMM function from the HMM library.

```r
set.seed(1234) # Set seed to get repeatable results
sim <- simHMM(hmm, length = 100)
```

Question 3

Next, using the observations gained from the simulation, question 3 asks to compute the filtered and smoothed probability distributions for each of the 100 time points, as well as the most probable path.

First, recall the formula for filtered and smoothed probability distributions:

$$\textit{Filtering: } p(z^t|x^{0:t}) = \frac{\alpha(z^t)}{\sum_{z^t}\alpha(z^t)}$$

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

So, to compute the filtered and smoothed probability distribution, we first need alpha and beta. These can be computed with the forward-backward algorithm. Here, I am using the built in functions from the HMM library.

```r
# To compute smoothed and filtered probability distribution, we first need alpha
# and Beta (forward and backward probs) obtained by the forward-backward algorithm.
alpha <- prop.table(exp(forward(hmm, sim$observation)))
beta <- prop.table(exp(backward(hmm, sim$observation)))
```

We can then compute the filtered and smoothed probability distributions as in the formulas above.

```r
# Compute filtered and smoothed distributions.
filtered <- t(apply(alpha, MARGIN = 1, "/", apply(alpha, MARGIN = 2, sum)))
smoothed <- t(apply(alpha*beta, MARGIN = 1, "/", apply(alpha*beta, MARGIN = 2, sum)))
```

The most probable path can be computed using the Viterbi algorithm. Again I am using the built in function from the HMM library.

```r
# Compute most probable path with viterbi algorithm.
viterbi <- viterbi(hmm, sim$observation)
```

Question 4

Now we are asked to compute the accuracy of the filtered and smoothed probability distributions, and of the most probable path.

First, make the predictions for the filtered and smoothed probability distributions respectively. This does not have to be done to the most probable path, since this is already returned by the built in Viterbi function.

```r
# Predictions for filtered and smoothed distributions.
pred_filtered <- apply(filtered, MARGIN = 2, FUN = which.max)
pred_smoothed <- apply(smoothed, MARGIN = 2, FUN = which.max)
```

Then, the predictions can be used to compute the accuracy. We will find that the smoothed distribution gives us the best accuracy.

```
# Compute accuracy
sum(sim$states == pred_filtered) / 100 # Accuracy Filtered Distribution
sum(sim$states == pred_smoothed) / 100 # Accuracy Smoothed Distribution
sum(sim$states == viterbi) / 100       # Accuracy Most Probable Path
```

```
## [1] 0.63
## [1] 0.75
## [1] 0.49
```

Question 5

Question 5 explores how the results can differ depending on the simulated samples and asks to repeat the previous exercise with different simulated samples. In addition, the following two questions should also be answered:

1. In general, the smoothed distributions should be more accurate than the *filtered distributions*. Why?
2. In general, the smoothed distributions should be more accurate than the *most probable paths*. Why?

To answer this, I first created a function to generate simulations and compute accuracies for the filtered, smoothed and most probable path.

```
# Function to generate different simulated samples and compute accuracies.
GenerateSims <- function(sim, hmm) {

  # Alpha and Beta
  alpha <- prop.table(exp(forward(hmm, sim$observation)))
  beta <- prop.table(exp(backward(hmm, sim$observation)))

  # Filtered and smoothed Distributions
  filtered2 <- t(apply(alpha, MARGIN = 1, "/", apply(alpha, MARGIN = 2, sum)))
  smoothed2 <- t(apply(alpha*beta, MARGIN = 1, "/", apply(alpha*beta, MARGIN = 2, sum)))

  # Compute accuracies and return as array.
  return(c(sum(sim$states == apply(filtered2, MARGIN = 2, FUN = which.max)) / 100,
           sum(sim$states == apply(smoothed2, MARGIN = 2, FUN = which.max)) / 100,
           sum(sim$states == viterbi(hmm, sim$observation)) / 100))
}
```

I then created a loop which created 100 simulations and computed the accuracies for them using the function above.

```
# Generate simulations and compute accuracies
accuracies <- matrix(nrow = 100, ncol = 3)
for (i in 1:nrow(accuracies)) {
  accuracies[i,] <- GenerateSims(simHMM(hmm, length = 100), hmm)
}
```

The accuracies matrix are then used to visualize the results using a box plot.

```
# Visualize data with Boxplot
boxplot(accuracies,
        ylab = "Accuracy",
        col = c("skyblue", "lightgreen", "lightcoral"),
        names = c("Filtered", "Smoothed", "Viterbi"))
```

```
# Add Legend and horizontal gridlines
legend("topright", legend = c("Filtered", "Smoothed", "Viterbi"),
       fill = c("skyblue", "lightgreen", "lightcoral"))
abline(h = seq(0, 1, by = 0.1), col = "gray", lty = 2)
```

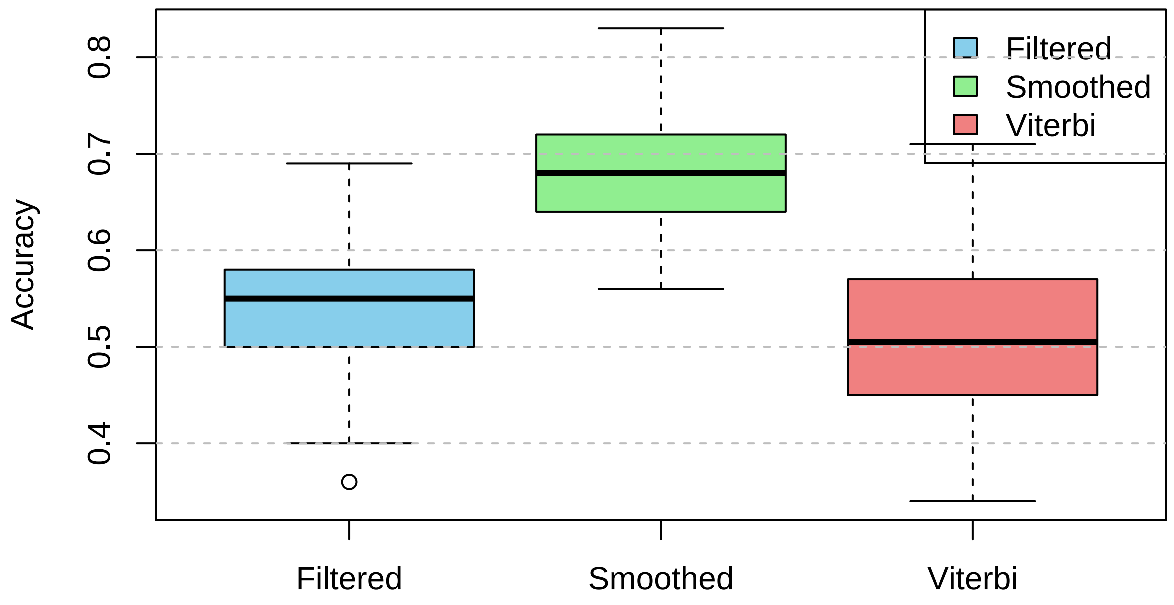

Figure 1: Accuracy Distribution for Different Methods

As seen in the graph, the smoothed distribution is generally more accurate than both the filtered distribution and the most probable path. The reason for this, is that the smoothed distribution uses more information than both the filtered distribution and the most probable path.

The filtered distribution is the probability distribution over hidden states at a given time step, given the observations up to that step, i.e., it does not consider any future observations.

The most probable path (Viterbi algorithm) seeks to find the sequence of states that maximizes the joint probability. This most likely sequence is also based solely on past observations.

The smoothed distribution instead considers both the previous and the future observations when calculating the probability distribution over hidden states at a given time step. In other words, the distribution is based on a lot more data compared to the two alternatives, resulting in a more accurate and less uncertain model.

Question 6.

Question 6 seeks answers to the question: 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?

To answer this, I use apply() to compute the entropy of the filtered distributions using the function entropy.empirical from the entropy library. The result is visualized in a plot which shows how the entropy depends on the number of observations.

```
# Compute entropy of the filtered distributions and visualize.
entropy <- apply(filtered, MARGIN = 2, entropy.empirical)
plot(entropy,
     type = "l",
     xlab = "Observations",
     ylab = "Entropy value")
```

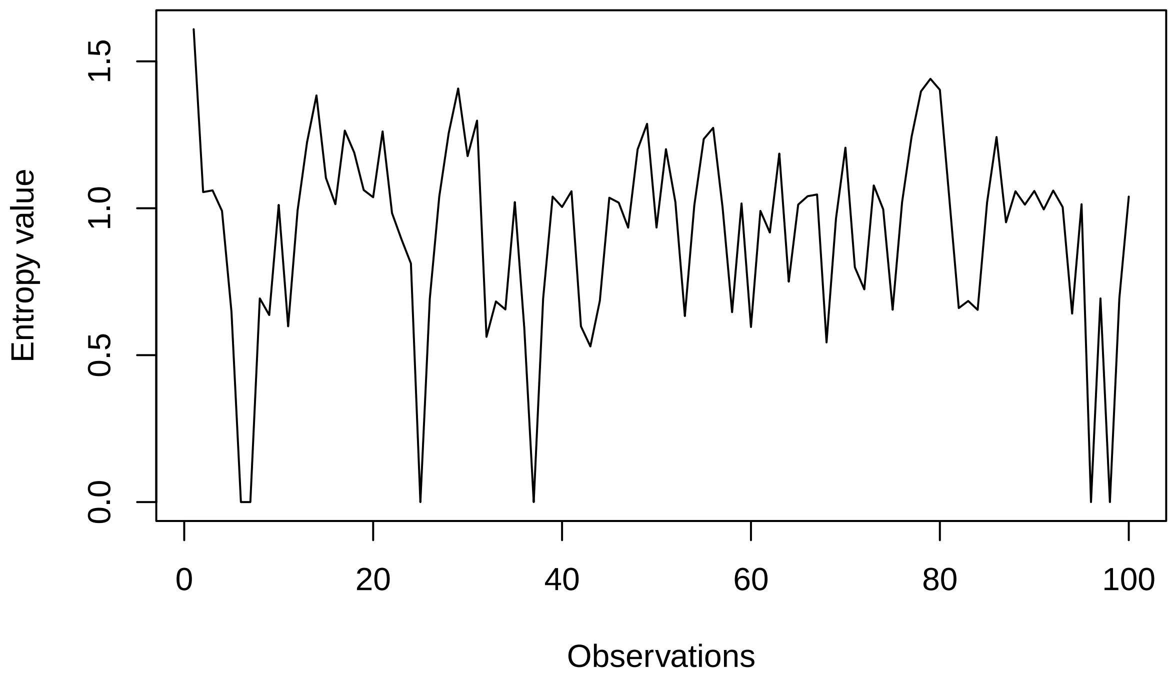

Figure 2: Entropy vs. Number of Observations

The entropy value can be seen as a measure of uncertainty in the model, where higher entropy means more uncertainty. As seen, no real relationship between the entropy and and number of observations can be found, and thus you can conclude that more observations does not give the model a better understanding of the robots current location.

Question 7.

Question 7 asks to consider any of the samples above of length 100 and then compute the probabilities of the hidden states for the time step 101.

For this, I simply multiplied the transition probabilities matrix with the last time step of the filtered probabilities. This results in a probability matrix for the hidden states at time step 101.

```r
# Multiply transition probabilities with the last filtered step.
transition_probs %*% filtered[ ,100]
```

```
##        [,1]
##  [1,] 0.000
##  [2,] 0.000
##  [3,] 0.125
##  [4,] 0.375
##  [5,] 0.375
##  [6,] 0.125
##  [7,] 0.000
##  [8,] 0.000
##  [9,] 0.000
## [10,] 0.000
```

This states that the robot is in either state 4 or 5 with a 37.5% probability and in either state 3 or 6 with a 12.5% probability in time step 101. The probability of the robot being in any other state is zero.