

Advanced Machine Learning

Hugo Morvan (hugmo418)

2024-09-20

Lab 2: Hidden Markov Models

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.

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.

```
library(HMM)
set.seed(123)

n_states = 10
n_symbols = 10
# Transition probability : stay at current state 0.5 or move to the next
transProbs = matrix(0, n_states, n_states)

for(i in 1:n_states){
  transProbs[i,i] = 0.5 #Stay
  transProbs[i,(i+n_symbols)%%(n_states)+1] = 0.5 # Move to the next state
  #All the weird indexing is to go around the issues of the 1-index in R
}
print("Transition Probs:")
```

```
## [1] "Transition Probs:"
```

```
print(transProbs)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## [2,] 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## [3,] 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
```

```
## [4,] 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
## [5,] 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
## [6,] 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
## [7,] 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
## [8,] 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
## [9,] 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
## [10,] 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5
```

```
# Emission probability : the sensor inaccuracy ?
emissionProbs = matrix(0, n_states, n_symbols)

for(i in 1:n_states){
  for(j in -2:2){
    emissionProbs[i, (i+j+n_symbols-1)%(n_symbols)+1] = 0.2
    #All the weird indexing is to go around the issues of the 1-index in R
  }
}
print("Emission Probs:")
```

```
## [1] "Emission Probs:"
```

```
print(emissionProbs)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
## [2,] 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
## [3,] 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
## [4,] 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
## [5,] 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
## [6,] 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
## [7,] 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
## [8,] 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
## [9,] 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
## [10,] 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

```
myHMM = initHMM(States = 1:10, Symbols = 1:10,
               startProbs = NULL ,
               transProbs = transProbs,
               emissionProbs = emissionProbs)
```

Question 2:

Simulate the HMM for 100 time steps.

```
sim = simHMM(myHMM, 100)
```

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.

```

obs = sim$observation

get_probs = function(myHMM, obs){
  log_forwards = forward(hmm = myHMM, observation = obs)
  forwards = exp(log_forwards) #  $\alpha(z_t)$ 

  filtered = prop.table(forwards, margin = 2)

  log_backwards = backward(myHMM, obs)
  backwards = exp(log_backwards) #  $\beta(z_t)$ 

  log_alpha_beta = log_forwards+log_backwards
  alpha_beta = exp(log_alpha_beta)

  smoothed = prop.table(alpha_beta)
  #sum(smoothed) # =1

  most_probable_path = viterbi(myHMM, obs)

  return(list("filtered" = filtered, "smoothed" = smoothed, "most_probable_path" = most_probable_path))
}
res = get_probs(myHMM, obs)

```

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 elementwise 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.

```

get_accuracies = function(sim, get_probs_result){

  filtered_count = table(sim$states == apply(get_probs_result$filtered, 2, which.max))
  filtering_accuracy = filtered_count[2]/sum(filtered_count)
  names(filtering_accuracy) = "Filtering accuracy"

  smoothed_count = table(sim$states == apply(get_probs_result$smoothed, 2, which.max))
  smoothing_accuracy = smoothed_count[2]/sum(smoothed_count)
  names(smoothing_accuracy) = "Smoothing accuracy"

  viterbi_count = table(sim$states == get_probs_result$most_probable_path)
  viterbi_accuracy = viterbi_count[2]/sum(viterbi_count)
  names(viterbi_accuracy) = "Viterbi accuracy"

  return(list("filtering_accuracy" = filtering_accuracy,
             "smoothing_accuracy" = smoothing_accuracy,
             "viterbi_accuracy" = viterbi_accuracy))
}

```

```
acc = get_accuracies(sim, res)
print(acc)
```

```
## $filtering_accuracy
## Filtering accuracy
##           0.53
##
## $smoothing_accuracy
## Filtering accuracy
##           0.53
##
## $viterbi_accuracy
## Viterbi accuracy
##           0.36
```

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 ?

```
sim2 = simHMM(myHMM, 100)

res2 = get_probs(myHMM, sim2$observation)

acc2 = get_accuracies(sim2, res2)
print(acc2)
```

```
## $filtering_accuracy
## Filtering accuracy
##           0.59
##
## $smoothing_accuracy
## Filtering accuracy
##           0.59
##
## $viterbi_accuracy
## Viterbi accuracy
##           0.48
```

The smoothed distribution should be more accurate than the filtering because it combines information from both the forward and backward probabilities, while the filtering only has the forward probabilities.

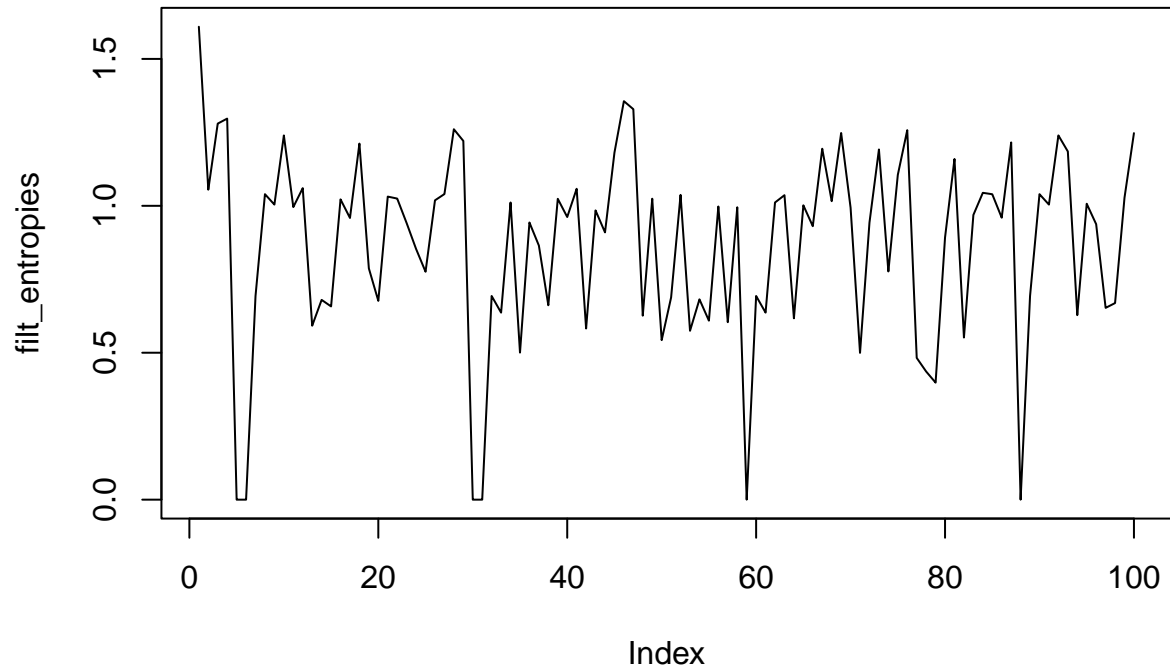
The smoothed distribution should also be more accurate than the most probable path because

Question 6:

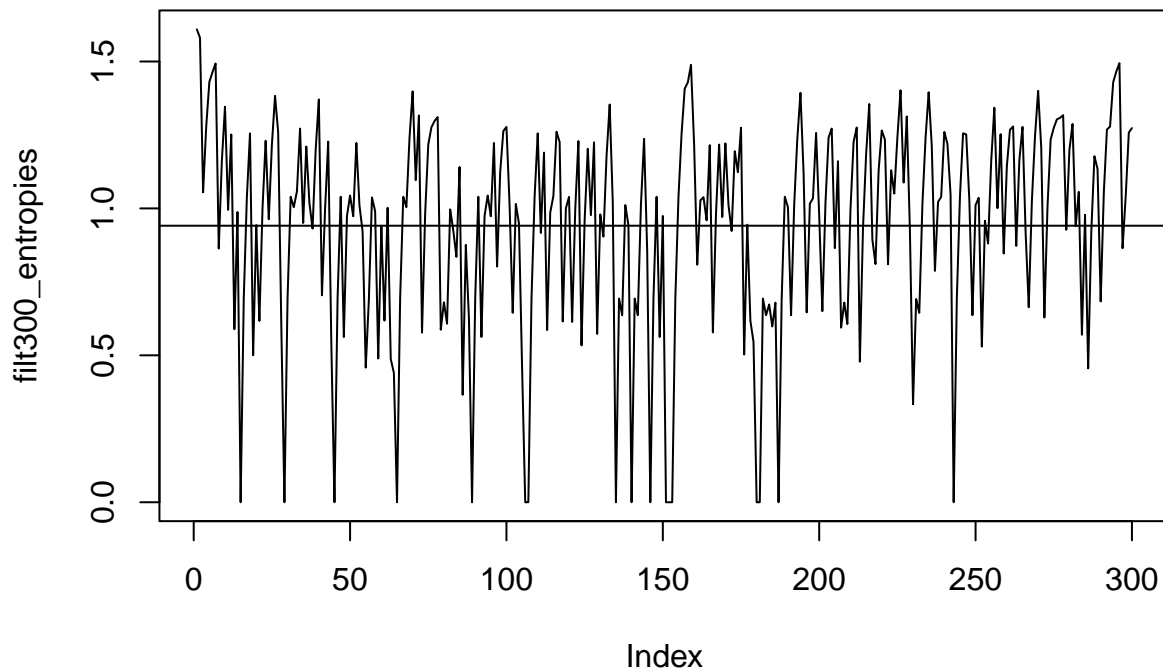
Is it always true that the later in time (i.e., the more observations you have received) the better you know where the robot is ? Hint: You may want to compute the entropy of the filtered distributions with the function `entropy.empirical` of the package `entropy`.

```
library(entropy)

filt_entropies = apply(res$filtered, 2, entropy.empirical)
plot(filt_entropies, type='l')
```



```
# Simulating for 300 time steps:
sim300 = simHMM(myHMM, 300)
res300 = get_probs(myHMM, sim300$observation)
filt300_entropies = apply(res300$filtered, 2, entropy.empirical)
mean300 = mean(filt300_entropies)
plot(filt300_entropies, type="l")
abline(a=mean300, b = 0)
```



It does not seem like the entropy reduces later in time, meaning that we do not know better where the robot is over time.

Question 7:

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

```
pred101 = prop.table(res$smoothed[,100]%*%transProbs)
print("state prediction probabilities")
```

```
## [1] "state prediction probabilities"
```

```
print(pred101)
```

```
##      [,1] [,2]      [,3]      [,4] [,5]      [,6]      [,7] [,8] [,9] [,10]
## [1,]  0    0 0.04883721 0.2226744 0.375 0.2773256 0.07616279  0    0    0
```

```
plot(x = c(1:10),prop.table(res$smoothed[,100]), type = "h", col = "red")
lines(x = c(1:10)+0.05,pred101, type = "h", col = "blue")
abline(v = sim$states[100]-0.05, col="darkred")
legend("topright", legend=c("true state @ t=100", "smoothed state prob. @ t=100", "state prob @ t = 101",
col=c("darkred", "red", "blue"), lty=c("solid","solid","solid"), cex=0.8)
```

