AML Lab 2 (Hidden Markov Models)

Roshni Sundaramurthy

24/09/2019

Contents

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.

A hidden Markov model (HMM) over $Z^{0:T}; X^{0:T}$ where $X^{0:T}$ are observed and $Z^{0:T}$ are unobserved.

There are three probability distributions in HMMs: a prior probability $p(Z^0)$, a transition probability $p(Z^{t+1}|Z^t)$ and an emission (observation) probability $p(X^t|Z^t)$.

Hence, a HMM is a DBN that defines,

$$p(Z^{0:T}, X^{0:T}) = p(Z^0) \prod_{t=1}^{T-1} p(Z^{t+1}|Z^t) \prod_{t=0}^{T} p(X^t|Z^t)$$

The transition and emission matrix are the main parameters to build HMM.

- The transition probability matrix is a probability of switching from one sector to another.
- Emission probability matrix is a selection probability of the element in a list.

So here, we consider that the robot is in the sectors [i-2, i+2] with equal probability. For say, if i=1, then the emission probability will be in 9,10,1,2,3 with 0.2 probability.

```
library(HMM)

# Functions available in HMM
ls("package:HMM")
```

```
## [1] "backward"        "baumWelch"       "dishonestCasino" "forward"
## [5] "initHMM"         "posterior"       "simHMM"          "viterbi"
## [9] "viterbiTraining"
```

```
# ring is divided into 10 sectors
# States
states = c(1:10)

# Symbols
symbols = c(1:10)

# sectors [i-2, i+2] with equal probability
# transition probability matrix
transProb = 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),
                     byrow = TRUE, nrow = 10,
                     ncol = 10)

# emission probability matrix # row&col sum=1
emissProb = 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),
                    byrow = TRUE, nrow = 10,
                    ncol = 10)

# Building the model
HMM.Model <- initHMM(States = states, Symbols = symbols,
                    transProbs = transProb, emissionProbs = emissProb)
print(HMM.Model)
```

```
## $States
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $Symbols
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $startProbs
##   1   2   3   4   5   6   7   8   9  10
## 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##
## $transProbs
##      to
## from   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
##
## $emissionProbs
##       symbols
## states   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
```

Question 2: Simulate the HMM for 100 time steps.

Question 3: Discard the hidden states from the sample obtained above. Use the remaining obser- vations to compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path.

Filtered probability distribution, t(online):

This is computed using forward function. It computes the posterior distribution over the *current state* given all the previous evidence. It achieves better noise reduction than simply estimating the hidden state based on the current estimate $p(Z^t, x^t)$

$$p(Z^t, x^{0:t}) = \frac{\alpha(Z^t)}{\sum_{z^t} \alpha(z^t)}$$

where capital variables are unknown variables, and the lowercase ones are known variables. We sum up over all possible state realizations in the previous time step by their transition probability to the current state, and for each of those, we weigh the probability by the likelihood of emission from the state (which doesn't depend on z_t so we can move it out of the sum). Then the normalizing constant here is computed by summing up $p(Z^t, x^{0:t})$ for all possible realizations of Z^t.

Smoothed probability distribution, T(offline):

Smoothed probability distribution (Forward-Backward) computes the posterior distribution over the past state given all the evidence and gives most likely state at each position. This is computed using posterior function. Although noise and uncertainty are significantly reduced as a result of conditioning on past and future data, the smoothing process can only be run offline.

$$p(Z^t, x^{0:T}) = \frac{\alpha(Z^t)\beta(Z^t)}{\sum_{z^t} \alpha(z^t)\beta(z^t)}$$

Most probable path:

The Viterbi algorithm (dynamic programming approach) calculates the possible state for a sequence of observations for a given HMM. It helps us to determine the most likely sequence of states the system went to produce those observations.

$$z^T_{max} = argmax_{z^T} P(z^t|x^T)$$

We will now take max-x's instead of summations, and we also have to keep a series of "backtrace pointers" so that we can reproduce our path. The time and space complexity is linear in terms of the length of the sequence.

Function for simulating HMM, finding probability distributions, most probable states and path, and accuracy

```
# Function

FindRobot = function(hmm_model,a){
  # we simulate 100 observation elements with a simHMM function using our hmm model.

  simhmm <- simHMM(hmm_model, 100)
  simulated <- data.frame(hidden_state=simhmm$states, observation=simhmm$observation)
```

```
  # printing simulated data
  #print(head(simulated))


  ###### filtered probability distributions for each of the 100 time points ##########
  ###### FORWARD ##########

  # discard hidden states & use only observations
  prob_log_scaled = forward(hmm_model,simulated$observation)

  # obtaining a normalized probability distribution
  expon = exp(prob_log_scaled)
  prob_dist = prop.table(expon,2)

  # find out the most probable states
  maxpt_filtered = apply(X=prob_dist, MARGIN=2, FUN=which.max) # Margin=2 indicates columns

  # find accuracy
  acc_filter = (table(maxpt_filtered==simulated$hidden_state))[2]

    #(sum(maxpt_filtered == simulated$hidden_state)/length(simulated$hidden_state))*100


  ########## smoothed probability distributions for each of the 100 time points ##########
  ###### FORWARD & BACKWARD ##########

  # Compute the posterior probabilities for the states
  posterior = posterior(hmm_model,simulated$observation)
  # print(posterior)

  # find out the most probable states
  maxpt_smoothed = apply(X=posterior, MARGIN=2, FUN=which.max) # Margin=2 indicates columns

  # find accuracy
  acc_smooth = (table(maxpt_smoothed==simulated$hidden_state))[2]


  ############### Compute the most probable path of states ################
  maxpt_path = viterbi(hmm_model,simulated$observation)

  # find accuracy
  acc_path = (table(maxpt_path==simulated$hidden_state))[2]

  if(a==1)
    return(prob_dist)
  else if(a==2)
    return(posterior)
  else if(a==3)
    return(c(simulated,data.frame(maxpt_filtered,maxpt_smoothed,maxpt_path)))
  else if(a==4)
    return(c(acc_filter,acc_smooth,acc_path))

}
```

```
Robot1<-FindRobot(HMM.Model,1) # filter dist
Robot2<-FindRobot(HMM.Model,2) # smooth_dist
Robot3<-FindRobot(HMM.Model,3) # probable states
Robot4<-FindRobot(HMM.Model,4) # accuracy
```

States:

```
state = Robot3$hidden_state
state
```

```
##   [1]  8  8  8  8  8  9 10 10  1  1  1  1  1  1  2  2  3  4  5  5  6  6  7
##  [24]  8  8  9  9  9 10 10 10 10  1  2  2  3  4  5  6  7  8  8  8  8  9 10
##  [47]  1  2  2  3  4  4  5  5  6  7  7  8  8  8  9  9 10 10  1  2  3  3  3
##  [70]  3  3  3  4  5  5  6  6  6  7  7  8  9 10 10  1  1  1  1  2  3  3  3
##  [93]  4  5  5  6  6  7  7  7
```

Observations:

```
element = Robot3$observation
element
```

```
##   [1] 10  7  6  7  7  8  8  2  2  1  9  9  2  2 10  1  5  5  6  5  5  4  5
##  [24]  9 10  9 10  7  2 10  9  8  3  3 10  3  6  7  7  8 10  9  7  7 10  2
##  [47]  9  4  2  4  6  3  4  5  6  7  9  6  7  9  8  9  2  8  1  2  3  3  4
##  [70]  2  5  4  2  5  7  7  4  7  7  7  8 10  2 10  3  3  3  3  2  1  3  5
##  [93]  6  4  6  6  7  5  9  6
```

Filtered probability distribution:

```
filter = Robot1
filter[,1:10]
```

```
##       index
## states   1         2          3          4         5 6   7 8   9   10
##      1  0.2 0.0000000 0.00000000 0.00000000 0.0000000 0 0.0 0 0.0 0.25
##      2  0.2 0.2857143 0.00000000 0.00000000 0.0000000 0 0.0 0 0.0 0.00
##      3  0.2 0.2857143 0.33333333 0.00000000 0.0000000 0 0.0 0 0.0 0.00
##      4  0.2 0.2857143 0.33333333 0.00000000 0.0000000 0 0.0 0 0.0 0.00
##      5  0.0 0.1428571 0.25000000 0.58333333 0.3888889 0 0.0 0 0.0 0.00
##      6  0.0 0.0000000 0.08333333 0.33333333 0.6111111 0 0.0 0 0.0 0.00
##      7  0.0 0.0000000 0.00000000 0.08333333 0.0000000 1 0.5 0 0.0 0.00
##      8  0.0 0.0000000 0.00000000 0.00000000 0.0000000 0 0.5 0 0.0 0.00
##      9  0.0 0.0000000 0.00000000 0.00000000 0.0000000 0 0.0 1 0.5 0.25
##      10 0.2 0.0000000 0.00000000 0.00000000 0.0000000 0 0.0 0 0.5 0.50
```

Smoothed probability distribution:

```
smooth = Robot2
smooth[,1:10]
```

```
##       index
## states          1          2         3         4       5      6        7
##      1  0.23958333 0.10416667 0.0000000 0.0000000 0.00000 0.0000 0.000000
##      2  0.40625000 0.37500000 0.2083333 0.0000000 0.00000 0.0000 0.000000
##      3  0.30208333 0.43750000 0.5416667 0.4166667 0.00000 0.0000 0.000000
##      4  0.00000000 0.08333333 0.2500000 0.5833333 0.78125 0.3125 0.078125
##      5  0.00000000 0.00000000 0.0000000 0.0000000 0.21875 0.6875 0.578125
##      6  0.00000000 0.00000000 0.0000000 0.0000000 0.00000 0.0000 0.343750
```

```
##     7  0.00000000 0.00000000 0.0000000 0.0000000 0.00000 0.0000 0.000000
##     8  0.00000000 0.00000000 0.0000000 0.0000000 0.00000 0.0000 0.000000
##     9  0.00000000 0.00000000 0.0000000 0.0000000 0.00000 0.0000 0.000000
##     10 0.05208333 0.00000000 0.0000000 0.0000000 0.00000 0.0000 0.000000
##       index
## states         8        9 10
##     1  0.0000000 0.000000  0
##     2  0.0000000 0.000000  0
##     3  0.0000000 0.000000  0
##     4  0.0000000 0.000000  0
##     5  0.2708333 0.000000  0
##     6  0.6145833 0.578125  0
##     7  0.1145833 0.421875  1
##     8  0.0000000 0.000000  0
##     9  0.0000000 0.000000  0
##     10 0.0000000 0.000000  0
```

Most probable path:

```
path = Robot3$maxpt_path
path
```

```
##   [1]  8  8  8  8  8  9 10  1  1  1  1  1  1  1  1  2  3  3  4  4  4  5  6
##  [24]  7  8  8  8  9 10 10 10 10  1  1  2  3  4  5  6  7  8  8  8  9 10  1
##  [47]  1  2  2  3  4  4  4  4  5  6  7  7  7  7  8  9 10 10  1  1  1  1  2
##  [70]  2  3  3  3  4  5  5  6  7  8  9 10  1  1  1  1  1  1  1  1  1  2  3
##  [93]  4  4  4  4  5  6  7  7
```

Most probable states for filtered probability distribution:

```
filtered_states = Robot3$maxpt_filtered
filtered_states
```

```
##   [1]  1  9  8  8  9  9 10 10  1  1  1  1  1  2  2  2  3  4  4  5  5  6  6
##  [24]  7  8  9  9  9 10  1  1 10  1  1  2  2  4  5  5  6  8  8  9  9  9 10
##  [47]  1  2  2  3  4  5  5  6  6  7  7  8  8  9  9 10 10 10  1  1  1  2  3
##  [70]  3  4  4  4  4  5  6  6  6  7  7  8  8 10  1  1  2  2  3  4  3  3  4
##  [93]  4  5  6  6  7  7  7  8
```

Most probable states for smoothed probability distribution:

```
smoothed_states = Robot3$maxpt_smoothed
smoothed_states
```

```
##   [1]  8  8  8  8  9  9 10 10 10 10  1  1  1  2  2  3  3  4  4  5  5  6  7
##  [24]  7  8  8  9  9 10 10 10 10  1  1  2  3  4  5  6  7  8  8  9  9 10 10
##  [47]  1  2  3  3  4  5  5  6  6  7  7  8  8  9  9 10 10 10  1  1  1  2  2
##  [70]  3  3  4  4  5  5  6  6  7  7  8  9  9 10  1  1  1  2  2  3  3  3  4
##  [93]  4  5  5  6  6  7  7  8
```

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.

```
# Accuracy

paste("Accuracy for filtered dist =",Robot4[1],"%")
```

```
## [1] "Accuracy for filtered dist = 47 %"
```

```
paste("Accuracy for smoothed dist =",Robot4[2],"%")
```

```
## [1] "Accuracy for smoothed dist = 71 %"
```

```
paste("Accuracy for most probable path =",Robot4[3],"%")
```

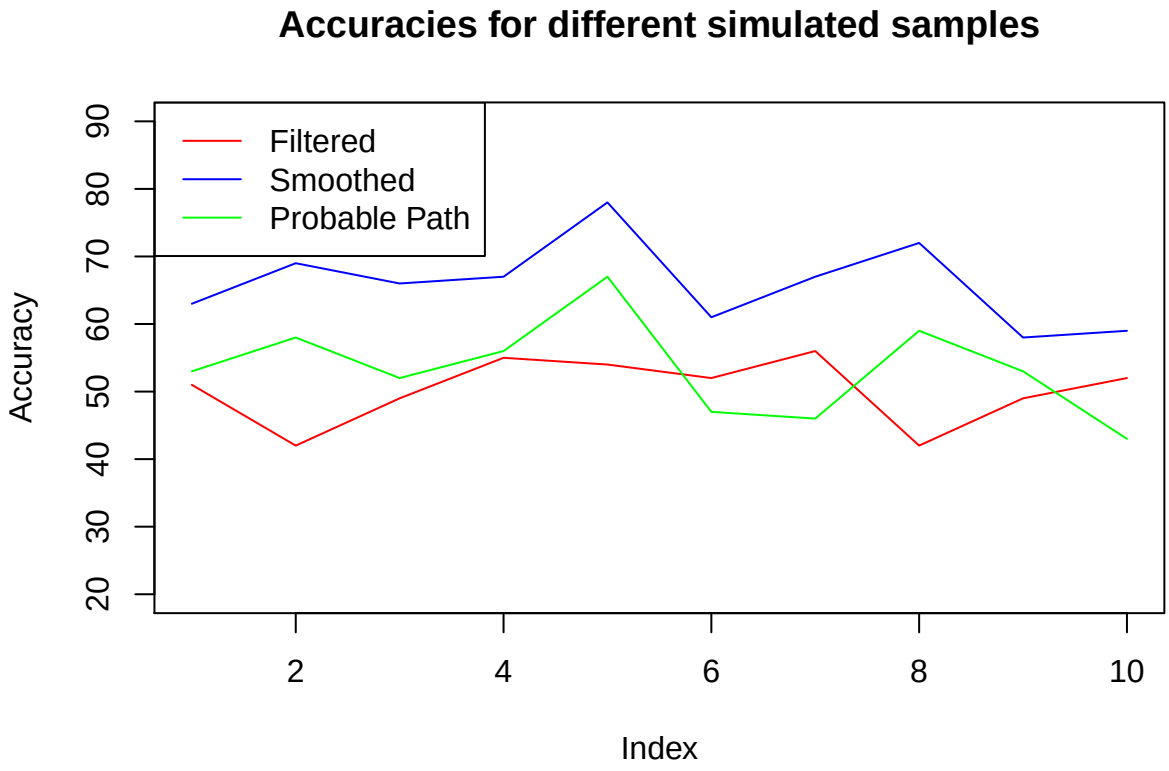
```
## [1] "Accuracy for most probable path = 40 %"
```

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 ?

```
sim = sapply(1:10,FUN = function(x){FindRobot(HMM.Model,4)})
```

```
## [1] "Row1=Filtered distribution ; Row2=Smoothed distribution ; Row3=Most probable path"
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## TRUE   51   42   49   55   54   52   56   42   49    52
## TRUE   63   69   66   67   78   61   67   72   58    59
## TRUE   53   58   52   56   67   47   46   59   53    43
```



From the plot, it is obvious that smoothed distributions is more accurate than the filtered distributions and most probable paths. The smoothed distribution estimates the state better than filtering because more evidence is available. Also, filtering estimates $P(Z^t)$ by using evidence up to time t. i.e, $P(Z^1)$ is estimated by $P(Z^1|x^1)$, i.e. it ignores future observation x^2.

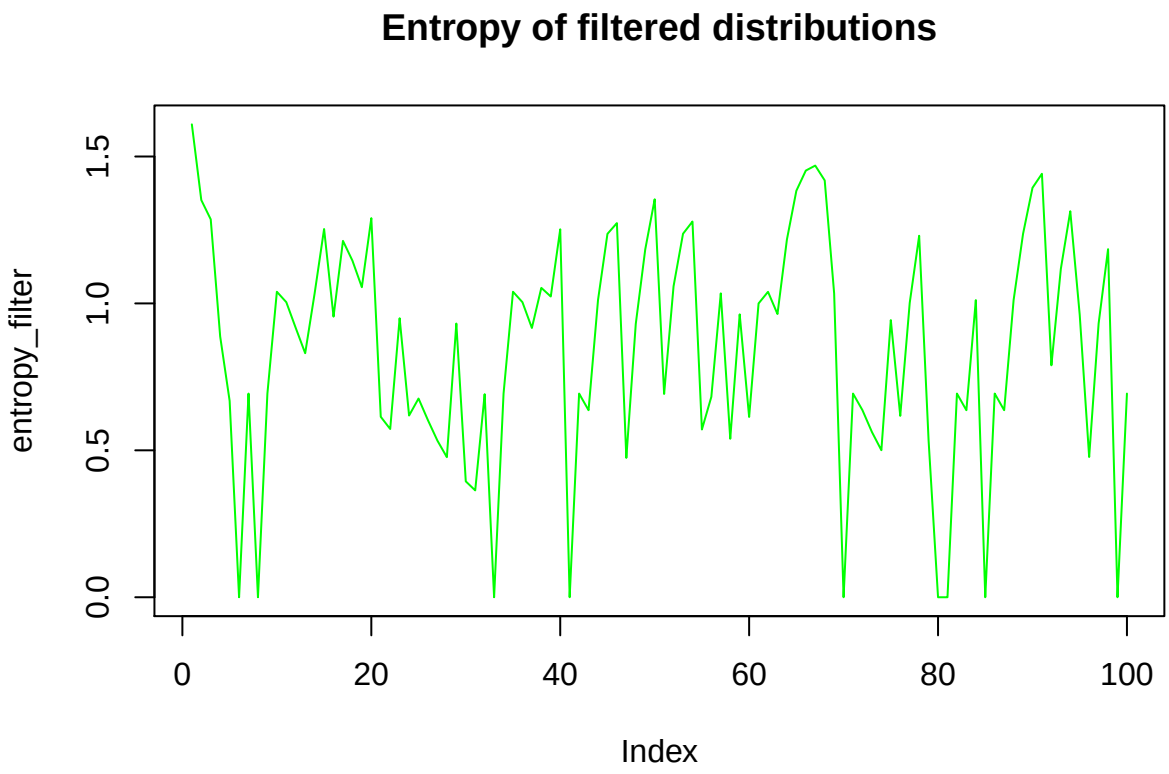
At t=2, the new observation x^2 also has some Z^1 information. Hence, we can update the distribution about past state by future evidence by computing $P(Z^1|x^1, x^2)$ by smoothing distribution.

Question 6: Is it true that the more observations you have 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 is a measure of uncertainty and it estimates the Shannon entropy of random variable.

```
library(entropy)
# Entropy of filtered distributions
# entropy.empirical returns an estimate of the Shannon entropy

entropy_filter <- apply(Robot1, 2, entropy.empirical)
plot(entropy_filter, col="green", type="l", main="Entropy of filtered distributions")
```



From the plot, we can see that entropy is zero for few states and it indicates that the robot's state is certain. Other entropy values seems to be so random and uncertain. Also, even if we add more observations, we don't know exactly where the robot will be due to uncertainty and the next state of robot depends on the previous state.

Question 7: Consider any of the samples above of length 100. Compute the probabilities of the hidden states for the time step 101.

Computed probabilities of the hidden states for the time step 101:

```
# samples at 100th state * transition prob matrix
Robot1[,100] %*% HMM.Model$transProbs
```

```
##       to
##        1 2 3    4   5    6 7 8 9 10
##   [1,] 0 0 0 0.25 0.5 0.25 0 0 0  0
```

```
#Robot1[,100] %*% (HMM.Model$transProbs)^1
# Robot1[,100] %*% (HMM.Model$transProbs)^2 # time step 102
```

We compute the posterior distribution over the future state given all the previous evidence. For time step 101, considering transition probability matrix and applying it to the 100th time step of filtered probability distribution.

Reference

1. https://en.wikipedia.org/wiki/Hidden_Markov_model
2. https://www.datatechnotes.com/2017/12/hidden-markov-model-example-in-r.html
3. https://luisdamiano.github.io/BayesHMM/articles/introduction.html

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(HMM)

# Functions available in HMM
ls("package:HMM")

# ring is divided into 10 sectors
# States
states = c(1:10)

# Symbols
symbols = c(1:10)

# sectors [i-2, i+2] with equal probability
# transition probability matrix
transProb = 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),
                     byrow = TRUE, nrow = 10,
                     ncol = 10)

# emission probability matrix # row&col sum=1
emissProb = 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),
                     byrow = TRUE, nrow = 10,
                     ncol = 10)
```

```r
# Building the model
HMM.Model <- initHMM(States = states, Symbols = symbols,
                     transProbs = transProb, emissionProbs = emissProb)
print(HMM.Model)

# Function

FindRobot = function(hmm_model,a){
  # we simulate 100 observation elements with a simHMM function using our hmm model.

  simhmm <- simHMM(hmm_model, 100)
  simulated <- data.frame(hidden_state=simhmm$states, observation=simhmm$observation)

  # printing simulated data
  #print(head(simulated))


  ###### filtered probability distributions for each of the 100 time points ##########
  ###### FORWARD ##########

  # discard hidden states & use only observations
  prob_log_scaled = forward(hmm_model,simulated$observation)

  # obtaining a normalized probability distribution
  expon = exp(prob_log_scaled)
  prob_dist = prop.table(expon,2)

  # find out the most probable states
  maxpt_filtered = apply(X=prob_dist, MARGIN=2, FUN=which.max) # Margin=2 indicates columns

  # find accuracy
  acc_filter = (table(maxpt_filtered==simulated$hidden_state))[2]

    #(sum(maxpt_filtered == simulated$hidden_state)/length(simulated$hidden_state))*100


  ########## smoothed probability distributions for each of the 100 time points ##########
  ###### FORWARD & BACKWARD ##########

  # Compute the posterior probabilities for the states
  posterior = posterior(hmm_model,simulated$observation)
  # print(posterior)

  # find out the most probable states
  maxpt_smoothed = apply(X=posterior, MARGIN=2, FUN=which.max) # Margin=2 indicates columns

  # find accuracy
  acc_smooth = (table(maxpt_smoothed==simulated$hidden_state))[2]


  ############### Compute the most probable path of states ################
  maxpt_path = viterbi(hmm_model,simulated$observation)
```

```
  # find accuracy
  acc_path = (table(maxpt_path==simulated$hidden_state))[2]

  if(a==1)
    return(prob_dist)
  else if(a==2)
    return(posterior)
  else if(a==3)
    return(c(simulated,data.frame(maxpt_filtered,maxpt_smoothed,maxpt_path)))
  else if(a==4)
    return(c(acc_filter,acc_smooth,acc_path))

}

Robot1<-FindRobot(HMM.Model,1) # filter dist
Robot2<-FindRobot(HMM.Model,2) # smooth_dist
Robot3<-FindRobot(HMM.Model,3) # probable states
Robot4<-FindRobot(HMM.Model,4) # accuracy
state = Robot3$hidden_state
state
element = Robot3$observation
element
filter = Robot1
filter[,1:10]
smooth = Robot2
smooth[,1:10]
path = Robot3$maxpt_path
path
filtered_states = Robot3$maxpt_filtered
filtered_states
smoothed_states = Robot3$maxpt_smoothed
smoothed_states

# Accuracy

paste("Accuracy for filtered dist =",Robot4[1],"%")

paste("Accuracy for smoothed dist =",Robot4[2],"%")

paste("Accuracy for most probable path =",Robot4[3],"%")

sim = sapply(1:10,FUN = function(x){FindRobot(HMM.Model,4)})
paste("Row1=Filtered distribution ; Row2=Smoothed distribution ; Row3=Most probable path")
sim

plot(sim[1,], col="red", ylim = c(20,90), type="l", ylab = "Accuracy",
     main="Accuracies for different simulated samples")
lines(sim[2,], col="blue")
lines(sim[3,], col="green")
legend("topleft", legend = c("Filtered", "Smoothed", "Probable Path"),
       col = c("red","blue","green"), lty=c(1,1))
library(entropy)
# Entropy of filtered distributions
```

```r
# entropy.empirical returns an estimate of the Shannon entropy

entropy_filter <- apply(Robot1, 2, entropy.empirical)
plot(entropy_filter, col="green", type="l", main="Entropy of filtered distributions")

# samples at 100th state * transition prob matrix
Robot1[,100] %*% HMM.Model$transProbs
#Robot1[,100] %*% (HMM.Model$transProbs)^1
# Robot1[,100] %*% (HMM.Model$transProbs)^2 # time step 102
```