PGM_Lab2

Maria(marse306),Jasleen(jasma846),vinay(vinbe289),Tejashree(tejma768)

September 30, 2019

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.

```
states = 1:10
symbols = 1:10
startProbs= rep(0.1,10)

# The probability of transition from current state to next state given current state
len = length(states)

# The probability to stay in that state or to move = 0.5

transProbs=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),
                    len,len,byrow = TRUE)
#emissionProbs is a (number of states)x(number of states)-sized matrix
#Emission probabilities: how likely the states are at any particular timestep
#emission probability hence probability of each state  = 1/5

emissionProbs = 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
                         ),len,len , byrow = TRUE)
```

```
HMM_model = initHMM(States = states, Symbols = symbols,
        startProbs=startProbs, transProbs=transProbs,
        emissionProbs=emissionProbs)
```

Question 2

Simulate the HMM for 100 time steps.

```
nSim = 100
HMM_simulation = simHMM(HMM_model, nSim)
```

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.

```
path = findpath(HMM_model, HMM_simulation$observation)
filtered_path = path[[1]]
smoothed_path = path[[2]]
viterbi_path = path[[3]]
```

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 of filtered probability distribution:  56 %
```

```
##
##
## Accuracy of smoothed probability distribution:  66 %
```

```
##
##
## Accuracy of viterbi probability distribution:  50 %
```

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 ?

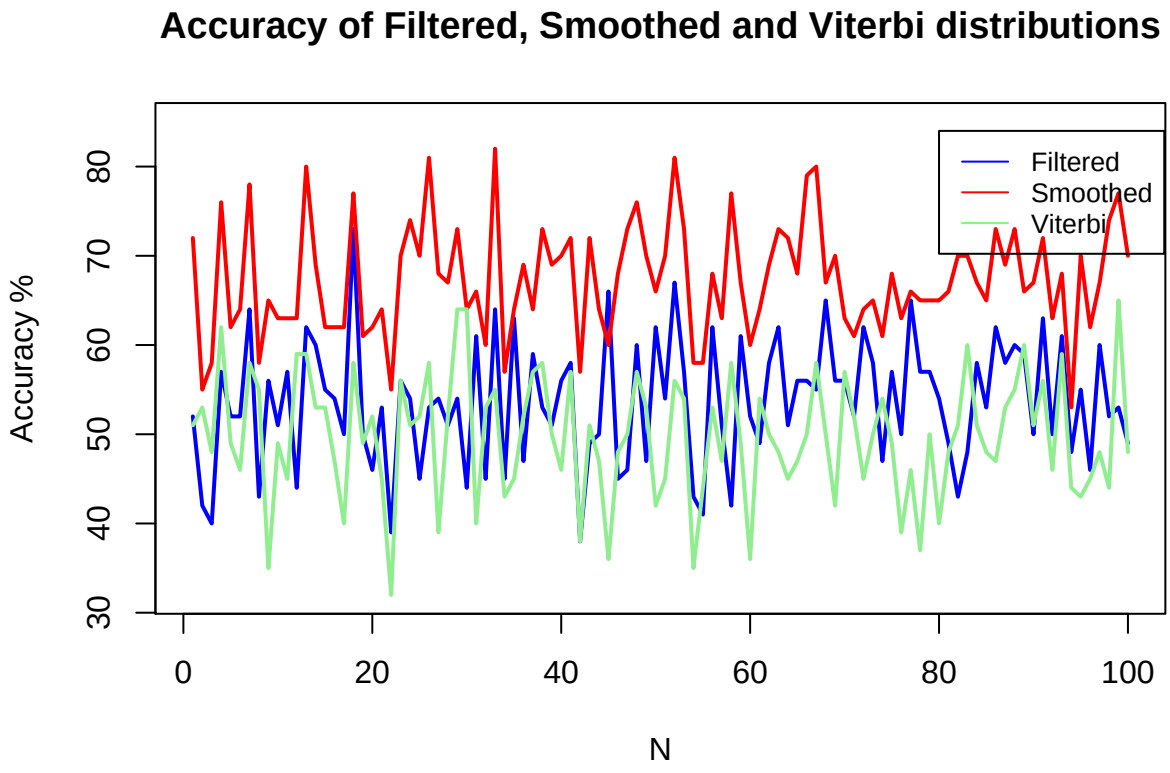
```r
# Repeating  the above excercise with 100 simulations

accuracy_df = data.frame(matrix(NA , nrow = 20 , ncol = 3))
colnames(accuracy_df) = c("Filtering","Smoothing","Viterbi")

#Running for different simulations

for(i in 1:100){
  simulation = simHMM(HMM_model, 100)
  paths = findpath(HMM_model, simulation$observation)
  accuracy_df[i,1] = accuracy(simulation$states,paths[[1]] )
  accuracy_df[i,2] = accuracy(simulation$states,paths[[2]])
  accuracy_df[i,3] = accuracy(simulation$states, paths[[3]])

}
```



From the above plot we see that the smoothed distributions are mostly more accurate than filtered and the most probable paths.

In case of filtered distribution, forward algorithm is used which calculates the most likely state given the past observations. Smoothing algorithm takes in account the past output and also all the observations and hence we get better smoothing results. Smoothing try to maximize each individual prediction.

Viterbi is constrained to produce a valid path so the accuracy of results is less than smoothing.

Question 6

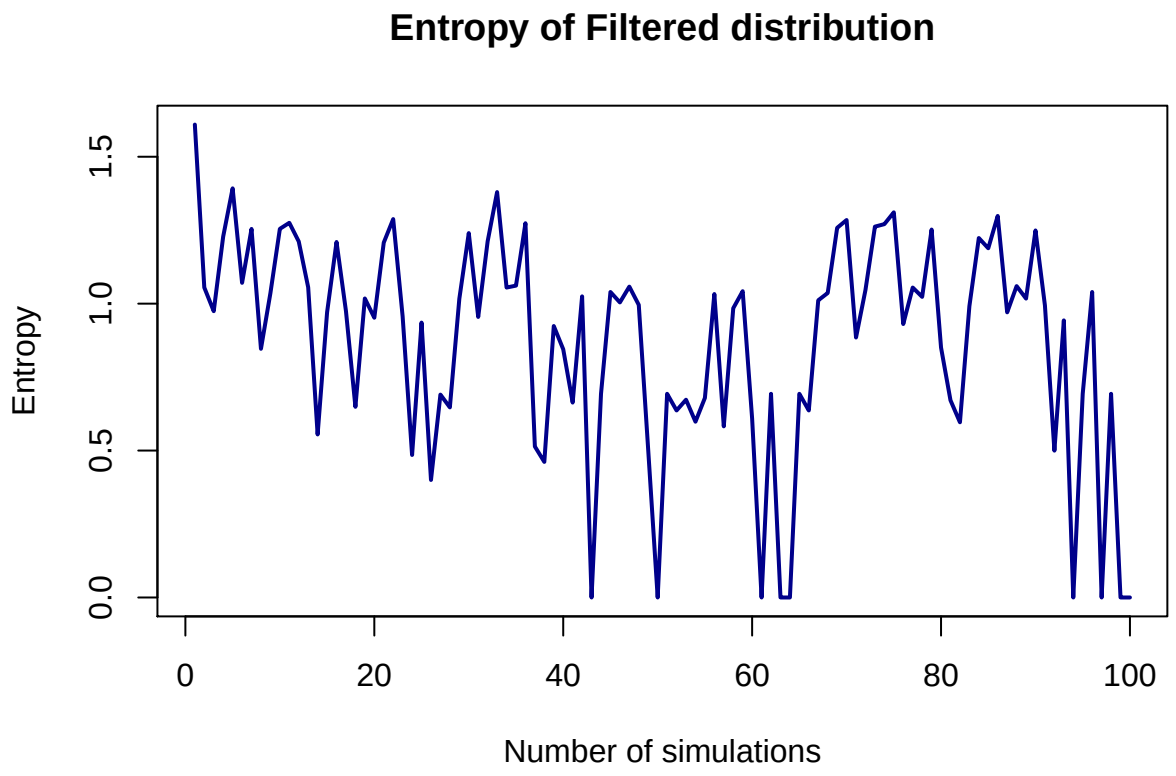
Is it true that the more observations you have the better you know where the robot is ?

Hint: You may want to compute the entropy of the filtered distributions with the function entropy.empirical of the package entropy.

```
res = forward_backward_viterbi(HMM_model,HMM_simulation$observation)
filtered = res$f

entropy = apply(filtered , 2 , entropy.empirical)

plot(x = 1:length(entropy), y = entropy,col = "darkblue",
     lwd = 2,xlab = "Number of simulations", ylab = "Entropy",type = "l",
     main = "Entropy of Filtered distribution")
```



As we can see in the plot, even as the number of observations increases the entropy is random. The entropy for probability distribution gives the measure of information content of the distribution. If the measure of uncertainty is more, then entropy value is more. Here if we need an advantage in accuracy the entropy should decrease gradually which is not happening here. So here we cannot leverage the increase in number of samples for better accuracy.

From the plot of entropy we see that at some points the entropy is very less but at next step it can be high. The reason for the above is noise in the observations.

Question 7

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

```
smoothed_dist = res$f*res$b
prob = transProbs%*%smoothed_dist[,100]
rownames(prob) = paste("sector" , 1:10)
colnames(prob) = c("probability")

cat("\nProbabilities of hidden states:\n\n")
```

```
##
## Probabilities of hidden states:
```

```
prob
```

```
##           probability
## sector 1         0.05
## sector 2         0.00
## sector 3         0.00
## sector 4         0.00
## sector 5         0.00
## sector 6         0.00
## sector 7         0.00
## sector 8         0.00
## sector 9         0.00
## sector 10        0.05
```

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(HMM)
library(entropy)
states = 1:10
symbols = 1:10
startProbs= rep(0.1,10)

# The probability of transition from current state to next state given current state
len = length(states)

# The probability to stay in that state or to move = 0.5

transProbs=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),
```

```
                    len,len,byrow = TRUE)
#emissionProbs is a (number of states)x(number of states)-sized matrix
#Emission probabilities: how likely the states are at any particular timestep
#emission probability hence probability of each state  = 1/5

emissionProbs = 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
                         ),len,len , byrow = TRUE)


HMM_model = initHMM(States = states, Symbols = symbols,
        startProbs=startProbs, transProbs=transProbs,
        emissionProbs=emissionProbs)
nSim = 100
HMM_simulation = simHMM(HMM_model, nSim)
# this method returns the o/p of forward, backward and viterbi algo
forward_backward_viterbi = function(model , observations){
 f = exp(forward(model, observations))
 f = prop.table(f,2)
 b = exp(backward(model, observations))
 b = prop.table(b,2)
 prob_path = viterbi(model,observations)

 return(list("f" = f,"b" = b,"prob_path"=prob_path))

}

#Function to return all three  paths
findpath = function(model , observations){
  res = forward_backward_viterbi(model, observations)
  f = res$f
  b = res$b
  probable_path = res$prob_path
  filtered_path = apply(f, 2, which.max)
  smoothed_path = apply(f*b, 2, which.max)
  return(list(filtered_path,smoothed_path,probable_path))
}

#Accuracy
accuracy = function(states , path){
  mtable = table(states, path)
  acc = sum(diag(mtable))/sum(mtable)
  return(acc*100)
}
path = findpath(HMM_model, HMM_simulation$observation)
```

```
filtered_path = path[[1]]
smoothed_path = path[[2]]
viterbi_path = path[[3]]
acc_filtering = accuracy(HMM_simulation$states,filtered_path)
acc_smoothed = accuracy(HMM_simulation$states,smoothed_path)
acc_viterbi = accuracy(HMM_simulation$states, viterbi_path)


cat("\n\nAccuracy of filtered probability distribution: ",acc_filtering,"%")
cat("\n\nAccuracy of smoothed probability distribution: ",acc_smoothed,"%")
cat("\n\nAccuracy of viterbi probability distribution: ",acc_viterbi,"%")


# Repeating  the above excercise with 100 simulations

accuracy_df = data.frame(matrix(NA , nrow = 20 , ncol = 3))
colnames(accuracy_df) = c("Filtering","Smoothing","Viterbi")

#Running for different simulations

for(i in 1:100){
  simulation = simHMM(HMM_model, 100)
  paths = findpath(HMM_model, simulation$observation)
  accuracy_df[i,1] = accuracy(simulation$states,paths[[1]] )
  accuracy_df[i,2] = accuracy(simulation$states,paths[[2]])
  accuracy_df[i,3] = accuracy(simulation$states, paths[[3]])

}

plot(x = 1:nrow(accuracy_df), y = accuracy_df[,1],col = "blue",
     lwd = 2,xlab = "N", ylab = "Accuracy %",type = "l",ylim = c(32,85),
     main = "Accuracy of Filtered, Smoothed and Viterbi distributions")
lines(accuracy_df[,2], col = "red", lwd = 2)
lines(accuracy_df[,3], col = "lightgreen" , lwd = 2)
legend(80,84,legend = c("Filtered","Smoothed","Viterbi"),
       col = c("blue","red","lightgreen"),lty=1,cex = 0.8)


res = forward_backward_viterbi(HMM_model,HMM_simulation$observation)
filtered = res$f

entropy = apply(filtered , 2 , entropy.empirical)

plot(x = 1:length(entropy), y = entropy,col = "darkblue",
     lwd = 2,xlab = "Number of simulations", ylab = "Entropy",type = "l",
     main = "Entropy of Filtered distribution")

smoothed_dist = res$f*res$b
prob = transProbs%*%smoothed_dist[,100]
rownames(prob) = paste("sector" , 1:10)
colnames(prob) = c("probability")

cat("\nProbabilities of hidden states:\n\n")
```

```
prob
```