Hidden Markov Lab2

Aman Kumar Nayak

9/20/2020

Task

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.

```
#Building Hidden Markov Model
library(HMM)

States =  1:10
Symbols = 1:10

#Since
#the robot is in one of the sectors and decides with equal probability to stay in that sector or move t
#we get probability of staying as 0.5 and probability of moving as 0.5

transProbs = 0.5*diag(10)
for(i in 1:ncol(transProbs))
{
    if(i != ncol(transProbs))
    {
      transProbs[i+1 , i]  = 0.5
    }else{
      transProbs[1 , i]  = 0.5
    }
}


colnames(transProbs) = States
rownames(transProbs) = States


#emissionProbs
#If the robot is in the sector i, then the device will report that the robot is in the sectors [i - 2,
#so robot can be in any sector i with probability of 0.2 (5/10)

emissionProbs = 0.2 * diag(10)
```

```r
n = ncol(emissionProbs)
for(i in 1:n){
  if(i-1 == 0) {
    Col_Indx = c((n-1) , n , i , i+1 , i+2)
  }else if(i-1 ==  1){
    Col_Indx = c( n , i-1 ,i , i+1 , i+2)
  }else if(i+1 == n){
    Col_Indx = c( i-2 ,i-1 ,i , i+1 , 1)
  }else if(i+1 > n) {
    Col_Indx = c( i-2 ,i-1 ,i , 1 , 2)
  }else{
    Col_Indx = c( i-2 , i-1 ,i , i+1 , i+2)
  }
  emissionProbs[i , Col_Indx] = 0.2
}#for

hmm = HMM::initHMM(States , Symbols , transProbs = transProbs , emissionProbs = emissionProbs)
```

```
## Hidden Markov Model for given scenario is

## $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 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5
##   2  0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   3  0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   4  0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
##   5  0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
##   6  0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
##   7  0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
##   8  0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
##   9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
##   10 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 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.

```
suppressWarnings(RNGversion("3.5.1"))
set.seed(12345)
nSim = 100
simulateHmm = simHMM(hmm , nSim)
```

```
## Simulated Values
```

```
## $states
##   [1]  9  9  9  9  8  7  6  5  5  5  4  3  2  2  2  2  2  2  2  1  1  1 10  9  9
##  [26]  8  8  8  7  6  5  4  3  2  2  2  1 10  9  8  8  7  7  7  6  6  6  6  6  5
##  [51]  5  5  4  3  2  2  1  1  1  1 10 10  9  9  9  8  8  7  7  7  7  7  6  6  6
##  [76]  5  5  5  5  4  4  4  3  3  3  3  2  1 10 10  9  9  8  8  8  8  8  8  7  6
##
## $observation
##   [1]  7 10  8 10 10  6  4  6  3  3  2  4 10  4 10  3  3  1  2  2  1  1 10 10 10
##  [26]  6  6  8  7  4  7  2  3 10  3  3 10 10  9  6 10  9  7  9  8  6  6  5  6  3
##  [51]  6  5  2  4  4 10 10  1  3 10  1  2  8  8  7 10  6  8  6  7  6  8  4  7  7
##  [76]  4  4  6  4  6  4  5  2  4  3  3  4  3 10 10 10  8  7  7  6  8  7  9  7  5
```

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.

Filtered and Smoothed Probability

Now in order to calculate filtered probability, I am using forward(hmm, observation)

In order to calculate, smoothed probabilities, we can use posterior(hmm, observation) where The posterior probability of being in a state X at time k can be computed from the forward and backward probabilities.

Now most probable path is calculated considering that robot can move from State X to State X ± 1 , so it can only move to next/previous stage and not like 2 or 3 stages ahead / behind.

```
#Since we have received probabilities in log, taking anti-log
filterdProb = exp(HMM::forward(hmm , simulateHmm$observation))

smoothedProb = HMM::posterior(hmm , simulateHmm$observation)

probablePath = HMM::viterbi(hmm, simulateHmm$observation)
```

Calculating Accuracy

```r
fnAccuracy = function(prob , Margin = 2 , States){
  #prob = probability of filter or smoother
  #Margin = margin = 2
  #States : States obtained during simulation

  #normalized prob
  normalisedProb = prop.table(prob , margin = Margin)

  #predicted states
  PredStates = apply(normalisedProb, MARGIN = Margin, which.max)

  #Calculate Accuracy in percentage when compared with  States obtained during simulation
  percAcc = sum(PredStates == States) / length(States)

  return(percAcc)
}
```

Table 1: Accuracy Table

Filter	Smoothing	Probable.Path
0.49	0.63	0.23

Question 5

#Generating Different Sample

```r
suppressWarnings(RNGversion("3.5.1"))
set.seed(9999789)
nSim = 100
simulateHmm2 = simHMM(hmm , nSim)
```

```r
filterdProb2 = exp(HMM::forward(hmm , simulateHmm2$observation))

smoothedProb2 = HMM::posterior(hmm , simulateHmm2$observation)
probablePath2 = HMM::viterbi(hmm, simulateHmm2$observation)

filterAccuracy2 = fnAccuracy(prob = filterdProb2 , Margin = 2 , States = simulateHmm2$states)
smoothAccuracy2 = fnAccuracy(prob = smoothedProb2 , Margin = 2 , States = simulateHmm2$states)
probPathAcc2 = sum(probablePath2 == simulateHmm2$observation)/length(simulateHmm2$observation)
```

Table 2: Accuracy Table with Different Sample

Filter	Smoothing	Probable.Path
0.53	0.67	0.27

Now it can be seen that, smoothed distribution is more accurate than filters distribution as filtered distribution is influenced by noise while in smoothing, because it have access to both past and future values thus it is less vulnerable to sudden changes and gives more accurate results.

Question 6

Entropy of a random variable is average level of "Information", "Surprise" or "Uncertainty" inherent in variables possible outcome.

Now a random variable X, with possible outcome $ x_1,x_2,....x_n $ which occur with probability of $P(x_1), P(x_2), ..., P(x_n)$, the entropy of X is formally defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i)$$

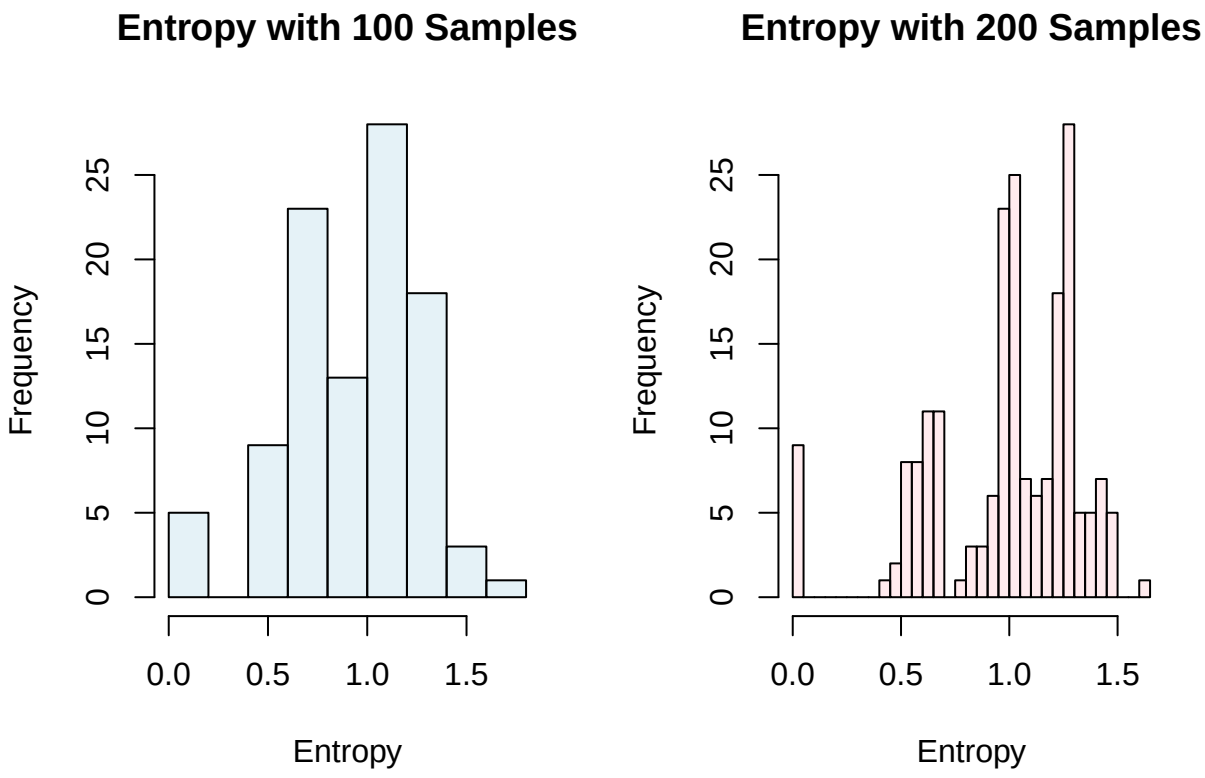
So higher entropy refer to more uncertainty, thus in order to check if we increase number of observations, do we have better understanding of where robot is, we would calculate entropy for filtered probabilities as they give result for robot location at time t with 1:t time information.

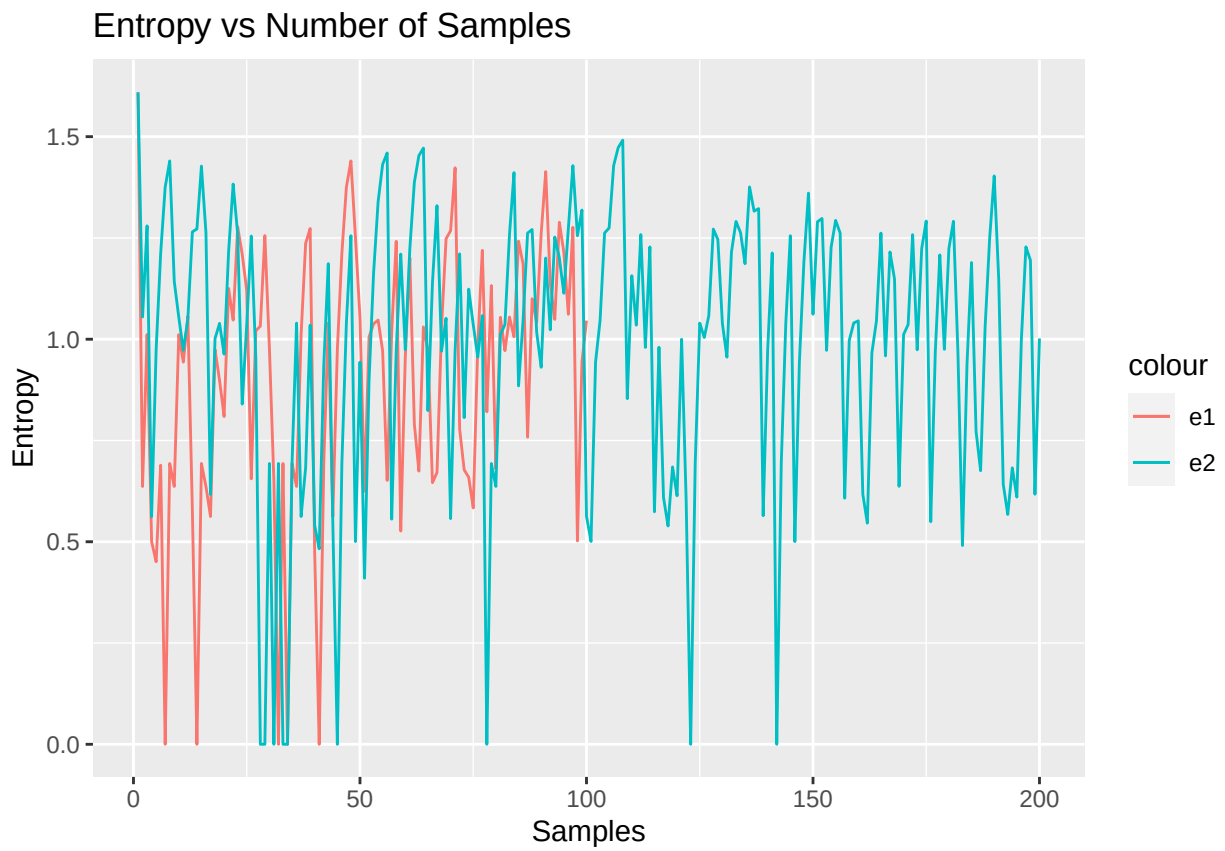
```
#genrating more samples with same seed values
suppressWarnings(RNGversion("3.5.1"))
set.seed(12345)
nSim = 200
simulateHmm3 = simHMM(hmm , nSim)
filterdProb3 = exp(HMM::forward(hmm , simulateHmm3$observation))

library(entropy)

e1Fwd = apply(X = filterdProb , MARGIN = 2 , FUN =  entropy.empirical)
#e1FwdBack = apply(X = smoothAccuracy , MARGIN = 2 , FUN =  entropy.empirical)

e2Fwd = apply(X = filterdProb3 , MARGIN = 2 , FUN =  entropy.empirical)
```





It can be seen from above plot of entropy (i.e. uncertainty) is reducing over time but with more and more data, entropy which is measure of uncertainity also increases. Thus gaining more and more obervation after a point won't help us in understanding where our robot is.

Question 7

Using prior knowledge of its position as time t = 100 and transition probability, we can estimate probabilites of hidden state at time stamp t = 101.

```
#normalized prob
normalisedProb = prop.table(filterdProb , margin = 2)

hiddenStates101 = normalisedProb[,100] %*% transProbs
cat("\n")
```

```
cat("Probabilities of the hidden states for the time step 101 is")
```

```
## Probabilities of the hidden states for the time step 101 is
```

```
cat("\n")
```

```
hiddenStates101
```

```
##      1 2 3         4        5         6        7 8 9 10
## [1,] 0 0 0 0.1051418 0.341903 0.3948582 0.158097 0 0  0
```

References

1. Package “HMM”
2. Entropy

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#Building Hidden Markov Model
library(HMM)

States =  1:10
Symbols = 1:10

#Since
#the robot is in one of the sectors and decides with equal probability to stay in that sector or move t
#we get probability of staying as 0.5 and probability of moving as 0.5

transProbs = 0.5*diag(10)
for(i in 1:ncol(transProbs))
{
    if(i != ncol(transProbs))
    {
      transProbs[i+1 , i]  = 0.5
    }else{
      transProbs[1 , i]  = 0.5
    }
}


colnames(transProbs) = States
rownames(transProbs) = States


#emissionProbs
#If the robot is in the sector i, then the device will report that the robot is in the sectors [i - 2,
#so robot can be in any sector i with probability of 0.2 (5/10)

emissionProbs = 0.2 * diag(10)
n = ncol(emissionProbs)
for(i in 1:n){
  if(i-1 == 0) {
    Col_Indx = c((n-1) , n , i , i+1 , i+2)
  }else if(i-1 ==  1){
    Col_Indx = c( n , i-1 ,i , i+1 , i+2)
  }else if(i+1 == n){
    Col_Indx = c( i-2 ,i-1 ,i , i+1 , 1)
  }else if(i+1 > n) {
    Col_Indx = c( i-2 ,i-1 ,i , 1 , 2)
  }else{
    Col_Indx = c( i-2 , i-1 ,i , i+1 , i+2)
  }
  emissionProbs[i , Col_Indx] = 0.2
```

```
}#for

hmm = HMM::initHMM(States , Symbols , transProbs = transProbs , emissionProbs = emissionProbs)

#Hidden Markov Model


cat("\n")
cat("Hidden Markov Model for given scenario is")
cat("\n")
cat("\n")
hmm

suppressWarnings(RNGversion("3.5.1"))
set.seed(12345)
nSim = 100
simulateHmm = simHMM(hmm , nSim)
cat("\n")
cat("Simulated Values")
cat("\n")
simulateHmm
cat("\n")

#Since we have received probabilities in log, taking anti-log
filterdProb = exp(HMM::forward(hmm , simulateHmm$observation))

smoothedProb = HMM::posterior(hmm , simulateHmm$observation)

probablePath = HMM::viterbi(hmm, simulateHmm$observation)


fnAccuracy = function(prob , Margin = 2 , States){
  #prob = probability of filter or smoother
  #Margin = margin = 2
  #States : States obtained during simulation

  #normalized prob
  normalisedProb = prop.table(prob , margin = Margin)

  #predicted states
  PredStates = apply(normalisedProb, MARGIN = Margin, which.max)

  #Calculate Accuracy in percentage when compared with  States obtained during simulation
  percAcc = sum(PredStates == States) / length(States)

  return(percAcc)
}


filterAccuracy = fnAccuracy(prob = filterdProb , Margin = 2 , States = simulateHmm$states)
smoothAccuracy = fnAccuracy(prob = smoothedProb , Margin = 2 , States = simulateHmm$states)
probPathAcc = sum(probablePath == simulateHmm$observation)/length(simulateHmm$observation)
```

```r
accDf = data.frame("Filter" = filterAccuracy ,
                   "Smoothing" = smoothAccuracy,
                   "Probable Path" = probPathAcc)

knitr::kable(accDf , caption = "Accuracy Table")

suppressWarnings(RNGversion("3.5.1"))
set.seed(9999789)
nSim = 100
simulateHmm2 = simHMM(hmm , nSim)
filterdProb2 = exp(HMM::forward(hmm , simulateHmm2$observation))

smoothedProb2 = HMM::posterior(hmm , simulateHmm2$observation)
probablePath2 = HMM::viterbi(hmm, simulateHmm2$observation)

filterAccuracy2 = fnAccuracy(prob = filterdProb2 , Margin = 2 , States = simulateHmm2$states)
smoothAccuracy2 = fnAccuracy(prob = smoothedProb2 , Margin = 2 , States = simulateHmm2$states)
probPathAcc2 = sum(probablePath2 == simulateHmm2$observation)/length(simulateHmm2$observation)


accDf = data.frame("Filter" = filterAccuracy2 ,
                   "Smoothing" = smoothAccuracy2,
                   "Probable Path" = probPathAcc2)

knitr::kable(accDf , caption = "Accuracy Table with Different Sample")

#genrating more samples with same seed values
suppressWarnings(RNGversion("3.5.1"))
set.seed(12345)
nSim = 200
simulateHmm3 = simHMM(hmm , nSim)
filterdProb3 = exp(HMM::forward(hmm , simulateHmm3$observation))

library(entropy)

e1Fwd = apply(X = filterdProb , MARGIN = 2 , FUN =  entropy.empirical)
#e1FwdBack = apply(X = smoothAccuracy , MARGIN = 2 , FUN =  entropy.empirical)

e2Fwd = apply(X = filterdProb3 , MARGIN = 2 , FUN =  entropy.empirical)




c1 <- rgb(173,216,230,max = 255, alpha = 80, names = "lt.blue")
c2 <- rgb(255,192,203, max = 255, alpha = 80, names = "lt.pink")
par(mfrow = c(1,2))
hist(e1Fwd , main = "Entropy with 100 Samples" , breaks = 10  ,col = c1 , xlab = "Entropy")
hist(e2Fwd , main = "Entropy with 200 Samples" , breaks = 50 ,col = c2 ,  xlab = "Entropy")


library(ggplot2)
ggplot()+
```

```
  geom_line(aes(x = 1:length(e1Fwd) , y = e1Fwd , color = "e1"))+
  geom_line(aes(x = 1:length(e2Fwd) , y = e2Fwd , color = "e2"))+
  xlab("Samples")+
  ylab("Entropy")+
  ggtitle("Entropy vs Number of Samples")


#normalized prob
normalisedProb = prop.table(filterdProb , margin = 2)

hiddenStates101 = normalisedProb[,100] %*% transProbs
cat("\n")
cat("Probabilities of the hidden states for the time step 101 is")
cat("\n")
hiddenStates101
```