Lab 2

Andreas Stasinakis(andst745)

September 23, 2019

Contents

```
library("HMM")
```

```
## Warning: package 'HMM' was built under R version 3.5.2
```

```
library("ggplot2")
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
library("entropy")
```

```
## Warning: package 'entropy' was built under R version 3.5.2
```

HIDDEN MARKOV CHAINS

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.

1.1 Build a HMM from scratch

Build a hidden Markov model (HMM) for the scenario described above.

```
#We have 10 different states
States = seq(1,10)
Symbols = seq(1,10)


#First we need the Transition matrix and the Emission Matrix
#Create the Transition matrix : P(z_t+1/p_z), where z is unobserved state
#From the instructions we know that being at one state the robot can either
#stay to the same state or move to the next
```

```r
#So for both cases we have the same probability as 0.5

#first we initialize the matrix adding the probs for staying at the same state
TransMatrix = diag(x = 1/2,nrow = 10,10)
#A for loop to add the probs for moving to the next state
for (i in 2:nrow(TransMatrix)){
  #at each row we add the prob to move to the next state
  TransMatrix[i-1,i] = 1/2
  #an if statement in order to go from the last state to the first one.
  if(i == nrow(TransMatrix)){
    TransMatrix[i,1] = 1/2
  }
}

#No we create the emission table. P(X_t/Z_t).
#So given the real state, what is the probabilities of the observed states
#Here we know that given the real state, the states in the interval
#[t-2,t+2] have 0.2 probs each.

EmissMatrix = diag(x = 1/5,nrow = 10,10)

#A for loop to add the probs for moving to the next state
for (i in 3:(nrow(EmissMatrix)-2)){
  #at each row we add the prob to move to the next state
  EmissMatrix[i,i-2] = 1/5
  EmissMatrix[i,i-1] = 1/5
  EmissMatrix[i,i+1] = 1/5
  EmissMatrix[i,i+2] = 1/5
  if(i == 3){
    #For the first row
    EmissMatrix[i-2,i-1] = 1/5
    EmissMatrix[i-2,i] = 1/5
    EmissMatrix[i-2,nrow(EmissMatrix)] = 1/5
    EmissMatrix[i-2,nrow(EmissMatrix)-1] = 1/5

    #for the second row
    EmissMatrix[i-1,i] = 1/5
    EmissMatrix[i-1,i+1] = 1/5
    EmissMatrix[i-1,i-2] = 1/5
    EmissMatrix[i-1,nrow(EmissMatrix)] = 1/5

    #For the last row
    EmissMatrix[nrow(EmissMatrix),i-2] = 1/5
    EmissMatrix[nrow(EmissMatrix),i-1] = 1/5
    EmissMatrix[nrow(EmissMatrix),nrow(EmissMatrix)-1] = 1/5
    EmissMatrix[nrow(EmissMatrix),nrow(EmissMatrix)-2] = 1/5

    #for the last -1 row
    EmissMatrix[nrow(EmissMatrix) -1,i-2] = 1/5
    EmissMatrix[nrow(EmissMatrix) -1,nrow(EmissMatrix)] = 1/5
    EmissMatrix[nrow(EmissMatrix) -1,nrow(EmissMatrix)-2] = 1/5
    EmissMatrix[nrow(EmissMatrix) -1,nrow(EmissMatrix)-3] = 1/5
  }
}



#We build the HMM from the scenario described using the initHMM function
set.seed(12345)
HMM = initHMM(States = States,
               Symbols = Symbols,
               transProbs= TransMatrix,
               emissionProbs = EmissMatrix)
```

In this task we have to build the Hidden Markov model for a specific scenario in order to detect the position of a robot. The robot can move from state *t* to state *t+1* or stay in the same state with equal probability. So first we create the transition matrix, which gives us the above probabilities. So from the transition matrix we can calculate the $P(Z_{t+1}/Z_t)$. The problem is that we are not able to observe the actual state. For that reason we need the emission matrix, which can give us the $P(X_t/Z_t)$. We also use as initial model(prior probabilities) equal to 0.1

1.2 Simulate from the HMM

Simulate the HMM for 100 time steps.

```
#We simulate from the HMM in order to create data 100 times
simHMM = simHMM(hmm = HMM,length = 100)
```

1.3 Forward - Backward Learning

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.

```
########## Functions we need for the tasks
#Function for implementing the Forward inference
#Input: The HMM object, obs : The observatios which we simulate,
#trueStates : The true hidden states we want to predict
forwardInf = function(HMM,obs,trueStates){

  len = length(obs)

  #Use forward inference to calculate the alpha function(Filtering)
  forw = forward(hmm = HMM,obs) #gives the log Probs

  #We have the log probs. So we take the expon of those probs and we also
  #We prop.table in order to normalize each column.
  #Because of computational probs, we found the path using the Logpros
  #Here we care only about the paths, which mean that we do not need either
  #to normalize them or tranform them to probs instead of Logprobs.
  #We keep them that way because for large Sample size the procedure produces
  #NAN because of underflow

  #filtering_path  = apply(forw, 2, which.max)
  #accur_filter = sum(trueStates == filtering_path)/len #we calaculate the accuracy
```

```r
  #We also need the filtering distribution though
  filtering_dist = prop.table(exp(forw),2) #We now have the probs for each time t

  #Accuracy for the Forward inference
  #We found which state has the higher Logprob and we classify it
  filtering_path  = apply(filtering_dist, 2, which.max)


  accur_filter = sum(trueStates == filtering_path)/len

  #We return i) the accuracy of the forward inference,
  # ii) The most probable path using filtering
  # iii) the filtering distribution P(Z_t/x_0:t)
  #So the filtering est is the prob of being at one state given the observed
  #data until this time
  return(list(accur_filter,filtering_path,filtering_dist))
}

#Function for backward inference, which will also give us the smoothing estim.
#Same input as in Forward

backwardInf = function(HMM,obs,trueStates){

  N = length(obs)

  #Now we will use Backward inference in order to found the smoothing probs
  backw = backward(HMM, obs)

  #For calculating the smoothing probabilities we need the product alpha and Beta
  forw = forwardInf(HMM,obs,trueStates)[[3]]

  #We have the logProbs, so we add the probabilities
  Unorm_smooting = exp(backw)*forw

  #Here we care only about the paths, which mean that we do not need either
  #to normalize them or tranform them to probs instead of Logprobs.
  #We keep them that way because for large Sample size the procedure produces
  #NAN because of underflow

  #Now we have to normalize that probabilities
  smoothing_distr = prop.table(Unorm_smooting,2)

  smoothing_path  = apply(smoothing_distr, 2, which.max)

  #Accuracy for the FB inference
  accur_smoothing = sum(trueStates == smoothing_path)/N

  #Second way: Using the posterior function which should give us the same res
  probs = posterior(HMM,obs)
  smoothing_path_prost  = apply(probs, 2, which.max)
  accur_posterior = sum(trueStates == smoothing_path_prost)/N

  #return the smoothing path, the accuracy of backward inference
  #The accuracy of the posterior dis(same as the backward inference)
  #and the smoothing distribution
  return(list(smoothing_path,accur_smoothing,accur_posterior, smoothing_distr))

}


#keep only the observation but not the true states
simulated_Obs = as.vector(simHMM$observation)
#store also the true States in order to compare with our predictions
trueStates = as.vector(simHMM$states)

#Now we will use the functions above to estimate the filtering distribution
#for each time step we have the distribution of the unobserved states,
#given all the previous observed data

filtered_dis = as.matrix(forwardInf(HMM,obs = simulated_Obs,trueStates))[[3]]

#For each time step, we have the distribution of the unobserved states,
#given all the observed data until the end.
#We can also use the function posterior instead of doing all that
smoothing_dis = as.matrix(backwardInf(HMM,obs = simulated_Obs,trueStates))[[4]]

viterbi_path = viterbi(hmm = HMM,simulated_Obs)
viterbi_path
```

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

In this task we compute the filtered distribution $P(Z_t|x_{0:t})$ and the smoothing probability distribution $P(z_t|x_{0:T})$ for each time step. We also calculate and prin the most probable path using the Viterbi algorithm.

1.4 Accuracy of Filtered and smoothed probability distribution

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.

```
#We run the function above so we have the accuracies
#Now we try for different N(sample size)
Simulation_HMM = function(HMM,N){

  #We simulate from the HMM in order to create data N times
  simHMM = simHMM(hmm = HMM,length = N)
  simulated_Obs = as.vector(simHMM$observation)
  trueStates = as.vector(simHMM$states)
  #FORWARD INFERENCE
  #use the function to calculate the alpha function,smoothing est and accuracy
  accur_filter = forwardInf(HMM,obs = simulated_Obs,
                             trueStates = trueStates)[[1]]

  #BACKWARD INFERENCE
  accur_smoothing = backwardInf(HMM,obs = simulated_Obs,
                                 trueStates = trueStates)[[2]]

  probs = backwardInf(HMM,obs = simulated_Obs,
                       trueStates = trueStates)[[3]]

  #Estimate the moste probable path for the 100 simulations
  viterbi_path = viterbi(hmm = HMM,simulated_Obs)
  accur_viterbi = sum(trueStates == viterbi_path)/N

  return(list("filtering" = accur_filter, "smoothing" = accur_smoothing,
              "viterbi" = accur_viterbi,"Posterior" = probs))

}

set.seed(12345)
accur_100 = Simulation_HMM(HMM,100)
print(accur_100)
```

```
## $filtering
## [1] 0.53
##
## $smoothing
## [1] 0.74
##
## $viterbi
## [1] 0.56
##
## $Posterior
## [1] 0.74
```

In this task, we calculate the accuracy of the filtered and smoothing distribution, and the accuracy of the most probable path as well. As we can see above, the smoothing distribution has the highest accuracy, while the filtering and the viterbi(most probable path) accuracy are lower and very close to each other. The posterior accuracy is the smoothing accuracy. We just use 2 different ways. For the first one, we implent the formulas from the forward - backward algorithm on our own, while for the second one we use the function posterior, which directly gives us the posterior distribution of the unobserved states.

1.5 Compare different samples. Smoothing vs Filtering and Viterbi

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 ?

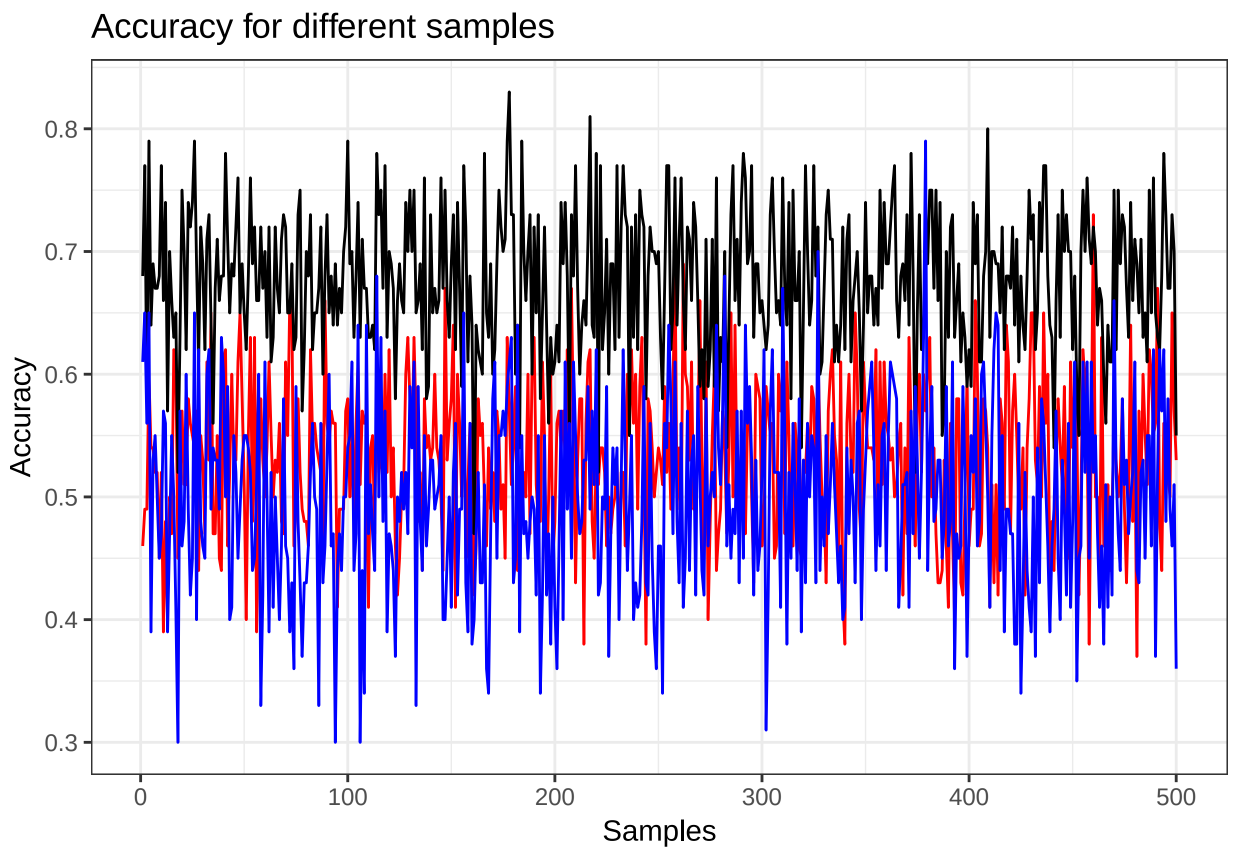
```
#For loop for generating different samples and test them
samples = 500
res = matrix(0,ncol = 4,nrow = samples)
for (i in 1:samples) {
```

```r
  res[i,] = unlist(Simulation_HMM(HMM,100))

}


df_plot = as.data.frame(res[,-4])
colnames(df_plot) = c("Filtering", "Smoothing", "Viterbi")
df_plot$X = c(1:nrow(res))


#Plot the accuracies for all 4 methods
ggplot(df_plot)+
  geom_line(mapping = aes(x = X,y = df_plot$Filtering), color = "red")+
  geom_line(mapping = aes(x = X,y = df_plot$Smoothing),color = "black")+
  geom_line(mapping = aes(x = X,y = df_plot$Viterbi), color = "blue")+
  labs(title = "Accuracy for different samples", x = "Samples",
       y = "Accuracy")+
  theme_bw()
```



As we can see from the plot, the accuracy for the smoothing distribution(black) is almost always higher than the accuracy of the filtered distribution(red) and the viterbi alogirthm(blue). This makes total sense because, we use all the data (past and future) in order to estimate the probabilities. On the other hand, for the filtered distribution, we only use the past data. For the Viterbi algorithm, which gives the most probable path, we can also observe that the accuracy is also lower than the smoothing distribution. A reason that this may happen is because again here only a part of the data is used(like filtering) but not all of them.

1.6 Compare the number of observations using the entropy

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.

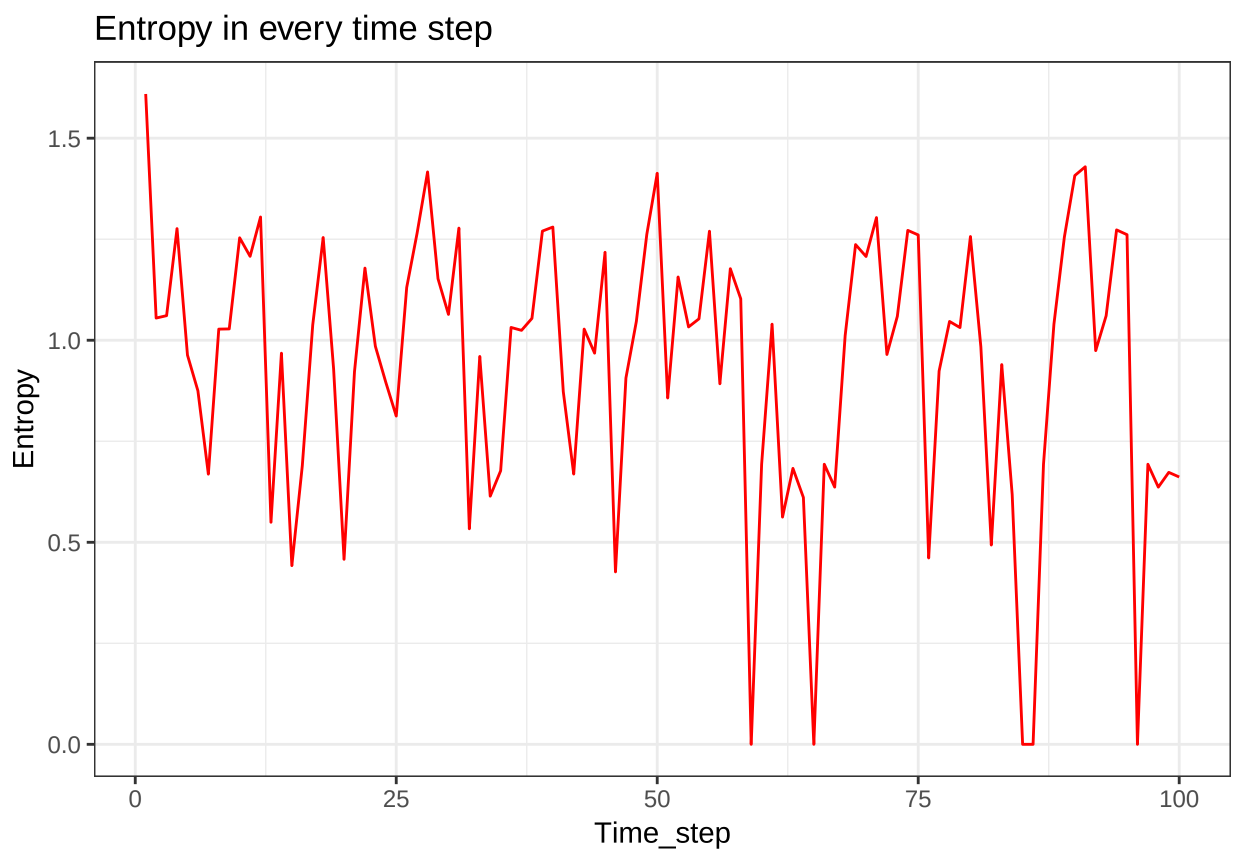
```r
#We calculate the entropy for each time step
#Each time step has each one distribution(Sums to 1), so we calculate the
#entropy for t1. After that we are in t2, which means we have more information.
#We want to see if that will give us less uncertaincy
entropies = rep(0,ncol(filtered_dis))

for (i in 1:ncol(filtered_dis)) {
  entropies[i] = entropy.empirical(filtered_dis[,i])


}


#plot the entropies for each time step
time_step = seq(1,100)

ggplot()+
  geom_line(mapping = aes(x = time_step, y = entropies), color = "red")+
  labs(title = "Entropy in every time step",x = "Time_step", y = "Entropy" )+
  theme_bw()
```



As we can see from the plot above, we can not say that the more observations the better the accuracy of the HMM would be. More specific, We plot the entropy of the distribution for each time step. The entropy of the filtered distribution is a meassure of the uncertainty. Therefore, the highest the Entropy is, the more uncertanty our distribution has. If the number of observations was important, we would have observed that while the time step is increasing, the entropy should reduce. In that case though, the entropy is fluctuating during time. Therefore, we can not agree that more observations, gives a better knowledge for the robot's position.

1.7 Predictions

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

```
pred = (TransMatrix%*%filtered_dis[,100])
pred
```

```
##         [,1]
##  [1,] 0.1875
##  [2,] 0.5000
##  [3,] 0.3125
##  [4,] 0.0000
##  [5,] 0.0000
##  [6,] 0.0000
##  [7,] 0.0000
##  [8,] 0.0000
##  [9,] 0.0000
## [10,] 0.0000
```

In this task we have to make a prediction for t+1 time step. We want to calculate the joint probability $P(Z_{t+1}, Z_t|X)$, where $X = X_0 : n$. We do the above derivation:

$$P(Z_{t+1}, Z_t|X) = P(Z_{t+1}|Z_t, X) \cdot P(Z_t|X)$$