Lab2

aswma317

2023-09-18

Question 1: Build a hidden Markov model (HMM)

```
#library(HMM)
#library(entropy)
####1: Build a HMM####
num_states <- 10#There are 10 possible States(unobserved)
num_symbols <- 10#There are 5 possible Symbols, but 10 states(observed)

#Uniform initial probability for each state
initial_probs <- rep(1/num_states, num_states)

#Transition probabilities(Unobserved states to unobserved states)
transition_probs <- matrix(0, num_states, num_states)
for (i in 1:num_states) {
  #It has 2 choices: move right or stay. Cannot move backward
  transition_probs[i,i] <- 1/2

  right <- ifelse(i+1 == 11, 1, i+1)#Considers circular reference
  transition_probs[i,right] <- 1/2
}

#Emission probabilities(Unobserved states to observed symbols)
emission_probs <- 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), num_states, num_states)

#Create the HMM
hmm_model <- initHMM(States = 1:num_states, Symbols = 1:num_symbols,
                     startProbs = initial_probs,
                     transProbs = transition_probs,
                     emissionProbs = emission_probs)
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

```r
####2: Simulate HMM for 100 steps####
#Simulate the HMM for 100 time steps
run_simulation <- function(num_steps, hmm_model){
  #set.seed(123)
  simulated_data <- simHMM(hmm_model, length = num_steps)

  # All the observations are within [i - 2, i + 2] distance
  return(simulated_data)
}
num_steps <- 100 #Time steps
simulated_data <- run_simulation(num_steps, hmm_model)
```

```
#Extract the sequence of states and observations
states_sequence <- simulated_data$states
observations_sequence <- simulated_data$observation

#Print the sequences (states and observations)
print("States Sequence:")
```

```
## [1] "States Sequence:"
```

```
print(states_sequence)
```

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

```
print("Observations Sequence:")
```

```
## [1] "Observations Sequence:"
```

```
print(observations_sequence)
```

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

As the problem states, all the observations are within [i - 2, i + 2] distance

Question 3: Compute the filtered and smoothed probability distributions for each of the 100 time points

```
####3: Compute filtered and smoothed probability distributions. Also most probable path####
get_probabilities <- function(hmm_model, simulated_data){

  #Extract the observations
  observations_sequence <- simulated_data$observation

  #Get the non-normalized forward probabilities(state*time steps)
  alpha <- exp(forward(hmm_model, observations_sequence))

  #alpha - alternate
  # Forward phase
  # a<-matrix(NA,nrow=100, ncol=length(States))
  # for(i in States){
  #   a[1,i]<-emissionProbs[sim$observation[1],i]*startProbs[i]
  # }
  #
```

```
# for(t in 2:100){
#   for(i in States){
#     a[t,i]<-emissionProbs[i,sim$observation[t]]*sum(a[t-1,]*transProbs[,i])
#   }
# }
#
# for(t in 1:100){
#   a[t,]<-a[t,]/sum(a[t,])
# }
#
# maxa=apply(a,1,which.max)
# table(maxa==sim$states)

#Get the backward probabilities
beta <- exp(backward(hmm_model, observations_sequence))

#beta - alternate
# Backward phase
# b<-matrix(NA,nrow=100, ncol=length(States))
# for(i in States){
#   b[100,i]<-1
# }
#
# for(t in 99:1){
#   for(i in States){
#     b[t,i]<-sum(b[t+1,]*emissionProbs[,sim$observation[t+1]]*transProbs[i,])
#   }
# }
#
# for(t in 1:100){
#   for(i in States){
#     b[t,i]<-b[t,i]*a[t,i]
#   }
#   b[t,]<-b[t,]/sum(b[t,])
# }
#
# maxb=apply(b,1,which.max)
# table(maxb==sim$states)


#Compute filtered probability distribution
filtered_probs <- apply(alpha, 2, prop.table)

#Compute smoothed probability distribution
smoothed_probs <- alpha * beta #Unnormalized
smoothed_probs <- apply(smoothed_probs, 2, prop.table)#Normalized

# Compute the most probable path
most_probable_path <- viterbi(hmm_model, observations_sequence)

#Viterbi - alternate
# viterbi <- function(observations, states, start_prob, transition_prob, emission_prob) {
#   T <- length(observations)
```

```
  #   N <- length(states)
  #
  #   #Initialize the Viterbi trellis and backpointer
  #   omega <- matrix(0, nrow = N, ncol = T)
  #   psi <- matrix(0, nrow = N, ncol = T)
  #
  #   #Initialization step
  #   for (i in 1:N) {
  #     omega[i, 1] <- log(start_prob[i]) + log(emission_prob[i, observations[1]])
  #     psi[i, 1] <- 0
  #   }
  #
  #   #Recursion step
  #   for (t in 2:T) {
  #     for (j in 1:N) {
  #       temp <- numeric(N)
  #       for (i in 1:N) {
  #         temp[i] <- log(transition_prob[i, j]) + omega[i, t - 1]
  #       }
  #       omega[j, t] <- log(emission_prob[j, observations[t]]) + max(temp)
  #       psi[j, t] <- which.max(temp)
  #     }
  #   }
  #
  #   #Termination step
  #   z_max <- numeric(T)
  #   z_max[T] <- which.max(omega[, T])
  #
  #   #Backtrack to find the most probable state sequence
  #   for (t in (T - 1):1) {
  #     z_max[t] <- psi[z_max[t + 1], t + 1]
  #   }
  #
  #   #Convert the state indices to state labels
  #   state_sequence <- states[z_max]
  #
  #   return(state_sequence)
  # }

  return(list(filtered_probs = filtered_probs,
              smoothed_probs = smoothed_probs,
              most_probable_path = most_probable_path))
}
prob_list <- get_probabilities(hmm_model, simulated_data)
prob_list$filtered_probs[,c(98,99,100)]
```

```
##       index
## states        98   99   100
##      1  0.0000000 0.00 0.000
##      2  0.0000000 0.00 0.000
##      3  0.3333333 0.25 0.125
##      4  0.6666667 0.75 0.500
##      5  0.0000000 0.00 0.375
```

```
##     6  0.0000000 0.00 0.000
##     7  0.0000000 0.00 0.000
##     8  0.0000000 0.00 0.000
##     9  0.0000000 0.00 0.000
##     10 0.0000000 0.00 0.000
```

```
prob_list$smoothed_probs[,c(98,99,100)]
```

```
##       index
## states  98   99   100
##     1  0.0 0.00 0.000
##     2  0.0 0.00 0.000
##     3  0.5 0.25 0.125
##     4  0.5 0.75 0.500
##     5  0.0 0.00 0.375
##     6  0.0 0.00 0.000
##     7  0.0 0.00 0.000
##     8  0.0 0.00 0.000
##     9  0.0 0.00 0.000
##     10 0.0 0.00 0.000
```

```
prob_list$most_probable_path
```

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

Question 4: Compute the accuracy of the filtered and smoothed probability distributions, and of the most probable path.

```
####4: Compute the accuracy of filtered, smoothed and probable path####
compute_accuracy <- function(prob_distr, true_path, probable_path = NULL){
  if(is.null(probable_path)){
    path <- apply(prob_distr,2, which.max)
    return((sum(path == true_path)/num_steps))
  }else{
    return((sum(probable_path == true_path)/num_steps))
  }
}

states_sequence <- simulated_data$states

#Get filtered probability distribution accuracy
prob_distr <- prob_list$filtered_probs
filtered_accuracy <- compute_accuracy(prob_distr, states_sequence)
print(paste0('Filtered accuracy:', filtered_accuracy))
```

```
## [1] "Filtered accuracy:0.69"
```

```r
#Get smoothed probability distribution accuracy
prob_distr <- prob_list$smoothed_probs
smoothed_accuracy <- compute_accuracy(prob_distr, states_sequence)
print(paste0('Smoothed accuracy:', smoothed_accuracy))
```

```
## [1] "Smoothed accuracy:0.78"
```

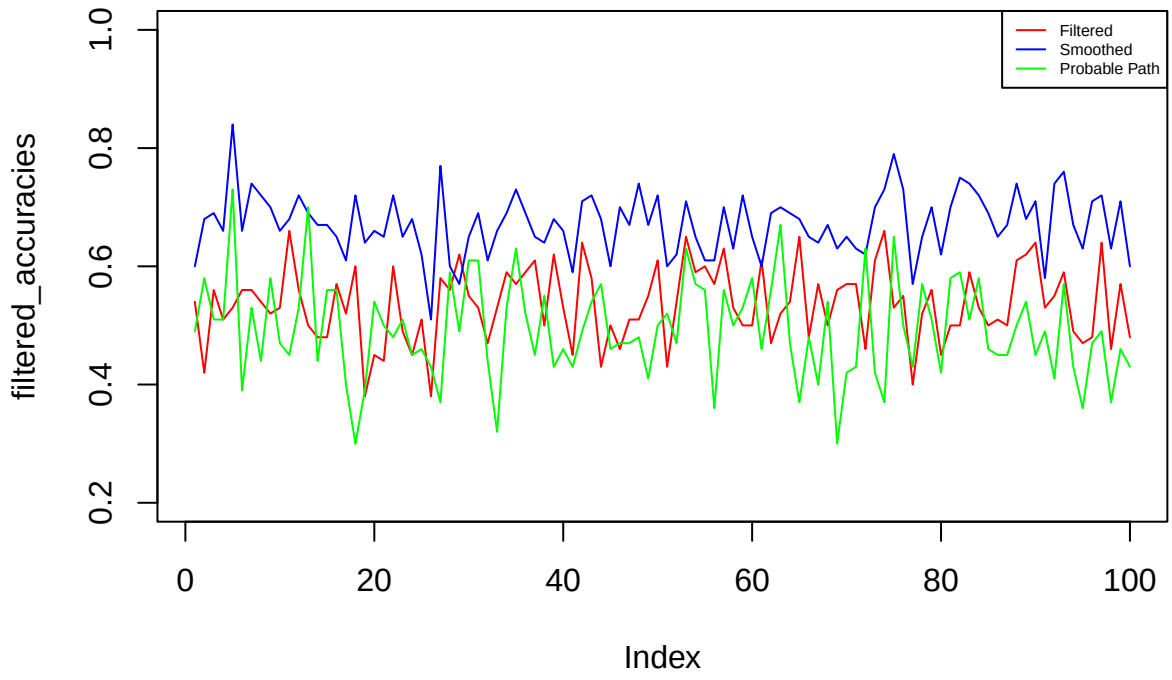
```r
#Get most probable path accuracy
most_probable_path <- prob_list$most_probable_path
probable_path_accuracy <- compute_accuracy(prob_distr, states_sequence,
                                           most_probable_path)
print(paste0('Probable path accuracy:', probable_path_accuracy))
```

```
## [1] "Probable path accuracy:0.43"
```

Question 5:Repeat the previous exercise with different simulated samples.

```r
#5: Repeat the previous exercise with different simulated samples
filtered_accuracies <- c()
smoothed_accuracies <- c()
probable_path_accuracies <- c()
for (i in seq(1,100,1)) {

  num_steps <- 100 #Time steps
  simulated_data <- run_simulation(num_steps, hmm_model)
  states_sequence <- simulated_data$states

  prob_list <- get_probabilities(hmm_model, simulated_data)

  #Get filtered probability distribution accuracy
  prob_distr <- prob_list$filtered_probs
  filtered_accuracies <- c(filtered_accuracies,
                           compute_accuracy(prob_distr, states_sequence))

  #Get smoothed probability distribution accuracy
  prob_distr <- prob_list$smoothed_probs
  smoothed_accuracies <- c(smoothed_accuracies,
                           compute_accuracy(prob_distr, states_sequence))

  #Get most probable path accuracy
  most_probable_path <- prob_list$most_probable_path
  probable_path_accuracies <- c(probable_path_accuracies,
                                compute_accuracy(prob_distr, states_sequence,
                                                 most_probable_path))
}
plot(filtered_accuracies,type = 'l', col = 'red',
     ylim = c(0.2, 1))
lines(smoothed_accuracies,type = 'l', col = 'blue')
lines(probable_path_accuracies,type = 'l', col = 'green')
#Adding legends
```

```
legend("topright",
       legend = c("Filtered", "Smoothed", "Probable Path"),
       col = c("red", "blue", "green"),
       lty = 1,
       cex = 0.5)
```



From the results above, we can see that in general, smoothed distributions are more accurate compared to filtered distributions, and this can be represented mathematically as well.

Smoothing is given by the following expression: $p(z^t|x^{0:T})$

While, filtering is given by the following expression: $p(z^t|x^{0:t})$

T in the smoothing expression means that the entire process has finished running and we calculate the probability of the hidden states given all the observations from start to the end of the process. While, the expression for filtering has t implying that the probability of hidden states are calculated only using the observations upto time step t and is hence more sensitive to short-term fluctuations or noisy data. In general, smoothed distributions are more accurate because they consider a broader context and hence are less sensitive to short-term fluctuations.

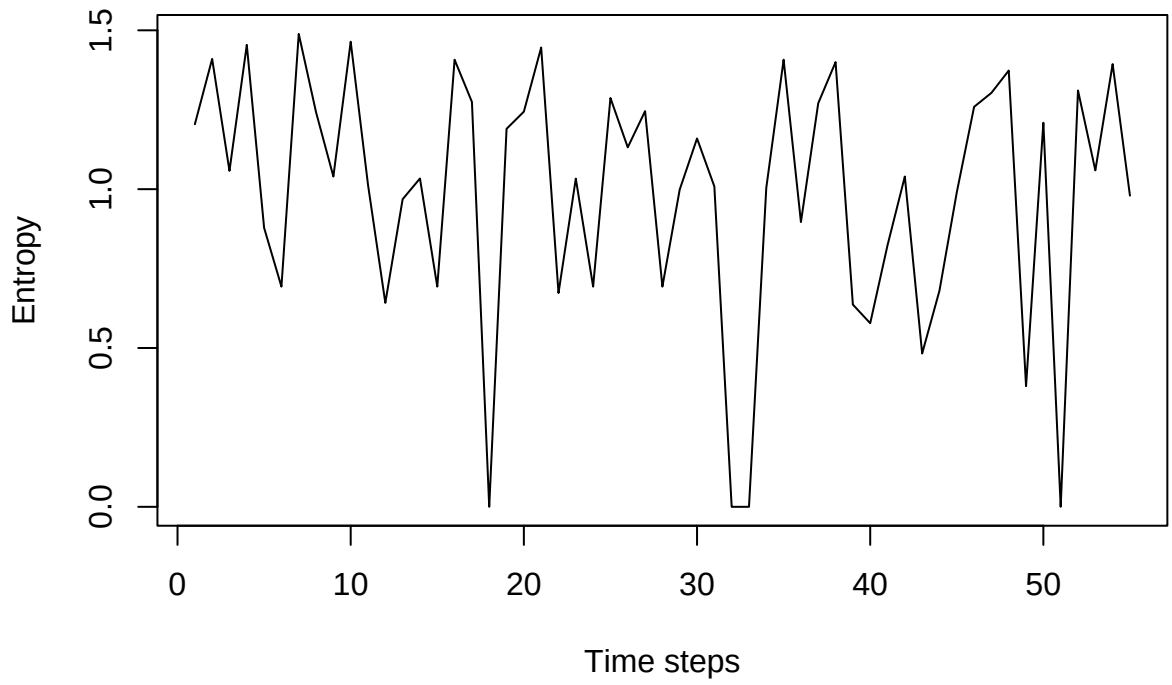
The Viterbi algorithm aims to find the single most probable sequence of states, by trying to maximize the joint probability of the observations and the states i.e., $p(x_1, ..., x_t, z_1, ..., z_t)$. It behaves the same way as the Smoothed distribution, in the sense that it accounts for the entire data while calculating the probability of the hidden state. However, the drawback and constraint with Viterbi algorithm is that it makes sure that the states in the probable path are continuous(no jumps) regardless of the probability distribution, while in the Smoothed distribution, we may find jumps from one state to non-neighboring states, because this states may be more probable. Because of this drawback and the noisy data, the probable path obtained from Viterbi algorithm is having lesser accuracy than the Smoothed distribution.

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 ?

In general, one would expect that later in time, as the number of observations increase we know better about the robot's position.

To do so, we are checking the entropy of the filtered distribution. However, since we have faulty/noisy observations(recall that there is only 20% probability that a given observation is correct) there is bound to be considerable amount of uncertainty even as the number of observations increase.

```
num_steps <- 300 #Time steps
simulated_data <- run_simulation(num_steps, hmm_model)

prob_list <- get_probabilities(hmm_model, simulated_data)

#Get filtered probability distribution accuracy
filtered_probs <- prob_list$filtered_probs

v_entropy <- c()
for (i in seq(30,num_steps,5)) {
  v_entropy <- c(v_entropy, entropy.empirical(filtered_probs[,i]))
}
plot(v_entropy,type = 'l', xlab = 'Time steps', ylab = 'Entropy')
```



Question 7: Compute the probabilities of the hidden states for the time step 101

To compute the probability of hidden states for time step 101 given time step 100, we can write it as:

$$p(z^{T+1}|x^{1:T}) = \sum_{z^T} p(z^{T+1}, z^T|x^{1:T})$$

that is,

$$p(z^{101}|x^{1:100}) = \sum_{z^{100}} p(z^{101}, z^{100}|x^{1:100})$$

$$= \sum_{z^{100}} p(z^{100}|x^{1:100}) \cdot p(z^{101}|z^{100})$$

That is, the probability distribution of the hidden state at time step 101 is given by the product of the 100th time step of the filtered distribution and the transition matrix.

```
num_steps <- 100 #Time steps
simulated_data <- run_simulation(num_steps, hmm_model)

prob_list <- get_probabilities(hmm_model, simulated_data)

#Get filtered probability distribution accuracy
filtered_probs <- prob_list$filtered_probs

#Product of the filtered prob at state 100 and the transition probabilities
filtered_probs[,100] %*% transition_probs
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0  0.4  0.5  0.1    0    0    0    0     0
```