732A96: Advance Machine Learning

LAB 2: HIDDEN MARKOV MODELS

Arian Barakat/ariba405

Background

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.

Questions

(1)

Question:
Build a HMM for the scenario described above

Answer:

See code and comments in appendix

! Note
In this particular HMM model, the robot is modeled to be able to move backward, forward to the next sector or stay in the same sector with equal probability.

(2)

Question:
Simulate the HMM for 100 time steps.

Answer:

See code and comments in appendix

(3)

Question:

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.

Answer:

See code and comments in appendix

(4)

Question:

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.]

Answer:

Table 1: Accuracy given Method

Filtered	0.55
Smoothed	0.58
Viterbi	0.36

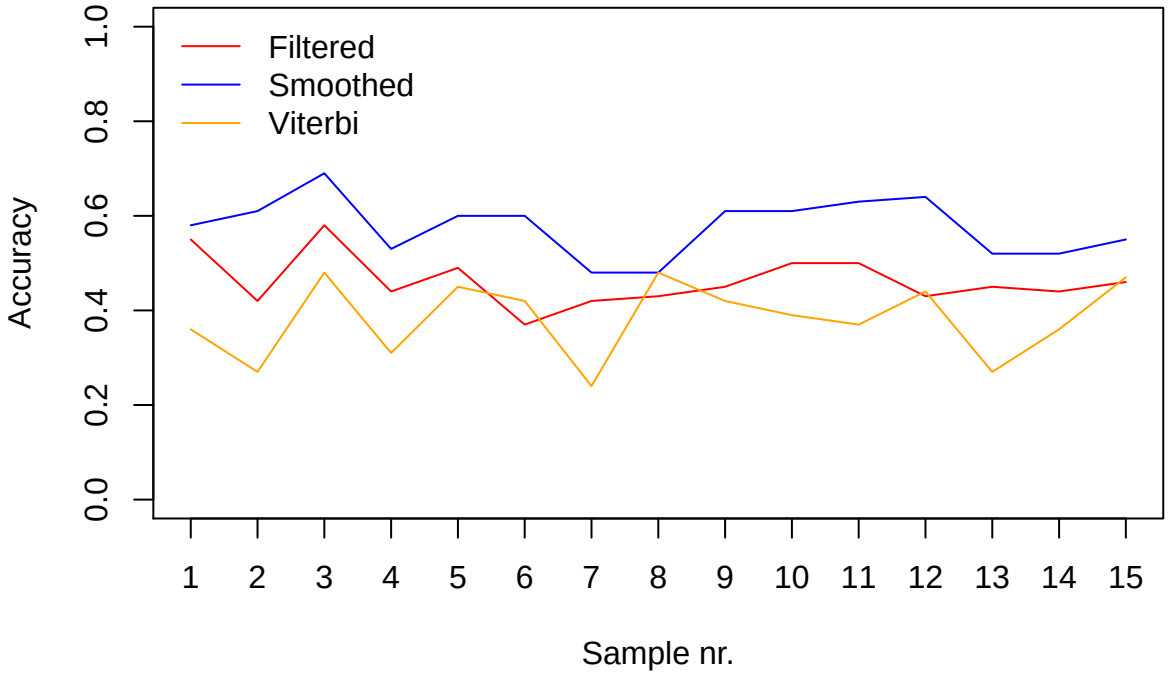

Figure 1: Accuracy with respect to Sample nr. and Method

(5)

Question:

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 ?

Answer:

Figure 1 displays the accuracy with respect to Sample nr. for the three different methods. As observed in the figure, the smoothed method seems to be generally more accurate than the other two methods. This observation can be linked to that the smoothed method uses all observed data when drawing inference about/predicting the hidden state (true location) of the robot while the other two use the observations sequentially. The former approach can be assumed to give better accuracy generally, however, it can be a non-pragmatic method in many cases.

Figure 2: Step plot of Entropy with respect to Observation number and Method

(6)

Question:

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.]

Answer:

Figure 2 displays the step plot of the empirical entropies with respect to observation number and method. From the figure, we can observe that the step plots stay pretty much at the same level, which indicates that our knowledge/certainty about the location of the robot does not improve as the number of observations increase. The observed result can be derived to the fact that the transition matrix stays the same over time (steps) and the states Z_{t+1} and Z_{t-1} are conditionally independent given Z_t.

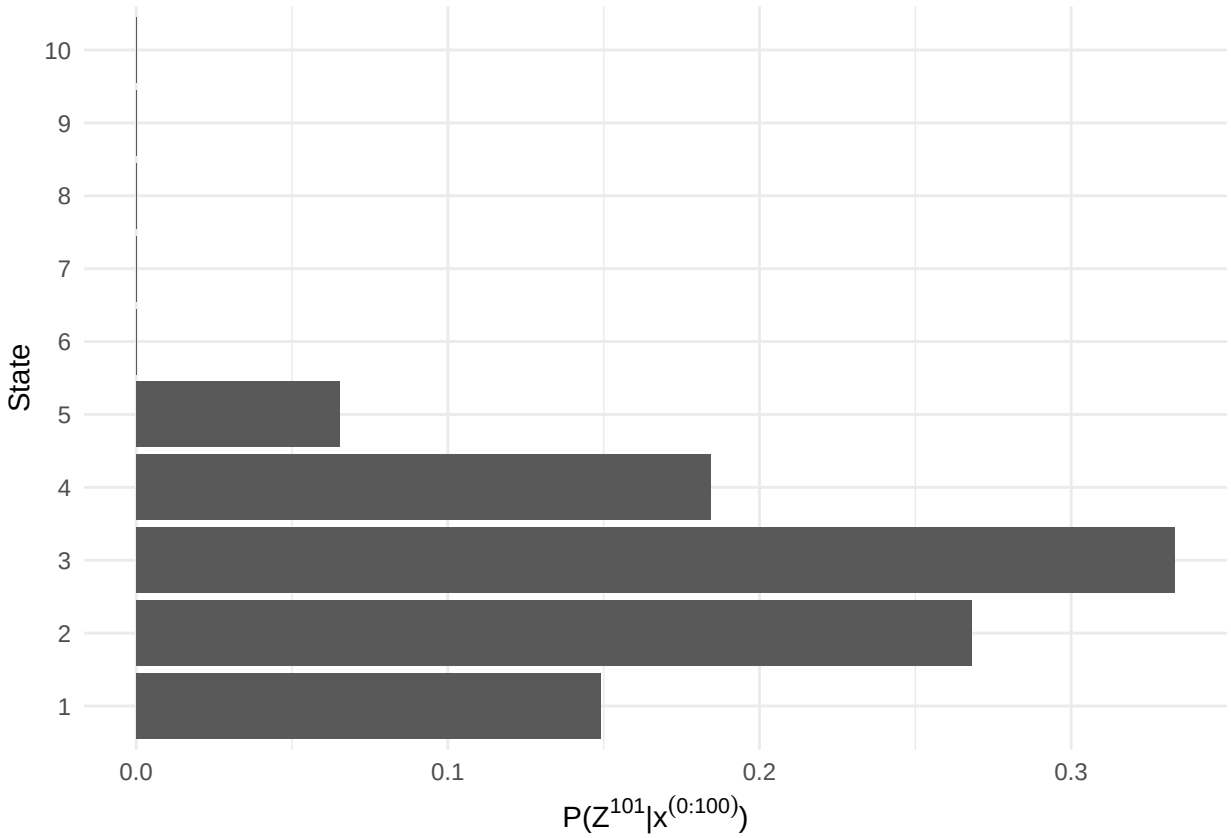

Figure 3: Probabilities of the hidden states for the time step 101

(7)

Question:

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

Answer:

Table 2: Probabilities of the hidden states for the time step 101

States	Pr
1	0.1490070
2	0.2680546
3	0.3333333
4	0.1843263
5	0.0652787
6	0.0000000
7	0.0000000
8	0.0000000
9	0.0000000
10	0.0000000

Appendix

```
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE,
                      fig.pos = "HTBP")

library(HMM)
library(knitr)
library(entropy)
library(ggplot2)
library(gridExtra)


# Question 1

# The transition Matrix

nrSectors <- 10
intervalLength <- 5
prTransChange <- 1/3 # Uniform
prEmiss <- 1/5 # Uniform


# Creating Trans.Matrix and filling it w/ pr
# + Assuming that the robot can go backward as well
A <- matrix(nrow = nrSectors,
            ncol = nrSectors)

# Creating Emiss.Matrix and filling it w/ pr
E <- matrix(nrow = nrSectors,
            ncol = nrSectors)


for(rows in 1:nrSectors){
  for(cols in 1:nrSectors){
    A[rows, cols] <- ifelse(abs(rows - cols) <= 1 || abs(rows - cols) >= 9,
                            prTransChange,
                            0)
    E[rows, cols] <- ifelse(abs(rows - cols) <= 2 || abs(rows - cols) >= 8,
                            prEmiss,
                            0)
  }
}



HMM_model <- initHMM(States = as.character(1:nrSectors),
                     Symbols = as.character(1:nrSectors),
                     transProbs = A,
                     emissionProbs = E)
```

```r
set.seed(123456)

HMM_model_sim<- simHMM(HMM_model,
                       length = 100)




# Question 3
simulated_states <- HMM_model_sim$states
simulated_obs <- HMM_model_sim$observation


logFilteredPr <- forward(HMM_model,
                         observation =  simulated_obs)

SmoothPr <- posterior(HMM_model,
                      observation = simulated_obs)


mostProbPath <- viterbi(HMM_model,
                        observation = simulated_obs)

methodPr <- list("Filtered" = logFilteredPr,
                 "Smoothed" = SmoothPr,
                 "Viterbi" = mostProbPath)







# Manual Implementation of the algorithms

#FB

emission_density <- function(x, z) {
  return(emission_probs[z, x])
}

transition_density <- function(z, previous_z) {
  return(trans_probs[previous_z, z])
}

transition_density2 <- function(z,previous_z){
  if( z == zt){

    return(0.5)

  } else if( (z + 1) == zt){
    return(0.5)
```

```
  } else return(0)

}

get_alpha_scalar <- function(zt, xt, previous_alpha, previous_z) {
  # Args:
  #   zt Scalar, hidden state at which to compute alpha.
  #   xt Scalar, observed state.
  #   previous_alpha Vector, alpha for all z_{t-1}.
  #   previous_z     Vector, all z_{t-1}.

  summation_term <- 0
  for (i in 1:length(previous_z)) {
    summation_term <- summation_term +
      previous_alpha[i] * transition_density(zt, previous_z[i])
  }

  alpha <- emission_density(xt, zt) * sum(summation_term)
  return(alpha)
}

get_alpha <- function(Zt, xt, previous_alpha, previous_z) {
  # Args:
  #   Zt Vector, hidden states at which to compute alpha.
  #   xt Scalar, observed state.
  #   previous_alpha Vector, alpha for all z_{t-1}.
  #   previous_z     Vector, all z_{t-1}.

  alpha <- sapply(Zt, function(zt) {
    get_alpha_scalar(zt, xt, previous_alpha, previous_z)
  })

  return(alpha)
}

get_beta_scalar <- function(zt, next_x, next_beta, next_z) {
  # Args:
  #   zt        Scalar, hidden state at which to compute alpha.
  #   next_x    Scalar, observed next state.
  #   next_beta Vector, alpha for all z_{t+1}.
  #   next_z    Vector, all z_{t+1}.

  summation_term <- 0
  for (i in 1:length(next_z)) {
    summation_term <- summation_term +
      next_beta[i] * emission_density(next_x, next_z[i]) * transition_density(next_z[i], zt)
  }

  # P(z_(t+1) | z_t) =
  # 0.5 if z_t = z_(t+1)
  # 0.5 if z_t = z_t + 1
  # 0 otherwise
```

```
  return(summation_term)
}

get_beta <- function(Zt, next_x, next_beta, next_z) {
  # Args:
  #   Zt        Vector, hidden states at which to compute alpha.
  #   next_x    Scalar, observed next state.
  #   next_beta Vector, alpha for all z_{t+1}.
  #   next_z    Vector, all z_{t+1}.

  beta <- sapply(Zt, function(zt) {
    get_beta_scalar(zt, next_x, next_beta, next_z)
  })

  return(beta)
}

fb_algorithm <- function(
  observations,
  emission_density,
  transition_density,
  possible_states,
  initial_density) {

  t_total <- length(observations)
  cardinality <- length(possible_states)

  # Alpha
  alpha <- matrix(NA, ncol=cardinality, nrow=t_total)

  for (i in 1:cardinality) {
    alpha[1, i] <-
      emission_density(observations[1], possible_states[i]) * initial_density[i]
  }

  for (t in 2:t_total) {
    alpha[t, ] <- get_alpha(possible_states, observations[t], alpha[t - 1, ], possible_states)
  }


  # Beta
  beta <- matrix(NA, ncol=cardinality, nrow=t_total)

  beta[t_total, ] <- 1

  for (t in (t_total - 1):1) {
    beta[t, ] <- get_beta(possible_states, observations[t + 1], beta[t + 1, ], possible_states)
  }

  return(list(alpha = alpha, beta = beta))
}

filtering <- function(alpha) {
```

```
  alpha / rowSums(alpha)
}

smoothing <- function(alpha, beta) {
  alpha * beta / rowSums(alpha * beta)
}

# Viterbi

get_omega <- function(Z, Omega, Z_next, x_next) {
  sapply(Z_next, function(z_next) {
    term1 <- log(eProbDensity(x_next, z_next))

    term2 <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega

    return(term1+ max(term2))
  })
}

get_phi <- function(Z, Z_next, Omega) {
  sapply(Z_next, function(z_next) {
    term <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega
    return(Z[which.max(term)])
  })
}

viterbi <- function(observations, possibleStates) {
  cardinality <- length(possibleStates)
  t_total <- length(observations)

  omega_0 <- vector("numeric", length = cardinality)
  for (i in 1:cardinality) {
    omega_0[i] <- log(initProbDensity(possibleStates[i])) +
      log(eProbDensity(observations[1], possibleStates[i]))
  }


  omega <- matrix(NA, nrow=t_total, ncol=cardinality)
  phi <- matrix(NA, nrow=t_total, ncol=cardinality)
  omega[1, ] <- omega_0

  for (i in 1:(t_total-1)) {
    omega[i+1, ] <- get_omega(possibleStates, omega[i, ], possibleStates, observations[i+1])
    phi[i+1, ] <- get_phi(possibleStates, possibleStates, omega[i, ])
  }

  mpp <- rep(NA, t_total)
  mpp[t_total] <- possibleStates[which.max(omega[t_total, ])]
  for (t in (t_total - 1):1) {
```

```
    mpp[t] <- phi[t + 1, possibleStates[mpp[t + 1]] == possibleStates]
  }

  return(list(path = mpp, omega = omega, phi = phi))

}


# Question 4

predictState <- function(methodList){

  predicted <- list()

  for(i in 1:length(methodList)){

    if(names(methodList)[i] == "Filtered"){
      predicted[[i]] <- as.vector(apply(exp(methodList[[i]]), MARGIN = 2, FUN = function(x){
        which.max(prop.table(x))
      }))
    }

    if(names(methodList)[i] == "Smoothed"){
      predicted[[i]] <- as.vector(apply(methodList[[i]], 2, which.max))
    }

    if(names(methodList)[i] == "Viterbi"){
      predicted[[i]] <- as.numeric(methodList[[i]])
    }

  }

  predicted <- lapply(predicted, as.character)
  names(predicted) <- c("Filtered",
                        "Smoothed",
                        "Viterbi")

  return(predicted)

}

probableState <- predictState(methodPr)

calcAccur <- function(predictedState, trueStates){
  accurMethod <- lapply(predictedState, FUN = function(x){
    sum(x == trueStates)/length(x)
  })

  return(unlist(accurMethod))
}


estimatedAccur <- calcAccur(probableState, simulated_states)
```

```r
kable(as.matrix(estimatedAccur), caption = "Accuracy given Method")
```

```r
#itSamples <- seq(100, 1500, 100)
itSamples <- rep(100, 15)
accurMatrix <- matrix(nrow = length(itSamples),
                      ncol = 3)
set.seed(123456)
for(iter in 1:length(itSamples)){

  HMM_model_sim_temp <- simHMM(HMM_model,
                               length = itSamples[iter])

  simulated_states_temp <- HMM_model_sim_temp$states
  simulated_obs_temp <- HMM_model_sim_temp$observation


  logFilteredPr_temp <- forward(HMM_model,
                        observation =  simulated_obs_temp)

  SmoothPr_temp <- posterior(HMM_model,
                     observation = simulated_obs_temp)


  mostProbPath_temp <- viterbi(HMM_model,
                         observation = simulated_obs_temp)

  methodPr_temp <- list("Filtered" = logFilteredPr_temp,
                        "Smoothed" = SmoothPr_temp,
                        "Viterbi" = mostProbPath_temp)

  probableState_temp <- predictState(methodPr_temp)

  accurMatrix[iter,] <- calcAccur(probableState_temp, simulated_states_temp)
}

xGrid <- 1:length(itSamples)
plot(x = xGrid,
     y = accurMatrix[,1],
     ylab = "Accuracy",
     xlab = "Sample nr.",
     col = "red",
     type = "l",
     ylim = c(0,1),
     xaxt = "n")
axis(1, at = xGrid)
lines(x = xGrid,
     y = accurMatrix[,2],
     col = "blue",
     type = "l")
```

```r
lines(x = xGrid,
     y = accurMatrix[,3],
     col = "orange",
     type = "l")
legend("topleft",
       legend = names(methodPr),
       col = c("red", "blue", "orange"),
       lty = rep(1,3),
       bty = "n")


# Question 6

calcEntropy <- function(methodList){

  entrop <- list()

  for(i in 1:length(methodList)){

    if(names(methodList)[i] == "Filtered"){
      entrop[[i]] <- as.vector(apply(exp(methodList[[i]]), MARGIN = 2, FUN = function(x){
        entropy.empirical(prop.table(x))
      }))
    }

    if(names(methodList)[i] == "Smoothed"){
      entrop[[i]] <- as.vector(apply(methodList[[i]], 2, entropy.empirical))
    }


  }

  names(entrop) <- c("Filtered","Smoothed")

  return(entrop)

}


itSamples <- 100
entropies <- list()

set.seed(123456)
for(iter in 1:length(itSamples)){

  HMM_model_sim_temp <- simHMM(HMM_model,
                               length = itSamples[iter])

  simulated_states_temp <- HMM_model_sim_temp$states
  simulated_obs_temp <- HMM_model_sim_temp$observation


  logFilteredPr_temp <- forward(HMM_model,
```

```
                                observation =  simulated_obs_temp)

  SmoothPr_temp <- posterior(HMM_model,
                             observation = simulated_obs_temp)


  methodPr_temp <- list("Filtered" = logFilteredPr_temp,
                        "Smoothed" = SmoothPr_temp)

  entropies[[iter]] <- calcEntropy(methodPr_temp)

}


dfList <- list()


for(iter in 1:length(itSamples)){

  dfList[[iter]] <- data.frame(Entropy = c(entropies[[iter]][[1]],entropies[[iter]][[2]]),
                               Method =  as.factor(c(rep("Filtered", itSamples[iter]),
                                                     rep("Smoothed", itSamples[iter]))),
                               Iteration = as.character(itSamples[iter]),
                               Observation = 1:itSamples[iter])

}


mergedDF <- Reduce(rbind, dfList)

ggplot(data = mergedDF) +
  geom_step(aes(y = Entropy, col = Method, x = Observation)) +
  xlab("Observation #") +
  scale_fill_manual(values = c("red", "blue")) +
  theme_minimal() +
  facet_grid(~Method)



# Question 7


# Will be using the table from the posterior function at step 100

hiddenStateStep101 <- SmoothPr[,100] %*% A


ggplot(data = data.frame(pr = as.vector(hiddenStateStep101),
                         states = as.factor(1:length(hiddenStateStep101)))) +
  geom_bar(aes(y = pr, x = states), stat = "identity") +
  xlab("State") + ylab(expression(paste("P(",Z^101,"|",x^(0:100),")"))) +
  coord_flip() +
  theme_minimal()
```

```
kable((data.frame(States = as.factor(1:length(hiddenStateStep101)),
                  Pr = as.vector(hiddenStateStep101))),
      caption = "Probabilities of the hidden states for the time step 101")
```