TDDE15 - Lab 2

Martin Friberg - marfr370

2021/09/20

Lab introduction

Build a hidden Markov model (HMM) for a scenario when we have a robot that walks around a ring. The ring is divided into 10 sectors. 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. 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.

Assignment 1

Build the hidden Markov model (HMM) for the scenario

The hidden markov model (HMM) was constructed by using the HMM library together with settings:

- States to S: 0-9,
- Observations to S: 0-9,
- Equal starting probability for all the states, i.e 0.1,
- A transition matrix corresponding to that the robot might move one state or forward or stay in the current state with equal probability,
- An emission matrix corresponding to where the signal might come from given the true state. If the robot is in state i, the signal might come from state [i-2, i+2] with equal probability (0.2).

The HMM model was then initialized with the initHMM function where the arguments were applied.

Assignment 2-5

The HMM model was simulated for a 100 time steps. This was done a 100 times in order to get good estimates of the different algorithms:

- filtered
- smoothed
- viterbi

From the values obtained by the simulation, the observations were used, but the hidden states were discarded from future calculations.

The forward and backward algorithms are used in order to obtain alpha and beta, which in their turn are used in order to calculate the filtered and smoothed probability distributions.

The forward algorithm:

$\alpha(z^0) := p(x^0|z^0)p(z^0)$

For $t = 1, .., T$ do

- $\alpha(z^t) := p(x^t|z^t) \sum_{z^{t-1}} \alpha(z^{t-1}) p(z^t|z^{t-1})$

Return $\alpha(z^0), .., \alpha(z^T)$

The backward algorithm:

$\beta(z^T) := 1$

For $t = T-1, .., 0$ do

- $\beta(z^t) := \sum_{z^{t+1}} \beta(z^{t+1}) p(x^{t+1}|z^{t+1}) p(z^{t+1}|z^t)$

Return $\beta(z^0), .., \beta(z^T)$

The values returned from the forward and backward algorithms in R are in log scale. Therefore, the alpha and beta values returned were then exponentialized by the function exp(value).

Filtering

The Filtering algorithm is used in order to answer the question "Where was is the robot at time t?". The question we are trying to answer can be further explained by saying that we have observed all emission signals up to the time point t and are trying to predict what the most likely state is at time t. By using the forward algorithm we discard all the emission signals from time t+1 up until time T. $p(z^t|x^{0:t})$

The filtering probabilities were the attained by the function:

$p(z^t)|x^{0:t}) = \frac{\alpha(z^t)}{\sum_{z^t} \alpha(z^t)}$

Smoothing

By using the backward algorithm we are trying to answer the question "Where was the robot at time t". By using the smoothed algorithm we take into consideration all emission signals from time 0-T. The equation we are trying to answer is therefore $p(z^t|x^{0:T})$

The smoothing probabilities were the attained by the function:

$p(z^t|x^{0:T}) = \frac{\alpha(z^t)\beta(z^T)}{\sum_{z^t} \alpha(z^t)\beta(z^t)}$

Viterbi algorithm

The result from the Viterbi algorithm answers the question "Which is the most likely path that the robot took". This means that the algorithm isn't trying to estimate the probability of the robot being in a state at time t, but instead is measuring the entire probability chain. $z^{0:T}_{max} = argmax_{z^{0:T}} p(z^{0:T}|x^{0:T})$

$w(z^0) := \log\, p(z^0) + \log\, p(x^0|z^0)$

For $t = 0, ., T-1$ do

- $w(z^{t+1} := \log\, p(x^{t+1}|z^{t+1}) + \max_{z^t}[\log\, p(z^{t+1}|z^t) + w(z^t)]$
- $\psi(z^{t+1}) := \text{argmax}_{z^t}[\log\, p(z^{t+1}|z^t) + w(z^t)$

$z^t_{max} = \text{argmax}_{z^T}\ w(z^t)$

For $t = T-1, .., 0$ do

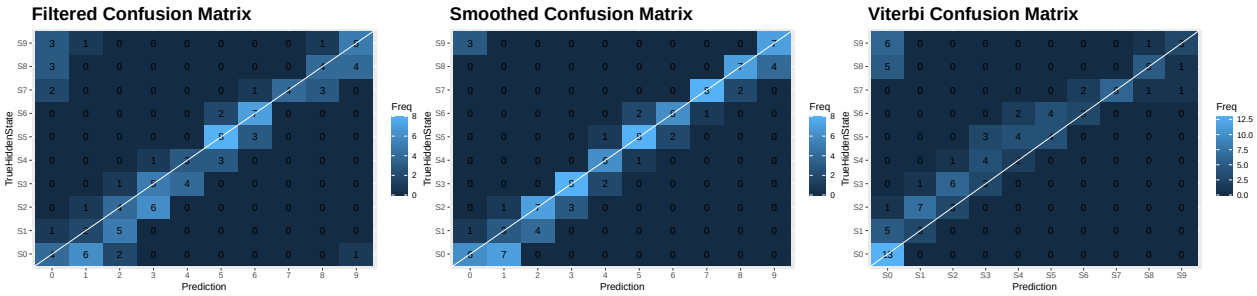
- $z^t_{max} := \psi(z^{t+1}_{max})$

Return $z^{0:T}_{max}$

Results

To measure the accuracy of predictions from the algorithms, the misclassification rate was used. By simulating the hidden markov model a hundred times, good estimates of the accuracy of the algorithms was obtained.

The predictions made by the filtered and smoothing algorithms were calculated by taking the state with the highest probability in each time step. These states were the compared to the hidden states from the simulation in order to calculate the misclassification rate for the time series. For the viterbi algorithm, the viterbi function from the package HMM was used, which returns the most likely path the robot has taken. To give an illustration, the confusion matrices for the different algorithms given one simulation of the hidden markov model is shown below.



As can be seen from the confusion matrices, most of the misclassifications from the algorithms are due to that the algorithm predicts the robot to be in a neighboring state of the actual hidden state. Even though these confusion matrices are based on a single simulation, hints about the excellence of the smoothing algorithm compared to the others can be drawn.

Comparison of algorithms

To gain further insight into the performance of the algorithms, the misclassification rate for each of the algorithms was calculated and averaged over 100 simulations. The results are shown in the graph below.

In general, the smoothed distributions should be more accurate than the filtered distributions and the most probable paths. Why?

The smoothing distribution indeed seems to be more accurate than the filtered distribution and the most probable path. This is due to the fact that the smoothing distribution uses all the observations from time steps 1-100 for both the forward and backward algorithms in order to predict with what probability the robot was in state x at time t. In comparison, the filtered algorithm only takes into consideration all the observations up until the given time. For example, if the probability for the given states are to be calculated at time 50, the filtered distribution only takes observations up until time 50 into consideration. The Viterbi distribution (the most probable path) do not use only the probability distribution at a certain time step, but also has to construct a path that coincides with the movement patterns of the robot. In our scenario, this generates a poorer prediction result that the other distributions can manage to provide.

Assignment 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 ?

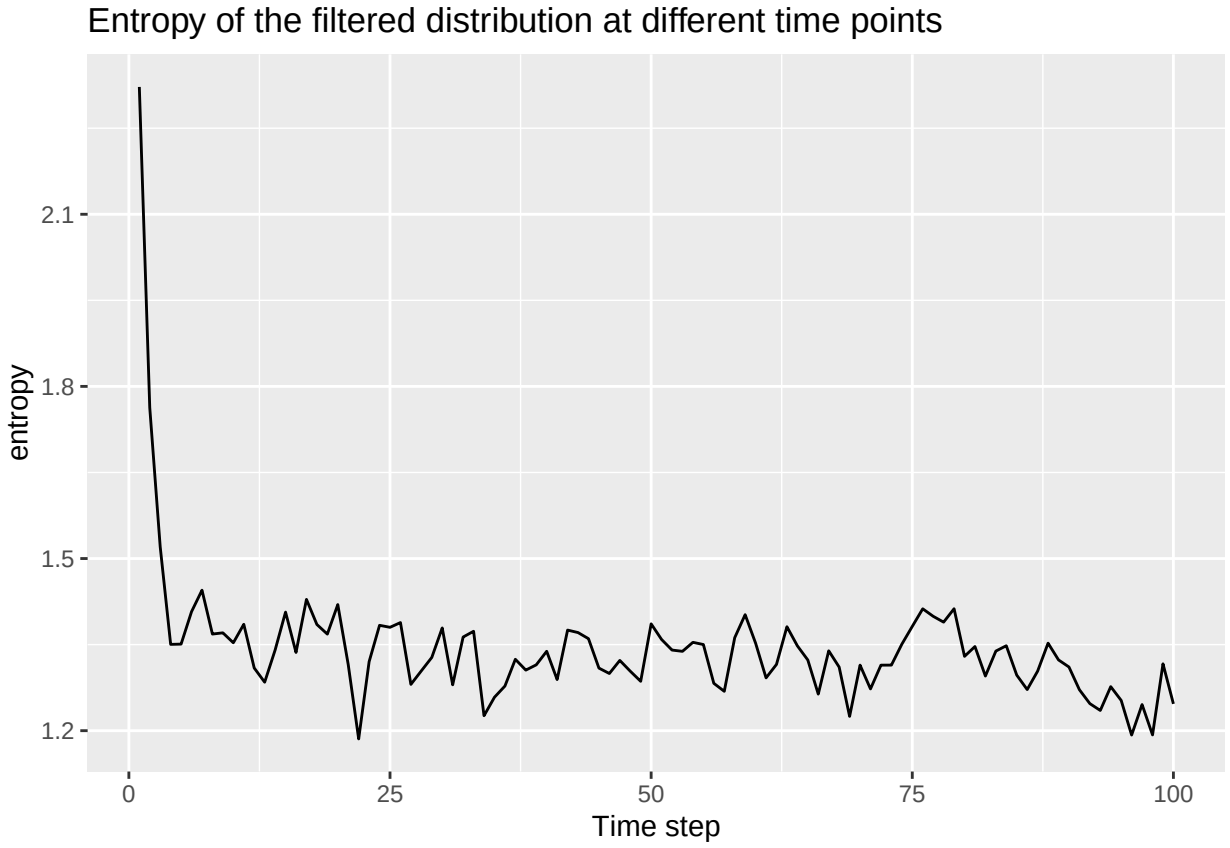
Shannon entropy

The Shannon entropy calculates how much information that we receive from observing an outcome. The less likely the outcome is, the more information we receive. A simplified example of calculating the entropy in bits is how many true and false questions it would take to know the outcome. For each question, the entropy increases by 1 bit. This is due to the fact that 1 bit can represent 2 facts (0 or 1). For a coin toss, the minimum surprise (minimum entropy) would be if the probability for heads was 1 and for tails 0 or vice versa, since we know the outcome beforehand. The maximum entropy on the other hand would be if the

probability of heads and tails were equal.

In our case the entropy decreases when the probability distribution between the states become more concentrated, i.e the information we receive by observing each state decreases.

To calculate whether or not the knowledge of the location of the robot increases for every time step, the filtered distribution was simulated a hundred times and the Shannon entropy at each time step was averaged over the simulations. This generated the following plot.



As can be seen from the plot, the entropy decreases during the first 4 time steps but stagnates thereafter and only fluctuates. The interpretation of this is that we become more sure of the location of the robot during the first 4 time steps but that we gain no further knowledge after that. The Shannon entropy fluctuates a lot which is due to the fact that depending on what state the emission signals in subsequent time steps are.

Assignment 7

Compute the probabilities of the hidden states for the time step 101.

In order to compute the probabilities of the hidden states for the following time steps, the probabilities of the observations in the current time steps must be multiplied with the transition matrix. This means that $p(SX_{t+1}) = p(SX_t) * 0.5 + p(SX_{t-1}) * 0.5$.

In our case, the probability of the robot being in a hidden state at time t (100) is

```
##  [1] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.83 0.17 0.00
```

The transition matrix then gives us the probability of the robot being in a certain state at time t+1 (101)

```
##  [1] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.417 0.500 0.083
```

Code

```
library(HMM)
options(digits=2)
library(ggplot2)
library("dplyr")
library("tidyr")
# The possible states that the robot might be in. The states goes in a circle.
states <- c('S0', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7', 'S8', 'S9')

# Symbols, i.e in what state does our signal say we are?
observations <- c('S0', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7', 'S8', 'S9')

# The robot starts in a state with equal probability
start_probability <- rep(0.1, 10)

# Function for creating the transition matrix. There is a 50% chance that
# the robot stays in the current state and a 50% chance that the robot moves
# on to the next state. If the state is 10, the robot moves to state 1.
create_trans <- function(){
  matrix <- matrix(rep(0,100), nrow=10, ncol=10)
  diag(matrix) <- 0.5
  diag(matrix[1:9,2:10]) <- 0.5
  matrix[10,1] <- 0.5
  return(matrix)
}

transProbs <- create_trans()

# Constructing a matrix that contains the probabilities that the robot is in a
# state SX when the signal comes from state SY.
emissionProbs <- matrix(rep(0,100), nrow=10, ncol=10)
diag(emissionProbs) <- 0.2
diag(emissionProbs[1:9,2:10]) <- 0.2
diag(emissionProbs[1:8,3:10]) <- 0.2
diag(emissionProbs[2:10,1:9]) <- 0.2
diag(emissionProbs[3:10,1:8]) <- 0.2
emissionProbs[1,9:10] <- 0.2
emissionProbs[2,10] <- 0.2
emissionProbs[9:10,1] <- 0.2
emissionProbs[10,2] <- 0.2


# The hidden markov model
hmm_model <- initHMM(states, observations, start_probability, transProbs, emissionProbs)

# Computing the position where the robot IS most likely to be at time t.
# Filtered algorithm
filtered_algo <- function(alpha, forward){
  filtered <- matrix(rep(0,1000), ncol=100, nrow=10)
  for (i in 1:dim(alpha)[2]){
    filtered[,i] <- alpha[,i] / sum(alpha[,i])
  }
```

```r
  return(filtered)
}

# Computing the position where the robot WAS most likely to be at time t.
# Smoothed algorithm
smoothed_algo <- function(alpha, beta){
  smoothed <- matrix(rep(0,1000), ncol=100, nrow=10)
  for (i in 1:dim(alpha)[2]){
    smoothed[, i] <- alpha[, i] * beta[, i] / sum(alpha[, i] * beta[, i])
  }
  return(smoothed)
}

# Finding the most probable state that the robot was in at time t with
# regards to the observations
state_func <- function(algo){
  state <- c()
  for (i in 1:dim(algo)[2]){
    # taking -1 to get the states in the form of 0-9
    state[i] <- which.max(algo[, i] )-1
  }
  return(state)
}

# Misclassification function
missclass <- function(true, pred){
  1 - sum(diag(table(true, pred)))/sum(table(true, pred))
}

# Function for printing the confusion matrices and misclassification rates
genHeatmap <- function (true_states, algo_state, name){
  conf_matr <- table(true_states, algo_state)
  prop.table(table(true_states, algo_state), margin=1)
  missclass(true_states, algo_state)
  conf_matr <- as.data.frame(conf_matr)
  names(conf_matr) <- c("TrueHiddenState","Prediction", "Freq")
  heatmap <- ggplot(conf_matr, aes(Prediction, TrueHiddenState, fill= Freq)) +
    geom_tile(aes(fill=Freq)) +
    geom_text(aes(label = round(Freq, 1))) +
    ggtitle(name) +
    theme(plot.title = element_text(size = 20, face = "bold"))+
    geom_abline(intercept = 0, slope = 1, color="white", size=0.1)
  return(heatmap)
}


simulation <- function(hmm){
  misclass_vec <- c()
  # 100 time steps simulated from our hidden markov model
  simulated <- simHMM(hmm, 100)
  # Our observations from our simulation, e.g our signal
  obs <- simulated$observation
  # Forward algo
```

```
  forward <- forward(hmm, obs)
  # Backward algo
  backward <- backward(hmm, obs)
  # Computing alpha as e^forward since forward is in log form
  alpha <- exp(forward)
  # Computing beta as e^backward since backward is in log form
  beta <- exp(backward)
  # calculating the probabilities the robot was in state x at a certain time point
  filtered <- filtered_algo(alpha)
  smoothed <- smoothed_algo(alpha, beta)
  # And most likely path
  viterbi <- viterbi(hmm, obs)
  # The predicted states
  filtered_state <- state_func(filtered)
  smoothed_state <- state_func(smoothed)
  # The true states
  true_states <- simulated$states

  # Misclassification rates
  misclass_vec[1] <- missclass(true_states, filtered_state)
  misclass_vec[2] <- missclass(true_states, smoothed_state)
  misclass_vec[3] <- missclass(true_states, viterbi)
  return(list("misclass_vec" = misclass_vec,
              "filtered_state" = filtered_state,
              "smoothed_state" = smoothed_state,
              "true_states" = true_states,
              "viterbi" =viterbi,
              "filtered"= filtered))
}

simulate <- simulation(hmm_model)
# What confusion matrices could look like for one simulation
filtered_state <- simulate$filtered_state
smoothed_state <- simulate$smoothed_state
viterbi <- simulate$viterbi
true_states <- simulate$true_states
print("filtered")
filter_confmatr <- genHeatmap(true_states, filtered_state, "Filtered Confusion Matrix")

print("smoothed")
smooth_confmatr <- genHeatmap(true_states, smoothed_state, "Smoothed Confusion Matrix")

print("viterbi")
viterbi_confmatr <- genHeatmap(true_states, viterbi, "Viterbi Confusion Matrix")

misclass_matrix <- matrix(0, nrow=100, ncol=3)
for (i in 1:dim(misclass_matrix)[1]){
  misclass_matrix[i,] <- simulation(hmm_model)$misclass_vec
}

# Plotting the misclassification distribution for the different algorithms
df <- data.frame(misclass_matrix)
```

```r
misclass_plot <- ggplot(data=df)+
  ggtitle("Missclassification Error") +
  geom_density(aes(x=X2, colour="Smoothed"), fill = "palegreen3", alpha=0.6 )+
  geom_density(aes(x=X3, color="Viterbi"), fill = "light blue", alpha=0.5)+
  geom_density(aes(x=X1, colour="Filtered"), fill="tomato", alpha=0.3)+

  labs(colour="Algorithm") +
  xlab('Misclass error') + ylab('Frequency') +
  xlim(0,1)

# Empirical Estimators of Entropy and Mutual Information and Related Quantities
library(entropy)

# Calculating the entropy at different time steps for 100 simulation and
# averaging the results
entropy_matrix <- matrix(0, ncol=100, nrow=100)
for (j in 1:100){
  entropy <- c()
  filtered <- simulation(hmm_model)$filtered
  for (i in 1:dim(filtered)[2]){

    entropy[i] <- entropy.empirical(filtered[, i], unit="log2")
  }
  entropy_matrix[,j] <- entropy
}

entropy <- c()
for (i in 1:100){
  entropy[i] <- sum(entropy_matrix[i,])/dim(entropy_matrix)[2]

}


entropy <- as.data.frame(entropy)
entropy$x <- seq(1,100,1)
entropy_plot <- ggplot() +
  geom_line(data=entropy, aes(y=entropy, x=x)) +
  xlab("Time step") +
  ggtitle("Entropy of the filtered distribution at different time points")

# Calculating the probability that robot is in a certain hidden state at time
# step 101 given the probabilities at time 100.
state_101 <- rep(0, 10)
for (i in 1:10){
  state_101[i] <- state_101[i] + 0.5 * filtered[i, 100]
  if (i==10){
    state_101[1] <- state_101[1] + 0.5 * filtered[i, 100]
  }else{
    state_101[i+1] <- state_101[i+1] + 0.5 * filtered[i, 100]

  }

}
```

```
state_101
filtered[,100]
```