TDDE15-Lab 2

oscho091

1: Build the hidden markov model

```r
# Load the HMM package
library(HMM)
library(entropy)
rm(list = ls())

# Define the hidden states and observation symbols
states <- 1:10
symbols <- 1:10

# Initialize the initial state probabilities: the robot is equally likely to start in any sector
start_probs <- rep(1/10, 10)  # A vector of 0.1's for each state

# Initialize the transition probability matrix with zeros
trans_probs <- matrix(0, nrow = 10, ncol = 10)

# Fill in the transition probabilities
for (i in 1:10) {
  # Stay in the current sector with probability 0.5
  trans_probs[i, i] <- 0.5

  # Move to the next sector with probability 0.5
  # Wrap around from sector 10 to sector 1
  next_sector <- ifelse(i == 10, 1, i + 1)
  trans_probs[i, next_sector] <- 0.5
}

# Initialize the emission probability matrix with zeros
emission_probs <- matrix(0, nrow = 10, ncol = 10)

# Fill in the emission probabilities
for (i in 1:10) {
  # Sectors [i-2, i-1, i, i+1, i+2] with wrap-around
  sectors <- ((i - 3):(i + 1)) %% 10 + 1  # Adjust for 1-based indexing
  # Assign equal probability to each possible observed sector 0.2
  emission_probs[i, sectors] <- 1/5
}

# Initialize the Hidden Markov Model
hmm_model <- initHMM(
  States = states,           # vector of states
  Symbols = symbols,         # vector of observation symbols
```

```
  startProbs = start_probs,  # Initial state probabilities
  transProbs = trans_probs,  # Transition probabilities matrix
  emissionProbs = emission_probs  # Emission probabilities matrix
)
```

2. Simulate 100 time steps

```
set.seed(12345)
simulation <- simHMM(hmm_model, length = 100)
```

3. Compute filtered, smooth probability distributions and most probable path

```
# Extract observations of the tracking device
observations <- simulation$observation

# Compute log forward probabilities using the forward algorithm
log_forward_probs <- forward(hmm_model, observations)

# Compute log backward probabilities using the backward algorithm
log_backward_probs <- backward(hmm_model, observations)

# Compute log smoothed probabilities loga + logb = log(a*b)
log_smoothed_probs <- log_forward_probs + log_backward_probs


# Convert log probabilities to normal scale
filtered_probs <- exp(log_forward_probs)
smoothed_probs <- exp(log_smoothed_probs)

# Compute the most probable path using the Viterbi algorithm
viterbi_path <- viterbi(hmm_model, observations)
```

4. Compute Accuracies

```
# normalize
filtered_probs <- apply(filtered_probs, 2, prop.table)
smoothed_probs <- apply(smoothed_probs, 2, prop.table)

# Find the most probable state at each time step
pred_filtered <- apply(filtered_probs, 2, which.max)
pred_smoothed <- apply(smoothed_probs, 2, which.max)

# Calculate accuracies
calc_accuracy = function(predicted_values, true_states) {
  accuracy <- mean(predicted_values == true_states) * 100
  return(accuracy)
}
```

```r
# Output the accuracies
cat("\nAccuracy of the Filtered Probabilities:", calc_accuracy(pred_filtered, simulation$states), "%\n")
```

```
##
## Accuracy of the Filtered Probabilities: 53 %
```

```r
cat("Accuracy of the Smoothed Probabilities:", calc_accuracy(pred_smoothed, simulation$states), "%\n")
```

```
## Accuracy of the Smoothed Probabilities: 74 %
```

```r
cat("Accuracy of the Viterbi Path:", calc_accuracy(viterbi_path, simulation$states), "%\n")
```

```
## Accuracy of the Viterbi Path: 56 %
```

5. Repeat with different simulated samples

```r
n_simulations <- 10

# Initialize vectors to store accuracies
accuracy_filtered <- numeric(n_simulations)
accuracy_smoothed <- numeric(n_simulations)
accuracy_viterbi <- numeric(n_simulations)

# Run the simulation multiple times
for (s in 1:n_simulations) {

  sim <- simHMM(hmm_model, length = 100)
  observations <- sim$observation

  log_forward_probs <- forward(hmm_model, observations)
  log_backward_probs <- backward(hmm_model, observations)

  log_smoothed_probs <- log_forward_probs + log_backward_probs

  filtered_probs <- exp(log_forward_probs)
  smoothed_probs <- exp(log_smoothed_probs)

  filtered_probs <- apply(filtered_probs, 2, prop.table)
  smoothed_probs <- apply(smoothed_probs, 2, prop.table)

  viterbi_path <- viterbi(hmm_model, observations)

  pred_filtered <- apply(filtered_probs, 2, which.max)
  pred_smoothed <- apply(smoothed_probs, 2, which.max)

  # Calculate and store accuracies
  accuracy_filtered[s] <- calc_accuracy(pred_filtered, sim$states)
  accuracy_smoothed[s] <- calc_accuracy(pred_smoothed, sim$states)
  accuracy_viterbi[s] <- calc_accuracy(viterbi_path, sim$states)
}
```

```r
# Output the results
cat("Filtered Probabilities:", mean(accuracy_filtered), "%\n")
```

```
## Filtered Probabilities: 51.3 %
```

```r
cat("Smoothed Probabilities:", mean(accuracy_smoothed), "%\n")
```

```
## Smoothed Probabilities: 69.5 %
```

```r
cat("Viterbi Path:", mean(accuracy_viterbi), "%\n")
```

```
## Viterbi Path: 53.6 %
```

```r
# Display accuracies for each simulation
results_df <- data.frame(
  Simulation = 1:n_simulations,
  Filtered = accuracy_filtered,
  Smoothed = accuracy_smoothed,
  Viterbi = accuracy_viterbi
)
print(results_df)
```

```
##    Simulation Filtered Smoothed Viterbi
## 1           1       46       68      61
## 2           2       49       77      65
## 3           3       49       58      56
## 4           4       60       80      65
## 5           5       54       64      39
## 6           6       54       69      53
## 7           7       52       67      55
## 8           8       52       67      50
## 9           9       45       68      45
## 10         10       52       77      47
```

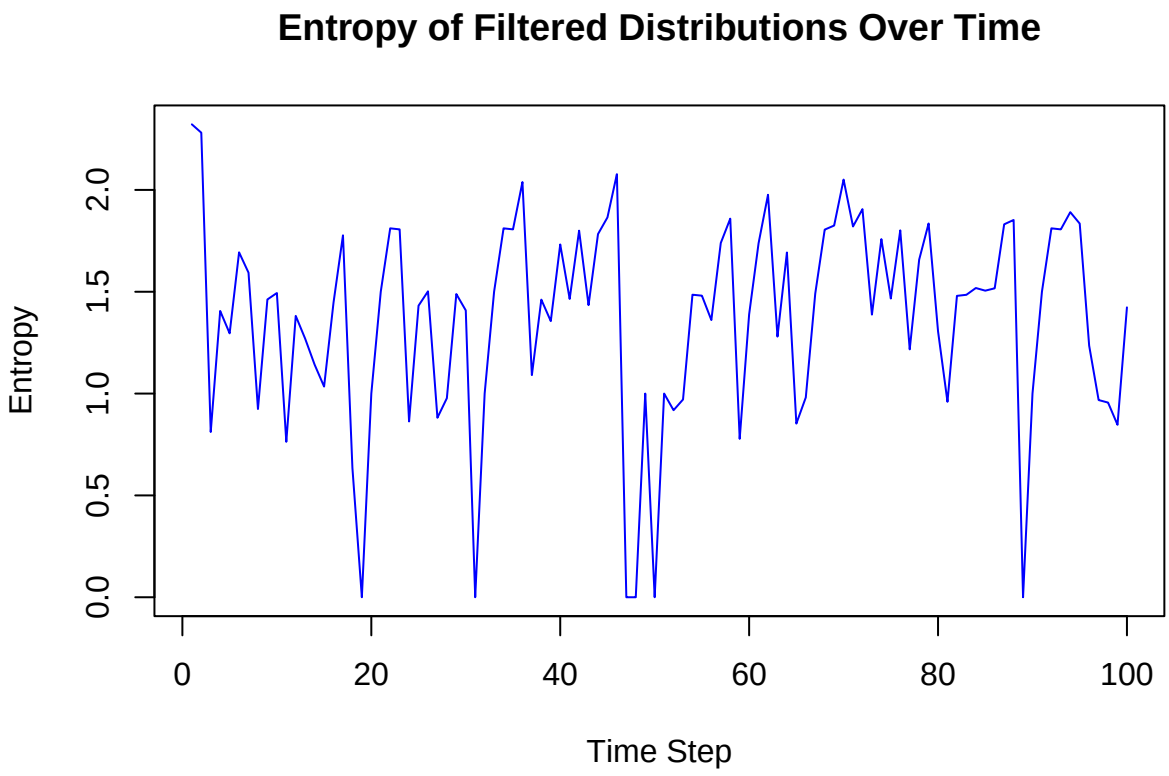
Smoothed distributions are generally more accurate because they leverage more information (both past and future observations). This is supported by the average accuracies attained in this task. The Viterbi path is less accurate than smoothed distributions because it commits to a single sequence of states, which don't account for the uncertainty in the observations.

6. Entropy

```r
entropy_filtered <- numeric(ncol(filtered_probs))  # Initialize vector to store entropy

# Calculate the entropy for each time step
for (t in 1:ncol(filtered_probs)) {
  # Calculate the empirical entropy for each time step's probability distribution
  entropy_filtered[t] <- entropy.empirical(filtered_probs[, t], unit = "log2")
}

# Plot entropy over time
plot(1:length(entropy_filtered), entropy_filtered, type = "l", col = "blue", xlab = "Time Step", ylab =
```



As seen in the graph the entropy seems to be fluctuating with no clear trend which suggest that later in time with more observations doesn't mean better predictions. Likely because the observations i-2:i+2 are too "nosy".

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

```
filtered_prob_t100 <- filtered_probs[, 100]  # Vector of probabilities for each state at t=100

# Compute the probabilities for time step 101 by multiplying with the transition matrix
prob_t101 <- trans_probs %*% filtered_prob_t100
print(prob_t101)
```

```
##             [,1]
##  [1,] 0.31842105
##  [2,] 0.06842105
##  [3,] 0.00000000
##  [4,] 0.00000000
##  [5,] 0.00000000
##  [6,] 0.00000000
##  [7,] 0.00000000
##  [8,] 0.00000000
##  [9,] 0.18157895
## [10,] 0.43157895
```