732A96: Advanced Machine Learning

Computer Lab 2: Hidden Markov Models

Hariprasath Govindarajan (hargo729)

September 25, 2019

Contents

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.

Question 1: HMM Initialization

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

Answer:

The Hidden Markov Model is initialized using the following parameters.

To build a HMM using the `initHMM()` function we need to provide the following parameters:

- `States` - Vector with the names of the states.
- `Symbols` - Vector with the names of the symbols.
- `startProbs` - Annotated vector with the starting probabilities of the states.
- `transProbs` - Annotated matrix containing the transition probabilities between the states.
- `emissionProbs` - Annotated matrix containing the emission probabilities of the states.'

We are given that the robot moves in 10 sectors of a ring. So, we have 10 states and the tracking device emissions also correspond to these 10 sectors. Initially, the robot could start in any of the sectors with equal probability. So, the start probabilities are equal for all the states. The robot can either move to the next sector or stay in the same sector with equal probability. So, the transition probabilities corresponding to the transition $i \to i$ and the transition $i \to i+1$ is set to 0.5. The tracking device emissions are inaccurate and emit with equal probability the 5 values $[i-2, i+2]$ when the robot is actually in state i. So, for each state i, the probabilities for the emissions $[i-2, i+2]$ are set to 0.2. We initialize the HMM with these parameters.

```
num_states = 10
num_symbols = 10

# Function to generate next number in a cyclic manner
cnext = function(i, d, N){
  nxt = (i+d)%%N
  ifelse(nxt==0, N, nxt)
}

# Starting probabilities
start_probs = rep(1/num_states, num_states)

# Transition probabilities matrix:
# Robot can stay in current sector or move to
# next sector with equal probability
trans_probs = matrix(0, nrow = num_states, ncol = num_states)
num_possible_trans = 2
p_t = 1/num_possible_trans # Equal probabilities, p = 0.5

for(i in 1:num_states){
  trans_probs[i, i] = p_t
  trans_probs[i, cnext(i, 1, 10)] = p_t
}

# Emission probabilities matrix:
# Tracking device can report values in the range [i-2, i+2]
```

```r
# when the robot is in sector-i
emission_probs = matrix(0, nrow = num_states, ncol = num_symbols)
num_possible_emissions = 5
p_e = 1/num_possible_emissions # Equal probabilities, p = 0.2

for(i in 1:num_states){
  emission_probs[i, cnext(i, -2, 10)] = p_e
  emission_probs[i, cnext(i, -1, 10)] = p_e
  emission_probs[i, cnext(i, 0, 10)] = p_e
  emission_probs[i, cnext(i, 1, 10)] = p_e
  emission_probs[i, cnext(i, 2, 10)] = p_e
}

# Initialize HMM
hmm = initHMM(States = 1:10, Symbols = 1:10, start_probs,
              trans_probs, emission_probs)
hmm
```

```
## $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: HMM Simulation

Question: Simulate the HMM for 100 time steps.

Answer:

Now, we simulate from the HMM built in the previous step for 100 time steps. The first 10 simulations are shown below:

```r
set.seed(12345)
hmm_sim = simHMM(hmm, length = 100)

cat("First 10 simulations: \nStates: ", hmm_sim$states[1:10],
    "\nObservations: ", hmm_sim$observation[1:10])
```

```
## First 10 simulations:
## States:  9 9 9 9 10 1 2 2 2 2
## Observations:  7 10 8 10 2 3 10 3 4 4
```

Question 3: Filtered and Smoothed distributions & Most probable path

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:

The function `hmm_analyze()` performs the specified type of analysis (Filtered, Smoothed or Viterbi) on the given HMM and the observations to compute the probability distributions (Filtered, Smoothed) and the most probable path (Viterbi). It also compares the obtained path or distribution with the true states to compute the confusion matrix and accuracy.

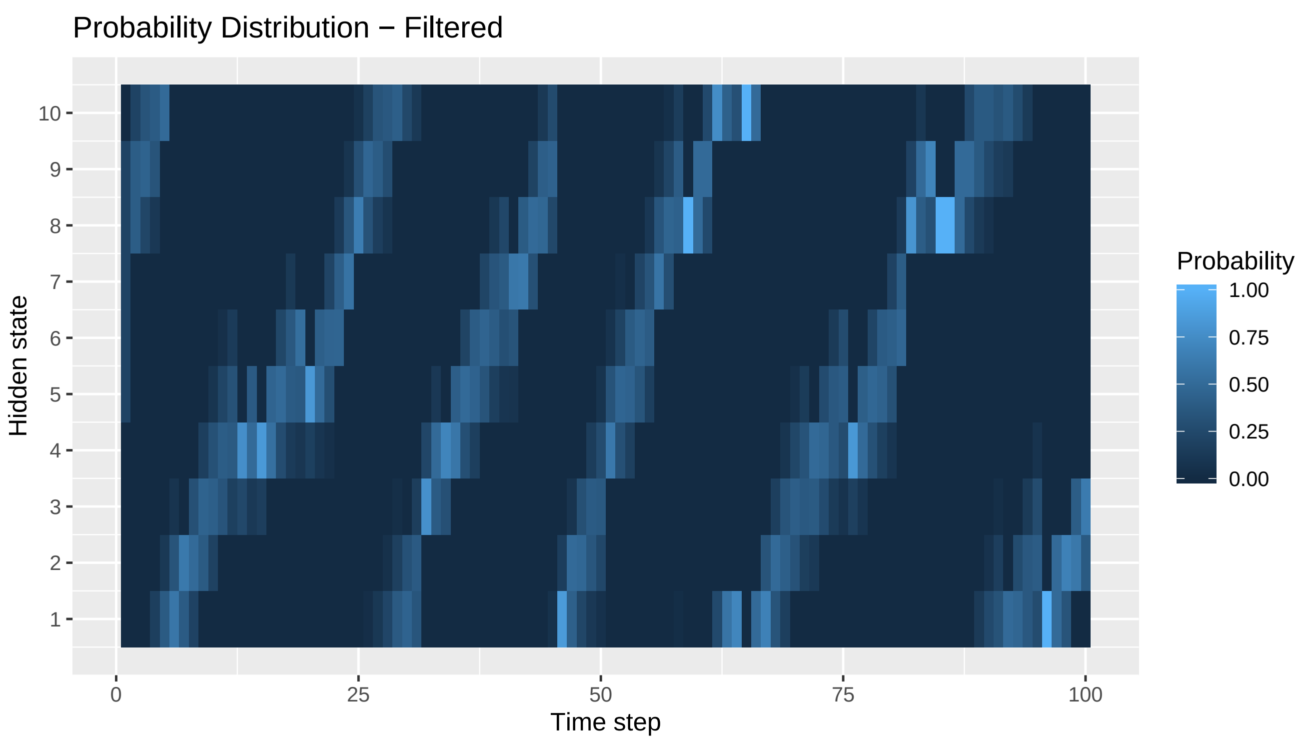
```r
hmm_analyze = function(hmm, observations, true_states, method = "Smoothed"){
  probs = c()
  most_prob_path = c()
  N = length(observations)

  if(method == "Filtered"){
    # Compute log probabilities
    log_fwd_probs =  forward(hmm, observations)

    # Convert to probabilities
    probs = prop.table(exp(log_fwd_probs), 2)
    # sapply(1:N, function(i) exp(log_fwd_probs[,i])/sum(exp(log_fwd_probs[,i])))

    # Compute most probable path from probabilities
    most_prob_path = sapply(1:N, function(i) which.max(log_fwd_probs[,i])[1])
  }
  else if(method == "Viterbi"){
    # Compute most probable path by Viterbi method
    most_prob_path = viterbi(hmm, observations)
  }
  else{
    # Note: posterior() returns probabilities directly
    probs = posterior(hmm, observations)

    # Compute most probable path from probabilities
    most_prob_path = sapply(1:N, function(i) which.max(probs[,i])[1])
  }

  names(most_prob_path) = c()

  # Compute confusion matrix
  conf_mat = table(most_prob_path, true_states)

  # Compute accuracy
  correct_count = sum(diag(conf_mat))
  accuracy = correct_count/N

  result = list(probs = probs, most_prob_path = most_prob_path,
                conf_mat = conf_mat, accuracy = accuracy)
```

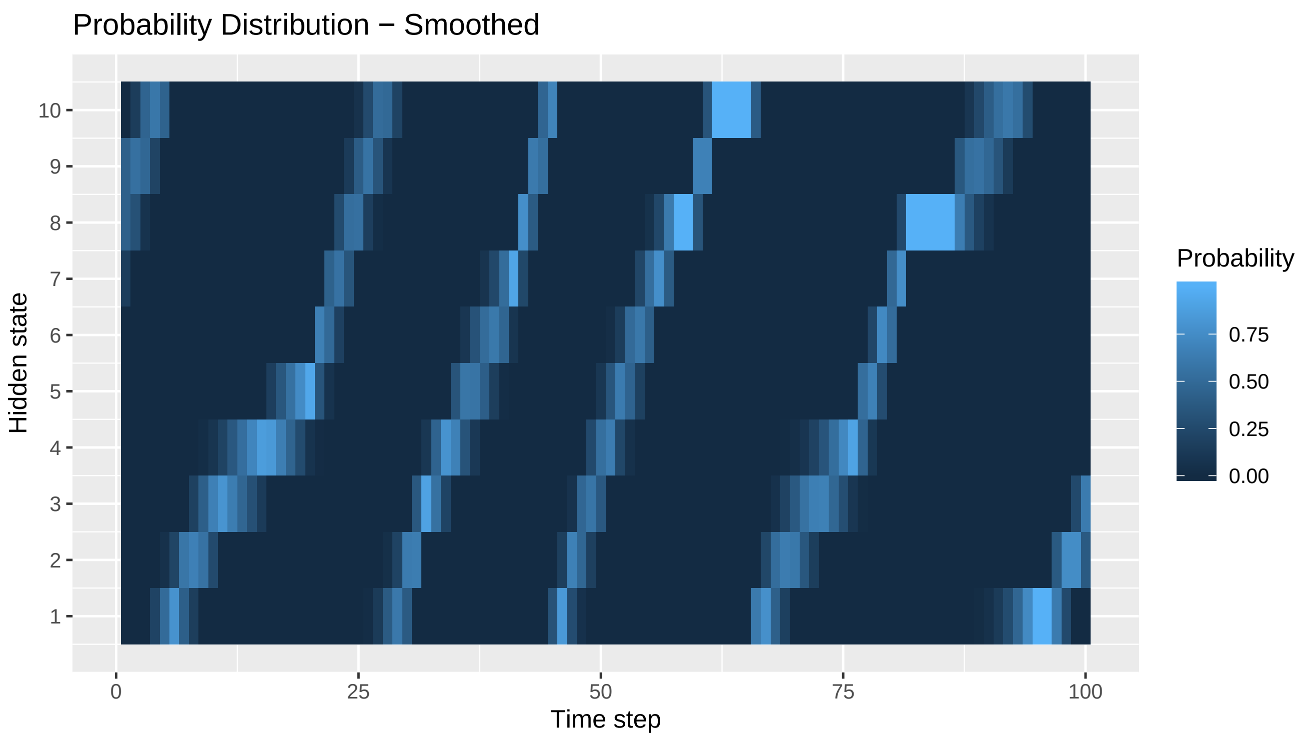
```
  return(result)
}
```

We compute the filtered and smoothed probability distributions and the most probable path using only the observations from our simulation.

```
hmm_obs = hmm_sim$observation
hmm_true_states = hmm_sim$states

# Analyze HMM using filtered, smoothed and Viterbi methods
hmm_post_res = hmm_analyze(hmm, hmm_obs, hmm_true_states, method = "Smoothed")
hmm_fwd_res = hmm_analyze(hmm, hmm_obs, hmm_true_states, method = "Filtered")
hmm_viterbi_res = hmm_analyze(hmm, hmm_obs, hmm_true_states, method = "Viterbi")
```

The probability distributions obtained using the filtered and smoothed methods are as follows:





For the filtered and smoothed distributions, we calculate a path based on the state with the highest probability at each time step. The paths obtained from the 3 methods are as follows:

```
## [1] "Path (Posterior): "

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

## [1] "Path (Forward): "

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

## [1] "Most Probable Path (Viterbi): "

##   [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
```

Question 4: Accuracy comparison of methods

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:

We have already calculated the paths using the 3 methods in the previous question. The confusion matrices and the accuracies for these methods are shown below.

Confusion Matrix (Smoothed):

```
##               true_states
## most_prob_path  1  2  3  4  5  6  7  8  9 10
##             1   9  0  0  0  0  0  0  0  0  1
##             2   1 10  1  0  0  0  0  0  0  0
##             3   0  3  7  0  0  0  0  0  0  0
##             4   0  0  2  9  1  0  0  0  0  0
##             5   0  0  0  3  5  0  0  0  0  0
##             6   0  0  0  0  3  5  0  0  0  0
##             7   0  0  0  0  0  1  5  0  0  0
##             8   0  0  0  0  0  0  1  9  3  0
##             9   0  0  0  0  0  0  0  2  6  2
##             10  1  0  0  0  0  0  0  0  1  9
```

Confusion Matrix (Filtered):

```
##               true_states
## most_prob_path 1 2 3 4 5 6 7 8 9 10
##             1  8 1 0 0 0 0 0 0 0  4
##             2  2 7 1 0 0 0 0 0 0  0
##             3  0 4 3 1 0 0 0 0 0  0
##             4  0 1 5 7 2 0 0 0 0  0
##             5  0 0 1 3 5 2 0 0 1  0
##             6  0 0 0 1 2 3 3 0 0  0
##             7  0 0 0 0 0 1 3 1 0  0
```

```
##              8  0 0 0 0 0 0 0 8 4  0
##              9  0 0 0 0 0 0 0 2 4  3
##              10 1 0 0 0 0 0 0 0 1  5
```

Confusion Matrix (Viterbi):

```
##                true_states
## most_prob_path  1  2  3  4  5  6  7  8  9 10
##              1 11  6  1  0  0  0  0  0  2  8
##              2  0  7  2  0  0  0  0  0  0  0
##              3  0  0  7  6  1  0  0  0  0  0
##              4  0  0  0  6  4  1  0  0  0  0
##              5  0  0  0  0  4  1  0  0  0  0
##              6  0  0  0  0  0  3  1  0  0  0
##              7  0  0  0  0  0  1  3  1  0  0
##              8  0  0  0  0  0  0  2  8  1  0
##              9  0  0  0  0  0  0  0  2  3  0
##              10 0  0  0  0  0  0  0  0  4  4
```

Accuracies of methods:

Method	Accuracy
Smoothed	0.74
Filtered	0.53
Viterbi	0.56

Question 5: Comparison of methods

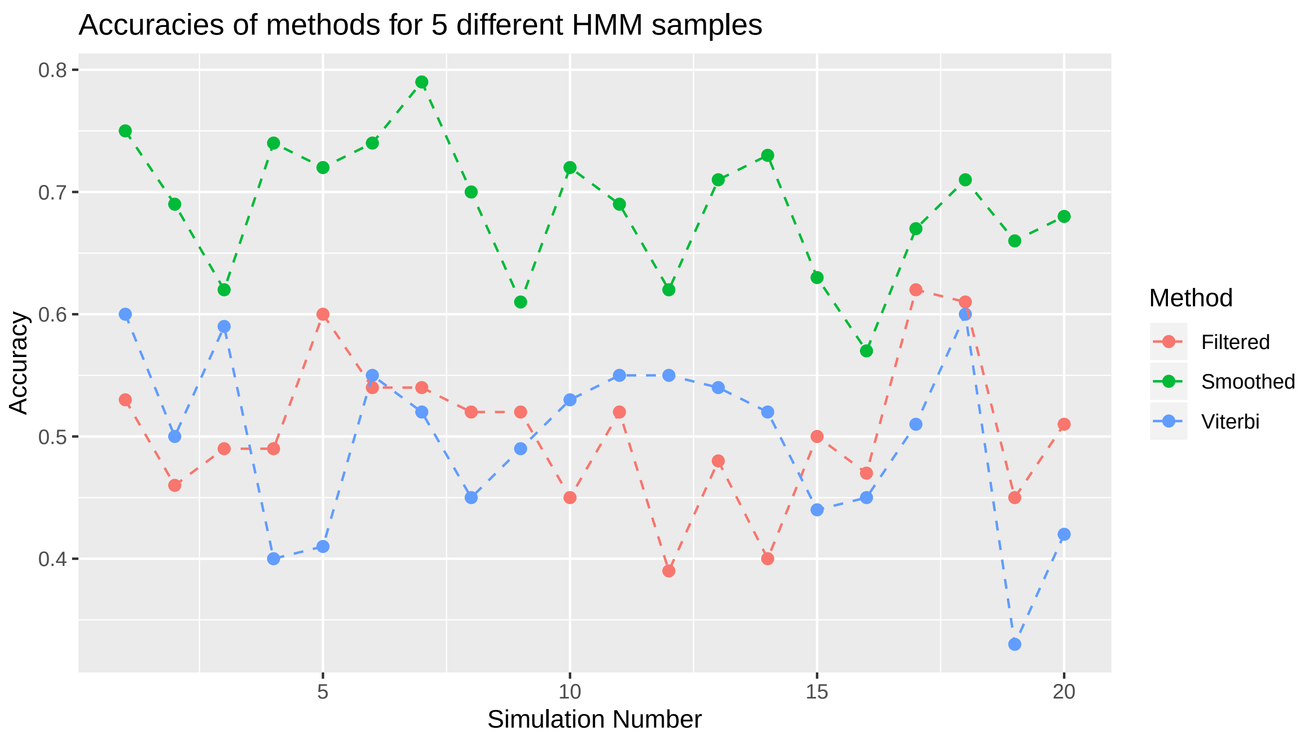
Question: Repeat the previous exercise with different simulated samples. In general, the smoothed distributions should be more accurate than the filtered distributions. Why? In general, the smoothed distributions should be more accurate than the most probable paths, too. Why?

Answer:

We repeat the previous exercise by generating 20 different samples of size 100. We compute the accuracies of each method.

```r
# Function to simulate from HMM and compute the path
# and accuracy using specified method
compare_methods = function(hmm, params, same_seed = F){
  # Get all parameter combinations
  params_df = expand.grid(params)
  result_df = data.frame(NumSim = c(), Method = c(), Accuracy = c(), Entropy = c())

  process_method = function(n_sim, method){
    if(same_seed){
      set.seed(12345)
    }

    hmm_sim = simHMM(hmm, length = n_sim)
    hmm_res = hmm_analyze(hmm, hmm_sim$observation, hmm_sim$states, method = method)
    entropy = entropy.empirical(hmm_res$probs[ , n_sim])

    result_df <<- rbind(result_df, data.frame(NumSim = n_sim, Method = method,
                                               Accuracy = hmm_res$accuracy,
                                               Entropy = entropy))
  }

  # Append row of details to data frame
  sapply(1:nrow(params_df), function(i) process_method(params_df[i,"NumSim"],
                                                       params_df[i,"Method"]))

  return(result_df)
}

params = list(NumSim = rep(100, 20),
              Method = c("Filtered", "Smoothed", "Viterbi"))

set.seed(12345)
hmm_compare = compare_methods(hmm, params)
```

The following plot shows the accuracies of each method for each simulation.



We see that the smoothed distributions are more accurate than the filtered distributions and the most probable paths. The smoothed distribution at each time step takes into account the observations that occurred both before and after that time step. On the other hand, the filtered distribution at each time step takes into account only those observations which occurred before that time step. The observations that occurred both before and after a time step can help in predicting the current state. So, the smoothed distributions should be more accurate as it takes into account more data into account compared to the filtered distributions. The Viterbi algorithm is a dynamic programming algorithm which computes the most probable path based on only the past observations at each time step. Also, it focusses on how likely a path is and not necessarily the likeliness of a state at a time step. So, we would expect it to be less accurate than the smoothed probability distributions.

Question 6: Relationship of certainty of robot's position to number of observations

Question: Is it true that the more observations you have the better you know where the robot is?

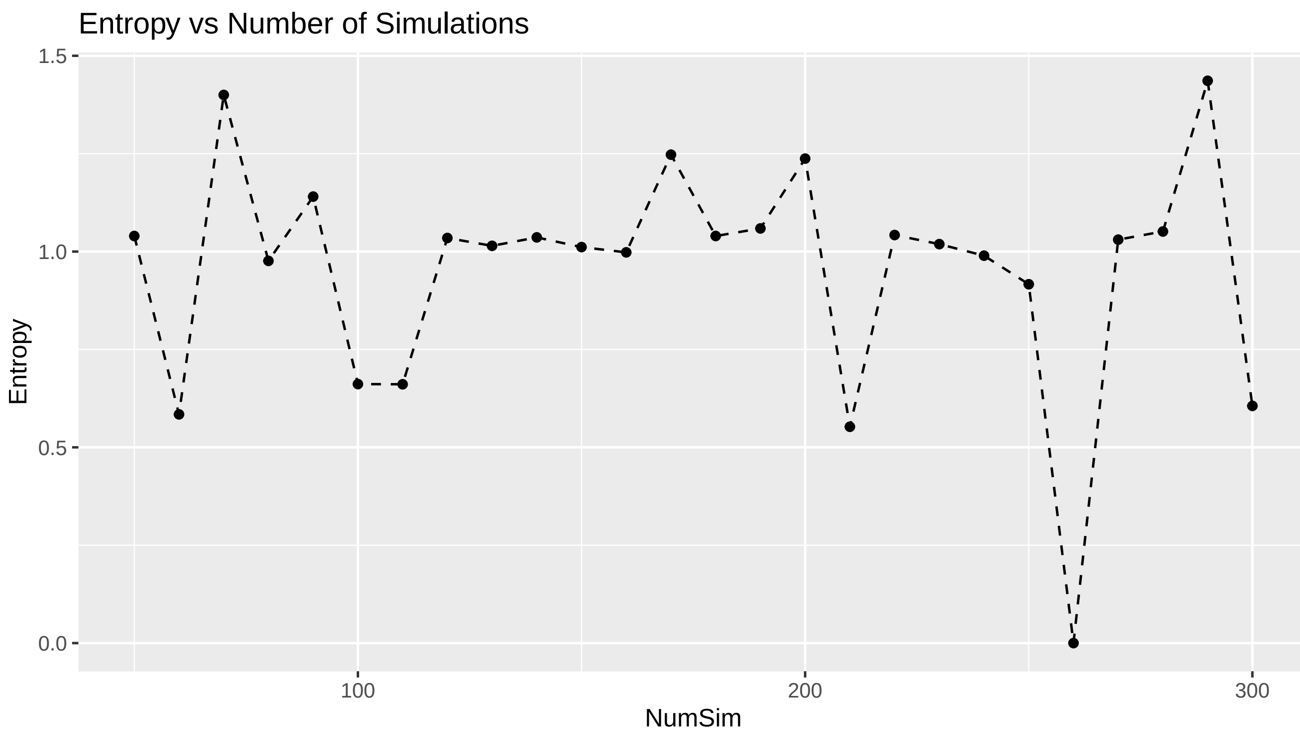
Hint: You may want to compute the entropy of the filtered distributions with the function `entropy.empirical` of the package `entropy`.

Answer:

To know better about where the robot is, we need to have a higher certainty in our prediction. So, we want the probability distribution for the state to be less spread out and more focussed on a specific state. This can be measured using entropy. Higher entropy indicates that the probabilities are well spread out. We, consider HMM simulations using different sample sizes in the range of 50 to 300 and compute their filtered probability distributions. Then, we compute the entropies for the last time step.

```
params = list(NumSim = seq(50, 300, 10),
              Method = c("Filtered"))

hmm_compare_size = compare_methods(hmm, params, same_seed = T)
```

The following plot shows the variation of entropy with number of simulations. We do not observe a consistent increasing or decreasing trend in entropy. It appears to be oscillating within a range of values. So, we can't really say that we know the position of the robot better after more number of observations.



Question 7: Prediction of state for time step 101

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

Answer:

```
# Function to predict next state for given HMM,
# probability distribution result and observations
hmm_predict = function(hmm, hmm_res, observations){
  trans_probs = hmm$transProbs
  N = length(observations)

  # Compute probabilities for next time step
  pred_dist = t(hmm_res$probs[,N, drop=F]) %*% trans_probs

  return(pred_dist)
}

pred_101 = c(hmm_predict(hmm, hmm_post_res, hmm_obs))
names(pred_101) = 1:10
```

The HMM had the following states so far:

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

The prediction probabilities for the states in time step 101 are:

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

Based on the highest probability, we would predict that the robot is in sector = 3.

Appendix

```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE, fig.width=8)
```

```r
set.seed(123456)

library(ggplot2)
options(kableExtra.latex.load_packages = FALSE)
library(kableExtra)
library(caret)
library(HMM)
library(entropy)
library(reshape2)

options(scipen=999)


# ------------------------------------------------------------------------------
# Question 1
# ------------------------------------------------------------------------------

num_states = 10
num_symbols = 10

# Function to generate next number in a cyclic manner
cnext = function(i, d, N){
  nxt = (i+d)%%N
  ifelse(nxt==0, N, nxt)
}

# Starting probabilities
start_probs = rep(1/num_states, num_states)

# Transition probabilities matrix:
# Robot can stay in current sector or move to
# next sector with equal probability
trans_probs = matrix(0, nrow = num_states, ncol = num_states)
num_possible_trans = 2
p_t = 1/num_possible_trans # Equal probabilities, p = 0.5

for(i in 1:num_states){
  trans_probs[i, i] = p_t
  trans_probs[i, cnext(i, 1, 10)] = p_t
}

# Emission probabilities matrix:
# Tracking device can report values in the range [i-2, i+2]
# when the robot is in sector-i
emission_probs = matrix(0, nrow = num_states, ncol = num_symbols)
num_possible_emissions = 5
```

```r
p_e = 1/num_possible_emissions # Equal probabilities, p = 0.2

for(i in 1:num_states){
  emission_probs[i, cnext(i, -2, 10)] = p_e
  emission_probs[i, cnext(i, -1, 10)] = p_e
  emission_probs[i, cnext(i, 0, 10)] = p_e
  emission_probs[i, cnext(i, 1, 10)] = p_e
  emission_probs[i, cnext(i, 2, 10)] = p_e
}

# Initialize HMM
hmm = initHMM(States = 1:10, Symbols = 1:10, start_probs,
              trans_probs, emission_probs)
hmm


# ------------------------------------------------------------------------------
# Question 2
# ------------------------------------------------------------------------------

set.seed(12345)
hmm_sim = simHMM(hmm, length = 100)

cat("First 10 simulations: \nStates: ", hmm_sim$states[1:10],
    "\nObservations: ", hmm_sim$observation[1:10])

# ------------------------------------------------------------------------------
# Question 3
# ------------------------------------------------------------------------------

hmm_analyze = function(hmm, observations, true_states, method = "Smoothed"){
  probs = c()
  most_prob_path = c()
  N = length(observations)

  if(method == "Filtered"){
    # Compute log probabilities
    log_fwd_probs =  forward(hmm, observations)

    # Convert to probabilities
    probs = prop.table(exp(log_fwd_probs), 2)
    # sapply(1:N, function(i) exp(log_fwd_probs[,i])/sum(exp(log_fwd_probs[,i])))

    # Compute most probable path from probabilities
    most_prob_path = sapply(1:N, function(i) which.max(log_fwd_probs[,i])[1])
  }
```

```r
  else if(method == "Viterbi"){
    # Compute most probable path by Viterbi method
    most_prob_path = viterbi(hmm, observations)
  }
  else{
    # Note: posterior() returns probabilities directly
    probs = posterior(hmm, observations)

    # Compute most probable path from probabilities
    most_prob_path = sapply(1:N, function(i) which.max(probs[,i])[1])
  }

  names(most_prob_path) = c()

  # Compute confusion matrix
  conf_mat = table(most_prob_path, true_states)

  # Compute accuracy
  correct_count = sum(diag(conf_mat))
  accuracy = correct_count/N

  result = list(probs = probs, most_prob_path = most_prob_path,
                conf_mat = conf_mat, accuracy = accuracy)

  return(result)
}

hmm_obs = hmm_sim$observation
hmm_true_states = hmm_sim$states

# Analyze HMM using filtered, smoothed and Viterbi methods
hmm_post_res = hmm_analyze(hmm, hmm_obs, hmm_true_states, method = "Smoothed")
hmm_fwd_res = hmm_analyze(hmm, hmm_obs, hmm_true_states, method = "Filtered")
hmm_viterbi_res = hmm_analyze(hmm, hmm_obs, hmm_true_states, method = "Viterbi")

ggplot(data = melt(hmm_fwd_res$probs)) +
  geom_raster(aes(x=index, y=states, fill=value)) +
  scale_y_continuous(breaks = 1:10) +
  labs(x = "Time step", y = "Hidden state", fill = "Probability") +
  ggtitle("Probability Distribution - Filtered")

ggplot(data = melt(hmm_post_res$probs)) +
  geom_raster(aes(x=index, y=states, fill=value)) +
  scale_y_continuous(breaks = 1:10) +
  labs(x = "Time step", y = "Hidden state", fill = "Probability") +
  ggtitle("Probability Distribution - Smoothed")
```

```r
print("Path (Posterior): ")
print(hmm_post_res$most_prob_path)

print("Path (Forward): ")
print(hmm_fwd_res$most_prob_path)

print("Most Probable Path (Viterbi): ")
print(hmm_viterbi_res$most_prob_path)


# ------------------------------------------------------------------------------
# Question 4
# ------------------------------------------------------------------------------

print(hmm_post_res$conf_mat)
print(hmm_fwd_res$conf_mat)
print(hmm_viterbi_res$conf_mat)

# Accuracy table
accuracy_df = data.frame(Method = c("Smoothed", "Filtered", "Viterbi"),
                         Accuracy = c(hmm_post_res$accuracy,
                                      hmm_fwd_res$accuracy,
                                      hmm_viterbi_res$accuracy))

kable(accuracy_df) %>%
  kable_styling(latex_option = "striped") %>%
  row_spec(0, bold = TRUE)


# ------------------------------------------------------------------------------
# Question 5
# ------------------------------------------------------------------------------

# Function to simulate from HMM and compute the path
# and accuracy using specified method
compare_methods = function(hmm, params, same_seed = F){
  # Get all parameter combinations
  params_df = expand.grid(params)
  result_df = data.frame(NumSim = c(), Method = c(), Accuracy = c(), Entropy = c())

  process_method = function(n_sim, method){
    if(same_seed){
      set.seed(12345)
    }

    hmm_sim = simHMM(hmm, length = n_sim)
```

```r
    hmm_res = hmm_analyze(hmm, hmm_sim$observation, hmm_sim$states, method = method)
    entropy = entropy.empirical(hmm_res$probs[ , n_sim])

    result_df <<- rbind(result_df, data.frame(NumSim = n_sim, Method = method,
                                              Accuracy = hmm_res$accuracy,
                                              Entropy = entropy))
  }

  # Append row of details to data frame
  sapply(1:nrow(params_df), function(i) process_method(params_df[i,"NumSim"],
                                                       params_df[i,"Method"]))

  return(result_df)
}

params = list(NumSim = rep(100, 20),
              Method = c("Filtered", "Smoothed", "Viterbi"))

set.seed(12345)
hmm_compare = compare_methods(hmm, params)

ggplot(data = hmm_compare) +
  geom_point(aes(x = rep(1:20, 3) , y = Accuracy, color = Method), size = 2) +
  geom_line(aes(x = rep(1:20, 3) , y = Accuracy, color = Method),
            linetype = "dashed") +
  labs(x = "Simulation Number", y = "Accuracy") +
  ggtitle("Accuracies of methods for 5 different HMM samples")


# ------------------------------------------------------------------------------
# Question 6
# ------------------------------------------------------------------------------

params = list(NumSim = seq(50, 300, 10),
              Method = c("Filtered"))

hmm_compare_size = compare_methods(hmm, params, same_seed = T)

ggplot(data = hmm_compare_size) + geom_point(aes(x=NumSim, y=Entropy)) +
  geom_line(aes(x=NumSim, y=Entropy), linetype = "dashed") +
  ggtitle("Entropy vs Number of Simulations")


# ------------------------------------------------------------------------------
# Question 7
# ------------------------------------------------------------------------------
```

```r
# Function to predict next state for given HMM,
# probability distribution result and observations
hmm_predict = function(hmm, hmm_res, observations){
  trans_probs = hmm$transProbs
  N = length(observations)

  # Compute probabilities for next time step
  pred_dist = t(hmm_res$probs[,N, drop=F]) %*% trans_probs

  return(pred_dist)
}

pred_101 = c(hmm_predict(hmm, hmm_post_res, hmm_obs))
names(pred_101) = 1:10

print(hmm_true_states)

print(pred_101)
```