732A96 - Advanced Machine Learning - Lab 2 - Group Report

Qinyuan Qi(qinqi464)

2024-09-20

Contribution:

Qinyuan Qi did all the work in this report.

Exercises

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

Build a hidden Markov model (HMM) for the scenario described above. Note that the documentation for the initHMM function says that the emissionProbs matrix should be of size [number of states]x[number of states]. This is wrong. It should be of size [number of states]x[number of symbols]. The package works correctly, though. It is just the documentation that is wrong

Answer:

According to the statement of the question, we know that the robot will have 10 states (10 sectors that the robot is located), and the robot has 10 symbols (10 possible sectors that the robot is located).

According to HMM documentation, the transProbs is a (number of states)x(number of states) matrix, and emissionProbs is a (number of states) x (number of symbols) matrix.

Because the robot is in one of the sectors and decides with equal probability to stay in that sector or move to the next sector,so the value set in the transProbs matrix is 0.5.

Since the robot will stay in a [i-2, i+2] sector with equal probability which is 0.2, so the value we can set in the emissionProbs matrix is 0.2.

Using the above information, we can build the HMM model as follows:

```
############################  Code For Exercises 1 ##############################
# define states
states <- c(1:10)

# define symbols
symbols <- c(1:10)

# define startProbs
```

```
startProbs = rep(0.1, 10)

# define transProbs matrix
transProbs <- matrix(c(0.5, 0.5,   0,   0,   0,   0,   0,   0,   0,   0,
                       0, 0.5, 0.5,   0,   0,   0,   0,   0,   0,   0,
                       0,   0, 0.5, 0.5,   0,   0,   0,   0,   0,   0,
                       0,   0,   0, 0.5, 0.5,   0,   0,   0,   0,   0,
                       0,   0,   0,   0, 0.5, 0.5,   0,   0,   0,   0,
                       0,   0,   0,   0,   0, 0.5, 0.5,   0,   0,   0,
                       0,   0,   0,   0,   0,   0, 0.5, 0.5,   0,   0,
                       0,   0,   0,   0,   0,   0,   0, 0.5, 0.5,   0,
                       0,   0,   0,   0,   0,   0,   0,   0, 0.5, 0.5,
                     0.5,   0,   0,   0,   0,   0,   0,   0,   0, 0.5),
                     nrow = length(symbols), byrow = TRUE)


# define emissionProbs matrix
emissionProbs <- 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),
                        nrow = length(symbols), byrow = TRUE)

hmm <- initHMM(States = states, Symbols = symbols, startProbs = startProbs,
               transProbs = transProbs, emissionProbs = emissionProbs)
```

Question 2

Simulate the HMM for 100 time steps.

Answer:

Use the model we created in the prev question, we can simulate the HMM for 100 time steps as follows:

```
############################  Code For Exercises 2 ##############################
# set seed since simHMM() will generate random states and observations
set.seed(12345)

simulation_num = 100

hmm_simulation = simHMM(hmm=hmm, length=simulation_num)

hmm_simulation
```

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

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

Question 3

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:

Code as below.

```
############################  Code For Exercises 3 ##############################
# remove the hidden states from the simulation data we just created
states_simulation = hmm_simulation$states
observations_simulation = hmm_simulation$observation

# compute the filtered and smoothed probability distributions for each of the 100 time points
# according to slide 7 of Lecture 5, run the forward-backward algorithm

# filtered probability distributions
# according to document of HMM,the probability returned is in natural logarithm scale
# however, it will generate several -inf, so we apply exp() here
# alpha value
forward_result = exp(forward(hmm, observations_simulation))

# normalize the result according to the formula in slide 8 of Lecture 5
filtered_probability = prop.table(forward_result, margin = 2)

# smoothed probability distributions
# according to document of HMM,the probability returned is in natural logarithm scale
# however, it will generate several -inf, so we apply exp() here
# beta value
backward_result = exp(backward(hmm, observations_simulation))

# normalize the result according to the formula in slide 8 of Lecture 5
smoothed_probability = prop.table(forward_result * backward_result, margin = 2)

# we use viterbi to compute the most probable path
most_probable_path = viterbi(hmm, observations_simulation)
```

Question 4

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:

Code as below, and please check the output of the code for the accuracy of the three methods.

```
############################  Code For Exercises 4 ##############################
# define function to calculate the accuracy
calculate_accuracy = function(observed_states, true_states){
  confusion_matrix <- table(observed_states,true_states)
  accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
  return(accuracy)
}

# init predicted states
filtered_states  <- integer(simulation_num)
smoothed_states <- integer(simulation_num)

for(i in 1:simulation_num){
  # get the most probable states from ith step of the filtered probability distributions
  filtered_states[i] <- states[which.max(filtered_probability[,i])]
  # get the most probable states from ith step of  the filtered probability distributions
  smoothed_states[i] <- states[which.max(smoothed_probability[,i])]
}

# calculate the accuracy of 3 methods
filtered_accuracy <- calculate_accuracy(filtered_states, states_simulation)
smoothed_accuracy <- calculate_accuracy(smoothed_states, states_simulation)
most_probable_accuracy <- calculate_accuracy(most_probable_path, states_simulation)
```

```
## [1] "The accuracy of the filtered probability distributions is:  0.53"
```

```
## [1] "The accuracy of the smoothed probability distributions is:  0.74"
```

```
## [1] "The accuracy of the most probable path is:  0.56"
```

Question 5

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 copy the code from the previous sectors, change the seed of random, and set simulation_num to 200.

The following is the result.

```
## [1] "The accuracy of the new filtered probability distributions is:  0.575"
```

```
## [1] "The accuracy of the new smoothed probability distributions is:  0.68"
```

```
## [1] "The accuracy of the new most probable path is:  0.51"
```

By checking the accuracy of the three methods, the smoothed distribution in general is more accurate than the filtered distribution. The reason behind this is filtered distribution only use the past data, means t=1 to current time t to calculate the probability of the current state while smoothed distribution use the past data and future data, this can be seen from their formula, which means it have the benefit of knowing the entire

sequence of observations (past and future). The difference also can be seen from the formulas list below. $\alpha(z^t)$ is for past,while $\beta(z^t)$ is for future.

$$p(z^t|x^{0:T}) = \frac{\alpha(z^t)}{\sum_{z_t} \alpha(z^t)} \quad \text{(Filtering)}$$

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

The same thing applies to the most probable path, the smoothed distribution in general is more accurate than the most probable path.

But the reason why smoothed distribution is more accurate than the most probable path is different.

According to the Viterbi Algorithm on slide 11 of Lecture5, in the first loop of the algorithm, it only considers the most probable next value uses max() function, which means, that when the algorithm finishes, it can only get one path.

However, smoothed distribution will calculate the probability of all possible paths, and then choose the most probable path, it means that it can get a global optimal path.

We also find that if we set simulation_num to a rather big number, say,500, because there is a multiplication of forward_result_new * backward_result_new, it will generate an underflow error.

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 ?

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

Answer:

We create a plot based on entropy Values to show the history data's influence on the current value.

Because the entropy of a distribution is a measure of the uncertainty of the distribution.

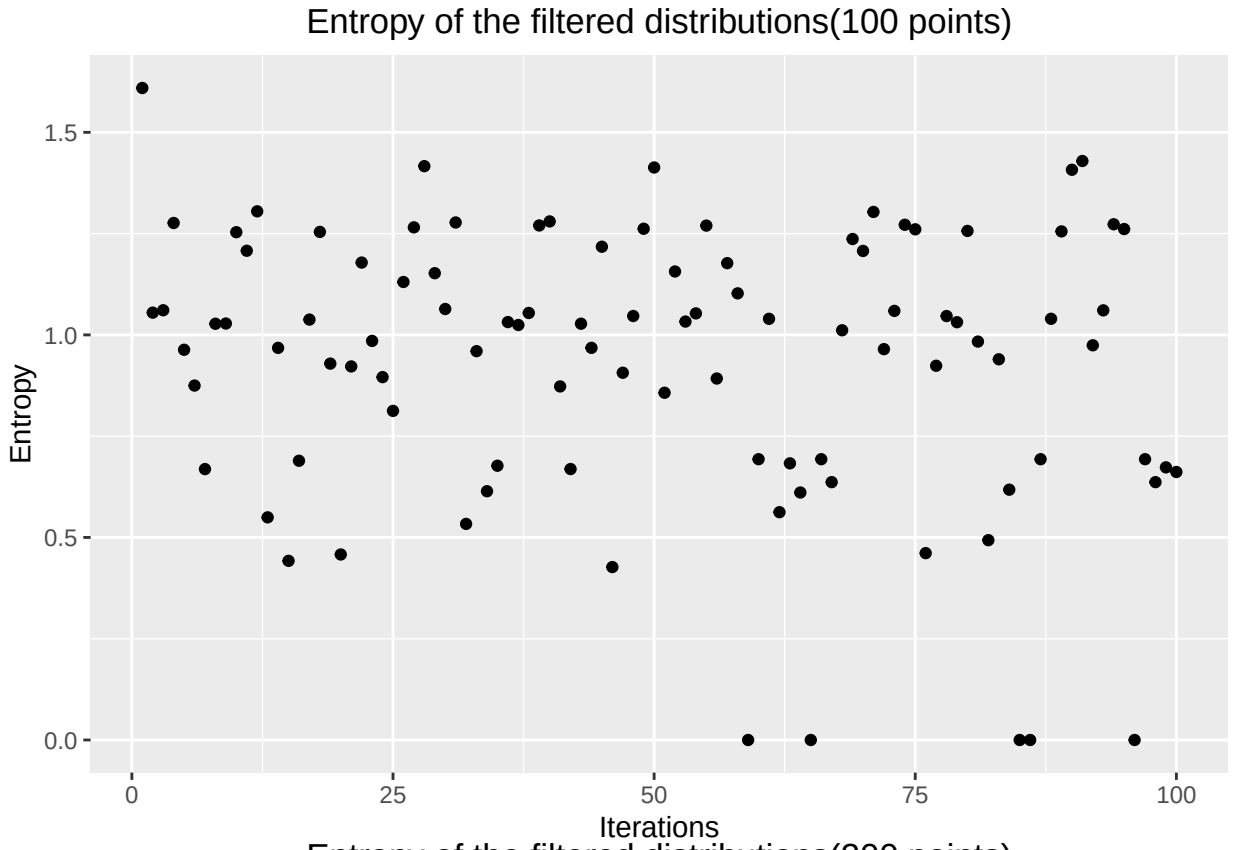
If entropy is very close to zero, means that it's highly predictable, but in our case,

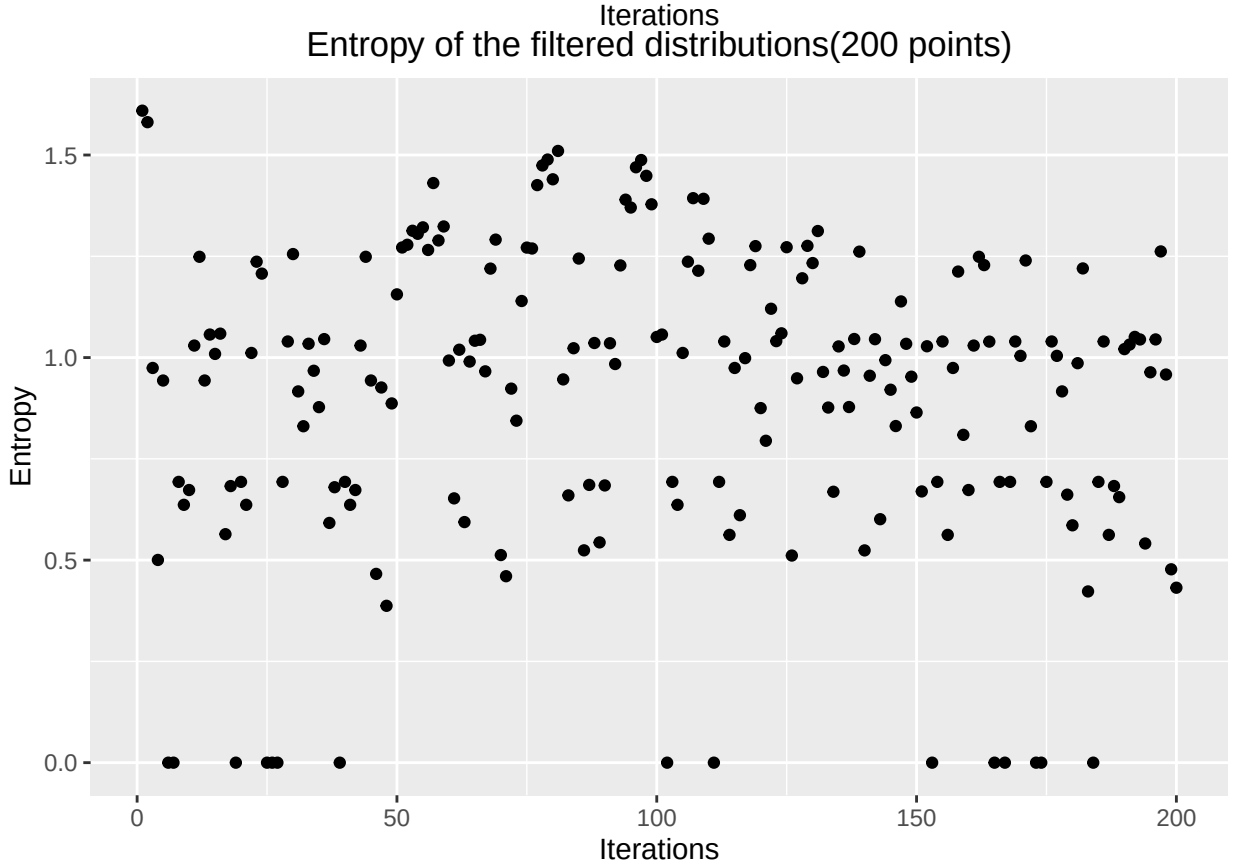
we find that the most of the scatter points are randomly distributed on [0.5-1.5], which means

it has relative big uncertainty.

And this can also be seen from the another different entropy plot below, which is generated from the new simulation data which contain 200 history points, and the entropy almost no difference from the 100 points data's case.

So we can say that in our case, history data has no influence on the current value.


Entropy of the filtered distributions(100 points)
Entropy
1.5
1.0
0.5
0.0
0
25
50
75
100
Iterations


Entropy of the filtered distributions(200 points)
Entropy
1.5
1.0
0.5
0.0
0
50
100
150
200
Iterations

Question 7

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

Answer:

We can use the filtered_probability dot product with the transProbs matrix to get the probability of the hidden states for the time step 101, result as follows.

```
## [1] "probability of the hidden states for the time step 101 is "
```

```
##      [,1]   [,2] [,3]   [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0 0.1875  0.5 0.3125    0    0    0    0    0     0
```

Appendix: All code for this report

```
############################  Init code For Assignment ############################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(HMM)
library(entropy)
library(ggplot2)
############################  Code For Exercises 1 ################################
# define states
states <- c(1:10)

# define symbols
symbols <- c(1:10)

# define startProbs
startProbs = rep(0.1, 10)

# define transProbs matrix
transProbs <- matrix(c(0.5, 0.5,   0,   0,   0,   0,   0,   0,   0,   0,
                       0, 0.5, 0.5,   0,   0,   0,   0,   0,   0,   0,
                       0,   0, 0.5, 0.5,   0,   0,   0,   0,   0,   0,
                       0,   0,   0, 0.5, 0.5,   0,   0,   0,   0,   0,
                       0,   0,   0,   0, 0.5, 0.5,   0,   0,   0,   0,
                       0,   0,   0,   0,   0, 0.5, 0.5,   0,   0,   0,
                       0,   0,   0,   0,   0,   0, 0.5, 0.5,   0,   0,
                       0,   0,   0,   0,   0,   0,   0, 0.5, 0.5,   0,
                       0,   0,   0,   0,   0,   0,   0,   0, 0.5, 0.5,
                     0.5,   0,   0,   0,   0,   0,   0,   0,   0, 0.5),
                     nrow = length(symbols), byrow = TRUE)


# define emissionProbs matrix
emissionProbs <- 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),
                        nrow = length(symbols), byrow = TRUE)

hmm <- initHMM(States = states, Symbols = symbols, startProbs = startProbs,
               transProbs = transProbs, emissionProbs = emissionProbs)
############################  Code For Exercises 2 ################################
# set seed since simHMM() will generate random states and observations
set.seed(12345)

simulation_num = 100

hmm_simulation = simHMM(hmm=hmm, length=simulation_num)
```

```r
hmm_simulation
############################  Code For Exercises 3 ##############################
# remove the hidden states from the simulation data we just created
states_simulation = hmm_simulation$states
observations_simulation = hmm_simulation$observation

# compute the filtered and smoothed probability distributions for each of the 100 time points
# according to slide 7 of Lecture 5, run the forward-backward algorithm

# filtered probability distributions
# according to document of HMM,the probability returned is in natural logarithm scale
# however, it will generate several -inf, so we apply exp() here
# alpha value
forward_result = exp(forward(hmm, observations_simulation))

# normalize the result according to the formula in slide 8 of Lecture 5
filtered_probability = prop.table(forward_result, margin = 2)

# smoothed probability distributions
# according to document of HMM,the probability returned is in natural logarithm scale
# however, it will generate several -inf, so we apply exp() here
# beta value
backward_result = exp(backward(hmm, observations_simulation))

# normalize the result according to the formula in slide 8 of Lecture 5
smoothed_probability = prop.table(forward_result * backward_result, margin = 2)

# we use viterbi to compute the most probable path
most_probable_path = viterbi(hmm, observations_simulation)
############################  Code For Exercises 4 ##############################
# define function to calculate the accuracy
calculate_accuracy = function(observed_states, true_states){
  confusion_matrix <- table(observed_states,true_states)
  accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
  return(accuracy)
}

# init predicted states
filtered_states  <- integer(simulation_num)
smoothed_states <- integer(simulation_num)

for(i in 1:simulation_num){
  # get the most probable states from ith step of the filtered probability distributions
  filtered_states[i] <- states[which.max(filtered_probability[,i])]
  # get the most probable states from ith step of  the filtered probability distributions
  smoothed_states[i] <- states[which.max(smoothed_probability[,i])]
}

# calculate the accuracy of 3 methods
filtered_accuracy <- calculate_accuracy(filtered_states, states_simulation)
smoothed_accuracy <- calculate_accuracy(smoothed_states, states_simulation)
most_probable_accuracy <- calculate_accuracy(most_probable_path, states_simulation)
# print it
```

```r
print(paste("The accuracy of the filtered probability distributions is: ", filtered_accuracy))
print(paste("The accuracy of the smoothed probability distributions is: ", smoothed_accuracy))
print(paste("The accuracy of the most probable path is: ", most_probable_accuracy))
############################  Code For Exercises 5 ##############################
set.seed(67890)

simulation_num = 200

hmm_simulation_new = simHMM(hmm=hmm, length=simulation_num)
states_simulation_new = hmm_simulation_new$states
observations_simulation_new = hmm_simulation_new$observation

forward_result_new = exp(forward(hmm, observations_simulation_new))
filtered_probability_new = prop.table(forward_result_new, margin = 2)

backward_result_new = exp(backward(hmm, observations_simulation_new))
smoothed_probability_new = prop.table(forward_result_new * backward_result_new, margin = 2)
most_probable_path_new = viterbi(hmm, observations_simulation_new)

# init predicted states
filtered_states_new  <- integer(simulation_num)
smoothed_states_new <- integer(simulation_num)

for(i in 1:simulation_num){
  filtered_states_new[i] <- states[which.max(filtered_probability_new[,i])]
  smoothed_states_new[i] <- states[which.max(smoothed_probability_new[,i])]
}

filtered_accuracy_new <- calculate_accuracy(filtered_states_new, states_simulation_new)
smoothed_accuracy_new <- calculate_accuracy(smoothed_states_new, states_simulation_new)
most_probable_accuracy_new <- calculate_accuracy(most_probable_path_new, states_simulation_new)

print(paste("The accuracy of the new filtered probability distributions is: ", filtered_accuracy_new))
print(paste("The accuracy of the new smoothed probability distributions is: ", smoothed_accuracy_new))
print(paste("The accuracy of the new most probable path is: ", most_probable_accuracy_new))
############################  Code For Exercises 6 ##############################
# we use entropy to check history data's influence on the prediction
entropyValues_1 <- apply(X=filtered_probability, MARGIN=2, FUN=entropy.empirical)
entropyValues_2 <- apply(X=filtered_probability_new, MARGIN=2, FUN=entropy.empirical)

# create a data frame for plotting
entropydf1 <- data.frame(x=1:100,entropy_1 = entropyValues_1)
entropydf2 <- data.frame(x=1:200,entropy_2 = entropyValues_2)

plot_6_1 <- ggplot(entropydf1) +
  geom_point(aes(x=x, y=entropy_1)) +
  labs(title = 'Entropy of the filtered distributions(100 points)',
       x = 'Iterations', y = 'Entropy') +
  theme(
    plot.title = element_text(hjust = 0.5)  # Center the title
  )

plot_6_2 <- ggplot(entropydf2) +
```

```
  geom_point(aes(x=x, y=entropy_2)) +
  labs(title = 'Entropy of the filtered distributions(200 points)',
       x = 'Iterations', y = 'Entropy') +
  theme(
    plot.title = element_text(hjust = 0.5)  # Center the title
  )

plot_6_1
plot_6_2
############################  Code For Exercises 7 ################################
# compute the probabilities of the hidden states for the time step 101
step101 <- filtered_probability[,100] %*% transProbs

print("probability of the hidden states for the time step 101 is ")
print(step101)
```