Lab 2

Arun Uppugunduri

2020-09-18

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.

Task 1

Build a hidden Markov model for the scenario described above

To create an HMM we use the initHMM function which takes in the arguments States = Vector with the different states for the model Symbols = Vector with the names of the symbols startProbs = vector with the starting probabilities of all teh states transProbs = Stochastic matrix holding the transition probabilities between states Z emissionProbs = Stochastic matrix holding the emission probabilities of the states X

stochastic matrix => Each row should sum to 1 because this represents all the transitions

```
library(HMM)

### StartProbs
# Initial state will be any of the 10 states with equal probability
startProbs = rep(0.1, 10)

### States
states = seq(1,10,1)

### Symbols
symbols = seq(1,10, 1)
```

TransitionMatrix: The transition matrix is a 10x10 matrix. From the description we learned that the robot chooses with equal probability to stay or move to the next sector. This results in two different alternatives with transition prob = 0.5

```
transition_prob <-    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)

#R sums by column as default, therfore we set byrow = TRUE
transition_matrix <- matrix(data = transition_prob,
                            ncol = 10,
                            nrow = 10,
                            byrow = TRUE)
```

Emmission Matrix: The emmission matrix is a 10x10 matrix. From the description we learned that the noise creates an uncertainty which results in that that the observation will have anuncertainty of +- 2 positions. This results in 5 different alternatives with each emmission prob = 0.2

```
emission_prob <- 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, 0.2,
                   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)

emission_matrix <- matrix(data = emission_prob,
                          ncol = 10,
                          nrow = 10,
                          byrow = TRUE)

# Create the hiddent markov model for the describe situation
HMM = initHMM(States = states,
              Symbols = symbols,
              startProbs = startProbs,
              transProbs = transprobs,
              emissionProbs = emission_matrix)
print(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
```

Task 2

Simulate the Hidden Markov Model for 100 steps

- simHMM(HMM, length) simulates a path of states and observations for a given HMM
- length = length of the sequence of observations and states

```
observation_sequence = 100
HMM_simulation = simHMM(hmm = HMM, length = observation_sequence)
```

Print the simulated path of the states and observations for the HMM - states = The simulated hidden state - observation = The simulated observations

```
print(HMM_simulation)
```

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

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

Task 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.

Compute the filtered and smoothed probabilty distributions: These two calculations are two independent parts of the forward-backward algorithm used to compute alpha and beta.

Smoothed probability: Is the probabilty of a certain state at a certain time given all of the data (the full sequence)

Filtered data: The probabilty of a certain state a at a certain time given the data up until this moment

Smoothed probability: As we can see the posterior function gives us the smoothing probability in its normal form (not logaritm) and normalized.

```
## Smoothed probability
# p(z^t|x^0:T) = alpha(z^t)*beta(z^t) ~ Normalized over Z^t
# The smoothed data is a conditional probability which computes
# the probability of a certain state at a certain time given all of
# the data. This is the same as computing the posterior probability
# of being in state X at time k for a given sequence of observations
# and a given HMM

smoothing_prob = posterior(HMM, HMM_simulation$observation)
print(smoothing_prob[,1:5])

##       index
## states         1         2          3         4         5
##      1  0.6855689 0.3711377 0.05670659 0.0000000 0.0000000
##      2  0.3144311 0.6288623 0.94329341 0.2268264 0.0000000
##      3  0.0000000 0.0000000 0.00000000 0.7731736 0.3969461
##      4  0.0000000 0.0000000 0.00000000 0.0000000 0.6030539
```

```
##      5  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
##      6  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
##      7  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
##      8  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
##      9  0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
##      10 0.0000000 0.0000000 0.00000000 0.0000000 0.0000000
```

Filtered probability

```
## Filtered probability
# P(Z^t|X^0:t) = alpha(Z^t) ~ Normalized over Z^t
# The filtered probability is the conditional probability which
# computes the probability of being in state X at time k given
# the sequence of data up until time k and a given HMM. To compute this we
use
# the forward-function which computes the forward probabilities which defines
as
# the probability of observing the sequence of observations e_1 uo to e_k and
# that the state at time k is X. Using the forward algorithm we compute the
# alpha(z^t)

# The forward algorithm return the log probability
alpha_log = forward(hmm = HMM, observation = HMM_simulation$observation)
# Convert to normal prob
alpha = exp(alpha_log)
#filtered_prob = alpha / colSums(alpha)
#print(filtered_prob)
# margin = 2 => divide by colSums
# Normalize the output
filtered_prob = prop.table(alpha, margin = 2)
print(filtered_prob[,1:5])
```

```
##       index
## states   1   2    3         4         5
##      1  0.2 0.2 0.25 0.0000000 0.0000000
##      2  0.2 0.4 0.75 0.5714286 0.2857143
##      3  0.2 0.4 0.00 0.4285714 0.5000000
##      4  0.2 0.0 0.00 0.0000000 0.2142857
##      5  0.2 0.0 0.00 0.0000000 0.0000000
##      6  0.0 0.0 0.00 0.0000000 0.0000000
##      7  0.0 0.0 0.00 0.0000000 0.0000000
##      8  0.0 0.0 0.00 0.0000000 0.0000000
##      9  0.0 0.0 0.00 0.0000000 0.0000000
##      10 0.0 0.0 0.00 0.0000000 0.0000000
```

Viterbi

```
## Viterbi
# Compute the most likely permitted path
viterbi_path = viterbi(HMM, HMM_simulation$observation)
```

```r
# Returns a vector of strings displaying the most probable path of states
print(viterbi_path)
```

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

Task 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.

To review the accuracy of the different probability distribution we went to look which state the model classifies as to be most likely in each observation. Looking at the example below we would for example classify the first observation to be in state 8 since this has the highest probability

```r
# Extract the state which has the highest conditional probability for each
observation
smoothing = apply(smoothing_prob, MARGIN = 2, which.max)
accuracy_smooth = mean(smoothing == HMM_simulation$states)

filtered = apply(filtered_prob, MARGIN = 2, which.max)
accuracy_filtered = mean(filtered == HMM_simulation$states)

accuracy_viterbi = mean(viterbi_path == HMM_simulation$states)

print(accuracy_table)
```

```
##   Smoothed Filtered Viterbi
## 1     0.68     0.48    0.53
```

The accuracy viterbi model has a lower accuracy than the filtered model. This is because the viterbi model computes the most likely path of states while the smoothed distribution computes the likeli state in every singel time step. The problem with the smoothed distribution is that the resulting path might not be a permitted path. The viterbi model will

give a permitted path but it will not be the most likely state in each time step. There could for example be alot of different permitted paths and consequently, by using the smoothing model and choosing the most likely state in each time step we will have a bigger chance of guessing righs, hence the higher accuracy.

Task 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 ?

```
# Function for creating new simulation computing the different probability
models, i.e smooting, and computing their respective accuracy at classifying
# the simulated data
computations = function(hmm, sampleSize){
  simulation = simHMM(hmm = hmm, length = sampleSize)

  smoothing = posterior(hmm, simulation$observation)
  #Compute filtering probability distribution
  alpha_log = forward(hmm, simulation$observation)
  alpha = exp(alpha_log)
  filtering = prop.table(alpha, margin = 2)
  #Compute viterbi probability distribution
  viterbi = viterbi(hmm, simulation$observation)

  #Compute accuracies
  smoothing_max = apply(smoothing, MARGIN = 2, FUN = which.max)
  accuracy_smoothing = mean(smoothing_max == simulation$states)
  filtering_max = apply(filtering, MARGIN = 2, FUN = which.max)
  accuracy_filtering = mean(filtering_max == simulation$states)
  accuracy_viterbi = mean(viterbi == simulation$states)

  accuracy = data.frame(Smooting = accuracy_smoothing, Filtering =
accuracy_filtering, Viterbi = accuracy_viterbi)

  result = list("accuracy" = accuracy,
                "smoothing" = smoothing,
                "filtering" = filtering,
                "viterbi" = viterbi)

  return(result)
}
```

Using the above function i create new simulations and compute their respective accuracies

```
# Compute new new accuracies using newly computed simulations

# Case 1
```

```
computation1 = computations(hmm = HMM, sampleSize = 100)
# Case 2
computation2 = computations(hmm = HMM, sampleSize = 100)
# Case 3
computation3 = computations(hmm = HMM, sampleSize = 100)
```

Running the previous steps again and computing the accuracy for the different models we can see that the the previous notation is consitent. Although the percertage varies in different runs it is always true that the accuracy_smoothed > accuracy_viterbi and accuracy_smoothed > accuracy_filtered The smoothing algorithm perfroms better since it makes use of both forward (alpha) and backward (beta) algorithms in its computations. The forward algorithm uses previous steps to compute the current alpha value while the backward algorithmuses the next step to compute the current beta value. Consequently it uses more information than the filtering which only uses the forward algorithm (alpha) alone. This allows for the algorithm to compute more accurate results.

Viterbi The accuracy viterbi model has a lower accuracy than the filtered model. This is because the viterbi mode computes the most likely path of states while the smoothed distribution computes the likeli state in every singel time step. The problem with the smoothed distribution is that the resulting path might not be a permitted path. The viterbi model will give a permitted path but it will not be the most likely state in each time step. There could for example be alot of different permitted paths and consequently, by using the smoothing model and choosing the most likely state in each time step we will have a bigger chance of guessing righs, hence the higher accuracy.

```
##   Smooting Filtering Viterbi
## 1     0.67      0.59    0.47

##   Smooting Filtering Viterbi
## 1     0.65      0.55    0.46

##   Smooting Filtering Viterbi
## 1      0.6      0.55    0.47
```

#Task 6 *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, also referred to as Shannor Entropy is a measure of dissorder or randomness of a system. A system with high entropy is consequently more uncertain due to its randomness. Entropy is a measure of uncertainty which allows us to make precise statements and perform computations with regard to one of life's most pressing issue, namely not knowing how things will turn out.

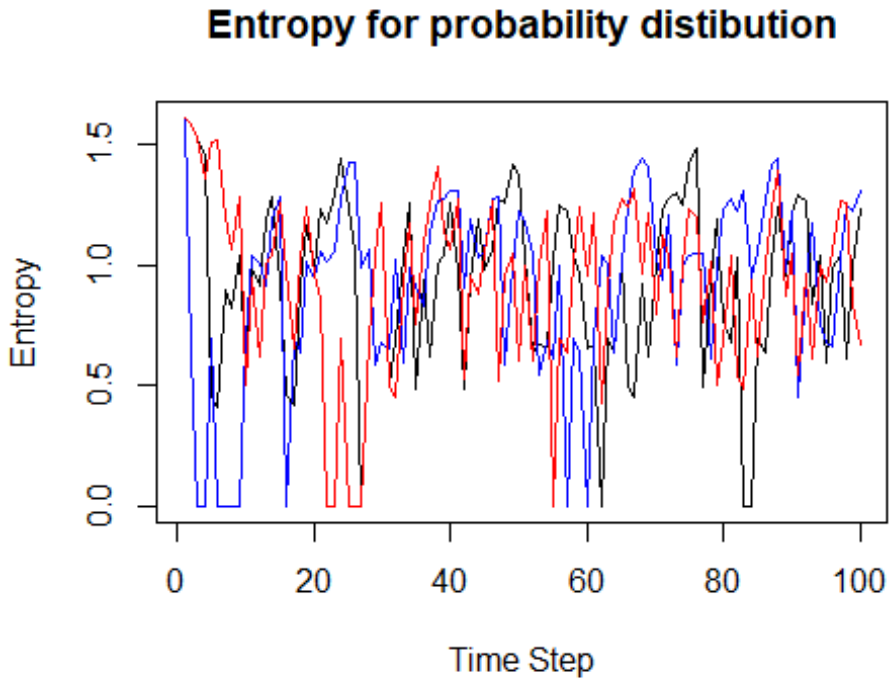
Entropy facts: - Entropy is maximal for uniform distributions. Since the possible - outcome is equally probable everywhere. I.e in a bernouli trial then max - entropy is reached using p = 0.5 => largest spread. - Entropy is additive for independent events

What we are trying to answer is if the increase of Time Step results in a lower uncertainty, in other words lower entropy. Considering this fact and knowing what the aim is it is reasonable that we use the filtering distribution to compute the entropy since this only

takes the observations up until the current time step into account in its computations. In this way we can then answer the question wheter the state of time step t +1 is more or less uncertain than the state of time step t.

The robots location is given by its porbability distribution in each time step. By looking at the entropy for the distribution in each time step we can make conclusions of the uncertainty of knowing what state the robot is in in a certain moment. Entropy increases when increasing the number of possible outcomes i.e states

```
plot(apply(computation1$filtering, MARGIN = 2, entropy.empirical),
     type = 'l', main = "Entropy for probability distibution", ylab =
"Entropy",  xlab = "Time Step")
points(apply(computation2$filtering, MARGIN = 2, entropy.empirical), type =
'l', col = "blue")
points(apply(computation3$filtering, MARGIN = 2, entropy.empirical), type =
'l', col = "red")
```



By observing the plots it is not possible to conclude any evidence which would say that the uncertainty of the state would get larger nore lower. At some cass the entropy is down to 0 which would mean that it has zero probability, but then is goes up again. By time stamp 100 we can sill see the same amount of fluctuation as there was in the beginning of the sequence. This makes sence in realtion to our theories which has stated that since the structure is stationary, then the transition matrix and emission matrix will stay as they currently are. Adding another step to the sequence will consequently not add another parameter any new parameters or affect the previous computations in anyway. Since the

transition and emission probabilites are stationary and will ot change the uncertainty will consequently also stay as it is.

Task 7

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

We can predict the probabilities of the hidden states at time step 101 by looking at the formula for the ml estimate of ml est. z(t+1)| z(t) = sum(t, T-1) over p(z(t), z(t+1) | x (0:t)) (Normalized) We know the probabilities of the current observed state p(z(t)|x(0:t)) - the filtered distribution and we want to know the distribution for z(t+1). In order to extract this we need to factorize the expression by marginalizing out z(t). we know that z(t+1) is independent from all previous states given z(t)

We know the probabilities of the current (100) step. The transition probabilities between time steps will always be the same so this we already have from the initial transition matrix. We can consequently predict the probabilities for the next step by doing the matrix multiplication

```
current_prob = computation1$filtering[,100]
next_prob = t(transition_matrix) %*%  current_prob

## Probabilities for the hidden states in step 101:  0.375 0.3000396
0.08751978 0 0 0 0 0 0.03748022 0.1999604
```

Code

Task1

```
### Task 1 ###
# Build a hidden Markov model for the scenario described above

#install.packages("HMM")
library(HMM)

# To create an HMM we use the initHMM function which takes in the arguments
# States = Vector with the different states for the model
# Symbols = Vector with the names of the symbols
# startProbs = vector with the starting probabilities of all teh states
# transProbs = Stochastic matrix holding the transition probabilities between
states Z
# emissionProbs = Stochastic matrix holding the emission probabilities of the
states X

# stochastic matrix => Each row should sum to 1 because this represents all
the
# transitions
```

```r
### StartProbs
# Initial state will be any of the 10 states with equal probability
startProbs = rep(0.1, 10)

### States
states = seq(1,10,1)

### Symbols
symbols = seq(1,10, 1)

### TransitionMatrix
# The transition matrix is a 10x10 matrix. From the description we learned
that the
# robot chooses with equal probability to stay or move to the next sector.
This results
# in two different alternatives with transition prob = 0.2

#Varför blir denna fel ?
transition_prob <-    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)

#R sums by column as default, therfore we set byrow = TRUE
transition_matrix <- matrix(data = transition_prob,
                            ncol = 10,
                            nrow = 10,
                            byrow = TRUE)

transition_matrix

transprobs = matrix(0, 10, 10)
diag(transprobs) = 0.5
diag(transprobs[,-1]) = 0.5
transprobs[10,1] = 0.5
transprobs

### Emmission Matrix
# The emmission matrix is a 10x10 matrix. From the description we learned
that the
# noise creates an uncertainty which results in that that the observation
```

```
will have an
# uncertainty of +- 2 positions. This results in 5 different alternatives
with each
# emmission prob = 0.2
emission_prob <- 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)

emission_matrix <- matrix(data = emission_prob,
                          ncol = 10,
                          nrow = 10,
                          byrow = TRUE)

# Create the hiddent markov model for the describe situation
HMM = initHMM(States = states,
              Symbols = symbols,
              startProbs = startProbs,
              transProbs = transprobs,
              emissionProbs = emission_matrix)


#Print the initialized Markov Model
print(HMM)
```

Task 2

```
### Task 2 ###
#Simulate the Hidden Markov Model for 100 steps

# simHMM(HMM, length) simulates a path of states and observations for a given
HMM
# length = length of the sequence of observations and staes


observation_sequence = 200
HMM_simulation = simHMM(hmm = HMM, length = observation_sequence)

#Print the simulated path of the states and observations for the HMM
# states = The simulated hidden state
# observation = The simulated observations
print(HMM_simulation)
```

Task 3

```
### Task 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.

## Compute the filtered and smoothed probabilty distributions
# These two calculations are two independent parts of the forward-backward
# algorithm used to compute alpha and beta.

# Smoothed probability: Is the probabilty of a certain state at a certain
# time given all of the data (the full sequence)

# Filtered data: The probabilty of a certain state a at a certain time
# given the data up until this moment


## Smoothed probability
# The smoothed data is a conditional probability which computes
# the probability of a certain state at a certain time given all of
# the data. This is the same as computing the posterior probability
# of being in state X at time k for a given sequence of observations
# and a given HMM

smoothing_prob = posterior(HMM, HMM_simulation$observation)
print(smoothing_prob)

## Filtered probability
# P(Z^t|X^0:t) = alpha(Z^t)/normalized
# The filtered probability is the conditional probability which
# computes the probability of being in state X at time k given
# the sequence of data up until time k and a given HMM. To compute this we
use
# the forward-function which computes the forward probabilities which defines
as
# the probability of observing the sequence of observations e_1 uo to e_k and
# that the state at time k is X. Using the forward algorithm we compute the
# alpha(z^t)

# The forward algorithm return the log probability
alpha_log = forward(hmm = HMM, observation = HMM_simulation$observation)
# Convert to normal prob
alpha = exp(alpha_log)
#filtered_prob = alpha / colSums(alpha)
#print(filtered_prob)
# margin = 2 => divide by colSums
# Normalize the output
```

```r
filtered_prob = prop.table(alpha, margin = 2)
print(filtered_prob)

## Viterbi
# Compute the most likeli path
viterbi_path = viterbi(HMM, HMM_simulation$observation)
# Returns a vector of strings displaying the most probable path of states
print(viterbi_path)
```

Task 4

```r
### Task 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.



## Smoothing Probability
# To review the accuracy of the smoothed probability distribution we went to
look which state the model classifies as to be most likely in each
# observation. Looking at the example below we would for example classify the
first observation to be in state 8 since this has the highest probability

# Should we see this as an observation or a second time step ??
# As we can see the table is normalized since it
smoothing_prob[,1:5]

# Using the following funciton we extract the state which has the highest
conditional probability for each observation
smoothing = apply(smoothing_prob, MARGIN = 2, which.max)
HMM_simulation$states
accuracy_smooth = mean(smoothing == HMM_simulation$states)
print(accuracy_smooth)
```

```r
## Filtered probability
filtered_prob[,1:5]
filtered = apply(filtered_prob, MARGIN = 2, which.max)
accuracy_filtered = mean(filtered == HMM_simulation$states)
print(accuracy_filtered)

# The accuracy for the smoothed probability is higher than the one for the
filtered probability. This is probably due to the fact that the smoothed
# probability uses all of the observations (the whole sequence of data) in
order to compute the probabilty for the most likely states. This compared
# to the filtered probability which is only conditioned on the data up to
that classification point, then becomes more uncertain since it has less
# data.

## Most likely path - Viterbi
viterbi_path
accuracy_viterbi = mean(viterbi_path == HMM_simulation$states)
print(accuracy_viterbi)

accuracy_table = data.frame(Smoothed = accuracy_smooth, Filtered =
accuracy_filtered, Viterbi = accuracy_viterbi)
print(accuracy_table)

# The accuracy viterbi model has a lower accuracy than the filtered model.
This is because the viterbi model computes the most likely path of states
# while the smoothed distribution computes the likeli state in every singel
time step. The problem with the smoothed distribution is that the resulting
path
# might not be a permitted path. The viterbi model will give a permitted path
but it will not be the most likely state in each time step. There could for
# example be alot of different permitted paths and consequently, by using the
smoothing model and choosing the most likely state in each time step we
# will have a bigger chance of guessing righs, hence the higher accuracy.
```

Task 5

```r
### Task 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 ?



# Function for creating new simulation computing the different probability
models, i.e smooting, and computing their respective accuracy at classifying
# the simulated data
computations = function(hmm, sampleSize){
  simulation = simHMM(hmm = hmm, length = sampleSize)
```

```r
  smoothing = posterior(hmm, simulation$observation)
  #Compute filtering probability distribution
  alpha_log = forward(hmm, simulation$observation)
  alpha = exp(alpha_log)
  filtering = prop.table(alpha, margin = 2)
  #Compute viterbi probability distribution
  viterbi = viterbi(hmm, simulation$observation)

  #Compute accuracies
  smoothing_max = apply(smoothing, MARGIN = 2, FUN = which.max)
  accuracy_smoothing = mean(smoothing_max == simulation$states)
  filtering_max = apply(filtering, MARGIN = 2, FUN = which.max)
  accuracy_filtering = mean(filtering_max == simulation$states)
  accuracy_viterbi = mean(viterbi == simulation$states)

  accuracy = data.frame(Smooting = accuracy_smoothing, Filtering =
accuracy_filtering, Viterbi = accuracy_viterbi)

  result = list("accuracy" = accuracy,
                "smoothing" = smoothing,
                "filtering" = prop.table(alpha, margin = 2),
                "filtering_2" = filtering)

  return(result)
}

# Compute new new accuracies using newly computed simulations
# Case 1
computation1 = computations(hmm = HMM, sampleSize = 100)
# Case 2
computation2 = computations(hmm = HMM, sampleSize = 100)
# Case 3
computation3 = computations(hmm = HMM, sampleSize = 100)

computation1$accuracy
computation2$accuracy
computation3$accuracy

# Running the previous steps again and computing the accuracy for the
different models we can see that the the previous notation is consitent.
# Although the percertage varies in different runs it is always true that the
accuracy_smoothed > accuracy_viterbi and accuracy_smoothed >
accuracy_filtered
# The smoothing algorithm perfroms better since it makes use of both forward
(alpha) and backward (beta) algorithms in its
# computations. The forward algorithm uses previous steps to compute the
current alpha value while the backward algorithm
# uses the next step to compute the current beta value. Consequently it uses
```

```
more information than the filtering
# which only uses the forward algorithm (alpha) alone. This allows for the
algorithm to compute more accurate results.
# Viterbi
# The accuracy viterbi model has a lower accuracy than the filtered model.
This is because the viterbi model computes the most likely path of states
# while the smoothed distribution computes the likeli state in every singel
time step. The problem with the smoothed distribution is that the resulting
path
# might not be a permitted path. The viterbi model will give a permitted path
but it will not be the most likely state in each time step. There could for
# example be alot of different permitted paths and consequently, by using the
smoothing model and choosing the most likely state in each time step we
# will have a bigger chance of guessing righs, hence the higher accuracy.
```

Task 6

```
### Task 6 ###
# 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.

#install.packages("entropy")
library(entropy)

# Entropy, also referred to as Shannor Entropy is a measure of dissorder or
randomness of a system. A system with high entropy is consequently
# more uncertain due to its randomness. Entropy is  a measure of uncertainty
which allows us to  make precise  statements and perform computations
# with regard to one of life's most pressing issue, namely not knowing how
things will turn out.

# Entropy facts: Entropy is maximal for uniform  distributions. Since the
possible
# outcome is equally probable everywhere. I.e in a bernouli trial then max
# entropy is reached using p = 0.5 => largest spread.
# Entropy is additive for independent events
# Entropy increases when increasing the number of possible outcomes i.e
states

# What we are trying to answer is if the increase of Time Step results in a
lower uncertainty, in other words lower entropy. Considering this fact and
knowing what
# the aim is it is reasonable that we use the filtering distribution to
compute the entropy since this only takes the observations up until the
current time step
# into account in its computations. In this way we can then answer the
```

```
question wheter the state of time step t +1 is more or less uncertain
# than the state of time step t.

# The robots location is given by its porbability distribution in each time
step. By looking at the entropy for the distribution in each time step we can
# make conclusions of the uncertainty of knowing what state the robot is in
in a certain moment

plot(apply(computation1$filtering, MARGIN = 2, entropy.empirical),
     type = 'l', main = "Entropy for Smoothed distribution", ylab =
"Entropy",  xlab = "Time Step")
points(apply(computation2$filtering, MARGIN = 2, entropy.empirical), type =
'l', col = "blue")
points(apply(computation3$filtering, MARGIN = 2, entropy.empirical), type =
'l', col = "red")

# By observing the plots it is not possible to conclude any evidence which
would say that the uncertainty of the state would get larger
# nore lower. At some cass the entropy is down to 0 which would mean that it
has zero probability, but then is goes up again.
# By time stamp 100 we can sill see the same amount of fluctuation as there
was in the beginning of the sequence. This makes
# sence in realtion to our theories which has stated that since the structure
is stationary, then the transition matrix and emission matrix
# will stay as they currently are. Adding another step to the sequence will
consequently not add another parameter any new parameters or
# affect the previous computations in anyway. Since the transition and
emission probabilites are stationary and will ot change the
# uncertainty will consequently also stay as it is.

# Is there a reason for that the entropy is lower for the filtered than the
smoothed?
```

Task 7

```
### Task 7 ###
# Consider any of the samples above of length 100. Compute the probabilities
of the hidden states at the time step 101.

# We can predict the probabilities of the hidden states at time step 101 by
looking at the formula for the ml estimate of
# ml est. z(t+1)| z(t) = sum(t, T-1) over  p(z(t), z(t+1) | x (0:t))
(Normalized)
# We know the probabilities of the current observed state p(z(t)|x(0:t)) -
the filtered distribution
# and we want to know the distribution for z(t+1). In order to extract this
we need to factorize the expression by marginalizing out z(t)
# we know that z(t+1) is independent from all previous states given z(t)

# We know the probabilities of the current (100) step. The transition
```

```
probabilities between time steps will always be the same
# so this we already have from the initial transition matrix. We can
consequently predict the probabilities for the next step
# by doing the matrix multiplication. Due to the form of the transition
matrix we take the transpose of the matrix in order to multiply with the
transition probabilityies for each state i.e column 1 for state 1 etc.
current_prob = computation1$filtering[,100]
next_prob = t(transition_matrix) %*%  current_prob

cat("Probabilities for the hidden states in step 101: ", next_prob)
which.max(next_prob != 0)
```

CODE

```
library(HMM)

### TASK 1 ###

### StartProbs
# Initial state will be any of the 10 states with equal probability
startProbs = rep(0.1, 10)

### States
states = seq(1,10,1)

### Symbols
symbols = seq(1,10, 1)

transition_prob <-    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)

#R sums by column as default, therfore we set byrow = TRUE
transition_matrix <- matrix(data = transition_prob,
                            ncol = 10,
                            nrow = 10,
                            byrow = TRUE)

transition_matrix

transprobs = matrix(0, 10, 10)
diag(transprobs) = 0.5
diag(transprobs[,-1]) = 0.5
```

```r
transprobs[10,1] = 0.5
transprobs

emission_prob <- 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)

emission_matrix <- matrix(data = emission_prob,
                          ncol = 10,
                          nrow = 10,
                          byrow = TRUE)

# Create the hiddent markov model for the describe situation
HMM = initHMM(States = states,
              Symbols = symbols,
              startProbs = startProbs,
              transProbs = transprobs,
              emissionProbs = emission_matrix)


observation_sequence = 100
HMM_simulation = simHMM(hmm = HMM, length = observation_sequence)

smoothing_prob = posterior(HMM, HMM_simulation$observation)

alpha_log = forward(hmm = HMM, observation = HMM_simulation$observation)
# Convert to normal prob
alpha = exp(alpha_log)
#filtered_prob = alpha / colSums(alpha)
#print(filtered_prob)
# margin = 2 => divide by colSums
# Normalize the output
filtered_prob = prop.table(alpha, margin = 2)

viterbi_path = viterbi(HMM, HMM_simulation$observation)


### TASK 4
smoothing = apply(smoothing_prob, MARGIN = 2, which.max)
accuracy_smooth = mean(smoothing == HMM_simulation$states)
filtered = apply(filtered_prob, MARGIN = 2, which.max)
accuracy_filtered = mean(filtered == HMM_simulation$states)
```

```r
accuracy_viterbi = mean(viterbi_path == HMM_simulation$states)
accuracy_table = data.frame(Smoothed = accuracy_smooth, Filtered =
accuracy_filtered, Viterbi = accuracy_viterbi)


### TASK 5
computations = function(hmm, sampleSize){
  simulation = simHMM(hmm = hmm, length = sampleSize)

  smoothing = posterior(hmm, simulation$observation)
  #Compute filtering probability distribution
  alpha_log = forward(hmm, simulation$observation)
  alpha = exp(alpha_log)
  filtering = prop.table(alpha, margin = 2)
  #Compute viterbi probability distribution
  viterbi = viterbi(hmm, simulation$observation)

  #Compute accuracies
  smoothing_max = apply(smoothing, MARGIN = 2, FUN = which.max)
  accuracy_smoothing = mean(smoothing_max == simulation$states)
  filtering_max = apply(filtering, MARGIN = 2, FUN = which.max)
  accuracy_filtering = mean(filtering_max == simulation$states)
  accuracy_viterbi = mean(viterbi == simulation$states)

  accuracy = data.frame(Smooting = accuracy_smoothing, Filtering =
accuracy_filtering, Viterbi = accuracy_viterbi)

  result = list("accuracy" = accuracy,
                "smoothing" = smoothing,
                "filtering" = filtering,
                "viterbi" = viterbi)

  return(result)
}

# Case 1
computation1 = computations(hmm = HMM, sampleSize = 100)
# Case 2
computation2 = computations(hmm = HMM, sampleSize = 100)
# Case 3
computation3 = computations(hmm = HMM, sampleSize = 100)



### TASK 6 ###
library(entropy)
```

```
### TASK 7 ###
current_prob = computation1$filtering[,100]
next_prob = t(transition_matrix) %*%  current_prob
```