Advanced Machine Learning (732A96) - Lab 02

Hector Plata (hecpl268)

Contents

The purpose of the lab is to put in practice some of the concepts convered in the elctures. To do so, you are asked to model the behaviour of a robot that walks around a ring. The ring is divided into 10 sectors. At any give 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 thought: If the robot is in the sector i, then th edevice will report that the robot is in the sectors $[i-2, i+2]$ with equal probability.

Task 1

Build a hidden Markon Model (HMM) for the scenario described above.

```
library(HMM)

# Defining the States and Symbols.
states = paste(rep("Z", 10), 1:10, sep="")
symbols = paste(rep("X", 10), 1:10, sep="")

# Transition probability matrix.
P = diag(0, 10, 10)
P[1:9, 2:10] = diag(0.5, 9, 9)
P = P + diag(0.5, 10, 10)
P[10, 1] = 0.5

# Emision probability matrix.
E = matrix(0, 10, 10)

for(i in 1:10)
{
  for(j in c(-2,-1,0,1,2))
  {
    if (i + j < 1)
    {
      k = 10 + i + j
```

```
    }
    else if (i + j > 10)
    {
      k = i - 10 + j
    }
    else
    {
      k = i + j
    }

    E[i, k] = 1/5
  }
}

# Defining the HMM.
hmm = initHMM(States=states,
              Symbols=symbols,
              transProbs=P,
              emissionProbs=E)

print(hmm)
```

```
## $States
##  [1] "Z1"  "Z2"  "Z3"  "Z4"  "Z5"  "Z6"  "Z7"  "Z8"  "Z9"  "Z10"
##
## $Symbols
##  [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10"
##
## $startProbs
##  Z1  Z2  Z3  Z4  Z5  Z6  Z7  Z8  Z9 Z10
## 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##
## $transProbs
##      to
## from   Z1  Z2  Z3  Z4  Z5  Z6  Z7  Z8  Z9 Z10
##   Z1  0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   Z2  0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   Z3  0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
##   Z4  0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
##   Z5  0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
##   Z6  0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
##   Z7  0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
##   Z8  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
##   Z9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
##   Z10 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5
##
## $emissionProbs
##       symbols
## states  X1  X2  X3  X4  X5  X6  X7  X8  X9 X10
##    Z1  0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
##    Z2  0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
##    Z3  0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
##    Z4  0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
##    Z5  0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
```

```
##    Z6  0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
##    Z7  0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
##    Z8  0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
##    Z9  0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
##    Z10 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

Task 2

Simulate the HMM for 100 time steps.

```
set.seed(4)
sim = simHMM(hmm, 100)
print(sim)
```

```
## $states
##   [1] "Z7"  "Z7"  "Z7"  "Z7"  "Z8"  "Z8"  "Z9"  "Z9"  "Z9"  "Z10" "Z10"
##  [12] "Z1"  "Z2"  "Z2"  "Z3"  "Z3"  "Z4"  "Z5"  "Z5"  "Z5"  "Z5"  "Z5"
##  [23] "Z5"  "Z6"  "Z7"  "Z8"  "Z8"  "Z9"  "Z9"  "Z9"  "Z9"  "Z10" "Z10"
##  [34] "Z10" "Z1"  "Z1"  "Z2"  "Z2"  "Z3"  "Z3"  "Z4"  "Z4"  "Z5"  "Z6"
##  [45] "Z7"  "Z7"  "Z8"  "Z9"  "Z9"  "Z9"  "Z10" "Z10" "Z10" "Z10" "Z10"
##  [56] "Z1"  "Z2"  "Z3"  "Z4"  "Z5"  "Z5"  "Z6"  "Z7"  "Z8"  "Z8"  "Z9"
##  [67] "Z10" "Z10" "Z10" "Z10" "Z1"  "Z1"  "Z2"  "Z3"  "Z4"  "Z5"  "Z6"
##  [78] "Z6"  "Z6"  "Z6"  "Z7"  "Z8"  "Z9"  "Z9"  "Z9"  "Z10" "Z1"  "Z1"
##  [89] "Z2"  "Z2"  "Z3"  "Z3"  "Z4"  "Z4"  "Z4"  "Z4"  "Z5"  "Z6"  "Z6"
## [100] "Z7"
##
## $observation
##   [1] "X7"  "X9"  "X7"  "X5"  "X10" "X10" "X10" "X7"  "X8"  "X9"  "X8"
##  [12] "X2"  "X4"  "X1"  "X5"  "X1"  "X5"  "X4"  "X6"  "X7"  "X7"  "X3"
##  [23] "X6"  "X7"  "X8"  "X9"  "X6"  "X7"  "X1"  "X7"  "X1"  "X10" "X8"
##  [34] "X10" "X3"  "X3"  "X10" "X4"  "X1"  "X2"  "X5"  "X2"  "X3"  "X6"
##  [45] "X8"  "X5"  "X10" "X8"  "X10" "X1"  "X8"  "X10" "X2"  "X9"  "X8"
##  [56] "X3"  "X4"  "X4"  "X5"  "X4"  "X6"  "X6"  "X5"  "X7"  "X7"  "X9"
##  [67] "X10" "X10" "X10" "X1"  "X9"  "X10" "X4"  "X1"  "X4"  "X7"  "X4"
##  [78] "X7"  "X5"  "X4"  "X8"  "X9"  "X7"  "X8"  "X1"  "X8"  "X10" "X2"
##  [89] "X10" "X3"  "X1"  "X3"  "X6"  "X3"  "X3"  "X2"  "X6"  "X5"  "X7"
## [100] "X8"
```

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.

```
sim_states = sim$states
sim_obs = sim$observation
filtered = prop.table(exp(forward(hmm, sim_obs)), 2)
smooth = posterior(hmm, sim_obs)
most_prob_path = viterbi(hmm, sim_obs)
print(most_prob_path)
```

```
##   [1] "Z6"  "Z7"  "Z7"  "Z7"  "Z8"  "Z8"  "Z8"  "Z8"  "Z8"  "Z9"  "Z10"
##  [12] "Z1"  "Z2"  "Z2"  "Z3"  "Z3"  "Z3"  "Z3"  "Z4"  "Z5"  "Z5"  "Z5"
##  [23] "Z5"  "Z5"  "Z6"  "Z7"  "Z7"  "Z8"  "Z9"  "Z9"  "Z9"  "Z9"  "Z10"
```

```
##  [34] "Z1"  "Z1"  "Z1"  "Z1"  "Z2"  "Z2"  "Z2"  "Z3"  "Z3"  "Z4"  "Z5"
##  [45] "Z6"  "Z7"  "Z8"  "Z8"  "Z8"  "Z9"  "Z9"  "Z9"  "Z10" "Z10" "Z10"
##  [56] "Z1"  "Z2"  "Z2"  "Z3"  "Z4"  "Z5"  "Z6"  "Z7"  "Z8"  "Z9"  "Z10"
##  [67] "Z1"  "Z1"  "Z1"  "Z1"  "Z1"  "Z1"  "Z2"  "Z3"  "Z4"  "Z5"  "Z5"
##  [78] "Z5"  "Z5"  "Z5"  "Z6"  "Z7"  "Z7"  "Z8"  "Z9"  "Z10" "Z1"  "Z1"
##  [89] "Z1"  "Z1"  "Z2"  "Z3"  "Z4"  "Z4"  "Z4"  "Z4"  "Z4"  "Z4"  "Z5"
## [100] "Z6"
```

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 canc ompare two vector 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.

```
# Getting the most probable states of each distribution.
filtered_mps = as.numeric(apply(filtered, 2, which.max))
smooth_mps = as.numeric(apply(smooth, 2, which.max))
numeric_mpp = as.numeric(gsub("[^0-9.]", "",  most_prob_path))  # Numeric most probably path.
y = as.numeric(gsub("[^0-9.]", "",  sim_states))

# Getting the accuracy for each object.

filtered_acc = sum(filtered_mps == y) / length(y)
smooth_acc = sum(smooth_mps == y) / length(y)
mpp_acc = sum(numeric_mpp == y) / length(y)

print(paste("Filtered accuracy:", filtered_acc))
```

```
## [1] "Filtered accuracy: 0.55"
```

```
print(paste("Smooth accuracy:", smooth_acc))
```

```
## [1] "Smooth accuracy: 0.69"
```

```
print(paste("Most probable path accuracy:", mpp_acc))
```

```
## [1] "Most probable path accuracy: 0.49"
```

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?

The smoothed distribution should be more accurate than the filtered distribution because it containes more information since it's conditioned on the whole history of the markov process, while the filtered is only conditioned up to the time step where the probability is being evaluated. As for the smoothed distribution against the most probable path something similar happens. The viterbi algorithm is a dynamic programming optimization problem and as such it works backwards from the last value function until the first one following the bellman equation. As such, viterbi doesn't have access to the full distribution at each time step like the smoothed distribution and it also has to work one step at a time recursively.

```r
library(ggplot2)

do_task_4 = function(hmm, n_sim=100, task_5=TRUE)
{
  # Smiulation.
  sim = simHMM(hmm, n_sim)

  sim_states = sim$states
  sim_obs = sim$observation
  filtered = prop.table(exp(forward(hmm, sim_obs)), 2)
  smooth = posterior(hmm, sim_obs)
  most_prob_path = viterbi(hmm, sim_obs)

  # Getting the most probable states of each distribution.
  filtered_mps = as.numeric(apply(filtered, 2, which.max))
  smooth_mps = as.numeric(apply(smooth, 2, which.max))
  numeric_mpp = as.numeric(gsub("[^0-9.]", "",  most_prob_path))  # Numeric most probably path.
  y = as.numeric(gsub("[^0-9.]", "",  sim_states))

  # Getting the accuracy for each object.
  filtered_acc = sum(filtered_mps == y) / length(y)
  smooth_acc = sum(smooth_mps == y) / length(y)
  mpp_acc = sum(numeric_mpp == y) / length(y)

  if (task_5 == TRUE)
  {
    return(c(filtered_acc, smooth_acc, mpp_acc))
  }
  else
  {
    return(filtered)
  }
}

n_times = 100
results = matrix(0, n_times, 3)
for (i in 1:n_times)
{
  results[i, ] = do_task_4(hmm)
}

results = as.data.frame(results, col.names=c("Filtered Acc", "Smooth Acc", "MPP Acc"))

p = ggplot() +
    geom_line(aes(x=1:n_times, y=results[, 1], color="Filtered Acc")) +
    geom_line(aes(x=1:n_times, y=results[, 2], color="Smooth Acc")) +
    geom_line(aes(x=1:n_times, y=results[, 3], color="MPP Acc")) +
    labs(x="Simulation", y="Accuracy")

print(p)
```

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`

As we can see below on the plot, more observations doesn't translate into a more accurate representation of the robots position. The entropy stays relatively the same over the whole range of observations. This means that the amount of information in the probability distribution (filtered) doesn't change regarding the number of observations which is expected from a markov process.

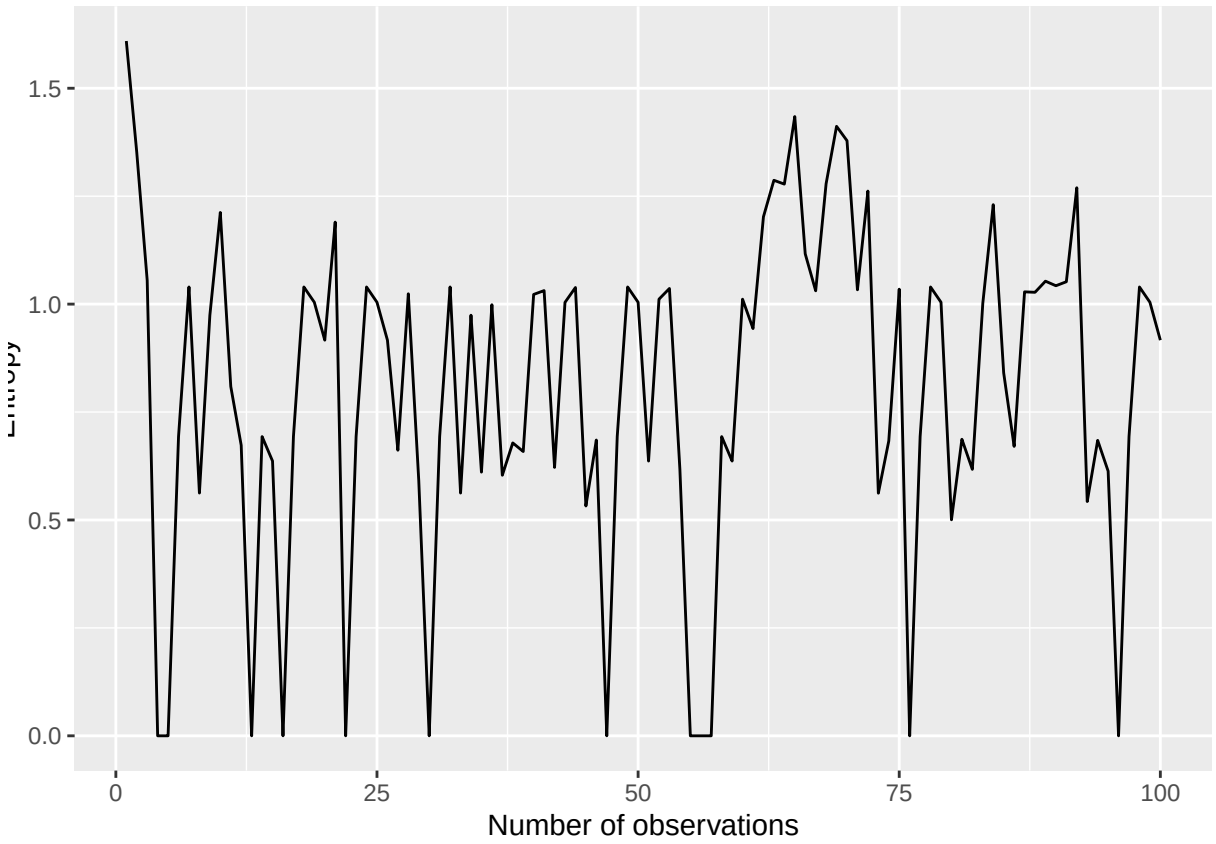
```
library(entropy)

entropies = c()

for (i in 1:ncol(filtered))
{
  entropy_i = entropy.empirical(filtered[, i])
  entropies = c(entropies, entropy_i)
}

p = ggplot() +
    geom_line(aes(x=1:ncol(filtered), y=entropies)) +
    labs(x="Number of observations", y="Entropy")

print(p)
```



Task 7

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

Below are the probabilities of the hidden states for the time step 101. The most probable state is $X10$

```
print(P %*% filtered[, 100])
```

```
##              [,1]
##  [1,] 0.00000000
##  [2,] 0.00000000
##  [3,] 0.00000000
##  [4,] 0.00000000
##  [5,] 0.27272727
##  [6,] 0.45454545
##  [7,] 0.22727273
##  [8,] 0.04545455
##  [9,] 0.00000000
## [10,] 0.00000000
```