TDDE15-Lab 2

frera064, oscho091, hjaoh082, olosw720

Contribution

All four members solved the problems on their own. We then discussed the answers and compiled the group report by combining our solutions. We used oscho091s solution for task1 and task2, olosw720s solution for task3 and task5, hjaoh082s solution for task4 and frera064s solution for task6 and task7.

1: Construct the hidden markov model

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

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

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

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

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

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

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

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

```r
# Initialize the Hidden Markov Model
hmm <- initHMM(
  States = states,           # vector of states
  Symbols = symbols,         # vector of observation symbols
  startProbs = start_probs,  # Initial state probabilities
  transProbs = trans_probs,  # Transition probabilities matrix
  emissionProbs = emission_probs  # Emission probabilities matrix
)
```

2: Simulate 100 timestreps

```r
# Q2
set.seed(12345)
simulation <- simHMM(hmm, length = 100)
```

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.

```r
nIter = 100
# using exp() to avoid -inf values that create
# NaN values in the probability distribution
alpha = exp(forward(hmm, simulation$observation))
beta = exp(backward(hmm, simulation$observation))

#filtered distribution
filteredProbs = matrix(0, nrow=10, ncol=nIter)
for (t in 1:nIter) {
  filteredProbs[, t] = alpha[,t] / sum(alpha[,t])
}

#smoothed probability distribution
smoothedProbs = matrix(0, nrow=10, ncol=nIter)
for (t in 1:nIter) {
  smoothedProbs[, t] = (alpha*beta)[,t] / sum((alpha*beta)[,t])
}

#most probable path via viterbi
probablePath = viterbi(hmm, simulation$observation)
```

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.

```r
predict_filtered = apply(filteredProbs, MARGIN = 2, FUN = which.max)
predict_smoothed = apply(smoothedProbs, MARGIN = 2, FUN = which.max)

calculate_accuracy = function(predicted_values, true_values) {
  cm = table(predicted_values, true_values)
  correct_predictions = sum(diag(cm))
  total_predictions = sum(cm)
  accuracy = correct_predictions / total_predictions
```

```
  return(accuracy)
}

cat("Filtered: ", calculate_accuracy(predict_filtered, simulation$states))
```

```
## Filtered:  0.53
```

```
cat(", Smoothed: ", calculate_accuracy(predict_smoothed, simulation$states))
```

```
## , Smoothed:  0.74
```

```
cat(", Viterbi: ", calculate_accuracy(probablePath, simulation$states))
```

```
## , Viterbi:  0.56
```

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?

```
set.seed(123)
nIter = 100
simulation = simHMM(hmm, nIter)
alpha = exp(forward(hmm, simulation$observation))
beta = exp(backward(hmm, simulation$observation))

#filtered distribution
filteredProbs = matrix(0, nrow=10, ncol=nIter)
for (t in 1:nIter) {
  filteredProbs[, t] = alpha[,t] / sum(alpha[,t])
}

smoothedProbs = matrix(0, nrow=10, ncol=nIter)
for (t in 1:nIter) {
  smoothedProbs[, t] = (alpha*beta)[,t] / sum((alpha*beta)[,t])
}

#most probable path via viterbi
probablePath = viterbi(hmm, simulation$observation)

filteredStates = apply(filteredProbs, 2, which.max)
smoothedStates = apply(smoothedProbs, 2, which.max)

filteredConfustionMatrix = table(filteredStates, simulation$states)
smoothedConfustionMatrix = table(smoothedStates, simulation$states)
viterbiConfusionMatrix = table(probablePath, simulation$states)

accuracyF = mean(filteredStates == simulation$states)*100
accuracyS = sum(diag(smoothedConfustionMatrix))/sum(smoothedConfustionMatrix)*100
accuracyV = sum(diag(viterbiConfusionMatrix))/sum(viterbiConfusionMatrix)*100

print(paste("The filtered accuracy is ",as.character(accuracyF),"%"))
```

```
## [1] "The filtered accuracy is  53 %"
```

```
#print(filteredConfustionMatrix)
print(paste("The smoothed accuracy is ",as.character(accuracyS),"%"))
```

```
## [1] "The smoothed accuracy is  64 %"
```

```
#print(smoothedConfustionMatrix)
print(paste("The viterbi accuracy is ",as.character(accuracyV),"%"))
```

```
## [1] "The viterbi accuracy is  36 %"
```

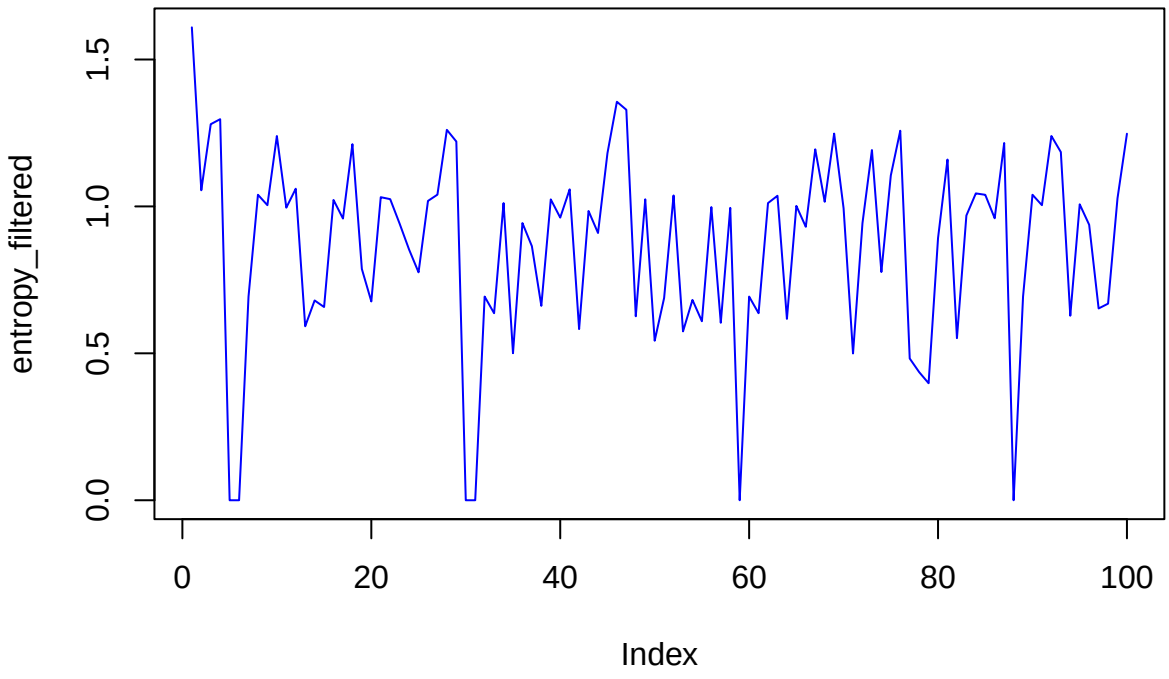
```
#print(viterbiConfusionMatrix)
```

Smoothed distributions are generally more accurate because they leverage more information (both past and future observations). This is supported by the average accuracies here.

The Viterbi path is less accurate than smoothed distributions because it commits to a single sequence of states, which don't account for the uncertainty in the observations.

6: 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?

```
entropy_filtered <- apply(filteredProbs, MARGIN = 2, FUN = entropy.empirical)
plot(entropy_filtered, type = "l", col= "blue")
```



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

```
prob101 <- hmm$transProbs%*%filteredProbs[,100]
prob101
```

```
##
## from       [,1]
##   1  0.00000000
##   2  0.04883721
##   3  0.22267442
##   4  0.37500000
##   5  0.27732558
##   6  0.07616279
##   7  0.00000000
##   8  0.00000000
##   9  0.00000000
##   10 0.00000000
```