Hjalmar_Öhman

2024-09-16

1

```r
set.seed(1)
library(HMM)

states = 1:10
symbols = 1:10
start_probs = rep(0.1, 10)

# Define the transition probabilities
trans_probs = matrix(0, nrow=10, ncol=10)
for (i in 1:9) {
  trans_probs[i, i] = 0.5  # Stay in the same sector
  trans_probs[i, i+1] = 0.5  # Move to the next sector
}
# Sector 10 transitions to Sector 1
trans_probs[10, 10] = 0.5
trans_probs[10, 1] = 0.5

# Define the emission probabilities
emission_probs = matrix(0, nrow=10, ncol=10)
for (i in 1:10) {
  # Get the sectors in the range [i-2, i+2]
  sectors = c((i-2):(i+2)) %% 10 # (%%10 transforms -1 to 9)
  sectors[sectors == 0] = 10  # (10%%10 transfroms 10 to 0, which is incorrect.)

  emission_probs[i, sectors] = 1/5  # Equal probability for the 5 neighboring sectors
}

hmm_model = initHMM(States=states, Symbols=symbols, startProbs=start_probs,
                    transProbs=trans_probs, emissionProbs=emission_probs)
print(hmm_model)
```

```
## $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
```

2

```
rm(list = setdiff(ls(), c("hmm_model", "trans_probs")))
simres = simHMM(hmm_model, 100)
```

3

```
simobv = simres$observation
alpha = exp(forward(hmm_model, simobv))
beta = exp(backward(hmm_model, simobv))

## Filtered
filtered = matrix(0,10, length(simobv))
for (i in 1:length(simobv)) {
  filtered[,i] = alpha[,i] / sum(alpha[,i])
}

## Smoothed
smoothed = matrix(0,10, length(simobv))
for (i in 1:length(simobv)) {
  smoothed[,i] = (alpha[,i]*beta[,i]) / sum(alpha[,i]*beta[,i])
}
```

```
## Viterbi
viterbi = viterbi(hmm_model, simobv)
```

4

```
predict_filtered = apply(filtered, MARGIN = 2, FUN = which.max)
predict_smoothed = apply(smoothed, 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, simres$states))
```

```
## Filtered:  0.56
```

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

```
## , Smoothed:  0.72
```

```
cat(", Viterbi: ", calculate_accuracy(viterbi, simres$states))
```

```
## , Viterbi:  0.51
```

5

```
rm(list = setdiff(ls(), c("hmm_model", "trans_probs")))
set.seed(12345)
simres = simHMM(hmm_model, 100)

simobv = simres$observation
alpha = exp(forward(hmm_model, simobv))
beta = exp(backward(hmm_model, simobv))

## Filtered
filtered = matrix(0,10, length(simobv))
for (i in 1:length(simobv)) {
  filtered[,i] = alpha[,i] / sum(alpha[,i])
}

## Smoothed
smoothed = matrix(0,10, length(simobv))
```

```r
for (i in 1:length(simobv)) {
  smoothed[,i] = (alpha[,i]*beta[,i]) / sum(alpha[,i]*beta[,i])
}

## Viterbi
viterbi = viterbi(hmm_model, simobv)

predict_filtered = apply(filtered, MARGIN = 2, FUN = which.max)
predict_smoothed = apply(smoothed, 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, simres$states))
```

```
## Filtered:  0.53
```

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

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

```r
cat(", Viterbi: ", calculate_accuracy(viterbi, simres$states))
```

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

Smoothed takes into account both past and future observation, while filtered only take into account past. Smoothed takes into account more observations than filtered.

Smoothed is more accurate than Viterbi, due to Viterbi having the constraint of predicting a valid path. Viterbi might miss important alternative states that are highly probable, but not part of the most likely path.

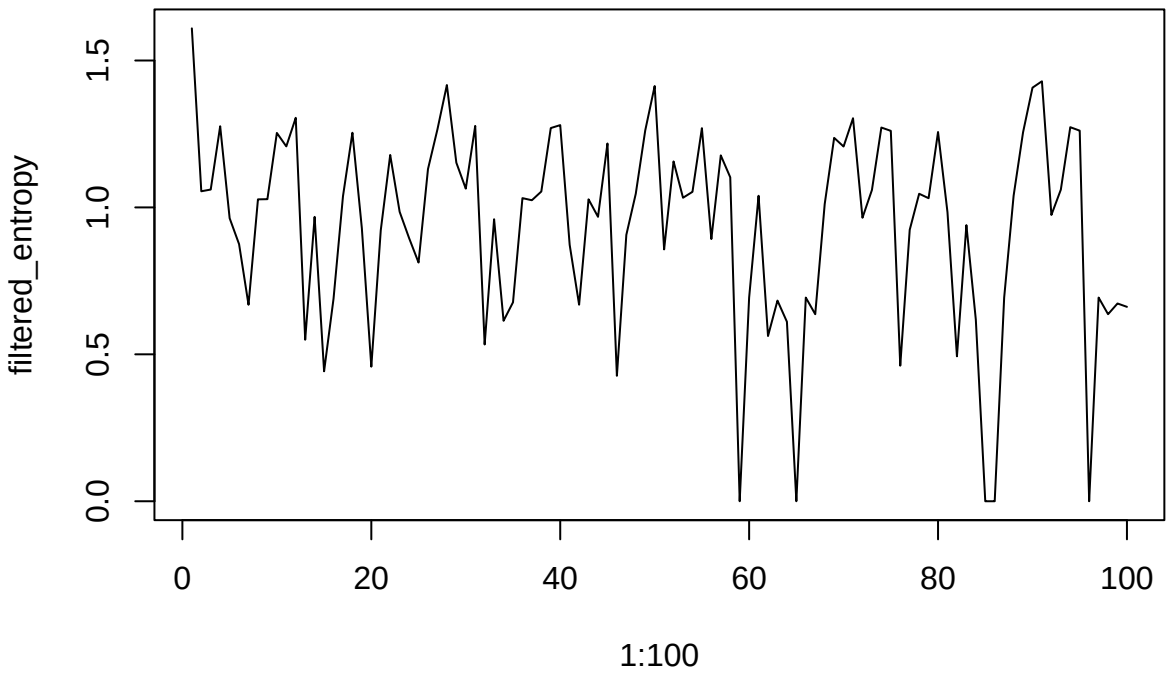
6

```r
library(entropy)

# Function to compute entropy for each time step
compute_entropy = function(filtered_distribution) {
  entropies = apply(filtered_distribution, 2, entropy.empirical)
  return(entropies)
}

filtered_entropy = compute_entropy(filtered)

plot(1:100, filtered_entropy, type="l")
```



It is not true that the later in time the better you know where the robot is. The entropy doesn't seem to be affected by the amount of observations, as seen in the graph.

7

```
filtered_100 = filtered[, 100]
predicted_101 = trans_probs %*% filtered_100
print(predicted_101)
```

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