Lab 2

Daniel Kouznetsov

2022-09-16

Assignment 1

Below code shows how I built a HMM for the scenario described.

```r
library(HMM)
sectors = c(1:10)
startProbs = c(.1, .1, .1, .1, .1, .1, .1, .1, .1, .1)
initTransMatrix = matrix(0, nrow=10, ncol=10)

setTransProbs <- function(transMatrix){
  for (row in 1:(length(transMatrix[1,]))) {
    transMatrix[row, row] = .5
    if (row < length(transMatrix[1,])){
      transMatrix[row, row+1] = .5
    }
    else{
      transMatrix[row, 1] = .5
    }
  }
  return(transMatrix)
}

transProbs = setTransProbs(initTransMatrix)
initEmissionMatrix = matrix(0, nrow=10, ncol=10)

setEmissionProbs = function(emissionMatrix){
  for(r in 1:10) {
    for (c in 1:10) {
      if((c+7-r) %% 10 >= 5) {
        emissionMatrix[r,c] = 0.2
      } else {
        emissionMatrix[r,c] = 0
      }
    }
  }
  return(emissionMatrix)
}

emissionProbs = setEmissionProbs(initEmissionMatrix)
rm(initEmissionMatrix, initTransMatrix) # deleting unused data
hmm = initHMM(sectors, sectors, startProbs, transProbs, emissionProbs)
```

HMM generated:

```
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
```

Assignment 2

Simulating HMM for 100 time steps.

```
set.seed(1)
simulated = simHMM(hmm, 100)
```

Assignment 3

Filtered and smoothed probability distributions and also the most probable path computed.

```r
simulated.observations = simulated$observation
simulated.states = simulated$states
rm(simulated) # remove unused data

forwardProbs = exp(forward(hmm, simulated.observations))
backwardProbs = exp(backward(hmm, simulated.observations))

filtering = forwardProbs / colSums(forwardProbs)
filtering.table = prop.table(filtering, margin=2)

smoothing = (forwardProbs * backwardProbs) / (colSums(forwardProbs) * colSums(backwardProbs))
smoothing.table = prop.table(smoothing, margin=2)

viterbi.path = viterbi(hmm, simulated.observations)
```

Assignment 4

Code for calculating the accuracy of a guessed path to the true path.

```r
smoothing.path = apply(smoothing.table, 2, which.max)
filtering.path = apply(filtering.table, 2, which.max)

cmp_paths = function(path) {
  bool.values = (path == simulated.states)
  correct.guesses = 0
  for (i in 1:length(bool.values)){
    if (bool.values[i]){
      correct.guesses = correct.guesses + 1
    }
  }
  percentage.correct = correct.guesses / length(bool.values)
  return(percentage.correct)
}

filtering.percentage = cmp_paths(filtering.path)
smoothing.percentage = cmp_paths(smoothing.path)
viterbi.percentage = cmp_paths(viterbi.path)
```

Results:

```r
filtering.percentage
```

```
## [1] 0.46
```

```r
smoothing.percentage
```

```
## [1] 0.68
```

```r
viterbi.percentage
```

```
## [1] 0.51
```

Assignment 5

I used the same code as above but changed the value for set.seed().

New values:

Seed Value	Filtering	Smoothing	Viterbi
20	0.27	0.5	0.52
50	0.45	0.64	0.45
150	0.34	0.61	0.42
200	0.26	0.51	0.37

In most of the cases smoothing gives us the best results, followed by the Viterbi algorithm (most probable path) and last filtering. Smoothing gives better results over filtering because smoothing looks at both past and future results while filtering only looks at the past. Why the Viterbi algorithm gives us worse results than smoothing in most cases is because the Viterbi algorithm produces a most likely sequence instead of looking at the current states when predicting the next state.

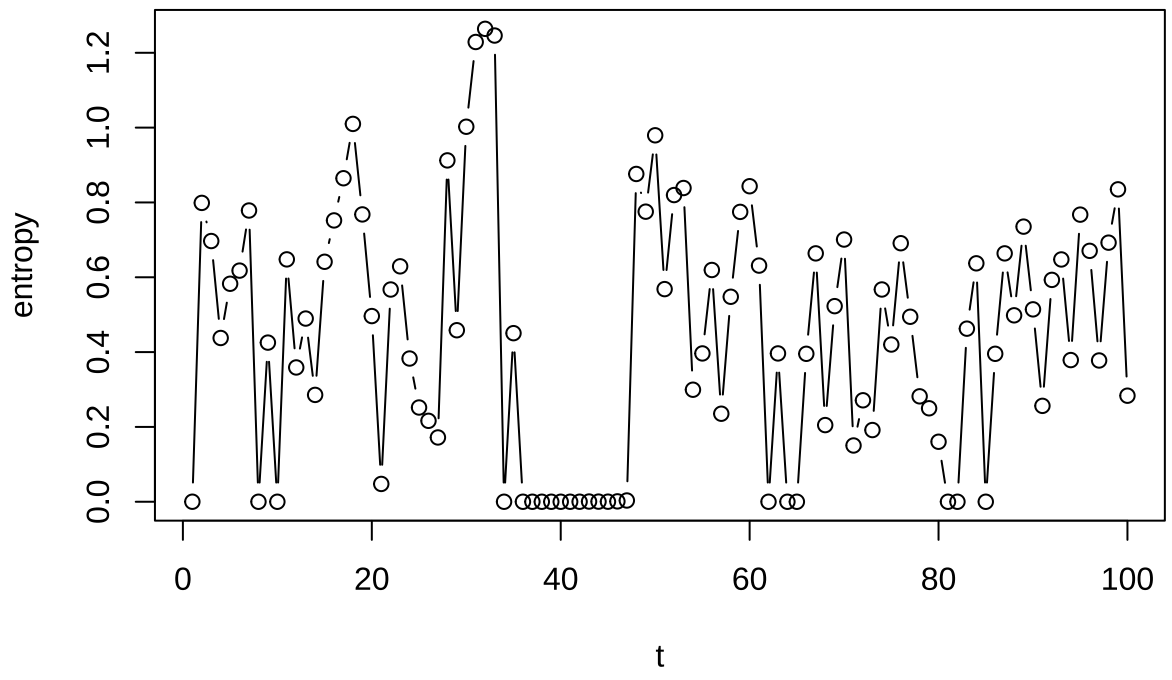
Assignment 6

The amount of uncertainty in our filtering distribution can be quantified using the Shannon entropy via the entropy library. With a higher entropy value (1) we have a lot of uncertainty, a lower entropy value gives us less uncertainty.

```r
library(entropy)

entropy.vector = c()
for (i in 1:100){
  entropy.vector = c(entropy.vector, entropy.empirical(filtering.table[,i]))
}

plot(1:100, entropy.vector, type="b", xlab="t", ylab="entropy")
```



The plot shows that entropy oscillates between 0 and 1 for all values despite our timepoint, t, increasing. This means that it does not matter how many observations we have. We see something weird between values ~35-45, entropy value goes down to 0 for many values in a row, I have no idea why that is though.

Assignment 7

Probabilities for timepoint t.

```
hundredone = filtering.table[,100] %*% hmm$transProbs
hundredone
```

```
##       to
##                  1   2         3 4 5 6 7 8 9 10
##   [1,] 0.04096286 0.5 0.4590371 0 0 0 0 0 0  0
```