lab2

Olof Swedberg

2024-09-14

1 Hidden Markov Model

```
set.seed(12345)
library(HMM)

states = 1:10
emissionSymbols = 1:10

transitionProb = matrix(0,nrow = 10, ncol = 10)
for (j in 1:10) {
  transitionProb[j,j] = 0.5
  transitionProb[j,j%%10 +1 ] = 0.5
}
print(transitionProb)
```

```
##       [,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
```

```
emissionProb = matrix(0,nrow = 10, ncol = 10)
for (j in 1:10) {
  for (i in 1:5) {
    emissionProb[(j+i-4)%%10+1,j] = 0.2
  }
}

print(emissionProb)
```

```
##       [,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
```

```
startProb = rep(0.1,10)

hmm = initHMM(States = states, Symbols = emissionSymbols,
              startProbs = startProb, transProbs = transitionProb,
              emissionProbs = emissionProb)
```

2 Simulate 100 time steps

```
nIter = 100
simulation = simHMM(hmm, nIter)
print(simulation)
```

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

3 Filtering, smoothing and most probable path

```
# 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])
}
print(filteredProbs[,100])
```

```
##  [1] 0.000 0.375 0.625 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
#smoothed probability distribution
smoothedProbs = matrix(0, nrow=10, ncol=nIter)
for (t in 1:nIter) {
  smoothedProbs[, t] = (alpha*beta)[,t] / sum((alpha*beta)[,t])
}
print(smoothedProbs[,100])
```

```
##  [1] 0.000 0.375 0.625 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

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

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

4 Accuracy of the filtered, smoothed and probable path probability distributions

```
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  74 %"
```

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

```
## [1] "The viterbi accuracy is  56 %"
```

```
#print(viterbiConfusionMatrix)
```

5 Repetition of samples and discussion

```
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])
}
```

```r
#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  46 %"
```

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

```
## [1] "The smoothed accuracy is  68 %"
```

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

```
## [1] "The viterbi accuracy is  61 %"
```

```r
#print(viterbiConfusionMatrix)
```

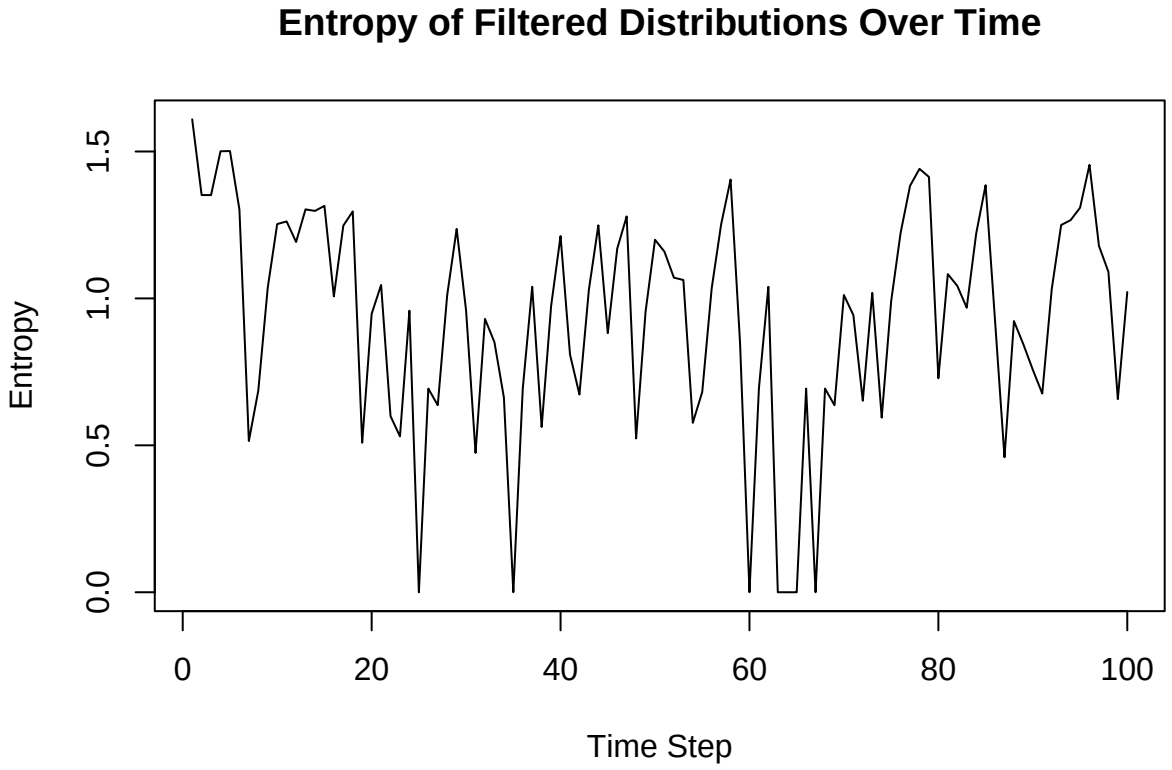
The filtered distribution use only the current and the past observation to estimate the current hidden state probability distribution, while the smoothed distribution uses all observations, both current and past. Therefore, the accuracy should be higher on the smoothed.

Furthermore, the smoothed distribution should overperform the most probable path distribution, as the viterbi algorithm does not take into account the uncertainty in each individual state at each time step. The viterbi path may ignore some important probability mass in alternative paths that are very close in likelihood. Smoothing provides the complete probability distribution for each state, reflecting the uncertainty and potentially being more accurate at each step.

6 Entropy of the filtered distributions

```r
library(entropy)

entropyVals = rep(0,nIter)
for (t in 1:nIter) {
  entropyVals[t] = entropy.empirical(filteredProbs[,t])
}
plot(1:nIter, entropyVals, type="l", main="Entropy of Filtered Distributions Over Time",
     xlab="Time Step", ylab="Entropy")
```



Since there is ambiguity in the observations, more observations will not necessarily improve accuracy. If the observations were of high quality, it is likely that more observations would lower the entropy. In the plot, it does not seem to increase or decrease, however there are some cases where the entrpoy is 0, indicating that the probability distribution is concentrated on a single state.

7 Probabilities of the hidden states for the time step 101

```
timeStep101=filteredProbs[,100]%*%hmm$transProbs
print(timeStep101)
```

```
##       to
##                1          2 3 4 5 6 7 8         9        10
##   [1,] 0.3417373 0.09173732 0 0 0 0 0 0 0.1582627 0.4082627
```