Lab2

Christian von Koch

2020-09-21

Assignment 1

```
set.seed(12345)
library(HMM)
library(entropy)
transitionMatrix=diag(0.5, 10)
diag(transitionMatrix[,-1])=0.5
transitionMatrix[10,1]=0.5
emissionMatrix=matrix(0,10,10)
for(i in 1:10) {
  for (j in 1:10) {
    if((j+7-i) %% 10 >= 5) {
      emissionMatrix[i,j]=0.2
    } else {
      emissionMatrix[i,j]=0
    }
  }
}
emissionMatrix
```

```
##       [,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
```

```
states=1:10
symbols=1:10
HMM_model=initHMM(States=states, Symbols=symbols, transProbs=transitionMatrix, emissionProbs=emissionMa
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
```

Assignment 2

```
simulation=simHMM(HMM_model, length=100)
```

Assignment 3

```
obsStates=simulation$observation
alpha=exp(forward(HMM_model, obsStates))
beta=exp(backward(HMM_model, obsStates))
calcFiltering = function(HMM, alpha, noSim) {
  filtered = matrix(0,10,noSim)
  for (i in 1:noSim) {
    filtered[,i]=alpha[,i]/sum(alpha[,i])
  }
  return(filtered)
}

calcSmoothing = function(HMM, alpha, beta, noSim) {
  smoothed=matrix(0,10,noSim)
```

```
  alphabeta=alpha*beta
  for (i in 1:noSim) {
    smoothed[,i]=alphabeta[,i]/sum(alphabeta[,i])
  }
  return(smoothed)
}

filtered=calcFiltering(HMM_model, alpha, 100)
smoothed=calcSmoothing(HMM_model, alpha, beta, 100)
posterior=posterior(HMM_model, obsStates)
mostProbPath=viterbi(HMM_model, obsStates)
mostProbPath
```

```
##   [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
```

As shown above the most probable path is shown.

Assignment 4

```
mostProb_filtered=apply(filtered, 2, which.max)
mostProb_smoothed=apply(smoothed, 2, which.max)

calcAccuracy = function(trueState, calcState) {
  n=length(trueState)
  return(sum(trueState==calcState)/n)
}

states=simulation$states
acc_mostProb=calcAccuracy(states, mostProbPath)
acc_filtered=calcAccuracy(states, mostProb_filtered)
acc_smoothed=calcAccuracy(states, mostProb_smoothed)
acc_mostProb
```

```
## [1] 0.56
```

```
acc_filtered
```

```
## [1] 0.53
```

```
acc_smoothed
```

```
## [1] 0.74
```

It is evident from the results obtained above that the smoothed distribution yielded the highest accuracy.

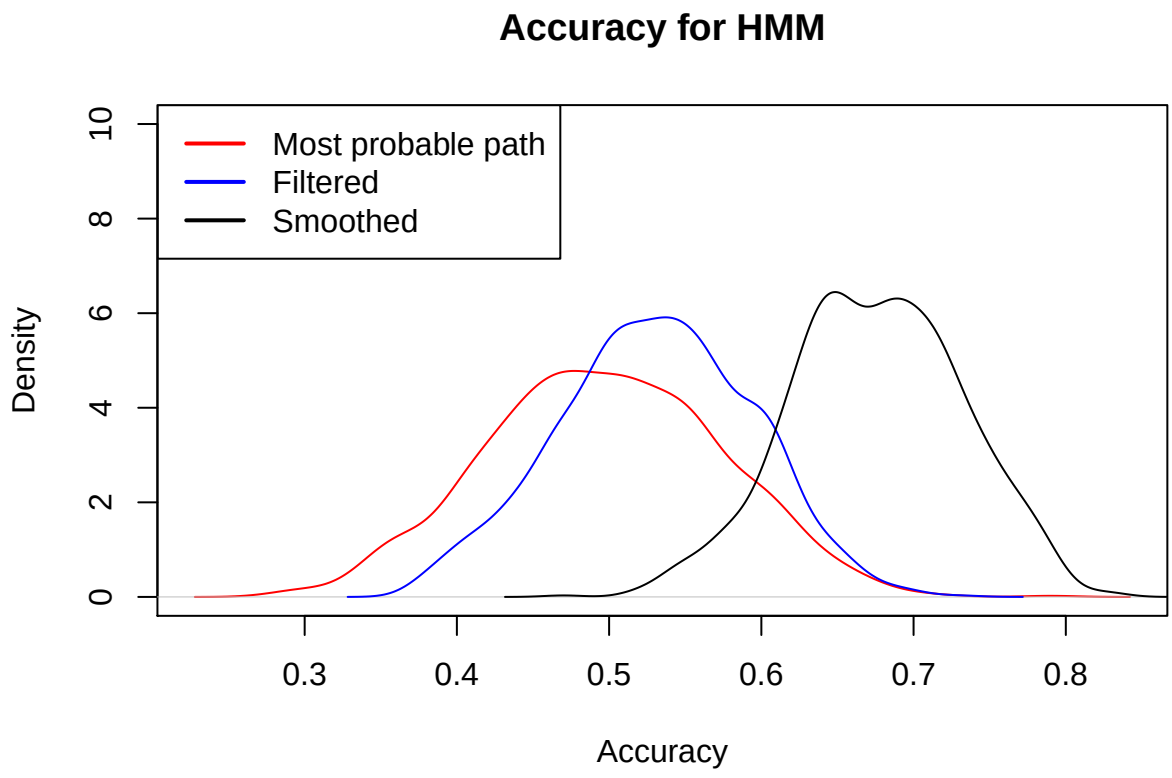
Assignment 5

```
simulationHMM=function(HMM_model, noSim) {
  simulation=simHMM(HMM_model, length=noSim)
  obsStates=simulation$observation
  states=simulation$states
  alpha=exp(forward(HMM_model, obsStates))
```

```
  beta=exp(backward(HMM_model, obsStates))
  filtered=calcFiltering(HMM_model, alpha, noSim)
  smoothed=calcSmoothing(HMM_model, alpha, beta, noSim)
  mostProbPath=viterbi(HMM_model, obsStates)
  mostProb_filtered=apply(filtered, 2, which.max)
  mostProb_smoothed=apply(smoothed, 2, which.max)
  acc_mostProb=calcAccuracy(states, mostProbPath)
  acc_filtered=calcAccuracy(states, mostProb_filtered)
  acc_smoothed=calcAccuracy(states, mostProb_smoothed)
  accVec=c(acc_mostProb, acc_filtered, acc_smoothed)
  return(matrix(accVec, 1, 3))
}

accMatrix=matrix(0,1000, 3)
for (i in 1:1000) {
  accMatrix[i,]=simulationHMM(HMM_model, 100)
}

plot(density(accMatrix[,1]), col="red", xlab="Accuracy", ylim=c(0,10), main="Accuracy for HMM")
lines(density(accMatrix[,2]), col="blue")
lines(density(accMatrix[,3]), col="black")
legend("topleft", box.lty = 1, legend = c("Most probable path","Filtered","Smoothed"),
       col=c("red","blue","black"), lwd = 2)
```



The smoothed distributions are more accurate than the filtered distributions since the smoothed distributions consider all of the data available whereas the filtered distributions only consider data up to a certain time

step. The smoothed distributions should also be more accurate than the distributions of the most probable path as well since the smoothed distributions don't have to satisfy the condition of a valid path which the most probable path obtained from the viterbi algorithm needs to.

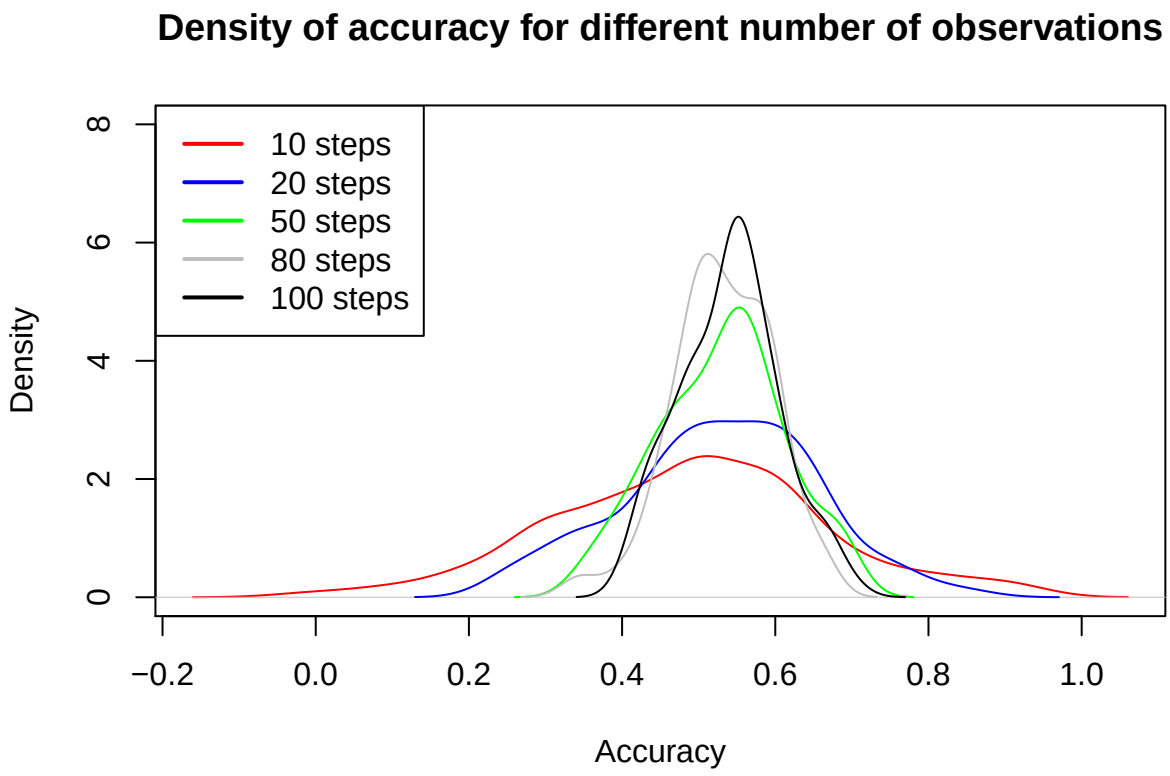
Assignment 6

```
noObs=c(10,20,50,80,100)
filteredMatrix=matrix(0,100,length(noObs))
entropyVec=rep(0,5)
counter=1
for (i in noObs) {
  simulation=simHMM(HMM_model, length=i)
  obsStates=simulation$observation
  states=simulation$states
  alpha=exp(forward(HMM_model, obsStates))
  beta=exp(backward(HMM_model, obsStates))
  filtered=calcFiltering(HMM_model, alpha, i)
  mostProb_filtered=apply(filtered, 2, which.max)
  entropyVec[counter]=entropy.empirical(mostProb_filtered)
  for (j in 1:100) {
    filteredMatrix[j,counter]=simulationHMM(HMM_model, i)[1,2]
  }
  counter=counter+1
}

entropyVec
```

```
## [1] 2.037647 2.768803 3.762932 4.236894 4.426092
```

```
plot(density(filteredMatrix[,1]), ylim=c(0,8), col="red", xlab="Accuracy",
     main="Density of accuracy for different number of observations")
lines(density(filteredMatrix[,2]), col="blue")
lines(density(filteredMatrix[,3]), col="green")
lines(density(filteredMatrix[,4]), col="grey")
lines(density(filteredMatrix[,5]), col="black")
legend("topleft", box.lty = 1, legend = c("10 steps","20 steps","50 steps", "80 steps", "100 steps"),
       col=c("red","blue","green", "grey", "black"), lwd = 2)
```



It is true that you can be more sure where the robot is since the distribution of the accuracy gets denser when more observations are available. This means that when running the simulation many times you can be more sure that the accuracy will be more concentrated for the filtered distribution with more observations than with less.

Assignment 7

```
timestep101=filtered[,100]%*%transitionMatrix
timestep101
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]      [,7] [,8]      [,9] [,10]
## [1,]    0    0    0    0    0    0 0.3053973  0.5 0.1946027     0
```

The probability for the robot to be in a specific sector of the ring is shown above.

Appendix

```r
## The purpose of the lab is to put in practice some of the concepts covered in the lectures.
## To do so, you are asked to model the behavior of a robot that walks around a ring. The ring is
## divided into 10 sectors. At any given 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 though: If the robot is in the sector i, then the device
## will report that the robot is in the sectors [i-2, i+2] with equal probability.

## 1.  Build a hidden Markov model (HMM) for the scenario described above.

#install.packages("HMM")
#install.packages("entropy")
set.seed(12345)
library(HMM)
library(entropy)
transitionMatrix=diag(0.5, 10)
diag(transitionMatrix[,-1])=0.5
transitionMatrix[10,1]=0.5
emissionMatrix=matrix(0,10,10)
for(i in 1:10) {
  for (j in 1:10) {
    if((j+7-i) %% 10 >= 5) {
      emissionMatrix[i,j]=0.2
    } else {
      emissionMatrix[i,j]=0
    }
  }
}
emissionMatrix
states=1:10
symbols=1:10
HMM_model=initHMM(States=states, Symbols=symbols, transProbs=transitionMatrix, emissionProbs=emissionMa

## 2. Simulate the HMM for 100 time steps.

simulation=simHMM(HMM_model, length=100)

## 3.  Discard the hidden states from the sample obtained above. Use the remaining observations to comp
## and smoothed probability distributions for each of the 100 time points. Compute also the most probab

obsStates=simulation$observation
alpha=exp(forward(HMM_model, obsStates))
beta=exp(backward(HMM_model, obsStates))
calcFiltering = function(HMM, alpha, noSim) {
  filtered = matrix(0,10,noSim)
  for (i in 1:noSim) {
    filtered[,i]=alpha[,i]/sum(alpha[,i])
  }
  return(filtered)
}

calcSmoothing = function(HMM, alpha, beta, noSim) {
```

```
  smoothed=matrix(0,10,noSim)
  alphabeta=alpha*beta
  for (i in 1:noSim) {
    smoothed[,i]=alphabeta[,i]/sum(alphabeta[,i])
  }
  return(smoothed)
}

filtered=calcFiltering(HMM_model, alpha, 100)
smoothed=calcSmoothing(HMM_model, alpha, beta, 100)
posterior=posterior(HMM_model, obsStates)
mostProbPath=viterbi(HMM_model, obsStates)

## 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.

mostProb_filtered=apply(filtered, 2, which.max)
mostProb_smoothed=apply(smoothed, 2, which.max)

calcAccuracy = function(trueState, calcState) {
  n=length(trueState)
  return(sum(trueState==calcState)/n)
}

states=simulation$states
acc_mostProb=calcAccuracy(states, mostProbPath)
acc_filtered=calcAccuracy(states, mostProb_filtered)
acc_smoothed=calcAccuracy(states, mostProb_smoothed)

## 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 ?


simulationHMM=function(HMM_model, noSim) {
  simulation=simHMM(HMM_model, length=noSim)
  obsStates=simulation$observation
  states=simulation$states
  alpha=exp(forward(HMM_model, obsStates))
  beta=exp(backward(HMM_model, obsStates))
  filtered=calcFiltering(HMM_model, alpha, noSim)
  smoothed=calcSmoothing(HMM_model, alpha, beta, noSim)
  mostProbPath=viterbi(HMM_model, obsStates)
  mostProb_filtered=apply(filtered, 2, which.max)
  mostProb_smoothed=apply(smoothed, 2, which.max)
  acc_mostProb=calcAccuracy(states, mostProbPath)
  acc_filtered=calcAccuracy(states, mostProb_filtered)
  acc_smoothed=calcAccuracy(states, mostProb_smoothed)
  accVec=c(acc_mostProb, acc_filtered, acc_smoothed)
  return(matrix(accVec, 1, 3))
}
```

```r
accMatrix=matrix(0,1000, 3)
for (i in 1:1000) {
  accMatrix[i,]=simulationHMM(HMM_model, 100)
}

plot(density(accMatrix[,1]), col="red", xlab="Accuracy", ylim=c(0,10), main="Accuracy for HMM")
lines(density(accMatrix[,2]), col="blue")
lines(density(accMatrix[,3]), col="black")
legend("topleft", box.lty = 1, legend = c("Most probable path","Filtered","Smoothed"),
       col=c("red","blue","black"), lwd = 2)

## Smoothed distribution more accurate in general than both the filtered and the most probable path dis

## 6. Is it true that the more observations you have the better you know where the robot is ?

noObs=c(10,20,50,80,100)
filteredMatrix=matrix(0,100,length(noObs))
entropyVec=rep(0,5)
counter=1
for (i in noObs) {
  simulation=simHMM(HMM_model, length=i)
  obsStates=simulation$observation
  states=simulation$states
  alpha=exp(forward(HMM_model, obsStates))
  beta=exp(backward(HMM_model, obsStates))
  filtered=calcFiltering(HMM_model, alpha, i)
  mostProb_filtered=apply(filtered, 2, which.max)
  entropyVec[counter]=entropy.empirical(mostProb_filtered)
  for (j in 1:100) {
    filteredMatrix[j,counter]=simulationHMM(HMM_model, i)[1,2]
  }
  counter=counter+1
}

entropyVec
plot(density(filteredMatrix[,1]), ylim=c(0,8), col="red", xlab="Accuracy",
     main="Density of accuracy for different number of observations")
lines(density(filteredMatrix[,2]), col="blue")
lines(density(filteredMatrix[,3]), col="green")
lines(density(filteredMatrix[,4]), col="grey")
lines(density(filteredMatrix[,5]), col="black")
legend("topleft", box.lty = 1, legend = c("10 steps","20 steps","50 steps", "80 steps", "100 steps"),
       col=c("red","blue","green", "grey", "black"), lwd = 2)

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

timestep101=filtered[,100]%*%transitionMatrix
timestep101
```