David Bjorelind Lab 2

Lab 2: Hidden Markov Models

David Björelind, davbj395

This is my personal lab report for lab 2.

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

```r
state = rep(1:10)
probs = rep(0, 10)
probs[1] = 1 # Robot starts in position 1

emissionP = matrix(0, 10, 10)
transP = matrix(0, 10, 10)
funcmod = function(i){
  if(i == 10){
    return(i)
  } else if(i<=0){
    return(i+10)
  }else{
    return(i%%10)
  }
}
for (i in 1:10){
  transP[i,i] = 0.5
  transP[i, funcmod(i+1)] = 0.5
  for(j in -2:2){
    emissionP[i,funcmod(i+j)] = 0.2
  }
}

robot = initHMM(States = state, Symbols = state, startProbs = probs, transProbs = transP, emissionProbs
```

2) Simulate the HMM for 100 time steps

```r
set.seed(12345) # Go to same result every time
niter = 100
sim_data = simHMM(robot, niter)
sim_data
```

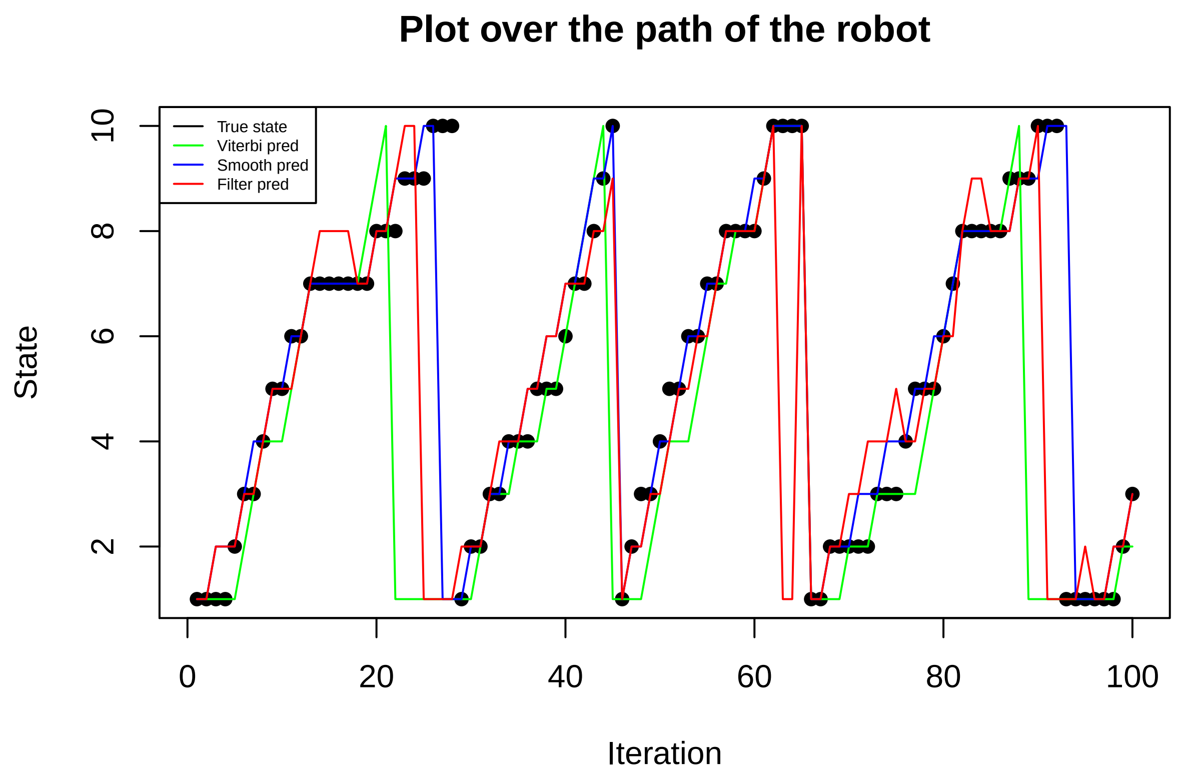
```
## $states
##   [1]  1  1  1  1  2  3  3  4  5  5  6  6  7  7  7  7  7  7  7  8  8  8  9  9  9
##  [26] 10 10 10  1  2  2  3  3  4  4  4  5  5  5  6  7  7  8  9 10  1  2  3  3  4
##  [51]  5  5  6  6  7  7  8  8  8  8  9 10 10 10 10  1  1  2  2  2  2  2  3  3  3
##  [76]  4  5  5  5  6  7  8  8  8  8  8  9  9  9 10 10 10  1  1  1  1  1  1  2  3
##
## $observation
##   [1]  2 10  3 10  1  3  5  6  3  3  4  7  9  7  9  6  6  5  8  7 10 10  9 10 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) 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
# Taking away the hidden states from the sample
obs = sim_data$observation
true = sim_data$states # The true states, used for comparisson later

alpha = exp(forward(robot, obs))
beta = exp(backward(robot, obs))
sum_alpha = apply(alpha, 2, sum)
sum_alphabeta = apply(alpha*beta, 2, sum)

# Filter probability distribution (using obs up the time t)
filter = t(apply(alpha, 1,"/", sum_alpha))

# Smoothed probability distribution (using all available obs)
smoothing = t(apply(alpha*beta, 1, "/", sum_alphabeta))

# Most probable path, using VITERBI algorithm
path = viterbi(robot, obs)
```

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
pred_smooth = apply(smoothing, MARGIN = 2, FUN=which.max)
pred_filter = apply(filter, MARGIN = 2, FUN=which.max)
acc_smooth = sum(true == pred_smooth)/100
acc_filter = sum(true == pred_filter)/100
acc_path = sum(true == path)/100

plot(true, main = "Plot over the path of the robot", xlab = "Iteration", ylab = "State", col = "black",
lines(path, col = "green")
lines(pred_smooth, col = "blue")
lines(pred_filter, col = "red")
legend("topleft", c("True state","Viterbi pred", "Smooth pred", "Filter pred"),
       col=c("black", "green", "blue", "red"), lty=1, cex=0.5)
```



```r
cat("Accuracy of Smoothing:", acc_smooth)
```

```
## Accuracy of Smoothing: 0.75
```

```r
cat("Accuracy of Filtering:", acc_filter)
```

```
## Accuracy of Filtering: 0.56
```

```r
cat("Accuracy of Viterbi algorithm:", acc_path)
```

```
## Accuracy of Viterbi algorithm: 0.57
```

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 ?

```r
samples = 5
for (i in 1:samples){
  sim_data = simHMM(robot, niter)

  obs = sim_data$observation
  true = sim_data$states # The true states, used for comparisson later
  alpha = exp(forward(robot, obs))
  beta = exp(backward(robot, obs))
  sum_alpha = apply(alpha, 2, sum)
  sum_alphabeta = apply(alpha*beta, 2, sum)
  filter = t(apply(alpha, 1,"/", sum_alpha))
  smoothing = t(apply(alpha*beta, 1, "/", sum_alphabeta))
  path = viterbi(robot, obs)
  pred_smooth = apply(smoothing, MARGIN = 2, FUN=which.max)
  pred_filter = apply(filter, MARGIN = 2, FUN=which.max)
  acc_smooth = sum(true == pred_smooth)/niter
  acc_filter = sum(true == pred_filter)/niter
  acc_path = sum(true == path)/niter

  cat("Accuracy of Smoothing, run ", i, ": ", acc_smooth, "\n")
  cat("Accuracy of Filtering, run ", i, ": ", acc_filter, "\n")
  cat("Accuracy of Viterbi algorithm, run ", i, ": ", acc_path, "\n" , "\n")
}
```

```
## Accuracy of Smoothing, run  1 :  0.65
## Accuracy of Filtering, run  1 :  0.44
## Accuracy of Viterbi algorithm, run  1 :  0.54
##
## Accuracy of Smoothing, run  2 :  0.79
## Accuracy of Filtering, run  2 :  0.47
## Accuracy of Viterbi algorithm, run  2 :  0.62
##
## Accuracy of Smoothing, run  3 :  0.68
## Accuracy of Filtering, run  3 :  0.56
## Accuracy of Viterbi algorithm, run  3 :  0.53
##
## Accuracy of Smoothing, run  4 :  0.78
## Accuracy of Filtering, run  4 :  0.62
## Accuracy of Viterbi algorithm, run  4 :  0.67
##
## Accuracy of Smoothing, run  5 :  0.64
## Accuracy of Filtering, run  5 :  0.53
## Accuracy of Viterbi algorithm, run  5 :  0.39
##
```

Repeating the simulation 5 times, the accuracy of each prediction is displayed above. The smoothed distribution takes all of the data into consideration, not only the data from time 0:t, like the filtering distribution does. Given that it is known what states are possible in the next time step, the smoothing distribution can make a slightly better guess on where the robot will be located.

The most probable path (from Viterbi algorithm) makes sure that path follows a proper route. This means not taking more than one step forward or going backwards. Because of this strong assumption, the most probable path can misclassify some states if the previous state is incorrect. Comparing Viterbi to the smoothed distribution, they both have the same available information. The difference being that Viterbi has an additional constraint.

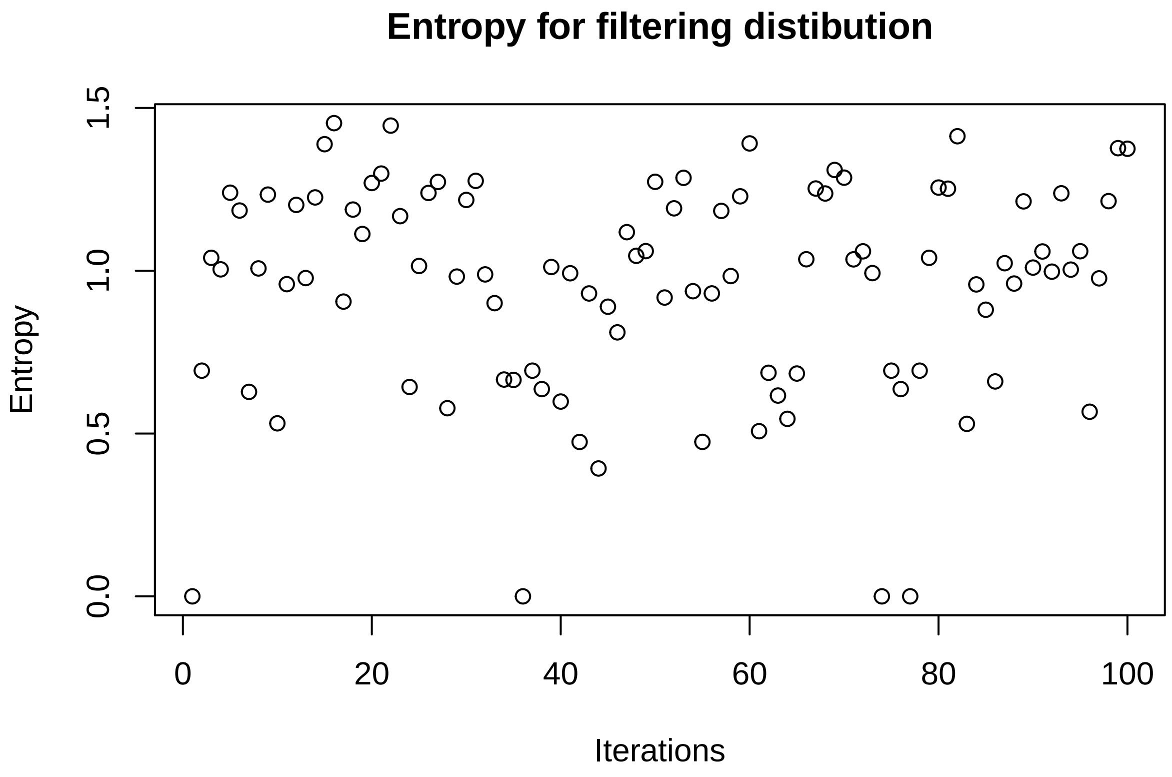
This can be seen in the graph in 4), where the filtering distribution moves back and forth a couple of time during the simulation.

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

```r
library(entropy) # Lower number means that we know better where the robot is
max = entropy.empirical(rep(0.1, 10)) # The largest possible entropy
cat("Maxmal value for entropy in this case is: ", max)
```

```
## Maxmal value for entropy in this case is:  2.302585
```

```r
ent_filter = apply(filter, 2, entropy.empirical)

plot(ent_filter, main = "Entropy for filtering distibution", xlab = "Iterations", ylab = "Entropy")
```

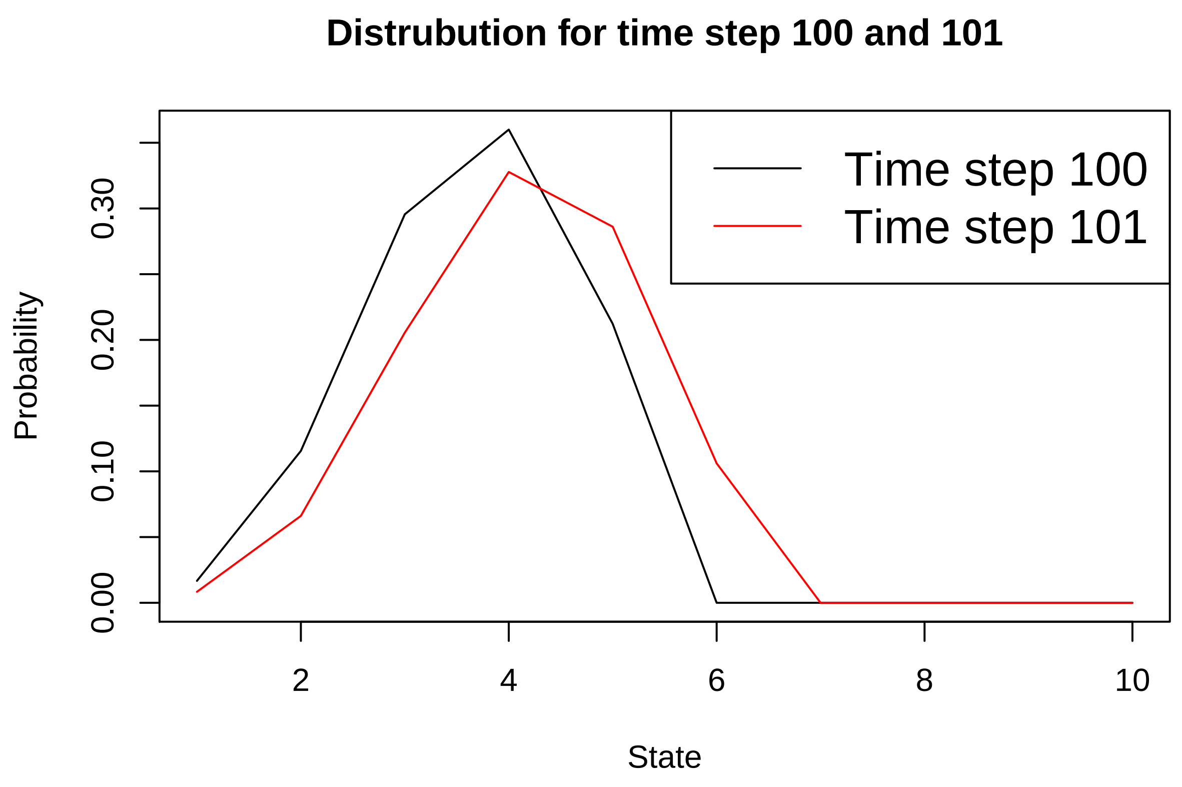


Entropy gives a level on "information" or "uncertainty" of a random variable. A lower entropy value means that the uncertainty is low, that we are sure of the variable. A value of 0 means that complete certainty of the variable. For evaluating if more observations improve robot placement prediction, it only makes sense to look at the **filtering** distribution, since each observation takes time 0:t into consideration.

From the graph above we can see that we **do not** make better predictions of where the robot is when using a larger time frame. However, we do have some 0's. This is probably due to the sequence previous to the observation and not with having large amount of data. An interesting observation is that the entropy array for **smoothing** contains more 0's than filtering, indicating that we do get additional information from knowing the future states of the robot.

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

```r
last_obs = filter[,100]
step_101 = last_obs%*%transP

plot(last_obs, type="line", main="Distrubution for time step 100 and 101", xlab="State", ylab="Probabil
lines(step_101[1,], col="red")
legend("topright", c("Time step 100", "Time step 101"),
       col=c("black", "red"), lty=1, cex=1.5)
```



Above is plotted distribution for last time step (in black) and predicted distribution for time step 101 (in red). Seems reasonable that time step 101 has moved to the right by about half a step, indicating the 50/50 chance for the robot to stay or move to the next state.