LAB2 HIDDEN MARKOV MODELS

Martynas Lukosevicius, Joel Nilsson, Alejo Perez Gomez

20/09/2021

Statement of Contribution

- code: Martynas Lukosevicius
- Answers to questions: Joel Nilsson, Alejo Perez Gomez, Martynas Lukosevicius

Question 1.

Build a hidden Markov model

```
library(HMM)
States <- c("S1","S2","S3","S4","S5","S6","S7","S8","S9", "S10")
positions <- c("P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10")
transProbs <- matrix(nrow = 10, ncol = 10)
emissionProbs <- matrix(nrow = 10, ncol = 10)

for (i in c(1:length(States))) {
  updateRows <- c(i:(i+1))
  updateRows[updateRows > length(States)] <- updateRows[updateRows >
                                                          length(States)] - length(States)
  updateRows[updateRows < 1] <-  length(States) + updateRows[updateRows < 1]
  transProbs[i,updateRows] <- rep(1/length(updateRows), length(updateRows))


  updateRows <- c((i-2):(i+2))
  updateRows[updateRows > length(positions)] <- updateRows[updateRows >
                                                             length(positions)] - length(positions)
  updateRows[updateRows < 1] <-  length(positions) + updateRows[updateRows < 1]
  emissionProbs[i,updateRows] <- rep(1/length(updateRows), length(updateRows))

}


HMM <- initHMM(States,
               positions,
               transProbs = transProbs,
               emissionProbs = emissionProbs)
```

Question 2

Simulate the HMM for 100 time steps

```r
simulatedData <- simHMM(HMM, 100)
```

Question 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
observations <- simulatedData$observation

# filtering
alphazt <- exp(forward(HMM, observations))
normalization <- colSums(alphazt)

filtering <- t(t(alphazt) / normalization)


# smoothing

betazt <- exp(backward(HMM, observations))
smoothing <- t(t(alphazt * betazt) / colSums(alphazt * betazt))

# viterbi

mostlikelypath <- viterbi(HMM, observations)
```

Question 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
extract_state <- function(vector){
  str_value <- names(which.max(vector))
  return(str_value)
}


filtering_predictions <- apply(filtering,2,extract_state)
smoothing_predictions <- apply(smoothing,2,extract_state)
filtering_acc <- sum(diag(table(filtering_predictions, simulatedData$states)))/
  length(filtering_predictions)
smoothing_acc <- sum(diag(table(smoothing_predictions, simulatedData$states)))/
  length(smoothing_predictions)
mostlikelypath_acc <- sum(diag(table(mostlikelypath, simulatedData$states)))/
  length(mostlikelypath)
```

Table 1: accuracies

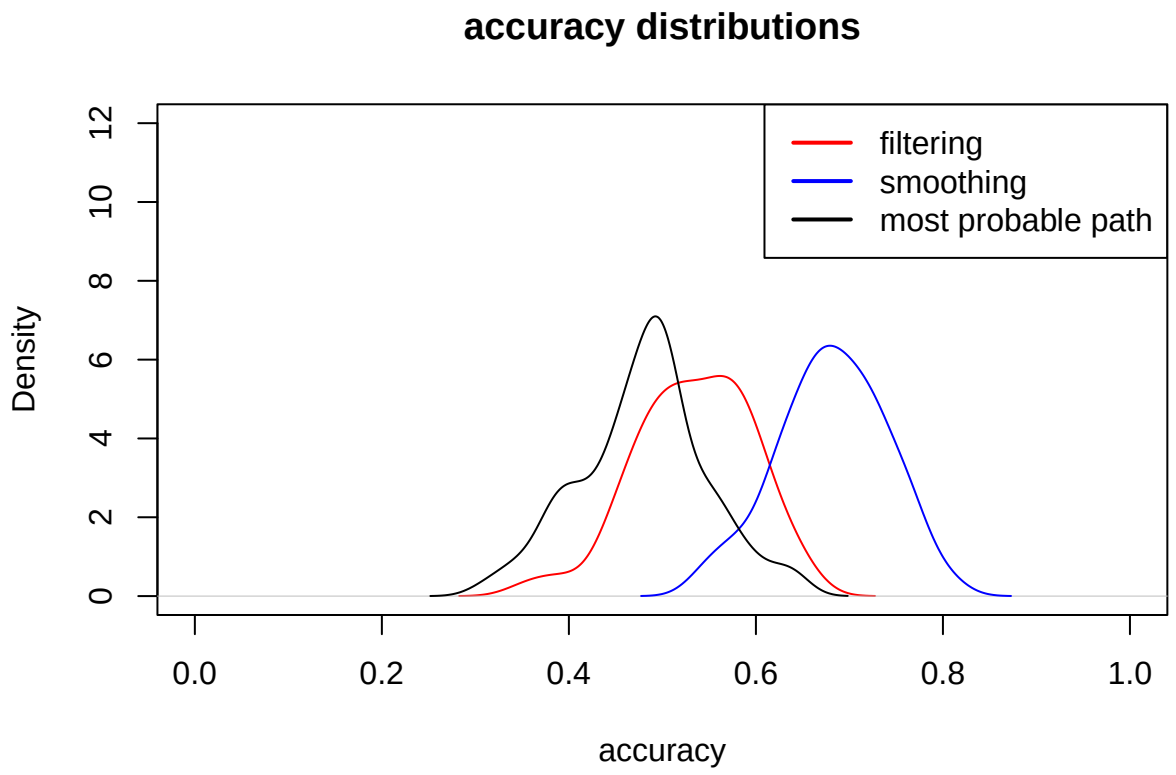
filtering	smoothing	most probable path
0.54	0.71	0.62

Question 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 ?

```
simualte_n <- function(n, HMM){
  filtering_acc <- vector(length = n)
  smoothing_acc <- vector(length = n)
  mostlikelypath_acc <- vector(length = n)

  for (i in 1:n) {
    simulatedData <- simHMM(HMM, 100)
    observations <- simulatedData$observation
    # filtering
    alphazt <- exp(forward(HMM, observations))
    normalization <- colSums(alphazt)

    filtering <- t(t(alphazt) / normalization)


    # smoothing

    betazt <- exp(backward(HMM, observations))
    smoothing <- t(t(alphazt * betazt) / colSums(alphazt * betazt))

    # viterbi

    mostlikelypath <- viterbi(HMM, observations)
    filtering_predictions <- apply(filtering,2,extract_state)
    smoothing_predictions <- apply(smoothing,2,extract_state)
    filtering_acc[i] <- sum(diag(table(filtering_predictions,
                                       simulatedData$states)))/
      length(filtering_predictions)
    smoothing_acc[i] <- sum(diag(table(smoothing_predictions,
                                       simulatedData$states)))/
      length(smoothing_predictions)
    mostlikelypath_acc[i] <- sum(diag(table(mostlikelypath,
                                            simulatedData$states)))/
      length(mostlikelypath)

  }

  return(list(filtering_acc = filtering_acc,
              smoothing_acc = smoothing_acc,
              mostlikelypath_acc = mostlikelypath_acc))
}
```

```r
results <- simualte_n(100, HMM)


histfilt <- density(results$filtering_acc)
histsmooth <- density(results$smoothing_acc)
histmostlikelypath <- density(results$mostlikelypath_acc)
plot(histfilt, xlim = c(0,1), ylim = c(0,12), col = "red",
     main = "accuracy distributions", xlab = "accuracy")
lines(histsmooth, col = "blue")
lines(histmostlikelypath, col = "black")

legend(x = "topright",          # Position
       legend = c("filtering", "smoothing", "most probable path"),  # Legend text
       col = c("red", "blue", "black"),           # Line colors
       lwd = 2)                 # Line wid
```




```r
filtering_acc_mean <- mean(results$filtering_acc)
smoothing_acc_mean <- mean(results$smoothing_acc)
mostlikelypath_acc_mean <- mean(results$mostlikelypath_acc)
```

Table 2: accuracy means

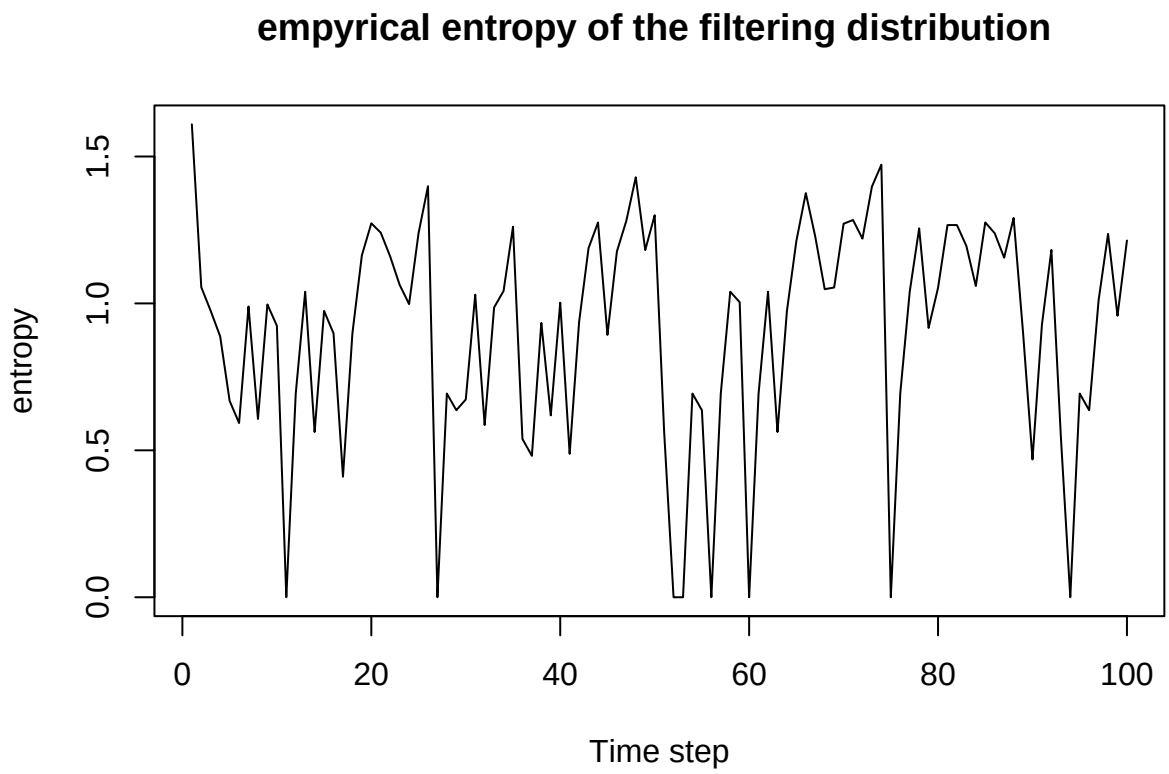
filtering	smoothing	most probable path
0.5335	0.6811	0.4797

- smoothed distribution is more accurate than the filtered distributions because smoothed takes into account future observations.
- smoothed distributions is more accurate than the most probable paths because there are constraints on the path for viterbi algorithm, the algorithm has to return a feasible path while the smoothened distribution returns predictions on one state at a time, which are then combined to a (possibly non-feasible) path.

Question 6

Is it always true that the later in time (i.e., the more observations you have received) the better you know where the robot is ?

```
library(entropy)

simulatedData <- simHMM(HMM, 100)

# 3.

observations <- simulatedData$observation

# filtering
alphazt <- exp(forward(HMM, observations))
normalization <- colSums(alphazt)

filtering <- t(t(alphazt) / normalization)
plot(apply(filtering,2, entropy.empirical),
     type = "l",
     main = "empyrical entropy of the filtering distribution ",
     xlab = "Time step", ylab = "entropy")
```



From the plot we can see that entropy is not decreasing over time. There are some timesteps where entropy is 0 but after that it increases again. This periodicity can't be explained by more certainty due to more samples observed. Other than that, we observed there is some seasonal component explaining the decrease of entropy (more certainty in the prediction) not dependent in the total number of observations registered so far.

Question 7

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

```
transProbs[is.na(transProbs)] <- 0
tbl <- filtering[,100] %*% transProbs
```

Table 3: p(z_t+1|X_1:100)

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
0.3472222	0.125	0	0	0	0	0	0.0277778	0.1527778	0.3472222

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(HMM)
States <- c("S1","S2","S3","S4","S5","S6","S7","S8","S9", "S10")
positions <- c("P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10")
transProbs <- matrix(nrow = 10, ncol = 10)
emissionProbs <- matrix(nrow = 10, ncol = 10)

for (i in c(1:length(States))) {
  updateRows <- c(i:(i+1))
  updateRows[updateRows > length(States)] <- updateRows[updateRows >
                                                          length(States)] - length(States)
  updateRows[updateRows < 1] <-  length(States) + updateRows[updateRows < 1]
  transProbs[i,updateRows] <- rep(1/length(updateRows), length(updateRows))


  updateRows <- c((i-2):(i+2))
  updateRows[updateRows > length(positions)] <- updateRows[updateRows >
                                                             length(positions)] - length(positions)
  updateRows[updateRows < 1] <-  length(positions) + updateRows[updateRows < 1]
  emissionProbs[i,updateRows] <- rep(1/length(updateRows), length(updateRows))

}


HMM <- initHMM(States,
               positions,
               transProbs = transProbs,
               emissionProbs = emissionProbs)
simulatedData <- simHMM(HMM, 100)
observations <- simulatedData$observation

# filtering
alphazt <- exp(forward(HMM, observations))
normalization <- colSums(alphazt)

filtering <- t(t(alphazt) / normalization)


# smoothing

betazt <- exp(backward(HMM, observations))
smoothing <- t(t(alphazt * betazt) / colSums(alphazt * betazt))

# viterbi

mostlikelypath <- viterbi(HMM, observations)
extract_state <- function(vector){
  str_value <- names(which.max(vector))
  return(str_value)
}


filtering_predictions <- apply(filtering,2,extract_state)
```

```r
smoothing_predictions <- apply(smoothing,2,extract_state)
filtering_acc <- sum(diag(table(filtering_predictions, simulatedData$states)))/
  length(filtering_predictions)
smoothing_acc <- sum(diag(table(smoothing_predictions, simulatedData$states)))/
  length(smoothing_predictions)
mostlikelypath_acc <- sum(diag(table(mostlikelypath, simulatedData$states)))/
  length(mostlikelypath)

knitr::kable(t(c(filtering_acc, smoothing_acc, mostlikelypath_acc)),
             caption = "accuracies",
             col.names = c("filtering", "smoothing", "most probable path") )
simualte_n <- function(n, HMM){
  filtering_acc <- vector(length = n)
  smoothing_acc <- vector(length = n)
  mostlikelypath_acc <- vector(length = n)

  for (i in 1:n) {
    simulatedData <- simHMM(HMM, 100)
    observations <- simulatedData$observation
    # filtering
    alphazt <- exp(forward(HMM, observations))
    normalization <- colSums(alphazt)

    filtering <- t(t(alphazt) / normalization)


    # smoothing

    betazt <- exp(backward(HMM, observations))
    smoothing <- t(t(alphazt * betazt) / colSums(alphazt * betazt))

    # viterbi

    mostlikelypath <- viterbi(HMM, observations)
    filtering_predictions <- apply(filtering,2,extract_state)
    smoothing_predictions <- apply(smoothing,2,extract_state)
    filtering_acc[i] <- sum(diag(table(filtering_predictions,
                                       simulatedData$states)))/
      length(filtering_predictions)
    smoothing_acc[i] <- sum(diag(table(smoothing_predictions,
                                       simulatedData$states)))/
      length(smoothing_predictions)
    mostlikelypath_acc[i] <- sum(diag(table(mostlikelypath,
                                            simulatedData$states)))/
      length(mostlikelypath)

  }

  return(list(filtering_acc = filtering_acc,
              smoothing_acc = smoothing_acc,
              mostlikelypath_acc = mostlikelypath_acc))
}
```

```r
results <- simualte_n(100, HMM)


histfilt <- density(results$filtering_acc)
histsmooth <- density(results$smoothing_acc)
histmostlikelypath <- density(results$mostlikelypath_acc)
plot(histfilt, xlim = c(0,1), ylim = c(0,12), col = "red",
     main = "accuracy distributions", xlab = "accuracy")
lines(histsmooth, col = "blue")
lines(histmostlikelypath, col = "black")

legend(x = "topright",          # Position
       legend = c("filtering", "smoothing", "most probable path"),  # Legend text
       col = c("red", "blue", "black"),           # Line colors
       lwd = 2)                 # Line wid


filtering_acc_mean <- mean(results$filtering_acc)
smoothing_acc_mean <- mean(results$smoothing_acc)
mostlikelypath_acc_mean <- mean(results$mostlikelypath_acc)
knitr::kable(t(c(filtering_acc_mean, smoothing_acc_mean, mostlikelypath_acc_mean)),caption = "accuracy m
library(entropy)

simulatedData <- simHMM(HMM, 100)

# 3.

observations <- simulatedData$observation

# filtering
alphazt <- exp(forward(HMM, observations))
normalization <- colSums(alphazt)

filtering <- t(t(alphazt) / normalization)
plot(apply(filtering,2, entropy.empirical),
     type = "l",
     main = "empyrical entropy of the filtering distribution ",
     xlab = "Time step", ylab = "entropy")
transProbs[is.na(transProbs)] <- 0
tbl <- filtering[,100] %*% transProbs
knitr::kable(tbl,caption = "p(z_t+1|X_1:100)", col.names = c("S1", "S2", "S3", "S4", "S5", "S6", "S7",
```