Project 3

Santiago Castro Dau, June Monge, Rachita Kumar, Sarah Lötscher

3/14/2022

Problem 6: Hidden Markov Models

(a) What is the maximum number of parameters to define the HMM?

Taking into account that,

1. each hidden variable Z can take on K different values,
2. each observed variable X can each take on M different values,
3. and that the HMM has L different states,

we have the following parameters in the model:

- Initial state probabilities: $I_k = P(Z_1 = k)$
- Transition probabilities: $T_{kk'} = P(Z_n = k' \mid Z_{n-1} = k)$
- Emission probabilities: $E_{km} = P(X_n = m \mid Z_n = k)$

Since the $\sum_k I_k = 1$ we only need to define $K - 1$ initial state probabilities.

Given that $Z_{n-1} = k$, there are K transition probabilities into state Z_n, one for each value that Z_n can take. Since $\sum_{k'=1}^{K} P(Z_n = k' \mid Z_{n-1} = k) = 1$ we only need to define $K - 1$ transition probabilities for each possible imputation of $Z_{n-1} = k$. Hence we need to define $K * (K - 1)$ transition probabilities.

Similarly because $\sum_{m=1}^{M} P(X_n = m \mid Z_n = k) = 1$, and Z_n can take on K different values, we need to define overall $K * (M - 1)$ emission probabilities.

In sum we need to define $(K - 1) + K(K - 1) + K(M - 1)$ parameters. This is equal to $K^2 + K(M - 1) - 1$ parameters.

(b) What is the stationary distribution π?

To find the stationary distribution $\pi \in \mathbb{R}^n$, is to obtain the spectral decomposition of a Markov matrix. A $n \times n$ matrix is called a Markov matrix if all entries are non negative and the sum of each column vector is equal to 1 (in this case the transpose of our transition matrix). A Markov matrix always has an eigenvalue of 1, and all other eigenvalues are in absolute value smaller or equal to 1. The eigenvector that corresponds to the eigenvalue of 1 is the stationary distribution, since it satisfies that $T^T \pi = \pi$ (note that this is equivalent to $\pi^T T = \pi^T$).

```r
## Define matrix
trans_mat = matrix(c(0.1, 0.4, 0.9, 0.6), nrow = 2, ncol = 2)

## Get spectral decomposition
eigen_decomp = eigen(t(trans_mat))

## Get first column (matrix columns contain the eigenvectors) and normalize
stat_dist = eigen_decomp$vectors[,1] / sum(eigen_decomp$vectors[,1])
stat_dist
```

```
## [1] 0.3076923 0.6923077
```

```
## Check
t(trans_mat) %*% stat_dist
```

```
##           [,1]
## [1,] 0.3076923
## [2,] 0.6923077
```

Alternatively one could compute a very high power of T^T (e.g. $(T^T)^n$) and obtain a matrix which columns contain the stationary distribution. An easy way to reason about this is that any matrix that is diagonalizable can be written as $A = S\Lambda S^{-1}$, where S is the matrix of eigenvectors and Λ is a diagonal matrix with the eigenvalues. One can show that the powers of such a matrix can be written as $A^n = S\Lambda^n S^{-1}$. Λ^n is also diagonal but the entries λ_{ij}^n, are the eigenvalues to the power of n. Since we are dealing with a Markov matrix all other eigenvalues except the one equal to 1 will go to 0 as n goes to infinity. Then when evaluating $S\Lambda^n S^{-1}$ only the convector corresponding to the eigenvalue equal to 1 will survive the matrix multiplication. This in turn will yield a matrix where the columns contain the stationary distribution.

```
## Define error
err = 1e-7

## Init distribution matrix, target, counter
dist_mat = t(trans_mat)
target = matrix(c(stat_dist, stat_dist), nrow = 2, ncol = 2, byrow = FALSE)
n = 0

## Loop while error is larger
while (sum(abs(dist_mat - target)) > err) {
  dist_mat = dist_mat %*% t(trans_mat)
  n = n + 1
}

## Print distribution and power
dist_mat
```

```
##           [,1]      [,2]
## [1,] 0.3076923 0.3076923
## [2,] 0.6923077 0.6923077
```

```
n
```

```
## [1] 13
```

Problem 7: Predicting protein secondary structure using HMMs

In this problem, we will try Predicting protein secondary structure using HMMs.

(a) Load data

We first read the data stored in the file proteins_train.tsv, proteins_test.tsv and proteins_new.tsv.

```
# read the data into D
Train_data <- read.table("./data/proteins_train.tsv", col.names=c("ProtName","AminoAcids","KnownPath"))
# read the data into D
Test_data <- read.table("./data/proteins_test.tsv",col.names=c("ProtName","AminoAcids","KnownPath") )
# read the data into D
New_data <- read.table("./data/proteins_new.tsv", col.names=c("ProtName","AminoAcids"))
```

(b) Get the parameters

Estimate the vector of initial state probabilities I, the matrix of transition probabilities T and the matrix for emission probabilities E by maximum likelihood

```r
#characters
unique.ss <- c("B", "C", "E", "G", "H", "I", "S", "T")
unique.aa <- c("A", "C", "D", "E", "F", "G", "H", "I",
                   "K", "L", "M", "N", "P", "Q", "R", "S",
                   "T", "U", "V", "W", "X", "Y")
```

```r
Get_Parameters<-function(Train_data,unique.ss,unique.aa){

  # Vector of initial state probabilities
  I<-vector()
  for(i in unique.ss) {I<-c(I,(sum(str_count(substr(Train_data$KnownPath,1,1),i))/nrow(Train_data)))}


  # Matrix of transition probabilities
  Tr<- matrix(0,length(unique.ss),length(unique.ss))
  for (i in 1:(nrow(Train_data))){
    for(j in 1:(nchar(Train_data$KnownPath[i])-1)){
      from=which(unique.ss == substr(Train_data$KnownPath[i], j, j))
      to=which(unique.ss == substr(Train_data$KnownPath[i], j+1, j+1))
      Tr[from,to]=Tr[from,to]+1
    }
  }
  Tr<-sweep(Tr, 1, rowSums(Tr), FUN = '/')

  # Matrix for emission probabilities
  E<- matrix(0,length(unique.ss),length(unique.aa))
  for (i in 1:(nrow(Train_data))){
    for(j in 1:(nchar(Train_data$KnownPath[i]))){
      aa=which(unique.aa == substr(Train_data$AminoAcids[i], j, j))
      ss=which(unique.ss == substr(Train_data$KnownPath[i], j, j))
      E[ss,aa]=E[ss,aa]+1
    }
  }
  E<-sweep(E, 1, rowSums(E), FUN = '/')

params<-list(I=I,Tr=Tr,E=E)
return(params)
}
```

```r
#Call function and get parameters
varnamess<-"KnownPath"
params<-Get_Parameters(Train_data,unique.ss,unique.aa)
I=params$I
Tr=params$Tr
E=params$E
```

(c) Estimate the stationary distribution π of the Markov chain

Estimate the stationary distribution π of the Markov chain by solving the eigenvalue problem and by using a brute-force approach

```
#Solving eigenvalue problem
library(expm)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

##
## Attaching package: 'expm'

## The following object is masked from 'package:Matrix':
##
##     expm
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.5

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
# Get the eigenvectors of P, note: R returns right eigenvectors
r=eigen(Tr)
rvec=r$vectors
# left eigenvectors are the inverse of the right eigenvectors
lvec=MASS::ginv(r$vectors)
# The eigenvalues
lam<-r$values
pi_eig<-lvec[1,]/sum(lvec[1,])
pi_eig
```

```
## [1] 0.0116560594 0.2042297412 0.2094674769 0.0343029736 0.3395299222
## [6] 0.0001018782 0.0889276425 0.1117843060
```

```
#Brute-force approach
pi_bru <- (Tr %^% 1000)[1,]
pi_bru
```

```
## [1] 0.0116560594 0.2042297412 0.2094674769 0.0343029736 0.3395299222
## [6] 0.0001018782 0.0889276425 0.1117843060
```

```
#Is Stationary distribution of The two approaches the same?
all.equal(pi_bru,pi_eig)
```

```
## [1] TRUE
```

(d) Predict with Viterbi algorithm

Predict the latent state sequence Z of a protein's amino acid sequence X using the Viterbi algorithm:

Viterbi function:

```r
viterbi <- function(E, Tr, I, p) {
    .as.array <- function(.) stringr::str_split(., "")[[1]]
    unique.ss <- c("B", "C", "E", "G", "H", "I", "S", "T")
    unique.aa <- c("A", "C", "D", "E", "F", "G", "H", "I",
                   "K", "L", "M", "N", "P", "Q", "R", "S",
                   "T", "U", "V", "W", "X", "Y")
    for (k in seq(nrow(p))) {
        sequence <- p$AminoAcids[k]
        aa.vec <- .as.array(sequence) %>% match(unique.aa)
        P      <- matrix(0, nrow(E), length(aa.vec))
        Ptr    <- matrix(0, nrow(E), length(aa.vec))

        ## sets the paths
        for (i in seq(length(aa.vec))) {
            if (i == 1) {
                P[, i] <- I + E[, aa.vec[i]]
            } else {
                for (j in seq(nrow(E))) {
                    p.loc    <- P[, i - 1] + Tr[, j] + E[j, aa.vec[i]]
                    P[j, i] <- max(p.loc)
                    Ptr[j, i] <- which.max(p.loc)
                }
            }
        }

        ## backtrace: computes the most likely path
        Phi <- vector(mode="integer",   length=length(aa.vec))
        Phi[length(Phi)] <- which.max(P[, ncol(P)])
        ## we start at the back, just as with Needleman-Wunsch or Smith-Waterman
        for (i in seq(from=length(aa.vec), to=2)) {
            Phi[i - 1] <- Ptr[Phi[i], i]
        }

        states <- unique.ss[Phi]
        p$PredictedStructure[k] <- paste(states, collapse="")
    }
    return(p)
}
```

Call function & export new table:

```r
PNew_data<-viterbi(E, Tr, I, New_data)
PTest_data<-viterbi(E, Tr, I, Test_data)
write.table(PNew_data, file='proteins_new_pedicted.tsv', quote=FALSE, sep='\t')
```

(e) Estimate confidence intervals for each parameter in I, E and T with bootstrapping

```r
#TAKES A WHILE TO RUN
## Bootstrapping
set.seed(0)
library(boot)
#Number of bootstraps
nbst<-1000
#Resample the Parameters
```

```
resamples<-lapply(1:nbst,function(x)
  Get_Parameters(Train_data[sample(nrow(Train_data),replace = TRUE),],unique.ss,unique.aa)
  );


Ilist<-lapply(resamples,'[[',1)
Trlist<-lapply(resamples,'[[',2)
Elist<-lapply(resamples,'[[',3)

#Calculate the Confidence Intervalls
I.mean<- apply(simplify2array(Ilist),c(1),mean)
I.sd<- apply(simplify2array(Ilist),c(1),sd)
Tr.mean<- apply(simplify2array(Trlist),c(1,2),mean)
Tr.sd<- apply(simplify2array(Trlist),c(1,2),sd)
E.mean<- apply(simplify2array(Elist),c(1,2),mean)
E.sd<- apply(simplify2array(Elist),c(1,2),sd)

I.lowerCI <- I.mean - 1.96 * I.sd /sqrt(nbst)
I.upperCI <- I.mean + 1.96 * I.sd /sqrt(nbst)

Tr.lowerCI <- Tr.mean - 1.96 * Tr.sd /sqrt(nbst)
Tr.upperCI <- Tr.mean + 1.96 * Tr.sd /sqrt(nbst)

E.lowerCI <- E.mean - 1.96 * E.sd /sqrt(nbst)
E.upperCI <- E.mean + 1.96 * E.sd /sqrt(nbst)
```

(f) Compute the accuracy of the predicted secondary structure

```
viterbiaccuracy<-vector()
for (i in 1:(nrow(PTest_data))){
  predicted = unlist(strsplit(PTest_data$PredictedStructure[i],""))
  given = unlist(strsplit(PTest_data$KnownPath[i],""))
  viterbiaccuracy<-c(viterbiaccuracy,sum(given==predicted)/nchar(PTest_data$PredictedStructure[i]))
}
#Get summary
summary(viterbiaccuracy)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.007752 0.217105 0.308155 0.300903 0.402715 0.683544
```

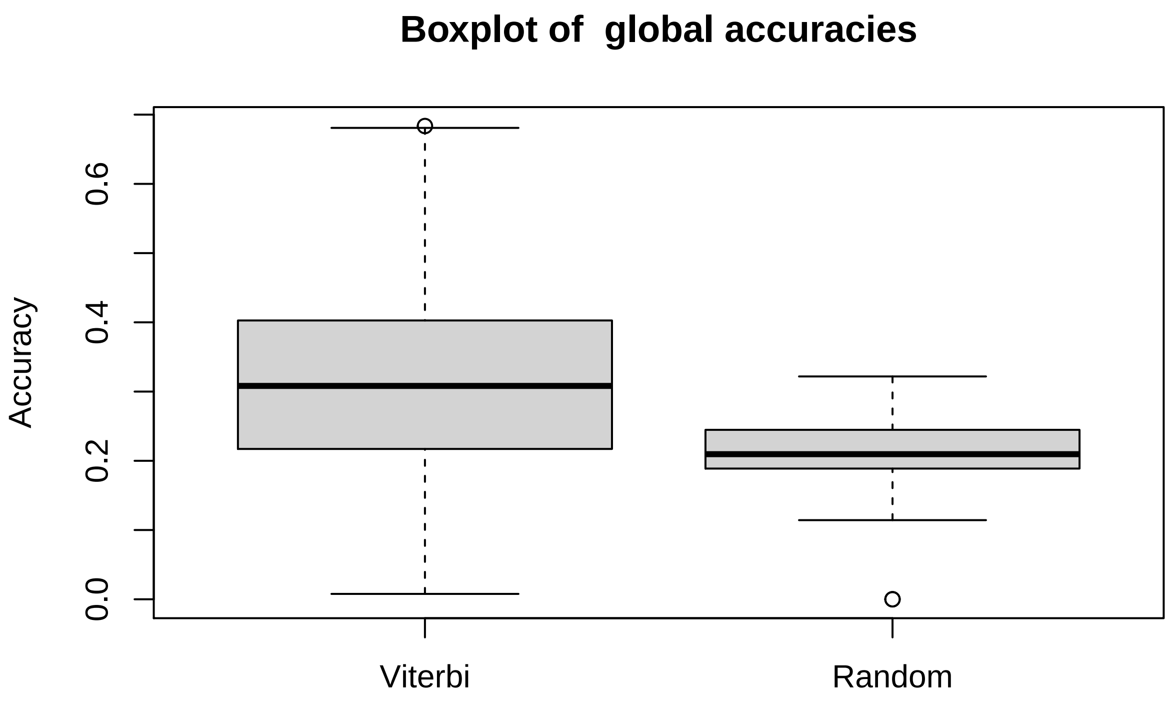
(g) Global accuracies of the Viterbi and the random approach

```
#Get probability matrix for the ss characters
probchar<-vector()
for(i in 1:length(unique.ss)){
  probchar[i]=(sum(str_count(Test_data$KnownPath,unique.ss[i]))/sum(nchar(Test_data$KnownPath)))
}

#Get vector of accuracy of randomly guessed secondary structures
randomaccuracy<-vector()
for (i in 1:(nrow(PTest_data))){
  sampled=sample(unique.ss, nchar(Test_data$KnownPath[i]),replace = TRUE,prob =probchar) #ramdom sampli
  given = unlist(strsplit(PTest_data$KnownPath[i],""))
  randomaccuracy<-c(randomaccuracy,sum(given==sampled)/nchar(PTest_data$PredictedStructure[i]))
}
#Get summary
summary(randomaccuracy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1892  0.2096  0.2114  0.2445  0.3218
```

```
boxplot(viterbiaccuracy,randomaccuracy,ylab='Accuracy',main='Boxplot of  global accuracies',names = c(
```


Boxplot of global accuracies
Accuracy
Viterbi
Random