SECONDARY PROTEIN STRUCTURE PREDICTION

Miguel Ángel Juárez Garzón

Exercise 1: Data sets and definitions

Data loading

```
df <- read.csv("prots-L30.txt",sep = ",",header = TRUE)
df[] <- lapply(df,as.character)
df <- transform(df, len = as.numeric(len))
summary(df)
```

```
##     pdb_id              seq                sst3                len
##  Length:174         Length:174         Length:174         Min.   :30
##  Class :character   Class :character   Class :character   1st Qu.:30
##  Mode  :character   Mode  :character   Mode  :character   Median :30
##                                                           Mean   :30
##                                                           3rd Qu.:30
##                                                           Max.   :30
```

```
df
```

pdb_id	seq	sst3	l...
<chr>	<chr>	<chr>	<dbl>
1AI0	FVNQHLCGSHLVEALYLVCGERGFFYTPKT	CCCHHHHHHHHHHHHHHHHHHCEEECCCC	30
1APH	FVNQHLCGSHLVEALYLVCGERGFFYTPKA	CCCCCCCCHHHHHHHHHHHHHHCEEECCCC	30
1B4G	MISSVCVSSYRGRKSGNKPPSKTCLKEEMA	CCCCCECCCCCCCCCCCCCCCCCCCCCCCC	30
1B9E	FVNQHLCGEHLVEALYLVCGERGFFYTPKT	CCCCCCCHHHHHHHHHHHHHHHCEEECCCC	30
1BH4	CGESCVWIPCISAALGCSCKNKVCYRNGIP	CCCCCCCCCCCCHHHHCCCCCCCCCCCCCC	30
1BHX	SGEADCGLRPLFEKKSLEDKTERELLESYI	CCCCCCCCCCCCCHHHCCCCCCHHHHHHHC	30
1BKV	PPGPPGPPGITGARGLAGPPGPPGPPGPPG	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC	30
1CAG	PPGPPGPPGPPGPPAPPGPPGPPGPPGPPG	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC	30
1D0R	HAEGTFTSDVSSYLEGQAAKEFIAWLVKGR	CCCCCCCHHHHHHHHHHHCHHHHHHCCCCC	30
1DF6	SCVYIPCTVTALLGCSCSNRVCYNGIPCAE	CCCCCCCCCCCHHHCCEEECCEEECCCECCC	30

1-10 of 174 rows Previous **1** 2 3 4 5 6 … 18 Next

Definitions

```
alphabet <- strsplit("RKDEQSCHNTWYMAILFVPG","")
lalp <- length(alphabet[[1]])
cat("Alphabet length: ", lalp, "\n")
```

```
## Alphabet length:  20
```

```
nseq <- length(df$seq)
cat("Number of sequences: ", nseq, "\n")
```

```
## Number of sequences:  174
```

```
lseq <- df$len[1]
cat("Sequence length: ", lseq, "\n")
```

```
## Sequence length:  30
```

Lista estructuras

```
strlist <- strsplit("CEH","")
nsta <- length(strlist[[1]])
cat("Number of structures per aa: ", nsta, "\n")
```

```
## Number of structures per aa:  3
```

Exercise 2: Train and test set

```
#Random sample
prot_splice <- sample(nseq,0.7 * nseq,replace = FALSE)
# Creating training and test dataset
dftrain <- df[prot_splice,]
dftest <- df[-prot_splice,]

ntrain <- length(dftrain$seq)
ntest <- length(dftest$seq)

cat("Train set size: ", ntrain, "\n")
```

```
## Train set size:  121
```

```
cat("Test set size: ", ntest, "\n")
```

```
## Test set size:  53
```

Amino acids associated to coil (C)

```
struc <- strsplit(dftrain$sst3[1],"")
seque <- strsplit(dftrain$seq[1],"")
idx <- which(struc[[1]] == "C")
if (length(idx) > 0){
aasubset <- seque[[1]][idx]
laasubset <- length(aasubset)
print(aasubset)
}
```

```
##  [1] "S" "C" "C" "P" "C" "C" "P" "S" "G" "C" "S" "G" "C" "G" "K" "T" "C" "D" "T"
## [20] "S" "C" "C" "Q"
```

Exercise 3: Frequency of amino acids relate to C in the first sequence

```
frecs <- matrix(0, nrow = 1, ncol = length(alphabet[[1]]))
colnames(frecs) <- alphabet[[1]]
frecs
```

```
##      R K D E Q S C H N T W Y M A I L F V P G
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
for (i in 1:lalp){
  #Where ddos an aa appear
  pos_aa <- grep(alphabet[[1]][i], aasubset)
  #Count of repeats of an aa
  num_aa <- length(pos_aa)
  frecs[1,alphabet[[1]][i]] <- num_aa
}

frecs
```

```
##      R K D E Q S C H N T W Y M A I L F V P G
## [1,] 0 1 1 0 1 4 9 0 0 2 0 0 0 0 0 0 0 0 2 3
```

Exercise 4: Absolute freqeuncies associated to C structure

```
frecs <- matrix(0, nrow = 1, ncol = length(alphabet[[1]]))
colnames(frecs) <- alphabet[[1]]

for (j in 1:ntrain){
  struc <- strsplit(dftrain$sst3[j],"")
  seque <- strsplit(dftrain$seq[j],"")
  idx <- which(struc[[1]] == "C")
  if (length(idx) > 0){
  aasubset <- seque[[1]][idx]
  laasubset <- length(aasubset)
  }

  for (i in 1:lalp){
    pos_aa <- grep(alphabet[[1]][i], aasubset)
    num_aa <- length(pos_aa)
    frecs[1,alphabet[[1]][i]] <- frecs[1,alphabet[[1]][i]] + num_aa
  }
}

frecs
```

```
##        R   K  D   E  Q   S   C  H  N   T  W  Y  M   A  I  L  F  V   P   G
## [1,] 104 148 79 107 77 129 158 49 70 126 15 66 29 123 82 82 85 88 185 212
```

Exercise 5: Emission matrix

We obtain the emission matrix and verify that it is a stochastic matrix by summing the rows of the matrix and checking that the sum is equal to 1.

```
frecs <- matrix(0, nrow = 3, ncol = length(alphabet[[1]]))
colnames(frecs) <- alphabet[[1]]
rownames(frecs) <- strlist[[1]]
frecs
```

```
##   R K D E Q S C H N T W Y M A I L F V P G
## C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## E 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
# Frequency of aa in C structure
for (j in 1:ntrain){
  struc <- strsplit(dftrain$sst3[j],"")
  seque <- strsplit(dftrain$seq[j],"")
  idx <- which(struc[[1]] == "C")
  if (length(idx) > 0){
  aasubset <- seque[[1]][idx]
  laasubset <- length(aasubset)
  }

  for (i in 1:lalp){
    pos_aa <- grep(alphabet[[1]][i], aasubset)
    num_aa <- length(pos_aa)
    frecs["C",alphabet[[1]][i]] <- frecs["C",alphabet[[1]][i]] + num_aa
  }
}

#Frequency of aa in E structure
for (j in 1:ntrain){
  struc <- strsplit(dftrain$sst3[j],"")
  seque <- strsplit(dftrain$seq[j],"")
  idx <- which(struc[[1]] == "E")
  if (length(idx) > 0){
  aasubset <- seque[[1]][idx]
  laasubset <- length(aasubset)
  }

  for (i in 1:lalp){
    pos_aa <- grep(alphabet[[1]][i], aasubset)
    num_aa <- length(pos_aa)
    frecs["E",alphabet[[1]][i]] <- frecs["E",alphabet[[1]][i]] + num_aa
  }
}

#Frequency of aa in H structure
for (j in 1:ntrain){
  struc <- strsplit(dftrain$sst3[j],"")
  seque <- strsplit(dftrain$seq[j],"")
  idx <- which(struc[[1]] == "H")
  if (length(idx) > 0){
  aasubset <- seque[[1]][idx]
  laasubset <- length(aasubset)
  }

  for (i in 1:lalp){
    pos_aa <- grep(alphabet[[1]][i], aasubset)
    num_aa <- length(pos_aa)
    frecs["H",alphabet[[1]][i]] <- frecs["H",alphabet[[1]][i]] + num_aa
  }
}

frecs
```

```
##     R   K  D   E  Q   S   C  H  N   T  W   Y  M   A  I   L  F   V   P   G
## C 104 148 79 107 77 129 158 49 70 126 15  66 29 123 82  82 85  88 185 212
## E  89  19 12  13 18  23 143 11 10  56 31 110  2  69 51  48 85  33  16  43
## H  89  81 45 139 43  99  78 61 31  41 18  70 25 158 69 273 51 160  21  65
```

```
probs <- frecs/rowSums(frecs)
probs
```

```
##            R          K          D          E          Q          S          C
## C 0.05163853 0.07348560 0.03922542 0.05312810 0.03823237 0.06405164 0.07845084
## E 0.10090703 0.02154195 0.01360544 0.01473923 0.02040816 0.02607710 0.16213152
## H 0.05504020 0.05009276 0.02782931 0.08596166 0.02659246 0.06122449 0.04823748
##            H          N          T           W          Y           M
## C 0.02432969 0.03475670 0.06256207 0.007447865 0.03277061 0.014399206
## E 0.01247166 0.01133787 0.06349206 0.035147392 0.12471655 0.002267574
## H 0.03772418 0.01917130 0.02535560 0.011131725 0.04329004 0.015460730
##            A          I          L          F          V          P          G
## C 0.06107249 0.04071500 0.04071500 0.04220457 0.04369414 0.09185700 0.10526316
## E 0.07823129 0.05782313 0.05442177 0.09637188 0.03741497 0.01814059 0.04875283
## H 0.09771181 0.04267161 0.16883117 0.03153989 0.09894867 0.01298701 0.04019790
```

```
rowSums(probs)
```

```
## C E H
## 1 1 1
```

Exercise 6: MTransition matrix and initial probabilities vector

Transition Matrix

We obtain the transition matrix and verify that it is a stochastic matrix by summing the rows of the matrix and checking that the sum is equal to 1.

```
frec_trans <- matrix(0, nrow = 3, ncol = length(strlist[[1]]))
colnames(frec_trans) <- strlist[[1]]
rownames(frec_trans) <- strlist[[1]]
frec_trans
```

```
##   C E H
## C 0 0 0
## E 0 0 0
## H 0 0 0
```

```
for (i in 1:length(dftrain$sst3)){
  struc <- unlist(strsplit(dftrain$sst3[i],""))
  for (j in 1:29){
    frec_trans[struc[j], struc[j+1]] <- frec_trans[struc[j], struc[j+1]] + 1
  }
}

frec_trans
```

```
##      C   E    H
## C 1638 144  111
## E  144 294    1
## H  111   1 1065
```

```
prob_trans <- frec_trans/rowSums(frec_trans)
prob_trans
```

```
##            C            E           H
## C 0.86529319 0.0760697306 0.058637084
## E 0.32801822 0.6697038724 0.002277904
## H 0.09430756 0.0008496177 0.904842821
```

```
rowSums(prob_trans)
```

```
## C E H
## 1 1 1
```

Initial probabilities vector

```
Pini <- rep(0,length(strlist[[1]]))
names(Pini) <- strlist[[1]]
for (i in 1:ntrain){
  struc <- strsplit(dftrain$sst3[i],"")
  idx = which(struc[[1]][1] == strlist[[1]])
  Pini[idx] <- Pini[idx]+1
}
Pini
```

```
##   C   E   H
## 121   0   0
```

```
Pini <- Pini/sum(Pini)
Pini
```

```
## C E H
## 1 0 0
```

Exercise 7: Secondary structure prediction

```
# Viterbi
# Parameters: emission matrix, an M x N matrix, where emission[j, k] is the probabili
ty of observing state k from hidden state j
# transition matrix, an N x N matrix, with transition[j, k] showing the probability o
f transitioning from state k to state j
# initial is a N x 1 vector of initial probabilities for hidden states. i.e. the p(x=
i) at time t=1
# observations is a L x 1 vector of observations

# Output: An L x 1 vector of the most likely hidden state sequence

myviterbi <- function(emission, transition, initial, observations) {

  # helper method that checks if the inputs are valid. if not, it returns an error me
ssage
  checkInputs(emission, transition, initial, observations)

  numStates <- nrow(transition)

  numObs <- length(observations)

  # initialize the two matrices, stateSeq will store the most likely states up until
this point, while prob state
  # is the corresponding likelihood
  probSeq <- matrix(data=0, nrow=numStates, ncol=numObs)

  stateSeq <- matrix(data=0, nrow=numStates, ncol=numObs)

  firstObs <- observations[1]

  # fill in first columns of both matrices

  probSeq[,1] <- initial[]*emission[,firstObs]

  stateSeq[,1] <- 0


  for (i in 2:numObs) {

    for (j in 1:numStates) {

      obs <- observations[i]

      # initialize this value to -1. This will be overwritten immediately by the for
loop since all possible values are >= 0
      probSeq[j,i] <- -1

      # the point of this for loop is to find the max and argmax for k
      for (k in 1:numStates) {

        value <- probSeq[k, i-1]*transition[k,j]*emission[j,obs]

        if (value > probSeq[j,i]) {
```

```
          # maximizing for k
          probSeq[j,i] <- value

          # argmaximizing for k
          stateSeq[j,i] <- k

        }
      }
    }
  }

  # MLP = most likely path
  MLP <- numeric(numObs)

  am <- which.max(probSeq[,numObs])
  MLP[numObs] <- am

  for (i in numObs:2) {
    MLP[i-1] <- stateSeq[am,i]
    am <- stateSeq[am,i]
  }

  # argmax for the stateSeq[,numObs]
  #am <- which.max(probSeq[,numObs])

  #MLP[numObs] <- stateSeq[am,numObs]

  # we backtrace using backpointers
  #for (i in numObs:2) {

  #  zm <- which.max(probSeq[,i])

  #  MLP[i-1] <- stateSeq[zm,i]

  #}

  return(MLP)

}


# helper method that checks if inputs are valid. If any of the inputs are invalid, an
error message is thrown.
checkInputs <- function(emission, transition, initial, observations){

  numStates <- nrow(transition)

  # checks if the dimensions of the matrices line up
  if (nrow(emission) != numStates) {
    stop("the dimensions of the emission matrix and transition matrix don't line u
p!")
  }

  if (nrow(transition) != ncol(transition)) {
    stop("the transition matrix is not square!")
```

```
  }

  if (length(initial) != numStates) {
    stop("the length of the initial vector of probabilities is not equal to the numbe
r of states in the transition matrix!")
  }

  # checks if the vectors are actually vectors
  if (is.vector(initial) != TRUE || is.vector(observations) != TRUE) {
    stop("one of the vectors is not actually a vector!")
  }

  # the next three check if probabilities sum up to 1

  if (sum(initial) != 1) {
    stop("the initial vector does not sum up to 1!")
  }

  for (i in 1:numStates) {
    if (sum(emission[i,]) != 1) {
      stop("one or more of the rows of the emission matrix do not sum to 1!")
    }
  }

  for (i in 1:numStates) {
    if (sum(transition[i,]) != 1) {
      stop("one or more of the rows of the transition matrix do not sum to 1!")
    }
  }

}
seq_i <- unlist(strsplit(dftest$seq[3],""))

res <- myviterbi(probs, prob_trans, Pini, seq_i)
res
```

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

```
tb <- as.matrix(table(res, unlist(strsplit(dftest$sst3[3],""))))

tb
```

```
##
## res  C
##   1 30
```

```
tb_tot <- matrix(0, nrow = 3, ncol = 3)
colnames(tb_tot) <- strlist[[1]]
rownames(tb_tot) <- strlist[[1]]

strtoi(rownames(tb))
```

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

```
for (i in 1:length(rownames(tb))){
  idx_i <- strtoi(rownames(tb)[i])

  for (j in 1:length(colnames(tb))){
    idx_j <- colnames(tb)[j]

    tb_tot[idx_i , idx_j] <- tb_tot[idx_i, idx_j] + tb[idx_i, idx_j]
  }

}



tb_tot
```

```
##    C E H
## C 30 0 0
## E  0 0 0
## H  0 0 0
```

Exercise 8: Confussion matrix and error rate

```
tb_tot <- matrix(0, nrow = 3, ncol = 3)
colnames(tb_tot) <- strlist[[1]]
rownames(tb_tot) <- strlist[[1]]


res <- list ()
for (k in 1:length(dftest$seq)) {
  seq_i <- unlist(strsplit(dftest$seq[k],""))

  res[[k]] <- myviterbi(probs, prob_trans, Pini, seq_i)
}

for (k in 1:length(res)){
  tb <- as.matrix(table(res[[k]],  unlist(strsplit(dftest$sst3[k],""))))
  mdim <- dim(tb) #Dimensiones de la matriz tb

  #Cambio de nombre a la columnas de la matriz tb
  c1 <- grep(1, res[[k]])
  cont1 <- length(c1)
  c2 <- grep(2, res[[k]])
  cont2 <- length(c2)
  c3 <- grep(3, res[[k]])
  cont3 <- length(c3)

  rname <- c()

  if (cont1 > 0){
    rname <- c("C")
  }
  if (cont2 > 0){
    rname <- c(rname, "E")
  }
  if (cont3 > 0){
    rname <- c(rname, "H")
  }

  rownames(tb) <- rname


  for (i in 1:mdim[1]){
  idx_i <- rname[i]

    for (j in 1:mdim[2]){
    idx_j <- colnames(tb)[j]

    tb_tot[idx_i , idx_j] <- tb_tot[idx_i, idx_j] + tb[idx_i, idx_j]

    }


  }

}
```

```
tb_tot
```

```
##     C   E   H
## C 740 123 189
## E  18  55   1
## H 144  13 307
```

```
#Error rate
error_rate <- sum(sum(tb_tot-diag(diag(tb_tot))))/sum(colSums(tb_tot))
error_rate
```

```
## [1] 0.3069182
```