Project 2 EM Algorithm

Team K

March 24, 2023

Problem 6: Hidden Markov Models

According to slides 18, the HMM is parameterized by:

$$\begin{aligned} &\text{Initial state probabilities:} && I_k = P(Z_1 = k) \\ &\text{Transition probabilities:} && T_{kl} = P(Z_n = l | Z_{n-1} = k) \\ &\text{Emission probabilities:} && E_{kx} = P(X_n = x | Z_n = k) \end{aligned}$$

(a)

I_k should sum up to 1 $\longrightarrow$ degree of freedom $= K - 1$

In the $K \times K$ matrix T, each row needs to sum up to 1 $\longrightarrow$ degree of freedom $= K - 1$ for each row

Similarly, in the $K \times M$ matrix X, each row needs to sum up to 1 $\longrightarrow$ degree of freedom $= M - 1$ for each row

Hence, the maximum number of free parameters is $(K - 1) + K \times (K - 1) + K \times (M - 1)$

(b)

Computing the stationary distribution is equivalent to solving $\pi^t = \pi^t T$.

Let $\pi^t = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix})$, we need to solve:

$$\begin{aligned} \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} &= \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{bmatrix} \\ &= \begin{bmatrix} 0.2\pi_1 + 0.6\pi_2 & 0.8\pi_1 + 0.4\pi_2 \end{bmatrix} \end{aligned}$$

Given that $\pi_1 = 1 - \pi_2$e can solve the equations:

$$\begin{cases} \pi_1 = 0.2\pi_1 + 0.6\pi_2 \\ \pi_2 = 0.8\pi_1 + 0.4\pi_2 \\ \pi_1 = 1 - \pi_2 \end{cases} \Rightarrow \begin{cases} \pi_1 = \dfrac{3}{7} \\ \pi_2 = \dfrac{4}{7} \end{cases}$$

Alternatively we can solve by solving eigens (which is porbably the more proper way to do it because solving the equations above by hand for high dimensional transition matrices is not a very smart idea). Since $\pi^t = \pi^t T$, we can re-write it into $\pi = T^t \pi$, which means π is just the corresponding eigenvector of eigenvalue $\lambda = 1$.

Note that the eigenvector should be normalized to satisfy the constraint that $\pi_1 + \pi_2 = 1$.

```r
# initialize the transition matrix
Tr <- matrix(c(0.2, 0.8, 0.6, 0.4), 2, 2, byrow=TRUE)

# solve for the eigenvalues and eigenvectors for transpose of T
eigens <- eigen(t(Tr))

# get the index of lambda = 1
index <- which(eigens$values %>% near(1.0))[1]
ev <- eigens$vectors[, index]

# normalize so that the probs sum to 1
pi <- ev / sum(ev)
pi
```

```
## [1] 0.4285714 0.5714286
```

```r
# check that the two computation confirms
near(pi[1], 3/7)
```

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

```r
near(pi[2], 4/7)
```

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

Problem 7: Predictinig protein secondary structure using HMMs

(a)

> Read `proteins train.tsv`, `proteins test.tsv` and `proteins new.tsv` into the memory and store each in a `data.frame`

```r
train <- read.csv("data/proteins_train.tsv", sep = "\t", header = FALSE)
test  <- read.csv("data/proteins_test.tsv",  sep = "\t", header = FALSE)
new   <- read.csv("data/proteins_new.tsv",   sep = "\t", header = FALSE)
```

```r
header <- c("identifier",
            "sequence",
            "structure")
colnames(train) <- header
colnames(test)  <- header
colnames(new)   <- header[1:2]
```

```r
head(train)
```

```
##   identifier
## 1    >101M:A
## 2    >102L:A
```

```
## 3    >102M:A
## 4    >103L:A
## 5    >103M:A
## 6    >104L:B
##
## 1                 MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRVKHLKTEAEMKASEDLKKHGVTVLTALGAILKKKGHHEAl
## 2   MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILRNAKLKPVYDSLDAVRl
## 3                 MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKAGVTVLTALGAILKKKGHHEAl
## 4 MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAKSELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILRNAKLKPVYDSLDAVRl
## 5                 MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKAGVTVLTALGAILKKKGHHEAl
## 6  MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKSAAELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILRNAKLKPVYDSLDAVRl
##
## 1                 CCCCHHHHHHHHHHHHHHGGGHHHHHHHHHHHHHHHHCGGGGGGCTTTTTCCSHHHHHHCHHHHHHHHHHHHHHHHHHHTTTTCCHHl
## 2   CCHHHHHHHHHCCEEEEEEECTTSCEEEETTEEEESSSCTTTHHHHHHHHHHHTSCCTTBCCHHHHHHHHHHHHHHHHHHHHHHHCTTHHHHHHHSCHHHl
## 3                 CCCCHHHHHHHHHHHHHHGGGHHHHHHHHHHHHHHHHCGGGGGGCTTTTTCCSHHHHHHCHHHHHHHHHHHHHHHHHHHTTTTCCHHl
## 4 CCHHHHHHHHHCCEEEEEEECTTSCEEEETTEECCCCCCCCCHHHHHHHHHHHHTSCCTTBCCHHHHHHHHHHHHHHHHHHHHHHHCTTTHHHHHHSCHHHl
## 5                 CCCCHHHHHHHHHHHHHHGGGHHHHHHHHHHHHHHHHCGGGGGGCTTTTTCCSHHHHHHCHHHHHHHHHHHHHHHHHHHTTTCCHHl
## 6  CCHHHHHHHHHHCCCSBCEECTTSCEECTTTCCCCCCSSCCHHHHHHHHHHHHSCSCCTTBCCHHHHHHHHHHHHHHHHHHHHHHTCTTHHHHHHHSCSSHl
```

(b)

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

First get all the possible amino acid and secondary structure states.

```
# get all the possible amino acid states
aa_states <- train$sequence %>%
  strsplit("") %>%
  unlist() %>%
  unique() %>%
  sort()
aa_states
```

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

```
# get all the possible secondary structure states, and sort them
ss_states <- train$structure %>%
  strsplit("") %>%
  unlist() %>%
  unique() %>%
  sort()
ss_states
```

```
## [1] "B" "C" "E" "G" "H" "I" "S" "T"
```

```
MLE <- function(data, aa_states, ss_states) {
  k <- length(ss_states) # num of latent state
  m <- length(aa_states) # num of observed state
```

```
  # initialize the parameters
  I <- rep(0.0, k)
  names(I) <- ss_states

  E <- matrix(0.0, nrow=k, ncol=m)
  dimnames(E) <- list(ss_states, aa_states)

  Tr <- matrix(0.0, nrow=k, ncol=k)
  dimnames(Tr) <- list(ss_states, ss_states)

  N <- nrow(data) # num of data points

  # iterate over each row of the data
  # TODO: maybe rewrite the for loops
  foreach (i=1:N) %do% {
    seq <- data$sequence[i]
    struct <- data$structure[i]

    ss_1st <- struct %>% substr(1, 1)
    I[ss_1st] <- I[ss_1st] + 1.0

    for (j in 1:nchar(seq)) {
      aa <- seq %>% substr(j, j)
      ss <- struct %>% substr(j, j)

      E[ss, aa] <- E[ss, aa] + 1.0

      if (j < nchar(seq)) {
        ss_next <- struct %>% substr(j+1, j+1)

        Tr[ss, ss_next] <- Tr[ss, ss_next] + 1.0
      }
    }
  }

  # convert thee counts to log probs
  I <- I / sum(I)
  E <- E / rowSums(E)
  Tr <- Tr / rowSums(Tr)

  return(list(I=I, E=E, Tr=Tr))
}
```

```
res <- MLE(train, aa_states, ss_states)
```

```
res$I
```

```
## B C E G H I S T
## 0 1 0 0 0 0 0 0
```

```
res$E
```

```
##              A          C          D          E          F          G          H
```

```
## B 0.04524422 0.03239075 0.06580977 0.02467866 0.04215938 0.05192802 0.03239075
## C 0.06450873 0.01585481 0.07227845 0.04888328 0.03388859 0.08271453 0.02583216
## E 0.05804938 0.02348868 0.02798043 0.04002518 0.05247046 0.04809315 0.02752267
## G 0.10814116 0.01677149 0.09696017 0.08333333 0.03511530 0.06289308 0.03022362
## H 0.12500000 0.01249647 0.05261579 0.08629271 0.03646569 0.03503601 0.02188647
## I 0.23529412 0.00000000 0.17647059 0.00000000 0.00000000 0.00000000 0.00000000
## S 0.05445111 0.01536492 0.08605701 0.05215985 0.02331694 0.14219287 0.02304738
## T 0.06342143 0.01168713 0.07736021 0.06015118 0.02348148 0.19911006 0.02122983
##            I          K          L          M          N          P          Q
## B 0.06221080 0.04575835 0.07506427 0.01645244 0.05449871 0.04215938 0.02982005
## C 0.04323518 0.06118295 0.07018550 0.02319447 0.05708306 0.08277187 0.03145159
## E 0.09306783 0.05767745 0.10511258 0.02374617 0.02400366 0.01782393 0.03135639
## G 0.03162124 0.05957372 0.05835080 0.01502446 0.05241090 0.06446541 0.03546471
## H 0.05491034 0.07296668 0.11285654 0.02974089 0.03641274 0.01929187 0.04131954
## I 0.05882353 0.00000000 0.23529412 0.05882353 0.00000000 0.00000000 0.00000000
## S 0.02749511 0.07244423 0.05067727 0.01489319 0.05492284 0.05761844 0.03093200
## T 0.02466091 0.07918297 0.04975071 0.01302740 0.06733501 0.06792473 0.03683054
##            R          S          T           U          V           W
## B 0.04627249 0.04627249 0.08020566 0.000000000 0.13110540 0.008740360
## C 0.04733507 0.07981880 0.07184839 0.000000000 0.05106224 0.008171106
## E 0.03833720 0.05198409 0.07779018 0.000000000 0.12651275 0.019740795
## G 0.03913347 0.06970650 0.04297694 0.000174703 0.02428372 0.024633124
## H 0.06412384 0.04048997 0.04356114 0.000000000 0.07183705 0.013061282
## I 0.11764706 0.00000000 0.00000000 0.000000000 0.05882353 0.058823529
## S 0.05364243 0.08807871 0.07224206 0.000000000 0.03935575 0.016106207
## T 0.04122661 0.06438643 0.04444325 0.000000000 0.02021123 0.010132418
##              X          Y
## B 0.0010282776 0.06580977
## C 0.0009461280 0.02775309
## E 0.0000000000 0.05521701
## G 0.0000000000 0.04874214
## H 0.0000000000 0.02963499
## I 0.0000000000 0.00000000
## S 0.0002695599 0.02473212
## T 0.0001072214 0.02433925
```

```
res$Tr
```

```
##              B          C            E           G          H            I
## B 0.0185089974 0.60874036 0.0303341902 0.020565553 0.02365039 0.000000e+00
## C 0.0288740867 0.51503859 0.1113882450 0.025176795 0.08471493 0.000000e+00
## E 0.0039195491 0.10814522 0.8126054988 0.004520356 0.00557892 0.000000e+00
## G 0.0078616352 0.11198463 0.0186932215 0.697938505 0.03336827 0.000000e+00
## H 0.0004412595 0.01782689 0.0003000565 0.002859362 0.91049492 1.765038e-05
## I 0.0000000000 0.00000000 0.0000000000 0.000000000 0.05882353 8.235294e-01
## S 0.0258777546 0.38250556 0.0857874520 0.018262686 0.06738999 6.738999e-05
## T 0.0179059669 0.22709484 0.0695866617 0.012866563 0.04026162 5.361068e-05
##            S          T
## B 0.15372751 0.14447301
## C 0.13873647 0.09607089
## E 0.02932509 0.03590536
## G 0.05765199 0.07250175
## H 0.01623835 0.05182152
## I 0.00000000 0.11764706
```

```
## S 0.35656042 0.06354876
## T 0.12040959 0.51182115
```

(c)

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

Eigenvalue method

```r
eigens <- eigen(t(res$Tr))

index <- which(eigens$values %>% near(1.0))[1]

ev <- eigens$vectors[, index]

pi_eigen <- ev / sum(ev)
pi_eigen
```

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

Brute-force

```r
# compute Tr %*% Tr until there's no huge difference between the two
# matrices
Tr <- res$Tr
pi_brute <- rep(0.0, length(ss_states))
pi_brute[1] <- 1.0

while (TRUE) {
  Tr <- Tr %*% res$Tr
  pi_new <- Tr[1, ]

  if (all(near(pi_new, pi_brute))) {
    break
  }

  pi_brute <- pi_new
}
pi_brute
```

```
##            B            C            E            G            H            I
## 0.0116560609 0.2042297695 0.2094675407 0.0343029787 0.3395298132 0.0001018781
##            S            T
## 0.0889276516 0.1117843074
```

```
near(pi_eigen, pi_brute, tol=1e-5)
```

```
##    B    C    E    G    H    I    S    T
## TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

(d)

Having estimated the parameters, i.e., the emission and transition matrices E, T and the vector of initial state probabilities I, you can predict the latent state sequence Z of a protein's amino acid sequence X using the Viterbi algorithm. Use the Viterbi algorithm provided in `viterbi.r` (carefully read the parameter description!) and iterate over each `data.frame` of `proteins_test.tsv` and `proteins_new.tsv` row by row and use the amino acid sequence to predict its secondary structure, which you add to the `data.frame` as a new column. Save the extended `data.frame` of `proteins_new.tsv` including the predicted secondary structure as a tsv file and hand it in together with your pdf.

```
predict_test <- data.frame(test[,2])
colnames(predict_test) <- c("AminoAcids")
predict_test <- viterbi(log(res$E), log(res$Tr), log(res$I), predict_test)
```

```
predict_new <- data.frame(new[,2])
colnames(predict_new) <- c("AminoAcids")
predict_new <- viterbi(log(res$E), log(res$Tr), log(res$I), predict_new)
```

```
# save predict new to a tsv file
write.table(predict_new, file="proteins_new.tsv", sep="\t", quote=FALSE, row.names=FALSE)
```

(e)

Estimate confidence intervals for each parameter in I, E and T with bootstrapping. In a single bootstrap run i estimate the probabilities for I_i, E_i and T_i the same as before, but not on the original data set `proteins train.tsv`, but on the resampled data set. i.e., sample with replacement as many rows from `proteins train.tsv` as the original data set has. Run a thousand bootstraps and compute the empirical 95% confidence intervals for each single parameter in $\{I_i\}_i$, $\{E_i\}_i$ and $\{T_i\}_i$.

```
num_cores <- detectCores()
registerDoParallel(num_cores)

# 1000 rounds of bootstrap
start_time <- Sys.time()
df <- foreach(i=1:1000, .combine=rbind) %dopar% {
  # allow duplicates
  picked <- sample(seq_len(nrow(train)), size=nrow(train), replace = TRUE)
  bootstrap <- train[picked, ]
  params <- MLE(bootstrap, aa_states = aa_states, ss_states = ss_states)
  return(params)
}
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 4.642284 mins
```

```r
stopImplicitCluster()
```

```r
compute_ci <- function(list_of_matrix) {
  # iterate over each matrix
  mat <- as.data.frame(list_of_matrix[[1]])
  dims <- dim(mat)
  df <- data.frame(matrix(0, nrow=length(list_of_matrix), ncol=prod(dims)))
  colnames(df) <- sapply(1:dims[2], function(i) {
    sapply(1:dims[1], function(j) {
      paste0(rownames(mat)[j], colnames(mat)[i])
    })
  }) %>% unlist()

  foreach (i=1:length(list_of_matrix)) %do% {
    v <- as.vector(list_of_matrix[[i]])
    foreach (j=1:length(v)) %do% {
      df[i, j] <- v[j]
    }
  }

  df <- apply(df, 2, quantile, probs=c(.025, .975), na.rm=TRUE)
  return(df)
}
```

```r
I_emp <- as.data.frame(df[, "I"])
```

```r
CI_I <- apply(I_emp, 1, quantile, probs=c(.025, .975))
CI_I
```

```
##       B C E G H I S T
## 2.5%  0 1 0 0 0 0 0 0
## 97.5% 0 1 0 0 0 0 0 0
```

```r
CI_E <- compute_ci(df[, "E"])
CI_E
```

```
##               BA         CA         EA        GA        HA  IA         SA
## 2.5%  0.03635432 0.06140271 0.05538598 0.0998906 0.1215706 0.0 0.05091579
## 97.5% 0.05343542 0.06749620 0.06045254 0.1174090 0.1285715 0.6 0.05800694
##               TA         BC         CC        EC         GC         HC IC
## 2.5%  0.06006786 0.02502431 0.01417279 0.0215623 0.01322209 0.01098700  0
## 97.5% 0.06722680 0.03956492 0.01764005 0.0254180 0.02032604 0.01420429  0
##               SC         TC         BD         CD         ED         GD
## 2.5%  0.01311891 0.01018346 0.05514814 0.06951432 0.02595040 0.08945896
## 97.5% 0.01799385 0.01316080 0.07738191 0.07532314 0.02974233 0.10461799
##               HD  ID         SD         TD         BE         CE         EE
## 2.5%  0.05109790 0.0 0.08087129 0.07303920 0.01791084 0.04564179 0.03807956
## 97.5% 0.05430261 0.5 0.09101875 0.08184575 0.03196311 0.05190119 0.04221728
##               GE         HE IE         SE         TE         BF         CF
## 2.5%  0.07613831 0.08338833  0 0.04795279 0.05628855 0.03364529 0.03218576
```

```
## 97.5% 0.09024001 0.08897424  0 0.05608233 0.06384228 0.05146769 0.03564340
##               EF         GF         HF IF         SF         TF         BG
## 2.5%  0.04969470 0.03090631 0.03468322  0 0.02104954 0.02127275 0.04143068
## 97.5% 0.05507111 0.03971236 0.03815637  0 0.02563673 0.02573626 0.06278365
##               CG         EG         GG         HG IG        SG        TG
## 2.5%  0.07946805 0.04540583 0.05675606 0.03349610  0 0.1362845 0.1934535
## 97.5% 0.08599828 0.05094661 0.06953215 0.03686181  0 0.1486114 0.2048606
##               BH         CH         EH         GH         HH IH         SH
## 2.5%  0.02500634 0.02385950 0.02558289 0.02600525 0.01997901  0 0.02030362
## 97.5% 0.04037768 0.02792535 0.02935934 0.03476747 0.02358552  0 0.02580083
##               TH         BI         CI         EI         GI         HI  II
## 2.5%  0.01896464 0.05163789 0.04135709 0.08840919 0.02730177 0.05294560 0.0
## 97.5% 0.02381493 0.07233106 0.04507839 0.09802903 0.03589331 0.05690486 0.2
##               SI         TI         BK         CK         EK         GK
## 2.5%  0.02467129 0.02231430 0.03644211 0.05768884 0.05529917 0.05415599
## 97.5% 0.03029152 0.02696016 0.05544313 0.06459800 0.05997668 0.06526282
##               HK IK         SK         TK         BL         CL        EL
## 2.5%  0.07022963  0 0.06802870 0.07382026 0.06220728 0.06681375 0.1016875
## 97.5% 0.07597935  0 0.07713352 0.08426053 0.08755281 0.07394112 0.1085223
##               GL        HL        IL         SL         TL         BM
## 2.5%  0.05223900 0.1094512 0.0000000 0.04742590 0.04655491 0.01088041
## 97.5% 0.06419476 0.1161465 0.3333333 0.05431043 0.05277708 0.02233837
##               CM         EM         GM         HM  IM         SM         TM
## 2.5%  0.02156331 0.02201638 0.01204814 0.02842268 0.0 0.01307265 0.01148549
## 97.5% 0.02470488 0.02545947 0.01825394 0.03099260 0.2 0.01692754 0.01457712
##               BN         CN         EN         GN         HN IN         SN
## 2.5%  0.04438215 0.05441928 0.02208392 0.04669289 0.03473972  0 0.05064608
## 97.5% 0.06539827 0.06012690 0.02566504 0.05844281 0.03801255  0 0.05921028
##               TN         BP         CP         EP         GP         HP IP
## 2.5%  0.06383882 0.03436652 0.07890250 0.01637896 0.05857837 0.01810663  0
## 97.5% 0.07108151 0.05098896 0.08632734 0.01922291 0.06995933 0.02059791  0
##               SP         TP         BQ         CQ         EQ         GQ
## 2.5%  0.05448146 0.06404413 0.02221834 0.02950241 0.02943856 0.03121538
## 97.5% 0.06096205 0.07164541 0.03760643 0.03346524 0.03344531 0.04028004
##               HQ IQ         SQ         TQ         BR         CR         ER
## 2.5%  0.03946531  0 0.02815193 0.03421533 0.03703656 0.04495441 0.03628876
## 97.5% 0.04312750  0 0.03394187 0.03994318 0.05605778 0.04960018 0.04040545
##               GR         HR        IR         SR         TR         BS
## 2.5%  0.03438360 0.06184631 0.0000000 0.05019414 0.03840282 0.03696016
## 97.5% 0.04387436 0.06652689 0.3333333 0.05738436 0.04428575 0.05515210
##               CS         ES         GS         HS IS         SS         TS
## 2.5%  0.07694033 0.04860567 0.06211096 0.03850595  0 0.08347292 0.05969466
## 97.5% 0.08283756 0.05545475 0.07717283 0.04254071  0 0.09277657 0.06956620
##               BT         CT         ET         GT         HT IT         ST
## 2.5%  0.06707277 0.06912433 0.07452380 0.03785619 0.04209703  0 0.06808870
## 97.5% 0.09435711 0.07483974 0.08109058 0.04832572 0.04513253  0 0.07636469
##               TT BU CU EU           GU HU IU SU TU        BV         CV
## 2.5%  0.04136754  0  0  0 0.0000000000  0  0  0  0 0.1156525 0.04857190
## 97.5% 0.04769402  0  0  0 0.0005461571  0  0  0  0 0.1481715 0.05368858
##              EV         GV         HV        IV         SV         TV
## 2.5%  0.1216887 0.02020666 0.06983723 0.0000000 0.03615796 0.01815962
## 97.5% 0.1320152 0.02837896 0.07376382 0.1666667 0.04267890 0.02255817
##               BW          CW         EW         GW         HW        IW
## 2.5%  0.00450197 0.007091147 0.01806123 0.02106288 0.01223447 0.0000000
```

```
## 97.5% 0.01309054 0.009258211 0.02133212 0.02823351 0.01393273 0.1666667
##               SW          TW          BX           CX EX GX HX IX           SX
## 2.5%  0.01445074 0.008782257 0.000000000 0.0004732258  0  0  0  0 0.0000000000
## 97.5% 0.01800780 0.011621889 0.003161473 0.0015402601  0  0  0  0 0.0008233997
##                 TX         BY         CY         EY         GY         HY IY
## 2.5%  0.0000000000 0.05386809 0.02608006 0.05275737 0.04380760 0.02794666  0
## 97.5% 0.0003273403 0.07712194 0.02944142 0.05792962 0.05418989 0.03129721  0
##               SY         TY
## 2.5%  0.02223253 0.02223887
## 97.5% 0.02721855 0.02661784
```

```r
CI_Tr <- compute_ci(df[, "Tr"])
CI_Tr
```

```
##               BB         CB          EB          GB           HB IB         SB
## 2.5%  0.01325998 0.02676024 0.003325464 0.005710393 0.0002728514  0 0.02330928
## 97.5% 0.02395156 0.03080354 0.004493492 0.010116071 0.0006300112  0 0.02828759
##               TB        BC        CC        EC        GC         HC IC
## 2.5%  0.01617699 0.5882315 0.5036490 0.1042021 0.1054826 0.01685712  0
## 97.5% 0.01974824 0.6286807 0.5265273 0.1123444 0.1185946 0.01886254  0
##              SC        TC         BE        CE        EE         GE          HE
## 2.5%  0.3752402 0.2207318 0.02296978 0.1064088 0.8075828 0.01529154 0.000138275
## 97.5% 0.3896505 0.2332971 0.03767474 0.1166721 0.8172479 0.02174412 0.000477201
##       IE         SE         TE         BG         CG          EG        GG
## 2.5%   0 0.08057031 0.06552050 0.01444630 0.02351172 0.003788339 0.6947707
## 97.5%  0 0.09060692 0.07358031 0.02720892 0.02678666 0.005227585 0.7010563
##                HG IG         SG         TG         BH         CH          EH
## 2.5%  0.002435176  0 0.01617693 0.01137460 0.01712739 0.07976080 0.004616074
## 97.5% 0.003326148  0 0.02047911 0.01441385 0.03039583 0.08992643 0.006639567
##               GH        HH        IH         SH         TH BI CI EI GI
## 2.5%  0.02910476 0.9090564 0.0000000 0.06214951 0.03749803  0  0  0  0
## 97.5% 0.03797283 0.9118201 0.1666667 0.07256586 0.04328742  0  0  0  0
##                 HI        II           SI           TI        BS        CS
## 2.5%  0.000000e+00 0.8000000 0.0000000000 0.0000000000 0.1393661 0.1331387
## 97.5% 5.571802e-05 0.8333333 0.0002095701 0.0001640494 0.1680994 0.1444835
##               ES         GS         HS IS        SS        TS        BT
## 2.5%  0.02747730 0.05161159 0.01513282  0 0.3487431 0.1158668 0.1283285
## 97.5% 0.03136233 0.06435682 0.01746364  0 0.3646175 0.1254673 0.1611581
##               CT         ET         GT         HT  IT         ST        TT
## 2.5%  0.09219082 0.03412151 0.06575021 0.05022682 0.0 0.05912877 0.5075616
## 97.5% 0.10023637 0.03771215 0.07866412 0.05340817 0.2 0.06772309 0.5162743
```

```r
# options(repr.plot.width = 20, repr.plot.height =30)
ggplot(gather(data), aes(value)) +
  geom_histogram(binwidth = 1e-3, color = "black", fill = "black") +
  facet_wrap(~key, nrow = 8, scales = 'free')
```

(f)

Use the following measure to compute the accuracy of the predicted secondary structure $P = (p_i)$ for the `data.frame` of `proteins_test.tsv` given the real secondary structure $S = (s_i)$:

$$a(P, S) = \frac{1}{L} \sum_i \begin{cases} 1 & \text{if } p_i = s_i \\ 0 & \text{if } p_i \neq s_i \end{cases}$$

with sequence length L. Compute the accuracy for every protein in your `data.frame` and store the accuracies in a vector. What is the accuracy of the Viterbi algorithm over all sequences (i.e. call `summary` on the vector of accuracies)?

```
accuracy_viterbi <- foreach (i=1:nrow(test), .combine = c) %dopar% {
  predicted <- predict_test$PredictedStructure[i] %>%
    strsplit("") %>%
    unlist()
  truth <- test$structure[i] %>%
    strsplit("") %>%
    unlist()
  acc <- sum(truth == predicted) / length(predicted)
  return(acc)
}
summary(accuracy_viterbi)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.007752 0.226253 0.322917 0.319801 0.407240 0.857143
```

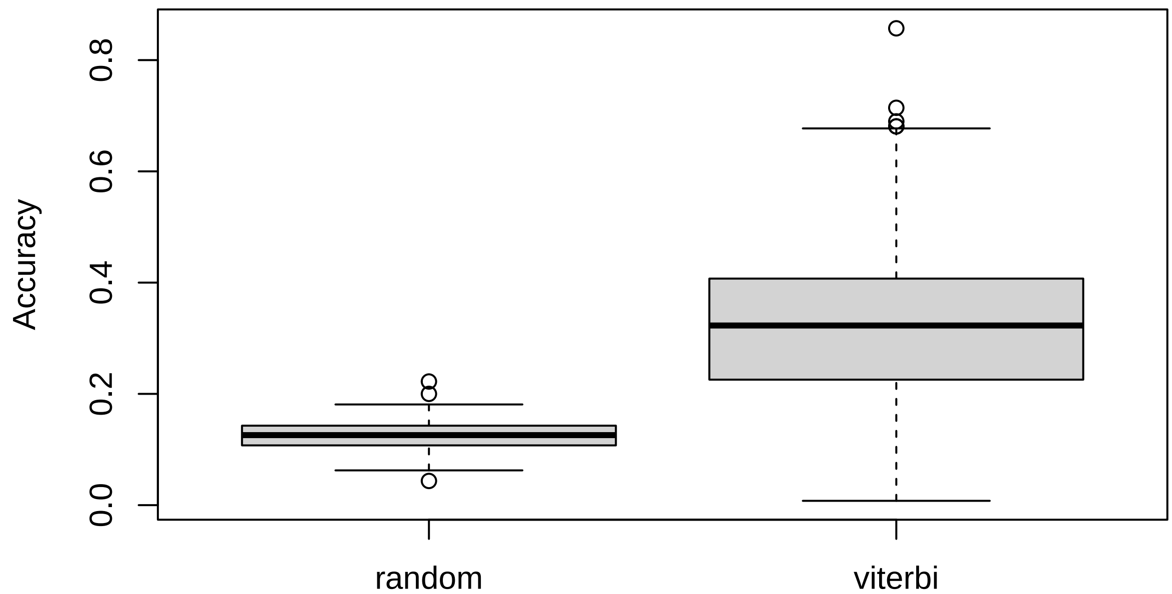
(g)

Instead of using the Viterbi algorithm, now randomly guess secondary structures for all sequences. Compare the global accuracies of the Viterbi and the random approach and plot all accuracy distributions using boxplots.

```
set.seed(2023)
accuracy_random <- foreach (i=1:nrow(test), .combine = c) %dopar% {
  truth <- test$structure[i] %>%
    strsplit("") %>%
    unlist()
  predicted <- sample(ss_states, size=length(truth), replace = TRUE)
  acc <- sum(truth == predicted) / length(truth)
  return(acc)
}
summary(accuracy_random)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.04348 0.10749 0.12585 0.12472 0.14286 0.22222
```

```
boxplot(accuracy_random, accuracy_viterbi,
        ylab = 'Accuracy', names = c('random', 'viterbi'))
```


Accuracy
0.0
0.2
0.4
0.6
0.8
random
viterbi