Assignment 3

Jieran Sun, Hui Jeong (HJ) Jung, Gudmundur Björgvin Magnusson

2023-03-13

Problem 6

a) The number of parameters that define a model depends on the number of number of hidden states = K as well as the number of possible emitted values = M.

$DoF = K * (K - 1) + K * (M - 1) + K - 1 = K * ((K - 1) + (M - 1) + 1) - 1$

(b) We can calculate the stationary distribution according to the ergodicity theorem which states that the stationary distribution π is the solution of $\pi^t = \pi^t * T$. This can be done be finding the leading eigen vector and setting its magnitude to 1. Alternitively, for a small matrix like this we can solve the induced system of equitions directly:

$$\begin{pmatrix} x1 & x2 \end{pmatrix} * \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{pmatrix} = \begin{pmatrix} x1 & x2 \end{pmatrix}$$

This can be simplified to below.

$$x1 * 0.2 + x2 * 0.6 = x1 \quad (1)$$
$$x1 * 0.8 + x2 * 0.4 = x2 \quad (2)$$
$$x1 + x2 = 1 \quad (3)$$

Thus this gives us the results $x1 = 0.428571 = \frac{3}{7}$ and $x2 = 0.571429 = \frac{4}{7}$. Thus $\pi = \begin{pmatrix} \frac{3}{7} & \frac{4}{7} \end{pmatrix}$

Problem 7

a)

Read the data into memory

```r
suppressPackageStartupMessages(library(dplyr))

setwd("C:/Users/zoidp/OneDrive/ETH/StatisticalModelsInComputationalBiology/Project_3_student/")

# Load stuff
source("code/viterbi.r")

data_new <- data.table::fread("data/proteins_new.tsv",data.table = FALSE,header = FALSE)
data_test <- data.table::fread("data/proteins_test.tsv",data.table = FALSE,header = FALSE)
```

```r
data_train <- data.table::fread("data/proteins_train.tsv",data.table = FALSE,header = FALSE)

# Function for converting string to array
str2array <- 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")
```

b)

Here we set up self contained functions that take a data and a set of indices and compute I, T and E arrays. These functions are made to work with boot package for the later sections.

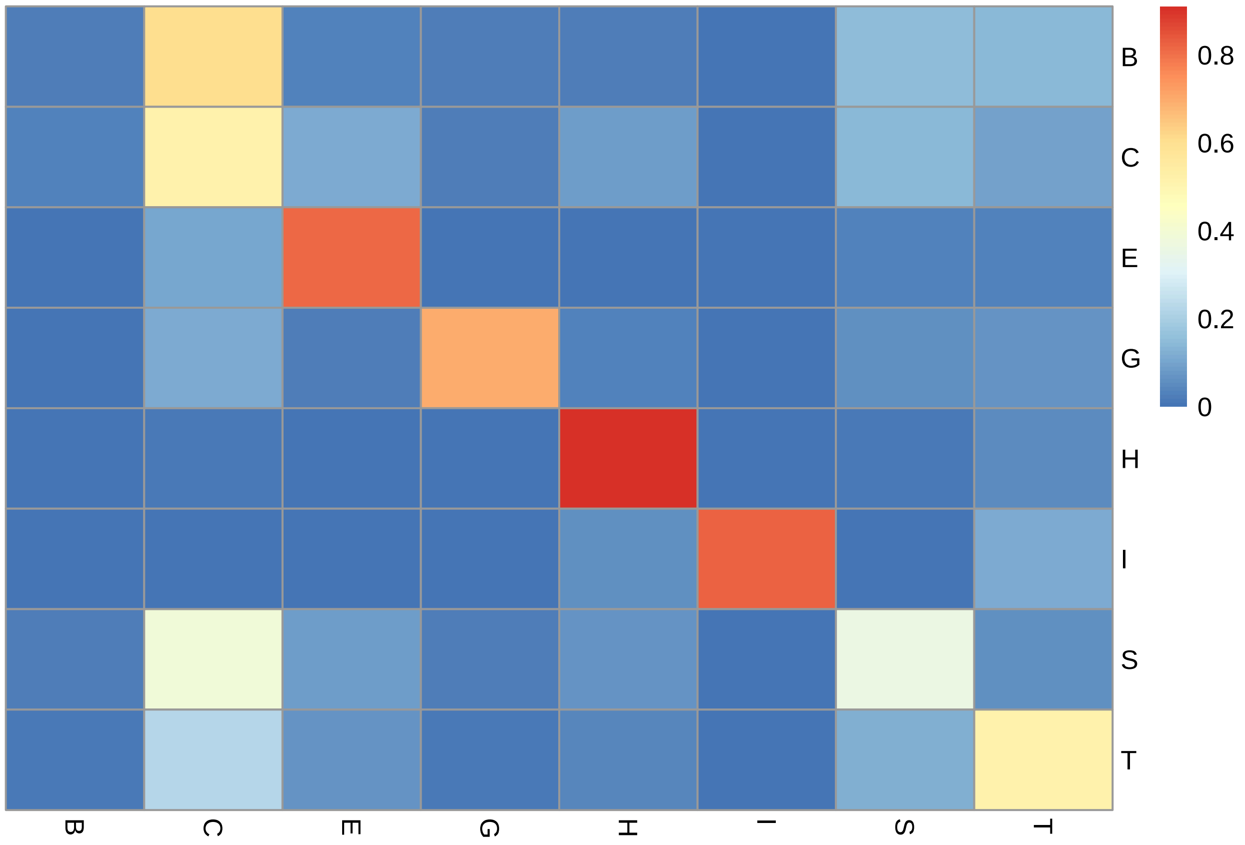
```r
Comp_I <- function(data, indices) {

  unique.ss <- c("B", "C", "E", "G", "H", "I", "S", "T")

  temp <- as.data.frame(data$V3[indices])
  counts  <- apply(temp,1,
                   FUN = function(x) substr(x,start=1,stop=1)) %>%
    table()

    a <- counts[unique.ss] # Order vector
    a[is.na(a)] <- 0       # Fix 0

    I_vec <- as.vector(a/sum(a))

 return(I_vec)
}

# Compute I from training data

I_vec <- Comp_I(data_train,1:nrow(data_train))
I_vec
```

Function for computing inital state probabilities

```
## [1] 0 1 0 0 0 0 0 0
```

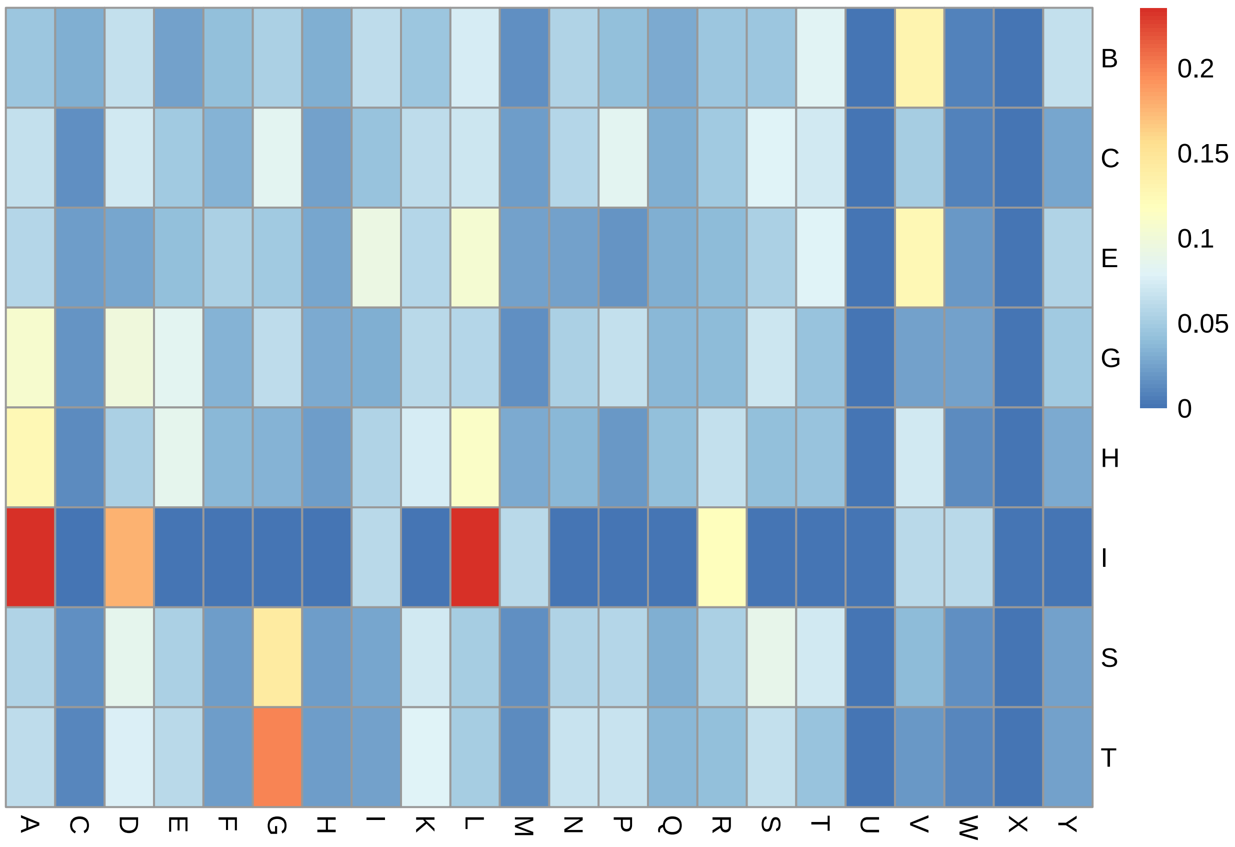
Function for computing Transition probabilities

```r
Comp_T <- function(data, indices) {
  str2array <- function(.) stringr::str_split(., "")[[1]]
  unique.ss <- c("B", "C", "E", "G", "H", "I", "S", "T")
  T_mat <- matrix(0,length(unique.ss),length(unique.ss))

  # Count occurrences of transition

  for (seq in data$V3[indices]) {
    arr <- str2array(seq)
    for (i in 1:(length(arr)-1) ) {
     k <- which(arr[i] == unique.ss)
     l <- which(arr[i+1] == unique.ss)
     T_mat[k,l] <- T_mat[k,l] + 1
    }
  }

  # Divide each element by its row-wise sum to get probabilities

  T_mat <- t(apply(T_mat,1, function(x) x/(sum(x))))


return(T_mat)
}

# Compute T from training data

T_mat <- Comp_T(data_train,1:800)

# Plot Heatmap of transition probs
rownames(T_mat) <- unique.ss
colnames(T_mat) <- unique.ss
pheatmap::pheatmap(T_mat,cluster_rows = F,cluster_cols = F)
```



Function for computing Emission Probabilities

```r
Comp_E <- function(data, indices) {

  str2array <- 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")

  E_mat <- matrix(0,length(unique.ss),length(unique.aa))

  for (i in indices) {
    aa <- str2array(data[i,2])
    ss <- str2array(data[i,3])

    for (j in 1:length(aa) ) {
     k <- which(ss[j] == unique.ss)
     x <- which(aa[j] == unique.aa)
     E_mat[k,x] <- E_mat[k,x] + 1
    }
  }

  E_mat <- t(apply(E_mat,1, function(x) x/sum(x)))
  return(E_mat)
}

# Compute E from training data

E_mat <- Comp_E(data_train,1:800)

# Plot Emission probabilities

rownames(E_mat) <- unique.ss
colnames(E_mat) <- unique.aa
pheatmap::pheatmap(E_mat,cluster_rows = F,cluster_cols = F)
```



c)

Here we estimate stationary distribution three different ways. First we compute the leading eigen vector and divide each element by its sum. We were not entirely sure what was meant by brute-force in the second part, so we tried both exponentiating the transistion matrix n times and extracting a row from it as well as simply simulating the markov chain for 500 000 iterations and computing the distribtuion of states. All methods gave fairly similar estimations.

```
# Eigen Value approach

eigs  <- eigen(t(T_mat))
```

```r
pi <- eigs$vectors[,1]/sum(eigs$vectors[,1])

###### BRUTE FORCE 1 : T to the power of n ######

n = 10

pi_brute <- T_mat

for (i in 1:n) {
  pi_brute <- pi_brute %*% pi_brute
}

pi2 <- pi_brute[1,]


###### BRUTE FORCE 2 : SIMULATION ###########

num.iters = 500000
states_Z     <- numeric(num.iters)
states_X     <- numeric(num.iters)

# Start chain
states_Z[1]  <- which(rmultinom(1, 1, I_vec) == 1)
states_X[1]  <- which(rmultinom(1, 1, E_mat[states_Z[1],] ) == 1)

# Simulate num.iters steps

for(t in 2:num.iters) {

  # probability vector to simulate next state
  p_z  <- T_mat[states_Z[t-1], ]
  p_x  <- E_mat[states_Z[t-1], ]

  ## draw from multinomial and determine states
  states_Z[t] <-  which(rmultinom(1, 1, p_z) == 1)
  states_X[t] <-  which(rmultinom(1, 1, p_x) == 1)
}

# Eigen-solving
pi
```

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

```r
# Exponentiation
pi2
```

```
##            B            C            E            G            H            I
## 0.0116560594 0.2042297412 0.2094674769 0.0343029736 0.3395299222 0.0001018782
##            S            T
## 0.0889276425 0.1117843060
```

```
# MC simulation (for emitted as well)
table(states_Z)/sum(table(states_Z))
```

```
## states_Z
##        1        2        3        4        5        6        7        8
## 0.011588 0.205260 0.209782 0.035102 0.335738 0.000054 0.090196 0.112280
```

```
table(states_X)/sum(table(states_X))
```

```
## states_X
##        1        2        3        4        5        6        7        8
## 0.083940 0.016184 0.058864 0.061594 0.037010 0.076904 0.024342 0.053944
##        9       10       11       12       13       14       15       16
## 0.066482 0.086952 0.023156 0.044244 0.042760 0.035758 0.050934 0.059102
##       17       18       19       20       21       22
## 0.059670 0.000012 0.069144 0.013744 0.000310 0.034950
```

d)

Here we take the logs of our parameters and feed them into the viterbi function to generate predictions.

```
E <- log(E_mat)
Tr <- log(T_mat)
I <- log(I_vec)

colnames(data_train)[2] <- "AminoAcids"
colnames(data_test)[2] <- "AminoAcids"
colnames(data_new)[2] <- "AminoAcids"

test_pred <- viterbi(E=E,Tr=Tr,I=I,p=data_test)
new_pred  <- viterbi(E=E,Tr=Tr,I=I,p=data_new)

write.table(new_pred,"proteins_new.tsv",row.names = F,col.names = F)
```

e)

We use the boot package and our previously defined functions to do boot strapping and confidence interval (percentilce method) estimation.

```
## I

I_bs<- boot::boot(data = data_train,statistic = Comp_I,R = 1000)

I_conf <- c()

## boot::boot.ci does not like that all values are 1 for H

# for (i in 1:8) {
#   temp <- boot::boot.ci(I_bs,index = i,type = "perc")
#   I_conf[i] <- paste0(signif(temp$percent[,4:5],3),collapse = "-")
# }

I_conf <- c("0-0","1-1","0-0","0-0","0-0","0-0","0-0","0-0")

## T

T_bs<- boot::boot(data = data_train,statistic = Comp_T,R = 1000,
                  parallel = "snow",ncpus = 6)

T_bs$t[is.nan(T_bs$t)] <- 0 # fix weird NaNs

T_conf <- c()
for (i in 1:64) {
  temp <- boot::boot.ci(T_bs,index = i,type = "perc")
  T_conf[i] <- paste0(signif(temp$percent[,4:5],3),collapse = "-")
}

T_conf <- matrix(T_conf,
                 nrow = nrow(T_mat),
                 ncol = ncol(T_mat),
                 dimnames = list(unique.ss,unique.ss))

## E

E_bs<- boot::boot(data = data_train,statistic = Comp_E,R = 1000,
                  parallel = "snow",ncpus = 6)

E_conf <- c()
for (i in 1:(dim(E_mat)[1]*dim(E_mat)[2])) {
  temp <- boot::boot.ci(E_bs,index = i,type = "perc")
  E_conf[i] <- paste0(signif(temp$percent[,4:5],3),collapse = "-")
}

E_conf <- matrix(E_conf,
                 nrow = nrow(E_mat),
                 ncol = ncol(E_mat),
                 dimnames = list(unique.ss,unique.aa))

# Report Confidence intervals for I
I_conf
```

```
## [1] "0-0" "1-1" "0-0" "0-0" "0-0" "0-0" "0-0" "0-0"
```

```
# Report Confidence intervals for T
T_conf
```

```
##   B                     C              E                   G
## B "0.0131-0.0238"     "0.588-0.63"   "0.023-0.0378"      "0.0144-0.0273"
## C "0.0268-0.0309"     "0.505-0.528"  "0.107-0.116"       "0.0235-0.0268"
## E "0.00333-0.00455"   "0.104-0.112"  "0.808-0.817"       "0.00372-0.0053"
## G "0.0056-0.0101"     "0.106-0.119"  "0.0153-0.022"      "0.695-0.701"
## H "0.00027-0.000641" "0.0169-0.0189" "0.000141-0.000483" "0.00245-0.0033"
## I "0-0"               "0-0"          "0-0"               "0-0"
## S "0.0235-0.0284"    "0.376-0.389"   "0.0807-0.0908"     "0.0161-0.0204"
## T "0.0159-0.0199"    "0.221-0.233"   "0.0654-0.0734"     "0.0115-0.0143"
##   H                 I            S               T
## B "0.0172-0.0307"   "0-0"        "0.14-0.168"    "0.129-0.161"
## C "0.0798-0.0899"   "0-0"        "0.133-0.144"   "0.092-0.1"
## E "0.00462-0.00665" "0-0"        "0.0273-0.0313" "0.0343-0.0378"
## G "0.0287-0.0379"   "0-0"        "0.0519-0.0635" "0.0665-0.079"
## H "0.909-0.912"     "0-5.44e-05" "0.0151-0.0174" "0.0502-0.0535"
## I "0-0.167"         "0-0.833"    "0-0"           "0-0.2"
## S "0.0626-0.0725"   "0-0.000211" "0.348-0.365"   "0.0591-0.0679"
## T "0.0374-0.0433"   "0-0.000168" "0.116-0.125"   "0.507-0.516"
```

```
# Report Confidence intervals for E
E_conf
```

```
##   A               C               D               E
## B "0.0367-0.0533" "0.0252-0.0404" "0.0544-0.077"  "0.0177-0.0326"
## C "0.0611-0.0676" "0.0141-0.0174" "0.0694-0.0752" "0.0459-0.052"
## E "0.0557-0.0604" "0.0213-0.0254" "0.026-0.03"    "0.0381-0.0422"
## G "0.0992-0.117"  "0.0135-0.0204" "0.089-0.105"   "0.0765-0.0907"
## H "0.121-0.129"   "0.0109-0.0141" "0.051-0.0543"  "0.0838-0.089"
## I "0-0.6"         "0-0"           "0-0.5"         "0-0"
## S "0.0511-0.0578" "0.0131-0.0179" "0.081-0.091"   "0.0484-0.0566"
## T "0.0598-0.0669" "0.0101-0.0133" "0.0729-0.0822" "0.0563-0.0641"
##   F               G               H               I
## B "0.0336-0.0521" "0.0411-0.0629" "0.0243-0.0401" "0.0517-0.0733"
## C "0.0321-0.0357" "0.0795-0.086"  "0.0238-0.028"  "0.0413-0.0453"
## E "0.0498-0.0551" "0.0453-0.0506" "0.0256-0.0295" "0.0882-0.0983"
## G "0.0308-0.0396" "0.0566-0.0691" "0.0256-0.0346" "0.0273-0.0365"
## H "0.0348-0.0381" "0.0333-0.0368" "0.0199-0.0237" "0.0529-0.0569"
## I "0-0"           "0-0"           "0-0"           "0-0.2"
## S "0.0209-0.0257" "0.136-0.149"   "0.0205-0.0257" "0.025-0.0304"
## T "0.0211-0.0258" "0.193-0.205"   "0.0188-0.0236" "0.0225-0.0273"
##   K               L               M               N
## B "0.0364-0.0553" "0.0629-0.0875" "0.0103-0.0226" "0.0433-0.0655"
## C "0.0579-0.0649" "0.0669-0.0738" "0.0217-0.0247" "0.0541-0.06"
## E "0.0555-0.06"   "0.102-0.109"   "0.0218-0.0255" "0.0222-0.0259"
## G "0.0535-0.0655" "0.0519-0.065"  "0.0117-0.0183" "0.0462-0.0582"
## H "0.0705-0.0758" "0.109-0.116"   "0.0284-0.0311" "0.0347-0.0382"
## I "0-0"           "0-0.333"       "0-0.2"         "0-0"
## S "0.0683-0.0771" "0.0472-0.0541" "0.0129-0.017"  "0.0503-0.0593"
## T "0.0742-0.0851" "0.0465-0.0529" "0.0115-0.0144" "0.0639-0.0711"
##   P               Q               R               S
## B "0.0331-0.0512" "0.022-0.0371"  "0.0371-0.0549" "0.0374-0.0557"
## C "0.0791-0.0864" "0.0296-0.0334" "0.045-0.0499"  "0.0767-0.0829"
## E "0.0162-0.0193" "0.0293-0.0334" "0.0361-0.0406" "0.0484-0.0555"
## G "0.0586-0.0705" "0.031-0.0406"  "0.0345-0.044"  "0.0625-0.0774"
## H "0.0181-0.0206" "0.0396-0.0431" "0.0618-0.0666" "0.0385-0.0425"
## I "0-0"           "0-0"           "0-0.333"       "0-0"
## S "0.0542-0.0609" "0.0283-0.0339" "0.0501-0.0573" "0.0835-0.0929"
## T "0.0642-0.0717" "0.0342-0.0396" "0.0383-0.0441" "0.0593-0.0692"
##   T               U            V               W
## B "0.0665-0.0939" "0-0"        "0.115-0.148"   "0.00484-0.0137"
## C "0.0689-0.0747" "0-0"        "0.0486-0.0538" "0.00715-0.00922"
## E "0.0744-0.0815" "0-0"        "0.121-0.132"   "0.0181-0.0214"
## G "0.0373-0.0487" "0-0.000558" "0.0203-0.0284" "0.021-0.0284"
## H "0.042-0.0453"  "0-0"        "0.07-0.0737"   "0.0122-0.014"
## I "0-0"           "0-0"        "0-0.167"       "0-0.167"
## S "0.0681-0.076"  "0-0"        "0.036-0.043"   "0.0142-0.0181"
## T "0.0415-0.0475" "0-0"        "0.018-0.0224"  "0.00879-0.0115"
##   X                  Y
## B "0-0.00322"        "0.0543-0.0786"
## C "0.000478-0.00157" "0.0259-0.0295"
## E "0-0"              "0.0527-0.058"
## G "0-0"              "0.0435-0.054"
## H "0-0"              "0.0279-0.0315"
## I "0-0"              "0-0"
## S "0-0.00083"        "0.0223-0.0273"
## T "0-0.000331"       "0.0222-0.0267"
```

f)

Here we compute and report the accuracy of the viterbi derived, predicted sequences. It is not very good.

```
computeAcc <- function(data) {
  acc <- numeric(nrow(data))
  for (i in 1:nrow(data)) {
    ss_t <- str2array(data[i,3])
    ss_p <- str2array(data[i,4])
    acc[i] <- sum(ss_t == ss_p)/length(ss_t)
  }
  data$Accuracy <- acc
  return(data)
}

test_pred_acc <- computeAcc(test_pred)

summary(test_pred_acc$Accuracy)
```

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

g)

Here we generate random secondary structure sequences of the appropriate length for each sequence in both our test and training set and and compute the accuracy. Alongside generating strings with uniform distribution of SS symbols we also tried generating strings that have the same distribution of symbols as is in the the data set. The viterbi derived predictions are markedly better then random guessing, but not substantially better then distribution informed random strings.

```
# Compute the distribution of ss symbols in data.

temp <- c(data_train$V3,data_test$V3) %>%
paste0(collapse = "") %>%
  str2array() %>%
  table()
```

```r
ss_dist <- as.vector(temp[unique.ss]/sum(temp))

# Define function for generating ss symbols according to some probability vector
randomPreds <- function(data,probs) {
  RandPreds <- c()
  for (i in 1:nrow(data)) {
      n = nchar(data$V3[i])
      RandPreds[i] <- paste0(sample(unique.ss,
                                    size = n,
                                    replace = T,
                                    prob = probs),collapse="")
  }
  data$RandPreds <- RandPreds
  return(data)
}

rand_preds_uniform <- randomPreds(rbind(data_test,data_train),NULL)
rand_preds_dist    <- randomPreds(rbind(data_test,data_train),ss_dist)

rand_pred_uniform_acc <- computeAcc(rand_preds_uniform)
rand_pred_dist_acc    <- computeAcc(rand_preds_dist)

# Accuracy Summary Statistics for Uniform Random Guessing
summary(rand_pred_uniform_acc$Accuracy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1076  0.1241  0.1254  0.1409  0.4286
```

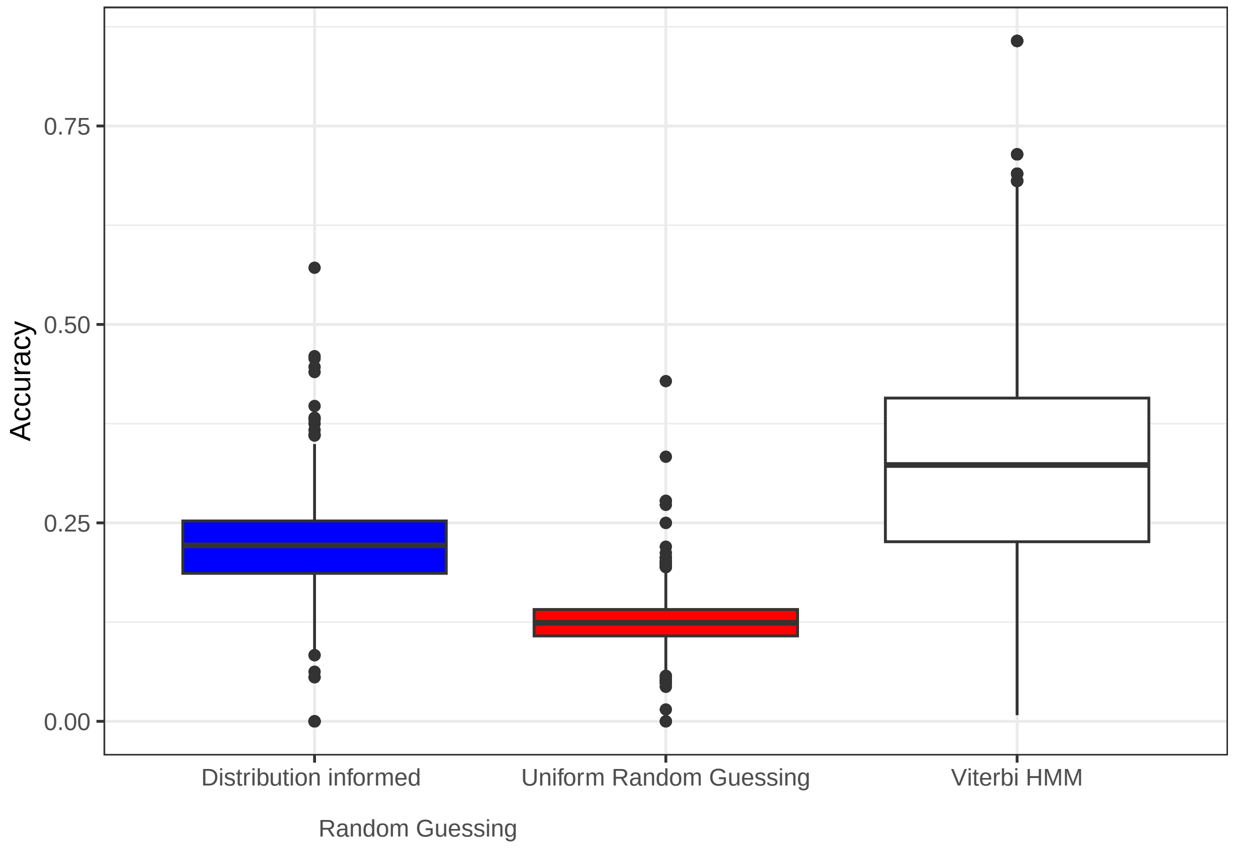
```r
# Accuracy Summary Statistics for training set
# distribution Informed Random Guessing

summary(rand_pred_dist_acc$Accuracy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1864  0.2217  0.2212  0.2524  0.5714
```

```r
suppressPackageStartupMessages(library(ggplot2))
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```r
# Gather accuracy scores in a single dataframe

acc_data <- data.frame(HMM=test_pred_acc$Accuracy,
           UnifRand=rand_pred_uniform_acc$Accuracy,
           DistRand=rand_pred_dist_acc$Accuracy
           )

# Plot the the accuracy of the 3 different methods

 ggplot(acc_data, aes(x = factor("Viterbi HMM"), y = HMM)) +
   geom_boxplot() +
    geom_boxplot(aes(x = factor("Uniform Random Guessing"),
                     y = UnifRand),
                     fill = "red") +
  geom_boxplot(aes(x = factor("Distribution informed \n
                               Random Guessing"),
                   y = DistRand), fill = "blue") +
    theme_bw() +
    theme(axis.title.x = element_blank()) +
    ylab("Accuracy")
```


Accuracy
Distribution informed
Random Guessing
Uniform Random Guessing
Viterbi HMM