BCB 568 Homework Assignment 4

Schuyler Smith

March 27, 2018

1. A first order Markov Model

a. Modeling the first order Markov chain

All the possible transitions states within the data set were defined using the supplied `zwc` function. This results in a table of all unique words of length *order* + 1 (the number of prior states considered and the state being transitioned to) along with a cumulative count of how many times each is observed across all reads in the dataset.

```
transitions <- data.table(zwc::fasta_wc(k=order+1, fasta_file))
transitions <- transitions[order(word)]
```

```
    word    count
 1:   BB    25155
 2:   BC   697536
 3:   BE    41163
 4:   BG    21870
 5:   BH    53472
 6:   BI       88
 7:   BS   158429
 8:   BT   156090
 9:   CB   638810
10:   CC 15806412
11:   CE  2165643
12:   CG   545480
13:   CH  1538660
14:   CI      969
15:   CS  2816193
16:   CT  1806401
```

```
    word    count
 1:   EB    68085
 2:   EC  2237258
 3:   EE 16050747
 4:   EG    76160
 5:   EH    99332
 6:   EI      163
 7:   ES   573109
 8:   ET   638637
 9:   GB    29957
10:   GC   329750
11:   GE    82845
12:   GG  2360588
13:   GH   107648
14:   GI       32
15:   GS   220623
16:   GT   229813
```

To identify all structures that compose the reads in the datset, the reads are split and the unique structures identified.

```
structures <- unique(unlist(apply(array(transitions[[1]]),1,
                            FUN=function(x){strsplit(x,"")})))
```

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

The `transitions` include the order-number of structures prior to the transition and also the structure being transitioned to, so here all prior states are identified by removing the transition from the end, and then the counts summed for each of the prior order-number states observed.

```
priors <- data.table(prior = strtrim(transitions[[1]], order),
                      count = transitions[[2]])
priors <- aggregate(priors[[2]], by=list(priors[[1]]), FUN=sum)
priors <- data.table(word = priors[,1], count = priors[,2])
```

```
   prior     count
1:     B   1153803
2:     C  25318568
3:     E  19743491
4:     G   3361256
5:     H  30866539
6:     I     21850
7:     S   8573688
8:     T  10568209
```

To create a matrix with the transition frequencies. A 0-matrix is initiated with rows equal to the number of prior states and columns for each structure. Then each transition is split into its prior and transition states, assigned to `y` and `x` respectively, and then filled into the matrix at the corresponding position with the ratio of counts of that transition by the number of times that prior state is observed in the data set.

```
P.hat=matrix(0,nrow=length(priors[[1]]), ncol=length(structures))
rownames(P.hat) <- priors[[1]]; colnames(P.hat) <- structures
for(i in 1:length(transitions[[1]])){
  x <- substr(transitions[[1]][i],order+1,order+1)
  y <- strtrim(transitions[[1]][i],order)
  P.hat[y,x] <- (transitions[[2]][i])/(priors[prior==y]$count)}
```

Or, for more expeditious `R` functionality it may be done as:

```
P.hat <- data.table(prior_state = strtrim(transitions[[1]], order),
    transition_state = substr(transitions[[1]], order+1, order+1),
    probability = apply(transitions, 1,
      FUN=function(x){prior <- strtrim(x[[1]], order)
        return(as.numeric(x[[2]])/priors[word == prior,][[2]])})
)
P.hat.mat <- data.matrix(dcast(P.hat, prior_state ~ transition_state,
                 value.var = "probability", fill = 0))[,-1]
```

```
      B     C     E     G     H     I     S     T
B 0.022 0.605 0.036 0.019 0.046 0.000 0.137 0.135
C 0.025 0.624 0.086 0.022 0.061 0.000 0.111 0.071
E 0.003 0.113 0.813 0.004 0.005 0.000 0.029 0.032
G 0.009 0.098 0.025 0.702 0.032 0.000 0.066 0.068
H 0.001 0.018 0.001 0.003 0.913 0.000 0.012 0.053
I 0.007 0.034 0.001 0.004 0.023 0.812 0.037 0.083
S 0.024 0.391 0.086 0.017 0.051 0.000 0.367 0.065
T 0.016 0.222 0.061 0.011 0.044 0.000 0.122 0.524
```

To calculate the *alpha*, or probability for each starting state, each sequence is read-in and then the starting states are tabled. In some cases, the length of the sequence was less than the *order*, for this part (creating the *alpha*-matrix) they were removed by only including intial states that were observed to be equal to the *order*, and then handled later on. `alpha` was then calculated with the ratio of starting states divided by the total sum of all the starting states of $length = order$ observed in the data set.

```
seqs <- readBStringSet(fasta_file)
seqs <-paste(seqs)
initial <- table(apply(array(seqs), 1, FUN=function(y){strtrim(y,order)}))
initial <- data.table(word = names(initial), count = as.vector(initial))
initial <- initial[apply(array(initial[[1]]),1,nchar) == order]

n<-sum(initial[[2]])
alpha <- data.frame(alpha = (initial[[2]])/n, row.names = initial[[1]])
```

```
  alpha
C     1
```

To calculate the log-likelihoods for each sequence and then the dataset under a given order, the likelihood of observing each read needs to be determined. This is done by taking the probability of the starting sequence from the alpha matrix and then the likelihood of the read is the product of that value and the probability of each transition under the given order. When finding the product of many probabilities the value quickly approches 0. Typically, the log-likelihood of a dataset would be evaluated as the *log* of the product of the probability of each state of each read and the product of the likelihood of every read. This gives a number too small for computers to handle in memory, so it is treated as 0, and the $log(0) = -Inf$, which serves no good. Using log-properties it can be evaluated instead of $log(prod(prod(x)))$ as $sum(sum(log(x)))$.

```
initials <- apply(array(sequences), 1, FUN=function(y){strtrim(y,order)})
for(x in 1:length(number_structures)){
  start <- initials[x]
likelihood[x] <- sum(log(alpha[initials[x],]),
    apply(array(1:(number_structures[x]-order)),1,FUN=function(y){
      next_in_chain <- transitions[y]
      transition_likelihood <-
                    log(transition_frequencies[start,next_in_chain])
      start <<- paste(substr(start,2,order),next_in_chain,sep="")
      return(transition_likelihood)}))
```

This gives us the log-likelihood of each read, and the likelihood for the dataset is the sum of `likelihood`.
Originally, the dataset being evaluated contained several sequences of $length <= order$ for $order = 1:8$. To handle sequences of $length = order$, is simple. The likelihood is just the $log(alpha\_probability)$ for that sequence.

```
if(number_structures[x] == order){
      likelihood[x] <- log(alpha[initials[x],])}
```

To handle sequences of $length < order$ was more complex. To be true to the dataset and the task they could not be ignored or tossed out, but they violated the conditions of our model. The decision was to include them with a penalization to their likelihood of occuring. They have no transitions, so the probability was first taken from the alpha matrix as the sum of the probabilities of all possible `alpha` states it could be if it was of $length = order$.

```
if(number_structures[x] < order){
  n <- 0
  while(n == 0){
    initial <- paste(start,"*",sep="")
    start <- strtrim(start,(nchar(start)-1))
    n <- sum(alpha[grepl(glob2rx(initial), rownames(alpha)),])}
```

To penalize this probability it was decided that the sum probability would be divided by the number of structures in the dataset raised to the *order* - the number of states in the starting frequency. So, the total number of structures in the dataset raised to the number of missing states in the sequence.

```
likelihood[x] <- log(n*((1/length(structures))^(order-(nchar(initial)-1))))
```

A smarter way to do this in terms of computational time, for the task of determining the likelihood of the entire data set, would be to simply take the table of transitions and multiply the counts by the *log* of their corresponding transition probabilities, do the same for the starting states and `alpha` and then sum them all. The individual likelihoods for each read will not be evaluated, but for this task that not necessary.

```
P.hat <- melt(P.hat)
P.hat <- data.table(word = paste(P.hat[,1], P.hat[,2], sep = ""),
                    prob = log(P.hat[,3]))
P.hat <- P.hat[prob != -Inf,]

likelihood <- log(P.hat[[3]]) * transitions[[2]]
number_structures <- width(seqs)
sequences <- seqs[which(number_structures < order)]
number_structures <- width(sequences)
short_read_likelihood <- numeric(length=length(number_structures))
if(length(number_structures) > 0){
  for(x in 1:length(number_structures)){
    start <- sequences[x]
    n <- 0
    while(n == 0){
      state <- paste(start,"*",sep="")
      start <- strtrim(start,(nchar(start)-1))
      n <- sum(alpha[grepl(glob2rx(state), rownames(alpha)),])
    }
    short_read_likelihood[x] <-
                log(n*((1/length(structures))^(order-(nchar(state)-1))))
  }
}
likelihood <- sum(sum(likelihood), sum(log(alpha$alpha) * initial[[2]]),
                                                 sum(short_read_likelihood))
```

b. Probability of observing ”CHC”

```
alpha["C",]*P.hat["C","H"]*P.hat["H","C"]
```

```
[1] 0.0011
```

2. Extend the model to an m-order Markov mode

a. Extension to Order 2.

Alpha.

```
    alpha
CB 0.0193
CC 0.8769
CE 0.0647
CG 0.0035
CH 0.0239
CT 0.0117
```

P.hat.

```
       B     C     E     G     H     I     S     T
BB 0.027 0.585 0.009 0.014 0.035 0.000 0.159 0.171
BC 0.026 0.526 0.046 0.027 0.083 0.000 0.138 0.155
BE 0.000 0.000 0.999 0.000 0.001 0.000 0.000 0.000
BG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
BH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
BI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
BS 0.018 0.267 0.090 0.009 0.025 0.000 0.544 0.047
BT 0.010 0.057 0.013 0.009 0.022 0.000 0.024 0.866
CB 0.024 0.622 0.035 0.017 0.051 0.000 0.119 0.131
CC 0.022 0.696 0.070 0.018 0.053 0.000 0.089 0.053
CE 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
CG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
CH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
CI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
CS 0.024 0.262 0.117 0.014 0.048 0.000 0.482 0.053
CT 0.001 0.000 0.000 0.003 0.028 0.000 0.000 0.968
EB 0.007 0.409 0.003 0.012 0.015 0.000 0.352 0.203
EC 0.018 0.481 0.019 0.032 0.048 0.000 0.225 0.178
EE 0.004 0.139 0.770 0.005 0.006 0.000 0.036 0.040
EG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
EH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
EI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
ES 0.016 0.320 0.031 0.014 0.022 0.000 0.540 0.057
ET 0.002 0.013 0.002 0.003 0.016 0.000 0.003 0.961
GB 0.010 0.671 0.010 0.025 0.040 0.000 0.126 0.118
GC 0.036 0.505 0.150 0.018 0.058 0.000 0.164 0.069
GE 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
GG 0.013 0.140 0.035 0.576 0.046 0.000 0.093 0.097
GH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
GI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
GS 0.037 0.510 0.052 0.025 0.064 0.000 0.206 0.106
GT 0.027 0.236 0.097 0.013 0.038 0.000 0.114 0.475
HB 0.009 0.473 0.005 0.017 0.031 0.000 0.176 0.290
HC 0.018 0.510 0.031 0.050 0.075 0.000 0.163 0.152
HE 0.001 0.001 0.991 0.000 0.004 0.000 0.001 0.002
HG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
HH 0.001 0.020 0.001 0.003 0.904 0.000 0.013 0.059
HI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
```

```
HS 0.014 0.486 0.026 0.026 0.039 0.000 0.286 0.122
HT 0.006 0.217 0.008 0.012 0.049 0.000 0.107 0.601
IB 0.000 0.875 0.000 0.000 0.000 0.000 0.118 0.007
IC 0.016 0.395 0.020 0.175 0.244 0.000 0.065 0.084
IE 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
IG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
IH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
II 0.008 0.042 0.001 0.005 0.028 0.768 0.045 0.102
IS 0.001 0.676 0.016 0.015 0.080 0.000 0.138 0.074
IT 0.000 0.257 0.001 0.013 0.089 0.000 0.081 0.559
SB 0.026 0.572 0.048 0.035 0.066 0.000 0.129 0.125
SC 0.041 0.473 0.166 0.031 0.100 0.000 0.116 0.072
SE 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
SG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
SH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
SI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
SS 0.023 0.452 0.086 0.018 0.059 0.000 0.302 0.060
ST 0.003 0.000 0.000 0.003 0.036 0.000 0.000 0.957
TB 0.019 0.663 0.048 0.008 0.022 0.000 0.124 0.117
TC 0.034 0.533 0.156 0.019 0.063 0.000 0.135 0.061
TE 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
TG 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000
TH 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
TI 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
TS 0.030 0.523 0.068 0.019 0.053 0.000 0.224 0.083
TT 0.026 0.346 0.109 0.016 0.053 0.000 0.195 0.254
```

b. Probability of observing ”CHC”

```
alpha["CH",] * P.hat["CH","C"]
```

```
[1] 0
```

3. Estimating Markov chain order

a. Simulation of 1000 chains under order 2 Markov model

To simulate each read, the number of structures in each read is determined, then a starting point chosen by sampling from the *alpha* distribution. The number of transitions is the difference in the number of structures in the read and the *order*. Then each transition is walked through in the `P.hat` matrix.

```
number_structures <- (width(readBStringSet(fasta_file)))
simulated_sequences <- character(length=length(number_structures))
```

This step is essentially the same as determining the likelihoods before, just that instead of reading the next state it is being sampled from `P.hat`.

(This step is very slow... and it seemed everything I tried to do to make it faster ened up doing the opposite. If there is a more efficient way, I'd love to know!)

```
for(x in 1:length(number_structures)){
    if(number_structures[x] < order){
    start <- sample(rownames(alpha), size = 1, prob = alpha$alpha)
    start <- strtrim(start,number_structures[x])
    initial <- paste(start,"*",sep="")
    n <- sum(alpha[grepl(glob2rx(initial), rownames(alpha)),])
    simulated_sequences[x] <- start
  }

  else if(number_structures[x] == order){
    start <- sample(rownames(alpha), size = 1, prob = alpha$alpha)
    simulated_sequences[x] <- start
  }
```

The issue in this step was that as the value of the order increased, the number of unique transition states increased. When one of these occurs at the end of the sequence it creates an absorbing-state of transition probability `NA` or 0. In a situation where the task is simulating x number of reads under the given probabilities this would be a non-issue, the read would terminate upon reaching this state. In this task the reads needed to reach a specified length, which created an issue. Removing these states would not be true to the data. The most true-to-the-data resolution would be to ensure that these transitions can only occur at the end of a read. To manually ensure this is not trivial. If the end transitions are removed and create their own "*omega*" matrix then there become new end-transitions and potentially new absorbing-states (this was tried and happened).

Programmatically, however, the solution is quite simple. Using a `while` loop and `try` the simulator is told to simulate a read to a given length. If it hits an absorbing state before the end of the read it will try to sample from `P.hat` and error because a transition does not exist. The `try` stops the error from cancelling the loop, and the `while` tells it to repeat that process until it creates a read of the specified length. If the absoribing-state is entered at the end of the sequence there is no error, as it will not attempt to transiton out from it. This makes these states possible to exist in the data and ensures that they are always at the end of the sequence if they do.

```
else{
  chain <- character((number_structures[x]-(order-1)))
    while(chain[length(chain)] == ""){
      start <- sample(rownames(alpha), size = 1, prob = alpha$alpha)
      chain[1] <- start
      try(for(y in 1:(number_structures[x]-order)){
        next_in_chain <- sample(structures, size = 1, prob = P.hat[start,])
        chain[(y+1)] <- next_in_chain
        start <- paste(substr(start,2,order),next_in_chain,sep="")
        }, silent = TRUE)}
    simulated_sequences[x] <- paste(chain, collapse = "")
}
}
```

b. Optimal order for simulated data

Optimal order is determined using a pairwise Likelihood-Ratio-Test for each consequent order. This is done by using the log-likelihood for each of the two orders compared, taking twice the difference as our test statistic and then calculating the chi-squared at $\alpha = .05$ and degress of freedom $= 7 * 8^{order^{H_0}+1}$ where H_0: the optimal order is the lesser of the two compared.

```
likelihood_ratio_test<- function(likelihood_test, confidence = 0.95){
  likelihood_test[,
    df := c(NA, apply(array(1:(length(likelihood_test[[2]])-1)), 1,
      FUN=function(i){7*(8^(i+1))}))]
  likelihood_test[, t := c(NA, 2*diff(likelihood_test[[2]]))]
  likelihood_test[,
    chisqr := c(NA, apply(array(2:length(likelihood_test[[2]])), 1,
      FUN=function(i){qchisq(p=confidence, df = 7*(8^(i+1)))}))]
```

If the test statistic (t) is $>$ the chi-squared value, then we reject H_0 and conclude that H_A is a more optimal order. The highest optimal order is taken as the highest order at which reject H_0.

```
return(likelihood_test[,
  reject_H0 := array(apply(likelihood_test,1,
    FUN=function(i){i[[4]] > i [[5]]}))])
```

```
   order        LL        df     t     chisqr reject_H0
1:     1 -223184.4        NA    NA         NA        NA
2:     2 -209330.6       448 27708        498      TRUE
3:     3 -209007.6      3584   646       3724     FALSE
4:     4 -207922.5     28672  2170      29067     FALSE
5:     5 -204823.4    229376  6198     230491     FALSE
6:     6 -198147.7   1835008 13351    1838160     FALSE
7:     7 -186065.2  14680064 24165   14688978     FALSE
8:     8 -169214.8 117440512 33701  117465722     FALSE
```

Here it is seen that at $\alpha = .05$ when $H_0 : order = 1$ we reject H_0 (`TRUE`) and accept $H_A : order = 2$. Then, when $H_0 : order = 2$ we fail to reject H_0 (`FALSE`) and conclude that 2 is the optimal order for the set of 1,000 simulated reads.

c. Optimal order for pdb.fa

```
   order        LL        df        t    chisqr reject_H0
1:     1 -90062831        NA       NA        NA        NA
2:     2 -84452092       448 11221477       498      TRUE
3:     3 -81479198      3584  5945788      3724      TRUE
4:     4 -80113917     28672  2730562     29067      TRUE
5:     5 -79083798    229376  2060239    230491      TRUE
6:     6 -78045823   1835008  2075949   1838160      TRUE
7:     7 -76795201  14680064  2501245  14688978     FALSE
8:     8 -75056752 117440512  3476897 117465722     FALSE
```

Here we find that at $\alpha = .05$, $order = 6$ is the optimal order for the model of the `pdb.fa` data set.

4. Analysis

The conclusion from the LRT for the `pdb.fa` data set was that there is statistically significant evidence that there is dependence betweeen secondary structure states along protein sequences, in this case shown to be dependence on the prior 6 structures.

I am no so certain that the Markov Chain Model is appropriate for this analysis. Without being an expert on the subject, it seems that it does not account for the evolutionary process. I think evolution and natural selection play a large role in both protein and DNA structure order that cannot always be adequately modeled in the absence of some sort of function information. I think more information about the outcome of what each structure series means would help to provide more support to whether or not the Markov Chain Model is appropriate for this analysis or not.