Project 3

Richard Affolter

11 3 2021

Problem 6: Hidden Markov Models

According to slide 18 the HMM is parametrized by

- The initial state probabilities, which can take up K different values, but since they have to sum up to 1 (they are probabilities) we have K-1 degrees of freedom.
- The transition probabilities, where we have aa K·K matrix, but since total the probability of jumping from a state i into a state j must sum up to 1, we have K·(K-1) degrees of freedom.
- The emission probabilities, where we have K·M matrix, but since the probability of emitting something (from any state i) must sum up to 1 we have K·(M-1) degrees of freedom.

In total we have K-1 + K·(K-1) + K·(M-1) = K·(K+M-1)-1 possible unique parameters.

Problem 7: Predicting protein secondary structure using HMMs

```
library(dplyr)
```

(a) Read `proteins_train.tsv`, `proteins_test.tsv` and `proteins_new.tsv` into the memory and store each in a data.frame

```
p_train = read.table(file = 'proteins_train.tsv', sep = '\t', header = FALSE)
p_test = read.table(file = 'proteins_test.tsv', sep = '\t', header = FALSE)
p_new = read.table(file = 'proteins_new.tsv2', sep = '\t', header = FALSE)
```

(b) Estimate the vector of initial state probabilities I, the matrix of transition probabilities T and the matrix for emission probabilities E.

```
# copy section from viterbi.r
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_I = function(df){
  I = rep(0, length(unique.ss))
  names(I) = unique.ss

  for(i in 1:dim(p_train)[1]){
    letter = substr(df[i,3],1,1)
    I[letter] = I[letter] + 1
  }

  return(I = I/sum(I))
}
I = get_I(p_train)
```

```r
get_T = function(df){
  T = matrix(0,nrow = length(unique.ss), ncol = length(unique.ss))
  rownames(T) = unique.ss
  colnames(T) = unique.ss

  for(i in 1:dim(df)[1]){
    sec_struc = strsplit(df[i,3], split = "") %>% unlist()
    for(j in 1:(length(sec_struc)-1)){
      T[sec_struc[j],sec_struc[j+1]] = T[sec_struc[j],sec_struc[j+1]] + 1
    }
  }
  return(T = T/rowSums(T))
}
T = get_T(p_train)
```

```r
get_E = function(df){
  E = matrix(0,nrow = length(unique.ss), ncol = length(unique.aa))
  rownames(E) = unique.ss
  colnames(E) = unique.aa

  for(i in 1:dim(df)[1]){
    sec_struc = strsplit(df[i,3], split = "") %>% unlist()
    aa_seq = strsplit(df[i,2], split = "") %>% unlist()
    for(j in 1:length(sec_struc)){
      E[sec_struc[j],aa_seq[j]] = E[sec_struc[j],aa_seq[j]] + 1
    }
  }
  return(E = E/rowSums(E))
}
E = get_E(p_train)
```

```r
print(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
```

```
print(T)
```

```
##              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
```

```
print(I)
```

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

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

We know from slide 9 that π is the solution of $\pi^t = \pi^t T$. If we transpose we get an eigenvalue problem:

$$(\pi^t)^\intercal = (\pi^t T)^\intercal = T^\intercal (\pi^t)^\intercal$$

So we are looking for the eigenvector of $T^\intercal$ with eigenvalue 1.

```
library(dplyr)
ev = eigen(t(T))
# find the eigenvalue with value 1
pos = which(near(ev$values,1))

#normalize
Pi = ev$vectors[,pos]/sum(ev$vectors[,pos])
names(Pi) = unique.ss
```

```
print(Pi)
```

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

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

```
source("viterbi.r")
get_prediction = function(x){
  df = data.frame(AminoAcids = x$V2)
  tmp = viterbi(E=log(E), Tr = log(T), I=log(I),p=df)
  return(tmp$PredictedStructure)
}

p_train$PredictedStructure = get_prediction(p_train)
p_test$PredictedStructure = get_prediction(p_test)
p_new$PredictedStructure = get_prediction(p_new)
```

```
#  save p_new as a new tsv
write.table(p_new, file = "proteins_new.tsv", sep = "\t",
            col.names = FALSE, row.names = FALSE)
```

Estimate confidence intervals for I, E and T with bootstrapping

```r
# initialize lists to store the intermediate results of each bootstrap
I_list = list()
T_list = list()
E_list = list()

num_row = dim(p_train)[1]
num_boot = 1000
set.seed(123)
# resample the data and get T,E,I
for(i in 1:num_boot){
  resample_idx = sample(num_row,num_row, replace = TRUE)
  resampled_data = p_train[resample_idx,]
  I_list[[i]] = get_I(resampled_data)
  T_list[[i]] = get_T(resampled_data)
  E_list[[i]] = get_E(resampled_data)
}

# transform to 3D array
T_simp = simplify2array(T_list)
E_simp = simplify2array(E_list)
I_simp = simplify2array(I_list)
```

```r
# get CI for T
T_simp[is.na(T_simp)] = 0 # set Na to 0
dim1 = dim(T_simp)[1]
dim2 = dim(T_simp)[2]
T_CI = array(rep(0,dim1*dim2*2),dim = c(dim1,dim2,2),
             dimnames = list(unique.ss, unique.ss,
                             c("lower_bound", "upper_bound")))

for(i in 1:dim1){
  for(j in 1:dim2){
    T_simp[i,j,] %>% quantile(probs = c(0.025, 0.975))-> CI
    T_CI[i,j,] = CI
  }
}


# get CI for E
E_simp[is.na(E_simp)] = 0 # set Na to 0
dim1 = dim(E_simp)[1]
dim2 = dim(E_simp)[2]
E_CI = array(rep(0,dim1*dim2*2),dim = c(dim1,dim2,2),
             dimnames = list(unique.ss, unique.aa,
                             c("lower_bound", "upper_bound")))

for(i in 1:dim1){
  for(j in 1:dim2){
    E_simp[i,j,] %>% quantile(probs = c(0.025, 0.975))-> CI
    E_CI[i,j,] = CI
  }
}
```

```r
# get CI for I
I_simp[is.na(I_simp)] = 0 # set Na to 0
dim1 = dim(I_simp)[1]
I_CI = array(rep(0,dim1*2),dim = c(dim1,2),
             dimnames = list(unique.ss,
                             c("lower_bound", "upper_bound")))

for(i in 1:dim1){
  I_simp[i,] %>% quantile(probs = c(0.025, 0.975))-> CI
  I_CI[i,] = CI
}
```

```r
print(E_CI)
```

```
## , , lower_bound
##
##            A          C          D          E          F          G          H
## B 0.03676734 0.02501236 0.05549371 0.01775480 0.03314254 0.04206863 0.02512017
## C 0.06125213 0.01428884 0.06958666 0.04586379 0.03226412 0.07930661 0.02372127
## E 0.05538309 0.02144143 0.02606907 0.03793699 0.04960879 0.04535426 0.02554873
## G 0.09963131 0.01318561 0.08923075 0.07649882 0.03067724 0.05676809 0.02613994
## H 0.12167461 0.01100334 0.05105861 0.08344614 0.03466675 0.03347175 0.01995728
## I 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## S 0.05081113 0.01319639 0.08112051 0.04814871 0.02092145 0.13565800 0.02025480
## T 0.05978333 0.01014689 0.07275748 0.05661756 0.02123467 0.19302044 0.01887476
##            I          K          L          M          N          P          Q
## B 0.05232129 0.03603897 0.06275400 0.01022830 0.04433873 0.03317499 0.02226881
## C 0.04124922 0.05773465 0.06721365 0.02170903 0.05397838 0.07913326 0.02944562
## E 0.08867271 0.05557262 0.10196712 0.02198602 0.02218738 0.01636054 0.02926314
## G 0.02710329 0.05366198 0.05232252 0.01201092 0.04668541 0.05900526 0.03148822
## H 0.05295297 0.07035946 0.10948208 0.02842216 0.03471894 0.01805097 0.03960625
## I 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## S 0.02481477 0.06735327 0.04730520 0.01302713 0.05048363 0.05451695 0.02813621
## T 0.02262409 0.07363529 0.04648344 0.01149246 0.06386166 0.06415254 0.03419772
##            R          S          T U          V           W            X
## B 0.03725310 0.03699540 0.06693646 0 0.11547757 0.004770939 0.0000000000
## C 0.04498770 0.07666992 0.06883841 0 0.04867268 0.007144053 0.0004768585
## E 0.03607158 0.04854637 0.07424088 0 0.12161808 0.018120399 0.0000000000
## G 0.03435139 0.06248685 0.03754746 0 0.02038358 0.020866395 0.0000000000
## H 0.06171792 0.03857053 0.04207148 0 0.06998260 0.012151680 0.0000000000
## I 0.00000000 0.00000000 0.00000000 0 0.00000000 0.000000000 0.0000000000
## S 0.05022786 0.08326547 0.06822848 0 0.03591667 0.014248324 0.0000000000
## T 0.03832699 0.05964353 0.04122727 0 0.01792738 0.008732325 0.0000000000
##            Y
## B 0.05426232
## C 0.02604410
## E 0.05244950
## G 0.04352208
## H 0.02792216
## I 0.00000000
## S 0.02199734
## T 0.02224705
##
```

```
## , , upper_bound
##
##            A          C          D          E          F          G          H
## B 0.05338880 0.04054847 0.07744943 0.03173209 0.05120859 0.06244158 0.03988036
## C 0.06743797 0.01763516 0.07527926 0.05189973 0.03574099 0.08597072 0.02785935
## E 0.06043838 0.02542578 0.02992468 0.04217873 0.05509939 0.05099863 0.02958970
## G 0.11741205 0.02063220 0.10447597 0.09037730 0.03976357 0.06905628 0.03480110
## H 0.12851502 0.01400158 0.05408677 0.08898454 0.03816065 0.03668935 0.02381554
## I 0.60000000 0.00000000 0.50000000 0.00000000 0.00000000 0.00000000 0.00000000
## S 0.05805143 0.01789614 0.09095919 0.05625398 0.02573186 0.14827598 0.02586826
## T 0.06705055 0.01332921 0.08174258 0.06396974 0.02584296 0.20551706 0.02368178
##            I          K          L          M          N          P          Q
## B 0.07277206 0.05541342 0.08767385 0.02266456 0.06517451 0.05056293 0.03764080
## C 0.04528141 0.06464560 0.07379133 0.02469485 0.06014208 0.08646788 0.03321223
## E 0.09796524 0.06009084 0.10854246 0.02564845 0.02578899 0.01924704 0.03347403
## G 0.03638378 0.06544153 0.06442876 0.01806227 0.05800521 0.07037314 0.04007322
## H 0.05687719 0.07552702 0.11605927 0.03117403 0.03810296 0.02057890 0.04301275
## I 0.20000000 0.00000000 0.33333333 0.20000000 0.00000000 0.00000000 0.00000000
## S 0.03056803 0.07686204 0.05404388 0.01696573 0.05901032 0.06112906 0.03371460
## T 0.02704710 0.08432300 0.05324485 0.01474431 0.07107599 0.07184491 0.03966188
##            R          S          T            U          V           W
## B 0.05685309 0.05553008 0.09504234 0.0000000000 0.14680182 0.012967310
## C 0.04959482 0.08265856 0.07496502 0.0000000000 0.05368082 0.009128129
## E 0.04039217 0.05533727 0.08117557 0.0000000000 0.13219682 0.021265092
## G 0.04378283 0.07699310 0.04835776 0.0005487835 0.02837790 0.028535449
## H 0.06671812 0.04227937 0.04514618 0.0000000000 0.07378659 0.013944385
## I 0.33333333 0.00000000 0.00000000 0.0000000000 0.16666667 0.166666667
## S 0.05716230 0.09302501 0.07612611 0.0000000000 0.04290983 0.018112361
## T 0.04413946 0.06884536 0.04745501 0.0000000000 0.02256677 0.011562781
##              X          Y
## B 0.0037716089 0.07700235
## C 0.0015825504 0.02948324
## E 0.0000000000 0.05777777
## G 0.0000000000 0.05440181
## H 0.0000000000 0.03137812
## I 0.0000000000 0.00000000
## S 0.0008517397 0.02715780
## T 0.0003410375 0.02654303
```

```
print(T_CI)
```

```
## , , lower_bound
##
##              B          C           E           G          H I          S
## B 0.0131209661 0.58817265 0.023168872 0.014354844 0.01771243 0 0.14038454
## C 0.0267288300 0.50355529 0.106102171 0.023566626 0.07953953 0 0.13329759
## E 0.0033557647 0.10410791 0.808187452 0.003772679 0.00454561 0 0.02735424
## G 0.0055202089 0.10577411 0.015581954 0.694726134 0.02909551 0 0.05208150
## H 0.0002690861 0.01689645 0.000145162 0.002465372 0.90902863 0 0.01504865
## I 0.0000000000 0.00000000 0.000000000 0.000000000 0.00000000 0 0.00000000
## S 0.0233965113 0.37482405 0.080840100 0.016019008 0.06281659 0 0.34764771
## T 0.0159324461 0.22088556 0.065791987 0.011470128 0.03716011 0 0.11576631
##            T
## B 0.12850389
## C 0.09204814
## E 0.03416854
## G 0.06602872
## H 0.05013353
## I 0.00000000
## S 0.05915445
## T 0.50739794
##
## , , upper_bound
##
##              B          C            E           G           H            I
## B 0.0241048387 0.62930731 0.0379940489 0.027510009 0.030259866 0.000000e+00
## C 0.0309498596 0.52694309 0.1165231071 0.026796572 0.090386222 0.000000e+00
## E 0.0044812182 0.11190271 0.8177724995 0.005350422 0.006714566 0.000000e+00
## G 0.0102427718 0.11889623 0.0222025853 0.701004829 0.037917849 0.000000e+00
## H 0.0006234217 0.01882506 0.0004780594 0.003258948 0.911855259 5.432982e-05
## I 0.0000000000 0.00000000 0.0000000000 0.000000000 0.166666667 8.333333e-01
## S 0.0285827289 0.38913790 0.0908119963 0.020477113 0.072436622 2.078243e-04
## T 0.0197666177 0.23318842 0.0735156743 0.014372511 0.043152108 1.631235e-04
##            S          T
## B 0.16800825 0.16130848
## C 0.14456346 0.10007656
## E 0.03133049 0.03772822
## G 0.06385304 0.07908116
## H 0.01736322 0.05343563
## I 0.00000000 0.20000000
## S 0.36516889 0.06769956
## T 0.12533618 0.51600705
```

```
print(I_CI)
```

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

(f) Compute the accuracy of the predicted secondary structure for the data.frame of proteins_test.tsv

```
accuracies = rep(0, nrow(p_test))
for(i in 1:nrow(p_test)){
  for(j in 1:nchar(p_test$V3[i])){
    if(substr(p_test$V3[i],j,j) == substr(p_test$PredictedStructure[i],j,j)){
      accuracies[i] = accuracies[i]+1
    }
  }
  accuracies[i] = accuracies[i]/nchar(p_test$V3[i])
}
summary(accuracies)
```

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

The mean accuracy over all sequences is 0.3198012.

(g) Randomly guess secondary structures and compare to the Viterbi

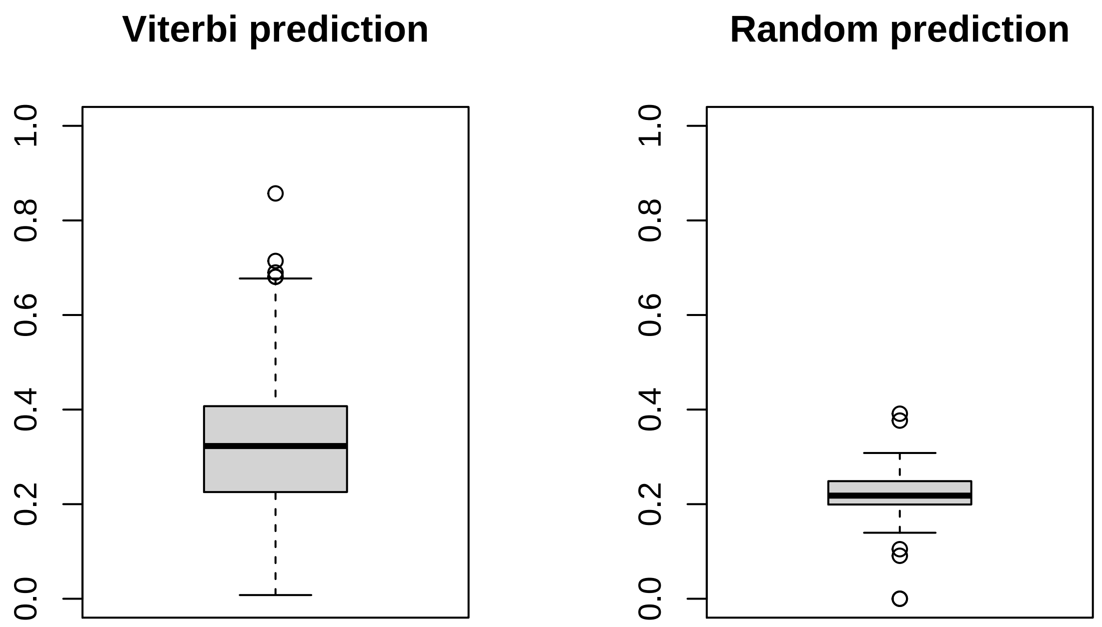
For the probability of the random guessing the stationary distribution is used.

```
random_accuracies = rep(0, nrow(p_test))
for(i in 1:nrow(p_test)){
  random_structure = sample(unique.ss, size=nchar(p_test$V3[i]),
                            replace = TRUE,prob = Pi)
  random_structure = paste(random_structure, collapse = "")
  for(j in 1:nchar(p_test$V3[i])){
    if(substr(p_test$V3[i],j,j) == substr(random_structure,j,j)){
      random_accuracies[i] = random_accuracies[i]+1
    }
  }
  random_accuracies[i] = random_accuracies[i]/nchar(p_test$V3[i])
}
summary(random_accuracies)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.0000  0.1994  0.2180  0.2182  0.2484  0.3913
```

The mean accuracy of the random sequences over all sequences is 0.2181645.

```
par(mfrow=c(1,2))
boxplot(accuracies, main = "Viterbi prediction", ylim = c(0, 1))
boxplot(random_accuracies, main = "Random prediction", ylim = c(0,1))
```



We can see that the Viterbi algorithm performs better than random guessing.