Project 3. Viterbi algorithm

March 10, 2024

Contents


```
suppressPackageStartupMessages(library(dplyr))
set.seed(42)
```

```
DSSP <- c("B", "C", "E", "G", "H", "I", "S", "T")
AA <- c("A", "C", "D", "E", "F", "G", "H", "I", "K",
    "L", "M", "N", "P", "Q", "R", "S", "T", "U",
    "V", "W", "X", "Y"
)
data_folder <- "../data/"
```

```
#' @param E n times m matrix with the emission log probabilities of the n latent variables and m observ
#' @param Tr n times n matrix with the transition log probabilities
#' @param I numeric vector of length n with the initial log probabilities of the n latent variables
#' @param p a data.frame with a Variable named AminoAcids which holds the amino acid sequence as a char
viterbi <- function(E, Tr, I, p) {
    .as.array <- 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")
    for (k in seq(nrow(p))) {
        sequence <- p$AminoAcids[k]
        aa.vec <- .as.array(sequence) %>% match(unique.aa)
        P      <- matrix(0, nrow(E), length(aa.vec))
        Ptr    <- matrix(0, nrow(E), length(aa.vec))

        ## sets the paths
        for (i in seq(length(aa.vec))) {
            if (i == 1) {
                P[, i] <- I + E[, aa.vec[i]]
            } else {
                for (j in seq(nrow(E))) {
                    p.loc    <- P[, i - 1] + Tr[, j] + E[j, aa.vec[i]]
                    P[j, i] <- max(p.loc)
                    Ptr[j, i] <- which.max(p.loc)
                }
            }
        }

        ## backtrace: computes the most likely path
        Phi <- vector(mode="integer",   length=length(aa.vec))
        Phi[length(Phi)] <- which.max(P[, ncol(P)])
```

```
        ## we start at the back, just as with Needleman-Wunsch or Smith-Waterman
        for (i in seq(from=length(aa.vec), to=2)) {
            Phi[i - 1] <- Ptr[Phi[i], i]
        }

        states <- unique.ss[Phi]
        p$PredictedStructure[k] <- paste(states, collapse="")
    }
    return(p)
}

loading_data <- function(data_file, sec_struct) {
    col_names <- c('SeqID', 'AminoAcids')
    if(sec_struct) { col_names <- append(col_names, 'SecondaryStructure')}
    data_df <- read.csv(data_file, header=FALSE, sep="\t")
    names(data_df) <- col_names
    return(data_df)
}

assign_row_names <- function(M) {
    row.names(M) <- DSSP
    return(M)
}

assign_col_names <- function(M, aa=TRUE) {
    if (aa) {
        colnames(M) <- AA
    } else {
        colnames(M) <- DSSP
    }

    return(M)
}

init_matrices <- function(data) {
    # Initialisation of I, T, and E matrices using maximum likelihood

    # I - 8 x 1 matrix - initial state probabilities
    I <- matrix(0, nrow=length(DSSP), ncol=1)
    # T - 8 x 8 matrix - transition probabilities
    T <- matrix(0, nrow=length(DSSP), ncol=length(DSSP))
    # E - 8 x 22 matrix - emission probabilities
    E <- matrix(0, nrow=length(DSSP), ncol=length(AA))

    I <- assign_row_names(I)
    T <- assign_row_names(T)
    E <- assign_row_names(E)

    T <- assign_col_names(T, aa=FALSE)
    E <- assign_col_names(E)

    I <- initial_states(data, I)
    T <- transition_states(data, T)
```

```
    E <- emission_states(data, E)

    # TODO: understand, why we need to take log of matrices?
    I <- log(I)
    T <- log(T)
    E <- log(E)

    return(list(I=I, T=T, E=E))
}

initial_states <- function(data, M) {
    # Retrieving DSSP profiles
    sec_struct <- data$SecondaryStructure

    # Computing frequencies of each letter in the first position
    frequency <- table(substr(sec_struct, 1, 1))

    # Computing relative frequencies
    M[row.names(M) %in% names(frequency), 1] <- as.numeric(frequency)/sum(frequency)

    # Return initial states matrix
    return(M)
}

transition_states <- function(data, M) {
    # Retrieving DSSP profiles
    sec_struct <- data$SecondaryStructure

    pairs <- lapply(sec_struct,
        function(x) substring(x, first=1:(nchar(x)-1), last=2:nchar(x))
    )
    frequencies <- table(unlist(pairs))

    # Assigning frequencies
    for (i in seq(nrow(M))) {
        for (j in seq(ncol(M))) {
            row_name <- row.names(M)[i]
            col_name <- colnames(M)[j]
            freq_name <- paste0(row_name, col_name)
            if (is.na(frequencies[freq_name])) {
                M[i, j] <- 1
            } else {
                M[i, j] <- as.numeric(frequencies[freq_name])+1
            }
        }
    }
    # Relative frequencies
    M <- M/rowSums(M)

    # Return transition states matrix
    return(M)
}
```

```
emission_states <- function(data, M) {
    # Retrieving profiles
    aa <- data$AminoAcids
    dssp <- data$SecondaryStructure

    aa_list <- lapply(aa,
        function(x) substring(x, first=1:(nchar(x)), last=1:nchar(x))
    )
    dssp_list <- lapply(dssp,
        function(x) substring(x, first=1:(nchar(x)), last=1:nchar(x))
    )

    frequencies <- table(paste(unlist(dssp_list), unlist(aa_list), sep=""))

    # Assigning frequencies
    for (i in seq(nrow(M))) {
        for (j in seq(ncol(M))) {
            row_name <- row.names(M)[i]
            col_name <- colnames(M)[j]
            freq_name <- paste0(row_name, col_name)
            if (is.na(frequencies[freq_name])) {
                M[i, j] <- 1
            } else {
                M[i, j] <- as.numeric(frequencies[freq_name])+1
            }
        }
    }
    # Relative frequencies
    M <- M/rowSums(M)

    # Return initial states matrix
    return(M)
}

eigen_stationary_distribution <- function(T) {
    # T transposed to get left eigenvectors of T
    e <- eigen(t(exp(T)))
    e_vec <- e$vectors
    e_val <- e$values
    e_val_one_index <- which(abs(e_val-1) < 1e-12)
    stat_dist <- matrix(e_vec[, e_val_one_index]/
        sum(e_vec[, e_val_one_index]), nrow=1, ncol=length(DSSP)
    )
    colnames(stat_dist) <- DSSP
    print("Stationary distribution of transmission probabilities (eigenvalue approach): ")
    print(stat_dist)
    return(stat_dist)
}

brute_force_stationary_distribution <- function(T) {
    T <- exp(T)
    while (sum(abs(T[1,]-T[2,])) > 1e-12) {
        T <- T %*% T
```

```
    }
    print("Stationary distribution of transmission probabilities (brute-force approach): ")
    print(T[1,])
}

run_viterbi_predictions <- function(data, params) {
    pred_df <- data.frame(AminoAcids=data$AminoAcids, PredictedStructure=NA)
    pred_df <- apply(pred_df, 1, function(row) {
        viterbi(params$E, params$T, params$I, data.frame(AminoAcids=row["AminoAcids"]))
    })
    pred_df <- data.frame(t(sapply(pred_df, function(x) x[1:max(lengths(pred_df))])))
    data$PredictedStructure <- pred_df$PredictedStructure
    data <- apply(data, 2, as.character)
    return(data)
}

run_random_predictions <- function(data, params) {
    rand_struct <- apply(data, 1, function(row) {
        paste0(sample(DSSP, nchar(row["AminoAcids"]), replace=TRUE), collapse="")
    })
    data <- cbind(data, RandomStructure=rand_struct)
    return(data)
}

save_to_tsv <- function(data, file_path) {
    write.table(data, file=file_path, sep="\t", row.names=FALSE, col.names=FALSE)
}

get_accuracies <- function(pred_df, column_name) {
    accuracies <- apply(pred_df, 1, function(row) {
        sum(
            strsplit(row["SecondaryStructure"], "")[[1]] == strsplit(row[column_name], "")[[1]])/
            nchar(row["SecondaryStructure"])
    })
    print(summary(accuracies))
    return(accuracies)
}

get_bootstrapped_data <- function(data) {
    bootstrapped_data <- slice_sample(data, n=nrow(data), replace=TRUE)
    return(bootstrapped_data)
}

process_bootstrap_params <- function(boot_params, params, n_boot) {
    boot_params <- array(unlist(boot_params), dim=c(dim(params), n_boot))
    dimnames(boot_params) <- list(rownames(params), colnames(params), NULL)
    return(boot_params)
}

compute_CI <- function(boot_params) {
    n_values <- dim(boot_params)[1] * dim(boot_params)[2]
    lower_CI <- matrix(NA, nrow=dim(boot_params)[1], ncol=dim(boot_params)[2])
    upper_CI <- matrix(NA, nrow=dim(boot_params)[1], ncol=dim(boot_params)[2])
```

```
    mapply(function(i) {
        mapply(function(i, j) {
            values <- boot_params[i, j, ]
            lower <- quantile(values, 0.025)
            upper <- quantile(values, 0.975)
            lower_CI[i, j] <<- lower
            upper_CI[i, j] <<- upper
        }, i, 1:dim(boot_params)[2])
    }, 1:dim(boot_params)[1])

    # Collecting CIs of parameters in the diagonal
    CI <- matrix(NA, nrow=dim(lower_CI)[1], ncol=2)
    if(dim(lower_CI)[2] == 1) {
        CI[, 1] <- lower_CI
        CI[, 2] <- upper_CI
    } else {
        mapply(function(i) {
                CI[i, 1] <<- lower_CI[i, i]
                CI[i, 2] <<- upper_CI[i, i]
            }, 1:dim(lower_CI)[1]
        )
    }

    # Returning diagonal parameters' CIs
    # and separately lower and upper bounds for CIs
    # of all parameters in the matrix
    return(list(diag_param_CI=CI, lower_CI=lower_CI, upper_CI=upper_CI))
}
```

```
# Loading data
prot_train_df <- loading_data(paste0(data_folder, "proteins_train.tsv"), sec_struct=TRUE)
prot_test_df <- loading_data(paste0(data_folder, "proteins_test.tsv"), sec_struct=TRUE)
prot_new_df <- loading_data(paste0(data_folder, "proteins_new.tsv"), sec_struct=FALSE)
```

```
# Initialisation of parameters
params <- init_matrices(prot_train_df)
```

```
# Stationary distribution
eigen_stationary_distribution(params$T)
```

```
## [1] "Stationary distribution of transmission probabilities (eigenvalue approach): "
##               B         C         E          G         H            I          S         T
## [1,] 0.01169951 0.2041994 0.2094351 0.03433775 0.3394477 0.0001497634 0.08894147 0.1117894
```

```
##               B         C         E          G         H            I          S         T
## [1,] 0.01169951 0.2041994 0.2094351 0.03433775 0.3394477 0.0001497634 0.08894147 0.1117894
```

```
brute_force_stationary_distribution(params$T)
```

```
## [1] "Stationary distribution of transmission probabilities (brute-force approach): "
##            B            C            E            G            H            I            S
## 0.0116995148 0.2041993650 0.2094350926 0.0343377464 0.3394476727 0.0001497634 0.0889414725 0.1117893
```

```
# Making predictions
prot_test_df <- run_viterbi_predictions(prot_test_df, params)
prot_new_df <- run_viterbi_predictions(prot_new_df, params)
```

```r
# Saving predictions
save_to_tsv(prot_new_df, paste0(data_folder, "proteins_new_pred.tsv"))

# Computing prediction accuracies
print("Accuracy statistics of Viterbi predictions:")
```

```
## [1] "Accuracy statistics of Viterbi predictions:"
```

```r
viterbi_acc <- get_accuracies(prot_test_df, column_name="PredictedStructure")
```

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

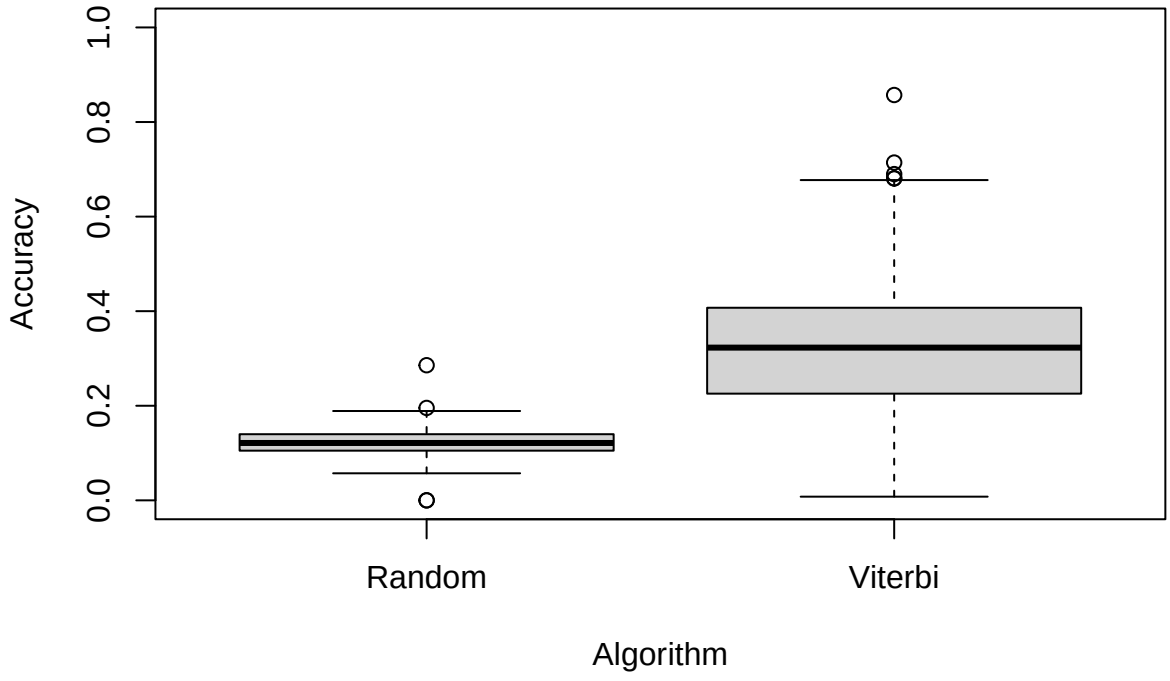
```r
# Making random "predictions"
prot_test_df <- run_random_predictions(prot_test_df, params)
print("Accuracy statistics of random 'predictions':")
```

```
## [1] "Accuracy statistics of random 'predictions':"
```

```r
rand_acc <- get_accuracies(prot_test_df, column_name="RandomStructure")
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1051  0.1213  0.1225  0.1398  0.2857
```

```r
# Plot box plot from accuracies
boxplot(rand_acc, viterbi_acc, ylim=c(0, 1), ylab="Accuracy", xlab="Algorithm",
    names=c("Random", "Viterbi")
)
```



```r
# Bootstrapping for parameter CFs
# TODO: change the number of bootstraps
N_BOOT <- 1000
boot_params <- list(I=list(), T=list(), E=list())

for (i in 1:N_BOOT) {
    boot_data <- get_bootstrapped_data(prot_train_df)
    params <- init_matrices(boot_data)

    boot_params$I <- list(boot_params$I, params$I)
    boot_params$T <- list(boot_params$T, params$T)
    boot_params$E <- list(boot_params$E, params$E)
}

boot_params$I <- process_bootstrap_params(boot_params$I, params$I, N_BOOT)
boot_params$T <- process_bootstrap_params(boot_params$T, params$T, N_BOOT)
boot_params$E <- process_bootstrap_params(boot_params$E, params$E, N_BOOT)

# Computing confidence intervals for parameter matrices
CI_I <- compute_CI(exp(boot_params$I))
CI_T <- compute_CI(exp(boot_params$T))
CI_E <- compute_CI(exp(boot_params$E))
print(CI_I)
```

```
## $diag_param_CI
##      [,1] [,2]
## [1,]    0    0
## [2,]    1    1
## [3,]    0    0
## [4,]    0    0
## [5,]    0    0
## [6,]    0    0
## [7,]    0    0
## [8,]    0    0
##
## $lower_CI
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    0
## [4,]    0
## [5,]    0
## [6,]    0
## [7,]    0
## [8,]    0
##
## $upper_CI
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    0
## [4,]    0
## [5,]    0
## [6,]    0
```

```
## [7,]    0
## [8,]    0
```

```
print(CI_T)
```

```
## $diag_param_CI
##           [,1]       [,2]
## [1,] 0.01349827 0.02423141
## [2,] 0.50389957 0.52696292
## [3,] 0.80787624 0.81686268
## [4,] 0.69401297 0.70014616
## [5,] 0.90893740 0.91188854
## [6,] 0.12500000 0.70212766
## [7,] 0.34801181 0.36505423
## [8,] 0.50743452 0.51588569
##
## $lower_CI
##              [,1]       [,2]         [,3]        [,4]        [,5]         [,6]       [,7]       [,8]
## [1,] 0.0134982734 0.58701186 0.0237314560 0.014400147 0.017480271 4.748338e-04 0.14038477 0.12893136
## [2,] 0.0269477683 0.50389957 0.1064581173 0.023543216 0.079508750 2.787212e-05 0.13330748 0.09223128
## [3,] 0.0033781548 0.10454340 0.8078762441 0.003828153 0.004688498 2.679735e-05 0.02753708 0.03417253
## [4,] 0.0057890817 0.10531532 0.0156064244 0.694012966 0.029126870 1.629979e-04 0.05239020 0.06658213
## [5,] 0.0002925743 0.01685508 0.0001602061 0.002441762 0.908937401 1.688962e-05 0.01508257 0.05022219
## [6,] 0.0217391304 0.02173913 0.0217391304 0.021739130 0.032258065 1.250000e-01 0.02173913 0.05000000
## [7,] 0.0235464082 0.37518413 0.0804200842 0.016082606 0.062181349 6.444462e-05 0.34801181 0.05981712
## [8,] 0.0161112900 0.22102575 0.0653281945 0.011469813 0.037293501 5.176982e-05 0.11603298 0.50743452
##
## $upper_CI
##              [,1]       [,2]         [,3]        [,4]        [,5]         [,6]       [,7]       [,8]
## [1,] 0.0242314124 0.62575667 0.0381929329 0.028054862 0.030711656 5.534034e-04 0.16851907 0.15971915
## [2,] 0.0309028072 0.52696292 0.1161312453 0.026786310 0.090287027 3.096107e-05 0.14476947 0.09993996
## [3,] 0.0045179585 0.11195041 0.8168626797 0.005300806 0.006743756 3.069935e-05 0.03138037 0.03794007
## [4,] 0.0102299548 0.11836590 0.0222227756 0.700146159 0.037708404 1.869884e-04 0.06331324 0.07932027
## [5,] 0.0006389966 0.01883349 0.0005058731 0.003267694 0.911888538 7.212656e-05 0.01744083 0.05347118
## [6,] 0.1250000000 0.12500000 0.1250000000 0.125000000 0.150000000 7.021277e-01 0.12500000 0.17142857
## [7,] 0.0287381018 0.38958530 0.0912683838 0.020550860 0.072803174 2.775851e-04 0.36505423 0.06777946
## [8,] 0.0200176087 0.23348328 0.0734940954 0.014360291 0.043270748 2.189784e-04 0.12511266 0.51588569
```

```
print(CI_E)
```

```
## $diag_param_CI
##           [,1]       [,2]
## [1,] 0.03649427 0.05406883
## [2,] 0.01424455 0.01751170
## [3,] 0.02614289 0.02964338
## [4,] 0.07570073 0.08985016
## [5,] 0.03488724 0.03826327
## [6,] 0.01666667 0.04545455
## [7,] 0.02054714 0.02571667
## [8,] 0.02266594 0.02687818
##
## $lower_CI
##           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]       [,8]       [
## [1,] 0.03649427 0.02557546 0.05480050 0.01785694 0.03369594 0.04170958 0.02525296 0.05206276 0.03668
## [2,] 0.06158530 0.01424455 0.06945877 0.04607720 0.03217176 0.07912638 0.02384385 0.04130862 0.05753
```

```
## [3,] 0.05521285 0.02169162 0.02614289 0.03793854 0.04970949 0.04531599 0.02558821 0.08851788 0.05543
## [4,] 0.09921299 0.01337151 0.08932105 0.07570073 0.03013586 0.05720275 0.02617671 0.02740374 0.05325
## [5,] 0.12150744 0.01099009 0.05096491 0.08373449 0.03488724 0.03334553 0.02002470 0.05291752 0.07028
## [6,] 0.02941176 0.01666667 0.02222222 0.01666667 0.01666667 0.01666667 0.01666667 0.02173913 0.01666
## [7,] 0.05086419 0.01301568 0.08120260 0.04812489 0.02081130 0.13617615 0.02054714 0.02483991 0.06801
## [8,] 0.06004591 0.01028668 0.07307416 0.05651190 0.02122520 0.19308419 0.01895803 0.02266594 0.07426
##           [,10]      [,11]      [,12]      [,13]      [,14]      [,15]      [,16]      [,17]
## [1,] 0.06290486 0.01077877 0.04430371 0.03439350 0.02263273 0.03769675 0.03663811 0.06614006 4.71698
## [2,] 0.06704022 0.02167431 0.05390784 0.07910986 0.02949378 0.04496360 0.07683403 0.06886244 2.72537
## [3,] 0.10180113 0.02207374 0.02215284 0.01632859 0.02913929 0.03625093 0.04858513 0.07446137 2.67873
## [4,] 0.05228375 0.01181385 0.04695472 0.05877748 0.03125437 0.03448247 0.06227866 0.03719875 1.65999
## [5,] 0.10964293 0.02833617 0.03477586 0.01805750 0.03964978 0.06160769 0.03850884 0.04202283 1.66322
## [6,] 0.03703704 0.02173913 0.01666667 0.01666667 0.01666667 0.02083333 0.01666667 0.01666667 1.66666
## [7,] 0.04746181 0.01314625 0.05069293 0.05441141 0.02819695 0.04993055 0.08332725 0.06836274 6.37621
## [8,] 0.04673258 0.01163744 0.06380421 0.06418135 0.03435538 0.03835882 0.05995569 0.04116116 5.10365
##           [,19]       [,20]        [,21]      [,22]
## [1,] 0.11495059 0.005116862 4.807635e-04 0.05357968
## [2,] 0.04851073 0.007162380 5.161066e-04 0.02615564
## [3,] 0.12109340 0.018244462 2.678730e-05 0.05271038
## [4,] 0.02060298 0.020940823 1.626267e-04 0.04365302
## [5,] 0.06980848 0.012100095 1.663225e-05 0.02798338
## [6,] 0.02083333 0.020833333 1.666667e-02 0.01666667
## [7,] 0.03617125 0.014350225 6.439140e-05 0.02220757
## [8,] 0.01816138 0.008839036 5.147402e-05 0.02222699
##
## $upper_CI
##           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]       [,8]       [
## [1,] 0.05406883 0.03976589 0.07715708 0.03229040 0.05116176 0.06177554 0.04054089 0.07238536 0.05514
## [2,] 0.06744052 0.01751170 0.07531605 0.05217770 0.03561375 0.08578075 0.02806185 0.04523890 0.06463
## [3,] 0.06043168 0.02538252 0.02964338 0.04194061 0.05525977 0.05090377 0.02946365 0.09779674 0.06006
## [4,] 0.11665884 0.02080558 0.10410661 0.08985016 0.03937356 0.06917045 0.03485827 0.03659869 0.06541
## [5,] 0.12828081 0.01405029 0.05426017 0.08887383 0.03826327 0.03661246 0.02380629 0.05675165 0.07551
## [6,] 0.24489796 0.04545455 0.22222222 0.04545455 0.04545455 0.04545455 0.04545455 0.09375000 0.04545
## [7,] 0.05791220 0.01789918 0.09105812 0.05617271 0.02586160 0.14771388 0.02571667 0.03040234 0.07680
## [8,] 0.06714667 0.01323517 0.08150564 0.06362472 0.02567428 0.20505652 0.02368500 0.02687818 0.08451
##           [,10]      [,11]      [,12]      [,13]      [,14]      [,15]      [,16]      [,17]
## [1,] 0.08738356 0.02338885 0.06587051 0.05133497 0.03767021 0.05567474 0.05462616 0.09457072 5.49148
## [2,] 0.07374590 0.02465660 0.06011531 0.08664032 0.03330648 0.04979351 0.08281403 0.07511246 3.01999
## [3,] 0.10850227 0.02535943 0.02583954 0.01925271 0.03345541 0.04063134 0.05537585 0.08115276 3.06861
## [4,] 0.06465623 0.01844824 0.05808596 0.07027915 0.04002958 0.04390842 0.07661173 0.04848396 7.72166
## [5,] 0.11619571 0.03111367 0.03813470 0.02049846 0.04300839 0.06658288 0.04252290 0.04514804 1.86956
## [6,] 0.19565217 0.09375000 0.04545455 0.04545455 0.04545455 0.15555556 0.04545455 0.04545455 4.54545
## [7,] 0.05416124 0.01700754 0.05906697 0.06074693 0.03372068 0.05716525 0.09256604 0.07613655 7.10536
## [8,] 0.05278027 0.01457211 0.07095243 0.07194532 0.03971283 0.04391404 0.06932631 0.04747090 5.64184
##           [,19]       [,20]        [,21]      [,22]
## [1,] 0.14670329 0.013854674 3.739416e-03 0.07754166
## [2,] 0.05362861 0.009207843 1.588456e-03 0.02961028
## [3,] 0.13143076 0.021340230 3.068617e-05 0.05791758
## [4,] 0.02841960 0.028607091 1.865002e-04 0.05449145
## [5,] 0.07368739 0.014029833 1.869569e-05 0.03133652
## [6,] 0.08888889 0.088888889 4.545455e-02 0.04545455
## [7,] 0.04278154 0.018105909 8.977560e-04 0.02762013
## [8,] 0.02262071 0.011510470 3.847323e-04 0.02652420
```

Render this .rmd into a pdf

```
library(rmarkdown)
render("viterbi.Rmd", pdf_document(TRUE), "viterbi.pdf")
```