```
##############################################################################
# GENERATE THE TRAINED HIDDEN MARKOV MODEL
#
# function: get.hmm()
#
# parameter: dna, a data frame containing the source data, each row of which
#            contains a 60-nucleotide sequence and its correct state (EI/IE/N)
#
# return value: the HMM model which is a list containing
#   States: a vector with the names of the states
#   Symbols: a vector with the names of the symbols
#   startProbs: a vector with the starting probabilities of the states
#   transProbs: a matrix containing the transition probabilities between the states
#   emissionProbs: a matrix containing the emission probabilities of the states
##############################################################################
get.hmm <- function(dna, subseq) {

  # Nucleotides
  nucleotids <- unlist(strsplit(paste(dna[, 'V3'], collapse = ''), ''))

  # Sequence of aminoacids
  amino <- sapply(seq(1, length(nucleotids), subseq), function(i) {
    paste(nucleotids[i:(i + subseq - 1)], collapse = '')
  })

  # Skeleton
  hmm <- vector(mode = 'list', length = 5)
  names(hmm) <- c('States', 'Symbols', 'startProbs', 'transProbs',
                  'emissionProbs')

  # Names of the states
  hmm[[1]] <- c('E', 'I', 'N')

  # Name of the aminoacids
  hmm[[2]] <- unique(amino)

  # Initial state probabilities (a priori)
  tt <- table(substr(dna[, 1], 1, 1))
  hmm[[3]] <- as.numeric(tt / sum(tt))
  names(hmm[[3]]) <- c('E', 'I', 'N')

  # Simulate transition probabilities
  set.seed(666)
  first <- sample(1:nrow(dna), 1)
  labs <- substr(dna[, 1], 1, 1)[first]

  # Conditional probabilities
  pI <- tt[names(tt) %in% c('E', 'N')] / sum(tt[names(tt) %in% c('E', 'N')])
  pE <- tt[names(tt) %in% c('I', 'N')] / sum(tt[names(tt) %in% c('I', 'N')])
  pN <- tt[names(tt) %in% c('E', 'I')] / sum(tt[names(tt) %in% c('E', 'I')])

  # Start simulation
  for (i in 2:nrow(dna)) {
    if (labs[i - 1] == 'I') {
      draw <- ifelse(rbinom(1, 1, pI[1]) == 1, 'E', 'N')
    } else if (labs[i - 1] == 'E') {
      draw <- ifelse(rbinom(1, 1, pE[1]) == 1, 'I', 'N')
    } else if (labs[i - 1] == 'N') {
      draw <- ifelse(rbinom(1, 1, pN[1]) == 1, 'E', 'I')
    }
    labs <- c(labs, draw)
  }

  # Extend the sequence times 20 (one per aminoacid)
  #ext.labs <- sapply(labs, function(x) { rep(x, 20) })
  ext.labs <- c()
```

```
  for (lab in labs) {
    ext.labs <- c(ext.labs, rep(lab, 60 / subseq))
  }

  # Transition probabilities
  trI <- table(ext.labs[which(ext.labs == 'I') + 1]) /
         sum(table(ext.labs[which(ext.labs == 'I') + 1]))
  trE <- table(ext.labs[which(ext.labs == 'E') + 1]) /
         sum(table(ext.labs[which(ext.labs == 'E') + 1]))
  trN <- table(ext.labs[which(ext.labs == 'N') + 1]) /
         sum(table(ext.labs[which(ext.labs == 'N') + 1]))

  # Fill the matrix
  res <- matrix(nrow = 3, ncol = 3)
  colnames(res) <- c('E', 'I', 'N')
  rownames(res) <- c('E', 'I', 'N')
  res[1, ] <- trE
  res[2, ] <- trI
  res[3, ] <- trN
  # if (FALSE) {
  #   res[1, ] <- c(0.95, 0.025, 0.025)
  #   res[2, ] <- c(0.025, 0.95, 0.025)
  #   res[3, ] <- c(0.025, 0.025, 0.95)
  # }
  hmm[[4]] <- res
  names(attr(hmm$transProbs, "dimnames"))[1] <- 'from'
  names(attr(hmm$transProbs, "dimnames"))[2] <- 'to'

  # Emission probabilities
  classes <- c()
  for (i in 1:nrow(dna)) {
    classes <- c(classes, rep(dna[i, 1], 60 / subseq))
  }
  amino2 <- cbind(amino, classes)
  amino2[, 2] <- substr(amino2[, 2], 1, 1)
  camino <- paste(amino2[, 2], amino2[, 1], sep = '')

  # Fill the matrix
  res <- matrix(nrow = 3, ncol = length(unique(amino)))
  uamino <- unique(amino)
  for (i in 1:length(unique(amino))) {
    aux <- c()
    aux[1] <- length(which(camino == paste('E', uamino[i], sep = ''))) /
              length(which(substr(camino, 1, 1) == 'E'))
    aux[2] <- length(which(camino == paste('I', uamino[i], sep = ''))) /
              length(which(substr(camino, 1, 1) == 'I'))
    aux[3] <- length(which(camino == paste('N', uamino[i], sep = ''))) /
              length(which(substr(camino, 1, 1) == 'N'))
    res[, i] <- aux
  }
  colnames(res) <- uamino
  rownames(res) <- c('E', 'I', 'N')
  hmm[[5]] <- res
  names(attr(hmm$emissionProbs, "dimnames"))[1] <- 'states'
  names(attr(hmm$emissionProbs, "dimnames"))[2] <- 'symbols'

  return(list(hmm, amino))
}

##############################################################################
# RUN THE VITERBI ALGORITHM TO CLASSIFY A SET OF DNA SEQUENCES
#
# function: classify()
#
# parameters:
#   dna, a data frame containing the source data, each row of which
```

```
#     contains a 60-nucleotide sequence and its correct state (EI/IE/N)
#   subseq (default=5): the number of nucleotides in a subsequence that we
#     consider to be the "symbol" emitted from the model
#
# return value: the HMM model which is a list containing
#   States: a vector with the names of the states
#   Symbols: a vector with the names of the symbols
#   startProbs: a vector with the starting probabilities of the states
#   transProbs: a matrix containing the transition probabilities between the states
#   emissionProbs: a matrix containing the emission probabilities of the states
##############################################################################
classify <- function(dna, subseq = 5) {

  # HMM matrix
  model <- get.hmm(dna, subseq)
  hmm <- model[[1]]
  amino <- model[[2]]

    # Run it
  total <- c()
  for (i in 1:nrow(dna)) {
    j <- (60 / subseq) * (i - 1) + 1
    obs2 <- amino[j:(j + 60 / subseq - 1)]

    #From when we used the HMM package...
    #trial <- HMM::viterbi(hmm, obs2)
    #post1 <- rowMeans(HMM::posterior(hmm, obs2))
    #post2 <- rowMeans(HMM::posterior(hmm, rev(obs2)))
    #lab1 <- names(which.max(post1))
    #lab2 <- names(which.max(post2))

    post1 <- viterbi(hmm, obs2)
    post1 <- as.vector(post1[, ncol(post1)])
    post2 <- viterbi(rev(hmm), obs2)
    post2 <- as.vector(post2[, ncol(post2)])

    if (!is.null(post1)) {
      lab1 <- hmm$States[which.max(post1)]
      lab2 <- hmm$States[which.max(post2)]
    }

    if (lab1 == lab2) {
      result <- lab1
    } else {
      result <- ifelse(which.max(c(max(post1), max(post2))) == 1, lab1, lab2)
    }

    total <- c(total, result)
    #total <- c(total, names(table(trial))[1])
  }

  tt <- table(dna[, 1], total)
  sr <- sum(diag(tt)) / sum(tt)
  er <- (tt[, 1] / rowSums(tt)[1])[1]
  ir <- (tt[, 2] / rowSums(tt)[2])[2]
  nr <- (tt[, 3] / rowSums(tt)[3])[3]

  return(list(tt, sr, er, ir, nr))
}

##############################################################################
# CROSS VALIDATE TO DETERMINE ACCURACY
#
# function: cross.validate()
#
# parameters:
```

```
#   dna: a data frame containing the source data, each row of which
#     contains a 60-nucleotide sequence and its correct state (EI/IE/N)
#   k: the number of folds in our k-fold cross-validation
#   subseq: the number of nucleotides in a subsequence that we consider to be
#     the "symbol" emitted from the model
#
# return value: the HMM model which is a list containing
#   States: a vector with the names of the states
#   Symbols: a vector with the names of the symbols
#   startProbs: a vector with the starting probabilities of the states
#   transProbs: a matrix containing the transition probabilities between the states
#   emissionProbs: a matrix containing the emission probabilities of the states
##############################################################################
cross.validate <- function(dna, k, subseq) {

  # Create the chunks
  chunks <- split(1:nrow(dna), factor(sort(rank(1:nrow(dna)) %% k)))

  # Calculate mean squared errors
  rss <- rep(NA, k)
  for (v in 1:k) {
    # Choose which rows go where
    kept <- sort(as.numeric(as.character(unlist(chunks[setdiff(1:k, v)]))))
    rest <- sort(as.numeric(as.character(unlist(chunks[v]))))

    # Penalized regression results
    model <- get.hmm(dna = dna[kept, ], subseq = subseq)
    hmm <- model[[1]]

    # Sequence of aminoacids
    nucleotids <- unlist(strsplit(paste(dna[rest, 'V3'], collapse = ''), ''))
    amino <- sapply(seq(1, length(nucleotids), subseq), function(i) {
      paste(nucleotids[i:(i + subseq - 1)], collapse = '')
    })

    # Perform k-fold cross-validation
    total <- c()
    for (i in 1:nrow(dna[rest, ])) {
      j <- (60 / subseq) * (i - 1) + 1
      obs2 <- amino[j:(j + 60 / subseq - 1)]

      post1 <- my.viterbi(hmm, obs2)
      post1 <- as.vector(post1[, ncol(post1)])
      post2 <- my.viterbi(rev(hmm), obs2)
      post2 <- as.vector(post2[, ncol(post2)])

      if (!is.null(post1)) {
        lab1 <- hmm$States[which.max(post1)]
        lab2 <- hmm$States[which.max(post2)]

        # If predictions coincide go for it, otherwise highest posterior
        if (lab1 == lab2) {
          result <- lab1
        } else {
          result <- ifelse(which.max(c(max(post1), max(post2))) == 1, lab1, lab2)
        }

        # Accumulate results
        total <- c(total, result)
      } else {
        # If it cannot predict go for the highest prior
        total <- c(total, 'N')
      }
    }

    # Mean squared errors
```

```
    tt <- table(total, dna[rest, 1])
    rss[v] <- sum(diag(tt)) / sum(tt)
  }

  # Result
  return(rss)
}

##############################################################################
# Disabled code for parallelized hyperparameter tuning
# library(parallel)
# library(doMC)
# registerDoMC(cores = 4)
# ks <- c(5, 10, 100, nrow(dna))
# ss <- c(3, 5, 6)
# for (k in ks) {
#   aux <- foreach(s = ss) %dopar% {
#     score <- cross.validate(dna, k = k, subseq = s, average = TRUE)
#     cat('k =', k, ', subseq =', s, ', score =',
#         100 * round(score, 3), '%\n', sep = '')
#   }
# }
# score <- cross.validate(dna, k = 10, subseq = 3); mean(score)
# score <- cross.validate(dna, k = 10, subseq = 5); mean(score)
# score <- cross.validate(dna, k = 10, subseq = 6); mean(score)
# score <- cross.validate(dna, k =  5, subseq = 3, TRUE); mean(score)
# score <- cross.validate(dna, k =  3, subseq = 5, TRUE); mean(score)
# score <- cross.validate(dna, k =  5, subseq = 5, FALSE); mean(score)
# score <- cross.validate(dna, k =  5, subseq = 6); mean(score)
# score <- cross.validate(dna, k = 100, subseq = 3); mean(score)
# score <- cross.validate(dna, k = 100, subseq = 5); mean(score)
# score <- cross.validate(dna, k = 100, subseq = 6); mean(score)
# score <- cross.validate(dna, k = nrow(dna), subseq = 3); mean(score)
# score <- cross.validate(dna, k = nrow(dna), subseq = 5); mean(score)
# score <- cross.validate(dna, k = nrow(dna), subseq = 6); mean(score)
##############################################################################



##############################################################################
# CROSS-VALIDATION
##############################################################################

# Get the DNA data
source('correct_dna.R')
res <- correct.dna()
dna <- res[['dna']]

# Sample from the DNA data
set.seed(666)
dna <- dna[sample(1:nrow(dna), nrow(dna)), ]

# Cross validate and show the score
score <- cross.validate(dna, k =  5, subseq = 5, FALSE)
mean(score)



##############################################################################
# FULL MODEL
##############################################################################

# Get the DNA data
source('correct_dna.R')
res <- correct.dna()
dna <- res[['dna']]
```

```
# Classify all of it
full <- classify(dna, 5)

# Extract our success rates per category and display them
sr <- full[[2]]
er <- full[[3]]
ir <- full[[4]]
nr <- full[[5]]
{
  cat('* Overall success rate: ', 100 * round(sr, 3), '%\n', sep = '')
  cat('* Intron-to-Exon success rate: ', 100 * round(er, 3), '%\n', sep = '')
  cat('* Exon-to-Intron success rate: ', 100 * round(ir, 3), '%\n', sep = '')
  cat('* Neither success rate: ', 100 * round(nr, 3), '%\n', sep = '')
}
# OUTPUT:
# * Overall success rate: 82.4%
# * Exon-to-Intron success rate: 77.3%
# * Intron-to-Exon success rate: 81.8%
# * Neither success rate: 83.7%
```