Wizja Komputerowa i Rozpoznawanie Obrazu - projekt

Implementacja ukrytego modelu Markova dla przewidywania struktury drugorzędowej białek z wykorzystaniem biblioteki „HMM” w R

Prowadzący: prof. dr hab. inż. Katarzyna Stąpor

**Grupa ISMiP**  
Michał Barczyk  
Dominik Korda  
Mateusz Piątkowski  
Michał Sitarz

Spis treści

[1. Wstęp 4](#_Toc452914895)

[2. Skrypt – Rozwiązanie 4](#_Toc452914896)

[3. Omówienie rozwiązania 10](#_Toc452914897)

[3.1. Budowa katalogu rozwiązania 10](#_Toc452914898)

[3.2. Przygotowanie danych 10](#_Toc452914899)

[3.3. Opis działania skryptu 12](#_Toc452914900)

[testPackage 12](#_Toc452914901)

[initializeCluster 12](#_Toc452914902)

[AminoAcidSymbols 12](#_Toc452914903)

[testSet 12](#_Toc452914904)

[observationVecTraining 12](#_Toc452914905)

[train 12](#_Toc452914906)

[cleanResultsVector 13](#_Toc452914907)

[predictHmm 13](#_Toc452914908)

[correctPositions 13](#_Toc452914909)

[numberOfStates 13](#_Toc452914910)

[evaluateEntry 13](#_Toc452914911)

[combineEntriesResult 13](#_Toc452914912)

[evaluateDataSet 13](#_Toc452914913)

[q3Score 13](#_Toc452914914)

[calculateQ3 13](#_Toc452914915)

[saveTestResultToFile 13](#_Toc452914916)

[generateAllStatesVector 13](#_Toc452914917)

[generateStateVector 13](#_Toc452914918)

[performUniformGrowthTest 14](#_Toc452914919)

[uniformGrowthTestEntry 14](#_Toc452914920)

[performUniformGrowthWithConstHelixes 14](#_Toc452914921)

[uniformGrowthTestEntryWithConstHelixes 14](#_Toc452914922)

[performUniformGrowthWithConstHelixesAndStrands 14](#_Toc452914923)

[uniformGrowthTestEntryWithConstHelixesAndStrands 14](#_Toc452914924)

[test2 14](#_Toc452914925)

[4. Badania i wyniki 14](#_Toc452914926)

[4.1. Model wzrostowy z równą liczbą stanów 14](#_Toc452914927)

[4.2. Wpływ prawdopodobieństw startowych na model 15](#_Toc452914928)

[Równe prawdopodobieństwa startowe 15](#_Toc452914929)

[Losowe prawdopodobieństwa startowe 15](#_Toc452914930)

[4.3. Model wzrostowy z stałą liczbą stanów reprezentujących helisy 15](#_Toc452914931)

[4.4. Model wzrostowy z stałą liczbą stanów reprezentujących helisy i kartki 16](#_Toc452914932)

[5. Wnioski 16](#_Toc452914933)

# Wstęp

Celem projektu było zbudowanie i nauczenie ukrytego modelu Markova, opisanego w artykule J. Martin „Analysis of an optimal hidden Markov model for secondary structure prediction” z użyciem biblioteki „HMM” dla pakietu R.

# Skrypt – Rozwiązanie

#############Instaling and testing libraries#############

[testPackage](#_testPackage) <- function(name) {

if (!(name %in% rownames(installed.packages()))) {

print(paste("Installing ", name, " package"))

install.packages(c(name))

}

}

testPackage("HMM")

testPackage("jsonlite")

testPackage("foreach")

testPackage("doParallel")

library(jsonlite)

library(HMM)

library(foreach)

library(doParallel)

#############Cluster initialization#############

[initializeCluster](#_initializeCluster) <- function(numberOfCores) {

cl <- makeCluster(numberOfCores)

clusterSetRNGStream(cl, 9956)

registerDoParallel(cl)

return(cl)

}

#############Definitions#############

#Definition of symbols used in HMM

[AminoAcidSymbols](#_AminoAcidSymbols) <- c("G", "P", "D", "E", "K", "R", "H", "S", "T", "N", "Q", "A", "M", "Y", "W", "V", "I", "L", "F", "C")

#############Data loading#############

#Loading Data to data frame

dataFrame <- fromJSON("Data/jsonProteins.json", flatten = TRUE)

#############Preparing training and test data sets#############

testRows <- sample(1:nrow(dataFrame), nrow(dataFrame)/3);

trainingSet <- dataFrame[testRows,]

[testSet](#_testSet) <- dataFrame[ - testRows,]

#############Preparing observation vector for learning#############

[observationVecTraining](#_observationVecTraining) <- unlist(strsplit(paste(trainingSet[, c("firstStructure")]), ''))

#############Baum Welch training#############

[train](#_train) <- function(model, observation, iter) {

return(tmp <- baumWelch(hmm = model, observation = observation, maxIterations = iter))

}

#############Predicting results form model#############

#Funtion that cleans results vector from digits

[cleanResultsVector](#_cleanResultsVector) <- function(resultsVector) {

return(gsub("[[:digit:]]", "", resultsVector))

}

#Fuction that returns clean results vector

[predictHmm](#_predictHmm) <- function(model, observation) {

return(cleanResultsVector(viterbi(model, observation)))

}

#############Evaluation functions#############

[correctPositions](#_correctPositions) <- function(predictedVector, labeledVector) {

return(predictedVector[labeledVector == predictedVector])

}

[numberOfStates](#_numberOfStates) <- function(vector, stateSymbol) {

return(length(which(vector == stateSymbol)))

}

[evaluateEntry](#_evaluateEntry) <- function(dataSetEntry, model) {

firstStructureVector <- unlist(strsplit(dataSetEntry$firstStructure, ''))

secondaryStructureVector <- unlist(strsplit(dataSetEntry$secondaryStructure, ''))

totalNumber <- length(firstStructureVector)

totalHelixesNumber <- numberOfStates(secondaryStructureVector, "H")

totalStrandsNumber <- numberOfStates(secondaryStructureVector, "B")

totalCoilsNumber <- numberOfStates(secondaryStructureVector, "C")

predictedVector <- predictHmm(model, firstStructureVector)

correctVector <- correctPositions(predictedVector, secondaryStructureVector)

correctNumber <- length(correctVector)

correctHelixesNumber <- numberOfStates(correctVector, "H")

correctStrandsNumber <- numberOfStates(correctVector, "B")

correctCoilsNumber <- numberOfStates(correctVector, "C")

result <- list()

result$TotalNumber <- totalNumber

result$CorrectNumber <- correctNumber

result$TotalHelixesNumber <- totalHelixesNumber

result$CorrectHelixesNumber <- correctHelixesNumber

result$TotalStrandsNumber <- totalStrandsNumber

result$CorrectStrandsNumber <- correctStrandsNumber

result$TotalCoilsNumber <- totalCoilsNumber

result$CorrectCoilsNumber <- correctCoilsNumber

return(result)

}

[combineEntriesResult](#_combineEntriesResult) <- function(ob1, ob2) {

result <- list()

result$TotalNumber <- ob1$TotalNumber + ob2$TotalNumber

result$CorrectNumber <- ob1$CorrectNumber + ob2$CorrectNumber

result$TotalHelixesNumber <- ob1$TotalHelixesNumber + ob2$TotalHelixesNumber

result$CorrectHelixesNumber <- ob1$CorrectHelixesNumber + ob2$CorrectHelixesNumber

result$TotalStrandsNumber <- ob1$TotalStrandsNumber + ob2$TotalStrandsNumber

result$CorrectStrandsNumber <- ob1$CorrectStrandsNumber + ob2$CorrectStrandsNumber

result$TotalCoilsNumber <- ob1$TotalCoilsNumber + ob2$TotalCoilsNumber

result$CorrectCoilsNumber <- ob1$CorrectCoilsNumber + ob2$CorrectCoilsNumber

return(result)

}

[evaluateDataSet](#_evaluateDataSet) <- function(dataSet, model) {

results <- foreach(i = 1:nrow(dataSet), .combine = 'combineEntriesResult', .export = c('evaluateEntry', 'numberOfStates', 'correctPositions', 'predictHmm', 'cleanResultsVector', 'combineEntriesResult'), .packages = c("HMM")) %dopar% {

evaluateEntry(dataSet[i,], model)

}

return(results)

}

#############Helper functions#############

[q3Score](#_q3Score) <- function(resultsOfTest) {

testResults <- foreach(i = 1:ncol(resultsOfTest), .combine = 'cbind', .export = c('calculateQ3')) %do% {

calculateQ3(resultsOfTest[,i])

}

return(testResults)

}

[calculateQ3](#_calculateQ3) <- function(resultOfTestIteration) {

result <- list()

result$TotalQ3 <- resultOfTestIteration$CorrectNumber / resultOfTestIteration$TotalNumber

result$HelixesQ3 <- resultOfTestIteration$CorrectHelixesNumber / resultOfTestIteration$TotalHelixesNumber

result$StrandsQ3 <- resultOfTestIteration$CorrectStrandsNumber / resultOfTestIteration$TotalStrandsNumber

result$CoilsQ3 <- resultOfTestIteration$CorrectCoilsNumber / resultOfTestIteration$TotalCoilsNumber

result$TestType <- resultOfTestIteration$TestType

result$NumberOfStates <- resultOfTestIteration$NumberOfStates

return(result)

}

[saveTestResultToFile](#_saveTestResultToFile) <- function(result, fileName) {

write.table(t(result), file = fileName, row.names = FALSE, na = "", col.names = TRUE, sep = ",")

}

[generateAllStatesVector](#_generateAllStatesVector) <- function(numberOfStates = 0, helixNumber = numberOfStates, strandNumber = numberOfStates, coilNumber = numberOfStates) {

return(c(generateStateVector("H", helixNumber), generateStateVector("B", strandNumber), generateStateVector("C", coilNumber)))

}

[generateStateVector](#_generateStateVector) <- function(symbol, numberOfStates) {

return(paste(symbol, seq(1:numberOfStates), sep=""))

}

#############Testing functions#############

[performUniformGrowthTest](#_performUniformGrowthTest) <- function(testSet, symbols, learningObservations, maxStatesNumber, minStatesNumber = 1, learningIterations = 100, numberOfCores = 1, modelInitialStartProbs = TRUE, modelInitialTransProbs = TRUE, modelInitialEmissionProbs = TRUE) {

cl <- initializeCluster(numberOfCores)

testResults <- foreach(i = minStatesNumber:maxStatesNumber, .combine = 'cbind', .export = c('uniformGrowthTestEntry', 'generateAllStatesVector', 'generateStateVector', 'train', 'evaluateDataSet', 'evaluateEntry', 'numberOfStates', 'correctPositions', 'predictHmm', 'cleanResultsVector', 'combineEntriesResult'), .packages = c("HMM", "foreach", "doParallel")) %dopar% {

uniformGrowthTestEntry(i, symbols, learningObservations, testSet, modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs, learningIterations)

}

stopCluster(cl)

return(testResults)

}

[uniformGrowthTestEntry](#_uniformGrowthTestEntry) <- function(numberOfStates, symbols, learningObservations, testSet, modelInitialStartProbs, modelInitialTransProbs , modelInitialEmissionProbs , learningIterations) {

prob <- function(x) {

x / sum(x)

}

startProbs <- NULL

transProbsMatrix <- NULL

emissionProbsMatrix <- NULL

states <- generateAllStatesVector(numberOfStates = numberOfStates)

if (modelInitialStartProbs) {

startProbs <- prob(runif(length(states)))

}

if (modelInitialEmissionProbs) {

emissionProbsMatrix <- apply(matrix(runif(length(states) \* length(symbols)), nrow = length(states), ncol = length(symbols)), 1, prob)

}

if (modelInitialTransProbs) {

transProbsMatrix <- apply(matrix(runif(length(states) \* length(states)), nrow = length(states), ncol = length(states)), 1, prob)

}

initializedModel <- initHMM(States = states, Symbols = symbols, startProbs = startProbs, transProbs = transProbsMatrix, emissionProbs = emissionProbsMatrix)

trainedModel <- train(initializedModel, learningObservations, learningIterations)

result <- evaluateDataSet(testSet, trainedModel$hmm)

result$TestType <- "Uniform Growth"

result$NumberOfStates <- numberOfStates

return(result)

}

[performUniformGrowthWithConstHelixes](#_performUniformGrowthWithConstHelixe) <- function(testSet, symbols, learningObservations, maxStatesNumber, helixesNumber, minStatesNumber = 1, learningIterations = 100, numberOfCores = 1, modelInitialStartProbs = TRUE, modelInitialTransProbs = TRUE, modelInitialEmissionProbs = TRUE){

cl <- initializeCluster(numberOfCores)

testResults <- foreach(i = minStatesNumber:maxStatesNumber, .combine = 'cbind', .export = c('uniformGrowthTestEntryWithConstHelixes', 'generateAllStatesVector', 'generateStateVector', 'train', 'evaluateDataSet', 'evaluateEntry', 'numberOfStates', 'correctPositions', 'predictHmm', 'cleanResultsVector', 'combineEntriesResult'), .packages = c("HMM", "foreach", "doParallel")) %dopar% {

uniformGrowthTestEntryWithConstHelixes(i, symbols, learningObservations, helixesNumber, testSet, modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs, learningIterations)

}

stopCluster(cl)

return(testResults)

}

[uniformGrowthTestEntryWithConstHelixes](#_uniformGrowthTestEntryWithConstHeli) <- function(numberOfStates, symbols, learningObservations, helixesNumber, testSet, modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs, learningIterations) {

prob <- function(x) {

x / sum(x)

}

startProbs <- NULL

transProbsMatrix <- NULL

emissionProbsMatrix <- NULL

states <- c(generateStateVector("H", helixesNumber), generateStateVector("C", numberOfStates), generateStateVector("B", numberOfStates))

if (modelInitialStartProbs) {

startProbs <- prob(runif(length(states)))

}

if (modelInitialEmissionProbs) {

emissionProbsMatrix <- apply(matrix(runif(length(states) \* length(symbols)), nrow = length(states), ncol = length(symbols)), 1, prob)

}

if (modelInitialTransProbs) {

transProbsMatrix <- apply(matrix(runif(length(states) \* length(states)), nrow = length(states), ncol = length(states)), 1, prob)

}

initializedModel <- initHMM(States = states, Symbols = symbols, startProbs = startProbs, transProbs = transProbsMatrix, emissionProbs = emissionProbsMatrix)

trainedModel <- train(initializedModel, learningObservations, learningIterations)

result <- evaluateDataSet(testSet, trainedModel$hmm)

result$TestType <- "Uniform Growth with Constant Helixes Number"

result$HelixesNumber <- helixesNumber

result$NumberOfStates <- numberOfStates

return(result)

}

[performUniformGrowthWithConstHelixesAndStrands](#_performUniformGrowthWithConstHelixe_1) <- function(testSet, symbols, learningObservations, maxStatesNumber, helixesNumber, strandsNumber, minStatesNumber = 1, learningIterations = 100, numberOfCores = 1, modelInitialStartProbs = TRUE, modelInitialTransProbs = TRUE, modelInitialEmissionProbs = TRUE){

cl <- initializeCluster(numberOfCores)

testResults <- foreach(i = minStatesNumber:maxStatesNumber, .combine = 'cbind', .export = c('uniformGrowthTestEntryWithConstHelixesAndStrands', 'generateAllStatesVector', 'generateStateVector', 'train', 'evaluateDataSet', 'evaluateEntry', 'numberOfStates', 'correctPositions', 'predictHmm', 'cleanResultsVector', 'combineEntriesResult'), .packages = c("HMM", "foreach", "doParallel")) %dopar% {

uniformGrowthTestEntryWithConstHelixesAndStrands(i, symbols, learningObservations, helixesNumber, strandsNumber, testSet, modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs, learningIterations)

}

stopCluster(cl)

return(testResults)

}

[uniformGrowthTestEntryWithConstHelixesAndStrands](#_uniformGrowthTestEntryWithConstHeli_1) <- function(numberOfStates, symbols, learningObservations, helixesNumber, strandsNumber, testSet, modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs, learningIterations) {

prob <- function(x) {

x / sum(x)

}

startProbs <- NULL

transProbsMatrix <- NULL

emissionProbsMatrix <- NULL

states <- c(generateStateVector("H", helixesNumber), generateStateVector("C", numberOfStates), generateStateVector("B", strandsNumber))

if (modelInitialStartProbs) {

startProbs <- prob(runif(length(states)))

}

if (modelInitialEmissionProbs) {

emissionProbsMatrix <- apply(matrix(runif(length(states) \* length(symbols)), nrow = length(states), ncol = length(symbols)), 1, prob)

}

if (modelInitialTransProbs) {

transProbsMatrix <- apply(matrix(runif(length(states) \* length(states)), nrow = length(states), ncol = length(states)), 1, prob)

}

initializedModel <- initHMM(States = states, Symbols = symbols, startProbs = startProbs, transProbs = transProbsMatrix, emissionProbs = emissionProbsMatrix)

trainedModel <- train(initializedModel, learningObservations, learningIterations)

result <- evaluateDataSet(testSet, trainedModel$hmm)

result$TestType <- "Uniform Growth with Constant Helixes and Strands Number"

result$HelixesNumber <- helixesNumber

result$StrandsNumber <- strandsNumber

result$NumberOfStates <- numberOfStates

return(result)

}

#############Sample#############

[test2](#_test2) <- function() {

x <- performUniformGrowthWithConstHelixesAndStrands(testSet, AminoAcidSymbols, observationVecTraining, 18, 13, 14, minStatesNumber = 10, learningIterations = 50, numberOfCores = 8)

saveTestResultToFile(x, "uniformGrowthConstHelixesAndStrands.csv")

saveTestResultToFile(q3Score(x), "uniformGrowthConstHelixesAndStrandsQ3.csv")

return(x)

}

# Omówienie rozwiązania

## Budowa katalogu rozwiązania

Katalog dołączony do sprawozdania ma następującą budowę. W głównym znajduję się plik script.R zawierający rozwiązanie. Katalog ten zawiera także foldery Data i Results. Katalog Results zawiera wyniki z przeprowadzonych zapisanych w formacie CSV. Katalog Data zawiera skrypt do przygotowania danych wejściowych oraz same dane wejściowe. Katalog ten jak i dane wejściowe w plik jsonProteins.json musi istnieć, by model poprawnie działał. Rozwiązanie szuka tego pliku względem katalogu Data(Data/jsonProteins.json), więc jako katalog roboczy musi zostać wskazany katalog zawierający katalog Data, a w nim plik jsonProteins.json. Przykładowe wykorzystanie funkcji napisanych w rozwiązaniu zostało ukazane w funkcji test2. Dołączony do sprawozdania katalog zawiera przygotowany już plik z danymi.

## Przygotowanie danych

Do przygotowania danych wejściowych został wykorzystany następujący skrypt w języku Python(w wersji 3):

import sqlite3

import json

from pathlib import Path

class Protein:

def \_\_init\_\_(self, id, firstStructure, secondaryStructure):

self.id = id

self.firstStructure = firstStructure

self.secondaryStructure = self.\_\_alterSecondaryStructure\_\_(secondaryStructure)

def \_\_replaceMultipleChars\_\_(self, str, chars, newValue):

for c in chars:

if c in str:

str = str.replace(c, newValue)

return str

def \_\_alterSecondaryStructure\_\_(self, str):

helixes = "GI"

strands = "E"

coils = "-ST"

str = self.\_\_replaceMultipleChars\_\_(str, helixes, 'H')

str = self.\_\_replaceMultipleChars\_\_(str, strands, 'B')

str = self.\_\_replaceMultipleChars\_\_(str, coils, 'C')

return str

def addToDatabase(object, cur):

cur.execute("insert into proteins values(:id, :firstStructure, :secondaryStructure)", object)

def createDataBase():

connection = sqlite3.connect('proteins.db')

cursor = connection.cursor()

cursor.execute('''CREATE TABLE proteins (id text, firstStructure text, secondaryStructure text)''')

connection.commit()

connection.close()

def step(line,file):

return Protein(line.rstrip(), file.readline().rstrip(), file.readline().rstrip())

def readFile(fileName,cursor, jsonList):

with fileName.open() as file:

x = True

while(x):

line = file.readline()

if(line == ''):

return

else:

protein = step(line, file)

addToDatabase(protein.\_\_dict\_\_, cursor)

jsonList.append(protein.\_\_dict\_\_)

def checkIfDataBaseExist():

dataBase = Path('proteins.db')

return dataBase.exists()

def readFiles(path):

if(not checkIfDataBaseExist()):

createDataBase()

dir = Path(path)

files = list(dir.glob('\*\*/\*.txt'))

connection = sqlite3.connect('proteins.db')

cursor = connection.cursor()

jsonList = list()

for file in files:

readFile(file,cursor, jsonList)

connection.commit()

connection.close()

with open("jsonProteins.json", 'w') as jsonFile:

json.dump(jsonList, jsonFile)

readFiles('Proteins/1')

Skrypt ten przekształca pliki tekstowe znajdujące się w katologu Proteins/1. W wyniku działania skryptu w katalogu uruchomienia powstają dwa pliki. Proteins.db zawiera bazę danych sqlite3 utworzonych z plików testowych, podobnie plik jsonProteins.json zawiera dane w formacie JSON.

Skrypt ten także przekształca przedstawienie struktury drugorzędowej białka. Jako że model przewiduje tylko trzy klasy, wszystkie podklasy zostają zastąpione symbolem klasy nadrzędnej. Do helis zaliczane są symbole: H, G, I. Symbolem helisy jest H. Do Beta kartek zaliczane są symbole: B, E. Symbolem beta kartki jest B. Do klasy coil(pozostałe) zaliczone są symbole: C, -, S, T. Symbolem coil jest C. Zostało to wykonane zgodnie z klasyfikacją DSSP.

## Opis działania skryptu

### testPackage

Skrypt na początku sprawdza czy są wymagane biblioteki, odpowiada za to funkcja testPackage, jeżeli wymagane biblioteki nie zostały wykryte, zostaną one zainstalowane.

### initializeCluster

Funkcja initializeCluster odpowiada za stworzenia klastra do wykonywania zrównoleglonych funkcji, jest wykorzystywana w funkcjach testowych.

### AminoAcidSymbols

AminoAcidSymbols to wektor zawierający symbole aminokwasów.

### testSet

testSet to struktura zawierająca dane do testowania modelu.

### observationVecTraining

observationVecTraining to wektor zawierający obserwacje, w tym przypadku symbole aminokwasów utworzone z struktury pierwszorzędowej baiłek na podstawie zbioru treningowego.

### train

train to funkcja ucząca model przekazany, jako jej parametr za pomocą algorytmu Baum Welch. Do uczenia wymagana jest przekazanie wektora obserwacji. Parametr iter określa maksymalną liczbę iteracji.

### cleanResultsVector

Funkcja cleanResultsVector czyści wektor wynikowy otrzymany z modelu, jako predykcja struktury drugorzędowej. Wektor ten zawiera numerację stanów, po przetworzeniu przez funkcję numeracja jest usuwana. Otrzymany wektor zawiera wyłącznie symbole helis-H, kartek-B i coil-C.

### predictHmm

predictHmm to funkcja zwracająca czysty przewidziany przez model podany w parametrze, wektor zawierający strukturę drugorzędową. Wykorzystuje algorytm viterbi.

### correctPositions

correctPositions to funkcja zwracająca wektor z poprawnie przewidzianymi pozycjami w strukturze drugorzędowej.

### numberOfStates

numberOfStates to funkcja zwracająca liczbę pozycji w podanym wektorze o danym symbolu.

### evaluateEntry

evaluateEntry to funkcja odpowiadająca za ocenienie modelu podanego w parametrze, na podstawie rekordu testowego. Rekord testowy to dane zawierające strukturę pierwszorzędową oraz drugorzędową białka. Funkcja zwraca listę zawierającą dane takie jak: całkowita liczba pozycji w strukturze drugorzędowej, liczba poprawnie przewidzianych pozycji w całości sekwencji oraz dla poszczególnych rodzajów struktury drugorzędowej. Zwrócona struktura zawiera także całkowitą liczbę poszczególnych rodzajów.

### combineEntriesResult

combineEntriesResult to funkcja łącząca wyniki dla dwóch rekordów danych.

### evaluateDataSet

evaluateDataSet funkcja pozwalająca ocenić model, przekazany przez parametr, na podstawie zbioru rekordów. Ma możliwość wykonywania równoległego, jeżeli jest zarejestrowany klaster.

### q3Score

q3Score funkcja pozwalająca wyliczyć Q3 dla testu. Test może zawierać wiele wyników dla poszczególnych modeli.

### calculateQ3

calculateQ3 liczy Q3 dla pojedynczego wyniku z ewaluacji.

### saveTestResultToFile

saveTestResultToFile pozwala na zapisanie wyników testu oraz Q3 do pliku w formacie csv.

### generateAllStatesVector

generateAllStatesVector pozwala wygenerować wektor zawierający stany dla modelu o podanej liczbie. Wykorzystywany w modelu przyrostowym. Możliwe jest tylko wygenerowanie n stanów dla każdego rodzaju stanu(helisa, kartka, coil) tej samej liczby stanów(Helisa = n, kartka = n, coil = n).

### generateStateVector

generateStateVector pozwala wygenerować wektor o liczbie stanów podanej w parametrze, związany z podanym symbolem w parametrze.

### performUniformGrowthTest

performUniformGrowthTest pozwala na przeprowadzenie badania z użyciem modelu wzrostowego. Parametry pozwalają określić liczbę wątków, na których będzie przeprowadzone badanie. Parametry związane z prawdopodobieństwem(modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs) przyjmują wartości TRUE, FALSE. Wartość TRUE oznacza, że wylosowane zostaną wartości zgodnie z rozkładem normalnym. Wartość FALSE spowoduje wybranie wszędzie takich samych wartości. Funkcja ta zwraca wyniki.

### uniformGrowthTestEntry

uniformGrowthTestEntry funkcja ta reprezentuje pojedynczą iterację badania.

### performUniformGrowthWithConstHelixes

performUniformGrowthWithConstHelixes pozwala na przeprowadzenie badania z użyciem modelu wzrostowego ze stałą liczbą stanów reprezentującą helisy, określoną poprzez parametr. Parametry pozwalają określić liczbę wątków, na których będzie przeprowadzone badanie. Parametry związane z prawdopodobieństwem(modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs) przyjmują wartości TRUE, FALSE. Wartość TRUE oznacza, że wylosowane zostaną wartości zgodnie z rozkładem normalnym. Wartość FALSE spowoduje wybranie wszędzie takich samych wartości. Funkcja ta zwraca wyniki.

### uniformGrowthTestEntryWithConstHelixes

uniformGrowthTestEntryWithConstHelixes funkcja ta reprezentuje pojedynczą iterację badania.

### performUniformGrowthWithConstHelixesAndStrands

performUniformGrowthWithConstHelixesAndStrands pozwala na przeprowadzenie badania z użyciem modelu wzrostowego ze stałą liczbą stanów reprezentującą helisy i kartki, określoną poprzez parametry. Parametry pozwalają określić liczbę wątków, na których będzie przeprowadzone badanie. Parametry związane z prawdopodobieństwem(modelInitialStartProbs, modelInitialTransProbs, modelInitialEmissionProbs) przyjmują wartości TRUE, FALSE. Wartość TRUE oznacza, że wylosowane zostaną wartości zgodnie z rozkładem normalnym. Wartość FALSE spowoduje wybranie wszędzie takich samych wartości. Funkcja ta zwraca wyniki.

### uniformGrowthTestEntryWithConstHelixesAndStrands

uniformGrowthTestEntryWithConstHelixesAndStrands funkcja ta reprezentuje pojedynczą iterację badania.

### test2

test2 przedstawia przykładowe wywołanie badania polegającego na przeprowadzeniu testu o stałych liczbach stanów reprezentujących helisy(13) i kartki(14). Zmienna jest liczba stanów reprezentująca coil, w zakresie od 10 do 18. Operacja ta jest przeprowadzona na 8 wątkach. Otrzymane wyniki zostają zapisane w pliku. Następnie zapisany do pliku jest wyniki Q3 dla przeprowadzonego badania.

# Badania i wyniki

## Model wzrostowy z równą liczbą stanów

Model wzrostowy z równą liczbą stanów polega na porównywaniu wyników otrzymanych z modeli, które budowane są według zasady, że każdy rozdaj(helisy, kartki, coil) reprezentowane są przez tą samą liczbę stanów w ukrytym modelu Markova. Jest to realizowane za pomocą funkcji [performUniformGrowthTest](#_performUniformGrowthTest).

## Wpływ prawdopodobieństw startowych na model

### Równe prawdopodobieństwa startowe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba stanów dla każdego rodzaju | Q3 helisy | Q3 kartki | Q3 coil | Q3 całkowite |
| 1 | 1 | 0 | 0 | 0.338490580005182 |
| 2 | 0.99907053034445 | 0 | 0 | 0.338175963282378 |
| 3 | 0.54505194095134 | 0.520149193221438 | 0 | 0.303216493319021 |
| 4 | 0 | 0 | 1 | 0.433264241033423 |
| 5 | 0 | 0.101597340468661 | 0.901883729870574 | 0.413943072880039 |
| 6 | 0.000984144341170038 | 0 | 0.999231130665072 | 0.433264241033423 |
| 7 | 0.0287588846364133 | 0 | 0.964034001110589 | 0.427416071362476 |
| 8 | 1 | 0 | 0 | 0.338490580005182 |
| 9 | 0.165226899945325 | 0.798183734695532 | 0.049848361881167 | 0.259706851241811 |
| 10 | 0 | 0.0211627341279494 | 0.983469309299047 | 0.259706851241811 |
| 11 | 0 | 1 | 0.0007688693349280 | 0.228578302550246 |
| 12 | 0.996008747949699 | 0.000324333090083516 | 0.0079022681645380 | 0.340637376466669 |
| 13 | 0.0223072717331875 | 0.990432173842536 | 0 | 0.233612170115113 |
| 14 | 0 | 0 | 1 | 0.433264241033423 |
| 15 | 0 | 0 | 1 | 0.433264241033423 |
| 16 | 0 | 0 | 1 | 0.433264241033423 |

Z badania można wywnioskować, że równe prawdopodobieństwa powodują „zablokowanie się” modelu w stanach reprezentujących jeden rodzaj. Model w takim przepadku przewiduje tylko jeden rodzaj. Badanie trwało na 8 wątkach powyżej 24 godzin. W związku z tym zjawiskiem „blokowania” wykluczono przydatność takiego rozwiązania do dalszych rozważań.

### Losowe prawdopodobieństwa startowe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba stanów dla każdego rodzaju | Q3 helisy | Q3 kartki | Q3 coil | Q3 całkowite |
| 1 | 0.291214470284238 | 0.179153342469925 | 0.517732188295165 | 0.364529113748352 |
| 2 | 0.4871834625323 | 0.340490330440079 | 0.213740458015267 | 0.334426400166563 |
| 3 | 0.358966408268734 | 0.257271204507385 | 0.327210559796438 | 0.3219342077868 |
| 4 | 0.132868217054264 | 0.266560073092736 | 0.510615458015267 | 0.328180303976681 |
| 5 | 0.348578811369509 | 0.268844221105528 | 0.311903625954198 | 0.314404191824554 |
| 6 | 0.373488372093023 | 0.243337901629359 | 0.235209923664122 | 0.283486015684642 |
| 7 | 0.548682170542636 | 0.247297091518197 | 0.174459287531807 | 0.316694427094177 |
| 8 | 0.416330749354005 | 0.345363179534034 | 0.204675572519084 | 0.307793740023596 |
| 9 | 0.307545219638243 | 0.232221714633775 | 0.369473600508906 | 0.317405788049136 |
| 10 | 0.303875968992248 | 0.364321608040201 | 0.365696564885496 | 0.344628357276702 |
| 11 | 0.304547803617571 | 0.376732145576367 | 0.342358460559796 | 0.337497397459921 |
| 12 | 0.231266149870801 | 0.399954317039744 | 0.311187977099237 | 0.304583940592685 |
| 13 | 0.425271317829457 | 0.317648850312167 | 0.370229007633588 | 0.376726351585814 |
| 14 | 0.405116279069767 | 0.515456068219887 | 0.238788167938931 | 0.357675758206676 |
| 15 | 0.251576227390181 | 0.478300593878483 | 0.289718511450382 | 0.319886876257894 |
| 16 | 0.332919896640827 | 0.275544388609715 | 0.387166030534351 | 0.343517940176279 |

Dzięki zastosowaniu losowych prawdopodobieństw startowych otrzymano bardziej zróżnicowane wyniki. Skuteczność takiego modelu jednak jest bardzo niska. Do dalszych badań wybrano model o liczbie stanów równej 13. Badanie trwało na 8 wątkach powyżej 24 godzin.

## Model wzrostowy z stałą liczbą stanów reprezentujących helisy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba stanów dla kartki i coil | Q3 helisy | Q3 kartki | Q3 coil | Q3 całkowite |
| 7 | 0.340279555578311 | 0.404091864790555 | 0.295361273473449 | 0.335023205997858 |
| 8 | 0.417746147150684 | 0.32613617984797 | 0.19263773162542 | 0.300589075330239 |
| 9 | 0.530950796170191 | 0.283761927866731 | 0.147328275919247 | 0.311192431274545 |
| 10 | 0.475039680507911 | 0.311175804625586 | 0.147577001202172 | 0.297857907890039 |
| 11 | 0.282115611079822 | 0.405790069545528 | 0.241097707581976 | 0.291752945376651 |
| 12 | 0.27929957503456 | 0.385007278020378 | 0.270986195746798 | 0.299053909318101 |
| 13 | 0.329015411397266 | 0.415089762251334 | 0.219209882684575 | 0.300731881470903 |
| 14 | 0.171010188930418 | 0.177502830341258 | 0.655266757865937 | 0.380971081756516 |
| 15 | 0.277661154062772 | 0.330260391395763 | 0.371346847407039 | 0.329614423420207 |

Badano modele o 13 stanach reprezentujących helisę i zmiennej liczbie stanów reprezentujących kartki i coil. Nie otrzymano znaczącej poprawy wyników. Bardzo trudno znaleźć jakąś zależność pomiędzy liczbą stanów, a dokładnością rozwiązania. Do dalszych badań wybrano model o party o 13 stanów reprezentujących helisę i 14 stanów reprezentujących strukturę kartki. Badanie trwało na 8 wątkach powyżej 12 godzin.

## Model wzrostowy z stałą liczbą stanów reprezentujących helisy i kartki

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Liczba stanów dla coil | Q3 helisy | Q3 kartki | Q3 coil | Q3 całkowite |
| 10 | 0.336801521073202 | 0.341575757575758 | 0.263673294847124 | 0.306614985190332 |
| 11 | 0.307753248125066 | 0.413818181818182 | 0.30883044686765 | 0.332211942809083 |
| 12 | 0.341449244744903 | 0.398707070707071 | 0.278212529399188 | 0.327366804402677 |
| 13 | 0.287313826977923 | 0.302141414141414 | 0.444644002565747 | 0.357936885215929 |
| 14 | 0.286468786310341 | 0.285333333333333 | 0.446440025657473 | 0.354609280725491 |
| 15 | 0.205239252139009 | 0.313212121212121 | 0.568954457985888 | 0.385179361538743 |
| 16 | 0.347945494876941 | 0.254060606060606 | 0.351806713705367 | 0.328354115625114 |
| 17 | 0.136738143023133 | 0.570909090909091 | 0.382852255719478 | 0.340201850294365 |
| 18 | 0.373085454737509 | 0.308606060606061 | 0.311567243959803 | 0.332193659267927 |

Badano modele o 13 stanach reprezentujących helisę, 14 stanach reprezentujących strukturę kartki i zmiennej liczbie stanów reprezentujących coil. Najlepszy model okazał się mieć 15 stanów reprezentujących strukturę rodzaju coil. Nadal nie można zauważyć jakiejkolwiek zależności pomiędzy liczbą stanów, a dokładnością rozwiązania. Badanie trwało na 8 wątkach powyżej 15 godzin.

# Wnioski

Najlepszym otrzymanym modelem ze względu na Q3 okazał się model mający 42 stany. Model ten posiadał 13 stanów reprezentujących helisę, 14 kartkę i 15 coil. Nie jest to jednak model do czegoś przydatny. Przy kryterium Q3 równym 38,5%, takie rozwiązanie jest gorsze niż strategia przewidywania tylko jednej z trzech struktur, co można zaobserwować w badaniu z równymi prawdopodobieństwami, gdzie dokładność wynosi około 43%. Dlatego też nie porównywano rozwiązania z PSIPRED.

Model jest bardzo skomplikowany obliczeniowo. Wykonanie projektu byłoby nie możliwe bez zrównoleglenia rozwiązania. Przy testach w bardzo małych przedziałach liczby stanów, obliczenia trwały bardzo długo. Do obliczeń zastosowano komputer z procesorem intel i7-4790K, o taktowaniu równym 4,4 GHz. Podczas obliczeń procesor był maksymalnie obciążony. Jedynym rozwiązaniem by przyspieszyć wykonywanie obliczeń jest wykorzystanie karty graficznej. Trzeba by było jednak zrównoleglić algorytmy viterbi i Baum Welch, gdyż karta graficzna nie nadaję się do zrównoleglenia tak jak jest to obecnie w rozwiązaniu.

Mimo że zastosowano metodykę przedstawianą w artykule, nie udało się odtworzyć się takiego rozwiązania jak to opisano w artykule. Podstawowym problem okazała się mała próba, związana z bardzo długo trwającymi testami. Otrzymane wyniki wydają się być zupełnie niezależne od prowadzonych działań na rzecz poprawy dokładności. Możliwe, że zwiększenie parametru maksymalnych iteracji przy uczeniu lub zwiększenie liczby obserwacji dałoby lepszy wynik, wiąże się to jednak z znacznym wydłużeniem czasu działania testu, który już jest bardzo długi.

Projekt ten pozwolił zapoznać się z ukrytymi modelami Markova, a także z możliwościami pakietu R.